علیرضا صمیمی 401110501

ابتدا پشته را نشان می‌دهیم.

```
  always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      top <= 0;
      for (int i = 0; i < DEPTH; i++) begin
        stack_mem[i] <= '0;
      end
    end else begin
      if (opcode == 3'b110 && !full) begin
        stack_mem[top] <= input_data;
        top <= top + 1;
      end else if (opcode == 3'b111 && !empty) begin
        top <= top - 1;
      end
    end
  end

  always_comb begin
    top_element = (top > 0) ? stack_mem[top-1] : 0;
    second_element = (top > 1) ? stack_mem[top-2] : 0;

    add_result = {top_element[WIDTH-1], top_element} + {second_element[WIDTH-1], second_element};
    mult_result = top_element * second_element;
  end

  always_comb begin
    if (opcode == 3'b100 && top > 1) begin
      output_data = add_result;
      overflow = (add_result[WIDTH] != add_result[WIDTH-1]);
    end else if (opcode == 3'b101 && top > 1) begin
      output_data = mult_result;
      overflow = (mult_result[2*WIDTH-1:WIDTH] != {WIDTH{mult_result[WIDTH-1]}});
    end else begin
      output_data = top_element;
      overflow = 0;
    end
  end
endmodule
```

این پشته پوش و پاپ را به صورت سنکرون اجرا می‌کند و بقیه مدار به صورت ترکیبی پیاده شده. دو متغیر top_element و second_element برای راحتی کار استفاده شده و برای اورفلو بیت‌های اضافی چک شده اند.

```
    rst_n = 0;
    opcode = 3'b000;
    input_data = 0;

    #10 rst_n = 1;

    for (int i = 1; i <= DEPTH; i++) begin
      #10 opcode = 3'b110;
      input_data = i;
    end

    #10 opcode = 3'b110;
    input_data = DEPTH + 1;

    for (int i = 1; i <= DEPTH; i++) begin
      #10 opcode = 3'b111;
    end

    #10 opcode = 3'b111;

    #10 opcode = 3'b110;
    input_data = 32'sh00000001;
    #10 opcode = 3'b110;
    input_data = -32'sh00000002;
    #10 opcode = 3'b100;

    #10 opcode = 3'b110;
    input_data = -32'sh00000003;
    #10 opcode = 3'b110;
    input_data = 32'sh00000004;
    #10 opcode = 3'b101;

    #10 opcode = 3'b110;
    input_data = 32'sh7FFFFFFF;
    #10 opcode = 3'b110;
    input_data = 32'sh00000001;
    #10 opcode = 3'b100;

    #10 opcode = 3'b110;
    input_data = -32'sh7FFFFFFF;
    #10 opcode = 3'b110;
    input_data = -32'sh00000002;
    #10 opcode = 3'b101;

    #10 $finish;
end
```

برای تست پشته آن را پر و خالی کرده تا سیگنال های full و empty تست شوند و سپس یک جمع و یک ضرب ساده و سپس یک جمع و یک ضرب اورفلودار (۴ حالت مثبت منفی تست شده) انجام و چک شده. این تست برای ۴ ۸ و ۱۶ بیت نیز تست شده که کد آن موجود است.

برای حساب کردن عبارات ریاضی infix ماژول infix_to_postfix_evaluator نوشته شده که ۴ استیت دارد. استیت‌های IDLE و DONE نیازی به توضیح ندارند. در پایین لاجیک مربوط به عوض کردن استیت‌ها آمده.

```
always_ff @(posedge clk or negedge rst_n) begin
  if (!rst_n) begin
    state <= IDLE;
  end else begin
    state <= next_state;
  end
end

always_comb begin
  next_state = state;
  case (state)
    IDLE: if (start) next_state = PARSE;
    PARSE: begin
      if (evalcnt) next_state = EVAL;
      else if (infix_input[infix_index] == 0 && ostack_empty) next_state = DONE;
    end
    EVAL: if (!evalcnt) next_state = PARSE;
    DONE: next_state = IDLE;
  endcase
end
```

در ابتدای PARSE اگر به یک اوپریتور رسیدیم عدد خوانده شده را پوش می‌کنیم.

```
PARSE: begin
  ostack_opcode <= 0;
  estack_opcode <= 0;
  if ((infix_input[infix_index] < "0" || infix_input[infix_index] > "9") && reading_number) begin
    if (is_negative) begin
      num <= -num;
      is_negative <= 0;
    end
    estack_input <= num;
    estack_opcode <= 3'b110;
    reading_number <= 0;
    num <= 0;
```

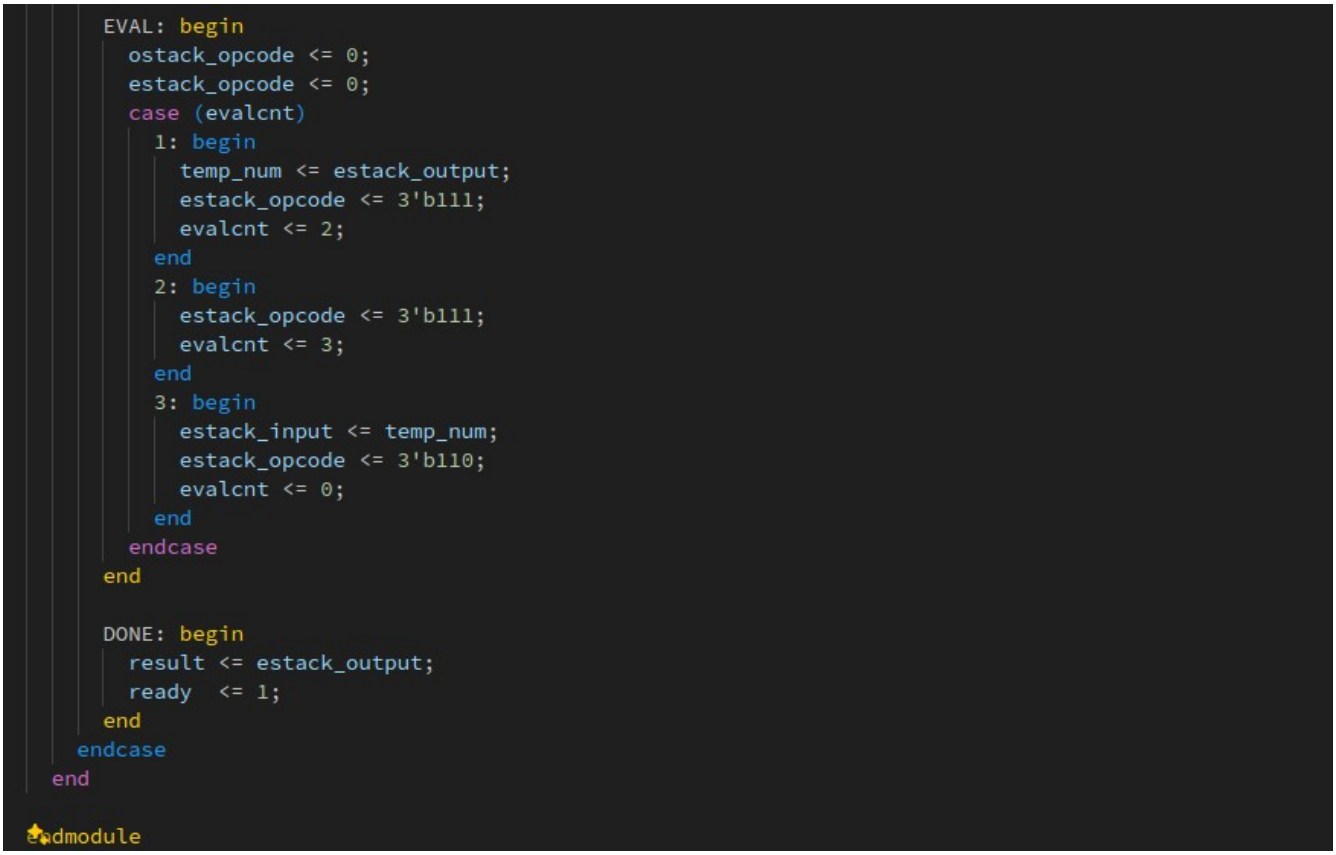
اسپیس‌ها را اسکیپ می‌کنیم و برای اوپریتورها طبق الگوریتم شانتینگ یارد جلو می‌رویم.

```
end else begin
  case (infix_input[infix_index])
    " ": infix_index <= infix_index + 1;
    "+", "*": begin
      if (!ostack_empty && ostack_output != "(" &&
          ((infix_input[infix_index] == "+" && ostack_output == "*") ||
          (infix_input[infix_index] == ostack_output))) begin
        evalcnt <= 1;
        estack_opcode <= ostack_output == "+" ? 3'b100 : 3'b101;
        ostack_opcode <= 3'b111;
      end else begin
        ostack_input  <= infix_input[infix_index];
        ostack_opcode <= 3'b110;
        infix_index   <= infix_index + 1;
      end
    end
    "(": begin
      ostack_input  <= infix_input[infix_index];
      ostack_opcode <= 3'b110;
      infix_index   <= infix_index + 1;
    end
    ")": begin
      if (!ostack_empty && ostack_output != "(") begin
        evalcnt <= 1;
        estack_opcode <= ostack_output == "+" ? 3'b100 : 3'b101;
        ostack_opcode <= 3'b111;
      end else if (!ostack_empty > 0 && ostack_output == "(") begin
        ostack_opcode <= 3'b111;
        infix_index   <= infix_index + 1;
      end
```

به این صورت که هر جا خواستیم یک اوپریتور را از استک خارج کنیم با قرار دادن ۱ در evalcnt به استیت EVAL می‌رویم تا آن اوپریتور را روی استک اوپرندها اعمال کند.

```
      "-": begin
        is_negative <= 1;
        infix_index <= infix_index + 1;
      end
      0: begin
        if (!ostack_empty) begin
          evalcnt <= 1;
          estack_opcode <= ostack_output == "+" ? 3'b100 : 3'b101;
          ostack_opcode <= 3'b111;
        end
      end
      default: begin  // Number
        reading_number <= 1;
        if (infix_input[infix_index] >= "0" && infix_input[infix_index] <= "9") begin
          num = num * 10 + (infix_input[infix_index] - "0");
          infix_index <= infix_index + 1;
        end
      end
    endcase
  end
end
```

اگر علامت منها دیدیم عدد بعدی را منفی می‌کنیم. اگر به کاراکتر نال رسیدیم به معنی اتمام عبارت ریاضی است و اگر یک رقم دیدیم شروع به خواندن عدد می‌کنیم.



```
      EVAL: begin
        ostack_opcode <= 0;
        estack_opcode <= 0;
        case (evalcnt)
          1: begin
            temp_num <= estack_output;
            estack_opcode <= 3'b111;
            evalcnt <= 2;
          end
          2: begin
            estack_opcode <= 3'b111;
            evalcnt <= 3;
          end
          3: begin
            estack_input <= temp_num;
            estack_opcode <= 3'b110;
            evalcnt <= 0;
          end
        endcase
      end

      DONE: begin
        result <= estack_output;
        ready  <= 1;
      end
    endcase
  end

endmodule
```

استیت EVAL از ۳ مرحله تشکیل شده که ۲ عدد روی استک اوپرندها را خارج و عدد متناظر با عبارت مورد نظرمان را در پشته قرار می‌دهد.

```
initial begin
  clk   = 0;
  rst_n = 0;
  start = 0;
  #10 rst_n = 1;
  infix_input[0:4] = "2 * 4";
  infix_input[5]   = 8'b0;
  #10 start = 1;
  #10 start = 0;
  @(ready);
  #10 rst_n = 0;
  #10 rst_n = 1;
  infix_input[0:16] = "2 * 3 + 5 * 4 + 3";
  infix_input[17]   = 8'b0;
  #10 start = 1;
  #10 start = 0;
  @(ready);
  #10 rst_n = 0;
  #10 rst_n = 1;
  infix_input[0:35] = "2 * 3 + 5 * 4 + 3 + -5 * (1 + 2 + 3)";
  infix_input[36]   = 8'b0;
  #10 start = 1;
  #10 start = 0;
  @(ready);
  #10 rst_n = 0;
  #10 rst_n = 1;
  infix_input[0:35] = "2 * 3 + (10 + 4 + 3) * -20 + (6 + 5)";
  infix_input[36]   = 8'b0;      You, 18 minutes ago • Uncommitted changes
  #10 start = 1;
  #10 start = 0;
  @(ready);
  #10 $finish;
end

always #5 clk = ~clk;
```

برای تست ماژول نوشته شده چند عبارت ریاضی را به شکل بالا به ماژول می‌دهیم و صبر می‌کنیم محاسبه کند تا نتیجه را چک کنیم.