DSD - Project Report

Mehran Talaei 401110364

June 29, 2024

STACKED BASED ALU

```verilog
`timescale 1ns/1ps

module STACK_BASED_ALU
#(parameter DATA_WIDTH = 8, parameter STACK_SIZE = 64)
(
    input clk,
    input [DATA_WIDTH-1:0] input_data,
    input [2:0] opcode,
    output reg [DATA_WIDTH-1:0] output_data,
    output reg overflow,
    output reg [DATA_WIDTH-1:0] debug_value
);

reg [DATA_WIDTH-1:0] stack [0:STACK_SIZE-1];
integer stack_pointer = 0;

always @(posedge clk) begin
    overflow <= 0;
    debug_value <= 0;
    case (opcode)
        3'b100: // addition
        begin
            if (stack_pointer >= 2) begin
                stack[stack_pointer - 2] <= stack[stack_pointer - 1] + stack[stack_pointer - 2];
                debug_value <= stack[stack_pointer - 2];
                if ((stack[stack_pointer - 2][DATA_WIDTH-1] == 1 && stack[stack_pointer - 1][DATA_WIDTH-1] == 0 && stack[stack_pointer - 2][DATA_WIDTH-1] == 0) ||
                    (stack[stack_pointer - 2][DATA_WIDTH-1] == 0 && stack[stack_pointer - 1][DATA_WIDTH-1] == 1 && stack[stack_pointer - 2][DATA_WIDTH-1] == 1))
                    overflow <= 1;
                stack_pointer <= stack_pointer - 1;
            end else begin
                overflow <= 1;
            end
            output_data <= debug_value;
        end

        3'b101: // multiplication
        begin
            if (stack_pointer >= 2) begin
                stack[stack_pointer - 2] <= stack[stack_pointer - 1] * stack[stack_pointer - 2];
                debug_value <= stack[stack_pointer - 2];
                if (|stack[stack_pointer - 2][2*DATA_WIDTH-1:DATA_WIDTH]) overflow <= 1;
                stack_pointer <= stack_pointer - 1;
            end else begin
                overflow <= 1;
            end
            output_data <= debug_value;
        end

        3'b110: // pushing
        begin
            if (stack_pointer < STACK_SIZE) begin
                stack[stack_pointer] <= input_data;
                stack_pointer <= stack_pointer + 1;
            end else begin
```

```
                overflow <= 1;
            end
            output_data <= {DATA_WIDTH{1'bz}};
        end

        3'b111: // popping
        begin
            if (stack_pointer > 0) begin
                output_data <= stack[stack_pointer-1];
                stack_pointer <= stack_pointer - 1;
            end else begin
                overflow <= 1;
                output_data <= {DATA_WIDTH{1'bz}};
            end
        end

        default: begin
            overflow <= 0;
            output_data <= {DATA_WIDTH{1'bz}};
        end
    endcase
end

endmodule
```

Testbench for STACK_BASED_ALU

```
`timescale 1ns/1ps

module tb_STACK_BASED_ALU;

  parameter DATA_WIDTH = 8;
  parameter STACK_SIZE = 64;

  reg clk;
  reg [DATA_WIDTH-1:0] input_data;
  reg [2:0] opcode;
  wire [DATA_WIDTH-1:0] output_data;
  wire overflow;
  wire [DATA_WIDTH-1:0] debug_value;

  STACK_BASED_ALU #(DATA_WIDTH, STACK_SIZE) alu (
    .clk(clk),
    .input_data(input_data),
    .opcode(opcode),
    .output_data(output_data),
    .overflow(overflow),
    .debug_value(debug_value)
  );

  // Clock generation
  initial begin
    clk = 0;
    forever #5 clk = ~clk; // 10ns period clock
  end

  // Helper function to return operation name
  function [79:0] get_operation_name(input [2:0] opcode);
    case (opcode)
```

```
      3'b100: get_operation_name = "Addition";
      3'b101: get_operation_name = "Multiplication";
      3'b110: get_operation_name = "Push";
      3'b111: get_operation_name = "Pop";
      default: get_operation_name = "Unknown";
    endcase
  endfunction

  // Task to display the stack content
  task display_stack;
    integer i;
    begin
      $display("Stack content:");
      for (i = 0; i < alu.stack_pointer; i = i + 1) begin
        $display("stack[%0d] = %h", i, alu.stack[i]);
      end
    end
  endtask

  // Test procedure
  initial begin
    // Initialize inputs
    input_data = 0;
    opcode = 3'b000;

    // Wait for the clock to stabilize
    #10;

    // Test cases
    $monitor("Time = %0t, Opcode = %b (%0s), Input = %h, Output = %h, Overflow = %b, Stack Pointer = %d, Debug Value = %h",
             $time, opcode, get_operation_name(opcode), input_data, output_data, overflow, alu.stack_pointer, debug_value);

    // Push some values onto the stack
    input_data = 8'h05;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    input_data = 8'h03;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    // Perform addition
    opcode = 3'b100; // Add
    #10;
    display_stack;

    // Push more values
    input_data = 8'h02;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    input_data = 8'h03;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    // Perform addition again
    opcode = 3'b100; // Add
```

```
    #10;
    display_stack;

    input_data = 8'h04;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    // Perform multiplication
    opcode = 3'b101; // Multiply
    #10;
    display_stack;

    // Push more values
    input_data = 8'h06;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    input_data = 8'h02;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    // Perform addition
    opcode = 3'b100; // Add
    #10;
    display_stack;

    // Perform multiplication again
    opcode = 3'b101; // Multiply
    #10;
    display_stack;

    // Pop a value from the stack
    opcode = 3'b111; // Pop
    #10;
    display_stack;

    // Test stack underflow
    opcode = 3'b111; // Pop
    #10;
    display_stack;

    opcode = 3'b111; // Pop
    #10;
    display_stack;

    // Push to stack overflow
    repeat (STACK_SIZE - 1) begin
      input_data = 8'h01;
      opcode = 3'b110; // Push
      #10;
      display_stack;
    end

    // Final push to cause overflow
    input_data = 8'h01;
    opcode = 3'b110; // Push
    #10;
    display_stack;

    // Finish the simulation
```

```
    $finish;
  end

endmodule
```

Output of the Testbench

```
# Time = 10000, Opcode = 110 (Push), Input = 05, Output = zz, Overflow = 0, Stack Pointer =           0, Debug Value = 00
# Time = 15000, Opcode = 110 (Push), Input = 05, Output = zz, Overflow = 0, Stack Pointer =           1, Debug Value = 00
# Stack content:
# stack[0] = 05
# Time = 20000, Opcode = 110 (Push), Input = 03, Output = zz, Overflow = 0, Stack Pointer =           1, Debug Value = 00
# Time = 25000, Opcode = 110 (Push), Input = 03, Output = zz, Overflow = 0, Stack Pointer =           2, Debug Value = 00
# Stack content:
# stack[0] = 05
# stack[1] = 03
# Time = 30000, Opcode = 100 (Addition), Input = 03, Output = zz, Overflow = 0, Stack Pointer =           2, Debug Value = 00
# Time = 35000, Opcode = 100 (Addition), Input = 03, Output = 00, Overflow = 0, Stack Pointer =           1, Debug Value = 05
# Stack content:
# stack[0] = 08
# Time = 40000, Opcode = 110 (Push), Input = 02, Output = 00, Overflow = 0, Stack Pointer =           1, Debug Value = 05
# Time = 45000, Opcode = 110 (Push), Input = 02, Output = zz, Overflow = 0, Stack Pointer =           2, Debug Value = 00
# Stack content:
# stack[0] = 08
# stack[1] = 02
# Time = 50000, Opcode = 110 (Push), Input = 03, Output = zz, Overflow = 0, Stack Pointer =           2, Debug Value = 00
# Time = 55000, Opcode = 110 (Push), Input = 03, Output = zz, Overflow = 0, Stack Pointer =           3, Debug Value = 00
# Stack content:
# stack[0] = 08
# stack[1] = 02
# stack[2] = 03
# Time = 60000, Opcode = 100 (Addition), Input = 03, Output = zz, Overflow = 0, Stack Pointer =           3, Debug Value = 00
# Time = 65000, Opcode = 100 (Addition), Input = 03, Output = 00, Overflow = 0, Stack Pointer =           2, Debug Value = 02
# Stack content:
# stack[0] = 08
# stack[1] = 05
# Time = 70000, Opcode = 110 (Push), Input = 04, Output = 00, Overflow = 0, Stack Pointer =           2, Debug Value = 02
# Time = 75000, Opcode = 110 (Push), Input = 04, Output = zz, Overflow = 0, Stack Pointer =           3, Debug Value = 00
# Stack content:
# stack[0] = 08
# stack[1] = 05
# stack[2] = 04
# Time = 80000, Opcode = 101 (Multiplication), Input = 04, Output = zz, Overflow = 0, Stack Pointer =           3, Debug Value = 00
```

```
# Time = 85000, Opcode = 101 (Multiplication), Input = 04, Output = 00, Overflow = 0, Stack Pointer =           2, Debug Value = 05
# Stack content:
# stack[0] = 08
# stack[1] = 14
# Time = 90000, Opcode = 110 (Push), Input = 06, Output = 00, Overflow = 0, Stack Pointer =           2, Debug Value = 05
# Time = 95000, Opcode = 110 (Push), Input = 06, Output = zz, Overflow = 0, Stack Pointer =           3, Debug Value = 00
# Stack content:
# stack[0] = 08
# stack[1] = 14
# stack[2] = 06
# Time = 100000, Opcode = 110 (Push), Input = 02, Output = zz, Overflow = 0, Stack Pointer =           3, Debug Value = 00
# Time = 105000, Opcode = 110 (Push), Input = 02, Output = zz, Overflow = 0, Stack Pointer =           4, Debug Value = 00
# Stack content:
# stack[0] = 08
# stack[1] = 14
# stack[2] = 06
# stack[3] = 02
# Time = 110000, Opcode = 100 (Addition), Input = 02, Output = zz, Overflow = 0, Stack Pointer =           4, Debug Value = 00
# Time = 115000, Opcode = 100 (Addition), Input = 02, Output = 00, Overflow = 0, Stack Pointer =           3, Debug Value = 06
# Stack content:
# stack[0] = 08
# stack[1] = 14
# stack[2] = 08
# Time = 120000, Opcode = 101 (Multiplication), Input = 02, Output = 00, Overflow = 0, Stack Pointer =           3, Debug Value = 06
# Time = 125000, Opcode = 101 (Multiplication), Input = 02, Output = 06, Overflow = 0, Stack Pointer =           2, Debug Value = 14
# Stack content:
# stack[0] = 08
# stack[1] = 0A
# Time = 130000, Opcode = 111 (Pop), Input = 02, Output = 06, Overflow = 0, Stack Pointer =           2, Debug Value = 14
# Time = 135000, Opcode = 111 (Pop), Input = 02, Output = 0A, Overflow = 0, Stack Pointer =           1, Debug Value = 00
# Stack content:
# stack[0] = 08
# Time = 145000, Opcode = 111 (Pop), Input = 02, Output = 08, Overflow = 0, Stack Pointer =           0, Debug Value = 00
# Stack content:
# Time = 155000, Opcode = 111 (Pop), Input = 02, Output = zz, Overflow = 1, Stack Pointer =           0, Debug Value = 00
# Stack content:
# Time = 160000, Opcode = 110 (Push), Input = 01, Output = zz, Overflow = 1, Stack Pointer =           0, Debug Value = 00
# Time = 165000, Opcode = 110 (Push), Input = 01, Output = zz, Overflow = 0, Stack Pointer =           1, Debug Value = 00
# Stack content:
# stack[0] = 01
# Time = 175000, Opcode = 110 (Push), Input = 01, Output = zz, Overflow = 0, Stack Pointer =           2, Debug Value = 00
# Stack content:
# stack[0] = 01
# stack[1] = 01
```

Calculator Module

- **input:**input must begin with the '(' and must end with the')'. operands and operators appears in order of the string.
- **output:**it shows the stack changing. at the end there is exactly one number in the stack which is the answer.

```
`timescale 1ns/1ps

module Calculator
(
    input clk,                    // Clock signal
    input [2:0] opcode,           // Microcode instruction
    input [15:0] operand           // Input operand
);

reg [2:0] alu_opcode;             // Operation code for ALU
wire [15:0] alu_result;            // Result from ALU
wire overflow;                    // Overflow flag from ALU

// Instantiate the STACK_BASED_ALU
STACK_BASED_ALU #(.DATA_WIDTH(16), .STACK_SIZE(64)) stack_alu (
    .clk(clk),
    .input_data(operand),
    .opcode(alu_opcode),
    .output_data(alu_result),
    .overflow(overflow)
);

integer pending_mult_stack [63:0]; // Stack to keep track of pending
    multiplications
integer pending_mult_index = 0;  // Index for the pending multiplication stack
integer pending_addition_stack [63:0]; // Stack to keep track of pending
    additions
integer pending_addition_index = 0;  // Index for the pending addition stack

// Always block to handle the operations
always @(negedge clk) begin
    // Handle multiplication
    if (pending_mult_stack[pending_mult_index] > 1) begin
        $display("multttttttttttttt");
        alu_opcode = 3'b101;     // Set opcode to multiplication
        #10
        pending_mult_stack[pending_mult_index] = 0;
    end
    // Handle addition
    else if (pending_addition_stack[pending_addition_index] > 1) begin
        $display("addddddddddd ");
        alu_opcode = 3'b100;     // Set opcode to addition
        pending_addition_stack[pending_addition_index] = 0;
    end
    // Handle input reading when no pending operations
    else begin
        $display("lets go");
        case (opcode)
            3'b000: begin // Addition
                $display("babababaababa");
                if (pending_addition_stack[pending_addition_index] == 1) begin
                    alu_opcode = 3'b100;     // Set opcode to addition
                    pending_addition_stack[pending_addition_index] = 1;
                end
```

```
                else begin
                    pending_addition_stack[pending_addition_index] = 1;
                    alu_opcode = 3'b000;
                end
            end
            3'b001: begin // Multiplication
                pending_mult_stack[pending_mult_index] = 1;
                alu_opcode = 3'b000;
            end
            3'b010: begin // Open parenthesis '('
                pending_mult_stack[pending_mult_index + 1] = 0; // Initialize
                    next level multiplication state
                pending_addition_stack[pending_addition_index + 1] = 0; //
                    Initialize next level addition state
                pending_mult_index = pending_mult_index + 1;
                pending_addition_index = pending_addition_index + 1;
                alu_opcode = 0;
            end
            3'b011: begin // Close parenthesis ')'
                if (pending_addition_stack[pending_addition_index] == 1) begin
                    alu_opcode = 3'b100;    // Set opcode to addition
                    pending_addition_stack[pending_addition_index] = 0;
                end
                else  begin
                    alu_opcode = 0;
                end
                pending_addition_index = pending_addition_index - 1;
                pending_mult_index = pending_mult_index - 1;
                if (pending_mult_stack[pending_mult_index] == 1)
                    pending_mult_stack[pending_mult_index] = 2;
            end
            3'b100: begin // Operand
                alu_opcode = 3'b110;
                if (pending_mult_stack[pending_mult_index] == 1)
                    pending_mult_stack[pending_mult_index] = 2;
            end
        endcase
    end
end

endmodule
```

Testbench for Calculator Module

Summary

The test steps follow the expression $(2 \times 3 + (10 + 4 + 3) \times -20 + (6 + 5)) =$.

0.1 Verilog Code

```
`timescale 1ns/1ps

module tb_Calculator;

  reg clk;
  reg [2:0] opcode;
  reg [15:0] operand;
  wire [15:0] alu_result;

```

```
// Instantiate the Calculator
Calculator calculator (
  .clk(clk),
  .opcode(opcode),
  .operand(operand)
);

// Clock generation
initial begin
  clk = 0;
  forever #5 clk = ~clk; // 10ns period clock
end

// Helper function to return operation name
function [79:0] get_operation_name(input [2:0] opcode);
  case (opcode)
    3'b000: get_operation_name = "Addition";
    3'b001: get_operation_name = "Multiplication";
    3'b010: get_operation_name = "Open Parenthesis";
    3'b011: get_operation_name = "Close Parenthesis";
    3'b100: get_operation_name = "Push Operand";
    3'b101: get_operation_name = "Equal Sign";
    default: get_operation_name = "Unknown";
  endcase
endfunction

// Task to display the stack content
task display_stack;
  integer i;
  begin
    $display("Stack content:");
    for (i = 0; i < calculator.stack_alu.stack_pointer; i = i + 1) begin
      $display("stack[%0d] = %h", i, calculator.stack_alu.stack[i]);
    end
  end
endtask

// Task to display the current pending multiplication and addition states
task display_pending_states;
  begin
    $display("Pending multiplication state: %0d", calculator.
        pending_mult_stack[calculator.pending_mult_index]);
    $display("Pending addition state: %0d", calculator.pending_addition_stack
        [calculator.pending_addition_index]);
  end
endtask

// Task to display the progress of the expression calculation
task display_progress(input [8*80:1] expression, input integer current_step);
  begin
    $display("Progress at step %0d: %0s", current_step, expression);
  end
endtask

// Test procedure
initial begin
  // Initialize inputs
  operand = 0;
  opcode = 3'b000;

  // Wait for the clock to stabilize
  #10;

```

```
    // Test cases
    $monitor("Time = %0t, Opcode = %b (%0s), Operand = %h, Alu_Result = %h,
       Pending Mult = %0d, Pending Add = %0d",
             $time, opcode, get_operation_name(opcode), operand, calculator.
                alu_result,
             calculator.pending_mult_stack[calculator.pending_mult_index],
             calculator.pending_addition_stack[calculator.
                pending_addition_index]);

    // Expression: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5))=

    // Step 1: Begin expression with '('
    opcode = 3'b010; // Open Parenthesis
    operand = 16'bz;
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 1);

    // Step 2: Push operand 2
    operand = 16'd2;
    opcode = 3'b100; // Push Operand
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 2);

    // Step 3: Multiply 2 *
    operand = 16'bz;
    opcode = 3'b001; // Multiplication
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 3);

    // Step 4: Push operand 3
    operand = 16'd3;
    opcode = 3'b100; // Push Operand
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 4);

    // Step 5: Addition operation after 2 * 3
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 5);

    // Step 6: Open parenthesis '('
    operand = 16'bz;
    opcode = 3'b010; // Open Parenthesis
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 6);

    // Step 7: Push operand 10
    operand = 16'd10;
    opcode = 3'b100; // Push Operand
    #10;
```

```
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 7);

    // Step 8: Addition 10 +
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 8);

    // Step 9: Push operand 4
    operand = 16'd4;
    opcode = 3'b100; // Push Operand
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 9);

    // Step 10: Addition 10 + 4 +
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 10);

    // Step 11: Push operand 3
    operand = 16'd3;
    opcode = 3'b100; // Push Operand
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 11);

    // Step 12: Addition 10 + 4 + 3
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 12);

    // Step 13: Close parenthesis ')'
    operand = 16'bz;
    opcode = 3'b011; // Close Parenthesis
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 13);

    // Step 14: Multiply (10 + 4 + 3) *
    operand = 16'bz;
    opcode = 3'b001; // Multiplication
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 14);

    // Step 15: Push operand -20
    operand = -16'd20;
    opcode = 3'b100; // Push Operand
```

```
    #20;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 15);

    // Step 16: Addition after first block
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 16);

    // Step 17: Open parenthesis '('
    operand = 16'bz;
    opcode = 3'b010; // Open Parenthesis
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 17);

    // Step 18: Push operand 6
    operand = 16'd6;
    opcode = 3'b100; // Push Operand
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 18);

    // Step 19: Addition 6 +
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 19);

    // Step 20: Push operand 5
    operand = 16'd5;
    opcode = 3'b100; // Push Operand
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 20);

    // Step 21: Close parenthesis ')'
    operand = 16'bz;
    opcode = 3'b011; // Close Parenthesis
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 21);

    // Step 22: Addition after second block
    operand = 16'bz;
    opcode = 3'b000; // Addition
    #10;
    display_stack;
    display_pending_states;
    display_progress("(2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =", 22);

    $finish;
  end
```

```

endmodule
```

Testbench Output

chase the stack values.

Output

```
# lets go
# babababaababa
# lets go
# Time = 10000, Opcode = 010 (arenthesis), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 0
# Stack content:
# Pending multiplication state: 0
# Pending addition state: 0
# Progress at step 1: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 20000, Opcode = 100 (sh Operand), Operand = 0002, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 0
# Stack content:
\textbf{# stack[0] = 0002}
# Pending multiplication state: 0
# Pending addition state: 0
# Progress at step 2: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 30000, Opcode = 001 (iplication), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 1, Pending Add = 0
# Stack content:
\textbf{# stack[0] = 0002}
# Pending multiplication state: 1
# Pending addition state: 0
# Progress at step 3: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 40000, Opcode = 100 (sh Operand), Operand = 0003, Alu_Result = zzzz, Pending Mult = 2, Pending Add = 0
# Stack content:
\textbf{# stack[0] = 0002}
\textbf{# stack[1] = 0003}
# Pending multiplication state: 2
# Pending addition state: 0
# Progress at step 4: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# multttttttttttt
# Time = 55000, Opcode = 100 (sh Operand), Operand = 0003, Alu_Result = 0000, Pending Mult = 2, Pending Add = 0
# lets go
# babababaababa
# Time = 60000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = 0000, Pending Mult = 0, Pending Add = 1
# Time = 65000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 5: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
```

```
# Time = 70000, Opcode = 010 (arenthesis), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 0
# Stack content:
\textbf{# stack[0] = 0006}
# Pending multiplication state: 0
# Pending addition state: 0
# Progress at step 6: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 80000, Opcode = 100 (sh Operand), Operand = 000a, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 0
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 000a}
# Pending multiplication state: 0
# Pending addition state: 0
# Progress at step 7: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# babababaababa
# Time = 90000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 000a}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 8: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 100000, Opcode = 100 (sh Operand), Operand = 0004, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 000a}
\textbf{# stack[2] = 0004}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 9: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# babababaababa
# Time = 110000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 1
# Time = 115000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = 0000, Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 000e}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 10: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 120000, Opcode = 100 (sh Operand), Operand = 0003, Alu_Result = 0000, Pending Mult = 0, Pending Add = 1
# Time = 125000, Opcode = 100 (sh Operand), Operand = 0003, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 000e}
\textbf{# stack[2] = 0003}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 11: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 130000, Opcode = 011 (arenthesis), Operand = zzzz, Alu_Result = zzzz, Pending Mult = 0, Pending Add = 1
```

```
# Time = 135000, Opcode = 011 (arenthesis), Operand = zzzz, Alu_Result = 0000,
    Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 0011}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 13: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 140000, Opcode = 001 (iplication), Operand = zzzz, Alu_Result = 0000,
    Pending Mult = 1, Pending Add = 1
# Time = 145000, Opcode = 001 (iplication), Operand = zzzz, Alu_Result = zzzz,
    Pending Mult = 1, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = 0011}
# Pending multiplication state: 1
# Pending addition state: 1
# Progress at step 14: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 150000, Opcode = 100 (sh Operand), Operand = ffec, Alu_Result = zzzz,
    Pending Mult = 2, Pending Add = 1
# multtttttttttttt
# Time = 165000, Opcode = 100 (sh Operand), Operand = ffec, Alu_Result = 0000,
    Pending Mult = 2, Pending Add = 1
# Stack content:
\textbf{# stack[0] = 0006}
\textbf{# stack[1] = feac}
# Pending multiplication state: 2
# Pending addition state: 1
# Progress at step 15: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# babababaababa
# Time = 170000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = 0000,
    Pending Mult = 0, Pending Add = 1
# Time = 175000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = 0011,
    Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = feb2}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 16: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 180000, Opcode = 010 (arenthesis), Operand = zzzz, Alu_Result = 0011,
    Pending Mult = 0, Pending Add = 0
# Time = 185000, Opcode = 010 (arenthesis), Operand = zzzz, Alu_Result = zzzz,
    Pending Mult = 0, Pending Add = 0
# Stack content:
\textbf{# stack[0] = feb2}
# Pending multiplication state: 0
# Pending addition state: 0
# Progress at step 17: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 190000, Opcode = 100 (sh Operand), Operand = 0006, Alu_Result = zzzz,
    Pending Mult = 0, Pending Add = 0
# Stack content:
\textbf{# stack[0] = feb2}
\textbf{# stack[1] = 0006}
# Pending multiplication state: 0
# Pending addition state: 0
# Progress at step 18: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# babababaababa
```

```
# Time = 200000, Opcode = 000 (Addition), Operand = zzzz, Alu_Result = zzzz,
    Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = feb2}
\textbf{# stack[1] = 0006}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 19: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 210000, Opcode = 100 (sh Operand), Operand = 0005, Alu_Result = zzzz,
    Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = feb2}
\textbf{# stack[1] = 0006}
\textbf{# stack[2] = 0005}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 20: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 220000, Opcode = 011 (arenthesis), Operand = zzzz, Alu_Result = zzzz,
    Pending Mult = 0, Pending Add = 1
# Time = 225000, Opcode = 011 (arenthesis), Operand = zzzz, Alu_Result = 0000,
    Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = feb2}
\textbf{# stack[1] = 000b}
# Pending multiplication state: 0
# Pending addition state: 1
# Progress at step 21: (2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) =
# lets go
# Time = 230000, Opcode = 011 (arenthesis), Operand = zzzz, Alu_Result = 0000,
    Pending Mult = 0, Pending Add = 1
# Time = 235000, Opcode = 011 (arenthesis), Operand = zzzz, Alu_Result = 0006,
    Pending Mult = 0, Pending Add = 1
# Stack content:
\textbf{# stack[0] = febd}
```