

**طراحی سیستم های دیجیتال**

**استاد: دکتر فصحتی**

**صادق محمدیان**

**شماره دانشجویی:۴۰۱۱۰۹۴۷۷**

می خواهیم یک STACK BASED ALU برای اعداد صحیح n بیتی طراحی کنیم .که دارای ورودی و خروجی های زیر می باشد:

* Input data: ورودی n بیتی
* Output data: خروجی n بیتی
* Opcode: opcode ورودی 3-bit که نشان می دهد چه عملیاتی باید انجام شود.
* Overflow: خروجی بیت سرریز
* Clk
* Rst
* محل اشارهstack pointer

ماژول ما باید از عملیات های زیر پیروی کند:

عملیات های ضرب و جمع تغییری در پشته ما ایجاد نمی کند. و اپرند های ما دوعدد بالایی پشته هستند.

ابتدا این مدار را طراحی می کنیم و برای آن ماژول تست می نویسیم. یک استک با عمق ۳۲ در نظر میگیریم در نتیجه stack pointer به بیت نیاز دارد.

در صورتی که سیگنال rst فعال شود sp و output\_data و overflow را صفر می کنیم.

و با push کردن بعد از قرار دادن داده در محلی که اشاره گر نشان می دهد اشاره گره را به خانه بالا تر می بریم و با pop کردن داده ای که پوینتر نشان می دهد را نشان می دهیم و سپس آن را صفر می کنیم در استک و در نهایت اشاره گر را به خانه پایینی می بریم.

در جمع کردن و ضرب کردن پس از چک کردن شرط وجود حداقل دو اپرند حاصل عملیات مورد نظر را خروجی می دهیم و برای چک کردن overflow به این صورت عمل می کنیم ابتدا دو اپرند را sign extend می کنیم سپس عملیات را انجام می دهیم و سپس نتیجه که داشتیم را sign extend می کنیم و با مقایسه این دو مقدار اگر برابر نبود یعنی overflow رخ داده است.

کد وریلاگ زیر مربوط به طراحی ما می باشد:

module STACK\_BASED\_ALU #(parameter n = 32) (input signed [n-1:0] input\_data,input clk,input rst,input [2:0] opcode,output reg signed [n-1:0] output\_data,output reg overflow,output reg [4:0] sp);

    reg signed [n-1:0] stack [0:31];

    reg signed [2\*n-1:0] real\_res;

    reg signed  [2\*n-1:0] se\_out;

    reg signed  [2\*n-1:0]  se\_stacko;

    reg signed  [2\*n-1:0]  se\_stackt;

    integer i;

    always @(posedge clk or posedge rst)

    begin

        if (rst)

        begin

            sp = 0;

            output\_data = 0;

            overflow = 0;

        end

        else

        begin

            case (opcode)

                3'b100:

                begin // Addition

                    if (sp >= 2)

                    begin

                        output\_data = stack[sp-1] + stack[sp-2];

                        se\_out={ {n{output\_data[n-1]}}, output\_data };

                        se\_stacko={ {n{stack[sp-1][n-1]}}, stack[sp-1] };

                        se\_stackt={ {n{stack[sp-2][n-1]}}, stack[sp-2] };

                        real\_res = se\_stacko + se\_stackt ;

                        if(real\_res==se\_out)

                        begin

                            overflow=0;

                        end

                        else

                        begin

                            overflow=1;

                        end

                    end

                    else

                    begin

                        overflow <= 0;

                    end

                end

                3'b101:

                begin // Multiply

                    if (sp >= 2)

                    begin

                        output\_data = stack[sp-1] \* stack[sp-2];

                        se\_out={ {n{output\_data[n-1]}}, output\_data };

                        se\_stacko={ {n{stack[sp-1][n-1]}}, stack[sp-1] };

                        se\_stackt={ {n{stack[sp-2][n-1]}}, stack[sp-2] };

                        real\_res = se\_stacko \* se\_stackt ;

                        if(real\_res==se\_out)

                        begin

                            overflow=0;

                        end

                        else

                        begin

                            overflow=1;

                        end

                    end

                    else

                    begin

                        overflow <= 0;

                    end

                end

                3'b110: begin // PUSH

                    if (sp < 32)

                    begin

                        stack[sp] <= input\_data;

                        sp <= sp + 1;

                    end

                end

                3'b111: begin // POP

                    if (sp > 0)

                    begin

                        output\_data <= stack[sp-1];

                        stack[sp-1] = {n{0}};

                        sp <= sp - 1;

                    end

                end

                default:

                begin

                    output\_data <= output\_data;

                    overflow <= overflow;

                end

            endcase

        end

    end

endmodule

حال ماژول تست خود را می نویسیم تا از عملکرد مدار خود مطمعن شویم:

کلاک را می سازیم و از ماژول بالا اینستنس می گیریم

حالت n=32:

module tb\_ALU;

    reg signed [31:0] input\_data;

    wire signed [31:0] output\_data;

    reg [2:0] opcode;

    wire overflow;

    reg clk, rst;

    parameter n = 32;

    wire [4:0] sp;

    STACK\_BASED\_ALU #(n) ALU (input\_data[n-1:0],clk,rst,opcode,output\_data[n-1:0],overflow,sp);

    initial

    begin

        clk = 0;

        //forever #5 clk = ~clk;

    end

    always #5 clk = ~clk;

    initial

    begin

        // Test sequence

        rst = 1; #10;

        rst = 0; #10;

        // Push 10

        input\_data = 1000000000; opcode = 3'b110; #10;

        // Display stack state

        $display("SP = %d", sp);

        // Push 20

        input\_data = -1000000000; opcode = 3'b110; #10;

        // Display stack state

        $display("SP = %d", sp);

        // Add top two

        opcode = 3'b100; #10;

        // Check result

        $display("SUM: %d, Overflow: %b", output\_data, overflow);

        // Push 30

        input\_data = 1000000000; opcode = 3'b110; #10;

        // Display stack state

        $display("SP = %d", sp);

        // Multiply top two

        opcode = 3'b101; #10;

        // Check result

        $display("MUL: %d, Overflow: %b", output\_data, overflow);

        // Pop

        opcode = 3'b111; #10;

        // Check result

        $display("%d, SP = %d", output\_data, sp);

        $stop;

    end

endmodule

نتیجه:



*حالت* n=16 *:*

module tb\_ALU16;

    reg signed [15:0] input\_data;

    wire signed [15:0] output\_data;

    reg [2:0] opcode;

    wire overflow;

    reg clk, rst;

    parameter n = 16;

    wire [4:0] sp;

    STACK\_BASED\_ALU #(n) ALU (input\_data[n-1:0],clk,rst,opcode,output\_data[n-1:0],overflow,sp);

    initial

    begin

        clk = 0;

    end

    always #5 clk = ~clk;

    initial

    begin

        rst = 1; #10;

        rst = 0; #10;

        input\_data = 32'd10; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        input\_data = -1000; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        // testing add operation

        opcode = 3'b100; #10;

        $display("SUM: %d, Overflow: %b", output\_data, overflow);

        input\_data = 100; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        // testing mul operation

        opcode = 3'b101; #10;

        $display("MUL: %d, Overflow: %b", output\_data, overflow);

        // Pop

        opcode = 3'b111; #10;

        // Check result

        $display("%d, SP = %d", output\_data, sp);

        $stop;

    end

endmodule

*نتایج:*

**

حالت n=8:

module tb\_ALU8;

    reg signed [7:0] input\_data;

    wire signed [7:0] output\_data;

    reg [2:0] opcode;

    wire overflow;

    reg clk, rst;

    parameter n = 8;

    wire [4:0] sp;

    STACK\_BASED\_ALU #(n) ALU (input\_data[n-1:0],clk,rst,opcode,output\_data[n-1:0],overflow,sp);

    initial

    begin

        clk = 0;

    end

    always #5 clk = ~clk;

    initial

    begin

        rst = 1; #10;

        rst = 0; #10;

        input\_data = 32'd10; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        input\_data = -32'd20; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        // testing add operation

        opcode = 3'b100; #10;

        $display("SUM: %d, Overflow: %b", output\_data, overflow);

        input\_data = 32'd30; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        // testing mul operation

        opcode = 3'b101; #10;

        $display("MUL: %d, Overflow: %b", output\_data, overflow);

        // Pop

        opcode = 3'b111; #10;

        // Check result

        $display("%d, SP = %d", output\_data, sp);

        $stop;

    end

endmodule

نتایج:



حالت n=4:

module tb\_ALU4;

    reg signed [3:0] input\_data;

    wire signed [3:0] output\_data;

    reg [2:0] opcode;

    wire overflow;

    reg clk, rst;

    parameter n = 4;

    wire [4:0] sp;

    STACK\_BASED\_ALU #(n) ALU (input\_data[n-1:0],clk,rst,opcode,output\_data[n-1:0],overflow,sp);

    initial

    begin

        clk = 0;

    end

    always #5 clk = ~clk;

    initial

    begin

        rst = 1; #10;

        rst = 0; #10;

        input\_data = 7; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        input\_data = 7; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        // testing add operation

        opcode = 3'b100; #10;

        $display("SUM: %d, Overflow: %b", output\_data, overflow);

        input\_data = 4; opcode = 3'b110; #10;

        // show stack pointer

        $display("SP = %d", sp);

        // testing mul operation

        opcode = 3'b101; #10;

        $display("MUL: %d, Overflow: %b", output\_data, overflow);

        // Pop

        opcode = 3'b111; #10;

        // Check result

        $display("%d, SP = %d", output\_data, sp);

        $stop;

    end

endmodule

نتایج:



حال به سراغ بخش دوم سوال می رویم:

ابتدا باید این عبارت ورودی را از infix به posfix تبدیل کنیم. برای این کار از الگوریتم گفته شده در این لینک استفاده می کنیم.

<https://www.geeksforgeeks.org/convert-infix-expression-to-postfix-expression>

برای محاسبه حاصل یک عبارت posfix با استک از لینک زیر کمک می گیریم:

<https://www.geeksforgeeks.org/evaluation-of-postfix-expression>

این ماژول یک عبارت به طول 400 را ورودی میگیرد و با یک خروجی ۵۰ بیتی خروجی می دهد و از دو استک با عمق ۳۲ بهره می برد .

module EXP\_CALC (

    input wire [399:0] expression,

    output reg [49:0] output\_value,

    input wire rst,

    input wire clk

);

از یکی از استک ها برای تبدیل عبارت از infix به posfix استفاده می شود و از دیگری برای محاسبه ی حاصل عبارت استفاده می شود .برای هرکدام یک کلاک جدا و مستقل تعریف می کنیم تا از ماژول قسمت الف استفاده کنیم و آن ها را کنترل کنیم.

سپس متغیر های مورد نیاز را تعریف می کنیم:

    reg [32\*50+7:0] postfixExpression;

    reg [7:0] stackInput;

    wire [7:0] stackOutput;

    reg [2:0] stackOp;

    reg [2:0] calculatorOp;

    wire stackOf;

    wire [5-1:0] sp;

    wire calculatorOf;

    wire [5-1:0] calculatorSp;

    parameter n = 8;

    reg isNeg;

    reg stackClk;

    reg calculatorClk;

    integer i;

    integer idx;

    integer j;

    reg signed [50-1:0] inputCalculator;

    wire [50-1:0] outputCalculator;

    reg [50-1:0] tmp;

ما در هر دفعه اعداد را میخوانیم و آن هارا یا یک کاراکتز w از هم جدا می کنیم و در posfixExpression میریزیم تا اعداد را از هم بتوانیم تشخیص دهیم و با ALU دوم مقدار نهایی را حساب می کنیم.

حال استک هارا تعریف می کنیم: یکی برای تبدیل عبارت ها و دیگری برای انجام عملیات منطقی

    STACK\_BASED\_ALU #(n) stack (

        .input\_data(stackInput[n-1:0]),

        .output\_data(stackOutput[n-1:0]),

        .opcode(stackOp),

        .overflow(stackOf),

        .sp(sp),

        .clk(stackClk),

        .rst(rst)

    );

    STACK\_BASED\_ALU #(50) calculator (

        .input\_data(inputCalculator[50-1:0]),

        .output\_data(outputCalculator[50-1:0]),

        .opcode(calculatorOp),

        .overflow(calculatorOf),

        .sp(calculatorSp),

        .clk(calculatorClk),

        .rst(rst)

    );

حال در یک بلاک always با لیست حساسیت rst و clk عملیات های مورد نظر را پیاده سازی می کنیم:

سیگنال ریست:

        if (rst)

        begin

            postfixExpression = {(8\*50){0}};

            stackClk = 0;

            calculatorClk = 0;

        end

        else

و این تکه مربوط به تبدیل عبارت و عملیت منطقی می باشد که در هرقسمت با کامنت عملکرد هر بخش مشخص شده است:

 postfixExpression = {(8\*50){0}};

            stackClk = 0;

            calculatorClk = 0;

            idx = 1;

            for (i = 50; i > 0; i = i - 1)

            begin

                if ((expression[8\*i-1 -: 8] >= "0" && expression[8\*i-1 -: 8] <= "9") || expression[8\*i-1 -: 8] == "-")

                begin

                    postfixExpression[8\*idx-1 -: 8] = expression[8\*i-1 -: 8];

                    idx = idx + 1;

                end

                else

                begin

                    case (expression[8\*i-1 -: 8])

                        "\*": begin

                            postfixExpression[8\*idx-1 -: 8] = "W";

                            idx = idx + 1; // Append W to postfix expression as a splitter between different numbers

                            for (j = 1 - (|sp) + 1; j < 1; j = j + 1)

                            begin

                                stackOp = 3'b111; #1; stackClk = 1;  #1; stackClk = 0;

                                if (stackOutput[7:0] == "(" || stackOutput[7:0] == "+")

                                begin

                                    stackOp = 3'b110; // opcode for pushing popped data

                                    stackInput[7:0] = stackOutput[7:0]; #1; stackClk = 1;  #1; stackClk = 0;

                                end

                                else

                                if (stackOutput[7:0] == "\*")

                                begin

                                    postfixExpression[8\*idx-1 -: 8] = stackOutput[7:0];

                                    idx = idx + 1;

                                    j = 0 - (|sp);

                                end

                            end

                            stackOp = 3'b110; // opcode for pushing

                            stackInput[7:0] = "\*"; #1; stackClk = 1; #1; stackClk = 0;

                        end

                        "(": begin

                            stackOp = 3'b110; // opcode for pushing

                            stackInput[7:0] = "("; #1; stackClk = 1; #1; stackClk = 0;

                        end

                        ")": begin

                            postfixExpression[8\*idx-1 -: 8] = "W";

                            idx = idx + 1; // Append W to postfix expression as a splitter between different numbers

                            for (j = 0; j < 1; j = j + 1)

                            begin

                                stackOp = 3'b111; #1; stackClk = 1; #1; stackClk = 0;

                                if (stackOutput[7:0] != "(")

                                begin

                                    postfixExpression[8\*idx-1 -: 8] = stackOutput[7:0];

                                    idx = idx + 1;

                                    j = j - 1;

                                end

                            end

                        end

                        "+": begin

                            postfixExpression[8\*idx-1 -: 8] = "W";

                            idx = idx + 1; // Append W to postfix expression as a splitter between different numbers

                            for (j = 1 - (|sp); j < 1; j = j + 1)

                            begin

                                stackOp = 3'b111; #1; stackClk = 1; #1; stackClk = 0;

                                if (stackOutput[7:0] == "(")

                                begin

                                    stackOp = 3'b110; // opcode for pushing popped data

                                    stackInput[7:0] = stackOutput[7:0]; #1; stackClk = 1; #1; stackClk = 0;

                                end

                                else if (stackOutput[7:0] == "\*" || stackOutput[7:0] == "+")

                                begin

                                    postfixExpression[8\*idx-1 -: 8] = stackOutput[7:0];

                                    idx = idx + 1;

                                    j = 0 - (|sp);

                                end

                            end

                            stackOp = 3'b110; // opcode for pushing

                            stackInput[7:0] = "+"; #1; stackClk = 1; #1; stackClk = 0;

                        end

                        default:

                        begin

                        end

                    endcase

                end

            end

            postfixExpression[8\*idx-1 -: 8] = "W";

            idx = idx + 1;

            for (j = 1 - (|sp); j < 1; j = j + 1)

            begin

                stackOp = 3'b111; // opcode for popping whatever is left on the stack

                #1;

                stackClk = 1;

                #1;

                stackClk = 0;

                postfixExpression[8\*idx-1 -: 8] = stackOutput[7:0];

                idx = idx + 1;

                j = 0 - (|sp);

            end

            inputCalculator[50-1:0] = {50{0}};

            for (i = 1; i <= 4\*50 + 1; i = i + 1)

            begin

                if ((postfixExpression[8\*i-1 -: 8] >= "0" && postfixExpression[8\*i-1 -: 8] <= "9") || postfixExpression[8\*i-1 -: 8] == "-") //start of converting

                begin

                    isNeg = 0;

                    isNeg=(postfixExpression[8\*i-1 -: 8] == "-") ? 1 : 0;

                    i=(postfixExpression[8\*i-1 -: 8] == "-") ? i + 1 : i;

                    for (j = 0; j < 1; j = j + 1)

                    begin

                        inputCalculator[50-1:0] = inputCalculator[50-1:0] \* 10 + (postfixExpression[8\*i-1 -: 8] - "0");

                        i = i + 1;

                        if (postfixExpression[8\*i-1 -: 8] == "W")

                        begin

                            j=0;

                        end

                        else

                        begin

                            j=-1;

                        end

                    end

                    i = i - 1;

                    inputCalculator[50-1:0] =(isNeg == 1) ? -inputCalculator[50-1:0] : inputCalculator[50-1:0];

                    calculatorOp = 3'b110; #1; calculatorClk = 1; #1;calculatorClk = 0;inputCalculator[50-1:0] = {50{0}};

                end

                else if (postfixExpression[8\*i-1 -: 8] != "W" && postfixExpression[8\*i-1 -: 8] != 0)

                begin

                    calculatorOp = (postfixExpression[8\*i-1 -: 8] == "\*") ? 3'b101 : 3'b100; // opcode for addition

                    #1; calculatorClk = 1; #1; calculatorClk = 0;

                    tmp[50-1:0] = outputCalculator[50-1:0];

                    calculatorOp = 3'b111;  #1; calculatorClk = 1; #1; calculatorClk = 0; calculatorOp = 3'b111;  #1; calculatorClk = 1; #1; calculatorClk = 0;

                    inputCalculator[50-1:0] = tmp[50-1:0]; calculatorOp = 3'b110; #1; calculatorClk = 1; #1; calculatorClk = 0;

                    inputCalculator[50-1:0] = {50{0}};

                end

            end

            calculatorOp = 3'b111; #1; calculatorClk = 1; #1; calculatorClk = 0;

            output\_value[50-1:0] = outputCalculator[50-1:0];

        end

حال ماژول تست خود را می نویسیم طبق معمول مقادیر ورودی ماژول را تعریف می کنیم سپس از ماژول

EXP\_CALC اینستنس می گیریم .

module tb\_EXP\_CALC;

    reg [399:0] expression;

    reg clk, rst;

    wire signed [49:0] output\_value;

    EXP\_CALC calculator (expression,output\_value,rst,clk);

    initial

    begin

        rst = 0; #10;

        rst = 1; #10;

        rst = 0; #10;   //reset the modules.

        expression = "(((7+3)\*5)+8)\*2";

        $display("TB expression is: %s", expression);

        clk = 0; #1000;

        clk = 1; #1000;

        $display("output\_value is: %0d", output\_value);

        expression = "2\*3+(10+4+3)\*-20+(6+5)";

        $display("TB expression is: %s", expression);

        clk = 0; #1000;

        clk = 1; #1000;

        $display("output\_value is: %0d", output\_value);

        expression = "1+2\*3+8+9";

        $display("TB expression is: %s", expression);

        clk = 0; #1000;

        clk = 1; #1000;

        $display("output\_value is: %0d", output\_value);

        expression = "2\*3+(10+4+3)\*-20+((6+5)\*(8+1)\*(9+1))+2";

        $display("TB expression is: %s", expression);

        clk = 0; #1000;

        clk = 1; #1000;

        $display("output\_value is: %0d", output\_value);

        expression = "1+(-2+0)";

        $display("TB expression is: %s", expression);

        clk = 0; #1000;

        clk = 1; #1000;

        $display("output\_value is: %0d", output\_value);

        $stop;

    end

endmodule

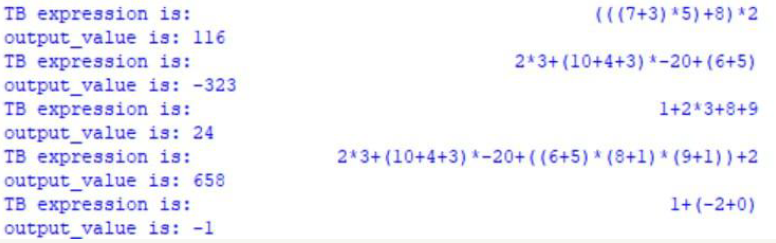
در تست اول عبارت (((7+3)\*5)+8)\*2 را بررسی می کنیم که حاصل آن 116میباشد.

در تست دوم عبارت 2\*3+(10+4+3)\*-20+(6+5) را مورد بررسی قرار می دهیم که حاصل آن -323می باشد

در تست سوم عبارت 1+2\*3+8+9مورد بررسی قرار می دهیم که حاصل آن 24 می باشد.

در تست چهارم عبارت 2\*3+(10+4+3)\*-20+((6+5)\*(8+1)\*(9+1))+2 را حساب می کنیم که برابر با 658 می باشد.

در تست آخر نیز حاصل1+(-2+0) را مورد بررسی قرار می دهیم که حاصل آن -1 می باشد

و نتایج شبیه سازی مطابق زیر می باشد که نشان میدهد کد ما به درستی کار می کند: