محمد حسین اسلامی

۴۰۱۱۰۵۵۳۷

طراحی سیستم‌های دیجیتال

سوال جبرانی

به عنوان سوال جبرانی، بخش اول سوال یک میانترم انجام شده است.

طراحی مدار:

مقدمه:

مدار شامل یک ماژول است که در این ماژول ما ALU را طراحی کرده‌ایم که با مکانیزم پشته کار می‌کند. منظور ما این است که این واحد می‌تواند صرفا با گرفتن یک دستور، عملیات ریاضی را برروی دو عضو بالایی پشته خود انجام دهد.

ورودی و خروجی:

مدار ما شامل چندین ورودی و خورجی می‌باشد که در ادامه به توضیح هرکدام می‌پردازیم.

ورودی:

Clk: این سیگنال به عنوان clock برنامه عمل می‌کند.

rst_n: این سیگنال که به صورت asynchronous و active-low است برای خالی کردن پشته و resest کردن ALU استفاده می‌شود.

input_data: این ورودی که signed می‌باشد برای push کردن مقادیر دلخواه در پشته استفاده می‌شود.

Opcode: شامل ۳ بیت می‌باشد که نوع عملیات ALU را مشخص می‌کند. در جلوتر به مقادیر ممکن opcode و کارایی هر کدام اشاره می‌کنیم.

خروجی:

output_data: این خروجی که signed است نتیجه عملیات‌های حسابی ما را و یا مقدار pop شده را نمایش می‌دهد.

overflow: می‌دانیم که عملیات‌های ریاضی می‌توانند *overflow* کنند بنابراین به یک بیت برای نشان دادن چنین حالتی نیاز داریم.

```
module stack_alu #(
    parameter N
) (
    input clk,
    input rst_n,
    input signed [N-1:0] input_data,
    input [2:0] opcode,
    output reg signed [N-1:0] output_data,
    output reg overflow
);
```

مقادیر Opcode:

در ادامه به مقادیر ممکن برای opcode می‌پردازیم (دقت شود که با توجه به صورت سوال، هرگونه عملیات محاسباتی تغییری در پشته ایجاد نمی‌کند):

100: با این مقدار، ALU دو مقدار بالای پشته را برداشته و عملیات جمع را برروی آن‌ها انجام می‌دهد و مقدار جمع را در output_data می‌ریزد.

101: این opcode برای انجام دستور ضرب است که مشابه دستور جمع عمل می‌کند با این تفاوت که ضرب دو عدد مشخص شده را در خروجی قرار می‌دهد.

110: این دستور، عملیات push را برای ما انجام می‌دهد و عددی را که در input_data قرار دارد را بالای پشته قرار می‌دهد.

111: این opcode برای عملیات pop استفاده می‌شود و با استفاده از آن مقدار بالای پشته در خروجی قرار می‌گیرد و از پشته خارج می‌شود.

0xx: این دستور No-Op است و به این معناست که اگر opcode MSB مقداری بگیرد هیچ‌گونه عملیاتی در ALU انجام نمی‌شود.

نحوه پیاده‌سازی مدار:

در ادامه به بخش‌های مختلف کد می‌پردازیم و نحوه کارکرد هر بخش را مشخص می‌کنیم.

در ابتدا باید تعدادی متغیر کمکی تعریف کنیم تا بتوانیم ALU و پشته را به کمک آن‌ها پیاده کنیم:

Stack: پشته در دید ما و این برنامه به مانند یک بلوک حافظه است که دو بعدیست. بعد اول آن که تعداد واحدهای پشته می‌شوند مقدار ثابت ۱۰۲۴ را دارد اما بعد دوم آن که در تعداد بیت‌های موجود در هر واحد را نشان می‌دهد به صورت یک پارامتر تعریف شده است. در واقع نحوه تعریف پشته به این صورت است که ۱۰۲۴ بلوک N بیتی باید بسازیم.

Sp: می‌دانیم برای آن که بتوانیم بالای یک پشته را نمایش دهیم و به عنصر بالایی آن دسترسی داشته باشیم به یک stack pointer احتیاج داریم. در مدار ما stack pointer به اولین خانه خالی پس از یک خانه پر اشاره می‌کند.

temp_result_add: این متغیر برای نگاه داری مقدار جمع استفاده می‌شود. دقت شود که این متغیر N بیتی است.

temp_result_mult: مانند متغیر قبلی برای نگه‌داری استفاده می‌شود با این تفاوت که نتیجه ضرب را در خود می‌ریزد.

overflow_control: این متغیر برای کنترل overflow در عملیات ضرب استفاده می‌شود که در ادامه دلیل وجود آن را توضیح می‌دهیم.

```
reg signed [N-1:0] stack [0:1023]; //here we are defining a momery block that will work
reg [10:0] sp; //Stackpointer

reg signed [N-1:0] temp_result_add;
reg signed [N-1: 0] temp_result_mult;
integer signed overflow_control;
```

در ادامه باید در ابتدای کار ALU تمامی مقادیر متغیرهای کمکی را ست کرده و برابر با صفر قرار داد و آن‌ها را initialize کرد.

```
initial begin //this block initializes
    sp = 0;
    overflow = 0;
    temp_result_add = 0;
    temp_result_mult = 0;
    overflow_control = 0;
end
```

حال به سراغ قسمت اصلی مدار می‌رویم. برای ورود به این قسمت دو شرط داریم. یک زمانی که clock به لبه بالارونده خود برسد و یا زمانی که rst_n صفر شود. این قسمت از مدار را به کمک یک بلوک always طراحی کردیم که لیست حساسیت آن را در قسمت قبل مشخص کردیم. در کل دقت شود که در این مدار ما پیام‌های خطایی نیز با توجه به شرایط صادر می‌کنیم و در کنار آن مقدار خروجی مدار را x می‌کنیم تا کاربر بتواند به خروجی معتبر و نامعتبر تفکیک قائل شود.

در ابتدای این بلوک، ابتدا بررسی می‌شود که آیا مقدار rst_n صفر هست یا نه. اگر چنین باشد و مقدار rst_n صفر باشد، کلیه پشته خالی می‌شود (دقت شود که در این جا منظور از خالی شده پشته، صفر شدن خانه‌های حافظه نیست بلکه جابه‌جایی sp است طوری که دیگر مقادیر در پشته قابل دسترسی نیستند)، sp برابر با صفر می‌شود و تمام خروجی‌های ALU صفر می‌شوند.

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        sp <= 0;
        overflow <= 0;
        output_data <= 0;
```

حال به پیاده‌سازی دستورات مختلف می‌پردازیم. به طور کلی این قسمت از بلوک always به کمک case بزرگی زده شده است که تمامی حالاتی که opcode می‌تواند به خود بگیرد را پیاده می‌کند. در ابتدا اگر 100 opcode باشد عملیات جمع انجام می‌شود. در ابتدای عملیات بررسی می‌شود که آیا حداقل دو عدد در پشته وجود دارند تا بتوانند عملیات جمع را انجام دهند یا نه. اگر چنین نبود پیغام خطای مناسب نشان داده می‌شود و اگر به اندازه کافی عدد برای انجام جمع وجود داشت، عملیات انجام می‌شود و مقدار جواب در خروجی قرار می‌گیرد. بعد از آن باید مقدار overflow را بررسی کنیم. می‌دانیم که زمانی در عملیات جمع overflow صورت می‌گیرد که جمع دو عدد هم علامت، جوابی با علامت مخالف داشته باشد و اگر دو عدد مختلف العلامت را با هم جمع کنیم هیچ‌گاه overflow رخ نمی‌دهد بنابراین ابتدا بررسی می‌کنیم که آیا دو عدد علامت مختلفی دارند و یا نه. اگر چنین باشد که overflow صفر می‌شود و اگر چنین نباشد، مقدار overflow با توجه به xor علامت یکی از ورودی‌های عملیات و خروجی عملیات می‌شود. همانور که معلوم است دو عدد بالای پشته را برداشته و عملیات را برروی آن‌ها انجام می‌دهیم. همچنین علامت ورودی‌ها و خروجی را از روی بیت *MSB* آن تشخیص می‌دهیم.

```
end else begin
    case (opcode)
        3'b100: begin
            if (sp >= 2) begin
                temp_result_add = stack[sp-1] + stack[sp - 2];
                output_data = temp_result_add[N-1:0];
                if (stack[sp-1][N-1] ^ stack[sp-2][N-1] == 1 ) begin
                    overflow = 0;
                end
                else begin
                    overflow = stack[sp-1][N-1] ^ temp_result_add[N-1];
                end
            end else begin
                $display("You do not have enough numbers to perform addition");
                output_data = 1'bx;
                overflow = 0;
            end
        end
```

بعد از این دستور به سراغ دستور ضرب می‌رویم. این دستور بسیار مانند دستور قبل است با این تفاوت که باید عملیات ضرب را برروی دو عدد بالای پشته انجام دهیم. از جایی که می‌دانیم که خروجی ضرب دو عدد *N* بیتی می‌تواند تا *2N* بیت نیز برود ولی خروجی ما *N* بیتی است، باید *N* بیت اول خروجی را در خروجی نهایی مدار قرار دهیم. در این قسمت برای تشخیص *overflow* کردن جواب، یک بار جواب اصلی‌ای که بدست آمده بود را تقسیم بر یکی از اعداد ورودی عملیات می‌کنیم تا ببینیم با مقداری که در خروجی قرار دادیم برابر است یا نه. اگر چنین باشد که یعنی هیچ بیتی از خروجی زده نشده است اما اگر چنین شود و جواب‌های متفاوتی داشته باشیم، این به ما نشان می‌دهد که مقدار موجود در *temp_result_mul* در *N* بیت جا نمی‌شده است پس ضرب ما *overflow* کرده است. البته در ابتدا بررسی می‌کنیم که تقسیمی که برای تشخیص *overflow* است بر صفر نباشد. از طرفی خیالمان راحت است که ضرب یک عدد درصفر *overflow* نمی‌کند زیرا جواب آن همیشه صفر می‌شود. دلیل وجود *overflow_control* به عنوان یک *signed integer* این است که بتوانیم تمام جواب تقسیم را در جایی ذخیره بکنیم. به عنوان مثال دو عدد *8-* و *1-* را در نظر بگیرید. جواب ضرب آن‌ها در ۴ بیت *overflow* می‌کند و خروجی مدار ما *8-* می‌شود. حال اگر جواب تقسیم را در ۴ بیت قرار دهیم، جواب تقسیم *8-* می‌شود با این که انتظار داریم تقسیم دو عدد منفی مثبت شود. به دلیل آن که جواب تقسیم را محدود کردیم، برنامه به عنوان یک عدد منفی نگاه می‌کند اما نمی‌داند که جواب تقسیم در بیت چهارم آن یک بوده و در بیت‌های بعدی آن صفر بوده سات پس جواب باید *8* باشد و نه منفی *8*. به همین دلیل است که اگر جواب تقسیم را محدود کنیم در این حالت خاص *overflow* نداریم اما در واقعیت می‌دانیم که *overflow* رخ می‌دهد.

```
end
3'b101: begin
    if (sp >= 2) begin
        temp_result_mult = stack[sp-1] * stack[sp-2];
        output_data = temp_result_mult[N-1:0];
        overflow_control = temp_result_mult / stack[sp- 1];
        if (stack[sp-1] != 0 && overflow_control != stack[sp-2])
            overflow = 1;
        else begin
            overflow = 0;
        end

    end else begin
            $display("You do not have enough numbers to perform multiplication");
            output_data = 1'bx;
            overflow = 0;
        end
end
```

در ادامه به عملیات *push* می‌پردازیم. در ابتدای این بخش بررسی می‌شود که آیا پشته جای خالی برای یک داده جدید دارد یا نه. اگر جای خالی نبود، پیغام مناسب به کاربر نشان داده می‌شود. در ادامه، مقدار موردنظر در ورودی قرار می‌گیرد. سپس این مقدار در خانه‌ای که *sp* به آن اشاره دارد قرار می‌گیرد و در ادامه *sp* یک مقدار افزایش می‌یابد تا به خانه خالی بعدی اشاره بکند. دقت شود که *sp* همیشه به اولین خانه خالی بعد از بالای پشته اشاره می‌کند.

```
3'b110: begin
    if (sp < 1024) begin
        stack[sp] = input_data;
        sp = sp + 1;
    end else begin
        $display("There is not enough space for another push");
    end
    output_data = 1'bx;
    overflow = 0;
end
```

آخرین عملیاتی که می‌خواهیم به آن پردازیم، عملیات *pop* است. وقتی *opcode* دستور *pop* به *ALU* داده می‌شود، مدار در ابتدا بررسی می‌کند که آیا پشته خالی هست یا نه. اگر پشته خالی باشد که پیغام مناسب نمایش داده می‌شود و کاربر نمی‌تواند مقداری از پشته بخواند اما اگر پشته خالی نباشد، در ابتدا مقدار خانه‌ای که *sp-1* به آن اشاره دارد در خروجی قرار می‌گیرد و سپس یک واحد از *sp* کم می‌شود. نکته قابل توجه در این دستور این است که با *pop* کردن، خانه حافظه‌ای که مقدار در آن قرار داشت پاک نمی‌شود بلکه دیگر غیرقابل دسترس می‌شود و تنها در حالتی می‌توان به مقداری که قبلا در آن خانه داشتیم برگردیم که همان مقدار را دوباره *push* کنیم.

```
3'b111: begin
    if (sp > 0) begin
        output_data = stack[sp - 1];
        sp = sp - 1;
    end else begin
        $display("There is no number to pop. The stack is empty.");
        output_data = 1'bx;
    end
end
```

در نهایت و در قسمت *case default* ای که نوشتیم، دستور *No-Op* قرار دارد که این دستور هیچ عملی را انجام نمی‌دهد و صرفا به کاربر می‌گوید که هیچ عملیاتی در حال انجام نیست. در این قسمت خروجی صفر می‌شود و *overflow* نیز همچنین زیرا هیچ عملیات محاسباتی‌ای در حال انجام نیست. دقت شود که در این قسمت نباید هیچ تغییری در پشته رخ دهد که رخ نمی‌دهد.

```
        default: begin
            $display("No Operation is happening right now");
            output_data = 0;
            overflow = 0;
        end
    endcase
```

Test Bench:

در ادامه می‌خواهیم به نحوه کارکرد test bench و مواردی که در این test bench بررسی می‌شوند بپردازیم. در ابتدای کار test bench یک وinstance از ماژول موردنظر می‌گیریم و همچنین تعدادی متغیر کمکی برای انجام test تعریف می‌کنیم. همچنین یک بلوک برای شبیه‌سازی clock نیزز تعریف می‌شود. همچنین در ابتدای بلوک initial مقادیر مورد نیاز به صفر initialize می‌شوند.

```
module stack_based_ALU_tb;

    parameter N = 32;
    reg clk = 0;
    reg rst_n;
    reg [2:0] opcode;
    reg signed [N-1:0] input_data;
    wire signed [N-1:0] output_data;
    wire overflow;

    integer signed count, count1;
    integer signed overflow_count = 0;
    integer signed correct_answer = 0;
    integer signed in1 = 0;
    integer signed in2 = 0;

    stack_alu #(N) alu (.clk(clk), .rst_n(rst_n), .input_data(input_data[N-1:0]), .opcode(opcode), .output_data(output_data[N-1:0]), .overflow(overflow));

    always begin
        #5 clk = ~clk;
    end

    initial begin
        #6
        input_data = 0;
        opcode = 0;
```

این test bench چندین قسمت دارد که در ادامه هر کدام را معرفی می‌کنیم.

قسمت اول که بین تمام N بیتی‌هایی که قرار است بررسی شوند قسمتی است که اعداد 8- تا 7 دوبه‌دو با هم جمع و سپس درهم ضرب می‌شوند. این قسمت دو for تو در تو دارد که اعداد مختلف را بررسی می‌کند. در هر بار انجام یکی از عملیات‌ها، دو عدد مورد نظر در پشته push می‌شوند و عملیات انجام می‌شود. در انتها تعداد کل overflow ها و تعداد جواب‌های درست مشخص می‌شود که می‌توان دید جمع آن‌ها ۲۵۶ می‌شود و این امر نشان می‌دهد که به درستی تمام عملیات‌ها تفکیک شده‌اند. برای بررسی صحت هر جواب خروجی مدار با خروجی خود عملیاتی که در test bench انجام شده است مقایسه می‌شود. در انتهای هرکدام از عکس‌هایی که تعداد جواب‌های درست و یا overflow را مشخص می‌کنند می‌توان دید که جمع این دو مقدار ۲۵۶ است و این نشان می‌دهد که تک تک اعداد در یکی از دو دسته قرار گرفته‌اند.

```
#1
for (count = -8; count < 8; count = count + 1) begin
    for (count1 = -8; count1 < 8; count1 = count1 + 1) begin
        input_data = count;
        opcode = 3'b110;
        #10
        input_data = count1;
        opcode = 3'b110;
        #10
        opcode = 3'b100;
        #10
        if (overflow == 1)
            overflow_count = overflow_count + 1;
        else if (output_data == (count + count1))
            correct_answer = correct_answer + 1;
        $display("Addition:: operand1: %d, operand2: %d, Output Data: %d, Overflow: %d", count, count1, output_data, overflow);
    end
end

#10
$display("Number of Overflows: %d. Number of Correct Answers: %d", overflow_count, correct_answer);
correct_answer = 0;
overflow_count = 0;
```

در عکس زیر در ابتدا چون *opcode* مقدار صفر دارد در یک *clock* پیام *no-op* برای ما می‌آید و سپس *ALU* شروع به محاسبه می‌کند. در هر خط می‌توانید دو ورودی *ALU*، خروجی آن، و مقدار *overflow* را ببینید. همانطور که مشاهده می‌کنید تمام این موارد برای ورودی‌های مختلف به درستی کار می‌کنند.

```
# No Operation is happening right now
# Addition:: operand1:          -8, operand2:          -8, Output Data:   0, Overflow: 1
# Addition:: operand1:          -8, operand2:          -7, Output Data:   1, Overflow: 1
# Addition:: operand1:          -8, operand2:          -6, Output Data:   2, Overflow: 1
# Addition:: operand1:          -8, operand2:          -5, Output Data:   3, Overflow: 1
# Addition:: operand1:          -8, operand2:          -4, Output Data:   4, Overflow: 1
# Addition:: operand1:          -8, operand2:          -3, Output Data:   5, Overflow: 1
# Addition:: operand1:          -8, operand2:          -2, Output Data:   6, Overflow: 1
# Addition:: operand1:          -8, operand2:          -1, Output Data:   7, Overflow: 1
# Addition:: operand1:          -8, operand2:           0, Output Data:  -8, Overflow: 0
# Addition:: operand1:          -8, operand2:           1, Output Data:  -7, Overflow: 0
# Addition:: operand1:          -8, operand2:           2, Output Data:  -6, Overflow: 0
# Addition:: operand1:          -8, operand2:           3, Output Data:  -5, Overflow: 0
# Addition:: operand1:          -8, operand2:           4, Output Data:  -4, Overflow: 0
# Addition:: operand1:          -8, operand2:           5, Output Data:  -3, Overflow: 0
# Addition:: operand1:          -8, operand2:           6, Output Data:  -2, Overflow: 0
# Addition:: operand1:          -8, operand2:           7, Output Data:  -1, Overflow: 0
# Addition:: operand1:          -7, operand2:          -8, Output Data:   1, Overflow: 1
# Addition:: operand1:          -7, operand2:          -7, Output Data:   2, Overflow: 1
# Addition:: operand1:          -7, operand2:          -6, Output Data:   3, Overflow: 1
# Addition:: operand1:          -7, operand2:          -5, Output Data:   4, Overflow: 1
# Addition:: operand1:          -7, operand2:          -4, Output Data:   5, Overflow: 1
# Addition:: operand1:          -7, operand2:          -3, Output Data:   6, Overflow: 1
# Addition:: operand1:          -7, operand2:          -2, Output Data:   7, Overflow: 1
# Addition:: operand1:          -7, operand2:          -1, Output Data:  -8, Overflow: 0
```

در عکس زیر تمام ۲۵۶ عملیات انجام شده است اما چون اعداد ما ۴ بیتی بودند بعضی از جمع‌ها overflow کرده‌اند.

```
# Addition:: operand1:           7, operand2:          -8, Output Data:  -1, Overflow: 0
# Addition:: operand1:           7, operand2:          -7, Output Data:   0, Overflow: 0
# Addition:: operand1:           7, operand2:          -6, Output Data:   1, Overflow: 0
# Addition:: operand1:           7, operand2:          -5, Output Data:   2, Overflow: 0
# Addition:: operand1:           7, operand2:          -4, Output Data:   3, Overflow: 0
# Addition:: operand1:           7, operand2:          -3, Output Data:   4, Overflow: 0
# Addition:: operand1:           7, operand2:          -2, Output Data:   5, Overflow: 0
# Addition:: operand1:           7, operand2:          -1, Output Data:   6, Overflow: 0
# Addition:: operand1:           7, operand2:           0, Output Data:   7, Overflow: 0
# Addition:: operand1:           7, operand2:           1, Output Data:  -8, Overflow: 1
# Addition:: operand1:           7, operand2:           2, Output Data:  -7, Overflow: 1
# Addition:: operand1:           7, operand2:           3, Output Data:  -6, Overflow: 1
# Addition:: operand1:           7, operand2:           4, Output Data:  -5, Overflow: 1
# Addition:: operand1:           7, operand2:           5, Output Data:  -4, Overflow: 1
# Addition:: operand1:           7, operand2:           6, Output Data:  -3, Overflow: 1
# Addition:: operand1:           7, operand2:           7, Output Data:  -2, Overflow: 1
# Number of Overflows:          64. Number of Correct Answers:         192
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(349)
#    Time: 7697 ps  Iteration: 0  Instance: /stack_based_ALU_tb
```

در دو عکس زیر مشابه دو عکس بالا را مشاهده می‌کنید با این تفاوت که ALU عملیات‌های خود را برروی اعداد ۸ بیتی انجام می‌دهد. همانطور که مشاهده خواهید کرد دیگر overflow نداریم زیرا اعداد در تعداد بیت داده شده جا می‌شوند.

```
# No Operation is happening right now
# Addition:: operand1:          -8, operand2:          -8, Output Data:  -16, Overflow: 0
# Addition:: operand1:          -8, operand2:          -7, Output Data:  -15, Overflow: 0
# Addition:: operand1:          -8, operand2:          -6, Output Data:  -14, Overflow: 0
# Addition:: operand1:          -8, operand2:          -5, Output Data:  -13, Overflow: 0
# Addition:: operand1:          -8, operand2:          -4, Output Data:  -12, Overflow: 0
# Addition:: operand1:          -8, operand2:          -3, Output Data:  -11, Overflow: 0
# Addition:: operand1:          -8, operand2:          -2, Output Data:  -10, Overflow: 0
# Addition:: operand1:          -8, operand2:          -1, Output Data:   -9, Overflow: 0
# Addition:: operand1:          -8, operand2:           0, Output Data:   -8, Overflow: 0
# Addition:: operand1:          -8, operand2:           1, Output Data:   -7, Overflow: 0
# Addition:: operand1:          -8, operand2:           2, Output Data:   -6, Overflow: 0
# Addition:: operand1:          -8, operand2:           3, Output Data:   -5, Overflow: 0
# Addition:: operand1:          -8, operand2:           4, Output Data:   -4, Overflow: 0
# Addition:: operand1:          -8, operand2:           5, Output Data:   -3, Overflow: 0
# Addition:: operand1:          -8, operand2:           6, Output Data:   -2, Overflow: 0
# Addition:: operand1:          -8, operand2:           7, Output Data:   -1, Overflow: 0
# Addition:: operand1:          -7, operand2:          -8, Output Data:  -15, Overflow: 0
# Addition:: operand1:          -7, operand2:          -7, Output Data:  -14, Overflow: 0
# Addition:: operand1:          -7, operand2:          -6, Output Data:  -13, Overflow: 0
# Addition:: operand1:           7, operand2:          -6, Output Data:    1, Overflow: 0
# Addition:: operand1:           7, operand2:          -5, Output Data:    2, Overflow: 0
# Addition:: operand1:           7, operand2:          -4, Output Data:    3, Overflow: 0
# Addition:: operand1:           7, operand2:          -3, Output Data:    4, Overflow: 0
# Addition:: operand1:           7, operand2:          -2, Output Data:    5, Overflow: 0
# Addition:: operand1:           7, operand2:          -1, Output Data:    6, Overflow: 0
# Addition:: operand1:           7, operand2:           0, Output Data:    7, Overflow: 0
# Addition:: operand1:           7, operand2:           1, Output Data:    8, Overflow: 0
# Addition:: operand1:           7, operand2:           2, Output Data:    9, Overflow: 0
# Addition:: operand1:           7, operand2:           3, Output Data:   10, Overflow: 0
# Addition:: operand1:           7, operand2:           4, Output Data:   11, Overflow: 0
# Addition:: operand1:           7, operand2:           5, Output Data:   12, Overflow: 0
# Addition:: operand1:           7, operand2:           6, Output Data:   13, Overflow: 0
# Addition:: operand1:           7, operand2:           7, Output Data:   14, Overflow: 0
# Number of Overflows:           0. Number of Correct Answers:         256
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(349)
#    Time: 7697 ps  Iteration: 0  Instance: /stack_based_ALU_tb
```

پس از این بخش، یک بار پشته و مدار خود را reset می‌کنیم تا ببینیم آیا پشته خالی می‌شود یا نه. این موضوع را با انجام یک عملیات pop بررسی می‌کنیم.

```
#10
rst_n = 0;
opcode = 3'b000;
#10
rst_n = 1;
#10
opcode = 3'b111;
#10
```

همانطور که از کد بالا پیداست پس از عملیات جمع گفته شده، یک no-op داریم و سپس مدار resest می‌شود. می‌توان دید که در عملیات pop نهایی خطا مبنی بر خالی بودن پشته به ما داده می‌شود.

```
# Addition:: operand1:           7, operand2:          -2, Output Data:   5, Overflow: 0
# Addition:: operand1:           7, operand2:          -1, Output Data:   6, Overflow: 0
# Addition:: operand1:           7, operand2:           0, Output Data:   7, Overflow: 0
# Addition:: operand1:           7, operand2:           1, Output Data:  -8, Overflow: 1
# Addition:: operand1:           7, operand2:           2, Output Data:  -7, Overflow: 1
# Addition:: operand1:           7, operand2:           3, Output Data:  -6, Overflow: 1
# Addition:: operand1:           7, operand2:           4, Output Data:  -5, Overflow: 1
# Addition:: operand1:           7, operand2:           5, Output Data:  -4, Overflow: 1
# Addition:: operand1:           7, operand2:           6, Output Data:  -3, Overflow: 1
# Addition:: operand1:           7, operand2:           7, Output Data:  -2, Overflow: 1
# Number of Overflows:          64. Number of Correct Answers:         192
# No Operation is happening right now
# There is no number to pop. The stack is empty.
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(351)
#    Time: 7737 ps  Iteration: 0  Instance: /stack_based_ALU_tb
```

در ادامه test bench که باز برای همه بیت‌ها یکسان است، برروی همین اعداد عملیات ضرب را انجام می‌دهیم. ابتدا حالتی را نشان می‌دهیم که اعداد ما 4 بیتی هستند و عملیات ضرب برروی آن‌ها اعمال می‌شود. ابتدا خود کد را می‌بینیم و سپس نتیجه آن را.

```
for (count = -8; count < 8; count = count + 1) begin //checking addition for all the possible numbers in 4 bits
    for (count1 = -8; count1 < 8; count1 = count1 + 1) begin
        input_data = count;
        opcode = 3'b110;
        #10
        input_data = count1;
        opcode = 3'b110;
        #10
        opcode = 3'b101;
        #10
        if (overflow == 1)
            overflow_count = overflow_count + 1;
        else if (output_data == (count * count1))
            correct_answer = correct_answer + 1;
        $display("Multiplication:: operand1: %d, operand2: %d, Output Data: %d, Overflow: %d", count, count1, output_data, overflow);
    end
end

#10
$display("Number of Overflows: %d. Number of Correct Answers: %d", overflow_count, correct_answer);
correct_answer = 0;
overflow_count = 0;

//##############################################################################
//     Common Test Bench For All Bits. This can applied to all bits in the question
//##############################################################################
```

در دو عکس زیر ابتدا و انتهای عملیات ضرب را برای ۴ بیت مشاهده می‌کنید. همانطور که انتظار می‌رفت، تعداد overflow ها بیشتر می‌شود زیرا در کل جمع اعداد بزرگتری نسبت به جمع می‌سازد که در محدوده قرار ندارند.

```
# Multiplication:: operand1:          -8, operand2:          -8, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -7, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -6, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -5, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -4, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -3, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -2, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:          -1, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -8, operand2:           0, Output Data:   0, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           1, Output Data:  -8, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           2, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:           3, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -8, operand2:           4, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:           5, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -8, operand2:           6, Output Data:   0, Overflow: 1
# Multiplication:: operand1:          -8, operand2:           7, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -7, operand2:          -8, Output Data:  -8, Overflow: 1
# Multiplication:: operand1:          -7, operand2:          -7, Output Data:   1, Overflow: 1
# Multiplication:: operand1:          -7, operand2:          -6, Output Data:  -6, Overflow: 1
# Multiplication:: operand1:          -7, operand2:          -5, Output Data:   3, Overflow: 1

# Multiplication:: operand1:           7, operand2:          -4, Output Data:   4, Overflow: 1
# Multiplication:: operand1:           7, operand2:          -3, Output Data:  -5, Overflow: 1
# Multiplication:: operand1:           7, operand2:          -2, Output Data:   2, Overflow: 1
# Multiplication:: operand1:           7, operand2:          -1, Output Data:  -7, Overflow: 0
# Multiplication:: operand1:           7, operand2:           0, Output Data:   0, Overflow: 0
# Multiplication:: operand1:           7, operand2:           1, Output Data:   7, Overflow: 0
# Multiplication:: operand1:           7, operand2:           2, Output Data:  -2, Overflow: 1
# Multiplication:: operand1:           7, operand2:           3, Output Data:   5, Overflow: 1
# Multiplication:: operand1:           7, operand2:           4, Output Data:  -4, Overflow: 1
# Multiplication:: operand1:           7, operand2:           5, Output Data:   3, Overflow: 1
# Multiplication:: operand1:           7, operand2:           6, Output Data:  -6, Overflow: 1
# Multiplication:: operand1:           7, operand2:           7, Output Data:   1, Overflow: 1
# Number of Overflows:         155. Number of Correct Answers:         101
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(351)
#    Time: 15427 ps  Iteration: 0  Instance: /stack_based_ALU_tb
```

حال در عکس زیر همین عملیات را برای ۸ بیت مشاهده می‌کنید. می‌توان دید که دیگر با توجه به اعداد داده شده overflow نداریم.

```
# Multiplication:: operand1:          -8, operand2:          -8, Output Data:   64, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -7, Output Data:   56, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -6, Output Data:   48, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -5, Output Data:   40, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -4, Output Data:   32, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -3, Output Data:   24, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -2, Output Data:   16, Overflow: 0
# Multiplication:: operand1:          -8, operand2:          -1, Output Data:    8, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           0, Output Data:    0, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           1, Output Data:   -8, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           2, Output Data:  -16, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           3, Output Data:  -24, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           4, Output Data:  -32, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           5, Output Data:  -40, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           6, Output Data:  -48, Overflow: 0
# Multiplication:: operand1:          -8, operand2:           7, Output Data:  -56, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -8, Output Data:   56, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -7, Output Data:   49, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -6, Output Data:   42, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -5, Output Data:   35, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -4, Output Data:   28, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -3, Output Data:   21, Overflow: 0
# Multiplication:: operand1:          -7, operand2:          -2, Output Data:   14, Overflow: 0
```

```
# Multiplication:: operand1:           7, operand2:          -2, Output Data:  -14, Overflow: 0
# Multiplication:: operand1:           7, operand2:          -1, Output Data:   -7, Overflow: 0
# Multiplication:: operand1:           7, operand2:           0, Output Data:    0, Overflow: 0
# Multiplication:: operand1:           7, operand2:           1, Output Data:    7, Overflow: 0
# Multiplication:: operand1:           7, operand2:           2, Output Data:   14, Overflow: 0
# Multiplication:: operand1:           7, operand2:           3, Output Data:   21, Overflow: 0
# Multiplication:: operand1:           7, operand2:           4, Output Data:   28, Overflow: 0
# Multiplication:: operand1:           7, operand2:           5, Output Data:   35, Overflow: 0
# Multiplication:: operand1:           7, operand2:           6, Output Data:   42, Overflow: 0
# Multiplication:: operand1:           7, operand2:           7, Output Data:   49, Overflow: 0
# Number of Overflows:          0. Number of Correct Answers:         256
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(351)
#    Time: 15427 ps  Iteration: 0  Instance: /stack_based_ALU_tb
```

در قسمت بعد به بررسی عملکرد دستور pop می‌پردازیم. می‌دانیم که در هر پشته تعداد محدودی عنصر قرار دارد که پس از تعدادی pop پشته خالی شده و پیغام مناسب باید چاپ شود. همچنین در ادامه می‌بینیم که مقادیر درستی در خروجی بعد از pop کردن قرار گرفته‌اند. در تست زیر کل پشته را خالی کرده و یک بار بیشتر نیز pop انجام می‌دهیم.

```
    for (count =  0; count < 513; count = count + 1) begin
        opcode = 3'b111;
        #10
        $display("Result of Poping: %d", output_data);
    end

    //##############################################################################################
    //                                    Pop Test Bench
    //##############################################################################################
```

در دو عکس زیر به ترتیب مقادیری که از پشته pop می‌شوند را مشاهده می‌کنید و پس از آن خطای خالی بودن پشته را مشاهده می‌کنید. همچنین می‌توان دید که مقادیر پشته به ترتیب برعکس عملیات خوانده می‌شوند که در این حالت، اعداد آخر ۷ و ۷ بودند. در انتهای عکس دوم نیز می‌توانید ببینید که پس از آمدن خطا مقدار pop شده x است که نشان‌گر نامعتبر بودن خروجی است.

```
# Result of Poping:    7
# Result of Poping:    7
# Result of Poping:    6
# Result of Poping:    7
# Result of Poping:    5
# Result of Poping:    7
# Result of Poping:    4
# Result of Poping:    7
# Result of Poping:    3
# Result of Poping:    7
# Result of Poping:    2
# Result of Poping:    7
# Result of Poping:    1
# Result of Poping:    7
# Result of Poping:    0
# Result of Poping:    7
# Result of Poping:   -1
# Result of Poping:    7
# Result of Poping:   -2
# Result of Poping:    7
# Result of Poping:   -3
# Result of Poping:    7
# Result of Poping:   -4
# Result of Poping:    7
# Result of Poping:   -5
```

```
# Result of Poping:   -8
# Result of Poping:   -5
# Result of Poping:   -8
# Result of Poping:   -6
# Result of Poping:   -8
# Result of Poping:   -7
# Result of Poping:   -8
# Result of Poping:   -8
# Result of Poping:   -8
# There is no number to pop. The stack is empty.
# Result of Poping:    X
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(351)
#    Time: 20557 ps  Iteration: 0  Instance: /stack_based_ALU_tb
# 1
```

پس از این موارد، در قسمت‌های بعد به صورت خاص به بررسی عملیات‌های جمع و ضرب برای اعداد بیشتر از ۴ بیت می‌پردازیم. در قسمت قبل تمام اعمال برروی تمام اعداد ممکن ۴ بیتی صورت گرفت. از این لحظه به بعد، تمرکز اصلی برروی اعداد ۸ بیتی، ۱۶ بیتی و ۳۲ بیتی است. قالب کلی عملیت‌ها به مانند حالت قبل و یا دو عکس زیر است و ما صرفا باید پارامتر مدار را تغییر دهیم تا اندازه اعداد کار شده مشخص شود. در ادامه دو عکس از کدهای مختلف جمع و ضرب را مشاهده خواهید کرد که به طور کلی برای تمام حالت‌های گفته شده ثابت است و صرفا برد اعداد رندوم و تعداد بیت‌های اعداد ورودی تغییر می‌کنند. نتیجه و خروجی مدار را در ادامه به صورت جزئی برای هر کدام از حالت‌های تعداد بیت‌ها مشخص می‌کنیم. دقت شود که در این قسمت‌ها ما از توابع رندم استفاده کردیم. با توجه به تعداد بیت‌های مشخص شده و range ای که اعداد می‌توانند با آن تعداد بیت داشته باشند برد اعداد رندوم ما نیز مشخص شده است. به طور کلی، دو عدد به عنوان ورودی به ما داده می‌شوند، هر دو عدد push شده و سپس عملیات برروی آن‌ها انجام می‌شود و در خروجی قرار می‌گیرد. در نهایت نتایج آن عملیات که شامل اعداد ورودی، خروجی و overflow می‌شود به کاربر نمایش داده می‌شوند.

حالت جمع:

```
#10

for (count = -8; count < 8; count = count + 1) begin
    for (count1 = -8; count1 < 8; count1 = count1 + 1) begin
        in1 = $random % 128;
        input_data = in1;
        opcode = 3'b110;
        #10
        in2 = $random % 128;
        input_data = in2;
        opcode = 3'b110;
        #10
        opcode = 3'b100;
        #10
        if (overflow == 1)
            overflow_count = overflow_count + 1;
        else if (output_data == (in1 + in2))
            correct_answer = correct_answer + 1;
        $display("Addition_8bit:: operand1: %d, operand2: %d, Output Data: %d, Overflow: %d", in1, in2, output_data, overflow);
    end
end

#10
$display("Number of Overflows: %d. Number of Correct Answers: %d", overflow_count, correct_answer);
correct_answer = 0;
overflow_count = 0;
```

حالت ضرب:

```
#10

for (count = -8; count < 8; count = count + 1) begin
    for (count1 = -8; count1 < 8; count1 = count1 + 1) begin
        in1 = $random % 20;
        input_data = in1;
        opcode = 3'b110;
        #10
        in2 = $random % 20;
        input_data = in2;
        opcode = 3'b110;
        #10
        opcode = 3'b101;
        #10
        if (overflow == 1)
            overflow_count = overflow_count + 1;
        else if (output_data == (in1 * in2))
            correct_answer = correct_answer + 1;
        else
            $display("WRONG: Multiplication_8bit:: operand1: %d, operand2: %d, Output Data: %d, Overflow: %d", in1, in2, output_data, overflow);
        $display("Multiplication_8bit:: operand1: %d, operand2: %d, Output Data: %d, Overflow: %d", in1, in2, output_data, overflow);
    end
end

#10
$display("Number of Overflows: %d. Number of Correct Answers: %d", overflow_count, correct_answer);
correct_answer = 0;
overflow_count = 0;
```

در ادامه به نتایج خروجی برای هر یک از سه حالت 8bit، 16bit و 32bit می‌پردازیم

8 bits:

خروجی جمع:

```
# Addition_8bit:: operand1:          36, operand2:        -127, Output Data:  -91, Overflow: 0
# Addition_8bit:: operand1:        -119, operand2:         -29, Output Data:  108, Overflow: 1
# Addition_8bit:: operand1:          13, operand2:          13, Output Data:   26, Overflow: 0
# Addition_8bit:: operand1:         -27, operand2:        -110, Output Data:  119, Overflow: 1
# Addition_8bit:: operand1:           1, operand2:          13, Output Data:   14, Overflow: 0
# Addition_8bit:: operand1:         118, operand2:          61, Output Data:  -77, Overflow: 1
# Addition_8bit:: operand1:         109, operand2:          12, Output Data:  121, Overflow: 0
# Addition_8bit:: operand1:         121, operand2:         -58, Output Data:   63, Overflow: 0
# Addition_8bit:: operand1:         -59, operand2:         -86, Output Data:  111, Overflow: 1
# Addition_8bit:: operand1:         101, operand2:          -9, Output Data:   92, Overflow: 0
# Addition_8bit:: operand1:        -110, operand2:          15, Output Data:  -95, Overflow: 0
# Addition_8bit:: operand1:         114, operand2:         -50, Output Data:   64, Overflow: 0
# Addition_8bit:: operand1:         -24, operand2:         -59, Output Data:  -83, Overflow: 0
# Addition_8bit:: operand1:          92, operand2:         -67, Output Data:   25, Overflow: 0
# Addition_8bit:: operand1:         -83, operand2:         -27, Output Data: -110, Overflow: 0
# Addition_8bit:: operand1:         -29, operand2:          10, Output Data:  -19, Overflow: 0
# Addition_8bit:: operand1:           0, operand2:          32, Output Data:   32, Overflow: 0
# Addition_8bit:: operand1:          42, operand2:         -99, Output Data:  -57, Overflow: 0
# Addition_8bit:: operand1:        -106, operand2:        -109, Output Data:   41, Overflow: 1
# Addition_8bit:: operand1:        -115, operand2:         -45, Output Data:   96, Overflow: 1
# Addition_8bit:: operand1:         107, operand2:         -43, Output Data:   64, Overflow: 0
# Addition_8bit:: operand1:        -126, operand2:         -82, Output Data:   48, Overflow: 1
# Addition_8bit:: operand1:          29, operand2:         -49, Output Data:  -20, Overflow: 0
# Addition_8bit:: operand1:          35, operand2:          10, Output Data:   45, Overflow: 0
```

```
# Addition_8bit:: operand1:         -23, operand2:         120, Output Data:   97, Overflow: 0
# Addition_8bit:: operand1:        -122, operand2:         -17, Output Data:  117, Overflow: 1
# Addition_8bit:: operand1:          97, operand2:        -107, Output Data:  -10, Overflow: 0
# Addition_8bit:: operand1:         -18, operand2:          96, Output Data:   78, Overflow: 0
# Addition_8bit:: operand1:         -27, operand2:         -88, Output Data: -115, Overflow: 0
# Addition_8bit:: operand1:         100, operand2:          41, Output Data: -115, Overflow: 1
# Addition_8bit:: operand1:         -31, operand2:          -9, Output Data:  -40, Overflow: 0
# Addition_8bit:: operand1:         110, operand2:           1, Output Data:  111, Overflow: 0
# Addition_8bit:: operand1:          85, operand2:         104, Output Data:  -67, Overflow: 1
# Addition_8bit:: operand1:         112, operand2:         -86, Output Data:   26, Overflow: 0
# Addition_8bit:: operand1:          15, operand2:         -15, Output Data:    0, Overflow: 0
# Addition_8bit:: operand1:          -3, operand2:          14, Output Data:   11, Overflow: 0
# Addition_8bit:: operand1:         -30, operand2:          60, Output Data:   30, Overflow: 0
# Addition_8bit:: operand1:         -87, operand2:         111, Output Data:   24, Overflow: 0
# Addition_8bit:: operand1:         -37, operand2:         -81, Output Data: -118, Overflow: 0
# Number of Overflows:         54. Number of Correct Answers:         202
```

خروجی ضرب:

```
# Addition_8bit:: operand1:         -37, operand2:         -81, Output Data: -118, Overflow: 0
# Number of Overflows:          54. Number of Correct Answers:          202
# Multiplication_8bit:: operand1:          18, operand2:           5, Output Data:   90, Overflow: 0
# Multiplication_8bit:: operand1:          15, operand2:          11, Output Data:  -91, Overflow: 1
# Multiplication_8bit:: operand1:           1, operand2:          13, Output Data:   13, Overflow: 0
# Multiplication_8bit:: operand1:          -9, operand2:         -11, Output Data:   99, Overflow: 0
# Multiplication_8bit:: operand1:         -10, operand2:          -7, Output Data:   70, Overflow: 0
# Multiplication_8bit:: operand1:          18, operand2:         -12, Output Data:   40, Overflow: 1
# Multiplication_8bit:: operand1:          18, operand2:         -15, Output Data:  -14, Overflow: 1
# Multiplication_8bit:: operand1:           1, operand2:           1, Output Data:    1, Overflow: 0
# Multiplication_8bit:: operand1:           9, operand2:         -19, Output Data:   85, Overflow: 1
# Multiplication_8bit:: operand1:          -3, operand2:          19, Output Data:  -57, Overflow: 0
# Multiplication_8bit:: operand1:          -5, operand2:         -19, Output Data:   95, Overflow: 0
# Multiplication_8bit:: operand1:          16, operand2:         -14, Output Data:   32, Overflow: 1
# Multiplication_8bit:: operand1:          -6, operand2:           1, Output Data:   -6, Overflow: 0
# Multiplication_8bit:: operand1:          -2, operand2:          12, Output Data:  -24, Overflow: 0
# Multiplication_8bit:: operand1:         -16, operand2:          -1, Output Data:   16, Overflow: 0
# Multiplication_8bit:: operand1:          -3, operand2:          -8, Output Data:   24, Overflow: 0
# Multiplication_8bit:: operand1:          14, operand2:          -4, Output Data:  -56, Overflow: 0
# Multiplication_8bit:: operand1:          11, operand2:           1, Output Data:   11, Overflow: 0
# Multiplication_8bit:: operand1:           0, operand2:          -9, Output Data:    0, Overflow: 0
# Multiplication_8bit:: operand1:          -2, operand2:         -13, Output Data:   26, Overflow: 0
# Multiplication_8bit:: operand1:           4, operand2:          -2, Output Data:   -8, Overflow: 0
# Multiplication_8bit:: operand1:          16, operand2:         -19, Output Data:  -48, Overflow: 1
# Multiplication_8bit:: operand1:           5, operand2:          -9, Output Data:  -45, Overflow: 0
# Multiplication_8bit:: operand1:          16, operand2:          -3, Output Data:  -48, Overflow: 0
# Multiplication_8bit:: operand1:          12, operand2:         -19, Output Data:   28, Overflow: 1
# Multiplication_8bit:: operand1:          14, operand2:         -11, Output Data:  102, Overflow: 1

# Multiplication_8bit:: operand1:          -3, operand2:           4, Output Data:  -12, Overflow: 0
# Multiplication_8bit:: operand1:         -12, operand2:         -11, Output Data: -124, Overflow: 1
# Multiplication_8bit:: operand1:           2, operand2:          -6, Output Data:  -12, Overflow: 0
# Multiplication_8bit:: operand1:         -13, operand2:          17, Output Data:   35, Overflow: 1
# Multiplication_8bit:: operand1:          16, operand2:           0, Output Data:    0, Overflow: 0
# Multiplication_8bit:: operand1:           5, operand2:          16, Output Data:   80, Overflow: 0
# Multiplication_8bit:: operand1:           2, operand2:          -8, Output Data:  -16, Overflow: 0
# Multiplication_8bit:: operand1:           1, operand2:          15, Output Data:   15, Overflow: 0
# Multiplication_8bit:: operand1:          13, operand2:         -19, Output Data:    9, Overflow: 1
# Multiplication_8bit:: operand1:           5, operand2:         -13, Output Data:  -65, Overflow: 0
# Multiplication_8bit:: operand1:          14, operand2:          19, Output Data:   10, Overflow: 1
# Multiplication_8bit:: operand1:          18, operand2:           3, Output Data:   54, Overflow: 0
# Multiplication_8bit:: operand1:          -6, operand2:           5, Output Data:  -30, Overflow: 0
# Multiplication_8bit:: operand1:           0, operand2:           7, Output Data:    0, Overflow: 0
# Multiplication_8bit:: operand1:         -16, operand2:         -17, Output Data:   16, Overflow: 1
# Number of Overflows:          67. Number of Correct Answers:          189
```

:16 bits

خروجی جمع:

```
# Addition_16bit:: operand1:      -32670, operand2:       15337, Output Data: -17333, Overflow: 0
# Addition_16bit:: operand1:         814, operand2:      -28019, Output Data: -27205, Overflow: 0
# Addition_16bit:: operand1:        1990, operand2:      -19286, Output Data: -17296, Overflow: 0
# Addition_16bit:: operand1:       -3936, operand2:       20279, Output Data:  16343, Overflow: 0
# Addition_16bit:: operand1:      -20752, operand2:      -22296, Output Data:  22488, Overflow: 1
# Addition_16bit:: operand1:      -25395, operand2:        2513, Output Data: -22882, Overflow: 0
# Addition_16bit:: operand1:       25531, operand2:      -27937, Output Data:  -2406, Overflow: 0
# Addition_16bit:: operand1:       21885, operand2:        8536, Output Data:  30421, Overflow: 0
# Addition_16bit:: operand1:      -12129, operand2:       12183, Output Data:     54, Overflow: 0
# Addition_16bit:: operand1:      -24058, operand2:      -20951, Output Data:  20527, Overflow: 1
# Addition_16bit:: operand1:       25540, operand2:       -2345, Output Data:  23195, Overflow: 0
# Addition_16bit:: operand1:      -11171, operand2:        4941, Output Data:  -6230, Overflow: 0
# Addition_16bit:: operand1:       18291, operand2:      -21889, Output Data:  -3598, Overflow: 0
# Addition_16bit:: operand1:       -2994, operand2:       26041, Output Data:  23047, Overflow: 0
# Addition_16bit:: operand1:       -6944, operand2:      -15815, Output Data: -22759, Overflow: 0
# Addition_16bit:: operand1:       20353, operand2:         817, Output Data:  21170, Overflow: 0
# Addition_16bit:: operand1:       28559, operand2:       32254, Output Data:  -4723, Overflow: 1
# Addition_16bit:: operand1:       -4003, operand2:      -30705, Output Data:  30828, Overflow: 1
# Addition_16bit:: operand1:       19251, operand2:       21865, Output Data: -24420, Overflow: 1
# Addition_16bit:: operand1:        9558, operand2:      -11622, Output Data:  -2064, Overflow: 0
# Addition_16bit:: operand1:       10588, operand2:        1865, Output Data:  12453, Overflow: 0
# Addition_16bit:: operand1:       30675, operand2:       -1478, Output Data:  29197, Overflow: 0
# Addition_16bit:: operand1:       30168, operand2:      -29064, Output Data:   1104, Overflow: 0
# Addition 16bit:: operand1:       13625, operand2:       -8155, Output Data:   5470, Overflow: 0

# Addition_16bit:: operand1:        3058, operand2:      -22898, Output Data: -19840, Overflow: 0
# Addition_16bit:: operand1:      -30574, operand2:      -11529, Output Data:  23433, Overflow: 1
# Addition_16bit:: operand1:      -25874, operand2:      -12731, Output Data:  26931, Overflow: 1
# Addition_16bit:: operand1:       25086, operand2:      -12179, Output Data:  12907, Overflow: 0
# Addition_16bit:: operand1:       -7001, operand2:      -28668, Output Data:  29867, Overflow: 1
# Addition_16bit:: operand1:        -990, operand2:      -25511, Output Data: -26501, Overflow: 0
# Addition_16bit:: operand1:      -20762, operand2:      -15219, Output Data:  29555, Overflow: 1
# Addition_16bit:: operand1:        2309, operand2:       28987, Output Data:  31296, Overflow: 0
# Addition_16bit:: operand1:        2511, operand2:      -30644, Output Data: -28133, Overflow: 0
# Addition_16bit:: operand1:       29643, operand2:      -13145, Output Data:  16498, Overflow: 0
# Addition_16bit:: operand1:       15640, operand2:      -18205, Output Data:  -2565, Overflow: 0
# Number of Overflows:         72. Number of Correct Answers:        184
```

خروجی ضرب:

```
# Multiplication_16bit:: operand1:          64, operand2:         -53, Output Data:  -3392, Overflow: 0
# Multiplication_16bit:: operand1:         299, operand2:          74, Output Data:  22126, Overflow: 0
# Multiplication_16bit:: operand1:         117, operand2:        -129, Output Data: -15093, Overflow: 0
# Multiplication_16bit:: operand1:          28, operand2:        -340, Output Data:  -9520, Overflow: 0
# Multiplication_16bit:: operand1:         -77, operand2:         334, Output Data: -25718, Overflow: 0
# Multiplication_16bit:: operand1:        -168, operand2:         212, Output Data:  29920, Overflow: 1
# Multiplication_16bit:: operand1:        -292, operand2:         133, Output Data:  26700, Overflow: 1
# Multiplication_16bit:: operand1:          61, operand2:         -37, Output Data:  -2257, Overflow: 0
# Multiplication_16bit:: operand1:         320, operand2:        -114, Output Data:  29056, Overflow: 1
# Multiplication_16bit:: operand1:        -341, operand2:         -24, Output Data:   8184, Overflow: 0
# Multiplication_16bit:: operand1:         284, operand2:         -34, Output Data:  -9656, Overflow: 0
# Multiplication_16bit:: operand1:          47, operand2:          44, Output Data:   2068, Overflow: 0
# Multiplication_16bit:: operand1:        -132, operand2:         -43, Output Data:   5676, Overflow: 0
# Multiplication_16bit:: operand1:        -242, operand2:          94, Output Data: -22748, Overflow: 0
# Multiplication_16bit:: operand1:         -46, operand2:        -243, Output Data:  11178, Overflow: 0
# Multiplication_16bit:: operand1:          36, operand2:        -131, Output Data:  -4716, Overflow: 0
# Multiplication_16bit:: operand1:         -37, operand2:         311, Output Data: -11507, Overflow: 0
# Multiplication_16bit:: operand1:        -247, operand2:         330, Output Data: -15974, Overflow: 1
# Multiplication_16bit:: operand1:         134, operand2:         147, Output Data:  19698, Overflow: 0
# Multiplication_16bit:: operand1:        -165, operand2:         298, Output Data:  16366, Overflow: 1
# Multiplication_16bit:: operand1:         253, operand2:         200, Output Data: -14936, Overflow: 1
# Multiplication_16bit:: operand1:         324, operand2:          45, Output Data:  14580, Overflow: 0
# Multiplication_16bit:: operand1:        -266, operand2:         165, Output Data:  21646, Overflow: 1
# Multiplication_16bit:: operand1:         193, operand2:          35, Output Data:   6755, Overflow: 0
# Multiplication_16bit:: operand1:        -269, operand2:          73, Output Data: -19637, Overflow: 0
# Multiplication_16bit:: operand1:        -135, operand2:         -68, Output Data:   9180, Overflow: 0
# Multiplication_16bit:: operand1:        -121, operand2:        -120, Output Data:  14520, Overflow: 0
# Multiplication_16bit:: operand1:        -161, operand2:         150, Output Data: -24150, Overflow: 0
# Multiplication_16bit:: operand1:        -123, operand2:         260, Output Data: -31980, Overflow: 0

# Multiplication_16bit:: operand1:          74, operand2:         -10, Output Data:   -740, Overflow: 0
# Multiplication_16bit:: operand1:        -314, operand2:         308, Output Data: -31176, Overflow: 1
# Multiplication_16bit:: operand1:         199, operand2:        -280, Output Data:   9816, Overflow: 1
# Multiplication_16bit:: operand1:         274, operand2:        -279, Output Data: -10910, Overflow: 1
# Multiplication_16bit:: operand1:         -34, operand2:        -261, Output Data:   8874, Overflow: 0
# Multiplication_16bit:: operand1:        -134, operand2:        -168, Output Data:  22512, Overflow: 0
# Multiplication_16bit:: operand1:         232, operand2:        -316, Output Data:  -7776, Overflow: 1
# Multiplication_16bit:: operand1:         -85, operand2:         194, Output Data: -16490, Overflow: 0
# Multiplication_16bit:: operand1:         -21, operand2:         249, Output Data:  -5229, Overflow: 0
# Multiplication_16bit:: operand1:          72, operand2:         205, Output Data:  14760, Overflow: 0
# Multiplication_16bit:: operand1:         113, operand2:        -281, Output Data: -31753, Overflow: 0
# Number of Overflows:          92. Number of Correct Answers:          164
```

:32bits

خروجی جمع:

```
# Addition_32bit:: operand1:   887803241, operand2:   374085420, Output Data:  1261888661, Overflow: 0
# Addition_32bit:: operand1:  -859165031, operand2:  1861210077, Output Data:  1002045046, Overflow: 0
# Addition_32bit:: operand1:  -798715232, operand2: -1927688166, Output Data:  1568563898, Overflow: 1
# Addition_32bit:: operand1:   473521464, operand2:  -564507972, Output Data:   -90986508, Overflow: 0
# Addition_32bit:: operand1:  -326337831, operand2:   130123023, Output Data:  -196214808, Overflow: 0
# Addition_32bit:: operand1: -1935390695, operand2:  1053574525, Output Data:  -881816170, Overflow: 0
# Addition_32bit:: operand1:  -319656231, operand2:  2071932918, Output Data:  1752276687, Overflow: 0
# Addition_32bit:: operand1: -2032156659, operand2:   487136570, Output Data: -1545020089, Overflow: 0
# Addition_32bit:: operand1:  -776790877, operand2:   537168704, Output Data:  -239622173, Overflow: 0
# Addition_32bit:: operand1:  1724600269, operand2:  -248517918, Output Data:  1476082351, Overflow: 0
# Addition_32bit:: operand1:  2082729976, operand2:   691344210, Output Data: -1520893110, Overflow: 1
# Addition_32bit:: operand1: -1025687419, operand2: -2049470965, Output Data:  1219808912, Overflow: 1
# Addition_32bit:: operand1: -1405152680, operand2:   849479525, Output Data:  -555673155, Overflow: 0
# Addition_32bit:: operand1:  -898719596, operand2:  1458217389, Output Data:   559497793, Overflow: 0
# Addition_32bit:: operand1:   584020293, operand2:  -658088271, Output Data:   -74067978, Overflow: 0
# Addition_32bit:: operand1:   113720077, operand2: -1002209656, Output Data:  -888489579, Overflow: 0
# Addition_32bit:: operand1:   341621544, operand2: -1140664200, Output Data:  -799042656, Overflow: 0
# Addition_32bit:: operand1: -1964401643, operand2: -1274526616, Output Data:  1056039037, Overflow: 1
# Addition_32bit:: operand1:  -656976207, operand2:  -379981614, Output Data: -1036957821, Overflow: 0
```

```
# Addition_32bit:: operand1: -1023486331, operand2:  -342096681, Output Data: -1365583012, Overflow: 0
# Addition_32bit:: operand1: -1708083660, operand2: -1503436212, Output Data:  1083447424, Overflow: 1
# Addition_32bit:: operand1: -1451125677, operand2:  -458658103, Output Data: -1909783780, Overflow: 0
# Addition_32bit:: operand1:   606373704, operand2:  1372736419, Output Data:  1979110123, Overflow: 0
# Addition_32bit:: operand1:  2084875768, operand2: -1434473388, Output Data:   650402380, Overflow: 0
# Addition_32bit:: operand1: -1353772962, operand2:  1947815400, Output Data:   594042438, Overflow: 0
# Addition_32bit:: operand1: -1962440682, operand2:  1038367099, Output Data:  -924073583, Overflow: 0
# Addition_32bit:: operand1:  1562418106, operand2:  -738115416, Output Data:   824302690, Overflow: 0
# Addition_32bit:: operand1:   332289319, operand2: -1435874220, Output Data: -1103584901, Overflow: 0
# Addition_32bit:: operand1:   680277329, operand2:  -909720429, Output Data:  -229443100, Overflow: 0
# Addition_32bit:: operand1:  1863188446, operand2: -1582372797, Output Data:   280815649, Overflow: 0
# Addition_32bit:: operand1:  1091807618, operand2: -1011469177, Output Data:    80338441, Overflow: 0
# Number of Overflows:         60. Number of Correct Answers:        196
```

خروجی ضرب:

```
# Multiplication_32bit:: operand1:       51089, operand2:       -7914, Output Data:  -404318346, Overflow: 0
# Multiplication_32bit:: operand1:       70667, operand2:       77722, Output Data:  1197413278, Overflow: 1
# Multiplication_32bit:: operand1:       67231, operand2:      -15549, Output Data: -1045374819, Overflow: 0
# Multiplication_32bit:: operand1:       79084, operand2:       65821, Output Data:   910420668, Overflow: 1
# Multiplication_32bit:: operand1:      -18392, operand2:       -7404, Output Data:   136174368, Overflow: 0
# Multiplication_32bit:: operand1:       77779, operand2:      -35173, Output Data:  1559246529, Overflow: 1
# Multiplication_32bit:: operand1:       15572, operand2:      -36236, Output Data:  -564266992, Overflow: 0
# Multiplication_32bit:: operand1:      -40635, operand2:       28406, Output Data: -1154277810, Overflow: 0
# Multiplication_32bit:: operand1:       20816, operand2:      -64068, Output Data: -1333639488, Overflow: 0
# Multiplication_32bit:: operand1:       79593, operand2:       20931, Output Data:  1665961083, Overflow: 0
# Multiplication_32bit:: operand1:        6414, operand2:       52868, Output Data:   339095352, Overflow: 0
# Multiplication_32bit:: operand1:       80293, operand2:       13329, Output Data:  1070225397, Overflow: 0
# Multiplication_32bit:: operand1:      -51307, operand2:       32153, Output Data: -1649673971, Overflow: 0
# Multiplication_32bit:: operand1:      -75956, operand2:       49121, Output Data:   563932620, Overflow: 1
# Multiplication_32bit:: operand1:       15968, operand2:       34200, Output Data:   546105600, Overflow: 0
# Multiplication_32bit:: operand1:       22784, operand2:        9072, Output Data:   206696448, Overflow: 0
# Multiplication_32bit:: operand1:       25490, operand2:       11231, Output Data:   286278190, Overflow: 0
# Multiplication_32bit:: operand1:      -54717, operand2:      -38720, Output Data:  2118642240, Overflow: 0
# Multiplication_32bit:: operand1:      -84235, operand2:       63273, Output Data: -1034833859, Overflow: 1
# Multiplication_32bit:: operand1:      -57201, operand2:       17679, Output Data: -1011256479, Overflow: 0
# Multiplication_32bit:: operand1:      -81563, operand2:       22986, Output Data: -1874807118, Overflow: 0

# Multiplication_32bit:: operand1:       15313, operand2:       61777, Output Data:   945991201, Overflow: 0
# Multiplication_32bit:: operand1:      -32542, operand2:      -78948, Output Data: -1725841480, Overflow: 1
# Multiplication_32bit:: operand1:      -71630, operand2:      -42272, Output Data: -1267023936, Overflow: 1
# Multiplication_32bit:: operand1:       87032, operand2:      -86081, Output Data:  1098133000, Overflow: 1
# Multiplication_32bit:: operand1:       89806, operand2:       28549, Output Data: -1731095802, Overflow: 1
# Multiplication_32bit:: operand1:      -55495, operand2:      -47774, Output Data: -1643749166, Overflow: 1
# Multiplication_32bit:: operand1:       -8170, operand2:       84800, Output Data:  -692816000, Overflow: 0
# Multiplication_32bit:: operand1:       90785, operand2:       39039, Output Data:  -750811681, Overflow: 1
# Multiplication_32bit:: operand1:       -8690, operand2:      -45862, Output Data:   398540780, Overflow: 0
# Multiplication_32bit:: operand1:       77667, operand2:      -88033, Output Data:  1752675581, Overflow: 1
# Multiplication_32bit:: operand1:      -43640, operand2:       82258, Output Data:   705228176, Overflow: 1
# Multiplication_32bit:: operand1:      -43442, operand2:       32755, Output Data: -1422942710, Overflow: 0
# Multiplication_32bit:: operand1:      -47548, operand2:       88742, Output Data:    75462680, Overflow: 1
# Multiplication_32bit:: operand1:       11422, operand2:       66092, Output Data:   754902824, Overflow: 0
# Multiplication_32bit:: operand1:       35499, operand2:       88880, Output Data: -1139816176, Overflow: 1
# Multiplication_32bit:: operand1:       89231, operand2:       36991, Output Data:  -994223375, Overflow: 1
# Multiplication_32bit:: operand1:       54957, operand2:        4401, Output Data:   241865757, Overflow: 0
# Multiplication_32bit:: operand1:      -77999, operand2:      -21239, Output Data:  1656620761, Overflow: 0
# Multiplication_32bit:: operand1:       23084, operand2:       43804, Output Data:  1011171536, Overflow: 0
# Number of Overflows:        112. Number of Correct Answers:        144
```

در قسمت بعد به سراغ انواع خطاهایی که ممکن است برای کاربر رخ دهد می‌رویم.

در ابتدا بررسی می‌کنیم که اگر بیشتر از ظرفیت پشته، بخواهیم به آن عضو اضافه بکنیم چه اتفاقی می‌افتد. بر اساس قسمت قبلی testbench می‌دانیم که در حال حاضر پشته پر است و با اضافه کردن صرفا یک عضو جدید، پیام پر بودن پشته را خواهیم گرفت. در ابتدا قطعه کدی را که این موضوع را بررسی می‌کند را می‌بینیم و سپس خروجی مدار. همانطور که در عکس خروجی پیداست، خروجی مدار x و یا در اینجا invalid شده است.

```
    #10

    input_data = count;
    opcode = 3'b110;
    #10;
    $display("Output: %d", output_data);
    opcode = 3'b000;

    //##############################################################################################
    //                                    Full Stack Test Bench
    //##############################################################################################
```

```
# Multiplication_32bit:: operand1:       89231, operand2:       36991, Output Data:  -994223375, Overflow: 1
# Multiplication_32bit:: operand1:       54957, operand2:        4401, Output Data:   241865757, Overflow: 0
# Multiplication_32bit:: operand1:      -77999, operand2:      -21239, Output Data:  1656620761, Overflow: 0
# Multiplication_32bit:: operand1:       23084, operand2:       43804, Output Data:  1011171536, Overflow: 0
# Number of Overflows:         112. Number of Correct Answers:         144
# There is not enough space for another push
# Output:          X
```

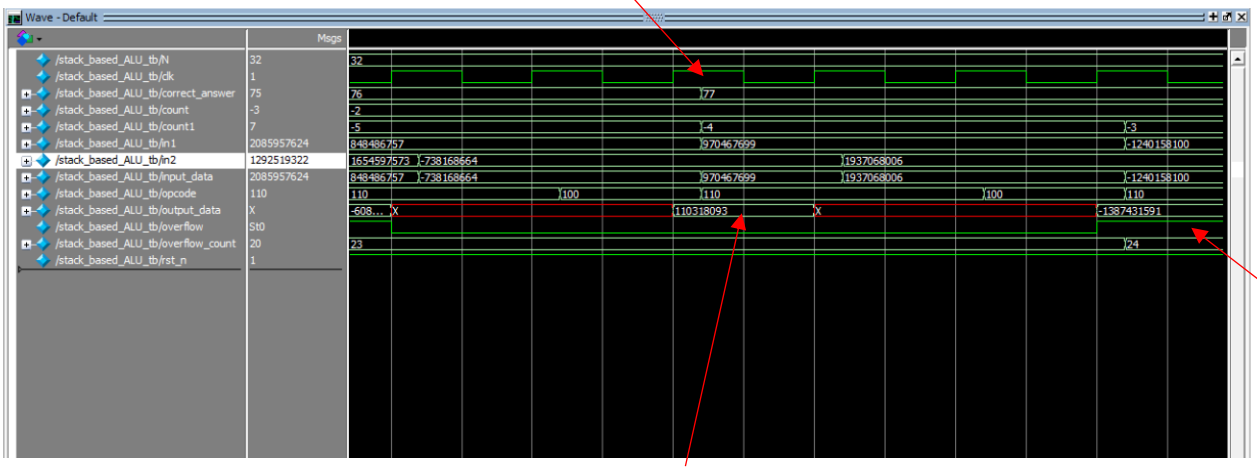
در نهایت می‌خواهیم خطاهایی را که هنگام انجام عملیات‌های جمع و ضرب رخ می‌دهند را ببینیم. دقت شود که این خطاها زمانی ظاهر می‌شوند که ما قصد داریم عملیات جمع و یا ضرب را در حالتی که کمتر از دو عضو در پشته وجود دارد انجام دهیم. در قطعه کد زیر می‌توان دید که پس از حالی کردن پشته، ابتدا یک بار عملیات جمع و ضرب را انجام می‌دهیم و خطای مناسب را دریافت می‌کنیم و همچنین بعد از اضافه کردن یک عضو دوباره امتحان می‌کنیم و دوباره خطای مناسب را دریافت می‌کنیم.

```
340         opcode = 3'b100; //zero elemments
341         #10
342         $display("Output: %d", output_data);
343         opcode = 3'b101; //zero elemments
344         #10
345         $display("Output: %d", output_data);
346         input_data = 1234;
347         opcode = 3'b110;
348         #10
349         opcode = 3'b100; //one elemment
350         #10
351         $display("Output: %d", output_data);
352         opcode = 3'b101; //one elemment
353         #10;
354         $display("Output: %d", output_data);
355
356         //##############################################################################################
357         //                                    Operation Error Test Bench
358         //##############################################################################################
359
360         $finish;
361     end
```

در عکس زیر که خروجی قطعه کد بالاست می‌توان دید که در هر حالت عملیات جمع و یا ضرب انجام نشده است و خروجی x است که نمایانگر invalid بودن عملیات است.

```
# Result of Poping:   473521464
# Result of Poping: -1927688166
# Result of Poping:  -798715232
# Result of Poping:  1861210077
# Result of Poping:  -859165031
# Result of Poping:   374085420
# Result of Poping:   887803241
# You do not have enough numbers to perform addition
# Output:           X
# You do not have enough numbers to perform multiplication
# Output:           X
# You do not have enough numbers to perform addition
# Output:           X
# You do not have enough numbers to perform multiplication
# Output:           X
# ** Note: $finish    : C:/Users/Hosein/Desktop/Question1_Final/tb.v(360)
#    Time: 97557 ps  Iteration: 0  Instance: /stack_based_ALU_tb
```

در آخر چند عکس از waveform را نیز می‌بینیم که درآن به درستی سیگنال set overflow شده است و مقادیر خروجی به درستی حساب شده‌اند. همچنین می‌توان نوع عملیات را نیز دید. مقدار خروجی در زمان‌هایی که در حال push کردن هستیم x می‌شود که این موضوع کاملا منطقی است.



چالش‌ها:

در انتها به چالش‌هایی که در مسیر طراحی این مدار برخوردیم می‌پردازیم:

- تشخیص آن که آیا overflow رخ داده است یا نه یکی از چالش‌های این مسیر بود. بررسی این موضوع در هرکدام از عملیات‌ها به طور کاملا متفاوتی انجام می‌شود و محدود کردن جواب‌ها به تعداد بیت خاصی باعث می‌شد که نتوان به راحتی بیت یا بیت‌هایی که می‌توان از آن‌ها وقوع overflow را تشخیص داد جدا و بررسی کرد.
- تنظیم پارمترهای رندوم برای اعداد با بیت‌های مختلف مقداری باعث چالش شده بود. ما می‌بایست مقدار درستی را به عنوان برد رندوم بودن اعداد انتخابی مشخص می‌کردیم تا بتوانیم هم تعداد خوبی از اعداد را بررسی کنیم و هم این که جواب خود را به یک سمت bias نکنیم.
- کار با stack pointer و مشخص کردن سیاست نمایش آن از دیگر چالش‌هایی بود که در شاید در حالت عادی خیلی خود را نشان نمی‌داد اما وقتی به حالات مرزی می‌رسیدیم ممکن بود به دلیل کم و یا زیاد کردن اشتباه stack pointer به مشکل بخوریم.