دانشگاه صنعتی شریف

دانشکده مهندسی کامپیوتر

پروژه جبرانی طراحی سیستم‌های دیجیتال

پوریا غفوری

۴۰۱۱۰۶۲۷۷

استاد: جناب آقای امین فصحتی

نیم سال دوم ۱۴۰۳-۱۴۰۲

هدف پروژه:

در این پروژه قصد داریم که یک STACK BASED ALU را با کمک Verilog طراحی کرده که می‌تواند اعداد را در خود پوش کند یا آن‌ها را پاپ کند و همچنین می‌تواند حاصل ضرب و یا حاصل جمع دو عدد بالای استک را خروجی دهد.

در بخش دوم این پروژه باید با کمک Verilog یک ماژول دیگر طراحی می‌کردیم که عبارات ریاضی را ورودی بگیرد و آن‌ها را به صورت پسوندی در بیاورد و سپس با کمک ماژول STACK BASED ALU مقدار آن‌ها را حساب کند.

نحوه پیاده‌سازی:

ابتدا ماژول STACK BASED ALU را تشریح می‌کنیم و نحوه کارکرد آن را توضیح می‌دهیم. قابل ذکر است که کد Verilog آن در پوشه code و با نام STACK_BASED_ALU.v قرار دارد.

```
module STACK_BASED_ALU #(parameter N)(
    input clk,
    input signed [N-1:0] input_data,
    input [2:0] opcode,
    output reg signed [N-1:0] output_data,
    output reg overflow
);
    reg signed [N-1:0] stack [0:511];
    reg [9:0] sp;

    initial begin
        sp = 0;
    end
```

در ابتدای کار آن را با پارامتر ورودی N (که سایز اعداد ما می‌باشد) تعریف می‌کنیم و ورودی و خروجی‌های مورد نیاز آن را قرار می‌دهیم که این ورودی و خروجی‌ها همان‌هایی هستند که در صورت سوال بیان شده‌اند و من فقط clk را به آن اضافه کرده‌ام که اعمال ماژول ما در لبه بالارونده کلاک انجام گیرد.

در ادامه متغیرهای مورد نیاز stack و sp را تعریف کردم که stack همان حافظه ما می‌باشد که عرض آن به اندازه بیت‌های اعداد ما می‌باشد و عمق آن در صورت سوال بیان نشده بود و من آن را ۵۱۲ قرار دادم به این معنا که Stack ما می‌تواند ۵۱۲ عدد N بیتی را در خود جای دهد. متغیر sp هم همان Stack pointer ما می‌باشد.

در ادامه در بلاک initial مقدار sp را برابر صفر قرار دادم تا هر وقت از ماژول خود یک instance گرفتیم آن stack pointer به خانه اول اشاره کند.

```
always @(posedge clk) begin
    case (opcode)
        3'b100: begin
            if (sp > 1) begin
                output_data = stack[sp-1] + stack[sp-2];
                if (((stack[sp-1] > 0) && (stack[sp-2] > 0) && (output_data <= 0)) ||
                    ((stack[sp-1] < 0) && (stack[sp-2] < 0) && (output_data >= 0))) begin
                    overflow = 1;
                end
                else begin
                    overflow = 0;
                end
            end
            else begin
                overflow = 1'bx;
                output_data = {N{1'bx}};
            end
        end
```

در عکس بالا داخل بلاک always را نشان می‌دهیم که با لبه بالارونده کلاک کار می‌کند و هر بار که کلاک می‌خورد opcode ورودی را بررسی می‌کند و بر اساس آن عمل می‌کند.

در این تصویر اگر opcode برابر ۱۰۰ بود به این معنا است که عملیات جمع باید انجام گیرد پس ما بررسی می‌کنیم که آیا اعداد موجود در استک بیشتر از یکی می‌باشد یا خیر (چون می‌خواهیم دو عدد بالای استک را با یکدیگر جمع کنیم) و اگر این تعداد بیشتر از یک بود دو عنصر بالای استک را با هم جمع کرده و در output_data می‌ریزیم و سپس برای بررسی overflow چک می‌کنیم که اگر هر دو عنصر جمع ما هم علامت بودند اما جواب جمع علامتی متفاوت با آن‌ها داشت به این معنا است که overflow رخ داده و بیت آن را یک می‌کنیم و در غیر این صورت عملیات جمع بدون سر ریز انجام شده و مقدار بیت آن را صفر قرار می‌دهیم. همچنین اگر تعداد اعداد موجود در استک کمتر از دو بود مقدار X را در output_data و overflow می‌ریزیم تا مقداری اشتباه را نشان ندهند.

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

در کیس بعدی بررسی می‌کنیم که آیا opcode برابر ۱۰۱ می‌باشد یا خیر که اگر برابر این مقدار بود به این معنا است که عملیات ضرب باید انجام شود. در ابتدا مشابه جمع بررسی می‌کنیم که آیا تعداد اعداد موجود در استک بیشتر از یکی است یا خیر که اگر بیشتر نبود مقدار X را در خروجی‌ها می‌ریزیم و اگر بیشتر از یک عدد در استک بود عملیات ضرب را انجام داده و خروجی آن را در output_data می‌ریزیم. برای بررسی overflow هم ابتدا بررسی می‌کنیم که اعدادی که عملیات ضرب روی آن‌ها انجام می‌شود برابر صفر نباشند (چون اگر صفر باشند جواب ضرب هم صفر شده و نمی‌تواند overflow رخ دهد) حال اگر صفر نبودند جواب ضرب را تقسیم بر عدد اول می‌کنیم اگر جواب تقسیم مخالف عدد دوم شد به این معنا است که overflow رخ داده و مقدار اشتباهی در output_data می‌باشد و بیت overflow را یک می‌کنیم و در غیر این صورت بیت overflow را برابر صفر قرار می‌دهیم.

```
3'b110: begin
    overflow = 1'bx;
    output_data = {N{1'bx}};
    if (sp < 512) begin
        stack[sp] = input_data;
        sp = sp + 1;
    end
end
```

در این کیس بررسی می‌کنیم که opcode برابر ۱۱۰ می‌باشد یا خیر که اگر برابر بود عملیات push باید صورت گیرد در ابتدای کار خروجی‌های ماژول را برابر X قرار می‌دهیم چون خروجی نداریم و بعد از آن بررسی می‌کنیم که آیا استک خالی است یا خیر و اگر استک پر بود کاری انجام نمی‌دهیم، اما اگر استک خالی بود عدد ورودی را در استک قرار داده و sp را یکی بالاتر می‌بریم.

در این کیس بررسی می‌کنیم که opcode برابر ۱۱۱ می‌باشد یا خیر که اگر برابر بود عملیات pop باید صورت گیرد. در ابتدا بیت overflow را X می‌کنیم چون معنایی ندارد. در ادامه بررسی می‌کنیم که آیا استک خالی است یا خیر. اگر خالی بود که X را در خروجی می‌ریزیم اما اگر پر بود مقدار آن را در خروجی ریخته و آن خانه از استک را X می‌کنیم و sp را یکی پایین‌تر می‌بریم.

```
3'b111: begin
    overflow = 1'bx;
    if (sp > 0) begin
        sp = sp - 1;
        output_data = stack[sp];
        stack[sp] = {N{1'bx}};
    end
    else begin
        output_data = {N{1'bx}};
    end
end
```

در آخر هم بررسی می‌کنیم که اگر opcode برابر هیچکدام از opcodeهای مشخص شده نبود کاری انجام نمی‌دهیم و خروجی‌ها را X می‌کنیم.

```
default: begin
    overflow = 1'bx;
    output_data = {N{1'bx}};
end
```

در اینجا ماژول STACK BASED ALU به پایان رسید حال نیاز داریم که برای آن Test bench بنویسیم تا از درستی ماژول خود مطمئن شویم. Test bench این ماژول در پوشه code و با نام TB_STACK.v موجود می‌باشد. در ادامه به توضیح آن و بررسی ورودی‌ها و خروجی‌های آن می‌پردازیم.

```
module TB_STACK ();
    reg clk;
    parameter N = 4;

    reg signed [N-1:0] input_data;
    reg [2:0] opcode;
    wire signed [N-1:0] output_data;
    wire overflow;

    integer i;

    reg signed [N-1:0] element1;
    reg signed [N-1:0] element2;

    reg signed [N-1:0] true_output;
    reg true_overflow;

    integer number_of_false_result;
    integer number_of_false_overflow;

    STACK_BASED_ALU #(N) Stack_based_alu(clk, input_data, opcode, output_data, overflow);
```

در ابتدا کلاک و پارامتر N را تعریف می‌کنیم. سپس ورودی‌ها و خروجی‌های ماژول STACK BASED ALU را تعریف می‌کنیم در ادامه نیز یک اینتیجر i تعریف می‌کنیم که برای loop هایی که در تست نوشتیم از آن استفاده کنیم. همچنین متغیرهای element1 و element2 را تعریف کردم که عناصری که با یکدیگر جمع یا ضرب می‌شوند را در آن‌ها بریزم تا بتوانم با استفاده از تابع‌هایی که تعریف کرده‌ام(در ادامه آن‌ها را توضیح خواهم داد) صحت عملکرد ماژول خود را بررسی کنم. متغیرهای

true_output و true_overflow هم خروجی‌های تابع‌ها می‌باشند. متغیرهای number_of_false هم تعریف کردم تا با مقایسه خروجی‌های ماژول و تابع‌ها بررسی کنم که چه تعداد از خروجی‌های ماژول‌ها غلطاند و در آخر هم از ماژول STACK BASED ALU یک instance گرفتم و ورودی‌ها و خروجی‌های آن را ست کردم.

در ادامه کلاک خود را تنظیم کردم و مقادیر number_of_false را برابر صفر قرار دادم.

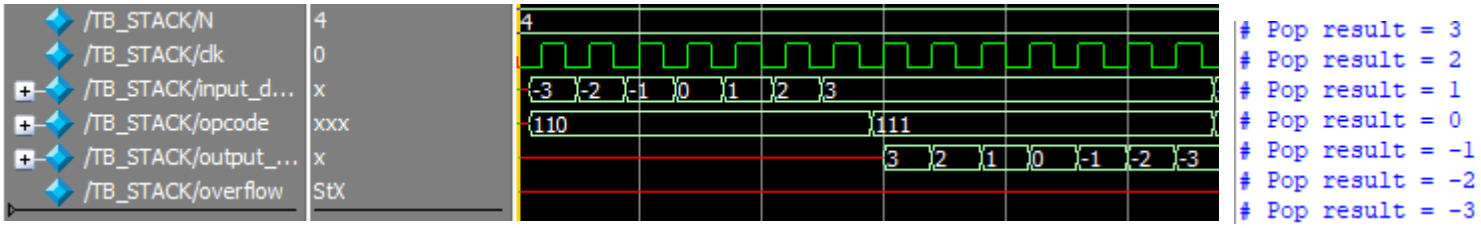
```
always #10 clk <= ~clk;

initial begin
    clk = 0;
    number_of_false_overflow = 0;
    number_of_false_result = 0;
    #5;
```

در تست اول صرفا عملکرد پوش و پاپ و ذخیره‌سازی اعداد در استک را بررسی کردم در ابتدا اعداد منفی سه تا سه را در استک پوش کردم و سپس اعداد را پاپ کردم و مقادیر آن‌ها را بررسی کردم که خروجی آن را می‌توانید در تصاویر زیر مشاهده کنید. (منطقا این قسمت به پارامتر N بستگی ندارد و به ازای Nهای متفاوت خروجی ما تغییر نخواهد کرد)

```
for (i = -3; i < 4; i = i + 1) begin
    input_data = i;
    opcode = 3'b110;
    #20;
end


for (i = 0; i < 7; i = i + 1) begin
    opcode = 3'b111;
    #10;
    $display("Pop result = %0d", output_data);
    #10;
end
```



همان طور که در تصاویر مشاهده می‌کنید اعداد به درستی در استک ذخیره شده بودند و به صورت برعکس در خروجی قابل مشاهده می‌باشند. بیت overflow هم X می‌باشد چون مقدار آن در پوش و پاپ معنایی ندارد و همچنین خروجی استک هنگام پوش کردن X است چون خروجی آن هنگام پوش کردن معنایی ندارد.

ابتدا تابع‌هایی که تعریف کرده‌ام را بررسی می‌کنیم و سپس به بررسی ادامه تست‌ها می‌پردازیم:

```
task print;
    input operand;
    input signed [N-1:0] element1;
    input signed [N-1:0] element2;
    input signed [N-1:0] output_data;
    input overflow;
    begin
        //Add
        if(operand == 0) begin
            $display("input 1 = %0d, input 2 = %0d, Adder output = %0d, overflow = %0d", element1, element2, output_data, overflow);
        end

        //Multiply
        else begin
            $display("input 1 = %0d, input 2 = %0d, Multiplier output = %0d, overflow = %0d", element1, element2, output_data, overflow);
        end
    end
endtask
```

ابتدا تابع print را تعریف کردم که ورودی‌های آن operand(که بررسی می‌کند که چه عملی دارد صورت می‌گیرد تا عبارت صحیح را چاپ کند) و element1 و element2 و output_data و overflow هستند که دو عنصر اول اعدادی هستند که عملیات روی آن‌ها انجام می‌شود و دو ورودی آخر هم خروجی‌های ماژول ما می‌باشند. در ادامه نیز بر اساس این که کدام عملیات انجام می‌شود عبارت مناسب را چاپ می‌کنیم.

```
task true_result;
    input operand;
    input signed [N-1:0] element1;
    input signed [N-1:0] element2;
    output signed [N-1:0] true_output;
    output true_overflow;
    reg signed [2*N - 1:0] double_bit_result;
    begin
        true_overflow = 0;
        //Add
        if (operand == 0) begin
            double_bit_result = element1 + element2;
            true_output = double_bit_result[N-1:0];
            if ((element1[N-1] == element2[N-1]) && (element1[N-1] != true_output[N-1])) begin
                true_overflow = 1;
            end
        end

        //Multiply
        else begin
            double_bit_result = element1 * element2;
            true_output = double_bit_result[N-1:0];
            if ((element1 != 0 && element2 != 0) && ((true_output / element1) != element2)) begin
                true_overflow = 1;
            end
        end
    end
endtask
```

در ادامه تابع true_result را تعریف کردم که ورودی اول آن operand است که بفهمد چه عملیاتی باید انجام شود. دو ورودی بعدی آن (element1 و element2) دو عددی هستند که عملیات روی آن‌ها انجام می‌شود. دو خروجی آن مقدار صحیح عملیات ضرب یا جمع و مقدار صحیح بیت overflow می‌باشد. در ادامه یک متغیر تعریف کردم که تعداد اندازه آن دوبرابر اعداد ورودی هستند تا بتوانیم با کمک آن مقادیر صحیح را حساب کنیم(در ادامه دقیق‌تر توضیح خواهم داد)

ابتدای کار که مقدار overflow را برابر صفر قرار می‌دهم(اگر در جایی overflow رخ دهد آن را یک خواهم کرد)

ابتدا بررسی می‌شود که اگر باید جمع انجام شود مقدار صحیح آن را حساب کنیم. ابتدا حاصل جمع دو عدد را حساب کرده و در متغیر بزرگ‌تر میریزم سپس N بیت اول آن عدد را برمی‌دارم و به عنوان خروجی صحیح نشان می‌دهم. سپس منطق overflow را بر اساس ورودی‌ها و مقدار صحیحی که از عملیات جمع به دست آمده بررسی می‌کنم که اگر overflow رخ داده است مقدار بیت آن را یک می‌کنم.

در ادامه عملیات ضرب بررسی می‌شود که دو عدد را در هم ضرب کرده و در متغیر بزرگ‌تر ذخیره می‌کنیم و سپس N بیت اول آن را برمی‌دارم و به عنوان خروجی صحیح قرار می‌دهم(چون می‌خواهم چک کنم که در هنگام overflow هم خروجی ضرب ALU ما به صورت صحیح split شده باشد و مقدار خروجی آن با N بیت اول حاصل ضرب دو عدد برابر می‌شود یا خیر) سپس overflow را بررسی می‌کنیم به این صورت که اگر هیچکدام از اعداد صفر نبودند و تقسیم جواب صحیح ضرب به یکی از اعداد برابر آن یکی عدد نشد به این معنا است که overflow رخ داده و بیت overflow را یک می‌کنم.

```
task print_true;
    input signed [N-1:0] true_output;
    input true_overflow;
    begin
        $display("True value: Result = %0d, overflow = %0d", true_output, true_overflow);
        $display();
    end
endtask
```

سپس تابع print_true را تعریف کردم که مقادیر صحیح جمع یا ضرب و بیت overflow را ورودی می‌گیرد و این مقادیر صحیح را چاپ می‌کند.

حال به بررسی تست‌ها می‌پردازیم:

اولین تست به این صورت می‌باشد که هر دفعه یک عدد رندوم N بیتی را در استک پوش می‌کند و عملیات را به صورت رندوم روی آن انجام می‌دهد.

```
input_data = ((-1) ** ($random() % 2)) * ($random() % (2**(N-1)));
opcode = 3'b110;
element1 = input_data;

#20;

input_data = ((-1) ** ($random() % 2)) * ($random() % (2**(N-1)));
opcode = 3'b110;
element2 = input_data;
```

در ابتدای کار برای این تست دو عدد رندوم N بیتی را در استک پوش می‌کنیم.(مقادیر element1 و element2 را هم برابر آن‌ها قرار می‌دهیم چون در ابتدا این دو عدد عملیات روی آن‌ها انجام می‌شوند)

سپس وارد for که در تصویر زیر آمده می‌شویم و بین عملیات جمع و ضرب یکی را رندوم انتخاب کرده و آن را انجام می‌دهیم.(تاخیر ۱۰ بعد از opcode برای این آورده شده که عملیات انجام شود و سپس خروجی آن را بررسی کنیم) در ادامه پس از اینکه عملیات انجام شد خروجی‌های مدار و اعدادی که عملیات بر روی آن‌ها انجام شده است را چاپ می‌کنیم. در خط بعدی مقدار صحیح خروجی‌ها را بر اساس ورودی‌ها حساب کرده و آن‌ها را چاپ می‌کنیم. در اخر هم بررسی می‌کنیم که اگر خروجی‌های ماژول ما با خروجی صحیح کی نبود به متغیرهای number of false یک واحد اضافه می‌کنیم. در آخر حلقه هم یک عدد دیگر در استک پوش می‌کنیم که در این حالت element1 ما همان element2 قدیمی می‌شود و element2 ما برابر با مقدار جدیدی که در

استک پوش کرده‌ایم می‌شود. این حلقه را ۳۰ بار اجرا می‌کنیم و بخشی از خروجی‌های آن را در تصاویر زیر می‌توانید مشاهده کنید.

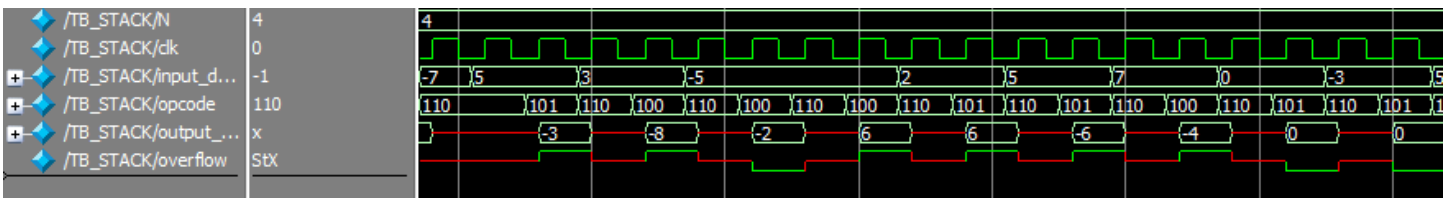
```
for (i = 0; i < 30; i = i + 1) begin
    if ($random() % 2 == 0) begin
        opcode = 3'b100;
        #10;
        print(0, element1, element2, output_data, overflow);
        true_result(0, element1, element2, true_output, true_overflow);
        print_true(true_output, true_overflow);
        if (overflow != true_overflow) begin
            number_of_false_overflow = number_of_false_overflow + 1;
        end
        if (output_data != true_output) begin
            number_of_false_result = number_of_false_result + 1;
        end
    end

    else begin
        opcode = 3'b101;
        #10;
        print(1, element1, element2, output_data, overflow);
        true_result(1, element1, element2, true_output, true_overflow);
        print_true(true_output, true_overflow);
        if (overflow != true_overflow) begin
            number_of_false_overflow = number_of_false_overflow + 1;
        end
        if (output_data != true_output) begin
            number_of_false_result = number_of_false_result + 1;
        end
    end

    #10;

    element1 = element2;
    input_data = ((-1) ** ($random() % 2)) * ($random() % (2**(N-1)));
    opcode = 3'b110;
    element2 = input_data;

    #20;
end
```



همان طور که در تصاویر مشاهده می‌کنید برای N = 4 ماژول ما به درستی کار کرده و خروجی‌های آن با خروجی‌های صحیح یکی می‌باشد.

```
input 1 = -7, input 2 = 5, Multiplier output = -3, overflow = 1
True value: Result = -3, overflow = 1

input 1 = 5, input 2 = 3, Adder output = -8, overflow = 1
True value: Result = -8, overflow = 1

input 1 = 3, input 2 = -5, Adder output = -2, overflow = 0
True value: Result = -2, overflow = 0

input 1 = -5, input 2 = -5, Adder output = 6, overflow = 1
True value: Result = 6, overflow = 1

input 1 = -5, input 2 = 2, Multiplier output = 6, overflow = 1
True value: Result = 6, overflow = 1

input 1 = 2, input 2 = 5, Multiplier output = -6, overflow = 1
True value: Result = -6, overflow = 1

input 1 = 5, input 2 = 7, Adder output = -4, overflow = 1
True value: Result = -4, overflow = 1

input 1 = 7, input 2 = 0, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0

input 1 = 0, input 2 = -3, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0
```

```
input 1 = -1064739199, input 2 = 1309649309, Multiplier output = -1691279843, overflow = 1
True value: Result = -1691279843, overflow = 1

input 1 = 1309649309, input 2 = 1295874971, Adder output = -1689443016, overflow = 1
True value: Result = -1689443016, overflow = 1

input 1 = 1295874971, input 2 = -114806029, Adder output = 1181068942, overflow = 0
True value: Result = 1181068942, overflow = 0

input 1 = -114806029, input 2 = -1993627629, Adder output = -2108433658, overflow = 0
True value: Result = -2108433658, overflow = 0

input 1 = -1993627629, input 2 = 482925370, Multiplier output = -813410994, overflow = 1
True value: Result = -813410994, overflow = 1

input 1 = 482925370, input 2 = 1924134885, Multiplier output = 1003950306, overflow = 1
True value: Result = 1003950306, overflow = 1

input 1 = 1924134885, input 2 = 1206705039, Adder output = -1164127372, overflow = 1
True value: Result = -1164127372, overflow = 1

input 1 = 1206705039, input 2 = -201295128, Multiplier output = 1531518872, overflow = 1
True value: Result = 1531518872, overflow = 1

input 1 = -201295128, input 2 = -561108803, Multiplier output = -1807811256, overflow = 1
True value: Result = -1807811256, overflow = 1
```

چند نمونه تست برای N = 32 که در اینجا هم ماژول به درستی کار کرده است.

چند نمونه تست برای N = 8 که در اینجا هم ماژول به درستی کار کرده است.

```
input 1 = -127, input 2 = 29, Multiplier output = -99, overflow = 1
True value: Result = -99, overflow = 1

input 1 = 29, input 2 = 27, Adder output = 56, overflow = 0
True value: Result = 56, overflow = 0

input 1 = 27, input 2 = -13, Adder output = 14, overflow = 0
True value: Result = 14, overflow = 0

input 1 = -13, input 2 = -109, Adder output = -122, overflow = 0
True value: Result = -122, overflow = 0

input 1 = -109, input 2 = 58, Multiplier output = 78, overflow = 1
True value: Result = 78, overflow = 1

input 1 = 58, input 2 = 101, Multiplier output = -30, overflow = 1
True value: Result = -30, overflow = 1

input 1 = 101, input 2 = 15, Adder output = 116, overflow = 0
True value: Result = 116, overflow = 0

input 1 = 15, input 2 = -24, Multiplier output = -104, overflow = 1
True value: Result = -104, overflow = 1

input 1 = -24, input 2 = -67, Multiplier output = 72, overflow = 1
True value: Result = 72, overflow = 1

input 1 = -67, input 2 = 29, Adder output = -38, overflow = 0
True value: Result = -38, overflow = 0
```

چند نمونه تست برای N = 16 که در اینجا هم ماژول به درستی کار کرده است.

```
input 1 = -8575, input 2 = 10653, Multiplier output = 7709, overflow = 1
True value: Result = 7709, overflow = 1

input 1 = 10653, input 2 = 31643, Adder output = -23240, overflow = 1
True value: Result = -23240, overflow = 1

input 1 = 31643, input 2 = -19725, Adder output = 11918, overflow = 0
True value: Result = 11918, overflow = 0

input 1 = -19725, input 2 = -22509, Adder output = 23302, overflow = 1
True value: Result = 23302, overflow = 1

input 1 = -22509, input 2 = 23354, Multiplier output = -10930, overflow = 1
True value: Result = -10930, overflow = 1

input 1 = 23354, input 2 = 30693, Multiplier output = -28446, overflow = 1
True value: Result = -28446, overflow = 1

input 1 = 30693, input 2 = 23439, Adder output = -11404, overflow = 1
True value: Result = -11404, overflow = 1

input 1 = 23439, input 2 = -1304, Multiplier output = -24680, overflow = 1
True value: Result = -24680, overflow = 1
```

پس از انجام این تست استک را خالی می‌کنیم که برای تست بعدی آماده شود.

```
for (i = 0; i < 32; i = i + 1) begin
    opcode = 3'b111;
    #20;
end
```

در تست بعدی که در تصویر زیر آمده مشابه تست اول عمل می‌کنیم تنها تفاوت آن این است که به جای اینکه یکی یکی در استک پوش کنیم تا عدد جدید با عددی که قبلا در استک است عملیات انجام دهد، دو عدد را در استک پوش می‌کنیم و سپس عملیات را انجام می‌دهیم تا هر دفعه عملیات با دو عدد جدید انجام شود. چند نمونه خروجی این تست هم در تصاویر زیر آمده است.

```
for (i = 0; i < 20; i = i + 1) begin
    input_data = ((-1) ** ($random() % 2)) * ($random() % (2**(N-1)));
    opcode = 3'b110;
    element1 = input_data;
    #20;
    input_data = ((-1) ** ($random() % 2)) * ($random() % (2**(N-1)));
    opcode = 3'b110;
    element2 = input_data;
    #20;
    if ($random() % 2 == 0) begin
        opcode = 3'b100;
        #10;
        print(0, element1, element2, output_data, overflow);
        true_result(0, element1, element2, true_output, true_overflow);
        print_true(true_output, true_overflow);
        if (overflow != true_overflow) begin
            number_of_false_overflow = number_of_false_overflow + 1;
        end
        if (output_data != true_output) begin
            number_of_false_result = number_of_false_result + 1;
        end
    end
    else begin
        opcode = 3'b101;
        #10;
        print(1, element1, element2, output_data, overflow);
        true_result(1, element1, element2, true_output, true_overflow);
        print_true(true_output, true_overflow);
        if (overflow != true_overflow) begin
            number_of_false_overflow = number_of_false_overflow + 1;
        end
        if (output_data != true_output) begin
            number_of_false_result = number_of_false_result + 1;
        end
    end
    #10;
end
```

چند نمونه تست برای N = 4 که همان طور که می‌بینید ماژول به درستی کار کرده است.

```
input 1 = -1, input 2 = 5, Multiplier output = -5, overflow = 0
True value: Result = -5, overflow = 0

input 1 = 3, input 2 = 1, Adder output = 4, overflow = 0
True value: Result = 4, overflow = 0

input 1 = -2, input 2 = -4, Adder output = -6, overflow = 0
True value: Result = -6, overflow = 0

input 1 = 3, input 2 = 1, Multiplier output = 3, overflow = 0
True value: Result = 3, overflow = 0

input 1 = -4, input 2 = 5, Adder output = 1, overflow = 0
True value: Result = 1, overflow = 0

input 1 = 1, input 2 = 6, Multiplier output = 6, overflow = 0
True value: Result = 6, overflow = 0

input 1 = 1, input 2 = -4, Adder output = -3, overflow = 0
True value: Result = -3, overflow = 0

input 1 = 3, input 2 = 4, Adder output = 7, overflow = 0
True value: Result = 7, overflow = 0

input 1 = 5, input 2 = 6, Multiplier output = -2, overflow = 1
True value: Result = -2, overflow = 1

input 1 = 6, input 2 = 1, Adder output = 7, overflow = 0
True value: Result = 7, overflow = 0
```

چند نمونه تست برای N = 8 که همان طور که مشاهده می‌کنید ماژول به درستی کار کرده است.

```
input 1 = -57, input 2 = 45, Multiplier output = -5, overflow = 1
True value: Result = -5, overflow = 1

input 1 = 91, input 2 = 65, Adder output = -100, overflow = 1
True value: Result = -100, overflow = 1

input 1 = -122, input 2 = -100, Adder output = 34, overflow = 1
True value: Result = 34, overflow = 1

input 1 = 115, input 2 = 81, Multiplier output = 99, overflow = 1
True value: Result = 99, overflow = 1

input 1 = -68, input 2 = 53, Adder output = -15, overflow = 0
True value: Result = -15, overflow = 0

input 1 = 41, input 2 = 38, Multiplier output = 22, overflow = 1
True value: Result = 22, overflow = 1

input 1 = 33, input 2 = -68, Adder output = -35, overflow = 0
True value: Result = -35, overflow = 0

input 1 = 43, input 2 = 92, Adder output = -121, overflow = 1
True value: Result = -121, overflow = 1

input 1 = 61, input 2 = 78, Multiplier output = -106, overflow = 1
True value: Result = -106, overflow = 1

input 1 = 54, input 2 = 121, Adder output = -81, overflow = 1
True value: Result = -81, overflow = 1
```

چند نمونه تست برای N = 16 که همان طور که می‌بینید ماژول به درستی کار کرده است.

```
input 1 = -26425, input 2 = 2349, Multiplier output = -9733, overflow = 1
True value: Result = -9733, overflow = 1

input 1 = 22875, input 2 = 20929, Adder output = -21732, overflow = 1
True value: Result = -21732, overflow = 1

input 1 = -18298, input 2 = -3428, Adder output = -21726, overflow = 0
True value: Result = -21726, overflow = 0

input 1 = 6003, input 2 = 22481, Multiplier output = 14819, overflow = 1
True value: Result = 14819, overflow = 1

input 1 = -3396, input 2 = 18741, Adder output = 15345, overflow = 0
True value: Result = 15345, overflow = 0

input 1 = 16681, input 2 = 4902, Multiplier output = -18666, overflow = 1
True value: Result = -18666, overflow = 1

input 1 = 27425, input 2 = -32580, Adder output = -5155, overflow = 0
True value: Result = -5155, overflow = 0

input 1 = 11691, input 2 = 11740, Adder output = 23431, overflow = 0
True value: Result = 23431, overflow = 0

input 1 = 32061, input 2 = 26958, Multiplier output = 11670, overflow = 1
True value: Result = 11670, overflow = 1
```

چند نمونه تست برای N = 32 که همان طور که می‌بینید ماژول به درستی کار کرده است.

```
input 1 = -484042553, input 2 = 370411821, Multiplier output = 1310546427, overflow = 1
True value: Result = 1310546427, overflow = 1

input 1 = 769284443, input 2 = 1611485633, Adder output = -1914197220, overflow = 1
True value: Result = -1914197220, overflow = 1

input 1 = -1019266938, input 2 = -835718500, Adder output = -1854985438, overflow = 0
True value: Result = -1854985438, overflow = 0

input 1 = 966137715, input 2 = 1751537617, Multiplier output = 357775843, overflow = 1
True value: Result = 357775843, overflow = 1

input 1 = -573115716, input 2 = 442812725, Adder output = -130302991, overflow = 0
True value: Result = -130302991, overflow = 0

input 1 = 349159721, input 2 = 313299750, Multiplier output = 206386966, overflow = 1
True value: Result = 206386966, overflow = 1

input 1 = 272722721, input 2 = -573701956, Adder output = -300979235, overflow = 0
True value: Result = -300979235, overflow = 0

input 1 = 1442229675, input 2 = 1851633116, Adder output = -1001104505, overflow = 1
True value: Result = -1001104505, overflow = 1
```

در تست بعدی همه عناصر موجود در استک را پاپ می‌کنیم اما به تعداد بیشتر از مقادیری که پوش کرده بودیم(در استک ۴۰ عدد پوش کرده بودیم اما در اینجا ۴۹ عدد پاپ می‌کنیم) پس منطقا باید ۹ مقدار خروجی آخر ما برابر X باشد چون استک ما خالی می‌باشد.

```
for (i = 0; i < 49; i = i + 1) begin
    opcode = 3'b111;
    #10;
    $display("ouput pop = %0d", output_data);
    #10;
end
```

همان طور که مشاهده می‌کنید در آخر کار مقادیری پاپ شده‌اند و ۹ پاپ آخر ما برابر X شده است چون استک خالی است.

```
ouput pop = 3
ouput pop = 5
ouput pop = -1
ouput pop = x
ouput pop = x
ouput pop = x
ouput pop = x
ouput pop = x
ouput pop = x
ouput pop = x
ouput pop = x
ouput pop = x
```

در تست بعدی که در تصویر زیر می‌بینید ما edge case ها را بررسی می‌کنیم در ابتدا کوچکترین عدد منفی ممکن را در استک پوش می‌کنیم و سپس بزرگترین عدد مثبت را پوش می‌کنیم و ابتدای کار عملیات جمع را روی آن‌ها انجام می‌دهیم و سپس عملیات ضرب که خروجی‌های آن‌ها را در تصاویر زیر می‌توانید ببینید.(در آخر کار هم بزرگترین عدد مثبت را پاپ می‌کنم تا برای تست بعدی آماده شویم)

```
input_data = -(2**(N-1));
element1 = input_data;
opcode = 3'b110;

#20;

input_data = 2**(N-1) - 1;
element2 = input_data;
opcode = 3'b110;

#20;

opcode = 3'b100;
#10;
print(0, element1, element2, output_data, overflow);
true_result(0, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

opcode = 3'b101;
#10;
print(1, element1, element2, output_data, overflow);
true_result(1, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

opcode = 3'b111;
```

```
input 1 = -8, input 2 = 7, Adder output = -1, overflow = 0
True value: Result = -1, overflow = 0

input 1 = -8, input 2 = 7, Multiplier output = -8, overflow = 1
True value: Result = -8, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 4 به درستی کار کرده است.

```
input 1 = -128, input 2 = 127, Adder output = -1, overflow = 0
True value: Result = -1, overflow = 0

input 1 = -128, input 2 = 127, Multiplier output = -128, overflow = 1
True value: Result = -128, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 8 به درستی کار کرده است.

```
input 1 = -32768, input 2 = 32767, Adder output = -1, overflow = 0
True value: Result = -1, overflow = 0

input 1 = -32768, input 2 = 32767, Multiplier output = -32768, overflow = 1
True value: Result = -32768, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 16 به درستی کار کرده است.

```
input 1 = -2147483648, input 2 = 2147483647, Adder output = -1, overflow = 0
True value: Result = -1, overflow = 0

input 1 = -2147483648, input 2 = 2147483647, Multiplier output = -2147483648, overflow = 1
True value: Result = -2147483648, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 32 به درستی کار کرده است.

در ادامه کوچکترین عدد منفی ممکن را پوش کردم تا ضرب و جمع آن را با خودش بررسی کنم.(توجه شود که در تست قبلی بزرگترین عدد مثبت را بعد از تست پاپ کردیم پس تنها عنصر استک کوچکترین عدد منفی می‌باشد)

```
input_data = -(2**(N-1));
element2 = input_data;
opcode = 3'b110;

#20;

opcode = 3'b100;
#10;
print(0, element1, element2, output_data, overflow);
true_result(0, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

opcode = 3'b101;
#10;
print(1, element1, element2, output_data, overflow);
true_result(1, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;
```

در تصاویر زیر خروجی‌های آن را می‌توانید مشاهده کنید.

همان طور که مشاهده می‌کنید ماژول ما برای N = 4 به درستی کار کرده است.

```
input 1 = -8, input 2 = -8, Adder output = 0, overflow = 1
True value: Result = 0, overflow = 1

input 1 = -8, input 2 = -8, Multiplier output = 0, overflow = 1
True value: Result = 0, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 8 به درستی کار کرده است.

```
input 1 = -128, input 2 = -128, Adder output = 0, overflow = 1
True value: Result = 0, overflow = 1

input 1 = -128, input 2 = -128, Multiplier output = 0, overflow = 1
True value: Result = 0, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 16 به درستی کار کرده است.

```
input 1 = -32768, input 2 = -32768, Adder output = 0, overflow = 1
True value: Result = 0, overflow = 1

input 1 = -32768, input 2 = -32768, Multiplier output = 0, overflow = 1
True value: Result = 0, overflow = 1
```

همان طور که مشاهده می‌کنید ماژول ما برای N = 32 به درستی کار کرده است.

```
input 1 = -2147483648, input 2 = -2147483648, Adder output = 0, overflow = 1
True value: Result = 0, overflow = 1

input 1 = -2147483648, input 2 = -2147483648, Multiplier output = 0, overflow = 1
True value: Result = 0, overflow = 1
```

در تست زیر دوبار بزرگترین عدد مثبت را پوش می‌کنیم تا ضرب و جمع را با آن بررسی کنیم که خروجی‌های ماژول را در تصاویر زیر مشاهده می‌کنید.

```
input_data = (2**(N-1)) - 1;
element1 = input_data;
opcode = 3'b110;

#20;

input_data = (2**(N-1)) - 1;
element2 = input_data;
opcode = 3'b110;

#20;

opcode = 3'b100;
#10;
print(0, element1, element2, output_data, overflow);
true_result(0, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

opcode = 3'b101;
#10;
print(1, element1, element2, output_data, overflow);
true_result(1, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;
```

همان طور که مشاهده می‌کنید برای N = 4 ماژول به درستی کار کرده است.

```
input 1 = 7, input 2 = 7, Adder output = -2, overflow = 1
True value: Result = -2, overflow = 1

input 1 = 7, input 2 = 7, Multiplier output = 1, overflow = 1
True value: Result = 1, overflow = 1
```

همان طور که مشاهده می‌کنید برای N = 8 ماژول به درستی کار کرده است.

```
input 1 = 127, input 2 = 127, Adder output = -2, overflow = 1
True value: Result = -2, overflow = 1

input 1 = 127, input 2 = 127, Multiplier output = 1, overflow = 1
True value: Result = 1, overflow = 1
```

همان طور که مشاهده می‌کنید برای N = 16 ماژول به درستی کار کرده است.

```
input 1 = 32767, input 2 = 32767, Adder output = -2, overflow = 1
True value: Result = -2, overflow = 1

input 1 = 32767, input 2 = 32767, Multiplier output = 1, overflow = 1
True value: Result = 1, overflow = 1
```

همان طور که مشاهده می‌کنید برای N = 32 ماژول به درستی کار کرده است.

```
input 1 = 2147483647, input 2 = 2147483647, Adder output = -2, overflow = 1
True value: Result = -2, overflow = 1

input 1 = 2147483647, input 2 = 2147483647, Multiplier output = 1, overflow = 1
True value: Result = 1, overflow = 1
```

در تست بعدی که در تصویر زیر مشاهده می‌کنید عدد صفر را در استک پوش می‌کنیم و عملیات‌های جمع و ضرب آن را با بزرگترین عدد مثبت بررسی می‌کنیم. خروجی‌های آن را در تصاویر زیر می‌توانید مشاهده کنید. در آخر هم استک را خالی می‌کنیم تا برای تست بعدی آماده شویم.

```
element1 = (2**(N-1)) - 1;
input_data = 0;
element2 = input_data;
opcode = 3'b110;

#20;

opcode = 3'b100;
#10;
print(0, element1, element2, output_data, overflow);
true_result(0, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

opcode = 3'b101;
#10;
print(1, element1, element2, output_data, overflow);
true_result(1, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

for(i = 0; i < 6; i = i + 1) begin
    opcode = 3'b111;
    #20;
end
```

همان طور که مشاهده می‌کنید برای N = 4 ماژول ما به درستی کار کرده است.

```
input 1 = 7, input 2 = 0, Adder output = 7, overflow = 0
True value: Result = 7, overflow = 0

input 1 = 7, input 2 = 0, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0
```

همان طور که مشاهده می‌کنید برای N = 8 ماژول ما به درستی کار کرده است.

```
input 1 = 127, input 2 = 0, Adder output = 127, overflow = 0
True value: Result = 127, overflow = 0

input 1 = 127, input 2 = 0, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0
```

همان طور که مشاهده می‌کنید برای N = 16 ماژول ما به درستی کار کرده است.

```
input 1 = 32767, input 2 = 0, Adder output = 32767, overflow = 0
True value: Result = 32767, overflow = 0

input 1 = 32767, input 2 = 0, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0
```

همان طور که مشاهده می‌کنید برای N = 32 ماژول ما به درستی کار کرده است.

```
input 1 = 2147483647, input 2 = 0, Adder output = 2147483647, overflow = 0
True value: Result = 2147483647, overflow = 0

input 1 = 2147483647, input 2 = 0, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0
```

در تست بعدی هر ۵۱۲ خانه استک را با صفر پر می‌کنیم و سپس عدد یک را در استک می‌خواهیم پوش کنیم اما چون استک پر است منطقا نباید پوش شود سپس عملیات جمع و ضرب را انجام می‌دهیم تا مشاهده کنیم که یک وارد نشده باشد. (در اینجا element1 و element2 را برابر صفر قرار داده‌ام تا تابع‌هایی که تعریف کرده‌ایم به درستی کار کنند و این دو عنصر تاثیری در ماژول نخواهند داشت) در تصاویر زیر خروجی‌های آن را مشاهده می‌کنید.

```
for(i = 0; i < 512; i = i + 1) begin
    input_data = 0;
    opcode = 3'b110;
    #20;
end

input_data = 1;
opcode = 3'b110;

#20;

opcode = 3'b100;
element1 = 0;
element2 = 0;
#10;
print(0, element1, element2, output_data, overflow);
true_result(0, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;

opcode = 3'b101;
#10;
print(1, element1, element2, output_data, overflow);
true_result(1, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;
```

برای این تست چون N هیچ تاثیری ندارد صرفا یک عکس قرار می‌دهیم.

```
input 1 = 0, input 2 = 0, Adder output = 0, overflow = 0
True value: Result = 0, overflow = 0

input 1 = 0, input 2 = 0, Multiplier output = 0, overflow = 0
True value: Result = 0, overflow = 0
```

همان طور که مشاهده می‌کنید خروجی‌های ALU ما برابر صفر شده است که به این معنا می‌باشد که یک پوش نشده چون استک پر بوده است.

در تست زیر ابتدا یک عنصر را از استک پاپ می‌کنیم و سپس عدد یک را پوش می‌کنیم و سپس عملیات جمع را انجام می‌دهیم.

```
opcode = 3'b111;

#20;

input_data = 1;
element2 = 1;
opcode = 3'b110;

#20;

opcode = 3'b100;
#10;
print(0, element1, element2, output_data, overflow);
true_result(0, element1, element2, true_output, true_overflow);
print_true(true_output, true_overflow);
if (overflow != true_overflow) begin
    number_of_false_overflow = number_of_false_overflow + 1;
end
if (output_data != true_output) begin
    number_of_false_result = number_of_false_result + 1;
end
#10;
```

```
input 1 = 0, input 2 = 1, Adder output = 1, overflow = 0
True value: Result = 1, overflow = 0
```

همان طور که مشاهده می‌کنید در اینجا عدد یک وارد استک شده و عملیات بر روی آن انجام شده است چون ابتدا یک عنصر را پاپ کردیم استک ما یک جای خالی دارد پس ۱ می‌توانید در آن قرار گیرد.

در تست آخر هم No-op ها را بررسی می‌کنیم که خروجی ماژول ما باید X باشد.

در آخر هم تعداد اشتباه‌های ماژول را چاپ می‌کنیم.

```
for (i = 0; i < 4; i = i + 1) begin
    opcode = i;
    #10;
    $display("Output on No-Op = %0d", output_data);
    #10;
end
$display();
$display("Number of false overflow = %0d, Number of false result = %0d for N = %0d", number_of_false_overflow, number_of_false_result, N);
$finish();
```

```
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x

Number of false overflow = 0, Number of false result = 0 for N = 4
```

همان طور که مشاهده می‌کنید برای No-op N = 4 صحیح عمل کرده است. همچنین تعداد غلط‌های ما صفر بوده یعنی اینکه برای این تست‌ها ماژول ما بی‌نقص عمل کرده است.

همان طور که مشاهده می‌کنید برای N = 8 هم مدار بی‌نقص بوده است.

```
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x

Number of false overflow = 0, Number of false result = 0 for N = 8
```

همان طور که مشاهده می‌کنید برای N = 16 هم مدار بی‌نقص بوده است.

```
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x

Number of false overflow = 0, Number of false result = 0 for N = 16
```

همان طور که مشاهده می‌کنید برای N = 32 هم مدار بی‌نقص بوده است.

```
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x
Output on No-Op = x

Number of false overflow = 0, Number of false result = 0 for N = 32
```

حال که از صحت کارکرد ماژول STACK BASED ALU مطمئن شدیم باید ماژولی طراحی کنیم که با کمک گرفتن از آن بتواند عبارات ریاضی را حل کند. کد این بخش در پوشه code با نام EQU_SOLVER.v می‌باشد.

برای طراحی این ماژول ابتدا نیاز داشتیم که یک پیش پردازش بر روی عبارت ورودی انجام دهیم تا به صورت پسوندی در بیاید و سپس حاصل این عبارت پسوندی را با کمک STACK BASED ALU به دست بیاوریم. در ادامه به توضیح دقیق‌تر درمورد آن می‌پردازیم. در این سوال چون در صورت سوال اعداد خاصی بیان نشده بودند من فرض کردم که اعداد ۳۲ بیتی داریم.

```
module Solver (
    input clk,
    input wire [4095:0] equation,
    output reg signed [31:0] result
);

    reg signed [7:0] postfix [0:511];
    reg signed [7:0] tmp [0:511];
    integer i, postfix_idx, tmp_idx;
    reg break;
    reg break_for;
    reg operand_on;
    reg pop_first;
    reg pop_second;
    reg pop_result;
    reg end_op;
    reg result_ready;
    reg number;

    reg signed [31:0] save_result;

    reg signed [31:0] input_data;
    reg [2:0] opcode;
    wire signed [31:0] output_data;
    wire overflow;

    STACK_BASED_ALU #(32) stack_based_alu(clk, input_data, opcode, output_data, overflow);
```

در ابتدا ورودی‌های و خروجی‌ها را قرار می‌دهیم. منطقا ورودی کلاک می‌خواهیم چون STACK BASED ALU ما عملیات‌های خود را بر اساس کلاک انجام می‌دهد. ورودی ۴۰۹۶ بیتی equation هم می‌خواهیم که همان عبارت ریاضی ورودی ما می‌باشد.(۴۰۹۶ بیت همان ۸ * ۵۱۲ می‌باشد به این معنا که عبارت ریاضی ما می‌تواند شامل ۵۱۲ کاراکتر باشد. ۸ هم به دلیل ASCII می‌باشد) و در نهایت هم یک خروجی ۳۲ بیتی داریم که جواب عبارت ما می‌باشد.

در ادامه آرایه ۵۱۲ تایی از رجیسترهای ۸ بیتی postfix را تعریف کرده‌ایم که عبارت ریاضی ما به صورت پسوندی در آن ذخیره می‌شود. tmp هم آرایه‌ای است که برای تبدیل کردن عبارت به پسوندی به آن نیاز داشتیم.

در ادامه متغیرهایی تعریف شده‌اند که کاربرد هرکدام در ادامه گفته خواهد شد.

همچنین یک instance از STACK BASED ALU خود گرفته‌ایم تا بتوانیم عبارات پسوندی را با آن حساب کنیم.

```
always @(equation) begin
    for (i = 0; i < 512; i = i + 1) begin
        postfix[i] = 8'bx;
        tmp[i] = 8'bx;
    end
    postfix_idx = 0;
    tmp_idx = 0;
    for (i = 511; i >= 0; i = i - 1) begin
        if (equation[i*8 +: 8] != 0)
        begin
            if (equation[i*8 +: 8] != 8'h2B &&
                equation[i*8 +: 8] != 8'h2A &&
                equation[i*8 +: 8] != 8'h28 &&
                equation[i*8 +: 8] != 8'h29)
            begin
                postfix[postfix_idx] = equation[i*8 +: 8];
                postfix_idx = postfix_idx + 1;
            end
            else if (equation[i*8 +: 8] == 8'h28) begin
                tmp[tmp_idx] = equation[i*8 +: 8];
                tmp_idx = tmp_idx + 1;
            end
            else if (equation[i*8 +: 8] == 8'h29) begin
                break = 0;
                while (tmp_idx != 0 && !break) begin
                    if (tmp[tmp_idx - 1] != 8'h28) begin
                        postfix[postfix_idx] = tmp[tmp_idx - 1];
                        postfix_idx = postfix_idx + 1;
                        tmp_idx = tmp_idx - 1;
                    end
                    else begin
                        tmp_idx = tmp_idx - 1;
                        break = 1;
                    end
                end
            end
```

در ادامه always ای تعریف کرده‌ام که به ازای هربار تغییر equation ورودی یک پیش پردازش را بر روی آن انجام می‌دهد تا آن را به صورت پسوندی در postfix ذخیره کند. در ابتدا آرایه‌های postfix و tmp را X می‌کنم تا اگر مقادیری از قبل در آن‌ها مانده بود از بین برود و در ادامه الگوریتم تبدیل کردن به postfix را بر روی equation ورودی اجرا می‌کنم ادامه کد را در تصویر زیر می‌توانید مشاهده کنید. این الگوریتم را با کمک گرفتن از لینکی که در README گیت‌هاب قرار داده‌ام پیاده سازی کردم و کد آن که به زبان جاوا بود را به یک‌سری تغییرات به وریلاگ تبدیل کردم. یکی از تفاوت‌هایی که وجود داشت این بود که ما صرفا باید عملگرهای ضرب و جمع را پیاده‌سازی می‌کردیم. تغییر دیگری که دادم این بود که اسپیس‌های موجود در عبارت ورودی را در عبارت پسوندی حذف نکردم چون اعداد با رقم‌های بیشتر از یک را نیز باید پشتیبانی می‌کردیم. اگر اسپیس را حذف می‌کردم نمی‌توانستیم اعداد را از هم جدا کنیم در این پیاده‌سازی اعداد مختلف با اسپیس از هم جدا می‌شوند و رقم‌های یک عدد بدون فاصله در کنار هم قرار می‌گیرند و این گونه می‌توانیم اعداد را از هم تشخیص دهیم.

```
        else if (equation[i*8 +: 8] == 8'h2B ||
                 equation[i*8 +: 8] == 8'h2A)
        begin
            if (tmp_idx == 0) begin
                tmp[tmp_idx] = equation[i*8 +: 8];
                tmp_idx = tmp_idx + 1;
            end
            else begin
                break = 0;
                while(tmp_idx != 0 && !break) begin
                    if (tmp[tmp_idx - 1] == 8'h28) begin
                        break = 1;
                    end
                    else if(tmp[tmp_idx - 1] == 8'h2A || tmp[tmp_idx - 1] == 8'h2B) begin
                        if (tmp[tmp_idx - 1] > equation[i*8 +: 8]) begin
                            break = 1;
                        end
                        else begin
                            postfix[postfix_idx] = tmp[tmp_idx - 1];
                            tmp_idx = tmp_idx - 1;
                            postfix_idx = postfix_idx + 1;
                        end
                    end
                end
                tmp[tmp_idx] = equation[i*8 +: 8];
                tmp_idx = tmp_idx + 1;
            end
        end
    end
end

while (tmp_idx != 0) begin
    postfix[postfix_idx] = tmp[tmp_idx - 1];
    tmp_idx = tmp_idx - 1;
    postfix_idx = postfix_idx + 1;
```

در آخر این پیش پردازش مقادیری که برای حساب کردن مقدار عبارت پسوندی نیاز هستند را مقدار دهی اولیه می‌کنیم.

```
save_result = 0;
postfix_idx = 0;
operand_on = 0;
pop_first = 0;
pop_second = 0;
pop_result = 0;
end_op = 0;
result_ready = 0;
```

در اینجا always را با لبه بالارونده کلاک تعریف کردم که حاصل عبارت پسوندی را حساب کند.

```
always @(posedge clk) begin
    if (pop_result) begin
        pop_result = 0;
        result_ready = 1;
    end
    else if (result_ready) begin
        result = output_data;
        result_ready = 0;
        end_op = 1;
    end
    else if (operand_on) begin
        opcode = 3'b111;
        pop_first = 1;
        operand_on = 0;
    end
    else if (pop_first) begin
        save_result = output_data;
        opcode = 3'b111;
        pop_second = 1;
        pop_first = 0;
    end
    else if (pop_second) begin
        input_data = save_result;
        opcode = 3'b110;
        pop_second = 0;
    end
```

اولین شرط زمانی درست می‌باشد که عبارت ما تمام شده باشد و آپکد پاپ فعال باشد به این معنا که ما می‌خواهیم جواب نهایی را پاپ کنیم و درون این شرط فلگش را صفر می‌کنیم تا دوباره داخل نشویم و فلگ result_ready را یک می‌کنیم تا در کلاک بعدی که جواب عبارت در output_data آماده می‌باشد، مقدارش را در result بریزیم و در آن‌جا فلگش را صفر می‌کنیم تا دوباره واردش نشویم و end_op را یک می‌کنیم به این معنا که عملیات ما تمام شده است. این دو شرط زمانی اتفاق می‌افتند که رشته پسوندی ما تمام شده باشد در آخر فلگ pop_result را بررسی می‌کنیم. سه شرط بعد از این دو مربوط به انجام عملیات‌های ضرب و جمع هستند و مربوط به این دو نیستند.

شرط بعدی(با فلگ operand_on) زمانی اتفاق می‌افتد که آپکد ضرب یا جمع اتفاق افتاده باشد پس عملیات در این کلاک انجام می‌شود و ما باید در کلاک بعدی عدد بالای استک را پاپ کنیم چون کار ما با آن تمام شده است. در شرط بعدی عملیات جمع تمام شده است و جواب آن آماده است پس جواب آن را در یک رجیستر ذخیره می‌کنیم تا آن را پوش کنیم. در این کلاک علملیات پاپ کردن عنصر اول دارد صورت می‌گیرد و ما آپکد را دوباره روی پاپ تنظیم می‌کنیم تا عنصر دوم نیز در کلاک بعدی پاپ شود. در کلاک بعدی وارد شرط آخر می‌شویم که عملیات پاپ عنصر دوم دارد انجام می‌شود پس ما آپکد را روی پوش و input_data را برابر save_result قرار می‌دهیم تا در کلاک بعدی جواب عملیات در استک پوش شود و در آن‌جا ذخیره شود.

```
else if (postfix_idx < 512 && postfix[postfix_idx] <= 57) begin
    if (postfix[postfix_idx] == 32) begin
       while(postfix[postfix_idx] == 32) begin
           postfix_idx = postfix_idx + 1;
        end
    end
    if (postfix[postfix_idx] == 45 || (postfix[postfix_idx] <= 57 && postfix[postfix_idx] >= 48)) begin
        input_data = 0;
        case (postfix[postfix_idx])
            45: begin
                postfix_idx = postfix_idx + 1;
                while(postfix[postfix_idx] <= 57 && postfix[postfix_idx] >= 48) begin
                    input_data = (10 * input_data) + (postfix[postfix_idx] - 48);
                    postfix_idx = postfix_idx + 1;
                end
                input_data = -1 * input_data;
            end
            default: begin
               while(postfix[postfix_idx] <= 57 && postfix[postfix_idx] >= 48) begin
                    input_data = (10 * input_data) + (postfix[postfix_idx] - 48);
                    postfix_idx = postfix_idx + 1;
                end
            end
        endcase
        opcode = 3'b110;
    end
    else begin
        case(postfix[postfix_idx])
            8'h2B: begin
                opcode = 3'b100;
                operand_on = 1;
            end
            8'h2A: begin
                opcode = 3'b101;
                operand_on = 1;
            end
        endcase
        postfix_idx = postfix_idx + 1;
    end
end
```

در ادامه در تصویر بالا به کد شکنی عبارت پسوندی می‌پردازیم به این صورت که تا زمانی که عبارت تمام نشده روی آرایه جلو می‌رویم و در ابتدای کار اگر اسپیس دیدیم آن‌قدر جلو می‌رویم تا اسپیس تمام شود. بعد از آن بررسی می‌کنیم که اگر به یک عدد و یا منفی رسیدیم عدد آن را حساب کنیم. در ابتدا بررسی می‌کنیم که اگر منفی داشتیم در آخر کار که عدد حساب شد آن را در

منفی یک ضرب کنیم. برای به دست آوردن اعداد هم هر دفعه که به یک رقم جدید از آن عدد می‌رسیم عدد قبلی را ضرب در ۱۰ کرده و با عدد جدید جمع می‌کنیم. این‌گونه اعداد با تعداد رقم بیشتر از یک هم هندل می‌شوند. حال بررسی می‌کنیم که اگر به + یا * رسیده بودیم opcode متناظر با آن را قرار می‌دهیم و operand_on را یک می‌کنیم تا سلسله مراتبی که در بالا توضیح دادیم اجرا شود.

به طور مثال فرض کنید عبارت ۱۲ + ۵ را داشته باشیم. پسوندی آن طبق الگوریتم ما به صورت +۱۲ ۵ در می‌آید که برای ۵ مستقیم عدد ۵ را قرار می‌دهد. در کلاک بعدی دو اسپیس را رد کرده و به ۱ می‌رسد و ۱ را در input_data قرار می‌دهد و می‌بینند که در ایندکس بعدی آن هم عدد وجود دارد نه اسپیس (چون اگر اسپیس بود طبق چیزی که بالاتر توضیح دادیم به این معنا است که به یک عدد جدید رسیده‌ایم) پس input_data را ضرب در ۱۰ می‌کند تا به عدد ۱۰ برسد سپس ۲ را با آن جمع می‌کند و به عدد ۱۲ می‌رسد. این گونه اعداد با ارقام بیشتر از یک هم هندل می‌شوند. همچنین بررسی شده است که اگر ابتدای عدد یک – بود در آخر input_data را ضرب در منفی یک کند.

```
    else begin
        if (end_op == 0) begin
            opcode = 3'b111;
            pop_result = 1;
        end
    end
end
```

در آخر هم اگر کل عبارت پسوندی ما تمام شده بود وارد این شرط می‌شویم ابتدا بررسی می‌کند که آیا حساب کردن عدد تمام شده است یا خیر اگر تمام نشده بود pop_result را یک می‌کند تا سلسله مراتبی که در بالا توضیح دادیم انجام شود.

دلیل قرار دادن end_op این بود که اگر چند کلاک اضافه‌تر ماندیم خروجی ما X نشود چون استک خالی شده و اگر چیزی پاپ کنیم X می‌گیریم.

در ادامه تست بنچی برای آن نوشتیم که در پوشه Code با نام TB_EQU_SOLVER.v موجود می‌باشد.

```
module TB_EQU_SOLVER();
    reg clk;
    reg [4095:0] equ;
    wire signed [31:0] result;

    Solver solver(clk, equ, result);

    always #10 clk = ~clk;
```

در ایتدا که ماژول را تعریف کردم و یک instance از solver گرفتم و ورودی و خروجی‌های مورد انتظارش را به آن دادم.

در ادامه همان طور که در تصویر زیر مشاهده می‌کنید به ازای عبارات مختلفی عملکرد مدار را بررسی می‌کنیم و هربار که جواب فرق می‌کند مقدار جواب را چاپ می‌کنیم. در پایان می‌توانید تصاویر خروجی آن را مشاهده کنید.

```
initial begin
    clk = 0;
    equ = "78";
    #80;
    equ = "(-2147483648 + 2147483647) * -12";
    #280;
    equ = "2 * 3 + (10 + 4 + 3) * -20 + (6 + 5)";
    #780;
    equ = "-1 * 0 + -2";
    #280;
    equ = "(((2 + 2) * 6) + -1) * -10";
    #480;
    equ = "3 + 4";
    #180;
    equ = "5 * 6";
    #180;
    equ = "(-3 + -4) * -5";
    #280;
    equ = "-40 + (-567 * (167 + 8))) + -12";
    #480;
    equ = "((81 + -12) * (0 + -1)) + 12";
    #480;
    equ = "0 * (12 * 24)";
    #280;
    equ = "-8 * -8";
    #180;
    equ = "-1 + -1 * -1 + -2";
    #380;
    equ = "(((((1 + 1) * 2) * 4) * 8) * 16) * -32";
    #680;
    equ = "(((-2 + -1) * (8 + 9) * -10) + (2 * 2) + -1) * 7";
    #880;
    equ = "1";
    #80;
    equ = "0 + 0";
    #180;
    equ = "-1 * -1";
    #180;
    $finish();
end
always @(result) begin
    $display("%0s = %0d", equ, result);
end
```

همان طور که مشاهده می‌کنید به ازای ورودی‌های مختلف ماژول ما به درستی عبارات را حساب کرده است پس می‌تاونیم از صحت عملکر آن اطمینان خاطر یابیم.

```
# 78 = 78
# (-2147483648 + 2147483647) * -12 = 12
# 2 * 3 + (10 + 4 + 3) * -20 + (6 + 5) = -323
# -1 * 0 + -2 = -2
# (((2 + 2) * 6) + -1) * -10 = -230
# 3 + 4 = 7
# 5 * 6 = 30
# (-3 + -4) * -5 = 35
# -40 + (-567 * (167 + 8))) + -12 = -99277
# ((81 + -12) * (0 + -1)) + 12 = -57
# 0 * (12 * 24) = 0
# -8 * -8 = 64
# -1 + -1 * -1 + -2 = -2
# (((((1 + 1) * 2) * 4) * 8) * 16) * -32 = -65536
# (((-2 + -1) * (8 + 9) * -10) + (2 * 2) + -1) * 7 = 3591
# 1 = 1
# 0 + 0 = 0
# -1 * -1 = 1
```