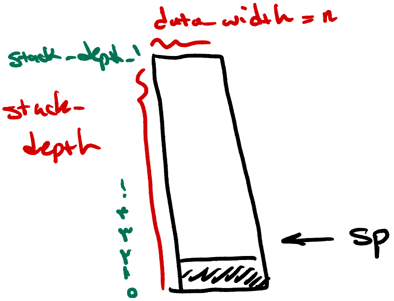
ابو الفضل دانایی ۴۰۱۱۰۲۶۹

به حل بخش الف سؤال یک می‌پردازیم:



sp: stack pointer « به پایین‌ترین خانه خالی اشاره می‌کند و در ابتدا صفر است »

```
module stack #(parameter DATA_WIDTH = 8, parameter STACK_DEPTH = 16) (
    input clk,
    input rst,
    input [2:0] opcode,
    input signed [DATA_WIDTH-1:0] data_in,
    output reg signed [DATA_WIDTH-1:0] data_out,
    output empty,
    output full,
    output reg overflow
);
```

ورودی ها:

opcode: ۳ بیت که مشخص می‌شود که چه کاری باید صورت بگیرد.

clk

rst: کل مدار reset می‌شود که کافیست sp به خانه صفر اشاره کند.

data_in: داده‌ای که دستور push وارد stack می‌شود.

خروجی ها:

empty: آیا stack خالی است «خالی است یا اشاره‌گر sp به خانه صفر sp==0»

full: آیا stack پر شده است که یعنی پایین‌ترین خانه خالی وجود ندارد یا به عبارتی دیگر پایین‌ترین خانه خالی، خانه بیرون آن است mem[stack_depth] یا پایین‌ترین خانه خالی است که وجود ندارد.

یعنی sp == stack_depth

overflow: برداشت من این است که تنها در عملیاتی که یک است حاصل جمع یا ضرب overflow کند.

data_out: خروجی حاصل از عمل‌های جمع یا ضرب از دو خانه بالایی

```
reg signed [DATA_WIDTH-1:0] stack_mem [0:STACK_DEPTH-1];
reg [$clog2(STACK_DEPTH)-1:0] sp;

reg signed [DATA_WIDTH-1:0] sum;
reg signed [2*DATA_WIDTH-1:0] product;

assign empty = (sp == 0);
assign full = (sp == STACK_DEPTH);
```

متغیرهای اضافه‌ای تعریف شده

sum حاصل جمع ابتدا در این جا محاسبه می‌شود. و با طول Data width

product " ضرب " " " " " " " " ذخیره ۲ عدد n بیتی، ۲n بیتی است و به این میزان بیت برای vector می‌گذاریم.

مقدار ۰= data out باتوجه به این دو vector توصیف می‌شود

stack_mem معادل حافظه stack است که به عنوان آرایه ۲ بعدی به طول data_width و عرض به طول stack_depth توصیف شده است.

sp stack pointer که چون قرار است stack_depth خانه را آدرس دهی کند. به log(stack_depth) بیت برای آدرس‌دهی نیاز داریم با سینتکس $clog2 ، log را بدست می‌آوریم.

نکته: چون می‌خواهیم بر روی اعداد صحیح علامت‌دار انجام دهیم، data_in ، data_out ، stack_mem ، sum ، product به صورت signed تعریف شده اند.

```
always @(posedge clk or posedge rst) begin
    if (rst) begin
        sp <= 0;
        overflow <= 0;
        data_out <= 0;
```

reset آسنکرون

اکنون بلاک always به لیست حساسیت کلاک و rst تریگر می‌کنیم

با فعال شدن پایه rst ، sp به خانه‌ی اشاره‌گر پشته حافظه است پایه‌های overflow و data_out ریست می‌شوند

```
end else begin
    overflow <= 0; // Reset overflow flag
    if (opcode == 3'b110) begin // Push
        if (!full) begin
            stack_mem[sp] <= data_in;
            sp <= sp + 1;
        end
    end else if (opcode == 3'b111) begin // Pop
        if (!empty) begin
            data_out <= stack_mem[sp-1];
            sp <= sp - 1;
        end
    end else if (opcode == 3'b100) begin // Add
        if (sp > 1) begin
            // Perform addition and check for overflow
            sum = stack_mem[sp-2] + stack_mem[sp-1];
            // Check for overflow
            if ((stack_mem[sp-2] > 0 && stack_mem[sp-1] > 0 && sum < 0) ||
                (stack_mem[sp-2] < 0 && stack_mem[sp-1] < 0 && sum > 0)) begin
                overflow <= 1;
            end
            data_out <= sum;
        end
    end else if (opcode == 3'b101) begin // Multiply
        if (sp > 1) begin
            // Perform multiplication and check for overflow
            product = stack_mem[sp-2] * stack_mem[sp-1];
            // Check for overflow by comparing to max and min values of the original data width
            if (product > $signed({1'b0, {(DATA_WIDTH-1){1'b1}}}) ||
                product < $signed({1'b1, {(DATA_WIDTH-1){1'b0}}})) begin
                overflow <= 1;
            end
            data_out <= product[DATA_WIDTH-1:0]; // Truncate to the original data width
        end
    end
end
```

در غیر این صورت با توجه با opcode کارهای مختلف را انجام می‌دهیم در ورودی opcode با آن حالت بندی می‌کنیم .

و به طور پیش‌فرض overflow را صفر می‌گذاریم .

توجه: sp: پایین‌ترین خانه حافظه که خالی است

حالت بندی روی opcode

opcode = 110 push

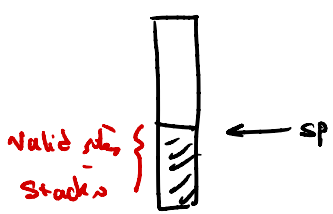
```
if (opcode == 3'b110) begin // Push
    if (!full) begin
        stack_mem[sp] <= data_in;
        sp <= sp + 1;
    end
```

(I) در صورتی که Stack پر نباشد با توجه به تعریف sp، مقدار data_in را در خانه‌ای که sp (II) را به آن اشاره می‌کند می‌ریزیم sp را یک واحد زیاد می‌کنیم به خانه بالاتر و همچنین اشاره می‌کند (III)

opcode = 111 pop

```
    end
end else if (opcode == 3'b111) begin // Pop
    if (!empty) begin
        data_out <= stack_mem[sp-1];
        sp <= sp - 1;
    end
```

(I) در صورتی که حافظه خالی نباشد با توجه به تعریف sp، مقداری در حافظه که sp به آن اشاره می‌کند در بالاترین خانه با داده valid حافظه (II) درون آن داده یک واحد از sp کم می‌کنیم (III)



opcode 101 : multiplier

```
    end
end else if (opcode == 3'b101) begin // Multiply
    if (sp > 1) begin
        // Perform multiplication and check for overflow
        product = stack_mem[sp-2] * stack_mem[sp-1];   (I)
        // Check for overflow by comparing to max and min values of the original data width
        if (product > $signed({1'b0, {(DATA_WIDTH-1){1'b1}}}) ||   (i)
    (II)    product < $signed({1'b1, {(DATA_WIDTH-1){1'b0}}})) begin   (ii)
            overflow <= 1;
        end
        data_out <= product[DATA_WIDTH-1:0]; // Truncate to the original data width   (III)
    end
end
```

sp>1 ⇐ حداقل دو خانه valid درون stack وجود دارد که mem[sp-2], mem[sp-1]

(I) بالاترین دو خانه هستند و آن ها را در هم ضرب کرده و نتیجه را در product ریخته می شود.

با شرط بالا overflow را بررسی می کنیم

کوچکترین عدد منفی با Data width بیت

(i) product > 0 11....1 (Data width-1)   (ii) product < 1 00...0 (Data-width-1)   با شرط 2 comp

بزرگترین عدد مثبت با Data width (N) بیت

(II) که اگر یکی از دو شرط صدق کند یعنی جزو اعداد نمایشی با N بیت نیست و overflow هست.

(III) N بیت سمت راست را به عنوان خروجی می دهیم.

opcode: 100

```verilog
    end
end else if (opcode == 3'b100) begin // Add
    if (sp > 1) begin
        // Perform addition and check for overflow
        sum = stack_mem[sp-2] + stack_mem[sp-1];
        // Check for overflow
        if ((stack_mem[sp-2] > 0 && stack_mem[sp-1] > 0 && sum < 0) ||
            (stack_mem[sp-2] < 0 && stack_mem[sp-1] < 0 && sum > 0)) begin
            overflow <= 1;
        end
        data_out <= sum;
    end
```

کاملاً مشابه حالت قبل است فقط دو خانه بالا stack را باهم جمع می‌زنیم و شرط overflow هم

این است جمع دو عدد هم علامت، حاصلی با علامت متضاد داشته باشد.

توجه کنید که تمامی حالات 1XX یک opcode با معنا است و برای همه‌ی حالات کاری می‌کنیم و

چون opcode ۳ بیتی است بقیه حالات 0XX خود به خود no op خواهند بود.

• خود عملیات ها پرینت میشه و stack در هر مرحله نشان با pop کردن تمام اعضایش را نمایش می دهم.

حاصل تست کیس های مختلف که انواع overflow و نبود آن را بررسی کرده ایم.

• تمام کارها را عددی واقعی کل چاپ کرده ام.

N = ۴

```
VSIM 5> run -all
# Pushing data    3 onto the stack
# Pushing data    5 onto the stack
# Addition result:          3 +          5 =  -8, Overflow: 1
# Pushing data    7 onto the stack
# Pushing data   -8 onto the stack
# Addition (without overflow) result:         7 +          8 =  -1
# Popping data  -8 from the stack after addition
# Pushing data  -7 onto the stack
# Pushing data  -8 onto the stack
# Addition (overflow) result:          -7 +         -8 =   1, Overflow: 1
# Popping data  -8 from the stack after addition
# Pushing data   3 onto the stack
# Pushing data   3 onto the stack
# Multiplication result:          3 *          3 =  -7, Overflow: 1
# Pushing data   7 onto the stack
# Pushing data   2 onto the stack
# Multiplication (overflow) result:          7 *          2 =  -2, Overflow: 1
# Pushing data  -3 onto the stack
# Pushing data  -4 onto the stack
# Multiplication result:         -3 *         -4 =  -4, Overflow: 1
# Pushing data   3 onto the stack
# Pushing data  -4 onto the stack
# Multiplication result:          3 *         -4 =   4, Overflow: 1
# Popping data  -4 from the stack after multiplication
# Pushing data   1 onto the stack
# Pushing data   2 onto the stack
# Addition (without overflow) result:         1 +          2 =   3
# Popping data   2 from the stack after addition
# Pushing data   2 onto the stack
```

```
# Pushing data  -4 onto the stack
# Multiplication result:         -3 *         -4 =  -4, Overflow: 1
# Pushing data   3 onto the stack
# Pushing data  -4 onto the stack
# Multiplication result:          3 *         -4 =   4, Overflow: 1
# Popping data  -4 from the stack after multiplication
# Pushing data   1 onto the stack
# Pushing data   2 onto the stack
# Addition (without overflow) result:         1 +          2 =   3
# Popping data   2 from the stack after addition
# Pushing data   2 onto the stack
# Pushing data   3 onto the stack
# Multiplication (without overflow) result:         2 *          3 =   6
# Popping data   3 from the stack after multiplication
# Popping data   2 from the stack
# Final stack contents:
# stack_mem[0] = 3
# stack_mem[1] = 5
# stack_mem[2] = 7
# stack_mem[3] = -7
# stack_mem[4] = 3
# stack_mem[5] = 3
# stack_mem[6] = 7
# stack_mem[7] = 2
# stack_mem[8] = -3
# stack_mem[9] = -4
# stack_mem[10] = 3
# stack_mem[11] = 1
# ** Note: $stop    : C:/Users/Abolfazl/Desktop/MQ1-signed/tb_4bit.v(259)
#    Time: 350 ns  Iteration: 0  Instance: /tb_4bit
```

N = ۸

```
VSIM 13> run -all
# Pushing data    3 onto the stack
# Pushing data    5 onto the stack
# Addition (without overflow) result:          3 +          5 =    8
# Pushing data  127 onto the stack
# Pushing data    2 onto the stack
# Addition (overflow) result:        127 +          2 = -127, Overflow: 1
# Popping data    2 from the stack after addition
# Pushing data -128 onto the stack
# Pushing data   -1 onto the stack
# Addition (overflow) result:       -128 +         -1 =  127, Overflow: 1
# Popping data   -1 from the stack after addition
# Pushing data   10 onto the stack
# Pushing data   12 onto the stack
# Multiplication (without overflow) result:         10 *         12 =  120
# Pushing data   64 onto the stack
# Pushing data    3 onto the stack
# Multiplication (overflow) result:         64 *          3 =  -64, Overflow: 1
# Pushing data   15 onto the stack
# Pushing data   15 onto the stack
# Multiplication (overflow) result:         15 *         15 =  -31, Overflow: 1
# Pushing data  -20 onto the stack
# Pushing data   -5 onto the stack
# Multiplication (without overflow) result:        -20 *         -5 =  100
# Pushing data    7 onto the stack
# Pushing data   -8 onto the stack
# Multiplication (without overflow) result:          7 *         -8 =  -56
# Popping data   -8 from the stack after multiplication
# Pushing data    1 onto the stack
# Pushing data    2 onto the stack
```

```
# Multiplication (overflow) result:         15 *         15 =  -31, Overflow: 1
# Pushing data  -20 onto the stack
# Pushing data   -5 onto the stack
# Multiplication (without overflow) result:        -20 *         -5 =  100
# Pushing data    7 onto the stack
# Pushing data   -8 onto the stack
# Multiplication (without overflow) result:          7 *         -8 =  -56
# Popping data   -8 from the stack after multiplication
# Pushing data    1 onto the stack
# Pushing data    2 onto the stack
# Addition (without overflow) result:          1 +          2 =    3
# Popping data    2 from the stack after addition
# Final stack contents:
# stack_mem[0] = 3
# stack_mem[1] = 5
# stack_mem[2] = 127
# stack_mem[3] = -128
# stack_mem[4] = 10
# stack_mem[5] = 12
# stack_mem[6] = 64
# stack_mem[7] = 3
# stack_mem[8] = 15
# stack_mem[9] = 15
# stack_mem[10] = -20
# stack_mem[11] = -5
# stack_mem[12] = 7
# stack_mem[13] = 1
# ** Note: $stop    : C:/Users/Abolfazl/Desktop/MQ1-signed/tb_8bit.v(245)
#    Time: 330 ns  Iteration: 0  Instance: /tb_8bit
```

N = ۱۶

```
VSIM 27> run -all
# Pushing data      3 onto the stack
# Pushing data      5 onto the stack
# Addition (without overflow) result:           3 +           5 =       8
# Pushing data  32767 onto the stack
# Pushing data      2 onto the stack
# Addition (overflow) result:       32767 +           2 = -32767, Overflow: 1
# Popping data      2 from the stack after addition
# Pushing data -32768 onto the stack
# Pushing data     -1 onto the stack
# Addition (overflow) result:      -32768 +          -1 =  32767, Overflow: 1
# Popping data     -1 from the stack after addition
# Pushing data     10 onto the stack
# Pushing data     12 onto the stack
# Multiplication (without overflow) result:          10 *          12 =    120
# Pushing data    256 onto the stack
# Pushing data    128 onto the stack
# Multiplication (overflow) result:         256 *         128 = -32768, Overflow: 1
# Pushing data    300 onto the stack
# Pushing data    200 onto the stack
# Multiplication (overflow) result:         300 *         200 =  -5536, Overflow: 1
# Pushing data    -20 onto the stack
# Pushing data     -5 onto the stack
# Multiplication (without overflow) result:         -20 *          -5 =    100
# Pushing data      7 onto the stack
# Pushing data     -8 onto the stack
# Multiplication (without overflow) result:          7 *          -8 =    -56
# Popping data     -8 from the stack after multiplication
# Pushing data      1 onto the stack
# Pushing data      2 onto the stack
```

```
# Pushing data    -20 onto the stack
# Pushing data     -5 onto the stack
# Multiplication (without overflow) result:         -20 *          -5 =    100
# Pushing data      7 onto the stack
# Pushing data     -8 onto the stack
# Multiplication (without overflow) result:          7 *          -8 =    -56
# Popping data     -8 from the stack after multiplication
# Pushing data      1 onto the stack
# Pushing data      2 onto the stack
# Addition (without overflow) result:           1 +           2 =       3
# Popping data      2 from the stack after addition
# Final stack contents:
# stack_mem[0] = 3
# stack_mem[1] = 5
# stack_mem[2] = 32767
# stack_mem[3] = -32768
# stack_mem[4] = 10
# stack_mem[5] = 12
# stack_mem[6] = 256
# stack_mem[7] = 128
# stack_mem[8] = 300
# stack_mem[9] = 200
# stack_mem[10] = -20
# stack_mem[11] = -5
# stack_mem[12] = 7
# stack_mem[13] = 1
# ** Note: $stop    : C:/Users/Abolfazl/Desktop/MQ1-signed/tb_16bit.v(243)
#    Time: 330 ps  Iteration: 0  Instance: /tb_16bit
```

N = ۳۲

```
# Pushing data           3 onto the stack
# Pushing data           5 onto the stack
# Addition (without overflow) result:           3 +           5 =           8
# Pushing data  2147483647 onto the stack
# Pushing data           2 onto the stack
# Addition (overflow) result:  2147483647 +           2 = -2147483647, Overflow: 1
# Popping data           2 from the stack after addition
# Pushing data -2147483648 onto the stack
# Pushing data          -1 onto the stack
# Addition (overflow) result: -2147483648 +          -1 =  2147483647, Overflow: 1
# Popping data          -1 from the stack after addition
# Pushing data          10 onto the stack
# Pushing data          12 onto the stack
# Multiplication (without overflow) result:          10 *          12 =         120
# Pushing data       65536 onto the stack
# Pushing data       65536 onto the stack
# Multiplication (overflow) result:       65536 *       65536 =           0, Overflow: 1
# Pushing data      300000 onto the stack
# Pushing data      200000 onto the stack
# Multiplication (overflow) result:      300000 *      200000 =  -129542144, Overflow: 1
# Pushing data      -20000 onto the stack
# Pushing data       -5000 onto the stack
# Multiplication (without overflow) result:      -20000 *       -5000 =   100000000
# Pushing data           7 onto the stack
# Pushing data          -8 onto the stack
# Multiplication (without overflow) result:          7 *          -8 =         -56
# Popping data          -8 from the stack after multiplication
# Pushing data           1 onto the stack
# Pushing data           2 onto the stack
# Addition (without overflow) result:           1 +           2 =           3
```

```
# Multiplication (overflow) result:      300000 *      200000 =  -129542144, Overflow: 1
# Pushing data      -20000 onto the stack
# Pushing data       -5000 onto the stack
# Multiplication (without overflow) result:      -20000 *       -5000 =   100000000
# Pushing data           7 onto the stack
# Pushing data          -8 onto the stack
# Multiplication (without overflow) result:          7 *          -8 =         -56
# Popping data          -8 from the stack after multiplication
# Pushing data           1 onto the stack
# Pushing data           2 onto the stack
# Addition (without overflow) result:           1 +           2 =           3
# Popping data           2 from the stack after addition
# Final stack contents:
# stack_mem[0] = 3
# stack_mem[1] = 5
# stack_mem[2] = 2147483647
# stack_mem[3] = -2147483648
# stack_mem[4] = 10
# stack_mem[5] = 12
# stack_mem[6] = 65536
# stack_mem[7] = 65536
# stack_mem[8] = 300000
# stack_mem[9] = 200000
# stack_mem[10] = -20000
# stack_mem[11] = -5000
# stack_mem[12] = 7
# stack_mem[13] = 1
# ** Note: $stop    : C:/Users/Abolfazl/Desktop/MQ1-signed/tb_32bit.v(243)
#    Time: 330 ps  Iteration: 0  Instance: /tb_32bit
```