به نام خدا

طراحی سیستم‌های دیجیتال

گزارش کار پروژه‌ی امتیازی

دانشکده مهندسی کامپیوتر

دانشگاه صنعتی شریف

بهار 1403

نویسندگان:

- ✓ **آرین نوری 401106663**
- ✓ **کسری عزیززاده 401106222**

(سوال انتخاب شده برای پروژه‌ی امتیازی: سوال اول میان‌ترم)

بخش اول: ساخت ماژول اصلی و تست آن

در این سوال می‌خواهیم اعداد را درون یک STACK ذخیره کنیم و خروجی‌های جمع و ضرب را در دو خانه‌ای که دیرتر از بقیه پر می‌شوند را هر بار بتوانیم محاسبه کنیم، همچنین باید بتوانیم این دو عملیات به همراه PUSH و POP را با یک Opcode کنترل کنیم، از طرفی باید برنامه‌ی ما قادر به تشخیص خروجی سرریز یا همان overflow نیز باشد، به صورت ورودی و خروجی‌های اصلی ما به این شکل خواهند بود:

Opcode: 3 – bit / INPUT / 0XX: no action, 100: add, 101: multiply, 110: push, 111 pop
Input_data: N – bit / INPUT
Output_data: N – bit / OUTPUT
Overflow: 1 – bit / OUTPUT
Clock and Reset: 1 – bit / OUTPUT

توجه کنید که خروجی عملیات‌های جمع و ضرب تغییری در محتوای پشته نخواهد داد.

همچنین اندازه‌ی اعداد ورودی و خروجی را باید پارامتر گذاشته و اندازه‌ی خود Stack را به طور پیشفرض برابر 16 قرار می‌دهیم، برای تغییر اندازه کافیست در ابتدای ماژول STACK_BASED_ALU آن را تغییر دهید.

حالا به بررسی کد می‌پردازیم،

```
module STACK_BASED_ALU
                #(parameter n = 32 , parameter size = 16)
                (input clock, input reset, input [2:0] opcode,
                input signed [n - 1: 0] input_data,
                output reg signed [n - 1: 0] output_data,
                output reg overflow);

            integer pointer;
            reg signed [n-1:0] stack [size - 1:0];

            always @(posedge clock) begin
                if (reset) begin
                    pointer = 0;
                    output_data = 0;
                    overflow = 0;
```

همان‌طور که گفته شد اندازه‌ی اعداد و اندازه‌ی استک را به عنوان پارامتر تعریف می‌کنیم، متغیر pointer همان stack pointer ما است که به طور پیشفرض با استکی به طول 16 کار می‌کند، ابتدای لبه‌ی مثبت ساعت بیت reset را بررسی می‌کنیم،

```
if(opcode[2] != 0) begin
    case (opcode[1:0])
        2'b00: begin // Opcode: 100 => Addition
            overflow = 0;
            if (pointer >= 2) begin
                output_data = stack[pointer - 1] + stack[pointer - 2];
                // check if the sign has changed:
                if ((stack[pointer - 1] > 0 && stack[pointer - 2] > 0 && output_data < 0) ||
                (stack[pointer - 1] < 0 && stack[pointer - 2] < 0 && output_data > 0))
                    overflow = 1;
            end
        end
        2'b01: begin // Opcode: 101 => Multiply
            overflow = 0;
            if (pointer >= 2) begin
                output_data = stack[pointer - 1] * stack[pointer - 2];
                // check if the sign has changed:
                if (output_data / stack[pointer - 2] != stack[pointer - 1])
                overflow = 1;
            end
```

سپس در صورت فعال بودن بیت سوم آپکد، دستورهای جمع و ضرب را مطابق عکس صفحه‌ی قبل انجام می‌دهیم، منطق بررسی بیت overflow در جمع به این صورت است که در صورت تغییر علامت در خروجی، overflow را تشخیص می‌دهیم. برای ضرب نیز اگر overflow رخ دهد دیگر نباید حاصل تقسیم خروجی بر یکی از ورودی‌ها، ورودی دیگر شود، پس در این حالت نیز به درستی تشخیص می‌دهیم.

```
                    2'b10: begin // Opcode: 110 => Push
                        if (pointer < size) begin
                            stack[pointer] = input_data;
                            pointer = pointer + 1;
                        end
                    end
                    2'b11: begin // Opcode: 111 => Pop
                        if (pointer > 0) begin
                            output_data = stack[pointer - 1];
                            stack[pointer - 1] = {n{1'b0}};
                            pointer = pointer - 1;
                        end
                    end
                endcase
            end
        end
    end

endmodule
```

برای push بررسی می‌کنیم که اگر استک پر نباشد مقدار جدید را وارد کنیم و برای pop بررسی می‌کنیم اگر استک مقداری دارد آن را خروجی دهیم و به جای آن صفر می‌گذاریم.

حالا برای چهار مقدار n شامل 4 و 8 و 16 و 32، testbench می‌نویسیم.

N = 4

```
clock = 0;
reset = 1; #10;
reset = 0; #10
//PUSH TEST:
input_data = 4'b0011; opcode = 3'b110;
#10;
$display("pushing 3");     opcode = 3'b000;
#10
//PUSH TEST:
input_data = 4'b0100; opcode = 3'b110;
#10;
$display("pushing 4");     opcode = 3'b000;
#10;
//ADD TEST:
opcode = 3'b100;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
//MULTIPLY TEST:
opcode = 3'b101;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
//PUSH TEST:
input_data = 4'b0110; opcode = 3'b110;
#10;
$display("pushing 6");     opcode = 3'b000;
//PUSH TEST:
input_data = 4'b0010; opcode = 3'b110;
#10;
$display("pushing 2");     opcode = 3'b000;
opcode = 3'b101;
#10;
//POP TEST:
opcode =  3'b111;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
#10;
$stop;
```

```
pushing 3
pushing 4
result:   7, overflow: 0
result:  -4, overflow: 1
pushing 6
pushing 2
result:   2, overflow: 0
```

در این آزمایش دو عدد 3 و 4 را ابتدا وارد کرده سپس بدون مشکل جمع می‌کنیم، نمایش عدد 7 نشان می‌دهد overflow اتفاق نمی‌افتد، در ضرب 3 و 4 که حاصل 12 به ما می‌دهد overflow اتفاق می‌افتد و عددی که نمایش داده می‌شود منفی4 است، سپس دو مقدار 6 و 2 را وارد استک کردیم و با pop کردن آخرین مقدار یعنی 2 را خروجی می‌گیریم. پس در این بخش در تمامی قسمت‌ها استک ما درست کار کرد.

N = 8

```
clock = 0;
reset = 1; #10;
reset = 0; #10
//PUSH TEST:
input_data = 20;    opcode = 3'b110;
#10;
$display("pushing 20");    opcode = 3'b000;
#10
//PUSH TEST:
input_data = 40;    opcode = 3'b110;
#10;
$display("pushing 40");    opcode = 3'b000;
#10;
//ADD TEST:
opcode = 3'b100;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
//MULTIPLY TEST:
opcode = 3'b101;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
//PUSH TEST:
input_data = 100;  opcode = 3'b110;
#10;
$display("pushing 100"); opcode = 3'b000;
//PUSH TEST:
input_data = 120;  opcode = 3'b110;
#10;
$display("pushing 120"); opcode = 3'b000;
opcode = 3'b101;
#10;
//POP TEST:
opcode =  3'b111;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
#10;
$stop;
```

```
pushing 20
pushing 40
result:   60, overflow: 0
result:   32, overflow: 1
pushing 100
pushing 120
result:  120, overflow: 0
```

ابتدا دو مقدار 20 و 60 وارد استک شده سپس این دو را باهم جمع می‌کنیم که بدون مشکل اتفاق می‌افتد، ولی موقع ضرب این دو چون که عدد 800 را نمی‌توان با 8 بیت نمایش داد پس خروجی ما overflow کرده، در ادامه دو عدد 100 و 120 را به استک می‌دهیم و وقتی از آن pop می‌کنیم، آخرین مقدار یعنی 120 به ما خروجی داده می‌شود.

N = 16

```
initial begin
    clock = 0;
    reset = 1; #10;
    reset = 0; #10
    //PUSH TEST:
    input_data = 1000;      opcode = 3'b110;
    #10;
    $display("pushing 1000");     opcode = 3'b000;
    #10
    //PUSH TEST:
    input_data = 3500;      opcode = 3'b110;
    #10;
    $display("pushing 3500");     opcode = 3'b000;
    #10;
    //ADD TEST:
    opcode = 3'b100;
    #10;
    $display("result: %d, overflow: %d" , output_data , overflow);
    //MULTIPLY TEST:
    opcode = 3'b101;
    #10;
    $display("result: %d, overflow: %d" , output_data , overflow);
    //PUSH TEST:
    input_data = 2222;      opcode = 3'b110;
    #10;
    $display("pushing 2222");     opcode = 3'b000;
    //PUSH TEST:
    input_data = 3333;      opcode = 3'b110;
    #10;
    $display("pushing 3333");     opcode = 3'b000;
    opcode = 3'b101;
    #10;
    //POP TEST:
    opcode =  3'b111;
    #10;
    $display("result: %d, overflow: %d" , output_data , overflow);
    #10;
    $stop;
end
```

```
pushing 1000
pushing 3500
result:   4500, overflow: 0
result:  26592, overflow: 1
pushing 2222
pushing 3333
result:   3333, overflow: 0
```

ابتدا دو عدد 1000 و 3500 را وارد استک کرده، سپس این دو را باهم جمع می‌کنیم که واضحاً در 16 بیت قابل نمایش است و بیت overflow فعال نمی‌شود، در ادامه با ضرب این دو، به علت این که حاصل ضرب در 16 بیت قابل نمایش نیست، overflow رخ می‌دهد و جواب غلط می‌شود، سپس دو مقدار 2222 و 3333 که به راحتی در 16 بیت قابل نمایش هستند را وارد استک کرده و نشان می‌دهیم که هنگام pop آخرین عدد وارد شده که همان 3333 است از استک خارج می‌شود.

N = 32

```
clock = 0;
reset = 1; #10;
reset = 0; #10
//PUSH TEST:
input_data = 100000;opcode = 3'b110;
#10;
$display("pushing 100000");  opcode = 3'b000;
#10
//PUSH TEST:
input_data = 12345;   opcode = 3'b110;
#10;
$display("pushing 12345");    opcode = 3'b000;
#10;
//ADD TEST:
opcode = 3'b100;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
//MULTIPLY TEST:
opcode = 3'b101;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
//PUSH TEST:
input_data = 1000000;     opcode = 3'b110;
#10;
$display("pushing 1000000");      opcode = 3'b000;
//PUSH TEST:
input_data = 3000000;     opcode = 3'b110;
#10;
$display("pushing 3000000");      opcode = 3'b000;
opcode = 3'b101;
#10
//MULTIPLY TEST:
opcode = 3'b101;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
#10;
//POP TEST:
opcode =  3'b111;
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
#10;
$display("result: %d, overflow: %d" , output_data , overflow);
$stop;
```

```
pushing 100000
pushing 12345
result:      112345, overflow: 0
result:  1234500000, overflow: 0
pushing 1000000
pushing 3000000
result:  2112827392, overflow: 1
result:     3000000, overflow: 0
result:     1000000, overflow: 0
```

ابتدا دو عدد 100000 و 12345 را وارد استک کرده و جمع و ضرب را با آن‌ها انجام می‌دهیم، هر دو عملیات بدون اشکال و سرریز کردن انجام می‌شوند، حالا دو عدد بزرگتر مثل 1000000 و 3000000 را وارد پشته می‌کنیم، ضرب این دو عدد را نمی‌توانیم با 32 بیت نمایش دهیم برای همین overflow اتفاق می‌افتد، در آخر نیز این دو مقدار را به درستی از استک pop می‌کنیم.

پس مدار STACK_BASED_ALU را در هر چهار حالت بررسی کردیم و از صحت این مدار اطمینان پیدا کردیم.