بسم الله الرحمن الرحیم

سوال ۱)

کد این سوال در فایل Q1_A در فایل‌های ارسالی قابل مشاهده است. توضیح کد این سوال:

```
module STACK_BASED_ALU#(parameter n = 4)(input clk, input signed [n - 1:0]input_data, input [2:0]opcode, output reg overflow, output reg signed[n - 1:0]output_data);
  reg[n - 1:0] stack[511:0];
  integer pc = -1, i = 0;
  reg signed [2 * n - 1:0] temp_output;
  always @(posedge clk) begin
    overflow = 0;
    case(opcode)
        3'b100: begin
            if(pc >= 1) begin
                output_data = stack[pc] + stack[pc - 1];
                if((stack[pc][n - 1] == 0 && stack[pc - 1][n - 1] == 0 && output_data[n - 1] == 1) ||
                (stack[pc][n - 1] == 1 && stack[pc - 1][n - 1] == 1 && output_data[n - 1] == 0)) overflow = 1;
            end
        end
        3'b101: begin
            if(pc >= 1) begin
                temp_output = stack[pc] * stack[pc - 1];
                output_data = temp_output;
                if((stack[pc][n - 1] == 0 && stack[pc - 1][n - 1] == 0 && output_data[n - 1] == 1)) overflow = 1;
                for(i = n; i < 2 * n; i = i + 1) begin
                    if(temp_output[i] == 1) overflow <= 1;
                end
            end
        end
        3'b110: begin
            if(pc < 511)begin
                pc = pc + 1;
                stack[pc] <= input_data;
            end
        end
        3'b111: begin
            if(pc > 0)begin
                output_data <= stack[pc];
                pc = pc - 1;
            end
        end
    endcase
    end
endmodule
```

در ابتدا که پارامترهای گفته شده در سوال به عنوان ورودی خروجی ماژول تعریف شده‌اند. ابعاد input_data و output_data متناسب با n است که در ابتدا توسط پارامتر ذکر شده است و در instance گیری قابل تغییر است.

سپس یک استک به طول دیتاها و عمق ۵۱۲ تعریف می‌کنیم. در خط بعد سه متغیر pc و i و temp_out تعریف می‌شوند که در ادامه کاربرد آن‌ها را توضیح خواهم داد.

بلاک always با لبه مثبت کلاک فعال می‌شود. در این بلاک در ابتدا مقدار overflow صفر می‌شود که اگر از دفعه قبل یک مانده بود، صفر شود. سپس با توجه به opcode ورودی با فرمت case بررسی می‌کنیم که باید چه عملیاتی روی استک و ... انجام گیرد:

(قبل از توضیح هر case، pc همان اشاره‌گر به اولین خانه استک است. یعنی اگر ما یک عدد را در استک اضافه می‌کنیم، pc به همان خانه اخر اشاره میکند (همان program counter))

add-100:

ابتدا چک می‌شود که ایا حداقل دو مقدار درون استک داریم که آن دو را باهم جمع کنیم یا نه. برای این کار چک می‌کنیم که pc بزرگتر از 1 باشد.(یعنی حداقل در خانه 0 و 1 آن مقداری موجود باشد). اگر این شرط برقرار بود دو خانه بالایی استک را با هم جمع کرده و درون data_out قرار می‌دهیم.

حال باید overflow را چک کنیم. در شرطی که برای یک کردن overflow آمده است بیان می‌شود که اگر جمع دو عدد مثبت منفی شد **یا** جمع دو عدد منفی مثبت شد، overflow یک بشود.

mult-101:

برای ضرب نیز ابتدا همان شرط وجود حداقل دو مقدار درون استک را مانند بخش قبل چک میکنیم. سپس ضرب دو مقدار روی استک را درون temp_out قرار می‌دهیم که اندازه‌اش دو برابر ورودی هاست و این بدین معناست که در آن overflow رخ نمی‌دهد.(اساس این کار برای این است که بتوانیم تشخیص دهیم ایا در واقع overflow رخ داده است یا خیر، که در ادامه توضیح می دهم)

مقدار temp_out را درون output_data قرار می‌دهیم. ممکن است مقداری از مقادیر سرریز شوند که این امر را در خطوط بعدی چک می کنیم. ابتدا به طور جدا چک می‌شود که ضرب دو عدد مثبت منفی نشود. ولی چون ضرب است ممکن است بیت های سرریز صرفا باعث تغییر مثبت و منفی شدن عدد نشوند. برای همین یک فور میزنیم که در آن چک می‌شود ایا متغیر temp_out از خانه n به بعد، مقداری برابر با 1 دارد یا خیر؟ اگر داشته باشد که یعنی سرریز رخ داده است و output_data ما نتوانسته آن مقدار را نشان دهد و اگر هم هیچ خانه‌ای 1 نباشد یعنی overflow رخ نداده‌است.

push-110:

ابتدا با شرط این که pc کوچکتر از ۵۱۱ باشد چک می‌شود که استک پر نباشد. اگر پر نبود، اشاره‌گر pc را یک واحد افزایش داده و سپس مقدار درون input_data را درون بالاترین نقطه استک یعنی همان جایی که pc به آن اشاره دارد قرار می‌دهیم.

:pop-111

ابتدا با شرط این که pc بزرگتر از ۰ باشد چک می کنیم که استک خالی نباشد. و اگر خالی نبود، مقداری که در بالاترین خانه استک قرار دارد را(یعنی همان محلی که pc به آن اشاره می کند) در output_data قرار می‌دهیم و سپس مقدار pc را یک واحد کاهش می‌دهیم.

شرح کد اصلی به تفصیل بیان شد. اما بخش testbench که در بخش الف از ما خواسته‌شده است:

کد testbench در فایلی با نام TB_Q1_A در فایل‌های ارسالی قابل مشاهده است. توضیح آن به شرح زیر است:

```
module TB_Q1_A();
    reg clk;
    wire overflow4;
    reg [2:0] opcode4;
    reg [3:0] input_data4;
    wire [3:0] output_data4;
    wire overflow8;
    reg [2:0] opcode8;
    reg [7:0] input_data8;
    wire [7:0] output_data8;
    wire overflow16;
    reg [2:0] opcode16;
    reg [15:0] input_data16;
    wire [15:0] output_data16;
    wire overflow32;
    reg [2:0] opcode32;
    reg [31:0] input_data32;
    wire [31:0] output_data32;
    STACK_BASED_ALU #(4) sba1(clk, input_data4, opcode4, overflow4, output_data4);
    STACK_BASED_ALU #(8) sba2(clk, input_data8, opcode8, overflow8, output_data8);
    STACK_BASED_ALU #(16) sba3(clk, input_data16, opcode16, overflow16, output_data16);
    STACK_BASED_ALU #(32) sba4(clk, input_data32, opcode32, overflow32, output_data32);

  always
    #1 clk = ~clk;
  initial
     clk <= 0;
```

ابتدا برای هر کدام از بخش های 4و 8 و 16 و 32 بیتی، متغیر های overflow و input_data و output_data را مخصوص آنها تعریف می کنیم.(متغیر clk یا همان کلاک عمومی است و نیاز به تعریف اختصاری ندارد). همانطور که قابل مشاهده هست ابعاد های متغیر ها متناسبا متفاوت است.

سپس با ۴ بار از ماژول اصلی instance می‌گیریم که به هرکدام متغیرهای مخصوص خودش را می‌دهیم. بطور مثال به instance ای که برای 16 بیتی ها گرفته شده است، ورودی، خروجی های مخصوص 16 بیتی را می‌دهیم.

با استفاده از یک always یک خطی کلاک را هریک واحد زمانی یک‌بار not می کنیم تا واقعا تبدیل به clock شود!

و در بخش بعد نیز مقدار اولیه آن را صفر قرار می‌دهیم.

در ادامه ما حالات مختلف را برای حالت های 4 و 8 و 16 و 32 مورد بررسی قرار داده‌ایم. در ادامه به توضیح بخش 4 بیتی میپردازم و بخش های دیگر نیز به همین شیوه بررسی شده اند و دیگر نیاز به توضیح مجدد آنها نیست و صرفا تصاویر کد و نتیجه شبیه‌سازی آن را در این فایل قرار می‌دهم.

توضیح بخش ۴ بیتی:

```
initial begin
  $display ("for 4 bit: ");
  $monitor("time:%d | clk=%b | opcode=%b | input_data=%b | output_data=%b | overflow4=%b", $time, clk, opcode4, input_data4, output_data4, overflow4);
  opcode4 = 3'b110;
  input_data4 = 4'b1100;                                    //push 1100
  #2 opcode4 = 3'b110;
  input_data4 = 4'b0001;                                    //push 0001
  #2 opcode4 = 3'b100;                                      //add 1100, 0001
  #2 opcode4 = 3'b101;                                      //mult 1100, 0001
  #2 opcode4 = 3'b111;                                      //pop 0001
  #2 opcode4 = 3'b110;
  input_data4 = 4'b1110;                                    //push 1110
  #2 opcode4 = 3'b100;                                      //add 1100, 1110(overflow)
  #2 opcode4 = 3'b111;                                      //pop 1110
  #2 opcode4 = 3'b110;
  input_data4 = 4'b010;                                     //push 0010
  #2 opcode4 = 3'b101;                                      //mult 1100 , 0010(overflow)
  #2 opcode4 = 3'b111;                                      //pop 0010
  #2 opcode4 = 3'b111;                                      //pop 1100
  #2

```

نتیجه شبیه‌سازی:

```
# for 4 bit:
# time:                   0 | clk=0 | opcode=110 | input_data=1100 | output_data=xxxx | overflow4=x
# time:                   1 | clk=1 | opcode=110 | input_data=1100 | output_data=xxxx | overflow4=0
# time:                   2 | clk=0 | opcode=110 | input_data=0001 | output_data=xxxx | overflow4=0
# time:                   3 | clk=1 | opcode=110 | input_data=0001 | output_data=xxxx | overflow4=0
# time:                   4 | clk=0 | opcode=100 | input_data=0001 | output_data=xxxx | overflow4=0
# time:                   5 | clk=1 | opcode=100 | input_data=0001 | output_data=1101 | overflow4=0
# time:                   6 | clk=0 | opcode=101 | input_data=0001 | output_data=1101 | overflow4=0
# time:                   7 | clk=1 | opcode=101 | input_data=0001 | output_data=1100 | overflow4=0
# time:                   8 | clk=0 | opcode=111 | input_data=0001 | output_data=1100 | overflow4=0
# time:                   9 | clk=1 | opcode=111 | input_data=0001 | output_data=0001 | overflow4=0
# time:                  10 | clk=0 | opcode=110 | input_data=1000 | output_data=0001 | overflow4=0
# time:                  11 | clk=1 | opcode=110 | input_data=1000 | output_data=0001 | overflow4=0
# time:                  12 | clk=0 | opcode=100 | input_data=1000 | output_data=0001 | overflow4=0
# time:                  13 | clk=1 | opcode=100 | input_data=1000 | output_data=0100 | overflow4=1
# time:                  14 | clk=0 | opcode=111 | input_data=1000 | output_data=0100 | overflow4=1
# time:                  15 | clk=1 | opcode=111 | input_data=1000 | output_data=1000 | overflow4=0
# time:                  16 | clk=0 | opcode=110 | input_data=0010 | output_data=1000 | overflow4=0
# time:                  17 | clk=1 | opcode=110 | input_data=0010 | output_data=1000 | overflow4=0
# time:                  18 | clk=0 | opcode=101 | input_data=0010 | output_data=1000 | overflow4=0
# time:                  19 | clk=1 | opcode=101 | input_data=0010 | output_data=1000 | overflow4=1
# time:                  20 | clk=0 | opcode=111 | input_data=0010 | output_data=1000 | overflow4=1
# time:                  21 | clk=1 | opcode=111 | input_data=0010 | output_data=0010 | overflow4=0
# time:                  22 | clk=0 | opcode=111 | input_data=0010 | output_data=0010 | overflow4=0
# time:                  23 | clk=1 | opcode=111 | input_data=0010 | output_data=0010 | overflow4=0
```

در خط اول که یک مانیتور هست که به ازای تغییر هر کدام از مقادیر نام برده در آن ، یک دور چاپ می‌شود.

ابتدا opcode را روی حالت push قرار داده و input_data را نیز برابر 1100 قرار میدهیم. پس از یک واحد زمانی این تغیرات اعمال می‌شود. حال ما یک ثانیه دیگر نیز صبر می کنیم تا کلاک در سطح پایین قرار گیرد. در مجموع دو واحد زمانی باید بین دستورات وقفه انداخت. پس از دو واحد زمانی وقفه با همان شیوه قبل عدد 0001 را push

می‌کنیم. سپس opcode را روی حالت add قرار می‌دهیم تا نتیجه جمع در output_data قابل مشاهده باشد. سپس opcode را روی حالت ضرب قرار میدهیم.(این دو حالت بدون تست overflow بودند)

حالا مقدار روی استک را یعنی مقدار 0001 را با قرار دادن opcode روی حالت pop، برمیداریم که یعنی این مقدار در آن لحظه در output_data قابل مشاهده است. سپس با تنظیم opcode و input_data عدد 1000 را به داخل استک میفرستیم. در مرحله بعد این دو را با هم جمع کرده که با وجود مشاهده نتیجه در output_data، بیت overflow نیز یک می‌شود. سپس عدد 1000 را خارج کرده و عدد 0010 را push می‌کنیم. حال opcode را روی حالت ضرب قرار می‌دهیم تا دو عدد بالایی استک را یعنی 1100 و 0010 را با هم ضرب کند. این ضرب نیز دارای سرریز است و در نتیجه بیت overflow یک میشود. در انتها نیز دو مقدار درون استک را با دستو pop خارج می‌کنیم.

این توضیح برای بخش ۴ بیتی بود. برای بخش ۸ و ۱۶ و ۳۲ نیز به همین صورت است. اما دیگر نیاز به توضیح نیست و صرفا تصاویر کد و نتیجه آن را در این گزارش می‌اورم:

```
52      $display ("for 8 bit: ");
53      $monitor("time:%d | clk=%b | opcode=%d | input_data=%b | output_data=%b | overflow4=%b", $time, clk, opcode8, input_data8, output_data8, overflow8);
54      opcode8 = 3'b110;
55      input_data8 = 8'b01110000;                                  //push 01110000
56      #2 opcode8 = 3'b110;
57      input_data8 = 8'b00000010;                                  //push 00000010
58      #2 opcode8 = 3'b100;                                        //add 01110000, 00000010
59      #2 opcode8 = 3'b101;                                        //mult 01110000, 00000010
60      #2 opcode8 = 3'b111;                                        //pop 00000010
61      #2 opcode8 = 3'b110;
62      input_data8 = 8'b01110100;                                  //push 01110100
63      #2 opcode8 = 3'b100;                                        //add 01110000, 01110100(overflow)
64      #2 opcode8 = 3'b111;                                        //pop 01110100
65      #2 opcode8 = 3'b110;
66      input_data8 = 8'b00001000;                                  //push 00001000
67      #2 opcode8 = 3'b101;                                        //mult 01110000 , 00001000(overflow)
68      #2 opcode8 = 3'b111;                                        //pop 00001000
69      #2 opcode8 = 3'b111;                                        //pop 01110000
70      #2
```

```
# for 8 bit:
# time:                24 | clk=0 | opcode=6 | input_data=01110000 | output_data=xxxxxxxx | overflow4=0
# time:                25 | clk=1 | opcode=6 | input_data=01110000 | output_data=xxxxxxxx | overflow4=0
# time:                26 | clk=0 | opcode=6 | input_data=00000010 | output_data=xxxxxxxx | overflow4=0
# time:                27 | clk=1 | opcode=6 | input_data=00000010 | output_data=xxxxxxxx | overflow4=0
# time:                28 | clk=0 | opcode=4 | input_data=00000010 | output_data=xxxxxxxx | overflow4=0
# time:                29 | clk=1 | opcode=4 | input_data=00000010 | output_data=01110010 | overflow4=0
# time:                30 | clk=0 | opcode=5 | input_data=00000010 | output_data=01110010 | overflow4=0
# time:                31 | clk=1 | opcode=5 | input_data=00000010 | output_data=11100000 | overflow4=1
# time:                32 | clk=0 | opcode=7 | input_data=00000010 | output_data=11100000 | overflow4=1
# time:                33 | clk=1 | opcode=7 | input_data=00000010 | output_data=00000010 | overflow4=0
# time:                34 | clk=0 | opcode=6 | input_data=01110100 | output_data=00000010 | overflow4=0
# time:                35 | clk=1 | opcode=6 | input_data=01110100 | output_data=00000010 | overflow4=0
# time:                36 | clk=0 | opcode=4 | input_data=01110100 | output_data=00000010 | overflow4=0
# time:                37 | clk=1 | opcode=4 | input_data=01110100 | output_data=11100100 | overflow4=1
# time:                38 | clk=0 | opcode=7 | input_data=01110100 | output_data=11100100 | overflow4=1
# time:                39 | clk=1 | opcode=7 | input_data=01110100 | output_data=01110100 | overflow4=0
# time:                40 | clk=0 | opcode=6 | input_data=00001000 | output_data=01110100 | overflow4=0
# time:                41 | clk=1 | opcode=6 | input_data=00001000 | output_data=01110100 | overflow4=0
# time:                42 | clk=0 | opcode=5 | input_data=00001000 | output_data=01110100 | overflow4=0
# time:                43 | clk=1 | opcode=5 | input_data=00001000 | output_data=10000000 | overflow4=1
# time:                44 | clk=0 | opcode=7 | input_data=00001000 | output_data=10000000 | overflow4=1
# time:                45 | clk=1 | opcode=7 | input_data=00001000 | output_data=00001000 | overflow4=0
# time:                46 | clk=0 | opcode=7 | input_data=00001000 | output_data=00001000 | overflow4=0
# time:                47 | clk=1 | opcode=7 | input_data=00001000 | output_data=00001000 | overflow4=0
```

:9

```
    $display ("for 16 bit: ");
    $monitor("time:%d | clk=%b | opcode=%d | input_data=%b | output_data=%b | overflow4=%b", $time, clk, opcode16, input_data16, output_data16, overflow16);
    opcode16 = 3'b110;
    input_data16 = 16'b0111000000000001;                                    //push 0111000000000001
    #2 opcode16 = 3'b110;
    input_data16 = 16'b0000010000000001;                                    //push 0000010000000000
    #2 opcode16 = 3'b100;                                                   //add 0111000000000001, 0000010000000000
    #2 opcode16 = 3'b101;                                                   //mult 0111000000000001, 0000010000000000(overflow)
    #2 opcode16 = 3'b111;                                                   //pop 0000010000000000
    #2 opcode16 = 3'b110;
    input_data16 = 16'b0111010000000010;                                    //push 0111010000000010
    #2 opcode16 = 3'b100;                                                   //add 0111000000000001, 0111010000000010(overflow)
    #2 opcode16 = 3'b111;                                                   //pop 0111010000000010
    #2 opcode16 = 3'b110;
    input_data16 = 16'b0000000000000001;                                    //push 0000000000000001
    #2 opcode16 = 3'b101;                                                   //mult 0111000000000001 , 0000000000000001
    #2 opcode16 = 3'b111;                                                   //pop 0000000000000001
    #2 opcode16 = 3'b111;                                                   //pop 0011000000000001
    #2
```

```
# for 16 bit:
# time:                48 | clk=0 | opcode=6 | input_data=0111000000000001 | output_data=xxxxxxxxxxxxxxxx | overflow4=0
# time:                49 | clk=1 | opcode=6 | input_data=0111000000000001 | output_data=xxxxxxxxxxxxxxxx | overflow4=0
# time:                50 | clk=0 | opcode=6 | input_data=0000010000000001 | output_data=xxxxxxxxxxxxxxxx | overflow4=0
# time:                51 | clk=1 | opcode=6 | input_data=0000010000000001 | output_data=xxxxxxxxxxxxxxxx | overflow4=0
# time:                52 | clk=0 | opcode=4 | input_data=0000010000000001 | output_data=xxxxxxxxxxxxxxxx | overflow4=0
# time:                53 | clk=1 | opcode=4 | input_data=0000010000000001 | output_data=0111010000000010 | overflow4=0
# time:                54 | clk=0 | opcode=5 | input_data=0000010000000001 | output_data=0111010000000010 | overflow4=0
# time:                55 | clk=1 | opcode=5 | input_data=0000010000000001 | output_data=0111010000000001 | overflow4=1
# time:                56 | clk=0 | opcode=7 | input_data=0000010000000001 | output_data=0111010000000001 | overflow4=1
# time:                57 | clk=1 | opcode=7 | input_data=0000010000000001 | output_data=0000010000000001 | overflow4=0
# time:                58 | clk=0 | opcode=6 | input_data=0111010000000010 | output_data=0000010000000001 | overflow4=0
# time:                59 | clk=1 | opcode=6 | input_data=0111010000000010 | output_data=0000010000000001 | overflow4=0
# time:                60 | clk=0 | opcode=4 | input_data=0111010000000010 | output_data=0000010000000001 | overflow4=0
# time:                61 | clk=1 | opcode=4 | input_data=0111010000000010 | output_data=1110010000000011 | overflow4=1
# time:                62 | clk=0 | opcode=7 | input_data=0111010000000010 | output_data=1110010000000011 | overflow4=1
# time:                63 | clk=1 | opcode=7 | input_data=0111010000000010 | output_data=0111010000000010 | overflow4=0
# time:                64 | clk=0 | opcode=6 | input_data=0000000000000001 | output_data=0111010000000010 | overflow4=0
# time:                65 | clk=1 | opcode=6 | input_data=0000000000000001 | output_data=0111010000000010 | overflow4=0
# time:                66 | clk=0 | opcode=5 | input_data=0000000000000001 | output_data=0111010000000010 | overflow4=0
# time:                67 | clk=1 | opcode=5 | input_data=0000000000000001 | output_data=0111000000000001 | overflow4=0
# time:                68 | clk=0 | opcode=7 | input_data=0000000000000001 | output_data=0111000000000001 | overflow4=0
# time:                69 | clk=1 | opcode=7 | input_data=0000000000000001 | output_data=0000000000000001 | overflow4=0
# time:                70 | clk=0 | opcode=7 | input_data=0000000000000001 | output_data=0000000000000001 | overflow4=0
# time:                71 | clk=1 | opcode=7 | input_data=0000000000000001 | output_data=0000000000000001 | overflow4=0
```

:9

```
    $display ("for 32 bit: ");
    $monitor("time:%d | clk=%b | opcode=%d | input_data=%b | output_data=%b | overflow4=%b", $time, clk, opcode32, input_data32, output_data32, overflow32);
    opcode32 = 3'b110;
    input_data32 = 32'b00000000000000000111000000000000;                    //push 00000000000000000111000000000000
    #2 opcode32 = 3'b110;
    input_data32 = 32'b00000000000000000000000000000010;                    //push 00000000000000000000000000000010
    #2 opcode32 = 3'b100;                                                   //add 00000000000000000111000000000000, b00000000000000000000000000000010
    #2 opcode32 = 3'b101;                                                   //mult 00000000000000000111000000000000, b00000000000000000000000000000010
    #2 opcode32 = 3'b111;                                                   //pop b00000000000000000000000000000010
    #2 opcode32 = 3'b110;
    input_data32 = 32'b11111111111111111111111111001111;                    //push 11111111111111111111111111001111
    #2 opcode32 = 3'b100;                                                   //add 00000000000000000111000000000000, 11111111111111111111111111001111(overflow)
    #2 opcode32 = 3'b111;                                                   //pop 11110100011111111111111111111111111110011110000010
    #2 opcode32 = 3'b110;
    input_data32 = 32'b00000000000000010000000000000000;                    //push 00000000000000010000000000000000
    #2 opcode32 = 3'b101;                                                   //mult 00000000000000000111000000000000 , 00000000000000010000000000000000
    #2 opcode32 = 3'b111;                                                   //pop 00000000000000010000000000000000
    #2 opcode32 = 3'b111;                                                   //pop 00000000000000000111000000000000
    #2


    $stop();
  end


endmodule
```

```
# for 32 bit:
# time:                72 | clk=0 | opcode=6 | input_data=00000000000000000111000000000000 | output_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | overflow4=0
# time:                73 | clk=1 | opcode=6 | input_data=00000000000000000111000000000000 | output_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | overflow4=0
# time:                74 | clk=0 | opcode=6 | input_data=00000000000000000000000000000010 | output_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | overflow4=0
# time:                75 | clk=1 | opcode=6 | input_data=00000000000000000000000000000010 | output_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | overflow4=0
# time:                76 | clk=0 | opcode=4 | input_data=00000000000000000000000000000010 | output_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx | overflow4=0
# time:                77 | clk=1 | opcode=4 | input_data=00000000000000000000000000000010 | output_data=00000000000000000111000000000010 | overflow4=0
# time:                78 | clk=0 | opcode=5 | input_data=00000000000000000000000000000010 | output_data=00000000000000000111000000000010 | overflow4=0
# time:                79 | clk=1 | opcode=5 | input_data=00000000000000000000000000000010 | output_data=00000000000000000111000000000000 | overflow4=0
# time:                80 | clk=0 | opcode=7 | input_data=00000000000000000000000000000010 | output_data=00000000000000000111000000000000 | overflow4=0
# time:                81 | clk=1 | opcode=7 | input_data=00000000000000000000000000000010 | output_data=00000000000000000000000000000010 | overflow4=0
# time:                82 | clk=0 | opcode=6 | input_data=11111111111111111111111111001111 | output_data=00000000000000000000000000000010 | overflow4=0
# time:                83 | clk=1 | opcode=6 | input_data=11111111111111111111111111001111 | output_data=00000000000000000000000000000010 | overflow4=0
# time:                84 | clk=0 | opcode=4 | input_data=11111111111111111111111111001111 | output_data=00000000000000000000000000000010 | overflow4=0
# time:                85 | clk=1 | opcode=4 | input_data=11111111111111111111111111001111 | output_data=00000000000000001101111111001111 | overflow4=0
# time:                86 | clk=0 | opcode=7 | input_data=11111111111111111111111111001111 | output_data=00000000000000001101111111001111 | overflow4=0
# time:                87 | clk=1 | opcode=7 | input_data=11111111111111111111111111001111 | output_data=11111111111111111111111111001111 | overflow4=0
# time:                88 | clk=0 | opcode=6 | input_data=00000000000000010000000000000000 | output_data=11111111111111111111111111001111 | overflow4=0
# time:                89 | clk=1 | opcode=6 | input_data=00000000000000010000000000000000 | output_data=11111111111111111111111111001111 | overflow4=0
# time:                90 | clk=0 | opcode=5 | input_data=00000000000000010000000000000000 | output_data=11111111111111111111111111001111 | overflow4=0
# time:                91 | clk=1 | opcode=5 | input_data=00000000000000010000000000000000 | output_data=11000000000000000000000000000000 | overflow4=1
# time:                92 | clk=0 | opcode=7 | input_data=00000000000000010000000000000000 | output_data=11000000000000000000000000000000 | overflow4=1
# time:                93 | clk=1 | opcode=7 | input_data=00000000000000010000000000000000 | output_data=00000000000000010000000000000000 | overflow4=0
# time:                94 | clk=0 | opcode=7 | input_data=00000000000000010000000000000000 | output_data=00000000000000010000000000000000 | overflow4=0
# time:                95 | clk=1 | opcode=7 | input_data=00000000000000010000000000000000 | output_data=00000000000000010000000000000000 | overflow4=0
```

کد ها و نتیجه آن ها بخش به بخش نمایش داده شده است. در صورت نیاز می‌توانید بخش های مختلف کد را شبیه‌سازی کنید و نتیجه آن را مشاهده کنید.

با تشکر از لطف شما

احسان محترم – 401106458