بسمه تعالی

آزمایش شماره ۸

استاد

دکتر علیرضا اجلالی

سینا مظاهری

متین داغیانی

دانشگاه صنعتی شریف

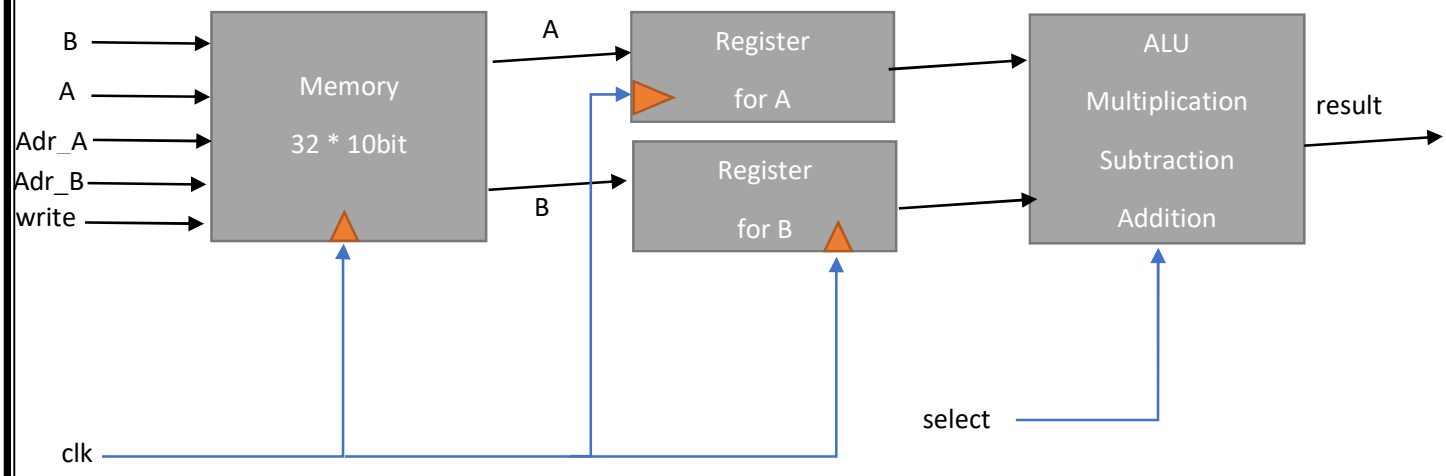
۱۴۰۰-۱۳۹۹

مقدمه و اهداف آزمایش

هدف از این آزمایش طراحی یک کامپیوتر اعداد مختلط می باشد که شامل یک حافظه ی ۳۲ کلمه ای و یک واحد پایپ لاین است. این کامپیوتر قادر است محاسبات جمع و تفریق و ضرب را برای اعداد مختلط مختلف محاسبه و حاصل را نمایش دهد.

تئوری آزمایش

معماری کلی این کامپیوتر بدین صورت است که شامل یک حافظه ۳۲ کلمه ای می باشد که با دادن آدرس هر دو عملوند **A** و **B** و همچنین فعال کردن سیگنال **write**، مادامی که لبه مثبت پالس ساعت می آید، آن ها را در آدرس مورد نظر بنویسد. همچنین فارغ از زمانی که سیگنال **write** فعال یا غیر فعال باشد، مقدار آدرس ورودی در لبه مثبت، برروی ثبات های آدرس نظیر **A** و **B** قرار می گیرد تا از این پس مقدار خروجی حافظه، مقدار خانه ی حافظه به ازای ثبات های آدرس باشد. به منظور رعایت معماری پایپ لاین یک ثبات حائل میان واحد محاسبه و منطق **(ALU)** و حافظه **(memory)** قرار می گیرد. با انتخابگر **select** نیز نوع عملیات را انتخاب می کنیم.



همانطور که می دانیم یک عدد مختلط به صورت $a + bi$ نمایش داده می شود که در آن **a , b** به ترتیب قسمت های موهومی و حقیقی می باشند. حال فرض می کنیم که هر دو عدد **a , b** در متمم مبنای ۲ به شکل ۵ بیتی باشند حال این دو عدد ۵ بیتی را به هم چسبانده و یک بردار ۱۰ بیتی تشکیل می دهیم که همان نمایش عدد

مختلط ما می باشد.(۵ بیت کم ارزش نمایش قسمت موهومی و ۵ بیت باارزش تر نمایش قسمت حقیقی می باشد) بنابراین دو عدد A , B ۱۰ بیتی هستند. از آن جایی که حافظه ۳۲ کلمه ای می باشد، ۵ بیت برای آدرس هر کدام از مقادیر A , B در نظر گرفته می شود. در قسمت خروجی نیز یک بردار ۲۲ بیتی خواهیم داشت زیرا عملی که بیشترین تعداد بیت را تولید می کند، عمل ضرب خواهد بود که سرریز در آن رخ داده باشد. در نتیجه دو بردار ۱۱ بیتی شامل قسمت های حقیقی و موهومی خواهند بود که از چسباندن آن ها بردار ۲۲ بیتی نتیجه حاصل می شود. در شکل بالا عمل ریست به منظور جلوگیری از پیچیدگی شکل نیامده اما در مدار لحاظ شده همچنین دو بیت سرریز در خروجی تولید می شود که در شکل نیامده است. هر یک از عملیات های جمع و تفریق و ضرب به شکل زیر در ALU محاسبه می شود. این امر برروی کد سخت افزاری به شکل دقیق پیاده سازی شده است.

فرض کنیم دو عدد مختلط a + bi و c + di را به ما داده باشند در اینصورت:

$$\text{حاصلضرب دو عدد} = (ac - bd) + (ad + bc)i$$

$$\text{حاصل جمع دو عدد} = (a + c) + (b + d)i$$

$$\text{حاصل تفریق دو عدد} = (a - c) + (b + d)i \quad .$$

کد پیاده سازی سخت افزاری

کد توصیف حافظه ی ۳۲ کلمه ای:

```
module memory(
    input wire clk, write, reset,
    input wire [9:0] data_in_A,
    input wire [9:0] data in B,
    input wire [4:0] address_A,
    input wire [4:0] address_B,
    output wire [9:0] data out A,
    output wire [9:0] data_out_B
    );
    reg [9:0] data_registers [0:31];  // MBR
    reg [4:0] address_register_A; // MAR
    reg [4:0] address_register_B;
    integer i;
    always @(posedge clk or negedge reset)
        begin
            if (!reset)
                begin
                    for (i = 0; i < 32; i = i + 1)
                        data_registers[i] <= 0;
                    address_register_A <= 0;
                    address_register_B <= 0;
                end
            else
                begin
                    if (write)
                        begin
                            data_registers[address_A] <= data_in_A;
                            data_registers[address_B] <= data_in_B;
                        end
                    address_register_A <= address_A;
                    address_register_B <= address_B;
                end
        end
        assign data_out_A[9:0] = data_registers[address_register_A][9:0];
        assign data_out_B[9:0] = data_registers[address_register_B][9:0];

endmodule
```

کد توصیف ثبات ها :

```
module register(
    input wire [9:0] data_in_A,
    input wire [9:0] data_in_B,
    input wire clk,
    input wire reset,
    output wire [9:0] data_out_A,
    output wire [9:0] data_out_B
    );

    reg [9:0] register A, register B;
    assign data_out_A = register_A;
    assign data_out_B = register_B;

    always @(posedge clk or negedge reset)
        begin
            if (!reset)
                begin
                    register A <= 0;
                    register_B <= 0;
                end
            else
                begin
                    register_A <= data_in_A;
                    register_B <= data_in_B;
                end
end
```

کد ALU:

```
module alu(
    input wire  [9:0] op A,
    input wire  [9:0] op_B,
    input wire [1:0] select,
    output reg  [21:0] result,
    output wire overflow_imaginary,
    output wire overflow_real
    );

    wire [9:0] result_adder_subtractor;
    wire [1:0] carry_out_result_adder, carry_out_result_multiplier;
    wire [19:0] result multiplier;
    wire overflow_adder_sub_imaginary, overflow_adder_sub_real,
overflow multiply imaginary, overflow_multiply_real;
    wire sign_bit_low, sign_bit_high;
    adder_subtractor addsub0(.op_A(op_A), .op_B(op_B), .select(select[0]),
.result(result_adder_subtractor), .carry_out(carry_out_result_adder),
.overflow_imaginary(overflow_adder_sub_imaginary),
.overflow real(overflow adder sub real));
    multiplier mul0(.op_A(op_A), .op_B(op_B), .result(result_multiplier),
.overflow_imaginary(overflow_multiply_imaginary),
.overflow_real(overflow_multiply_real)
,.carry_out(carry_out_result_multiplier));
```

```
    assign sign_bit_low = result_adder_subtractor[4];
    assign sign_bit_high = result_adder_subtractor[9];
    assign overflow real = (select[1]) ?
overflow_multiply_real:overflow_adder_sub_real;
    assign overflow imaginary = (select[1]) ?
overflow_multiply_imaginary:overflow_adder_sub_imaginary;
     always @(*)
        begin
            result = 21'b0;
            case(select[1])
                1'b0:
                    begin
                        if (!overflow_adder_sub_imaginary &&
!overflow_adder_sub_real)
                            begin
                                {result[15:11], result[4:0]} =
{result_adder_subtractor[9:5], result_adder_subtractor[4:0]};
                                {result[21:16], result[10:5]} =
{sign_bit_high, sign_bit_high, sign_bit_high, sign_bit_high, sign_bit_high,
sign bit high, sign bit low, sign_bit_low, sign_bit_low, sign_bit_low,
sign_bit_low, sign_bit_low};
                            end
                        else if (overflow_adder_sub_imaginary &&
!overflow_adder_sub_real)
                             begin
                                 {result[15:11], result[4:0]} =
{result adder subtractor[9:5], result adder subtractor[4:0]};
                                  {result[21:16], result[10:5]} =
{sign bit high, sign bit high, sign bit high, sign bit high, sign bit_high,
sign_bit_high, carry_out_result_adder[0], carry_out_result_adder[0],
carry_out_result_adder[0], carry_out_result_adder[0],
carry out result adder[0], carry_out_result_adder[0]};
                             end
                        else if (!overflow_adder_sub_imaginary &&
overflow_adder_sub_real)
                             begin
                                {result[15:11], result[4:0]} =
{result_adder_subtractor[9:5], result_adder_subtractor[4:0]};
                                {result[21:16], result[10:5]} =
{carry_out_result_adder[1], carry_out_result_adder[1],
carry_out_result_adder[1], carry_out_result_adder[1],
carry_out_result_adder[1], carry_out_result_adder[1], sign_bit_low,
sign_bit_low, sign_bit_low, sign_bit_low, sign_bit_low, sign_bit_low};
                             end
                        else
                            begin
                               {result[15:11], result[4:0]} =
{result_adder_subtractor[9:5], result_adder_subtractor[4:0]};
                               {result[21:16], result[10:5]} =
{carry_out_result_adder[1], carry_out_result_adder[1],
carry out result adder[1], carry out result adder[1],
carry_out_result_adder[1], carry_out_result_adder[1],
carry_out_result_adder[0],  carry_out_result_adder[0],
carry_out_result_adder[0],  carry_out_result_adder[0],
carry_out_result_adder[0],  carry_out_result_adder[0]};
                            end
                    end
```

```
                1'b1:
                    begin
                        if (!overflow_multiply_imaginary &&
!overflow_multiply_real)
                            begin
                                {result[20:11], result[9:0]} =
{result_multiplier[19:10], result_multiplier[9:0]};
                                {result[21], result[10]} =
{result_multiplier[19], result_multiplier[9]};
                            end
                        else if (overflow_multiply_imaginary &&
!overflow_multiply_real)
                             begin
                                {result[20:11], result[9:0]} =
{result multiplier[19:10], result multiplier[9:0]};
                                {result[21], result[10]} =
{result_multiplier[19], carry_out_result_multiplier[0]};
                             end
                        else if (!overflow_multiply_imaginary &&
overflow multiply real)
                             begin
                               {result[20:11], result[9:0]} =
{result_multiplier[19:10], result_multiplier[9:0]};
                                {result[21], result[10]} =
{carry_out_result_multiplier[1], result_multiplier[9]};
                             end
                        else
                            begin
                              {result[20:11], result[9:0]} =
{result_multiplier[19:10], result_multiplier[9:0]};
                                {result[21], result[10]} =
{carry out result multiplier[1], carry_out_result_multiplier[0]};
                            end
                    end
                default: result = 21'bz;
            endcase
        end

endmodule
```

کد جمع کننده و تفریق کننده:

```
module adder_subtractor(
    input wire [9:0] op_A,
    input wire [9:0] op_B,
    input wire select,
    output wire [9:0] result,
    output wire [1:0] carry out,
    output wire overflow_imaginary,
    output wire overflow_real
    );
    wire [4:0] complement_B_imaginary,complement_B_real, final_real,
final_imaginary;
```

```
    assign complement_B_imaginary = ~op_B[4:0] + 1'b1;
    assign complement_B_real = ~op_B[9:5] + 1'b1;
    assign final real = (select) ? complement B real:op B[9:5];
    assign final_imaginary = (select) ? complement_B_imaginary:op_B[4:0];
    assign result[4:0] = (select) ? op_A[4:0] + complement_B_imaginary :
op_A[4:0] + op_B[4:0];
    assign result[9:5] = (select) ? op_A[9:5] + complement_B_real : op_A[9:5]
+ op B[9:5];
    assign overflow_imaginary = result[4] & ~op_A[4] & ~final_imaginary[4] |
~result[4] & op A[4] & final imaginary[4];
    assign overflow_real = result[9] & ~op_A[9] & ~final_real[4] | ~result[9]
& op_A[9] & final_real[4];
    assign carry_out[0] = (op_A[4] & final_imaginary[4]) | ((op_A[3] &
final_imaginary[3]) | ((op_A[2] & final_imaginary[2]) | ((op_A[1] &
final imaginary[1]) | (op A[0] & final imaginary[0]) & (op A[1] ^
final_imaginary[1])) & (op_A[2] ^ final_imaginary[2])) & (op_A[3] ^
final_imaginary[3])) & (op_A[4] ^ final_imaginary[4]);
    assign carry_out[1] = (op_A[9] & final_real[4]) | ((op_A[8] &
final_real[3]) | ((op_A[7] & final_real[2]) | ((op_A[6] & final_real[1]) |
(op A[5] & final real[0]) & (op A[6] ^ final real[1])) & (op A[7] ^
final_real[2])) & (op_A[8] ^ final_real[3])) & (op_A[9] ^ final_real[4]);
endmodule
```

کد ضرب کننده:

```
module multiplier(
    input wire [9:0] op_A,
    input wire [9:0] op B,
    output wire [1:0] carry_out,
    output wire [19:0] result,
    output wire overflow_imaginary,
    output wire overflow_real
    );

    wire signed [9:0] temp_imaginary_1, temp_imaginary_2, temp_real_1,
temp_real_2, temp_sub;
    assign temp imaginary 1 = $signed(op A[9:5]) * $signed(op B[4:0]);
    assign temp_imaginary_2 = $signed(op_A[4:0]) * $signed(op_B[9:5]);
    assign temp_real_1 = $signed(op_A[9:5]) * $signed(op_B[9:5]);
    assign temp real 2 = $signed(op A[4:0]) * $signed(op B[4:0]);
    assign  result[9:0] = temp_imaginary_1 + temp_imaginary_2;
    assign  result[19:10] = temp_real_1 + temp_sub;
    assign overflow_imaginary = result[9] & ~temp_imaginary_1[9] &
~temp_imaginary_2[9] | ~result[9] & temp_imaginary_1[9] &
temp_imaginary_2[9];
    assign overflow_real = result[19] & ~temp_real_1[9] & ~temp_sub[9] |
~result[19] & temp real 1[9] & temp sub[9];
    assign temp_sub = ~temp_real_2 + 1'b1;
    assign carry out[0] = (temp imaginary 1[9] & temp imaginary 2[9]) |
((temp_imaginary_1[8] & temp_imaginary_2[8]) | ((temp_imaginary_1[7] &
temp imaginary 2[7]) | ((temp imaginary 1[6] & temp imaginary 2[6]) |
((temp imaginary 1[5] & temp imaginary 2[5]) | ((temp imaginary 1[4] &
temp_imaginary_2[4]) | ((temp_imaginary_1[3] & temp_imaginary_2[3]) |
((temp_imaginary_1[2] & temp_imaginary_2[2]) | ((temp_imaginary_1[1] &
```

```
temp_imaginary_2[1]) | (temp_imaginary_1[0] & temp_imaginary_2[0]) &
(temp_imaginary_1[1] ^ temp_imaginary_2[1])) & (temp_imaginary_1[2] ^
temp imaginary 2[2])) & (temp imaginary 1[3] ^ temp imaginary 2[3])) &
(temp_imaginary_1[4] ^ temp_imaginary_2[4])) & (temp_imaginary_1[5] ^
temp imaginary 2[5])) & (temp imaginary 1[6] ^ temp imaginary 2[6])) &
(temp_imaginary_1[7] ^ temp_imaginary_2[7])) & (temp_imaginary_1[8] ^
temp_imaginary_2[8])) & (temp_imaginary_1[9] ^ temp_imaginary_2[9]);
    assign carry out[1] = (temp real 1[9] & temp sub[9]) | ((temp real_1[8] &
temp_sub[8]) | ((temp_real_1[7] & temp_sub[7]) | ((temp_real_1[6] &
temp sub[6]) | ((temp real 1[5] & temp sub[5]) | ((temp real 1[4] &
temp_sub[4]) | ((temp_real_1[3] & temp_sub[3]) | ((temp_real_1[2] &
temp_sub[2]) | ((temp_real_1[1] & temp_sub[1]) | (temp_real_1[0] &
temp_sub[0]) & (temp_real_1[1] ^ temp_sub[1])) & (temp_real_1[2] ^
temp_sub[2])) & (temp_real_1[3] ^ temp_sub[3])) & (temp_real_1[4] ^
temp sub[4])) & (temp real 1[5] ^ temp sub[5])) & (temp real 1[6] ^
temp_sub[6])) & (temp_real_1[7] ^ temp_sub[7])) & (temp_real_1[8] ^
temp_sub[8])) & (temp_real_1[9] ^ temp_sub[9]);
endmodule
```

کد توصیف کامپیوتر:

```
module complexnumber_computer(
    input wire clk,
    input wire reset,
    input wire write,
    input wire [1:0] select,
    input wire [4:0] address_A,
    input wire [4:0] address_B,
    input wire signed [9:0] op_A,
    input wire signed [9:0] op B,
    output wire signed [21:0] result,
    output wire overflow_imaginary,
    output wire overflow_real
    );

    wire [9:0] data_out_A, data_out_B, data_out_reg_A, data_out_reg_B;

    memory mem0(.address_A(address_A), .address_B(address_B),
.data in A(op A), .data in B(op B), .data out A(data out A),
.data out B(data out B), .clk(clk), .reset(reset), .write(write));
    register register0(.clk(clk), .reset(reset), .data_in_A(data_out_A),
.data in B(data out B), .data_out_A(data_out_reg_A),
.data_out_B(data_out_reg_B));
    alu alu0(.op A(data out reg A), .op B(data out reg B), .select(select),
.result(result), .overflow imaginary(overflow_imaginary),
.overflow_real(overflow_real));

endmodule
```

شبیه سازی آزمایشگاهی:

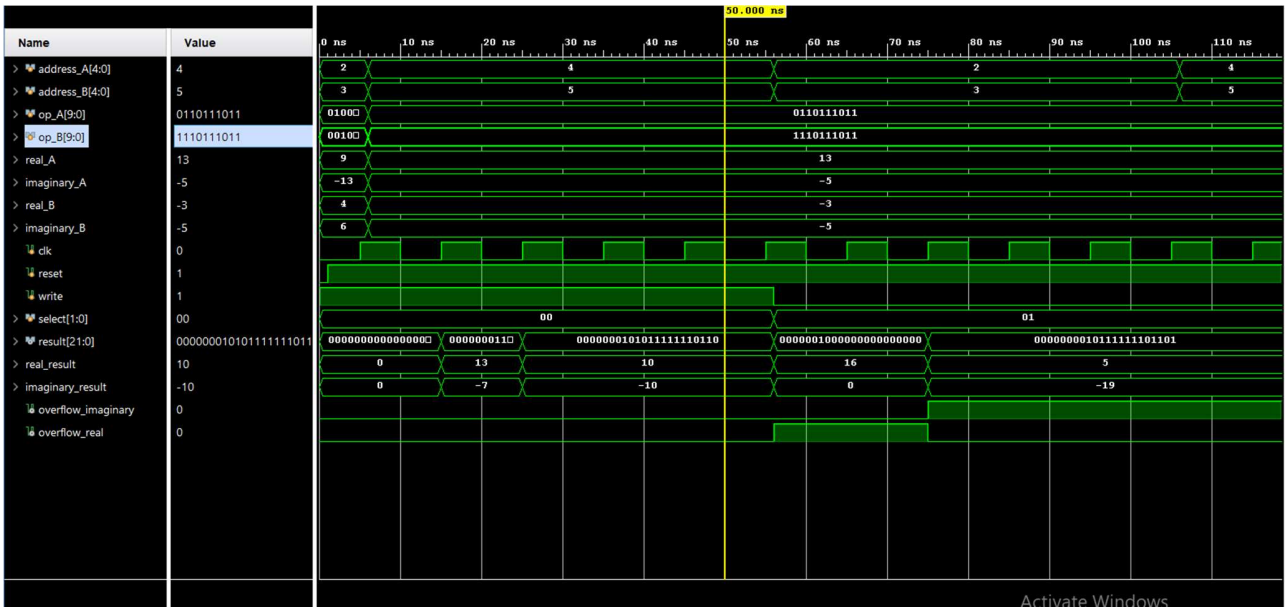
به منظور شبیه سازی یک فایل شبیه سازی که در پیوست این گزارش آمده نوشته شده تا تمامی عملیات های مورد نظر برروی دو عدد 13i – 9 و 6i + 4 و نیز برروی دو عدد 5i – 13 و 3-5i- شبیه سازی شود.

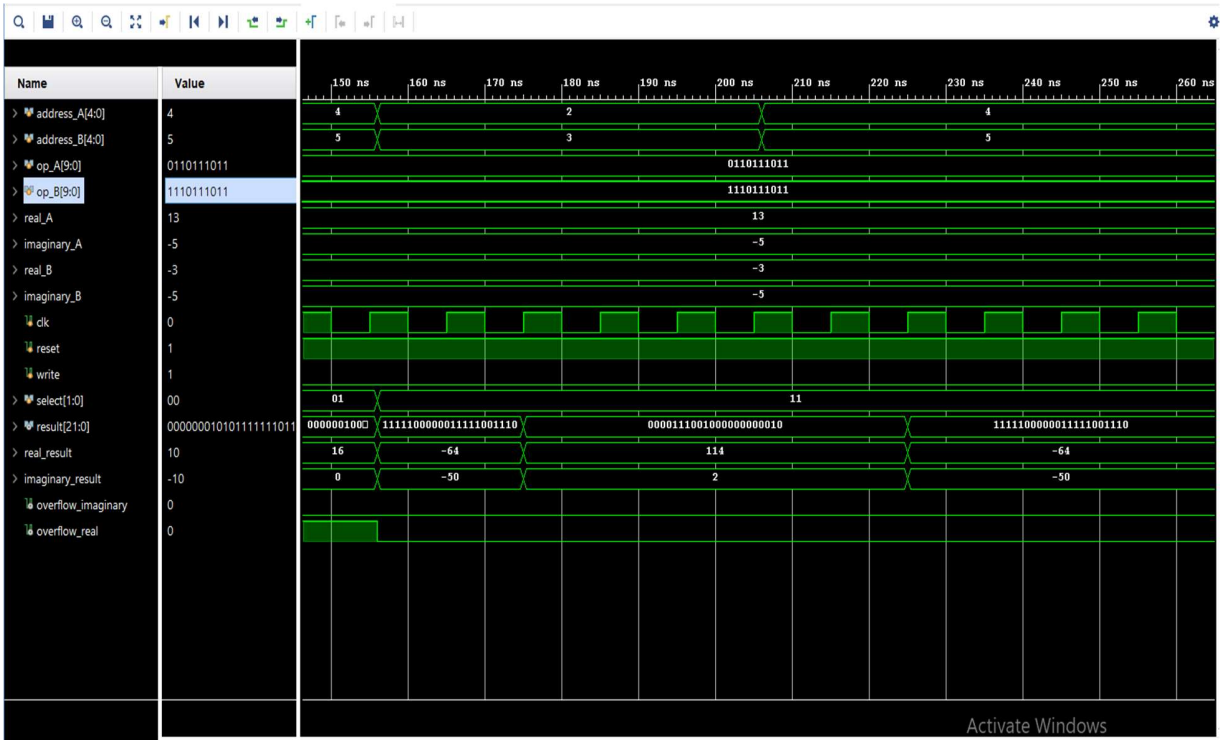
کد شبیه سازی:

```
module tb(

    );

    reg [9:0] op A, op B;
    reg clk, reset, write;
    reg [1:0] select;
    reg [4:0] address_A, address_B;
    wire [21:0] result;
    wire overflow_imaginary, overflow_real;

    complexnumber computer comp0(.clk(clk), .write(write), .reset(reset),
.select(select), .op_A(op_A), .op_B(op_B), .address_A(address_A),
.address_B(address_B), .result(result),
.overflow_imaginary(overflow_imaginary), .overflow_real(overflow_real));

    initial
        begin
            reset = 1'b0;
            clk = 1'b0;
            forever #5 clk = ~clk;
        end



    initial
        begin
               write = 1'b1;
               select = 2'b00;
               address_A = 4'b0010;
               address_B = 4'b0011;
               op_A = 10'b01001_10011; // 9 - 13i
               op_B = 10'b00100_00110; // 4 + 6i
               #1; // t = 1ns
               reset = 1'b1;
               #5; // t = 6ns
               address_A = 4'b0100;
               address_B = 4'b0101;
               op_A = 10'b01101_11011; // 13 - 5i
               op_B = 10'b11101_11011; // -3 -5i
               #50; // t = 16ns
               write = 1'b0;
               select = 2'b01;
               address_A = 4'b0010;
               address_B = 4'b0011;
```

```
            #50;
            address_A = 4'b0100;
            address_B = 4'b0101;
            #50;
            select = 2'b11;
            address_A = 4'b0010;
            address_B = 4'b0011;
            #50;
            address_A = 4'b0100;
            address_B = 4'b0101;
end
```

نتایج شکل موج:




Name
Value
address_A[4:0]
4
address_B[4:0]
5
op_A[9:0]
0110111011
op_B[9:0]
1110111011
real_A
13
imaginary_A
-5
real_B
-3
imaginary_B
-5
clk
0
reset
1
write
1
select[1:0]
00
result[21:0]
0000000101011111111011
real_result
10
imaginary_result
-10
overflow_imaginary
0
overflow_real
0
150 ns
160 ns
170 ns
180 ns
190 ns
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
Activate Windows

سنتز: