آزمایش دهم :پیاده سازی یک پردازنده ساده

پردازنده

برای معماری پردازنده پشته‌ای از سایت زیر کمک گرفتیم و با کمی تغییر پردازنده ای شبیه به آن را طراحی کردیم.

https://users.ece.cmu.edu/~koopman/stack_computers/sec3_2.html

ورودی و خروجی های ماژول پردازنده بترتیب عبارت است از clkو resetو in و out و error. که ریست به صورت آسنکرون است و هر بار که ریست فعال باشد همه چیز صفر می‌شود و در ابتدای حافظه دستوراتی زبان ماشین خواسته شده ریخته می‌شود. طبق صورت آزمایش ۸ خانه‌ی بالای مموری برای I/O رزرو شده است به طوری که خانه ۲۵۵ به خروجی و خانه ۲۵۴ به ورودی پردازنده و سیگنال ارور نیز به خانه ۲۵۳ متصل شده است.

پردازنده علاوه بر استک یک رجیستر دیگر بنام top_of_stack دارد که برای عملیات های جمع و تفریق استفاده می‌شود، به این صورت که یکی از عملوند ها بعد از این که از سر استک پاپ کردیم در آن قرار می‌دهیم و در کلاک بعدی با عملوند بعدی که در بالای پشته وجود دارد جمع یا تفریق انجام دهیم.

پردازنده به این گونه است که مموری مستقیم به محتویات رجیستر MAR دسترسی دارد، و کار خواندن دستور از حافظه ۳ کلاک به طول می‌انجامد که در حالات INST_FETCH_1, INST_FETCH_2, INST_FETCH_3 انجام می‌شود. همچنین با توجه به اینکه حافظه از خانه های ۸ بیتی تشکیل شده و دستورات ما اقلا به ۱۲ بیت نیاز دارند(۴بیت آپکد و ۸ بیت داده) ما طول تمام دستورات را ثابت و ۱۶بیتی درنظرگرفتیم و هر دستور را در دو خانه متوالی از حافظه ذخیره کردیم. که قسمت اول مربوط به آپ کد و قسمت دوم مربوط داده است.

بعد از آن پردازنده وارد حالت انجام دستور می‌شود. ما برای سهولت کار و یکسان شدن اجرای زمان اجرای دستورات و ساده تر شدن حالات پردازنده ، هر دستور را در دو کلاک اجرا می‌کنیم.

همچنین توجه کنید برای عملگر SUB با توجه به مطلب زیر در سایت ویکی پدیا باید سر استک را از خانه‌ی پایینی‌اش کم کنیم.

https://en.wikipedia.org/wiki/Stack_machine#Simple_compilers

BCD و Seven segment

ماژول BCD ورودی خود را از خروجی پردازنده به صورت یک عدد ۸ بیتی می‌گیرد و رقم های یکان، دهگان و صدگان آن را به صورت ۳ عدد ۴ بیتی مشخص میکند و هر کدام از این ۳ عدد ۴ بیتی را به صورت ورودی به یک ماژول

seven_segment خروجی می‌دهد و در نهایت خروجی های ماژول های seven_segment به سون سگمت برد fpga متصل می‌کنیم.

پردازنده و اجزای جانبی باهمدیگر

در این قسمت خروجی out پردازنده را به ماژول BCD متصل کردیم و خروجی های ماژول BCD را به سه ماژول seven segment متصل کردیم و خروجی ها این سه ماژول باید در نهایت به سون سگمنت های برد متصل شود.

کد مربوط به cpu and prepherals :

```
module cpu_and_prepherals(clk, reset, in, ones_7seg, tens_7seg, hundreds_7seg, error)
;

    input clk;
    input reset;
    input [7:0] in;

    output [6:0] ones_7seg;
    output [6:0] tens_7seg;
    output [6:0] hundreds_7seg;
    output error;

    //====================================

    wire [7:0] cpu_2_bcd;

    wire [3:0] ones;
    wire [3:0] tens;
    wire [3:0] hundreds;

    //====================================

    cpu cpu(.clk(clk),
            .reset(reset),
            .in(in),
            .out(cpu_2_bcd),
            .error(error)
            );


    BCD BCD(.in(cpu_2_bcd),
            .ones(ones),
```

```
            .tens(tens),
            .hundreds(hundreds)
            );


    seven_segment   ones_seven_segment(     .bcd(ones),
                                            .seg(ones_7seg)
                                            );

    seven_segment   tens_seven_segment(     .bcd(tens),
                                            .seg(tens_7seg)
                                            );

    seven_segment   hundreds_seven_segment( .bcd(hundreds),
                                            .seg(hundreds_7seg)
                                            );

endmodule
```

کد مربوط به بخش cpu:

```
`define INSTRUCTION_LENGHT 12
`define MEMORY_MAPED_IO_STARTING_ADDRESS 2'hF8

module cpu (clk, reset, in, out, error);

    input clk;
    input reset;
    input  [7:0] in;

    output [7:0] out;
    output error;

    //=====================================

    reg [7:0] memory [255:0];
    reg [7:0] stack [7:0];

    //The Program Counter is the register that holds
    //a pointer to the next instruction to be executed
    reg [7:0]   PC          = 0;
    reg [7:0]   top_of_stack = 0;
    reg [2:0]   SP          = 0;
    reg [11:0]  IR          = 0;
    reg [7:0]   MAR         = 0;

    reg s_flag = 0;
    reg z_flag = 0;
```

```
    //we assume that sp is always pointing to a empty memory location
    reg exe_clk_cycle = 0;
    reg [3:0] state;

    wire [3:0] opcode;
    wire [7:0] address;

    integer i;

    //====================================

    parameter PUSH_CONST    = 4'b0000;
    parameter PUSH_MEMORY   = 4'b0001;
    parameter POP           = 4'b0010;
    parameter JUMP          = 4'b0011;
    parameter JZ            = 4'b0100;
    parameter JS            = 4'b0101;
    parameter ADD           = 4'b0110;
    parameter SUB           = 4'b0111;

    parameter INST_FETCH_1  = 4'b1000;
    parameter INST_FETCH_2  = 4'b1001;
    parameter INST_FETCH_3  = 4'b1010;
    parameter EXECUTE       = 4'b1011;


    //====================================

    assign opcode   = IR[11:8];
    assign address  = IR[7:0];
    assign out      = memory[255];

    assign err = (out > 127) || (in[7] == 1);
    assign error = memory[253];

    always @(posedge clk, posedge reset)begin
        if(reset == 1'b1)begin
            MAR             <= 0;
            PC              <= 0;
            IR              <= 0;
            exe_clk_cycle   <= 0;
            SP              <= 0;
            top_of_stack    <= 0;
            state           <= INST_FETCH_1;

            memory[0]  <= 8'b_0000_0000;
            memory[1]  <= 8'b_0001_0111; // push constant 23
            memory[2]  <= 8'b_0000_0001;
            memory[3]  <= 8'b_1111_1110; // push input data from input IO (mem[254])
```

```
        memory[4]  <= 8'b_0000_0110;
        memory[5]  <= 8'b_0000_0000; // add two numbers on the stack
        memory[6]  <= 8'b_0001_0000;
        memory[7]  <= 8'b_0001_0111; // push constant 23
        memory[8]  <= 8'b_0000_0001;
        memory[9]  <= 8'b_1111_1110; // push input data from input IO (mem[254])
        memory[10] <= 8'b_0000_0110;
        memory[11] <= 8'b_0000_0000; // add two numbers on the stack
        memory[12] <= 8'b_0000_0110;
        memory[13] <= 8'b_0000_0000; // add two numbers on the stack
        memory[14] <= 8'b_0000_0000;
        memory[15] <= 8'b_0000_1100; // push constant 12
        memory[16] <= 8'b_0000_0111;
        memory[17] <= 8'b_0000_0000; // subtract
        memory[18] <= 8'b_0000_0010;
        memory[19] <= 8'b_1111_1111; // pop from stack to mem[255]

        for (i=20; i<256; i = i+1) begin
            memory[i] <= 0;
        end

        for(i=0; i<8; i=i+1)begin
            stack[i] <= 0;
        end
    end
    else begin
        memory[254] <= in;
        memory[253] <= err;
        case(state)
            INST_FETCH_1: begin //set PC to MAR
                MAR <= PC;
                PC <= PC + 1;
                exe_clk_cycle <= 0;
                state <= INST_FETCH_2;
                $display("state 1");
            end

            INST_FETCH_2: begin //fetch opcode to IR
                IR[11:8] <= memory[MAR][3:0];
                MAR <= PC;
                PC <= PC + 1;
                state <= INST_FETCH_3;
                $display("state 2");
            end

            INST_FETCH_3:begin //fetch remain of instruction
                IR[7:0] <= memory[MAR];
                state <= EXECUTE;
                $display("state 3");
```

```
        end

        EXECUTE: begin
            $display("state 4");
            exe_clk_cycle <= exe_clk_cycle + 1;
            if (opcode == PUSH_CONST) begin
                if (exe_clk_cycle == 0) begin
                    stack[SP] <= IR[7:0];
                    SP <= SP + 1;
                    state <= EXECUTE;
                end
                if(exe_clk_cycle == 1)begin
                    state <= INST_FETCH_1;
                end
            end

            if (opcode == PUSH_MEMORY)begin
                if (exe_clk_cycle == 0) begin
                    MAR <= address;
                    state <= EXECUTE;
                end
                if (exe_clk_cycle == 1)begin
                    stack[SP] <= memory[MAR];
                    SP <= SP + 1;
                    state <= INST_FETCH_1;
                end
            end

            if (opcode == POP) begin
                if (exe_clk_cycle == 0) begin
                    MAR <= address;
                    SP <= SP - 1;
                    state <= EXECUTE;
                end
                if (exe_clk_cycle == 1)begin
                    memory[MAR] <= stack[SP];
                    state <= INST_FETCH_1;
                end
            end

            if (opcode == JUMP) begin
                if (exe_clk_cycle == 0) begin
                    SP <= SP - 1;
                    state <= EXECUTE;
                end
                if (exe_clk_cycle == 1)begin
                    PC <= stack[SP];
                    state <= INST_FETCH_1;
                end
```

```
                end

                if (opcode == JZ) begin
                    if (exe_clk_cycle == 0) begin
                        if(z_flag == 1'b1)begin
                            SP <= SP - 1;
                            state <= EXECUTE;
                        end
                        else begin
                            state <= EXECUTE;
                        end
                    end
                    if (exe_clk_cycle == 1)begin
                        if(z_flag == 1'b1)begin
                            PC <= stack[SP];
                            state <= INST_FETCH_1;
                        end
                        else begin
                            state <=  INST_FETCH_1;
                        end
                    end
                end

                if (opcode == JS) begin
                    if (exe_clk_cycle == 0) begin
                        if(s_flag == 1'b1)begin
                            SP <= SP - 1;
                            state <= EXECUTE;
                        end
                        else begin
                            state <= EXECUTE;
                        end
                    end
                    if (exe_clk_cycle == 1)begin
                        if(s_flag == 1'b1)begin
                            PC <= stack[SP];
                            state <= INST_FETCH_1;
                        end
                        else begin
                            state <= INST_FETCH_1;
                        end
                    end
                end

                if (opcode == ADD) begin
                    if (exe_clk_cycle == 0) begin
                        top_of_stack <= stack[SP-1];
                        SP <= SP - 1;
                        state <= EXECUTE;
```

```
                        end
                        if (exe_clk_cycle == 1)begin
                            stack[SP-1] <= stack[SP-1] + top_of_stack;
                            if (((stack[SP-
1] + top_of_stack) & 8'b_1000_0000) == 8'b_1000_0000 )begin
                                s_flag <= 1;
                            end
                            else begin
                                s_flag <= 0;
                            end
                            if ((stack[SP-1] + top_of_stack) == 8'b_0000_0000)begin
                                z_flag <= 1;
                            end
                            else begin
                                z_flag <= 0;
                            end
                            state <= INST_FETCH_1;
                        end
                    end
                    if (opcode ==SUB) begin
                        if (exe_clk_cycle == 0) begin
                            top_of_stack <= stack[SP-1];
                            SP <= SP - 1;
                            state <= EXECUTE;
                        end
                        if (exe_clk_cycle == 1)begin
                            stack[SP-1] <= stack[SP-1] - top_of_stack;
                            if(((stack[SP-
1] - top_of_stack) & 8'b_1000_0000) == 8'b_1000_0000)begin
                                s_flag <= 1;
                            end
                            else begin
                                s_flag <= 0;
                            end
                            if((stack[SP-1] - top_of_stack) == 8'b_0000_0000) begin
                                z_flag <= 1;
                            end
                            else begin
                                z_flag <= 0;
                            end
                            state <= INST_FETCH_1;
                        end
                    end
                end
            endcase
        end
    end

endmodule
```

کد مربوط به BCD:

```
module BCD (in, ones, tens, hundreds);

    input [7:0] in;

    output reg [3:0] ones;
    output reg [3:0] tens;
    output reg [3:0] hundreds;

    //====================================

    integer i;

    //====================================

    always@(in)begin
        hundreds = 4'd0;
        tens = 4'd0;
        ones = 4'd0;
        for(i=7; i>=0; i=i-1)begin
            if(hundreds >= 5)
                hundreds = hundreds + 3;
            if(tens >= 5)
                tens = tens + 3;
            if(ones >= 5)
                ones = ones + 3;

            hundreds = hundreds << 1;
            hundreds[0] = tens[3];
            tens = tens << 1;
            tens[0] = ones[3];
            ones = ones << 1;
            ones[0] = in[i];
        end
    end
endmodule
```

کد مربوط به seven segment:

```
module seven_segment(bcd, seg);
```

```
    input [3:0] bcd;
    output reg [6:0] seg;

    //====================================

    always @(bcd)
    begin
        case (bcd)
            0 : seg = 7'b0000001;
            1 : seg = 7'b1001111;
            2 : seg = 7'b0010010;
            3 : seg = 7'b0000110;
            4 : seg = 7'b1001100;
            5 : seg = 7'b0100100;
            6 : seg = 7'b0100000;
            7 : seg = 7'b0001111;
            8 : seg = 7'b0000000;
            9 : seg = 7'b0000100;
            //switch off 7 segment character when the bcd digit is not a decimal numb
er.
            default : seg = 7'b1111111;
        endcase
    end

endmodule
```

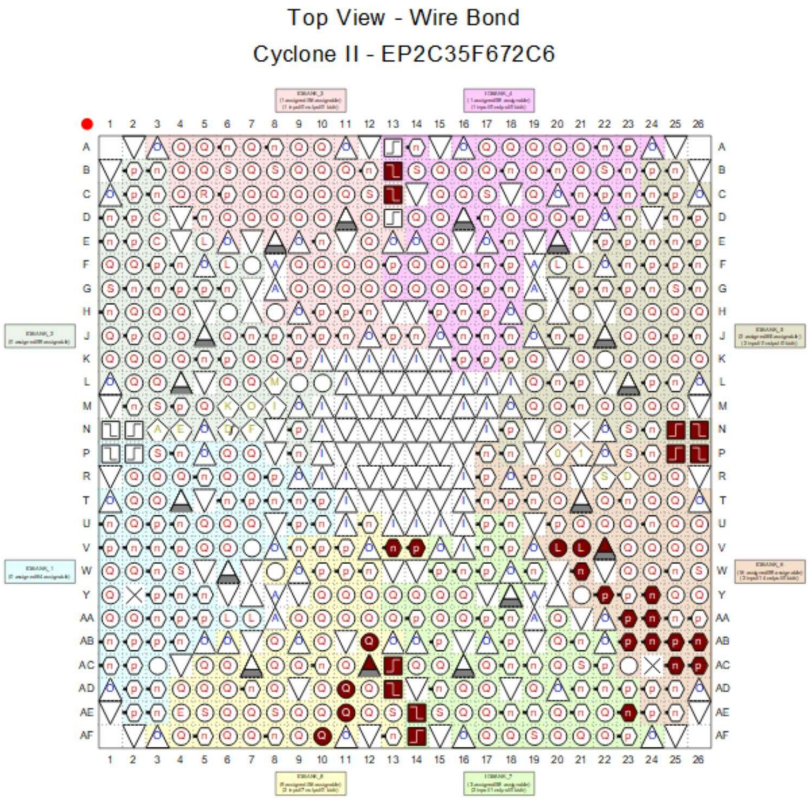
کد مربوط به تست بنچ:

```
module test_cpu();

    reg [7:0] data_in;
    reg clk;
    reg reset;

    wire [6:0] ones;
    wire [6:0] tens;
    wire [6:0] hundreds;

    wire error;
    wire [7:0] out;
    //====================================

    // cpu cpu(    .clk(clk),
    //             .reset(reset),
    //             .in(data_in),
    //             .out(out),
    //             .error(error)
```

```
    //         );

    cpu_and_prepherals  cpu_and_prepherals( .clk(clk),
                                            .reset(reset),
                                            .in(data_in),
                                            .ones_7seg(ones),
                                            .tens_7seg(tens),
                                            .hundreds_7seg(hundreds),
                                            .error(error)
                                          );

    //====================================

    initial begin
        reset = 0;
        clk = 0;
        #1
        reset = 1;
        #12
        reset = 0;
        data_in = 8'b_0110_0010;
        #15
        ;
        ////////////////////////////////////////////
        // #220
        // data_in = 8'b10101010;
    end

    always #5 clk = ~clk;

endmodule
```

اختصاص پین و کامپایل

برای پیاده‌سازی این مدار از دستگاه EP2C35F672C6 از خانواده Cyclone II استفاده خواهد شد. با توجه به راهنمای این دستگاه، پین‌ها را مطابق شکل 1 اختصاص می‌دهیم.



Named: * Edit:

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_P26	6	B6_N0	PIN_P26	3.3-V LV...default)		24mA (default)	
error	Output	PIN_AE23	7	B7_N0	PIN_AE23	3.3-V LV...default)		24mA (default)	
hundreds_7seg[6]	Output	PIN_Y24	6	B6_N1	PIN_Y24	3.3-V LV...default)		24mA (default)	
hundreds_7seg[5]	Output	PIN_AB25	6	B6_N1	PIN_AB25	3.3-V LV...default)		24mA (default)	
hundreds_7seg[4]	Output	PIN_AB26	6	B6_N1	PIN_AB26	3.3-V LV...default)		24mA (default)	
hundreds_7seg[3]	Output	PIN_AC26	6	B6_N1	PIN_AC26	3.3-V LV...default)		24mA (default)	
hundreds_7seg[2]	Output	PIN_AC25	6	B6_N1	PIN_AC25	3.3-V LV...default)		24mA (default)	
hundreds_7seg[1]	Output	PIN_V22	6	B6_N1	PIN_V22	3.3-V LV...default)		24mA (default)	
hundreds_7seg[0]	Output	PIN_AB23	6	B6_N1	PIN_AB23	3.3-V LV...default)		24mA (default)	
in[7]	Input	PIN_C13	3	B3_N0	PIN_C13	3.3-V LV...default)		24mA (default)	
in[6]	Input	PIN_AC13	8	B8_N0	PIN_AC13	3.3-V LV...default)		24mA (default)	
in[5]	Input	PIN_AD13	8	B8_N0	PIN_AD13	3.3-V LV...default)		24mA (default)	
in[4]	Input	PIN_AF14	7	B7_N1	PIN_AF14	3.3-V LV...default)		24mA (default)	
in[3]	Input	PIN_AE14	7	B7_N1	PIN_AE14	3.3-V LV...default)		24mA (default)	
in[2]	Input	PIN_P25	6	B6_N0	PIN_P25	3.3-V LV...default)		24mA (default)	
in[1]	Input	PIN_N26	5	B5_N1	PIN_N26	3.3-V LV...default)		24mA (default)	
in[0]	Input	PIN_N25	5	B5_N1	PIN_N25	3.3-V LV...default)		24mA (default)	
ones_7seg[6]	Output	PIN_V13	8	B8_N0	PIN_V13	3.3-V LV...default)		24mA (default)	
ones_7seg[5]	Output	PIN_V14	8	B8_N0	PIN_V14	3.3-V LV...default)		24mA (default)	
ones_7seg[4]	Output	PIN_AE11	8	B8_N0	PIN_AE11	3.3-V LV...default)		24mA (default)	
ones_7seg[3]	Output	PIN_AD11	8	B8_N0	PIN_AD11	3.3-V LV...default)		24mA (default)	
ones_7seg[2]	Output	PIN_AC12	8	B8_N0	PIN_AC12	3.3-V LV...default)		24mA (default)	
ones_7seg[1]	Output	PIN_AB12	8	B8_N0	PIN_AB12	3.3-V LV...default)		24mA (default)	
ones_7seg[0]	Output	PIN_AF10	8	B8_N0	PIN_AF10	3.3-V LV...default)		24mA (default)	
reset	Input	PIN_G26	5	B5_N0	PIN_B13	3.3-V LV...default)		24mA (default)	
tens_7seg[6]	Output	PIN_AB24	6	B6_N1	PIN_AB24	3.3-V LV...default)		24mA (default)	
tens_7seg[5]	Output	PIN_AA23	6	B6_N1	PIN_AA23	3.3-V LV...default)		24mA (default)	
tens_7seg[4]	Output	PIN_AA24	6	B6_N1	PIN_AA24	3.3-V LV...default)		24mA (default)	
tens_7seg[3]	Output	PIN_Y22	6	B6_N1	PIN_Y22	3.3-V LV...default)		24mA (default)	
tens_7seg[2]	Output	PIN_W21	6	B6_N1	PIN_W21	3.3-V LV...default)		24mA (default)	

All Pins

شکل 1 - نحوه‌ی اختصاص پین‌ها

ورودی in را بوسیله toggle switch ها ورودی reset را به push button متصل کردیم و خروجی error را نیز به LED و کلاک را نیز به external clock و خروجی های سون سگمنت را به hex0 , hex2 , hex1 که مربوط به سون سگمنت های برد fpga است وصل کردیم.

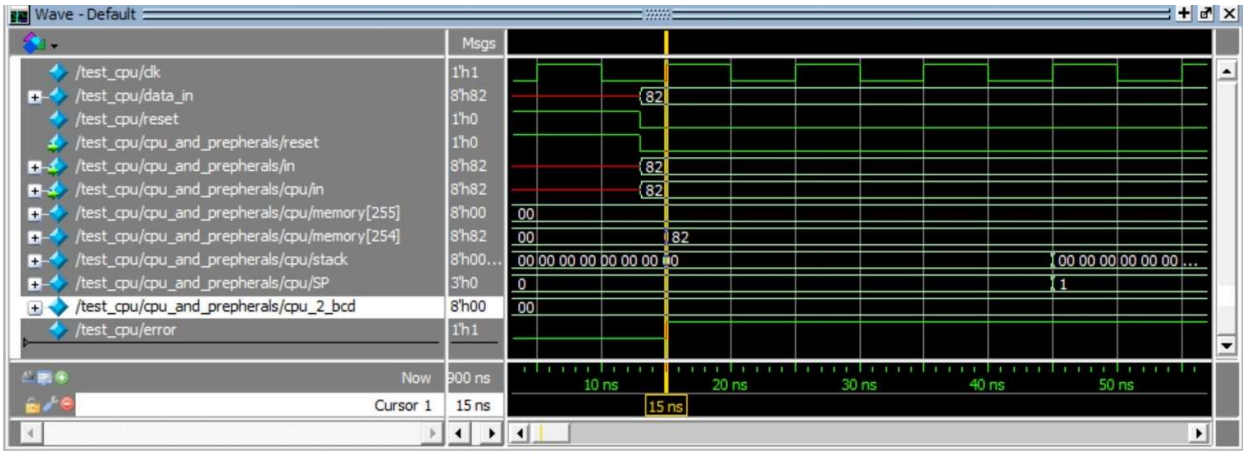
حال، پروژه را کامپایل می‌کنیم. نتیجه‌ی تحلیل و سنتز پس از کامپایل پروژه در شکل 2 نشان داده شده است.

Flow Summary	
Flow Status	Successful - Sun Aug 02 18:53:26 2020
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	cpu_and_prepherals
Top-level Entity Name	cpu_and_prepherals
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	2,997 / 33,216 (9 %)
Total combinational functions	2,061 / 33,216 (6 %)
Dedicated logic registers	2,157 / 33,216 (6 %)
Total registers	2157
Total pins	32 / 475 (7 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

Task	Time
Compile Design	00:01:01
Analysis & Synthesis	00:00:14
Fitter (Place & Route)	00:00:37
Assembler (Generate programming files)	00:00:05
TimeQuest Timing Analysis	00:00:05
EDA Netlist Writer	
Program Device (Open Programmer)	

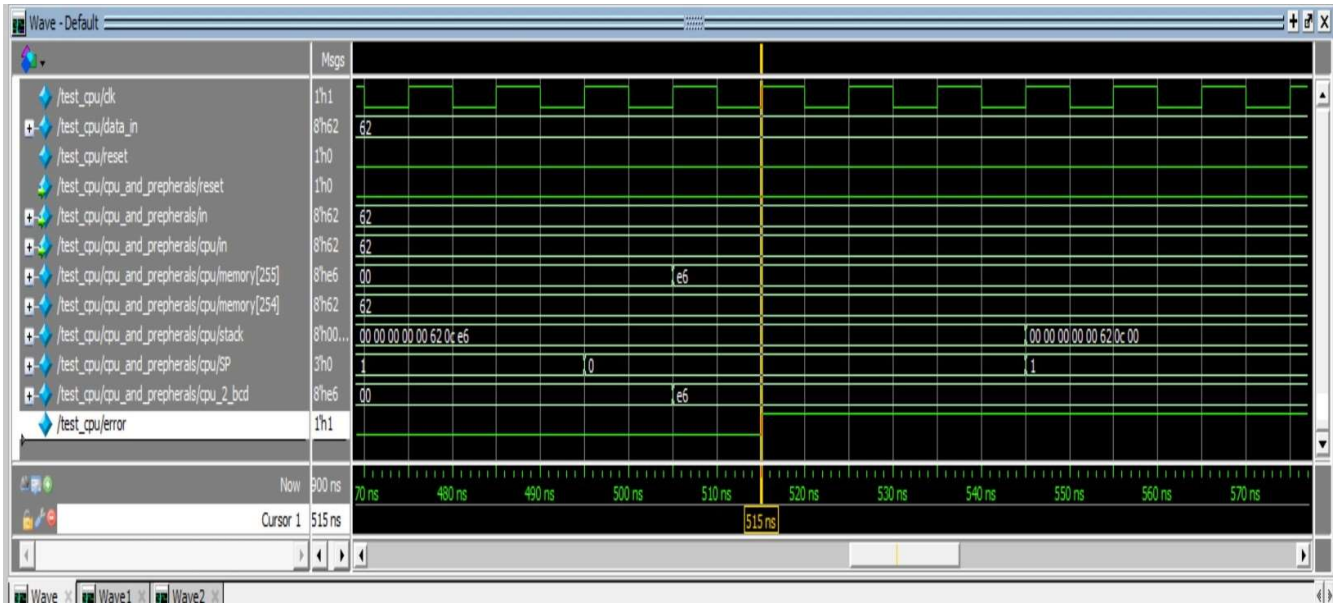
شکل 2 - نتیجه‌ی سنتز ماژول cpu_and_prepherals

شکل 3، تست عملکرد ماژول پیاده‌سازی شده به کمک Waveform را نشان می‌دهد.

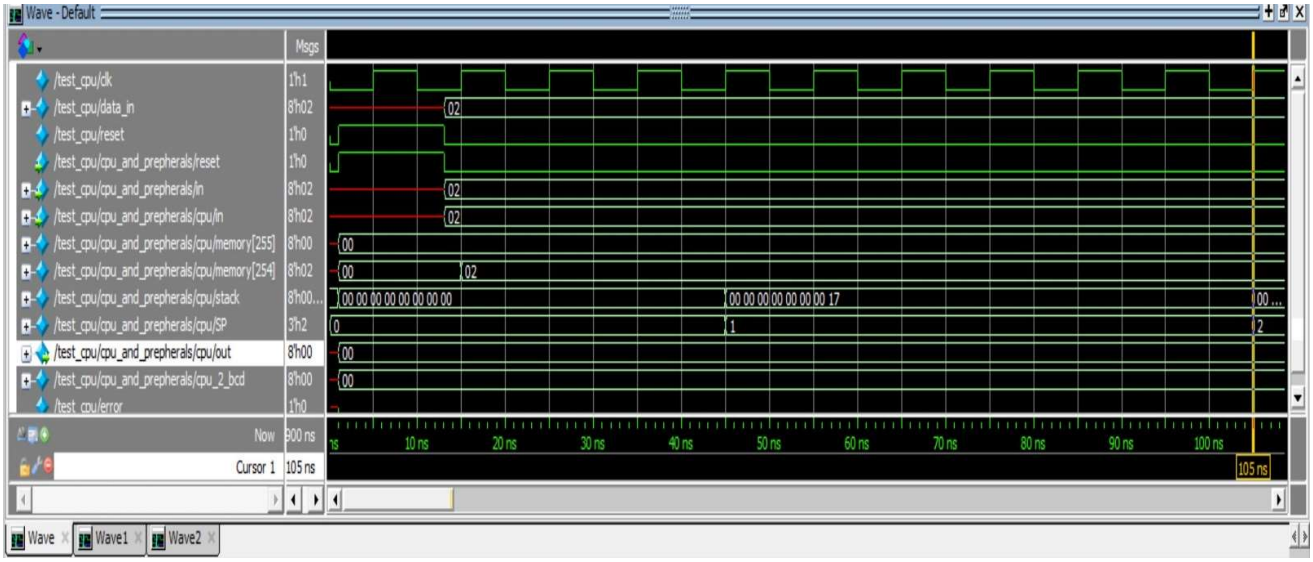


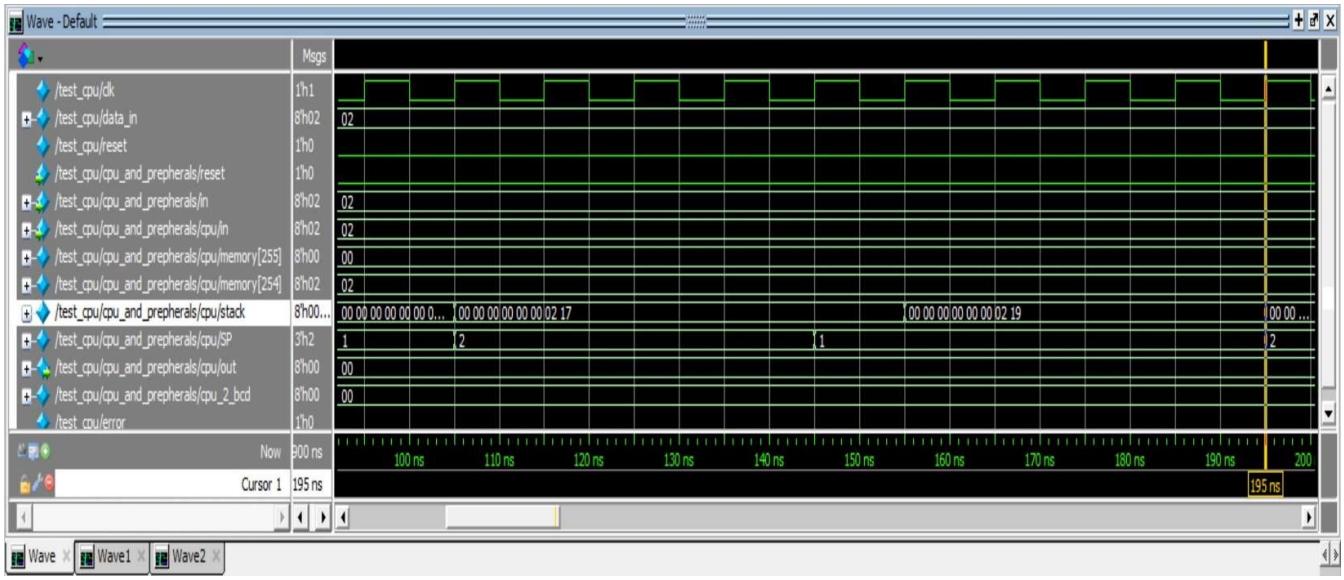
شکل 3 - شکل موج حاصل از تست عملکرد ماژول تست بنچ

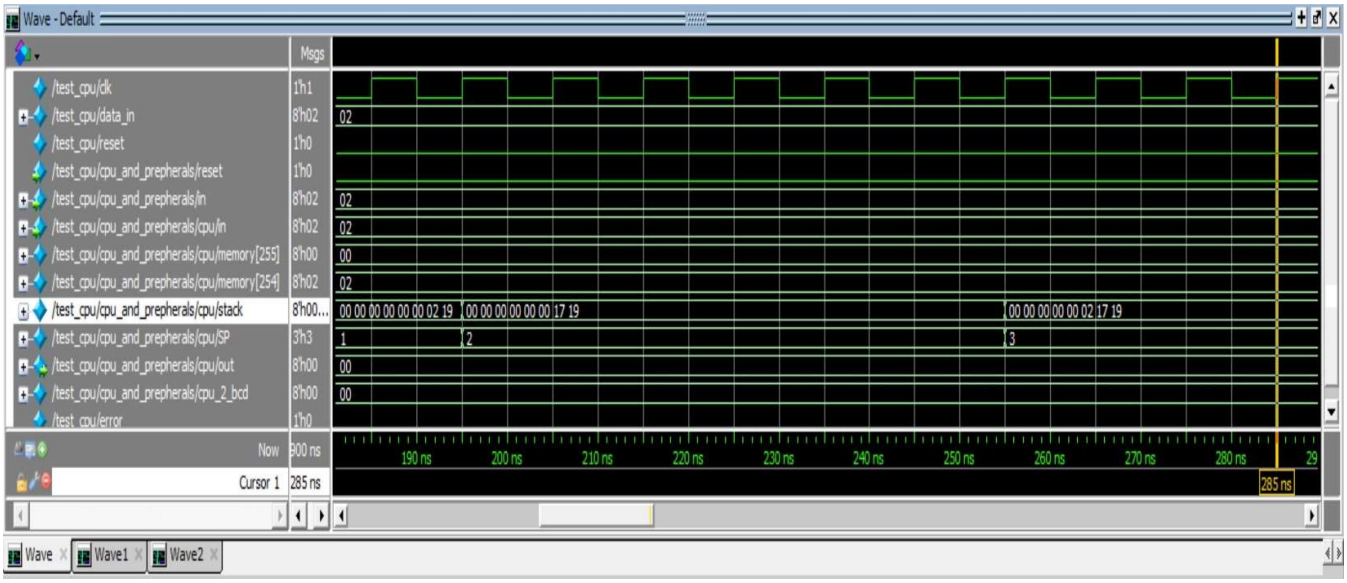
ابتدا سیگنال ارور را بررسی می‌کنیم. همانطور که در شل بالا میبینید پس از اینکه ریست تمام میشود با توجه به اینکه ورودی یک عدد منفی است سیگنال ارور برابر یک می‌شود.

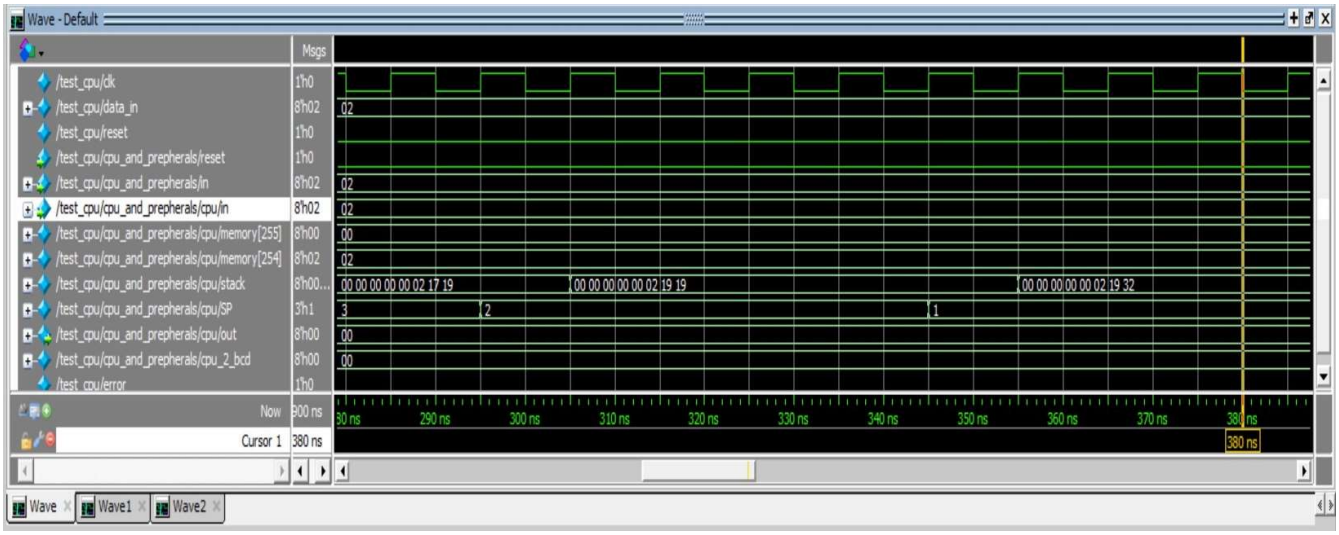


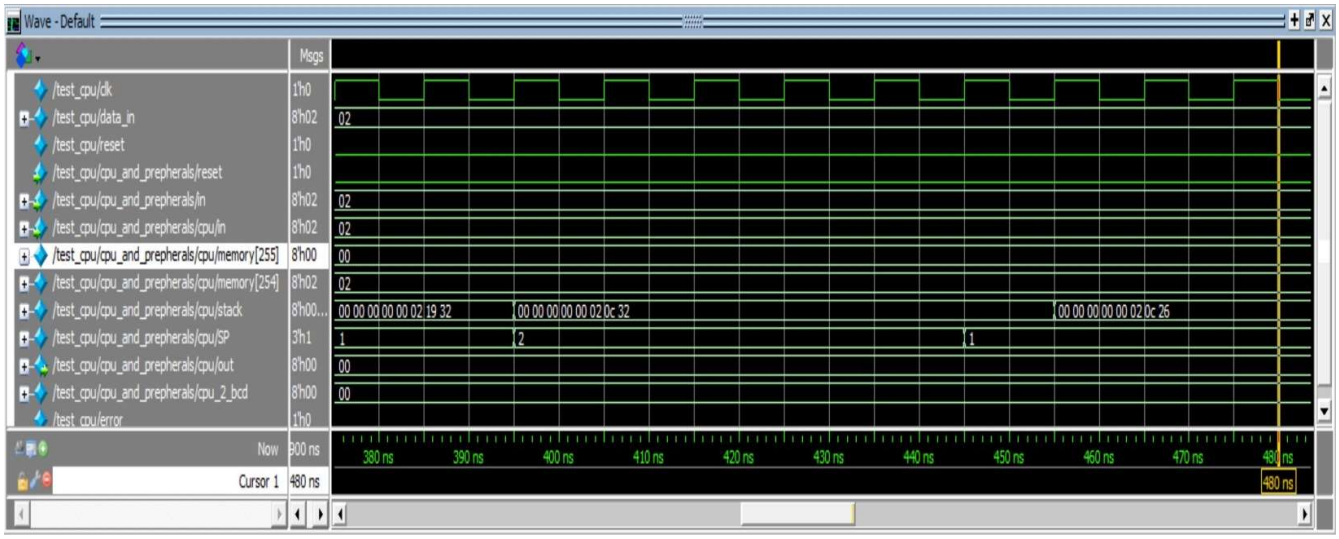
در شکل بالا نتیجه محاسبه عددی منفی است که پس از این که این عدد در خروجی نوشته می‌شود یک کلاک بعد سیگنال ارور برابر یک می‌شود. (x= 50 , y = (50 +23)*2 – 12 = 134 > 127)

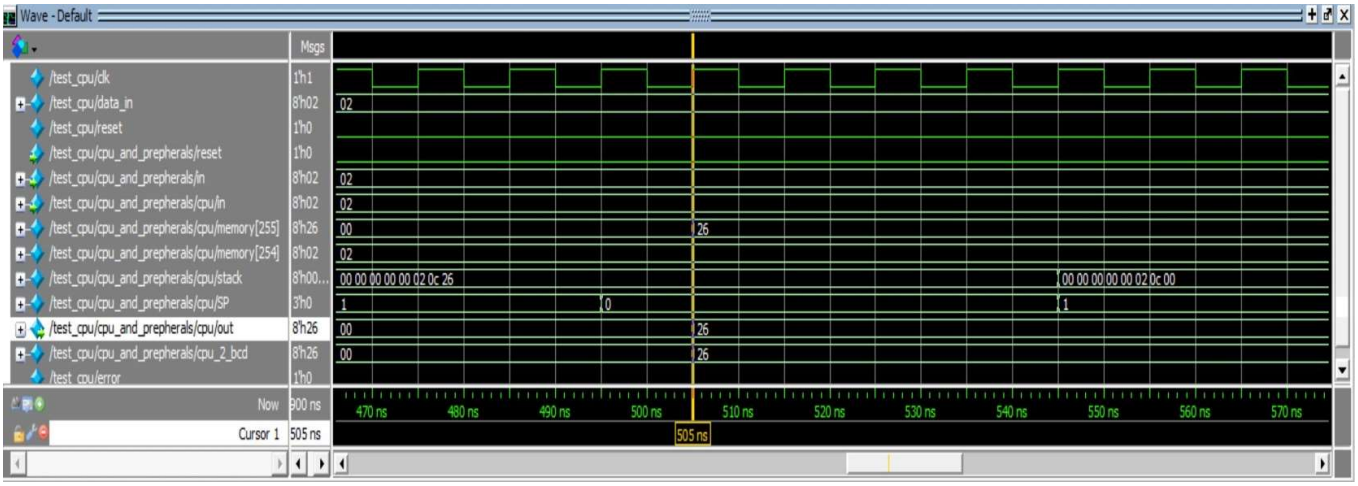



Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/cpu_and_prepherals/cpu_2_bcd
/test_cpu/error
Now
Cursor 1
100 ns
110 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns
190 ns
195 ns
Wave
Wave1
Wave2

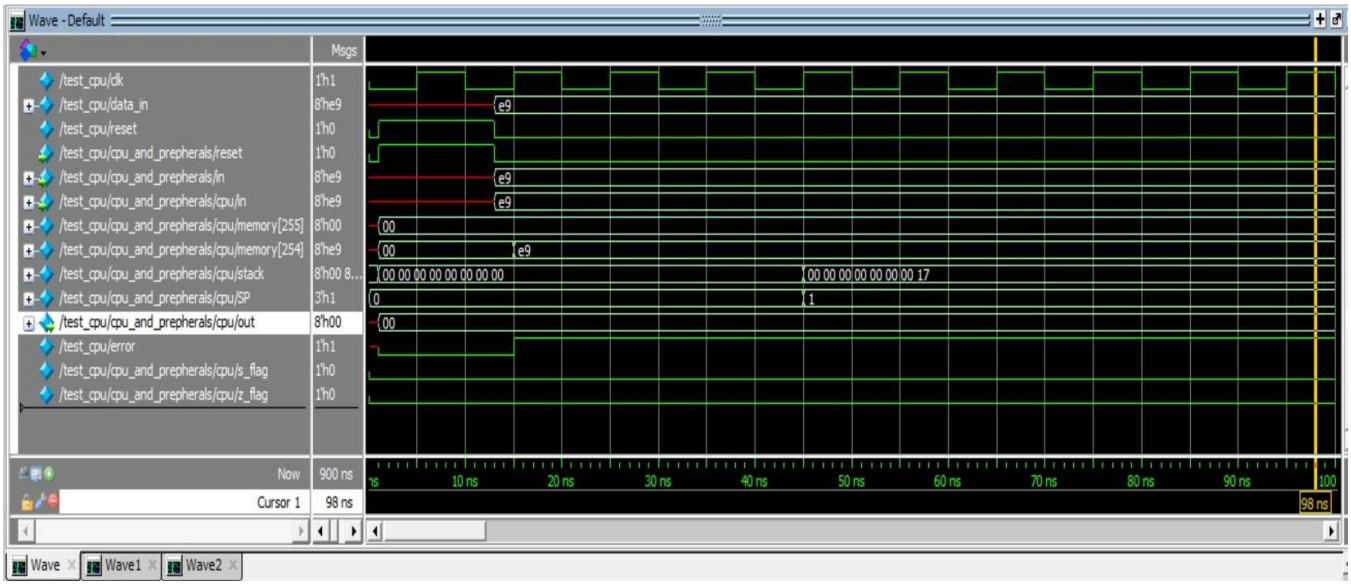

Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/cpu_and_prepherals/cpu_2_bcd
/test_cpu/error
Now
Cursor 1
190 ns
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
270 ns
280 ns
285 ns
Wave
Wave1
Wave2

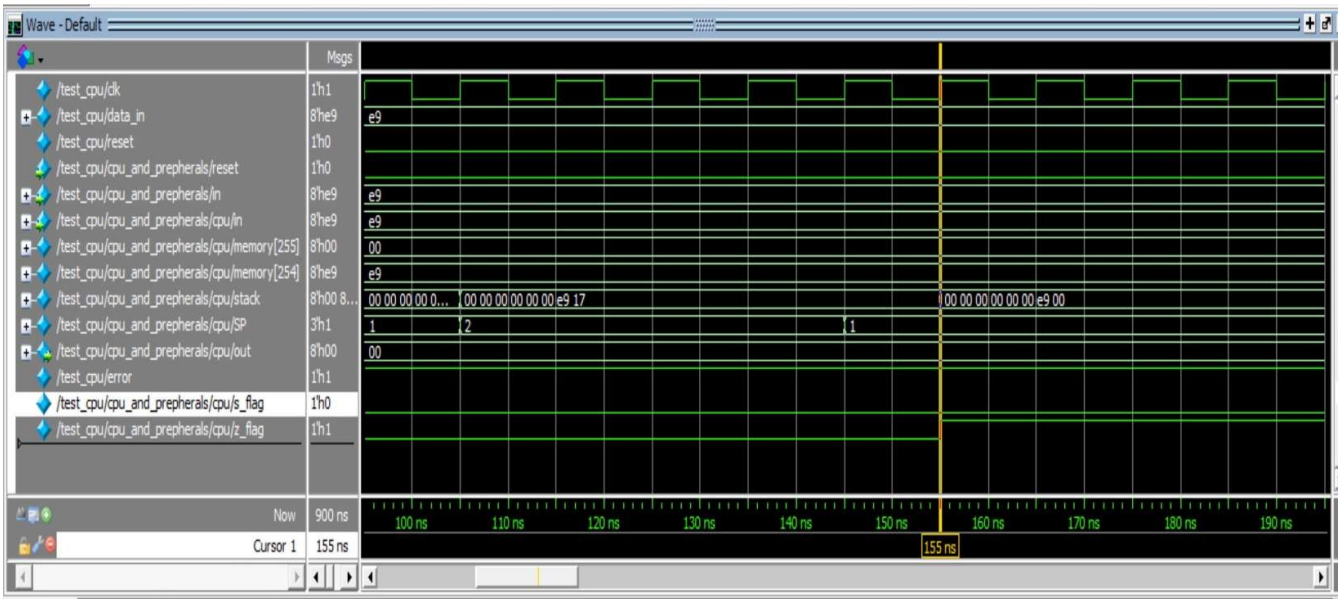

Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/cpu_and_prepherals/cpu_2_bcd
/test_cpu/error
Now
Cursor 1
Wave
Wave1
Wave2

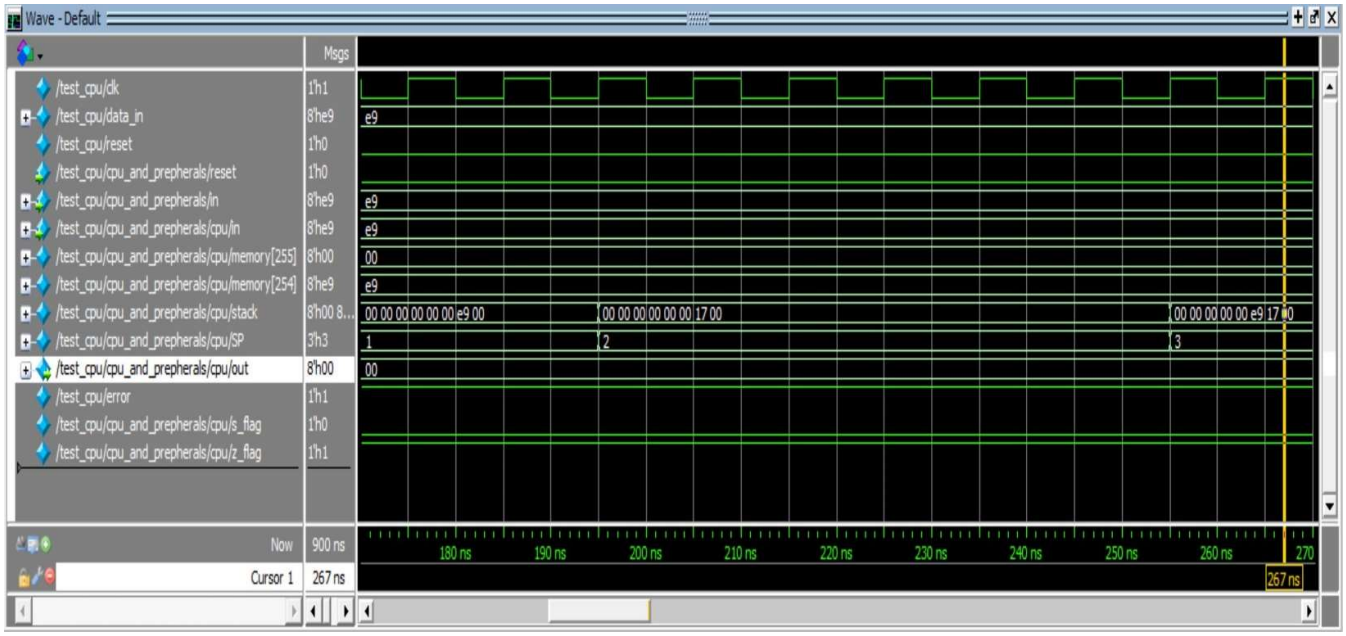

Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/cpu_and_prepherals/cpu_2_bcd
/test_cpu/error
Now
Cursor 1

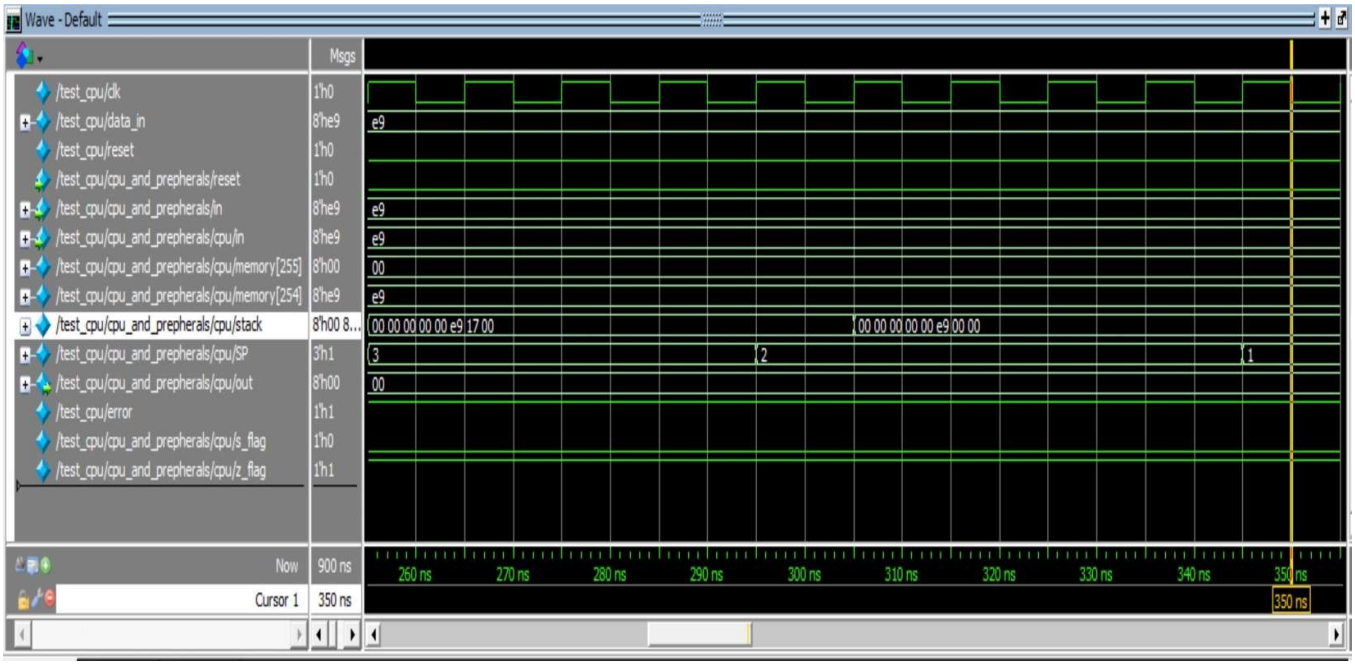

Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/cpu_and_prepherals/cpu_2_bcd
/test_cpu/error
Now
Cursor 1

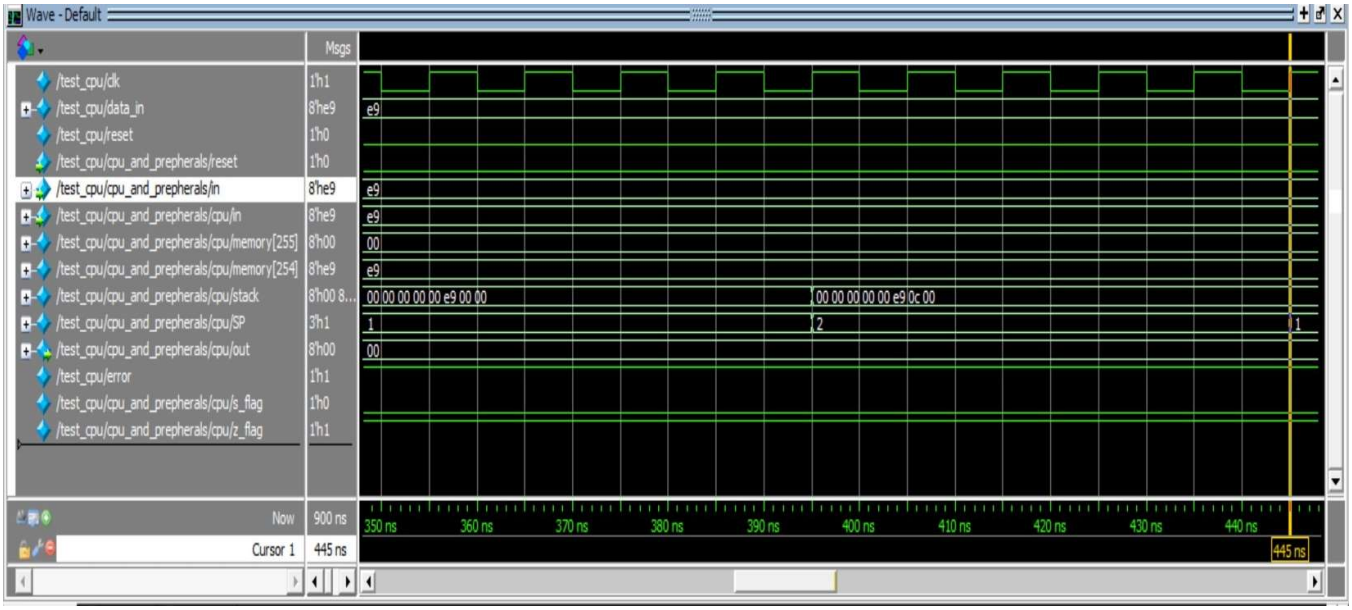
در شکل های بالا ما ورودی را برابر ۲ قرار دادیم و مراحل کار سی پی یو را مشاهده می‌کنید. در شکل اخر هم می‌بینیم که عدد
(23+2)*2 -12 =38 = 2*16 + 6 = 26hex در خروجی نوشته شده است.

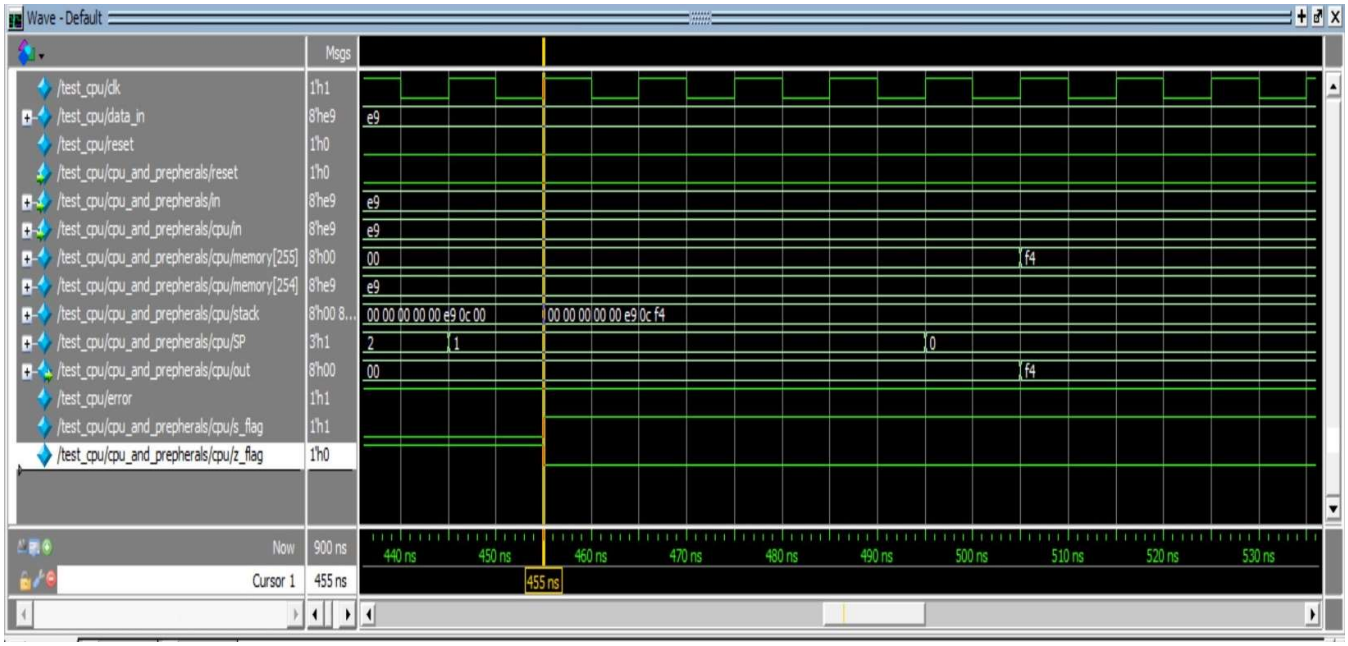





Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/error
/test_cpu/cpu_and_prepherals/cpu/s_flag
/test_cpu/cpu_and_prepherals/cpu/z_flag
Now
Cursor 1
180 ns
190 ns
200 ns
210 ns
220 ns
230 ns
240 ns
250 ns
260 ns
267 ns


Wave - Default
Msgs
/test_cpu/clk
/test_cpu/data_in
/test_cpu/reset
/test_cpu/cpu_and_prepherals/reset
/test_cpu/cpu_and_prepherals/in
/test_cpu/cpu_and_prepherals/cpu/in
/test_cpu/cpu_and_prepherals/cpu/memory[255]
/test_cpu/cpu_and_prepherals/cpu/memory[254]
/test_cpu/cpu_and_prepherals/cpu/stack
/test_cpu/cpu_and_prepherals/cpu/SP
/test_cpu/cpu_and_prepherals/cpu/out
/test_cpu/error
/test_cpu/cpu_and_prepherals/cpu/s_flag
/test_cpu/cpu_and_prepherals/cpu/z_flag
Now
Cursor 1
260 ns
270 ns
280 ns
290 ns
300 ns
310 ns
320 ns
330 ns
340 ns
350 ns





در این شکل ها هم به بررسی سیگنال های z_flag و s_flag پرداختیم. در ورودی ما عدد -۲۳ دادیم که همانطور که مشاهد می‌کنید پس از عملیات های جمع سیگنال z_flag یک می‌شود(و باقی می‌ماند). در شکل خر هم که ۰ را منهای ۱۲ می‌کند می‌بینیم که سیگنال z_flag صفر شده و سیگنال s_flag برابر یک می‌شود. همچنین می‌بینیم که این دو سیگنال در دستور های دیگر تغییری نمی‌کنند.