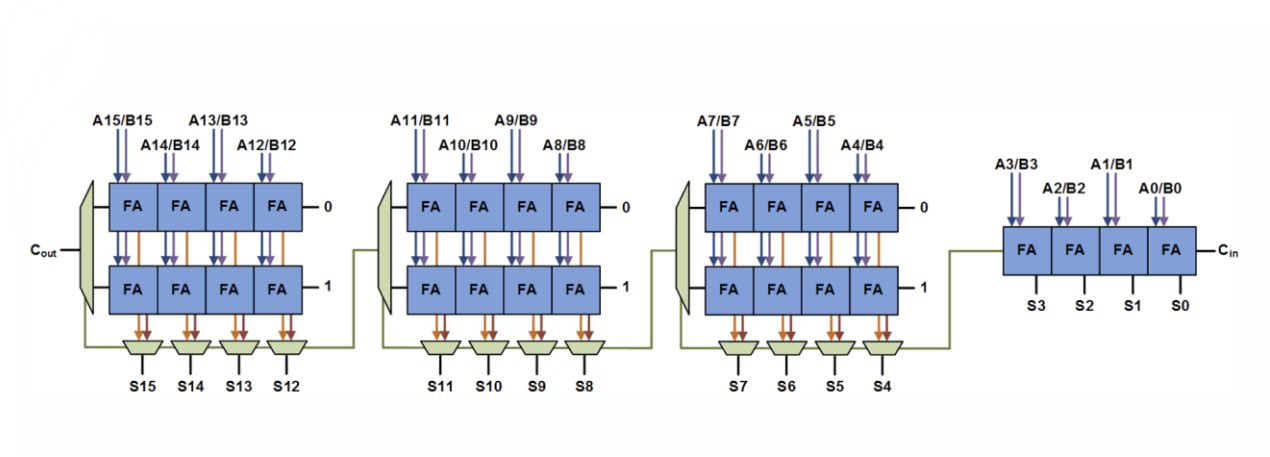
غزاله کریمی
402106372
تمرین سوم

ALU

جمع کننده ی carry select adder:

میخواهیم سخت افزار زیر را در سطح گیت پیاده سازی کنیم برای اینکار ابتدا ماژول های FA و mux2to1 را می سازیم.



ماژول FA و mux2to1 پیاده سازی شده در سطح گیت:

```
module mux2to1(
    input I0,I1,sel,
    output out
);
    wire W1,W2,W3;
    not(W1,sel);
    and(W2,I0,W1);
    and(W3,I1,sel);
    or(out,W2,W3);
endmodule
```

```
module FA(
    input a,b,cin,
    output s,cout
);
    wire W1,W2,W3,W4;
    xor(W1,a,b);
    xor(s,W1,cin);
    and(W2,a,b);
    and(W3,b,cin);
    and(W4,a,cin);
    or(cout,W2,W3,W4);

endmodule
```

سپس جمع کننده های 4 بیتی را که به صورت ripple carry هستند با استفاده از instance هایی از FA می سازیم و cin هر FA از cout واحد FA سمت راستش می آید.

ماژول rippleCarryAdder:

```
module rippleCarryAdder(
    input [3:0] A,B,
    input C_in,
    output [3:0] sum,
    output cout
);
    wire C1,C2,C3;
    FA FA0(.a(A[0]), .b(B[0]),.cin(C_in),.s(sum[0]),.cout(C1));
    FA FA1(.a(A[1]), .b(B[1]),.cin(C1),.s(sum[1]),.cout(C2));
    FA FA2(.a(A[2]), .b(B[2]),.cin(C2),.s(sum[2]),.cout(C3));
    FA FA3(.a(A[3]), .b(B[3]),.cin(C3),.s(sum[3]),.cout(cout));

endmodule
```

و همچنین با استفاده از mux2to1 یک mux2to1 چهار بیتی می سازیم :

```
module mux4to1_2to1(
    input [3:0] I0,
    input [3:0] I1,
    input sel,
    output [3:0] out
);
    mux2to1 mux0(.I0(I0[0]), .I1(I1[0]), .sel(sel), .out(out[0]));
    mux2to1 mux1(.I0(I0[1]), .I1(I1[1]), .sel(sel), .out(out[1]));
    mux2to1 mux2(.I0(I0[2]), .I1(I1[2]), .sel(sel), .out(out[2]));
    mux2to1 mux3(.I0(I0[3]), .I1(I1[3]), .sel(sel), .out(out[3]));
endmodule

```

ماژول Carry select adder:

در این ماژول، ورودی‌ها A و B (هرکدام ۱۶ بیت) به همراه Carry-in (Cin) دریافت می‌شوند. خروجی‌ها شامل sum (جمع ۱۶ بیتی) و Carry-out (Cout نهایی) هستند.

این جمع‌کننده به ۴ بلوک ۴ بیتی تقسیم شده است. تنها بلوک اول (کم‌ارزش‌ترین ۴ بیت) به صورت Ripple Carry معمولی عمل می‌کند، و بلوک‌های بعدی از دو جمع‌کننده‌ی موازی برای دو حالت (carry ۰ و ۱) استفاده می‌کنند. سپس با استفاده از مالتی‌پلکسر خروجی مناسب انتخاب می‌شود.

```
module CSAadder(
    input [15:0] A,B,
    input Cin,
    output [15:0] sum,
    output Cout
);
    wire C1, C2, C3;            //actual carry between groups
    wire [3:0] sum0_1, sum1_1, sum0_2, sum1_2, sum0_3, sum1_3;
    wire C0_1, C1_1, C0_2, C1_2, C0_3, C1_3;

```

سیم های C1,C2,C3 کری اوت نهایی هر بلوک هستند و سیم های sum0_i , sum1_i نتیجه جمع ripple carry adder ها هستند که یا با کری این 0 یا 1 هستند و C0_i , C1_i کری اوت بلوک ها با کری این 0 و یک هستند.

```
    //first 4 bits: normal ripple carry adder with actual carry in
    rippleCarryAdder rca0(
        .A(A[3:0]), .B(B[3:0]), .C_in(Cin),
        .sum(sum[3:0]), .cout(C1)
    );
```

این بخش ازتنها یک Ripple Carry Adder برای 4 بیت کم ارزش استفاده می‌کند. ورودی‌های A[3:0] و B[3:0] همراه با Cin وارد شده و خروجی sum[3:0] و Carry داخلی C1 تولید می‌شود.

```
    rippleCarryAdder rca1_0(
        .A(A[7:4]), .B(B[7:4]), .C_in(1'b0),
        .sum(sum0_1), .cout(C0_1)
    );
    rippleCarryAdder rca1_1(
        .A(A[7:4]), .B(B[7:4]), .C_in(1'b1),
        .sum(sum1_1), .cout(C1_1)
    );
    mux2to1 mux1(.I0(C0_1),.I1(C1_1),.sel(C1),.out(C2));
    mux4to1_2to1 mux2(.I0(sum0_1),.I1(sum1_1),.sel(C1),.out(sum[7:4]));
```

بیت‌های ۴ تا ۷ و بلوک دوم : برای این بخش، از دو ripple carry adder با Cin=0 و Cin=1 به صورت موازی محاسبه می‌شوند. بسته به C1 که از مرحله قبل آمده، خروجی درست با مالتی‌پلکسر انتخاب می‌شود. از یک mux یک بیتی 2 در 1 برای تعیین C2 استفاده میکنیم و از یک mux چهار بیتی 2 به 1 برای تعیین sum[7:4] استفاده میکنیم . و به همین ترتیب برای بیت های 8 تا 11 و 11 تا 15 و Cout نهایی از بلوک آخر تعیین می شود :

```
    rippleCarryAdder rca2_0(
        .A(A[11:8]), .B(B[11:8]), .C_in(1'b0),
        .sum(sum0_2), .cout(C0_2)
    );
    rippleCarryAdder rca2_1(
        .A(A[11:8]), .B(B[11:8]), .C_in(1'b1),
        .sum(sum1_2), .cout(C1_2)
    );
    mux2to1 mux3(.I0(C0_2),.I1(C1_2),.sel(C2),.out(C3));
    mux4to1_2to1 mux4(.I0(sum0_2),.I1(sum1_2),.sel(C2),.out(sum[11:8]));

    rippleCarryAdder rca3_0(
        .A(A[15:12]), .B(B[15:12]), .C_in(1'b0),
        .sum(sum0_3), .cout(C0_3)
    );
    rippleCarryAdder rca3_1(
        .A(A[15:12]), .B(B[15:12]), .C_in(1'b1),
        .sum(sum1_3), .cout(C1_3)
    );
    mux2to1 mux5(.I0(C0_3),.I1(C1_3),.sel(C3),.out(Cout));
    mux4to1_2to1 mux6(.I0(sum0_3),.I1(sum1_3),.sel(C3),.out(sum[15:12]));

endmodule
```

لازم به ذکر است از این جمع کننده به عنوان تفریق کننده هم استفاده می کنیم فقط کافی است در ALU عدد دوم را نات کنیم و Cin را برابر با یک بگذاریم تا مکمل دو عدد دوم را به جمع کننده پاس داده باشیم.

ضرب کننده:

ماژول ضربکننده 8 بیتی با استفاده از الگوریتم shift and add:

```
module mult8bit (
    input [7:0] a, b,
    output reg [15:0] p
);
    integer i;
    reg [15:0] temp, product;

    always @(*) begin
        temp = {8'b0, a};
        product = 16'b0;

        for (i = 0; i < 8; i = i + 1) begin
            if (b[i] == 1'b1) begin
                product = product + temp;
            end
            else begin
                product = product + 16'b0;
            end
            temp = temp << 1;
        end

        p = product;
    end
endmodule
```

این ماژول دو عدد 8 بیتی را میگیرد و نتیجه 16 بیتی را برمیگرداند برای اینکار temp را تعریف می کنیم که همان a است که هر مرحله از حلقه یک بیت شیفت به چپ داده می شود. سپس روی بیت های b یک For می زنیم و از کم ارزش ترین بیت b شروع می کنیم اگر b[i] یک بود product را با temp جمع می کنیم و اگر 0 باشد با 0 جمعش می کنیم و سپس در هر مرحله از حلقه temp را یک بیت به چپ شیفت میدهیم یا به عبارتی ضرب در 2 می کنیم و ذر انتها product را در p می ریزیم و همچنین temp را در ابتدا 16 بیتی در نظر گرفتیم و 8 بیت پر ارزش آن را 0 کردیم تا وقتی آن را شیفت می دهیم بیت های a از بین نروند.

ماژول ضرب کننده karatsuba:

این ضرب کننده بدون علامت است و قدر مطلق اعداد به آن داده می شوند و علامت حاصلضرب با xor علامت دو عدد تعیین می شود که این قضیه در ALU هندل می شود و همچنین خروجی ضرب کننده 32 بیتی است که با توجه به اینکه رجیستر ها 16 بیتی هستند در ALU فقط 16 بیت کم ارزش آن به عنوان نتیجه خروجی داده می شود.

```
module multiplier (
    input [15:0] A, B,
    output [31:0] P
);

    wire [7:0] XH = A[15:8];   // High 8 bits of A
    wire [7:0] XL = A[7:0];    // Low 8 bits of A
    wire [7:0] YH = B[15:8];   // High 8 bits of B
    wire [7:0] YL = B[7:0];    // Low 8 bits of B

    wire [15:0] Z0, Z2, Z1_0;
    wire [8:0]  xSum = XH + XL;
    wire [8:0]  ySum = YH + YL;

    //Z0 = XL * YL
    mult8bit m0 (.a(XL),     .b(YL),     .p(Z0));
    //Z2 = XH * YH
    mult8bit m1 (.a(XH),     .b(YH),     .p(Z2));
    //Z1_0= (XH + XL) * (YH + YL)
    mult8bit m2 (.a(xSum[7:0]), .b(ySum[7:0]), .p(Z1_0));
    wire [15:0] Z1 = Z1_0 - Z0 - Z2;

    //final product: P = Z2 << 16 + Z1 << 8 + Z0
    assign P = ({Z2, 16'b0}) + ({Z1, 8'b0}) + Z0;


endmodule
```

تقسیم کننده:

```
module divisor(
    input [15:0] dividend, divisor,
    output reg [15:0] quotient,
    output reg [15:0] remainder
);
    reg [16:0] R; //رجیستر ۱۷-بیتی برای باقی‌مانده به دلیل نیاز به تشخیص علامت
    reg [15:0] Q; //temp qoitient
```

R یک بیت بیشتر از ۱۶ بیت دارد تا علامت باقی‌مانده منفی را بتوان با بیت بالایی ([R[16]) بررسی کرد.

```
    integer i;

    always @(*) begin
        Q = dividend;
        R = 17'b0;
```

مقدار اولیه‌ی Q را برابر با `dividend` قرار می‌دهد. و R از ابتدا صفر است.

```

        for (i = 0; i < 16; i = i + 1) begin
            //{R, Q} << 1
            R = {R[15:0], Q[15]};
            Q = Q << 1;

```

این حلقه ۱۶ بار اجرا می‌شود، هر بار یک بیت از خارج‌قسمت را تولید می‌کند. و ابتدا پر ارزش ترین بیت خارج قسمت که همان مقسوم هست وارد به عنوان کم ارزش ترین بیت باقی مانده قرار میگیرد و سپس Q را یک بیت به چپ شیفت می دهیم که در عمل انگار {R,Q} را یک بیت به چپ شیفت دادیم تا جا برای بیت جدید باز شود که همان بیت های خارج قسمت اند.

```
            //subtract divisor from R
            R = R - divisor;

            if (R[16] == 1'b1) begin
                R = R + divisor;    //restore
                Q[0] = 1'b0;
            end else begin
                Q[0] = 1'b1;
            end
        end

        quotient  = Q;
        remainder = R[15:0];
    end
endmodule
```

سپس مقسوم الیه را از باقی مانده کم می کنیم و بیت علامت R را چک می کنیم اگر یک بود یعنی باقی مانده از مقسوم الیه کوچک تر بوده پس بیت آخر Q که همان بیتی است که شیفت داده شد را برابر 0 قرار می دهیم و R را به حالت قبل از تفریقش از مقسوم الیه بر می گردانیم.و در غیر اینصورت کم ارزش ترین بیت Q را 1 می کنیم.

```

        quotient  = Q;
        remainder = R[15:0];
    end
endmodule

```

و پس از 16 مرحله Q , R آماده اند توجه داریم که فقط 16 بیت کم ارزش R را در remainder می ریزیم . و همچنین در تقسیم اعداد بر 0 خارج قسمت همیشه 1111111111111111 می شود که اگر در alu فلگی بقسمت همیشه 1111111111111111 می شود که اگر در alu فلگی برای تقسیم بر صفر بود مفید بود.

ماژول : ALU

```
module alu (
    input  [2:0]  opcode,
    input  signed [15:0] A, B,
    output reg signed [15:0] result
);
    wire signed [15:0] B_sub;

    assign B_sub = ~B;

    wire [15:0] csa_sum_add, csa_sum_sub;
    wire csa_cout_add, csa_cout_sub;

    //ADD
    CSAadder add_unit (
        .A(A),
        .B(B),
        .Cin(1'b0),
        .sum(csa_sum_add),
        .Cout(csa_cout_add)
    );

    //SUB (A + ~B + 1)
    CSAadder sub_unit (
        .A(A),
        .B(B_sub),
        .Cin(1'b1),
        .sum(csa_sum_sub),
        .Cout(csa_cout_sub)
    );

```

این ماژول دو عدد علامتدار را ورودی میگیرد و براساس opcode عملیات لازم را انتخاب میکند و در result که آن هم علامتدار است می ریزد.

B_sub نات B است که از آن برای عملیات sub استفاده می شود.

Instance هایی از CSAadder می سازیم به عنوان جمع کننده و تفریق کننده. توجه داریم که برای تفریق به مکمل 2 عدد نیاز داریم که از B_sum و کری این 1 استفاده میکنیم به عنوان مکمل کننده و نتیجه این دو unit را در csa_sum_add, csa_sum_sub می ریزیم و Cout شان را در csa_cout_add, csa_cout_sub می ریزیم گرچه که نیازی بهشان نداریم ولی اگر فلگ carry داشتیم آنگاه مفید بود.

```
//MUL
wire [15:0] absA_mul = A[15] ? (~A + 1) : A;
wire [15:0] absB_mul = B[15] ? (~B + 1) : B;
wire [31:0] mul_unsigned;
wire mul_sign = A[15] ^ B[15];

multiplier mul_unit (
    .A(absA_mul),
    .B(absB_mul),
    .P(mul_unsigned)
);

wire [31:0] mul_signed = mul_sign ? (~mul_unsigned + 1) : mul_unsigned;
```

ابتدا با توجه به بیت علامت اعداد اگر 1 بود مکمل دو آن عدد و در غیر اینصورت خود عدد را به عنوان ورودی های ضربکننده قرار می دهیم و به نوعی قدر مطلق میگیریم و mul_sign را به عنوان بیت علامت نتیجه در نظر میگیریم تا اگر یک بود نتیجه را که بدون علامت است قرینه کنیم.

```
//DIV
wire [15:0] absA_div = A[15] ? (~A + 1) : A;
wire [15:0] absB_div = B[15] ? (~B + 1) : B;
wire [15:0] q_unsigned, r_unsigned;
wire div_sign = A[15] ^ B[15];
wire rem_sign = A[15];

divisor div_unit (
    .dividend(absA_div),
    .divisor(absB_div),
    .quotient(q_unsigned),
    .remainder(r_unsigned)
);

wire [15:0] q_signed = div_sign ? (~q_unsigned + 1) : q_unsigned;
wire [15:0] r_signed = rem_sign ? (~r_unsigned + 1) : r_unsigned;
```

برای تقسیم هم ابتدا قدر مطلق می گیریم و توجه داریم که در انتها برای تعیین علامت باقی مانده باید هم علامت با مقسوم باشد و و خارج قسمت با توجه به هم علامت بودن مقسوم و مقسوم الیه مثبت باید باشد و در غیر اینصورت منفی.

برای بهتر شدن alu میتوانستیم فلگ های zero,divisionbyzero,carry,overflow هم داشته باشیم ولی در داک اشاره نشده بود به هر حال پیاده سازی آن ها سخت نیست.

```
    //Final ALU output selection
    always @(*) begin
        case (opcode)
            3'b000,3'b100,3'b101: result = csa_sum_add;         //ADD
            3'b001: result = csa_sum_sub;         //SUB
            3'b010: result = mul_signed[15:0];    //MUL (16 LS bits only)
            3'b011: result = q_signed;            //DIV (quotient only)
            default: result = 16'b0;
        endcase
    end
endmodule
```

و در آخر با توجه به آپکد result را برابر با خروجی واحد مربوطه قرار می دهیم برای opcode های 000 خود جمع و 100 load و101 store عملیات جمع را انتخاب می کنیم.و برای 001 تفریق و برای 010 16 بیت کم ارزش واحد ضرب و برای 011 خارج قسمت واحد تقسیم را انتخاب می کنیم.

رجیستر فایل:

ماژول universal_register:

```
module universal_reg(
    input clk,write_en,reset,
    input [15:0] data_in,
    output reg [15:0] data_out
);
    always @(posedge clk) begin
        if(reset)
            data_out<=16'b0;
        else if(write_en)
            data_out<=data_in;

    end
endmodule
```

این ماژول روی لبه مثبت کلاک خروجی رجیستر را برابر با ورودی می گذارد در صورت فعال بودن write_en. و اگر reset فعال شود برابر 0 می شود.

ماژول :registerFile

```
module registerFile(
    input reset,
    input clk,
    input write_en,
    input [1:0] write_reg,
    input [15:0] write_data,
    input [1:0] read_reg1,read_reg2,
    output reg[15:0] read_data1,read_data2
);
    wire [15:0] reg_out[3:0];
    reg reg_en0, reg_en1, reg_en2, reg_en3;
```

این ماژول دو پورت برای خواندن و یک پورت برای نوشتن دارد و آدرس رجیستر مقصد نوشتن با write_reg دو بیتی و آدرس رجیستر های که باید از روی آن ها بخوانیم با read_reg1 و read_reg2 که هر کدام 2 بیت هستند مشخص می شود. reg_eni تعیین می کند write_en کدام رجیستر فعال شود. و reg out یک آرایه با سایز 4 است که هر کدام 16 بیت هستند که data_out هر یک از instance های universal_reg را نگه داری می کند.

```
13      always @(*) begin
14          reg_en0 = 0;
15          reg_en1 = 0;
16          reg_en2 = 0;
17          reg_en3 = 0;
18
19          if (write_en) begin
20              case (write_reg)
21                  2'b00: reg_en0 = 1;
22                  2'b01: reg_en1 = 1;
23                  2'b10: reg_en2 = 1;
24                  2'b11: reg_en3 = 1;
25              endcase
26          end
27      end
```

اگر write_en فعال باشد، فقط ثبات مشخص‌شده با write_reg فعال می‌شود تا مقدار جدید در آن نوشته شود. و هر بار همه سیگنال های کنترلی reg_en صفر می شوند تا همزمان هیچ یک از آن ها با هم یک نشوند. توجه داریم که اگر write_en فعال نباشد هر داده ای هم در data_in عوض شود باز هم در data_out نمی رود.

```
28
29      universal_reg reg0 (.clk(clk) , .write_en(reg_en0), .data_in(write_data),.data_out(reg_out[0]),.reset(reset));
30      universal_reg reg1 (.clk(clk) , .write_en(reg_en1), .data_in(write_data),.data_out(reg_out[1]),.reset(reset));
31      universal_reg reg2 (.clk(clk) , .write_en(reg_en2), .data_in(write_data),.data_out(reg_out[2]),.reset(reset));
32      universal_reg reg3 (.clk(clk) , .write_en(reg_en3), .data_in(write_data),.data_out(reg_out[3]),.reset(reset));
33
34      always @(negedge clk) begin
35          case (read_reg1)
36              2'b00: read_data1 <= reg_out[0];
37              2'b01: read_data1 <= reg_out[1];
38              2'b10: read_data1 <= reg_out[2];
39              2'b11: read_data1 <= reg_out[3];
40          endcase
41
42          case (read_reg2)
43              2'b00: read_data2 <= reg_out[0];
44              2'b01: read_data2 <= reg_out[1];
45              2'b10: read_data2 <= reg_out[2];
46              2'b11: read_data2 <= reg_out[3];
47          endcase
48      end
49
50  endmodule
```

چهار instance از ماژول universal_regh ساخته شده است که در صورت فعال‌بودن سیگنال write_en و لبه‌ی

بالا رونده کلاک، مقدار ورودی `write_data` را در خود ذخیره می‌کنند. و خروجی ها روی آرایه ی reg_out نوشته می شوند.

و روی لبه منفی کلاک, reg_out رجیستر های مورد نظر که با read_reg1,read_reg2 مشخص شدند روی read_data1 , read_data2 نوشته می شوند.

MEMORY:

```
module memory(
    input clk,
    input we,
    input re,
    input [15:0] address,
    input [15:0] write_data,
    output reg [15:0] read_data
);

```

ورودی های we, re برای کنترل خواندن و نوشتن هستند و اگر 1 باشند خواندن یا نوشتن اتفاق می افتد.address آدرس خانه ای از حافظه را مشخص میکند که باید از آن بخوانیم یا روی آن بنویسیم و write_data داده ای است که باید بنویسیم روی حافظه و مربوط به دستور STORE هست و read_data خروجی ای است که یا داده ای را از حافظه خروجی می دهیم که مربوط به دستور load هست و یا دستوری را fetch میکنیم و خروجی می دهیم.

```
    parameter PROGRAM_END_ADDR = 16'd31;  //Addresses below this are reserved for instruction
```

این پارامتر آدرس پایان فضای برنامه را مشخص می‌کند.و آدرس‌های ۰ تا ۳۱ برای کد برنامه (Instruction Memory) رزرو شده‌اند و اجازه نوشتن در آن‌ها وجود ندارد.فضای داده (Data Memory) از آدرس ۳۲ به بعد شروع می‌شود. این محدودیت برای این است که دستور store برنامه را خراب نکند.

```
    reg [15:0] memword [0:65535];

    always @(posedge clk) begin
        if (we && address > PROGRAM_END_ADDR) begin
            memword[address] <= write_data;
        end

    end

    always @(negedge clk) begin
        if (re)begin
            read_data <= memword[address];
        end
    end
endmodule

```

memword همان حافظه ما است که 16^2 آدرس دارد و هر خانه حافظه 2 بایت یا 16 بیتی است. و روی لبه منفی ساعت اگر we فعال باشد و آدرس کمتر از 32 نباشد تا حافظه دستور خراب نشود روی آن آدرس از حافظه داده را می نویسیم و اگر re فعال باشد داده آن خانه از حافظه را میخوانیم این داده هم میتواند مربوط به حافظه دستور باشد که address همان pc است و هم میتواند مربوط به حافظه داده باشد برای دستور load.

دلیل اینکه عملیات نوشتن روی لبه مثبت انجام می شود این است که عملیات خواندن از روی رجیستر روی لبه منفی انجام می شود و در کل مرحله MEM دستور STORE در control_unit یک کلاک طول میکشد و فقط در همان کلاک we مموری فعال می شود و اگر Write مموری روی لبه مثبت باشد آنگاه مشکل به وجود می آید.
و دلیل اینکه عملیات خواندن روی لبه منفی انجام می شود این است که برای دستور load چون نوشتن روی رجیستر در لبه مثبت انجام می شود در نتیجه بلافاصله بعد از خواندن از مموری روی رجیستر نوشته می شود و همه این ها در یک دوره تناوب انجام میشود و مشکلی به وجود نمی آید.

ماژول :Control unit

```
module control_unit (
    input clk,
    input reset,
    input [15:0] instruction_memory_out,
    output reg [15:0] pc,
    output reg [2:0] alu_op,
    output reg [1:0] rd, rs1, rs2,
    output reg mem_read,
    output reg mem_write,
    output reg reg_write,
    output reg mem_instr,
    output reg alu_src,      // if 0 from reg, if 1 from sign extended immediate
    output reg mem_to_reg,   // if 1 data comes from memory else from register
    output reg ready,
    output reg halted,
    output reg [2:0] current_state
);

    //Define FSM states
   localparam[2:0] FETCH = 3'b000,
            DECODE       = 3'b001,
            EXECUTE      = 3'b010,
            MEMORY       = 3'b011,
            WRITEBACK    = 3'b100,
            INCREMENT_PC = 3'b101,
            HALT         = 3'b110;

    reg[2:0] p_state, n_state;

```

واحد کنترلی instruction_memory_out را از datapath میگیرد.(در دیتاپث آدرس pc به مموری داده می شود و memword[pc] از مموری fetch میشود و در instruction_memory_out ماژول دیتا پث ریخته می شود و به control_unit ورودی داده می شود) و pc در این ماژول آپدیت می شود و به data path داده می شود تا دستور را از مموری fetch کند.

alu_op:همان opcode است که پس از دیکود دستور معلوم می شود.

rd,rs1,rs2:پس از دیکود دستور مشخص می شوند.

mem_read: سیگنال کنترلی برای مموری در دستورLOAD فعال می شود.

mem_write: سیگنال کنترلی برای مموری در دستورSTORE فعال می شود.

reg_write: سیگنال کنترلی برای رجیستر در دستورهای R_type فعال می شود.

alu_source:سیگنال کنترلی برای پورت دوم ALu که یا از رجیستر باشد یا برای محاسبه آدرس از واحد sign extend .

mem_to_reg:در مرحله Write back تعیین می کند داده از رجیستر و یا داده ازممموری روی رجیستر مقصد نوشته شود.

ready:پس از اتمام پردازش دستور pc را 1 عدد زیاد کرده وبه مرحله Fetch دستور بعد می رویم.

halted: زمانی که دستورات برنامه تمام شود 1 می شود.

current_state:حالت فعلی را خروجی می دهد تا در data path استفاده شود.

در ادامه حالت ها را تعیین میکنیم. هر دستور در چند حالت اجرا می‌شود. مثلاً:

دستور های R-type:

FETCH → DECODE → EXECUTE → WRITEBACK → INCREMENT_PC

دستور های M-type:

FETCH → DECODE →EXECUTE→ MEMORY → WRITEBACK(LOAD only) → INCREMENT_PC

و p_state حالت فعلی و n_state حالت بعدی است تا بتوانیم از حالتی به حالت دیگر برویم(لازمه FSM).

```
    //Instruction fields
    wire [2:0] opcode = instruction_memory_out[15:13];
    wire [1:0] dest   = instruction_memory_out[12:11];
    wire [1:0] src1   = instruction_memory_out[10:9];
    wire [1:0] src2   = instruction_memory_out[8:7];

    reg next_ready, next_halted;

    //Sequential logic
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            pc <= 0;
            p_state <= FETCH;
            ready <= 0;
            halted <= 0;
            current_state <= FETCH;
        end else begin
            p_state <= n_state;
            ready <= next_ready;
            halted <= next_halted;
            current_state <= n_state;
            if (p_state == INCREMENT_PC)
                pc <= pc + 1;
        end
    end

```

اطلاعات اصلی مثل opcode، مقصد، و دو منبع از بیت‌های دستور استخراج می‌شوند.

دلیل استفاده از next_halted , next_ready این است که نمی توانیم همزمان هم مقدارشان را در always ترکیبی و هم در always ترتیبی تغییر دهیم پس از این دو به عنوان متغیر میانی استفاده میکنیم.

در بلاک ترتیبی ااگر ریست فعال باشد، پردازنده به FETCH برمی‌گردد. دلیل اینکه pc را 0 میکنیم این است که به اولین خانه حافظه اشاره کند که اولین دستور برنامه در آن قرار دارد. و در غیر اینصورت در حالت عادی، حالت بعدی به حالت فعلی اختصاص می‌یابد. در INCREMENT_PC مقدار pc افزایش می‌یابد.

```
    //Combinational logic
    always @(*) begin
        //default
        alu_op     = 3'b000;
        alu_src    = 0;
        mem_read   = 0;
        mem_write  = 0;
        reg_write  = 0;
        mem_to_reg = 0;
        next_ready  = 0;
        next_halted = 0;

        rd  = dest;
        rs1 = src1;
        rs2 = src2;

        n_state = FETCH;

        case (p_state)
            FETCH: begin
                n_state = DECODE;
            end

            DECODE: begin
                case (opcode)
                    3'b000, 3'b001, 3'b010, 3'b011: n_state = EXECUTE;
                    3'b100, 3'b101: n_state = EXECUTE;
                    3'b110: n_state = HALT;
                    default: n_state = INCREMENT_PC;
                endcase
            end

            EXECUTE: begin
                alu_op = opcode;
                alu_src = (opcode == 3'b100 || opcode == 3'b101); //used as a select line for alu input mux
                case (opcode)
                    3'b000, 3'b001, 3'b010, 3'b011: n_state = WRITEBACK;
                    3'b100, 3'b101: n_state = MEMORY;
                    default: n_state = INCREMENT_PC;
                endcase
            end

            MEMORY: begin
                if (opcode == 3'b100) begin  //LOAD
                    mem_read = 1;
                    mem_to_reg = 1;
                    reg_write = 1;
                    n_state = INCREMENT_PC;
                end else if (opcode == 3'b101) begin  //STORE
                    rs2=dest;
                    mem_write = 1;
                    n_state = INCREMENT_PC;
                end
            end

            WRITEBACK: begin
                reg_write = 1; //mem_to_reg is already zero
                n_state = INCREMENT_PC;
            end

            INCREMENT_PC: begin
                next_ready = 1;
                n_state = FETCH;
            end

            HALT: begin
                next_ready = 1;
                next_halted = 1;
                n_state = HALT;
            end
        endcase
    end
endmodule
```

در مدار ترکیبی بر اساس حالت فعلی p_state، سیگنال‌های کنترلی مقداردهی می‌شوند و n_state مشخص می‌شود.

:FETCH

درخواست خواندن دستور از حافظه مرحله بعد Decode است.

DECODE:

تعیین مسیر بعدی بر اساس opcode توجه کنید که یک opcode خودم اضافه کردم به جز اونایی که تو داک بود که اگر opcode==110 باشد به معنای اتمام دستورات برنامه است و به حالت HALT می رویم.

EXECUTE:

پاس دادن opcode به عنوان alu_op. اگر دستور LOAD یا STORE باشد، ورودی دوم ALU از ایمیدیت می‌آید.

MEMORY:

اگر LOAD: از حافظه بخوان و در رجیستر بنویس
اگر STORE: مقدار را از رجیستر بگیر و در حافظه بنویس.و چون در data path رجیستری که مقدارش در حافظه نوشته می شود rs2 هست و در دستور store رجیستر dest همان است باید . rs2=dest

WRITEBACK:

برای دستورات R_type نتیجه‌ی ALU را در رجیستر مقصد ذخیره می‌کند.

INCREMENT_PC:

آماده برای دستور بعدی، next_ready را یک می کنیم تا ready در کلاک بعد 1 شود.به fetch دستور بعد می رویم.

HALT:

اجرای پردازنده متوقف می‌شود.و فلگ halted یک می شود. یعنی دستورات تمام شده.و همیشه در همین حالت باقی می مانیم.

ماژول :datapath

این ماژول تمام اجزا و واحد های پردازنده را به هم متصل می کند و سیگنال های کنترلی را از کنترل یونیت می گیرد وبا توجه به آن ها راجع به ورودی های یونیت ها تصمیم میگیرد و از مموری با توجه به pc فعلی دستور را می خواند و به کنترل یونیت می دهد.

ماژول signExtend:

```
module signExtend(
    input  [8:0] in,
    output [15:0] out
);
    assign out = {{7{in[8]}}, in};
endmodule

```

مقدار immediate که 9 بیتی هست رو با توجه به بیت علامت به 16 بیت گسترش می دهد.

```
c: > Users > reaka > OneDrive > Desktop > modelsim > = datapath.v
module datapath(
    input clk,
    input reset,
    input [2:0] alu_op,
    input [1:0] rd, rs1, rs2,
    input mem_read,
    input mem_write,
    input reg_write,
    input alu_src,
    input mem_to_reg,
    input [2:0] current_state,
    input [15:0] pc_in,
    output [15:0] instruction_memory_out
);

    //state identifiers for clarity
    localparam FETCH = 3'b000,
               EXECUTE = 3'b010,
               MEMORY = 3'b011,
               WRITEBACK = 3'b100;

    //memory Input and outputs
    wire [15:0] mem_read_data;
    wire [15:0] mem_address;
    wire [15:0] mem_write_data;
    wire mem_read_enable;

    memory mem (
        .clk(clk),
        .we(mem_write),
        .re(mem_read_enable),
        .address(mem_address),
        .write_data(mem_write_data),
        .read_data(mem_read_data)
    );
```

همانطور که مشخص است این ماژول سیگنال های کنترلی و pc و حالت فعلی را از control_unit گرفته و دستور را خروجی داده است.

سپس برای خوانایی بیشتر و عدم استفاده از کد ها برای چک کردن حالت های فعلی حالت ها را تعریف کردیم.

و سپس یک instance از ماژول مموری ساختیم و آن را در دیتاپث متصل کردیم. سیم هایی که به عنوان ورودی های مموری تعریف کردیم در خطوط بعدی با توجه به سیگنال ها تعیین و assign می شوند.. آدرس حافظه می‌تواند برای

واکشی دستور (FETCH) یا عملیات LOAD/STORE استفاده شود. سیگنال mem_read_enable برای واکشی یا خواندن داده فعال می شود.

```
36
37      //register File
38      wire [15:0] reg_data1, reg_data2;
39      wire [15:0] write_back_data;
40
41      registerFile reg_file (
42          .clk(clk),
43          .write_en(reg_write),
44          .write_reg(rd),
45          .write_data(write_back_data),
46          .read_reg1(rs1),
47          .read_reg2(rs2),
48          .read_data1(reg_data1),
49          .read_data2(reg_data2),
50          .reset(reset)
51      );
52
```

ابتدا یک instance از registerFile میسازیم و آن را به دیتا پث وصل می کنیم. rs1 و rs2 رجیسترهایی هستند که برای عملوندها خوانده می‌شوند.rd رجیستری است که در مرحله‌ی WRITEBACK در آن مقدار نوشته می‌شود.(STORE در کنترل یونیت هندل شده است)ورودی write_data می‌تواند نتیجه ALU یا داده بارگذاری شده از حافظه باشد.داده های خوانده شده در reg_data1 , reg_data2 ذخیره می شوند.

```
53      //sign Extend for offset (used in LOAD/STORE)
54      wire [15:0] extended_offset;
55      signExtend extender (
56          .in(instruction_memory_out[8:0]),
57          .out(extended_offset)
58      );
59
60      //ALU Input Selection
61      wire [15:0] alu_input_B = alu_src ? extended_offset : reg_data2;
62      wire [15:0] alu_result;
63
64      alu alu (
65          .opcode(alu_op),
66          .A(reg_data1),
67          .B(alu_input_B),
68          .result(alu_result)
69      );
70      reg [15:0] alu_out;
71
72      always @(posedge clk) begin
73          if (reset)
74              alu_out <= 16'b0;
75          else if (current_state == EXECUTE)
76              alu_out <= alu_result;
77      end
78
79
```

ابتدا 9 بیت آخر دستور را با واحد signExtend با حفظ علامت گسترش می دهیم تا برای دستورات LOAD/STORE که آفست دارند بتوانیم آن را با استفاده از ALU که 16 بیتی هست با مقدار رجیستر base جمع کنیم. و پورت دومALU با توجه به سیگنال alu_src بین خروجی signExtend و داده خوانده شده از رجیستر خوانده شده دوم انتخاب می

شود.و خروجی ALU فقط در مرحله EXECUTE در یک رجیستر ذخیره می‌شود تا در مراحل بعدی مثل MEMORY یا WRITEBACK قابل استفاده باشد و در کلاک های بعد عوض نشود.

```
    //Memory Addressing (Fetch = pc, otherwise = ALU result =IMMEDIAT + R[BASE])
    assign mem_address = (current_state == FETCH) ? pc_in : alu_out;

    //enable memory read if FETCH or LOAD
    assign mem_read_enable = (current_state == FETCH) || mem_read;

    //always store reg_data2 to memory on STORE
    assign mem_write_data = reg_data2;

    assign write_back_data = mem_to_reg ? mem_read_data : alu_out;

    //send out instruction fetched from memory
    reg [15:0] instruction_reg;

    always @(posedge clk or posedge reset) begin
        if (reset)
            instruction_reg <= 16'b0;
        else if (current_state== FETCH)
            instruction_reg <= mem_read_data;
    end

    assign instruction_memory_out = instruction_reg;


endmodule
```

mem_address:اگر در مرحله واکشی هستیم، آدرس حافظه همان مقدار PC است. در غیر این صورت از خروجی ALU استفاده می‌شود (مثلاً در LOAD و STORE که آدرس محاسبه شده‌اند).

`mem_read_enable`: فعال در واکشی و LOAD.

`mem_write_data`: در STORE، مقدار از رجیستر دوم در حافظه نوشته می‌شود.که در کنترل یونیت هندل شده و در اصل همان rsource هست.

داده‌ای که باید به رجیستر نوشته شود:

اگر `mem_to_reg` فعال باشد نتیجه از حافظه می‌آید (LOAD). و در غیر این صورت نتیجه از ALU می‌آید.

در مرحله FETCH، دستور از حافظه گرفته و در یک رجیستر ذخیره می‌شود.تا در کلاک های بعد عوض نشود از آنجایی که این پردازنده چند چرخه ای هست.خروجی `instruction_memory_out`، محتوای فعلی دستور در اجرای آن است.

ماژول :systemtop

این ماژول کنترل یونیت و دیتاپث را به هم وصل می کند و بالاترین لایه پردازنده هست.

```
module system(
    input clk,
    input reset,
    output halted
);

    wire [15:0] instruction;
    wire [15:0] pc;
    wire [2:0] alu_op;
    wire [1:0] rd, rs1, rs2;
    wire mem_read, mem_write, reg_write;
    wire alu_src, mem_to_reg;
    wire [2:0] current_state;
    wire cu_halted;

    // Wires for ready signal if needed externally
    wire ready;

    // Control Unit instantiation
    control_unit CU (
        .clk(clk),
        .reset(reset),
        .instruction_memory_out(instruction),
        .pc(pc),
        .alu_op(alu_op),
        .rd(rd),
        .rs1(rs1),
        .rs2(rs2),
        .mem_read(mem_read),
        .mem_write(mem_write),
        .reg_write(reg_write),
        .alu_src(alu_src),
        .mem_to_reg(mem_to_reg),
        .ready(ready),
        .halted(cu_halted),
        .current_state(current_state)
    );

    // Datapath instantiation
    datapath DP (
        .clk(clk),
        .reset(reset),
        .alu_op(alu_op),
        .rd(rd),
        .rs1(rs1),
        .rs2(rs2),
        .mem_read(mem_read),
        .mem_write(mem_write),
        .reg_write(reg_write),
        .alu_src(alu_src),
        .mem_to_reg(mem_to_reg),
        .current_state(current_state),
        .pc_in(pc),
        .instruction_memory_out(instruction)
    );

    assign halted=cu_halted;
endmodule
```

ماژول تست بنچ:

```
module tb_processor;

    reg clk = 0;
    reg reset = 1;
    wire halted;

    system uut (
        .clk(clk),
        .reset(reset),
        .halted(halted)
    );

    always #5 clk = ~clk;

    initial begin
        $display("starting simulation...");
        $readmemb("program.mem", uut.DP.mem.memword);

        #10 reset = 0;
    end

    always @(*) begin
        if (halted) begin
            $display("%t Processor halted.", $time);
            $finish;
        end
    end

endmodule
```

در این ماژول یک instance از بالاترین لایه پردازنده که همه اجزا را به هم متصل کرده یعنی system می سازیم. و یک کلاک با دوره تناوب 10 می سازیم. و ریست را یک می کنیم تا pc صفر شود و سایر یونیت ها هم همینطور.

فایل program.mem که حاوی دستورالعمل‌ها و داده‌هاست، در مموری ماژول system لود می‌شود.
مسیر دقیق حافظه داخل ساختار ماژول:

uut ← نمونه ماژول system

DP ←نمونه ماژول datapath در system

mem → نمونه ماژول memory در دیتاپث

memword → آرایه حافظه داخلی

بعد از ۱۰ واحد زمانی، سیگنال ریست صفر می‌شود → شروع اجرای واقعی برنامه.

بلوک always @(*)همیشه وقتی هر سیگنالی تغییر کند اجرا می‌شود . اگر `halted == 1` باشد، یعنی پردازنده اجرای برنامه را تمام کرده و متوقف میکند و پیام اتمام را پرینت می کند به همراه مدت زمان اجرا.

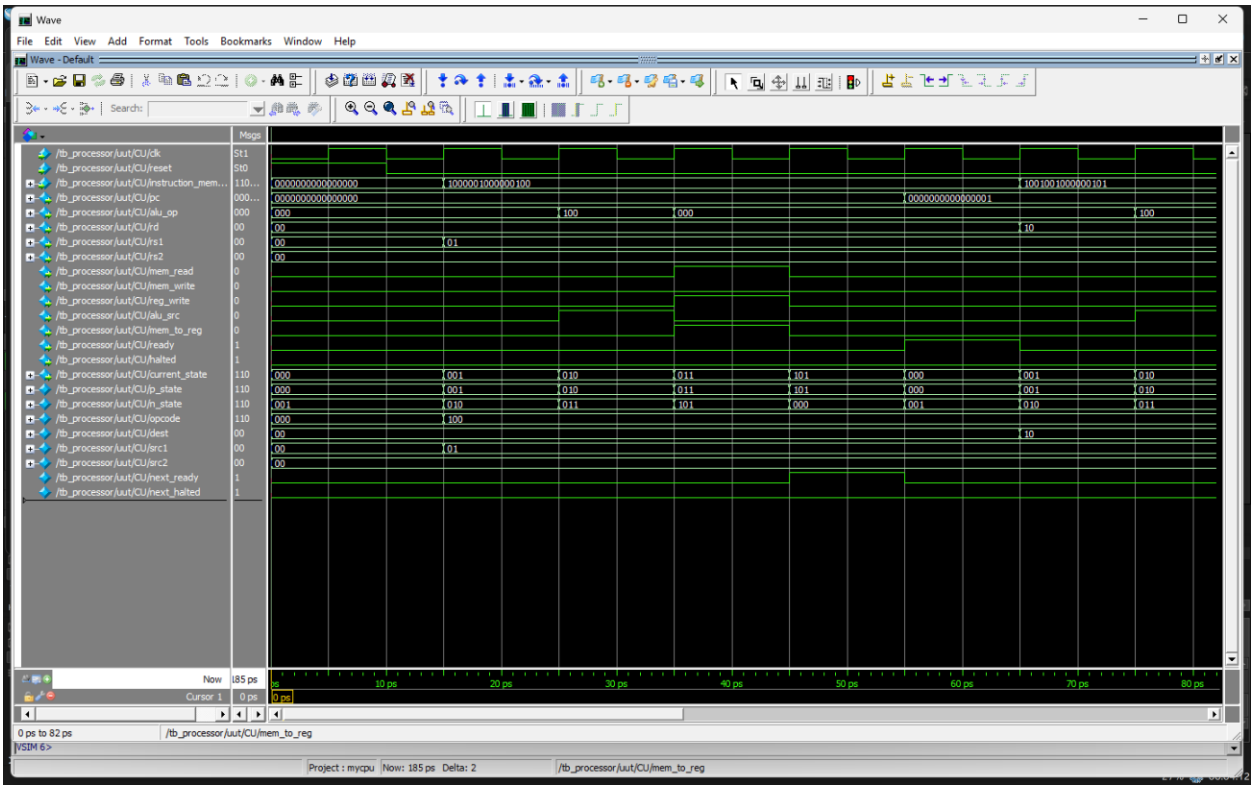
تست کیس جمع دو عدد منفی :

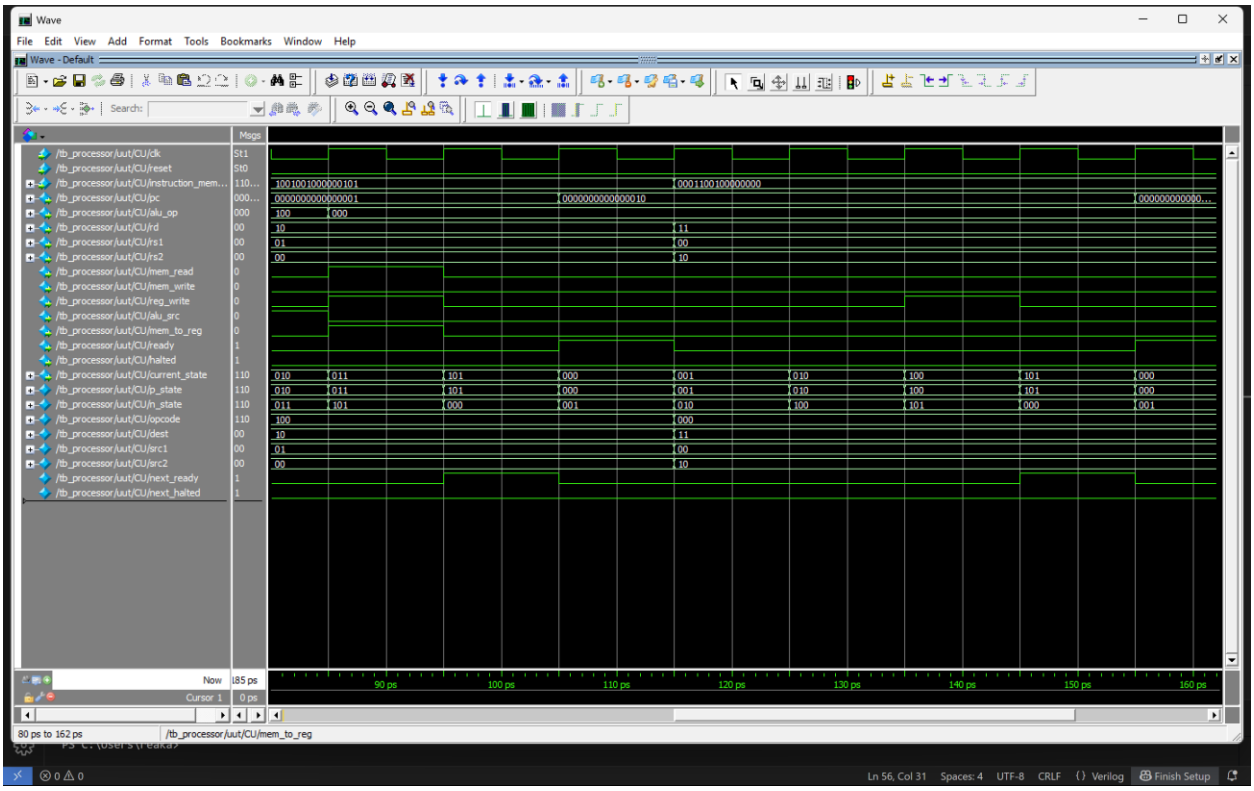
-8+ (-4)

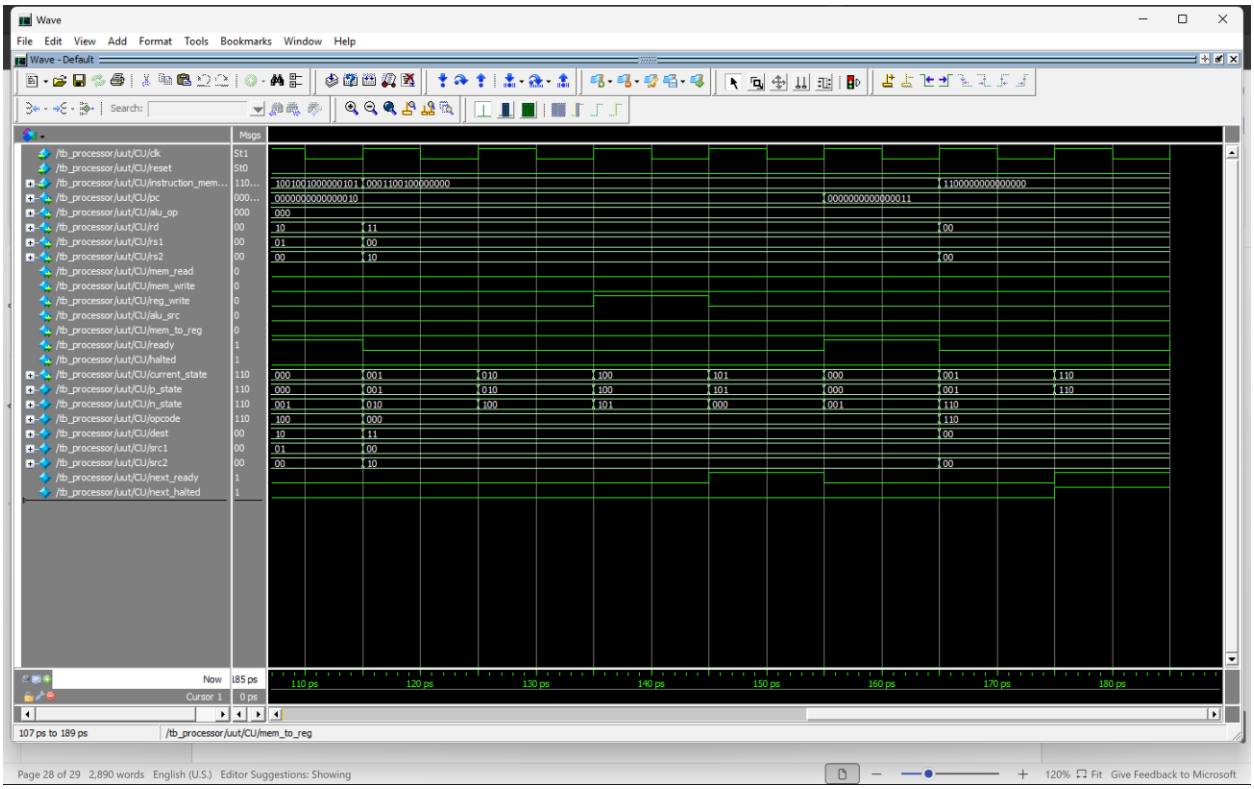
```
1000001000000100 //load r0,r1(000000100) r1 is 0
1001001000000101 //load r2,r1(000000101)
0001100100000000 //add r3=r2+r0
1100000000000000 //halt
1111111111111000 //data1 should be loaded into r0 = -8
1111111111111100 //data2 should be loaded into r2 = -4
```

نتایج :waveform

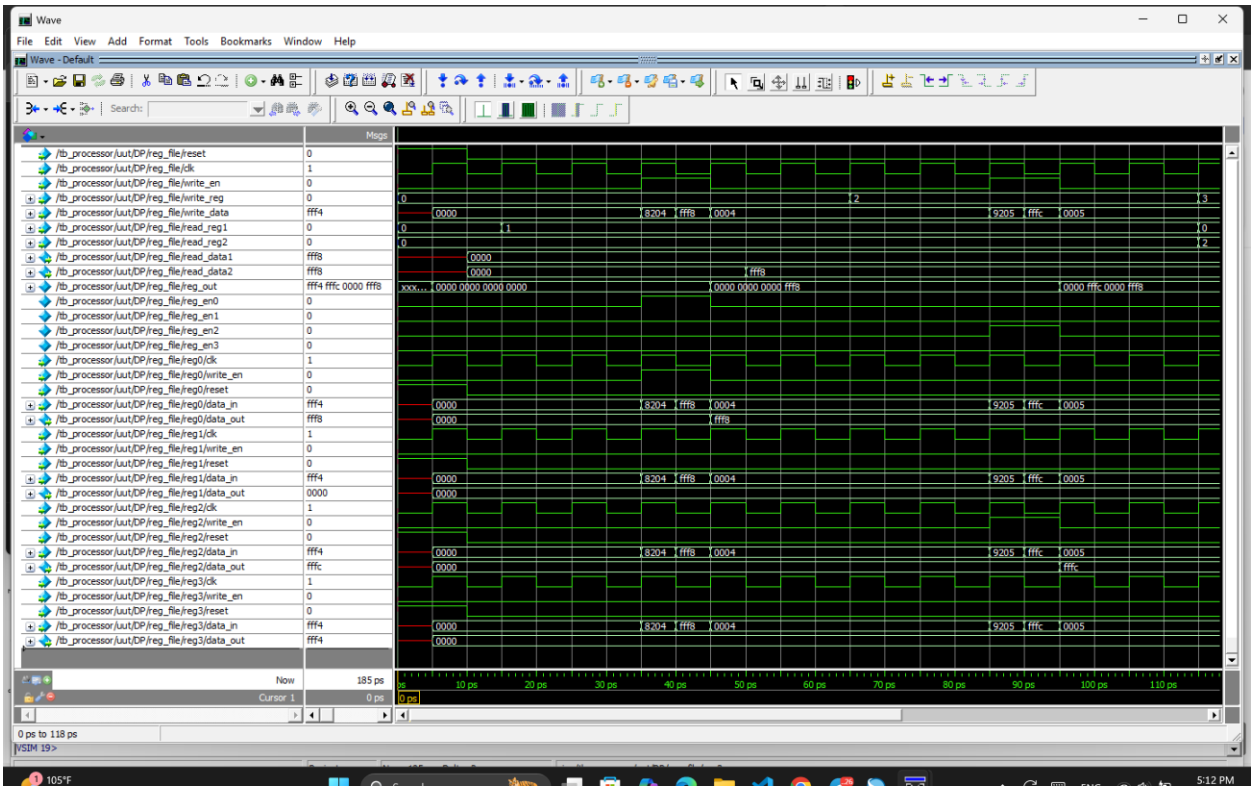
CU:

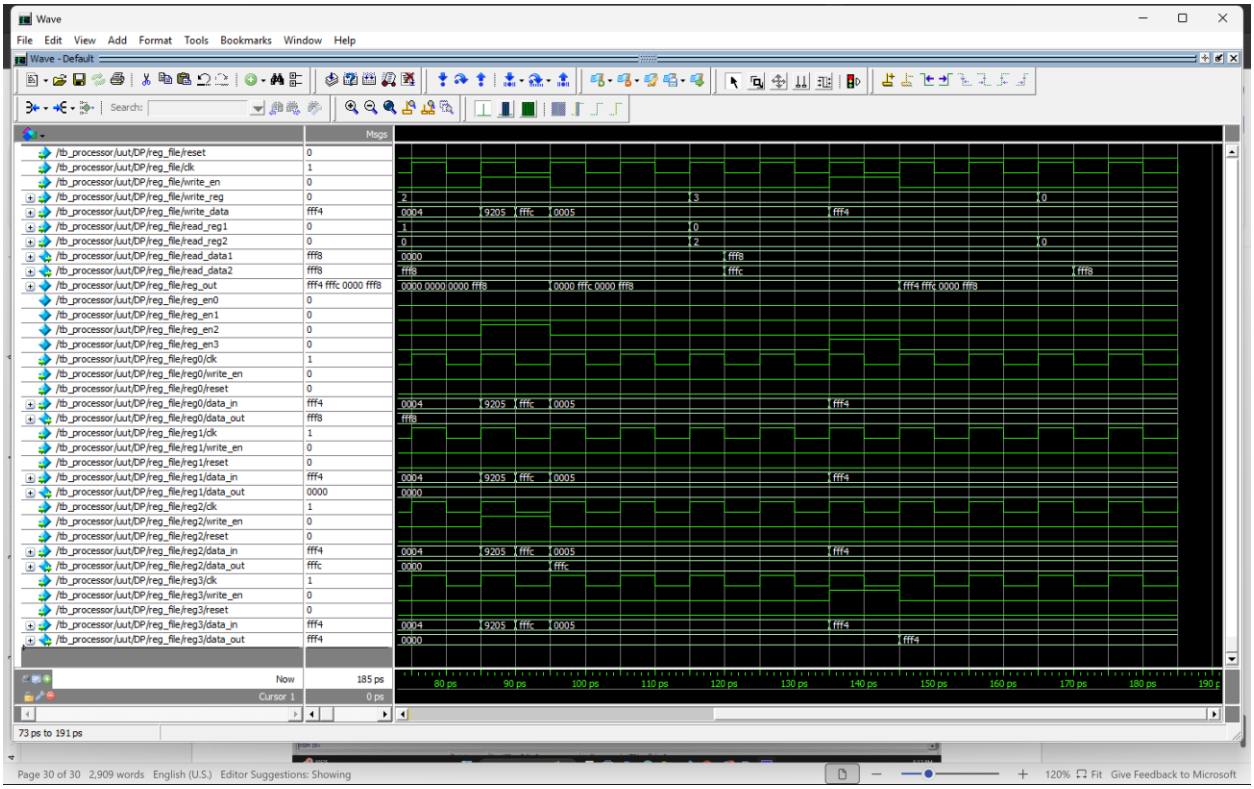

Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/tb_processor/uut/CU/clk
/tb_processor/uut/CU/reset
/tb_processor/uut/CU/instruction_mem...
/tb_processor/uut/CU/pc
/tb_processor/uut/CU/alu_op
/tb_processor/uut/CU/rd
/tb_processor/uut/CU/rs1
/tb_processor/uut/CU/rs2
/tb_processor/uut/CU/mem_read
/tb_processor/uut/CU/mem_write
/tb_processor/uut/CU/reg_write
/tb_processor/uut/CU/alu_src
/tb_processor/uut/CU/mem_to_reg
/tb_processor/uut/CU/ready
/tb_processor/uut/CU/halted
/tb_processor/uut/CU/current_state
/tb_processor/uut/CU/p_state
/tb_processor/uut/CU/n_state
/tb_processor/uut/CU/opcode
/tb_processor/uut/CU/dest
/tb_processor/uut/CU/src1
/tb_processor/uut/CU/src2
/tb_processor/uut/CU/next_ready
/tb_processor/uut/CU/next_halted
Now 185 ps
Cursor 1 0 ps
0 ps to 82 ps
/tb_processor/uut/CU/mem_to_reg
VSIM 6>
Project : mycpu Now: 185 ps Delta: 2


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
/tb_processor/uut/CU/clk
/tb_processor/uut/CU/reset
/tb_processor/uut/CU/instruction_mem...
/tb_processor/uut/CU/pc
/tb_processor/uut/CU/alu_op
/tb_processor/uut/CU/rd
/tb_processor/uut/CU/rs1
/tb_processor/uut/CU/rs2
/tb_processor/uut/CU/mem_read
/tb_processor/uut/CU/mem_write
/tb_processor/uut/CU/reg_write
/tb_processor/uut/CU/alu_src
/tb_processor/uut/CU/mem_to_reg
/tb_processor/uut/CU/ready
/tb_processor/uut/CU/halted
/tb_processor/uut/CU/current_state
/tb_processor/uut/CU/p_state
/tb_processor/uut/CU/n_state
/tb_processor/uut/CU/opcode
/tb_processor/uut/CU/dest
/tb_processor/uut/CU/src1
/tb_processor/uut/CU/src2
/tb_processor/uut/CU/next_ready
/tb_processor/uut/CU/next_halted
Now 185 ps
Cursor 1 0 ps
80 ps to 162 ps
/tb_processor/uut/CU/mem_to_reg
Ln 56, Col 31 Spaces: 4 UTF-8 CRLF Verilog Finish Setup

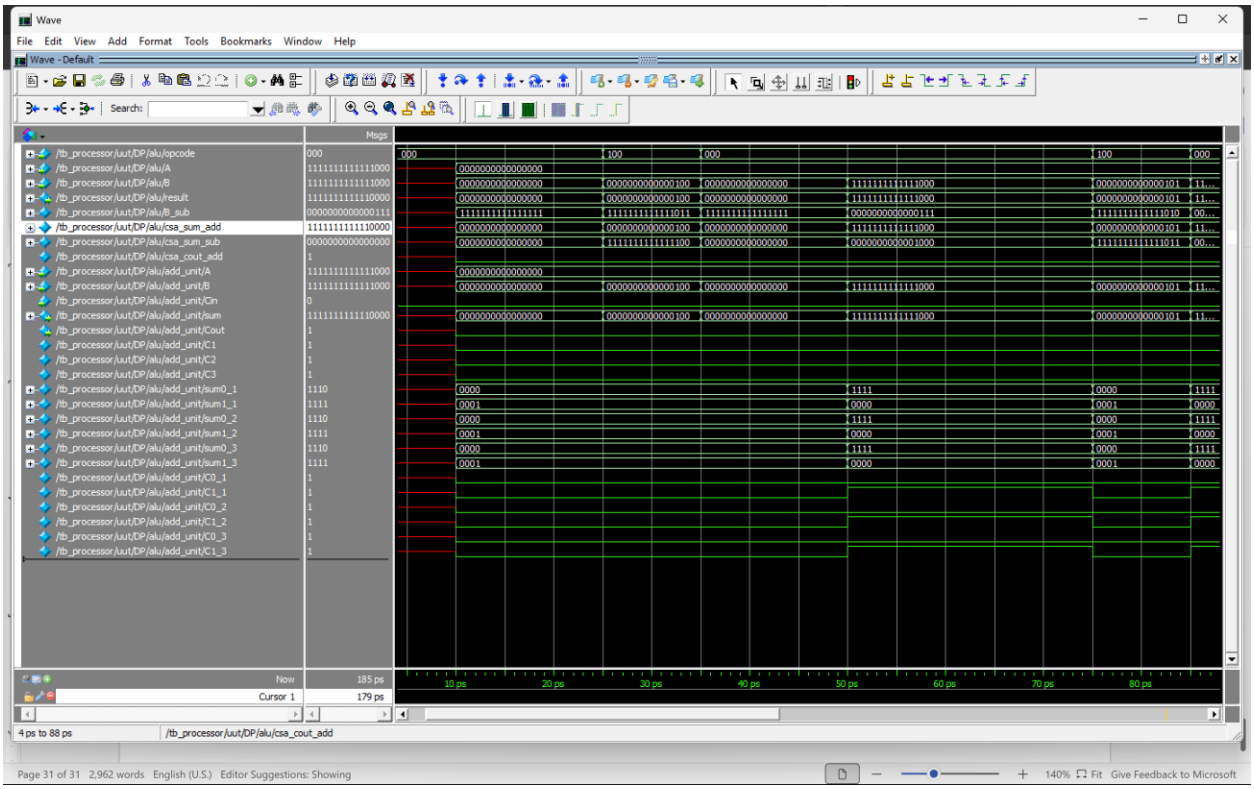


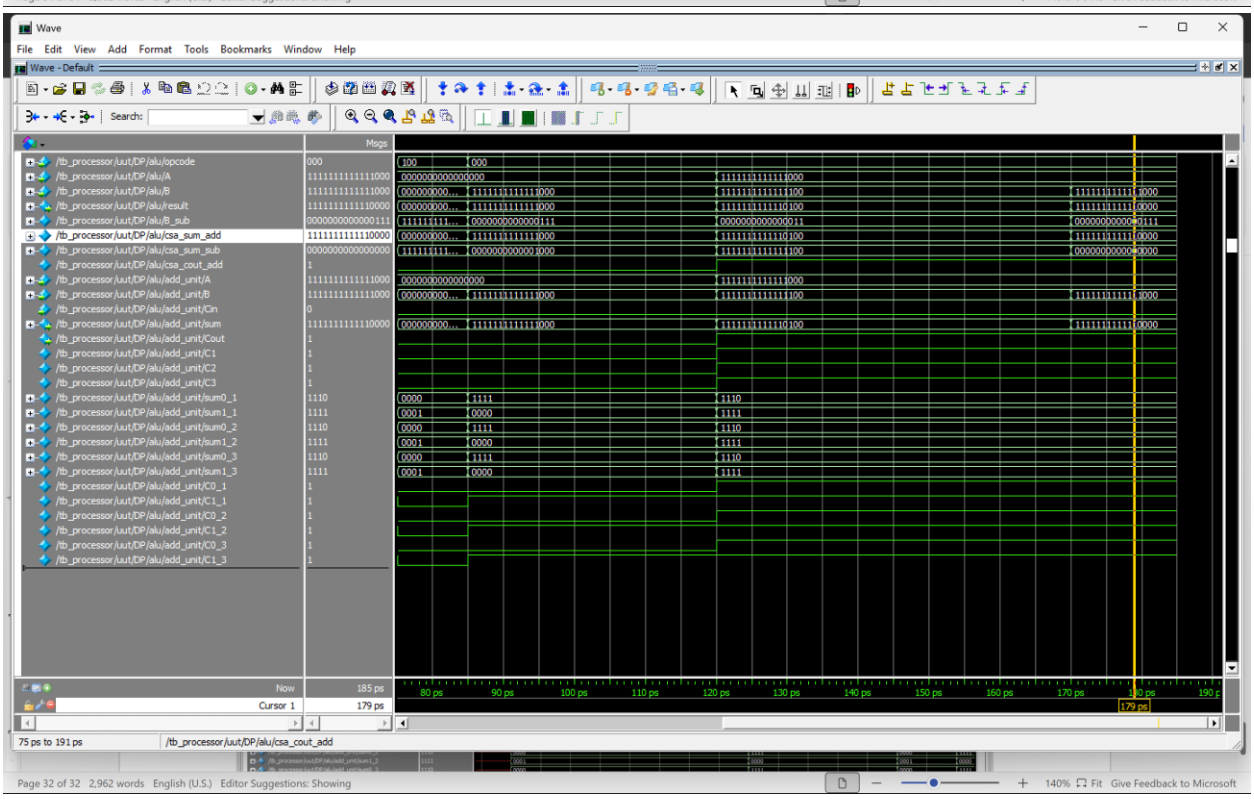
رجیسترفایل و 4 نمونه از universal_register: همانطور که در عکس آخر معلوم است data_out رجیستر 3 همان -12یا FFF4 است.



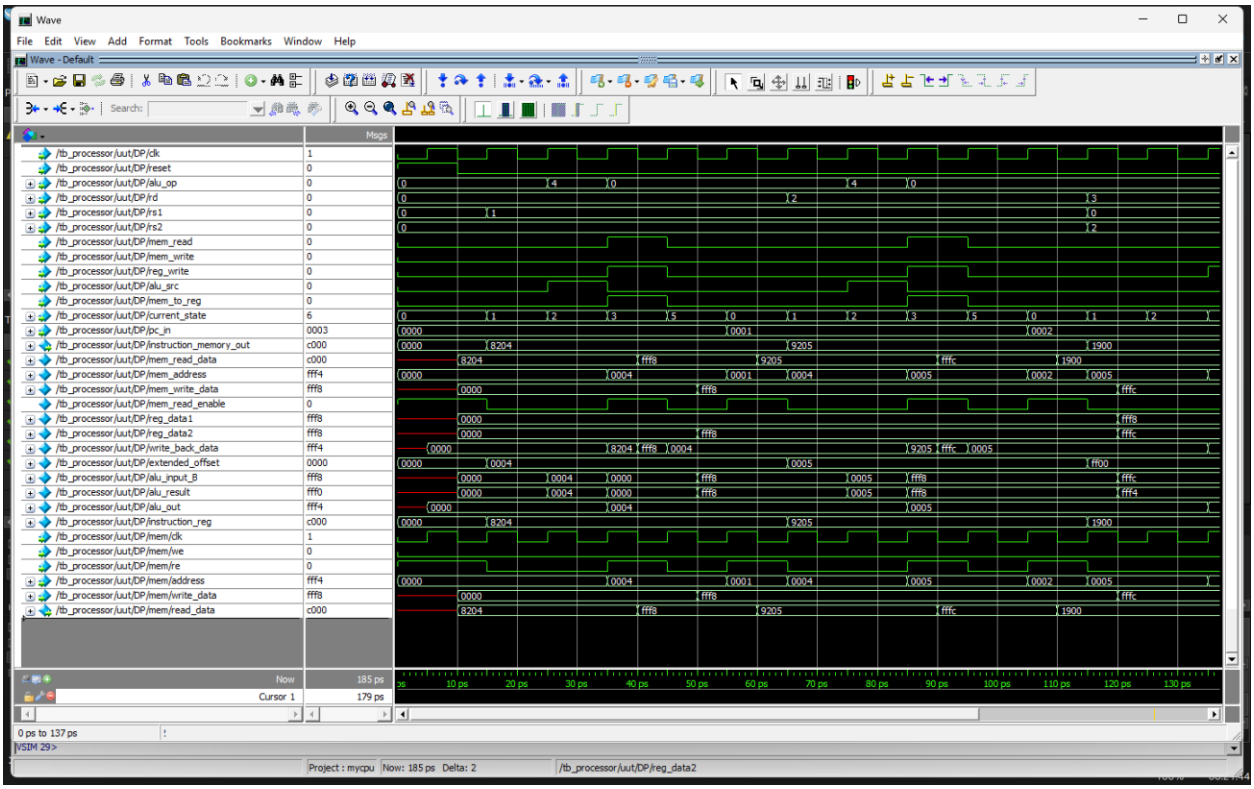


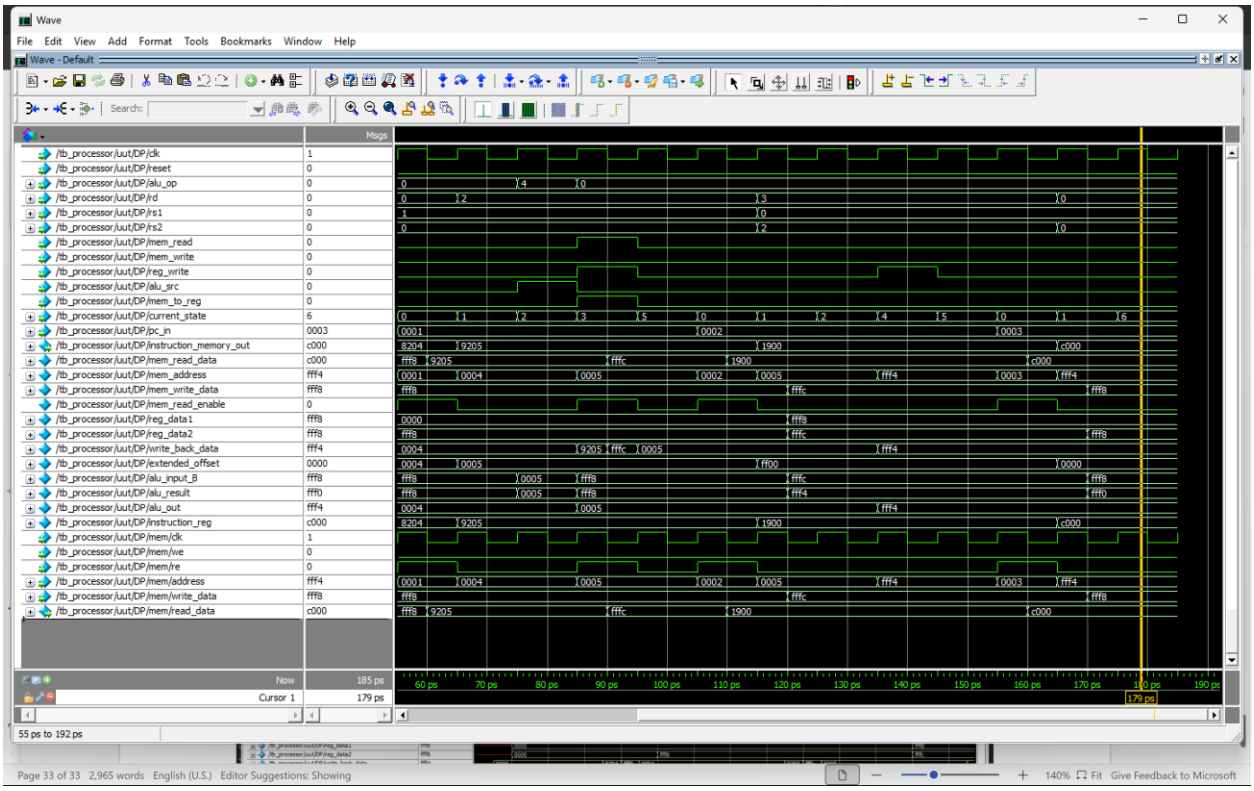
ALU و CSA adder : توجه داریم که alu یک مدار ترکیبی است و به طور مداوم ورودی هایش عوض می شود ولی ما با سیگنال های کنترلی مناسب فقط نتایج مورد نیاز را در رجیستر ها یا مموری ذخیره می کنیم و یا از نتیجه به عنوان آدرس استفاده می کنیم.


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_processor/uut/DP/alu/opcode
/tb_processor/uut/DP/alu/A
/tb_processor/uut/DP/alu/B
/tb_processor/uut/DP/alu/result
/tb_processor/uut/DP/alu/B_sub
/tb_processor/uut/DP/alu/csa_sum_add
/tb_processor/uut/DP/alu/csa_sum_sub
/tb_processor/uut/DP/alu/csa_cout_add
/tb_processor/uut/DP/alu/add_unit/A
/tb_processor/uut/DP/alu/add_unit/B
/tb_processor/uut/DP/alu/add_unit/Cin
/tb_processor/uut/DP/alu/add_unit/sum
/tb_processor/uut/DP/alu/add_unit/Cout
/tb_processor/uut/DP/alu/add_unit/C1
/tb_processor/uut/DP/alu/add_unit/C2
/tb_processor/uut/DP/alu/add_unit/C3
/tb_processor/uut/DP/alu/add_unit/sum0_1
/tb_processor/uut/DP/alu/add_unit/sum1_1
/tb_processor/uut/DP/alu/add_unit/sum0_2
/tb_processor/uut/DP/alu/add_unit/sum1_2
/tb_processor/uut/DP/alu/add_unit/sum0_3
/tb_processor/uut/DP/alu/add_unit/sum1_3
/tb_processor/uut/DP/alu/add_unit/C0_1
/tb_processor/uut/DP/alu/add_unit/C1_1
/tb_processor/uut/DP/alu/add_unit/C0_2
/tb_processor/uut/DP/alu/add_unit/C1_2
/tb_processor/uut/DP/alu/add_unit/C0_3
/tb_processor/uut/DP/alu/add_unit/C1_3
Now
185 ps
Cursor 1
179 ps
10 ps
20 ps
30 ps
40 ps
50 ps
60 ps
70 ps
80 ps
4 ps to 88 ps
/tb_processor/uut/DP/alu/csa_cout_add
Page 31 of 31 2,962 words English (U.S.) Editor Suggestions: Showing
140% Fit Give Feedback to Microsoft


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_processor/uut/DP/alu/opcode
/tb_processor/uut/DP/alu/A
/tb_processor/uut/DP/alu/B
/tb_processor/uut/DP/alu/result
/tb_processor/uut/DP/alu/B_sub
/tb_processor/uut/DP/alu/csa_sum_add
/tb_processor/uut/DP/alu/csa_sum_sub
/tb_processor/uut/DP/alu/csa_cout_add
/tb_processor/uut/DP/alu/add_unit/A
/tb_processor/uut/DP/alu/add_unit/B
/tb_processor/uut/DP/alu/add_unit/Cin
/tb_processor/uut/DP/alu/add_unit/sum
/tb_processor/uut/DP/alu/add_unit/Cout
/tb_processor/uut/DP/alu/add_unit/C1
/tb_processor/uut/DP/alu/add_unit/C2
/tb_processor/uut/DP/alu/add_unit/C3
/tb_processor/uut/DP/alu/add_unit/sum0_1
/tb_processor/uut/DP/alu/add_unit/sum1_1
/tb_processor/uut/DP/alu/add_unit/sum0_2
/tb_processor/uut/DP/alu/add_unit/sum1_2
/tb_processor/uut/DP/alu/add_unit/sum0_3
/tb_processor/uut/DP/alu/add_unit/sum1_3
/tb_processor/uut/DP/alu/add_unit/C0_1
/tb_processor/uut/DP/alu/add_unit/C1_1
/tb_processor/uut/DP/alu/add_unit/C0_2
/tb_processor/uut/DP/alu/add_unit/C1_2
/tb_processor/uut/DP/alu/add_unit/C0_3
/tb_processor/uut/DP/alu/add_unit/C1_3
Now
185 ps
Cursor 1
179 ps
80 ps
90 ps
100 ps
110 ps
120 ps
130 ps
140 ps
150 ps
160 ps
170 ps
180 ps
179 ps
75 ps to 191 ps
/tb_processor/uut/DP/alu/csa_cout_add
Page 32 of 32 2,962 words English (U.S.) Editor Suggestions: Showing
140% Fit Give Feedback to Microsoft

mem: و DP





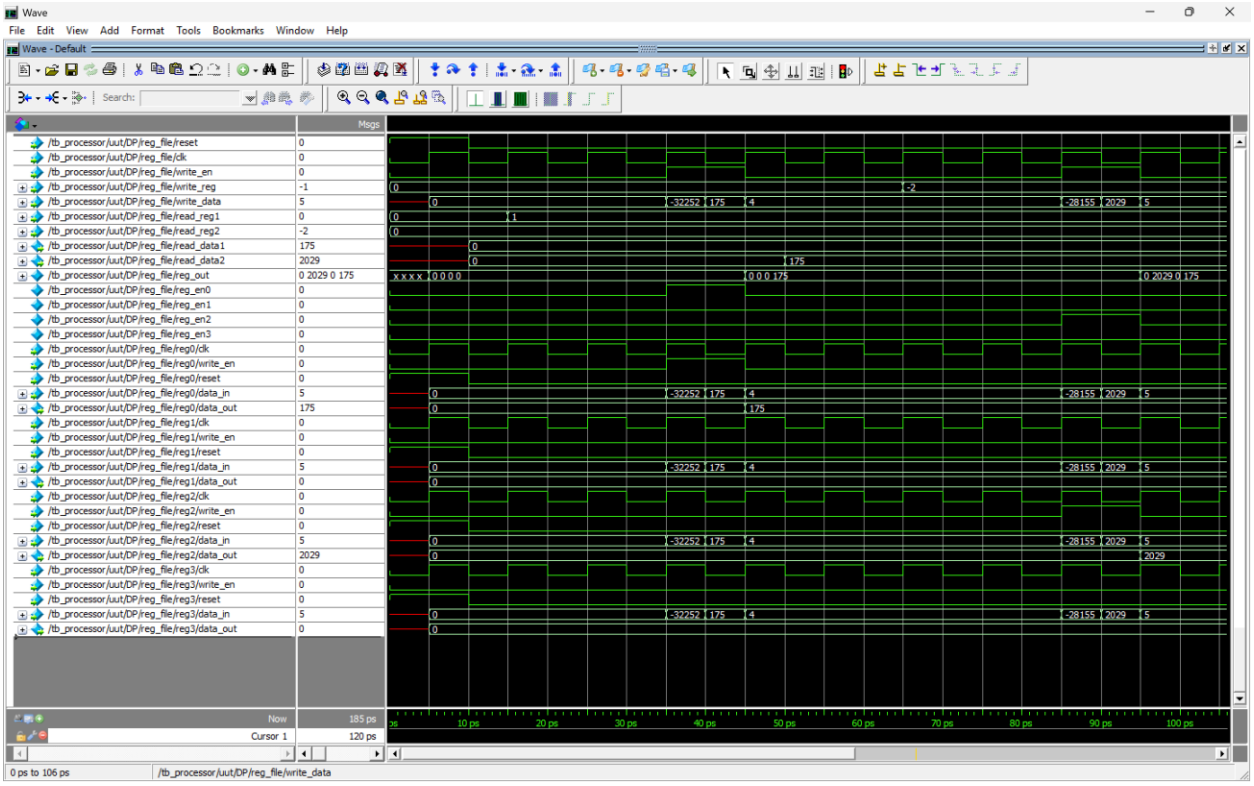
تست کیس جمع دو عدد مثبت:

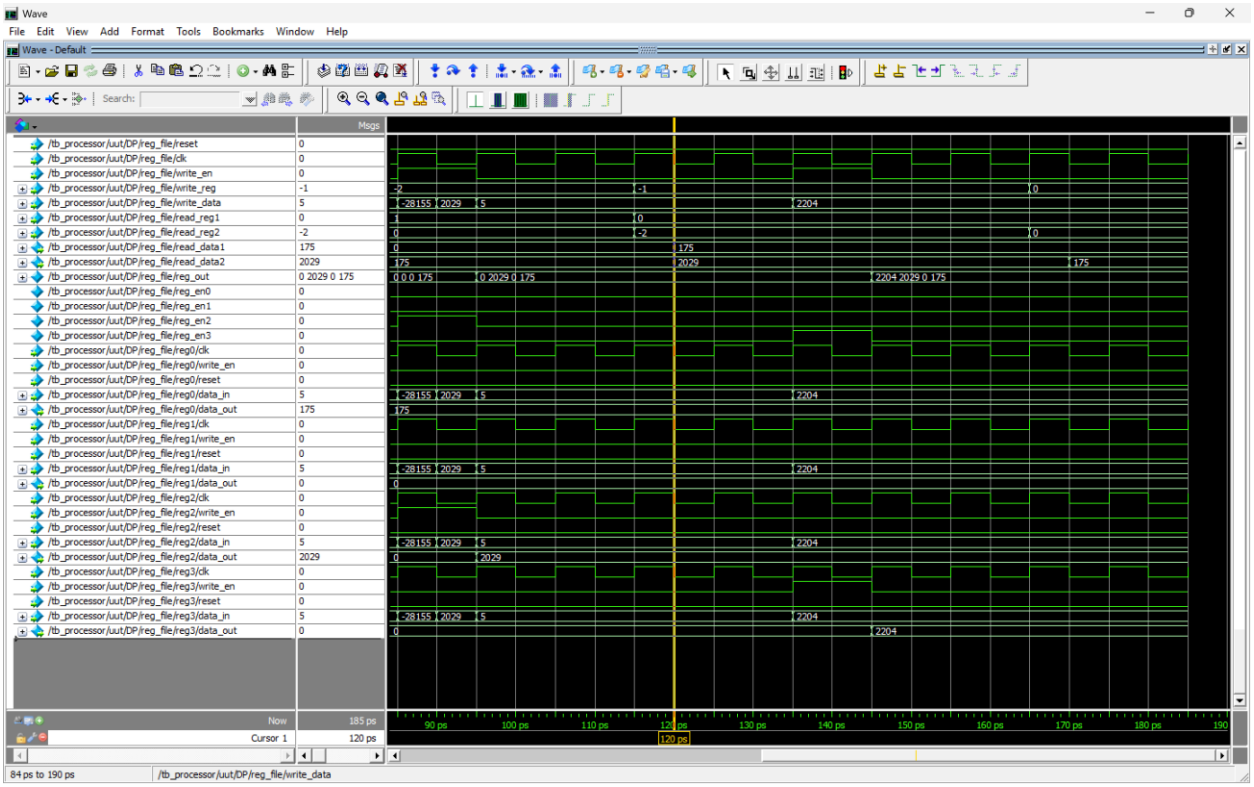
2029+175=2204

C: > Users > reaka > OneDrive > Desktop > modelsim > program.mem

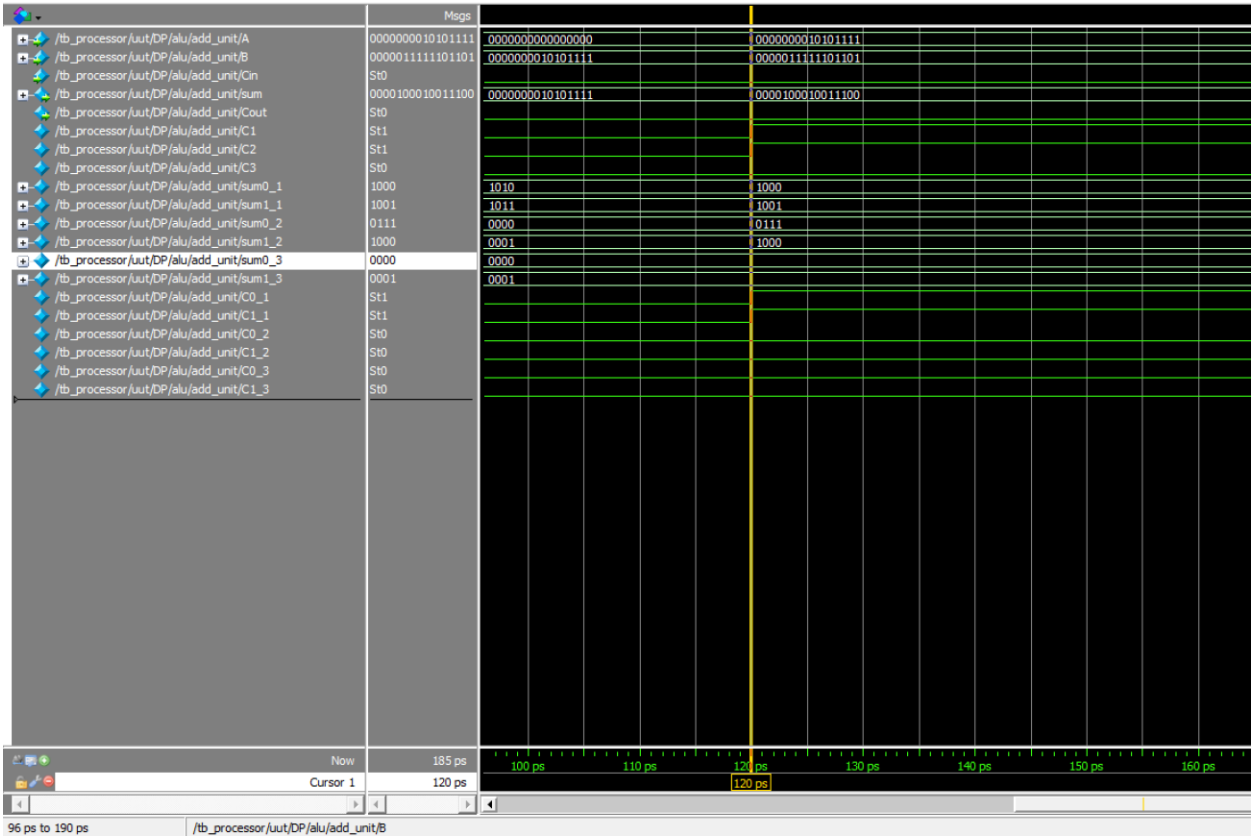
```
1000001000000100 //load r0,r1(000000100) r1 is 0
1001001000000101 //load r2,r1(000000101)
0001100100000000 //add r3=r2+r0
1100000000000000 //halt
0000000010101111 //data1 should be loaded into r0 = 175
0000011111101101 //data2 should be loaded into r2 = 2029
```

به دلیل شباهت با برنامه قبل فقط Waveform رجیسترفایل را در اینجا قرار می دهم تا درستی جمع کننده را چک کنیم.


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_processor/uut/DP/reg_file/reset
/tb_processor/uut/DP/reg_file/clk
/tb_processor/uut/DP/reg_file/write_en
/tb_processor/uut/DP/reg_file/write_reg
/tb_processor/uut/DP/reg_file/write_data
/tb_processor/uut/DP/reg_file/read_reg1
/tb_processor/uut/DP/reg_file/read_reg2
/tb_processor/uut/DP/reg_file/read_data1
/tb_processor/uut/DP/reg_file/read_data2
/tb_processor/uut/DP/reg_file/reg_out
/tb_processor/uut/DP/reg_file/reg_en0
/tb_processor/uut/DP/reg_file/reg_en1
/tb_processor/uut/DP/reg_file/reg_en2
/tb_processor/uut/DP/reg_file/reg_en3
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/reg_file/reg2/clk
/tb_processor/uut/DP/reg_file/reg2/write_en
/tb_processor/uut/DP/reg_file/reg2/reset
/tb_processor/uut/DP/reg_file/reg2/data_in
/tb_processor/uut/DP/reg_file/reg2/data_out
/tb_processor/uut/DP/reg_file/reg3/clk
/tb_processor/uut/DP/reg_file/reg3/write_en
/tb_processor/uut/DP/reg_file/reg3/reset
/tb_processor/uut/DP/reg_file/reg3/data_in
/tb_processor/uut/DP/reg_file/reg3/data_out
Now
185 ps
Cursor 1
120 ps
0 ps to 106 ps
/tb_processor/uut/DP/reg_file/write_data


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_processor/uut/DP/reg_file/reset
/tb_processor/uut/DP/reg_file/clk
/tb_processor/uut/DP/reg_file/write_en
/tb_processor/uut/DP/reg_file/write_reg
/tb_processor/uut/DP/reg_file/write_data
/tb_processor/uut/DP/reg_file/read_reg1
/tb_processor/uut/DP/reg_file/read_reg2
/tb_processor/uut/DP/reg_file/read_data1
/tb_processor/uut/DP/reg_file/read_data2
/tb_processor/uut/DP/reg_file/reg_out
/tb_processor/uut/DP/reg_file/reg_en0
/tb_processor/uut/DP/reg_file/reg_en1
/tb_processor/uut/DP/reg_file/reg_en2
/tb_processor/uut/DP/reg_file/reg_en3
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/reg_file/reg2/clk
/tb_processor/uut/DP/reg_file/reg2/write_en
/tb_processor/uut/DP/reg_file/reg2/reset
/tb_processor/uut/DP/reg_file/reg2/data_in
/tb_processor/uut/DP/reg_file/reg2/data_out
/tb_processor/uut/DP/reg_file/reg3/clk
/tb_processor/uut/DP/reg_file/reg3/write_en
/tb_processor/uut/DP/reg_file/reg3/reset
/tb_processor/uut/DP/reg_file/reg3/data_in
/tb_processor/uut/DP/reg_file/reg3/data_out
Now
185 ps
Cursor 1
120 ps
84 ps to 190 ps
/tb_processor/uut/DP/reg_file/write_data

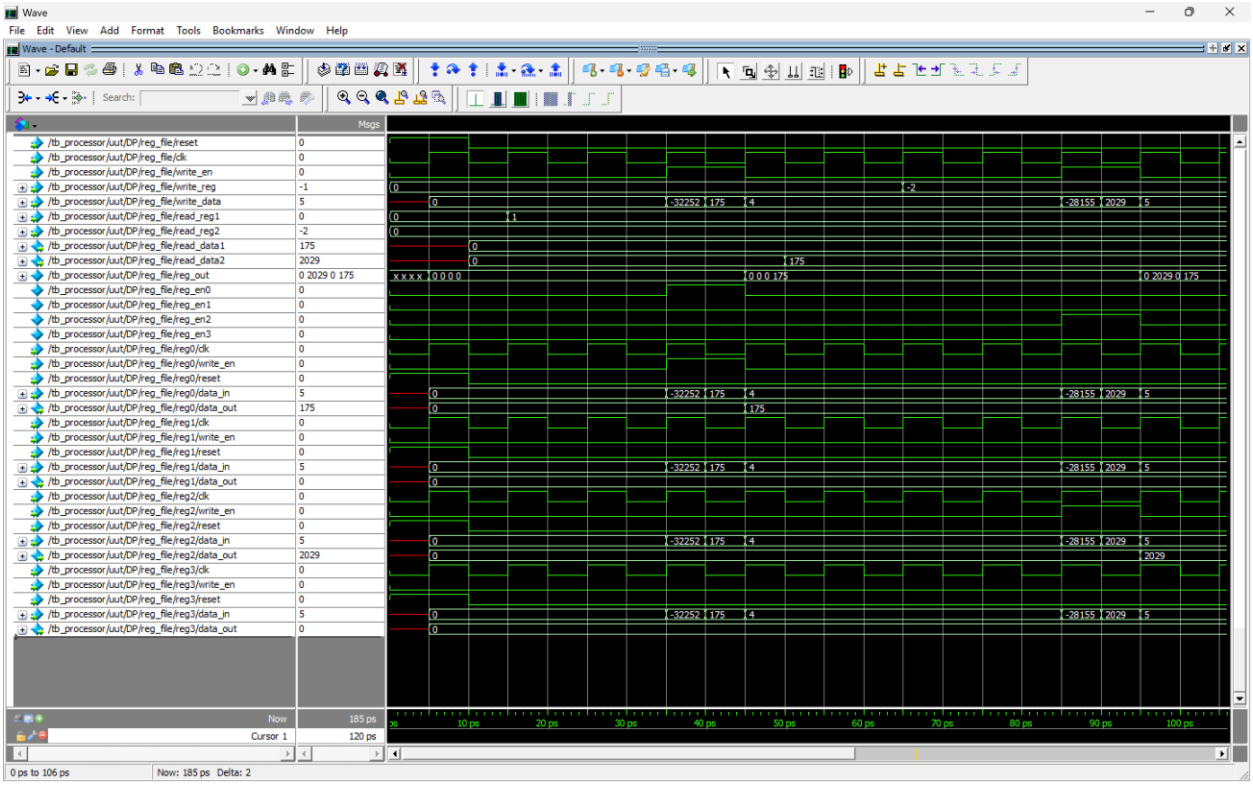
csaAdder:

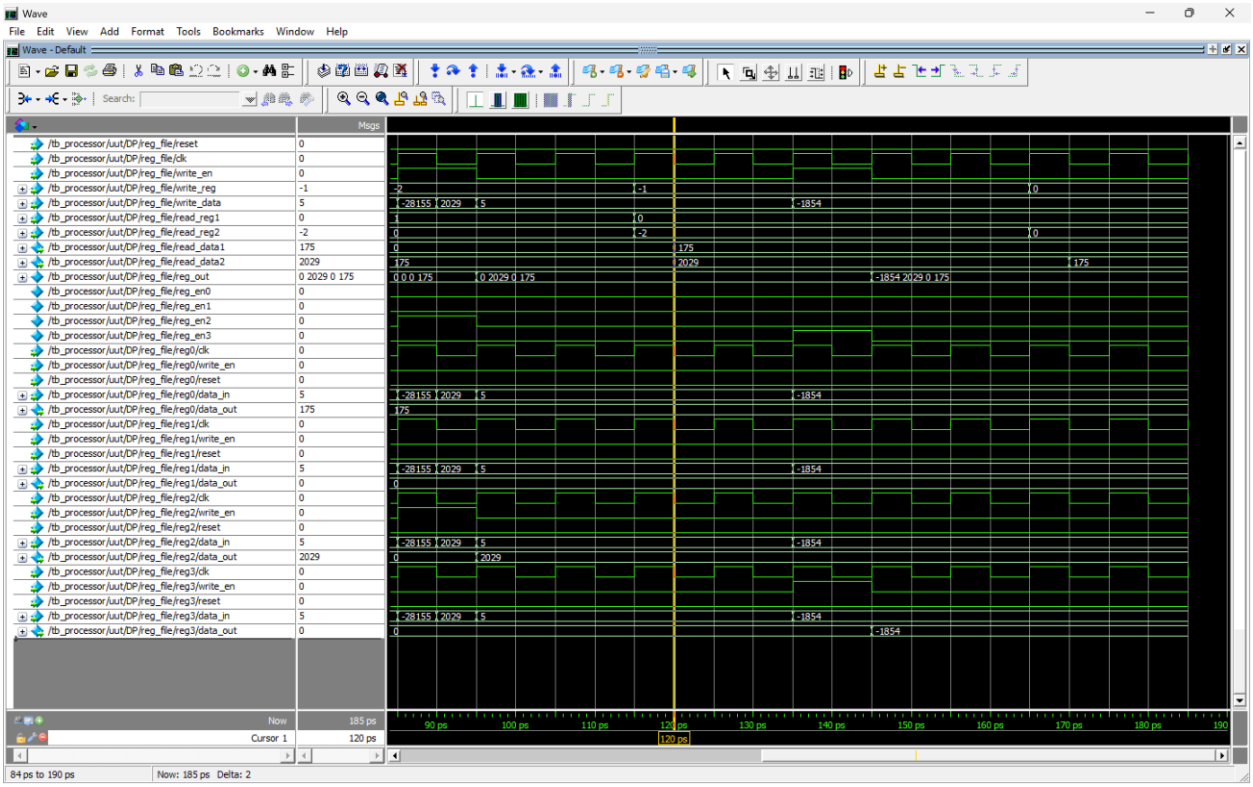


تست کیس تفریق:

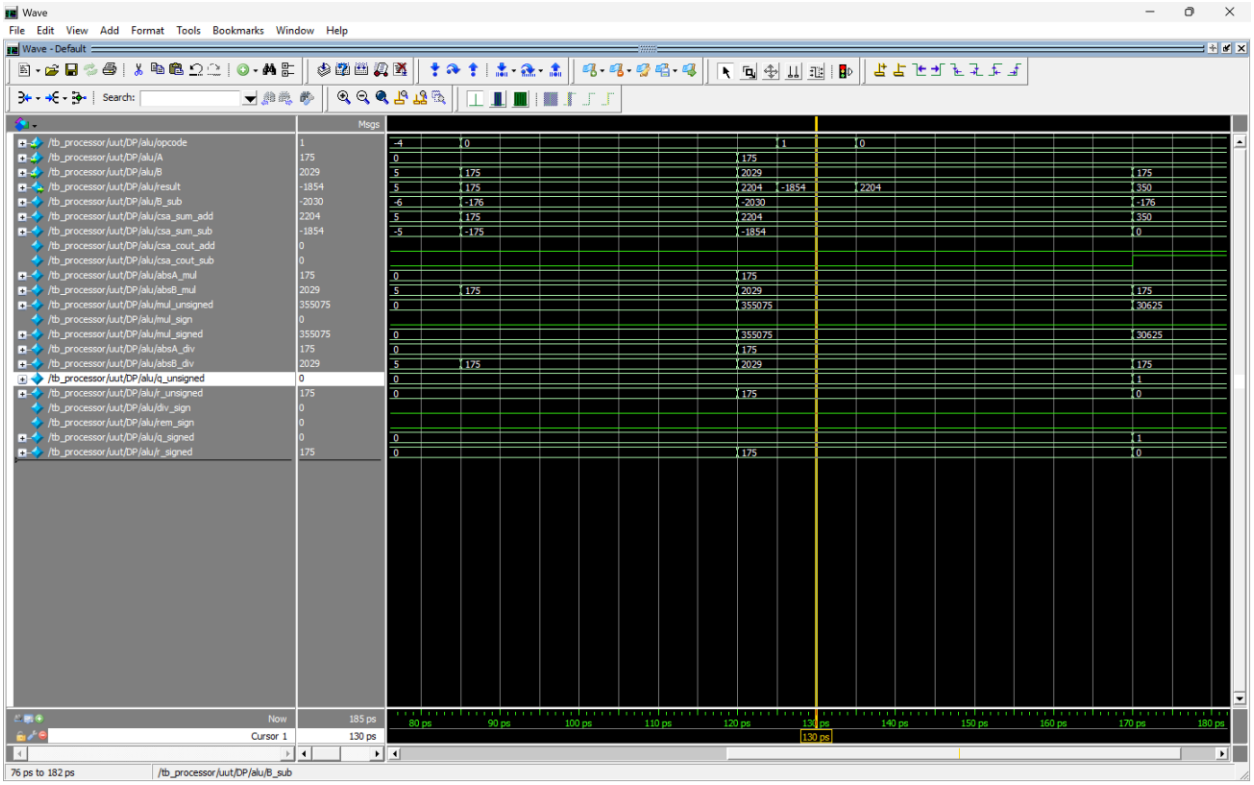
```
1000001000000100 //load r0,r1(000000100) r1 is 0
1001001000000101 //load r2,r1(000000101)
0011100100000000 //sub r3=r0-r2
1100000000000000 //halt
0000000010101111 //data1 should be loaded into r0 = 175
0000011111101101 //data2 should be loaded into r2 = 2029
```

175-2029=1854


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_processor/uut/DP/reg_file/reset
/tb_processor/uut/DP/reg_file/clk
/tb_processor/uut/DP/reg_file/write_en
/tb_processor/uut/DP/reg_file/write_reg
/tb_processor/uut/DP/reg_file/write_data
/tb_processor/uut/DP/reg_file/read_reg1
/tb_processor/uut/DP/reg_file/read_reg2
/tb_processor/uut/DP/reg_file/read_data1
/tb_processor/uut/DP/reg_file/read_data2
/tb_processor/uut/DP/reg_file/reg_out
/tb_processor/uut/DP/reg_file/reg_en0
/tb_processor/uut/DP/reg_file/reg_en1
/tb_processor/uut/DP/reg_file/reg_en2
/tb_processor/uut/DP/reg_file/reg_en3
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/reg_file/reg2/clk
/tb_processor/uut/DP/reg_file/reg2/write_en
/tb_processor/uut/DP/reg_file/reg2/reset
/tb_processor/uut/DP/reg_file/reg2/data_in
/tb_processor/uut/DP/reg_file/reg2/data_out
/tb_processor/uut/DP/reg_file/reg3/clk
/tb_processor/uut/DP/reg_file/reg3/write_en
/tb_processor/uut/DP/reg_file/reg3/reset
/tb_processor/uut/DP/reg_file/reg3/data_in
/tb_processor/uut/DP/reg_file/reg3/data_out
Now
185 ps
Cursor 1
120 ps
0 ps to 106 ps
Now: 185 ps Delta: 2


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Msgs
/tb_processor/uut/DP/reg_file/reset
/tb_processor/uut/DP/reg_file/clk
/tb_processor/uut/DP/reg_file/write_en
/tb_processor/uut/DP/reg_file/write_reg
/tb_processor/uut/DP/reg_file/write_data
/tb_processor/uut/DP/reg_file/read_reg1
/tb_processor/uut/DP/reg_file/read_reg2
/tb_processor/uut/DP/reg_file/read_data1
/tb_processor/uut/DP/reg_file/read_data2
/tb_processor/uut/DP/reg_file/reg_out
/tb_processor/uut/DP/reg_file/reg_en0
/tb_processor/uut/DP/reg_file/reg_en1
/tb_processor/uut/DP/reg_file/reg_en2
/tb_processor/uut/DP/reg_file/reg_en3
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/reg_file/reg2/clk
/tb_processor/uut/DP/reg_file/reg2/write_en
/tb_processor/uut/DP/reg_file/reg2/reset
/tb_processor/uut/DP/reg_file/reg2/data_in
/tb_processor/uut/DP/reg_file/reg2/data_out
/tb_processor/uut/DP/reg_file/reg3/clk
/tb_processor/uut/DP/reg_file/reg3/write_en
/tb_processor/uut/DP/reg_file/reg3/reset
/tb_processor/uut/DP/reg_file/reg3/data_in
/tb_processor/uut/DP/reg_file/reg3/data_out
Now
185 ps
Cursor 1
120 ps
84 ps to 190 ps
Now: 185 ps Delta: 2

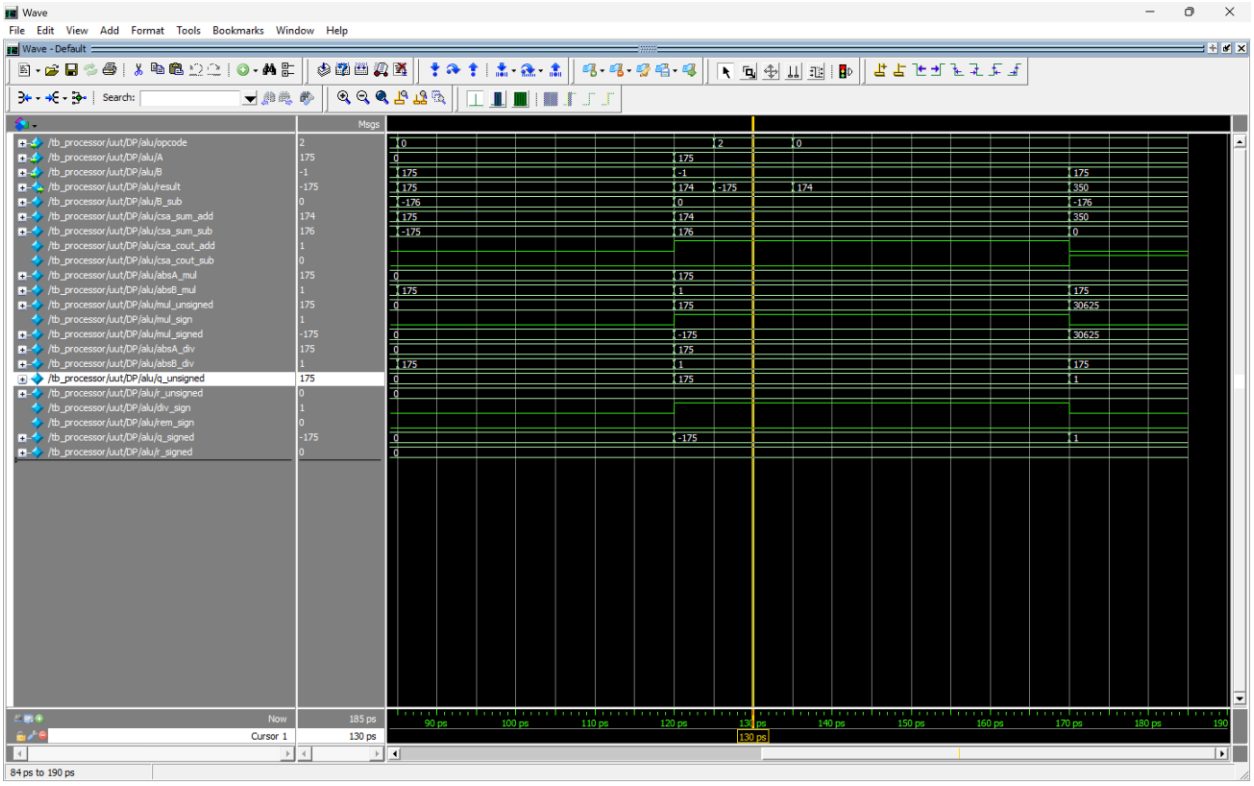
ALU:



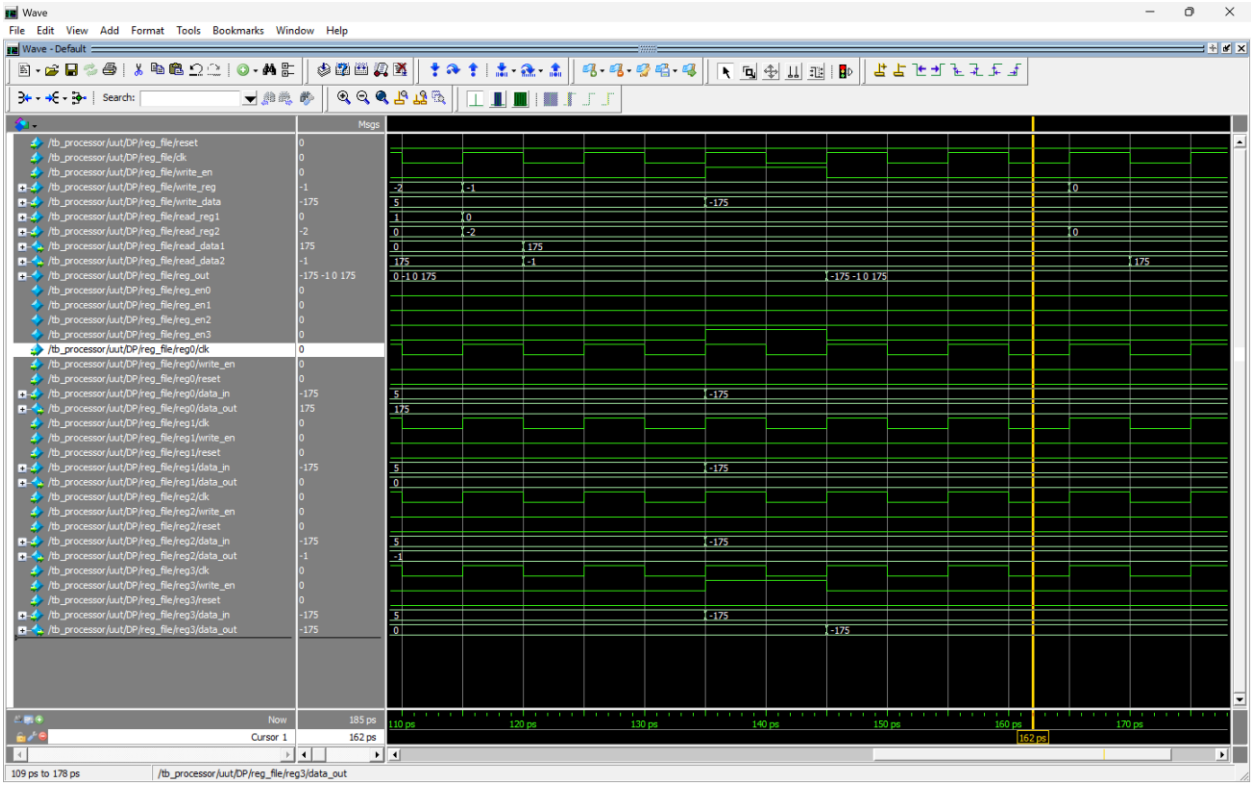
تست کیس ضرب:

```
1000001000000100 //load r0,r1(000000100) r1 is 0
1001001000000101 //load r2,r1(000000101)
0101100100000000 //mul r3=r0*r2 =175*-1
1100000000000000 //halt
0000000010101111 //data1 should be loaded into r0 = 175
1111111111111111 //data2 should be loaded into r2 = -1
```

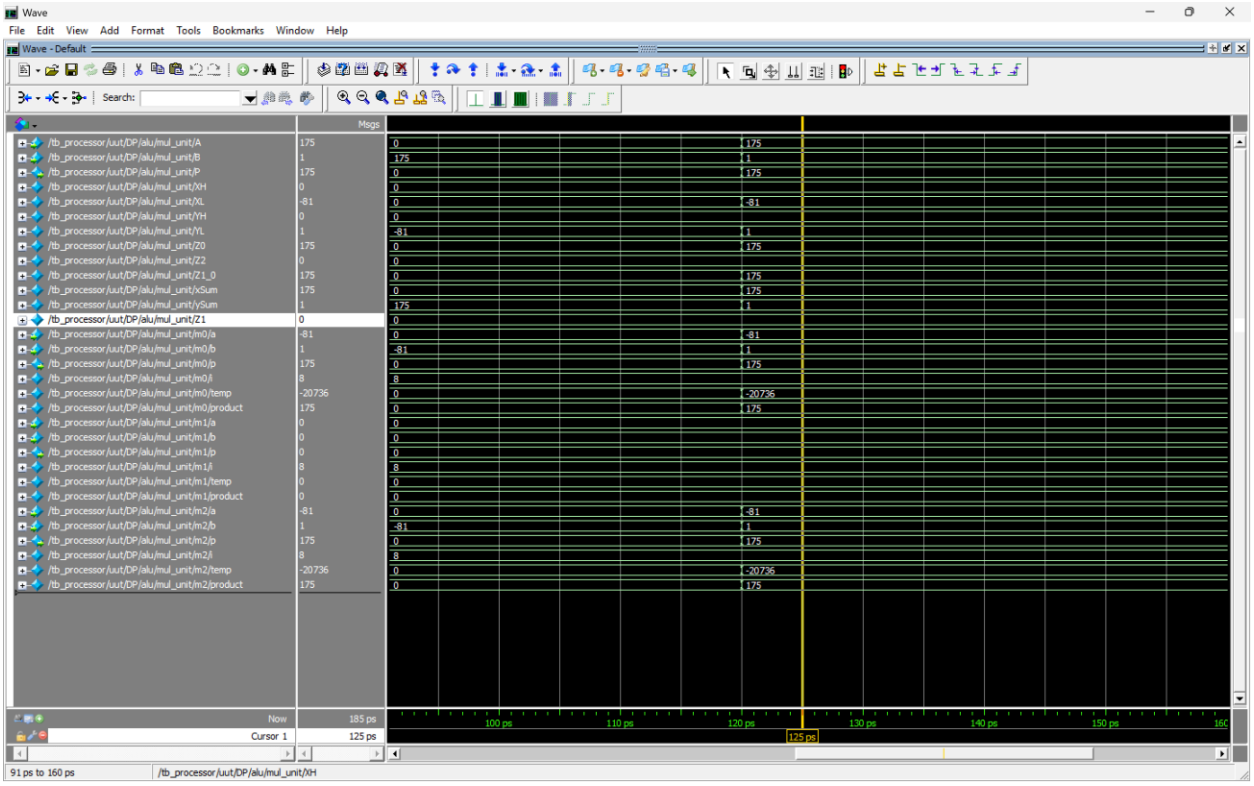
Alu:



Registers:

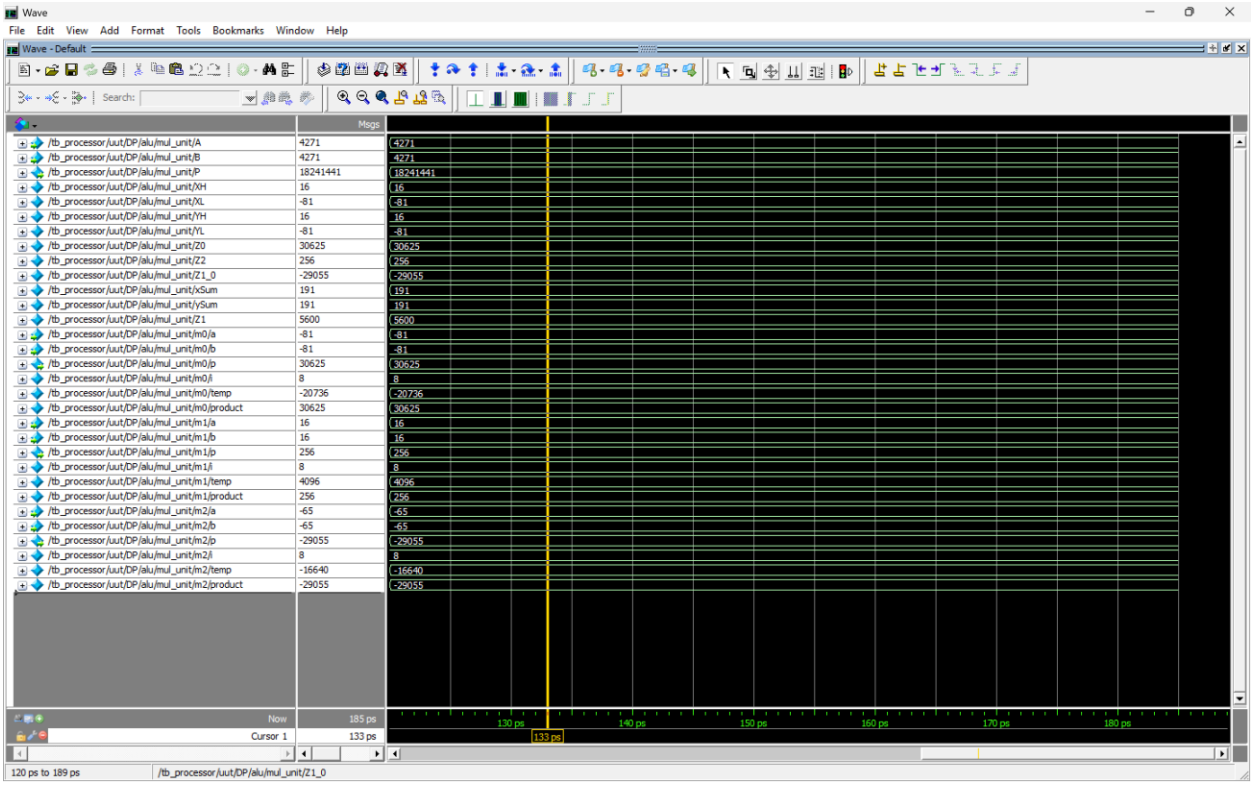


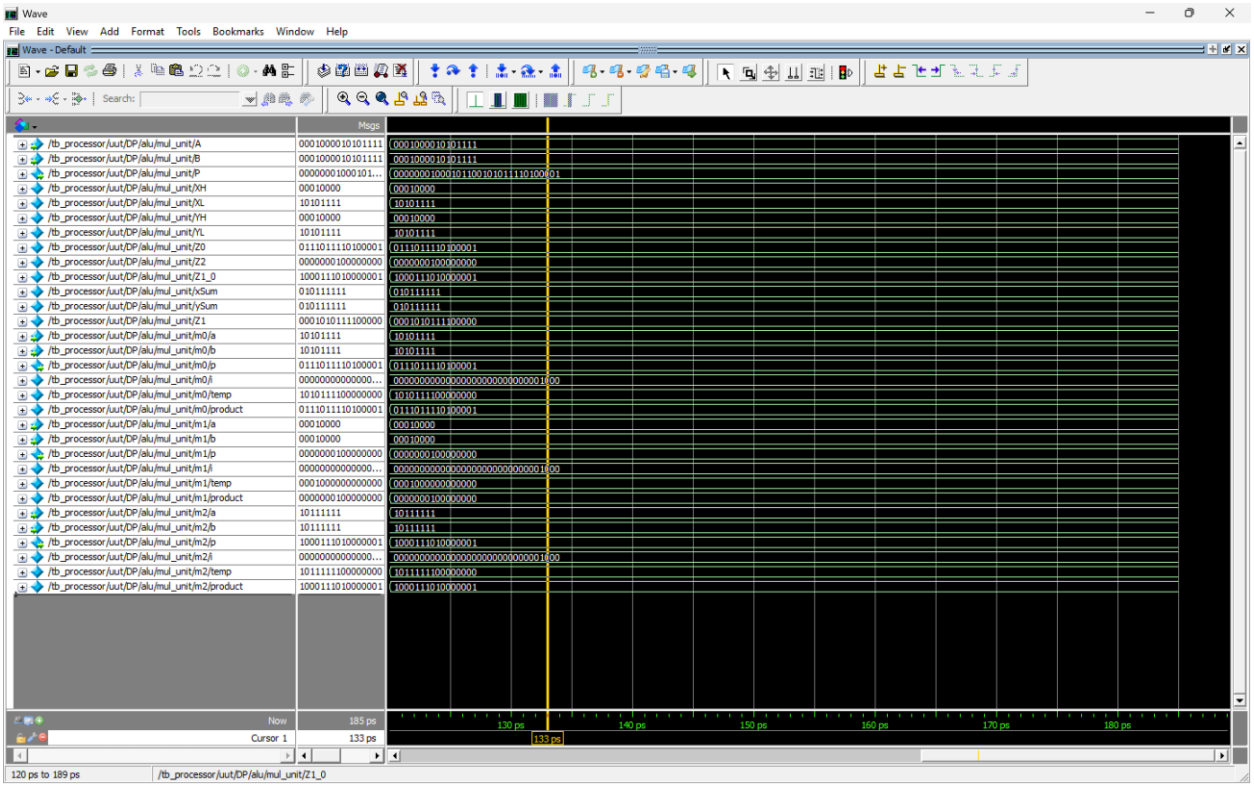
Multiplier:همانطور که می بینیم unsigned هست.



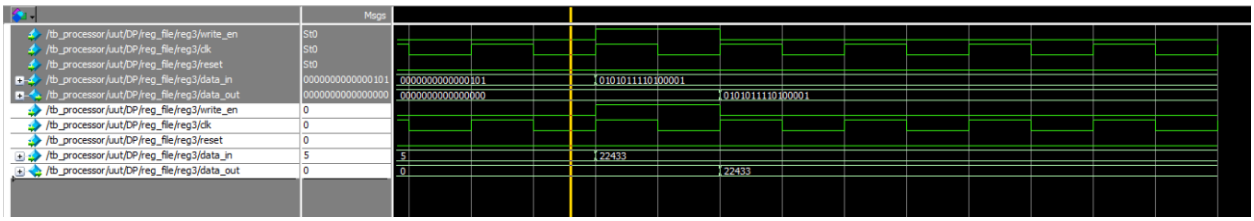
تست کیس ضرب:

```
1000001000000100 //load r0,r1(000000100) r1 is 0
1001001000000101 //load r2,r1(000000101)
0101100100000000 //mul r3=r0*r2 =4271*4271
1100000000000000 //halt
0001000010101111 //data1 should be loaded into r0 = 4271
0001000010101111 //data2 should be loaded into r2 = 4271
```





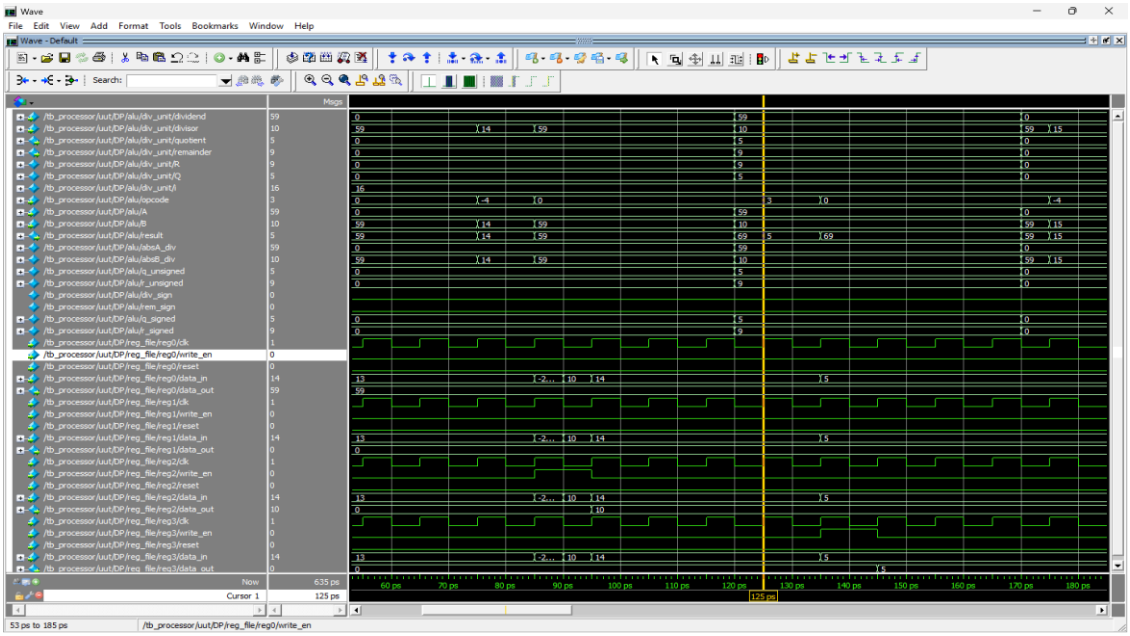
پس نتایج در 16 بیت جا نمی شود و نتیجه خروجی alu متفاوت از این مقدار خواهد بود:



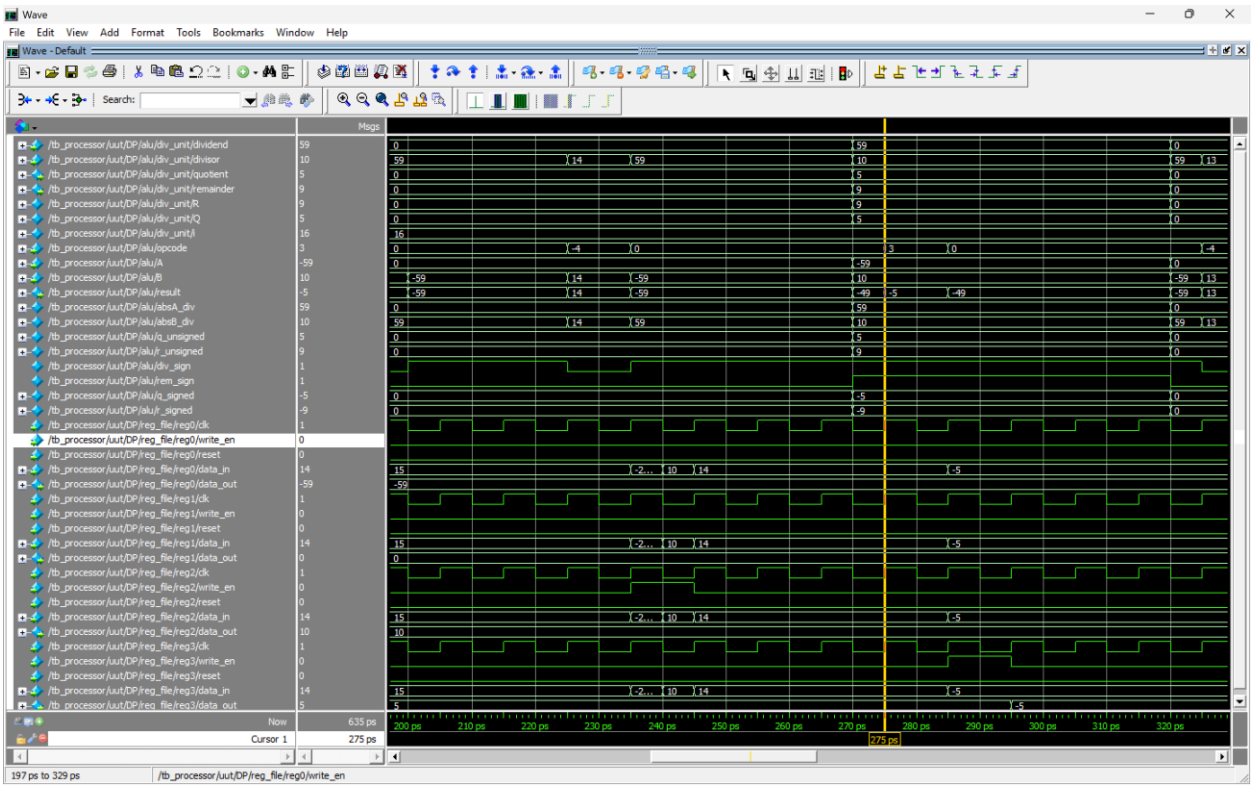
تست کیس تقسیم:

```
1000001000001101 //load r0,r1(000000100) r1 is 0
1001001000001110 //load r2,r1(000000101)
0111100100000000 //div r3=r0/r2 =59/10
1000001000001111 //load r0,r1(000000100) r1 is 0
1001001000001110 //load r2,r1(000000101)
0111100100000000 //div r3=r0/r2 =-59/10
1000001000001101 //load r0,r1(000000100) r1 is 0
1001001000010000 //load r2,r1(000000101)
0111100100000000 //div r3=r0/r2 =59/-10
1000001000001111 //load r0,r1(000000100) r1 is 0
1001001000010000 //load r2,r1(000000101)
0111100100000000 //div r3=r0/r2 =-59/-10
1100000000000000 //halt
0000000000111011//data1 should be loaded into r0 = 59
0000000000001010 //data2 should be loaded into r2 = 10
1111111111000101 //data3 = -59
1111111111110110 //data4 = -10
```

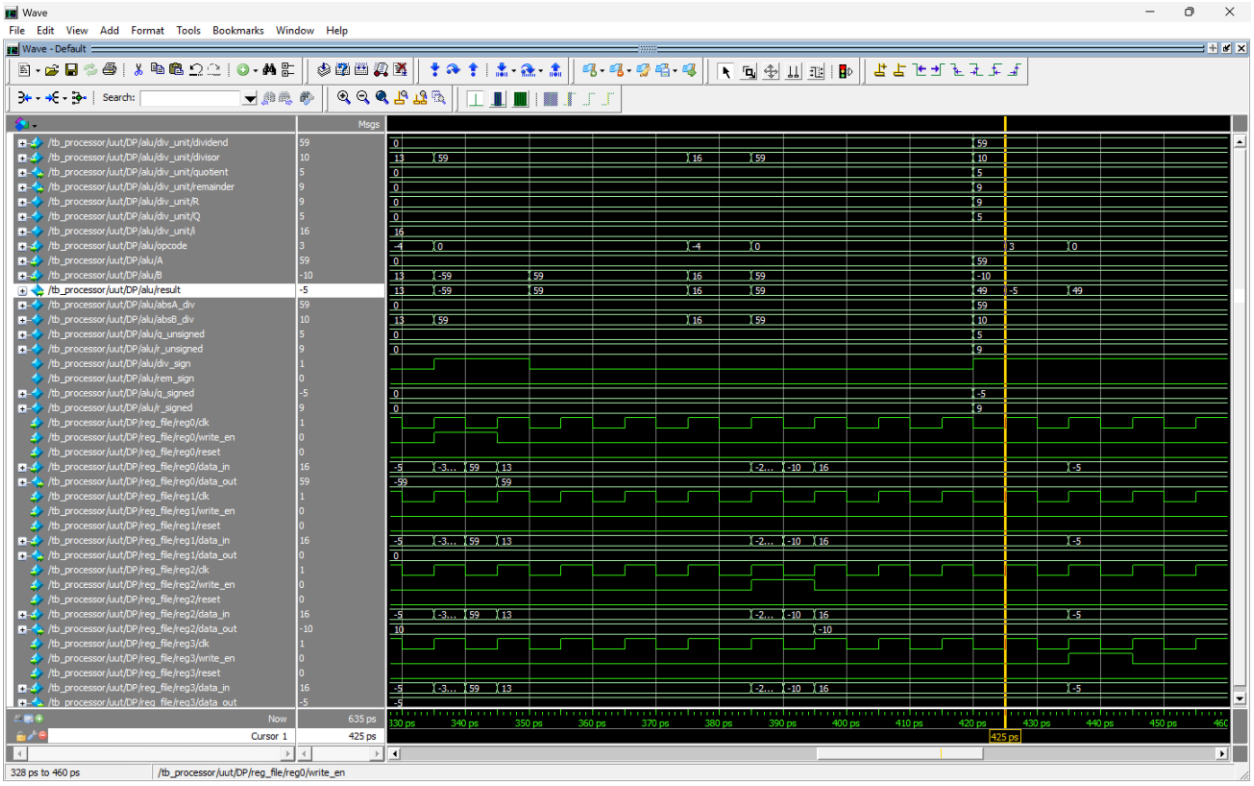
59/10:



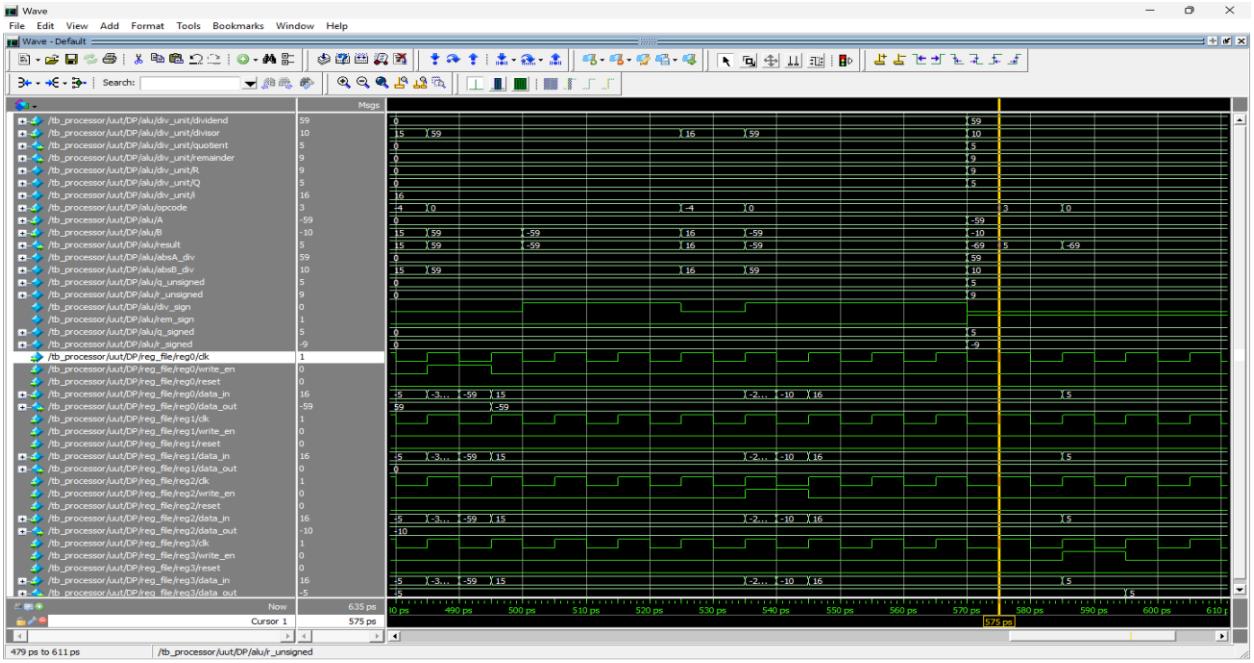
-59/10



59/-10


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_processor/uut/DP/alu/div_unit/dividend
/tb_processor/uut/DP/alu/div_unit/divisor
/tb_processor/uut/DP/alu/div_unit/quotient
/tb_processor/uut/DP/alu/div_unit/remainder
/tb_processor/uut/DP/alu/div_unit/R
/tb_processor/uut/DP/alu/div_unit/Q
/tb_processor/uut/DP/alu/div_unit/I
/tb_processor/uut/DP/alu/opcode
/tb_processor/uut/DP/alu/A
/tb_processor/uut/DP/alu/B
/tb_processor/uut/DP/alu/result
/tb_processor/uut/DP/alu/absA_div
/tb_processor/uut/DP/alu/absB_div
/tb_processor/uut/DP/alu/q_unsigned
/tb_processor/uut/DP/alu/r_unsigned
/tb_processor/uut/DP/alu/div_sign
/tb_processor/uut/DP/alu/rem_sign
/tb_processor/uut/DP/alu/q_signed
/tb_processor/uut/DP/alu/r_signed
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/reg_file/reg2/clk
/tb_processor/uut/DP/reg_file/reg2/write_en
/tb_processor/uut/DP/reg_file/reg2/reset
/tb_processor/uut/DP/reg_file/reg2/data_in
/tb_processor/uut/DP/reg_file/reg2/data_out
/tb_processor/uut/DP/reg_file/reg3/clk
/tb_processor/uut/DP/reg_file/reg3/write_en
/tb_processor/uut/DP/reg_file/reg3/reset
/tb_processor/uut/DP/reg_file/reg3/data_in
/tb_processor/uut/DP/reg_file/reg3/data_out
Now
635 ps
Cursor 1
425 ps
328 ps to 460 ps
/tb_processor/uut/DP/reg_file/reg0/write_en

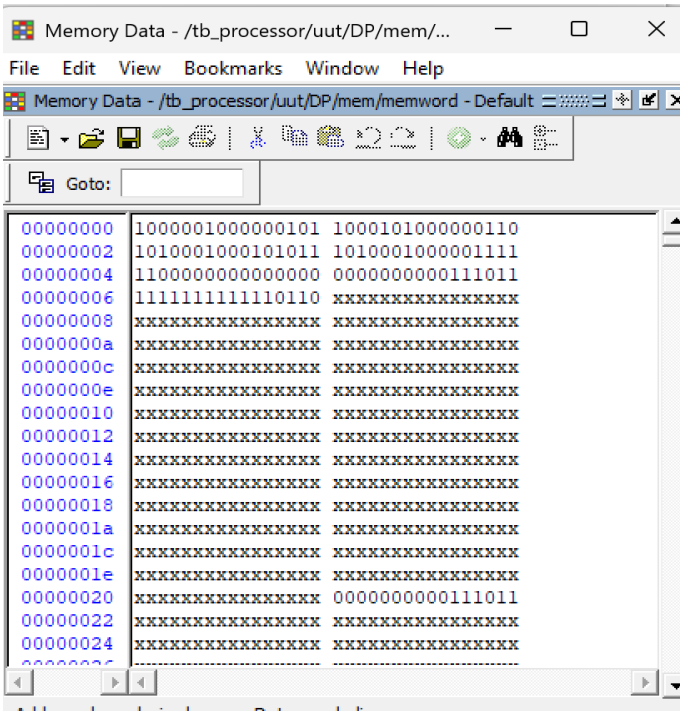
-59/-10


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/tb_processor/uut/DP/alu/div_unit/dividend
/tb_processor/uut/DP/alu/div_unit/divisor
/tb_processor/uut/DP/alu/div_unit/quotient
/tb_processor/uut/DP/alu/div_unit/remainder
/tb_processor/uut/DP/alu/div_unit/R
/tb_processor/uut/DP/alu/div_unit/Q
/tb_processor/uut/DP/alu/div_unit/I
/tb_processor/uut/DP/alu/opcode
/tb_processor/uut/DP/alu/A
/tb_processor/uut/DP/alu/B
/tb_processor/uut/DP/alu/result
/tb_processor/uut/DP/alu/absA_div
/tb_processor/uut/DP/alu/absB_div
/tb_processor/uut/DP/alu/q_unsigned
/tb_processor/uut/DP/alu/r_unsigned
/tb_processor/uut/DP/alu/div_sign
/tb_processor/uut/DP/alu/rem_sign
/tb_processor/uut/DP/alu/q_signed
/tb_processor/uut/DP/alu/r_signed
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/reg_file/reg2/clk
/tb_processor/uut/DP/reg_file/reg2/write_en
/tb_processor/uut/DP/reg_file/reg2/reset
/tb_processor/uut/DP/reg_file/reg2/data_in
/tb_processor/uut/DP/reg_file/reg2/data_out
/tb_processor/uut/DP/reg_file/reg3/clk
/tb_processor/uut/DP/reg_file/reg3/write_en
/tb_processor/uut/DP/reg_file/reg3/reset
/tb_processor/uut/DP/reg_file/reg3/data_in
/tb_processor/uut/DP/reg_file/reg3/data_out
Now
635 ps
Cursor 1
575 ps
479 ps to 611 ps
/tb_processor/uut/DP/alu/r_unsigned

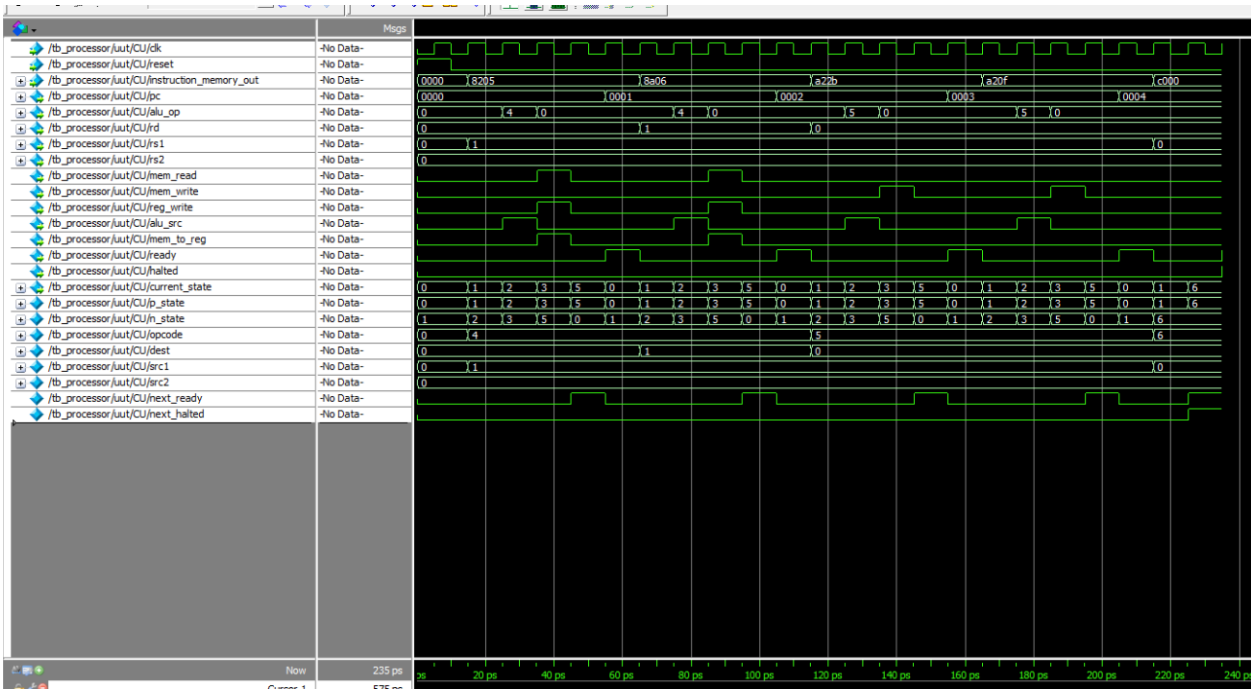
تست کیس : Load/Store

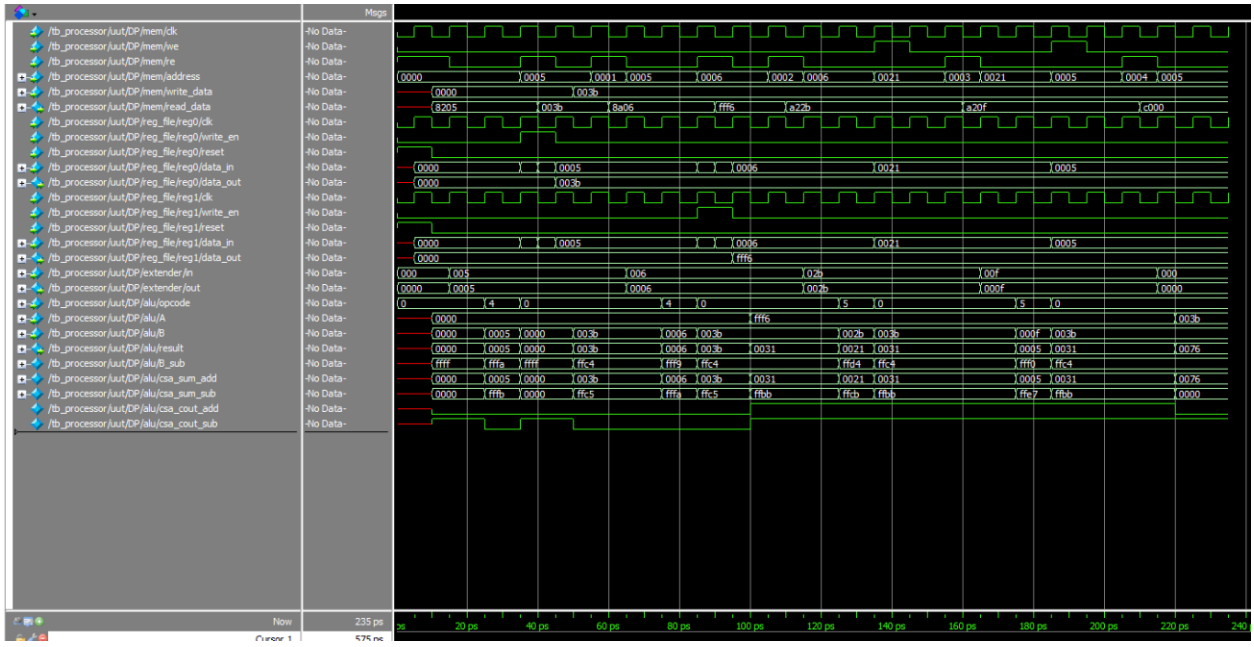
البته که از لود در تمام دستور های قبل استفاده کردیم و می توانستیم نتیجه ضرب تقسیم جمع و تفریق را هم استور کنیم ولی اینجا می خواهیم جداگانه بررسیشان بکنیم و توجه داریم که تا خانه 32 برای دستور رزرو شده و نمی توانیم در آنجا استور کنیم.

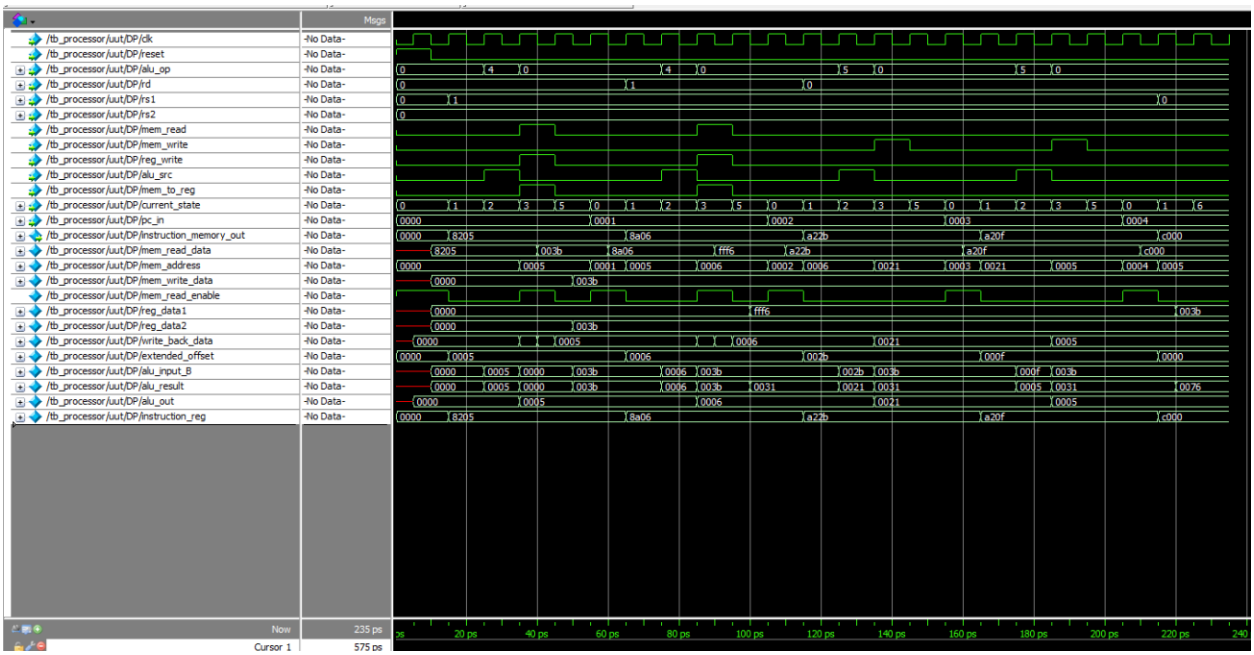
```
1000001000000101 //load r0,r1(000000100) r1 is 0
1000101000000110//load r1,r1(000000101)
1010001000101011 //store r0,r1(43)
1010001000001111 //should not be done because this place is reserved
1100000000000000 //halt
0000000000111011 //data1 = 59
1111111111110110 //data4 = -10
```



همانطور که می بینیم در خانه (43-10(33=(offset عدد 59 نوشته شده ولی در خانه 10-15=5 نوشته نشده چون رزرو برای دستور است.


Msgs
/tb_processor/uut/CU/clk
/tb_processor/uut/CU/reset
/tb_processor/uut/CU/instruction_memory_out
/tb_processor/uut/CU/pc
/tb_processor/uut/CU/alu_op
/tb_processor/uut/CU/rd
/tb_processor/uut/CU/rs1
/tb_processor/uut/CU/rs2
/tb_processor/uut/CU/mem_read
/tb_processor/uut/CU/mem_write
/tb_processor/uut/CU/reg_write
/tb_processor/uut/CU/alu_src
/tb_processor/uut/CU/mem_to_reg
/tb_processor/uut/CU/ready
/tb_processor/uut/CU/halted
/tb_processor/uut/CU/current_state
/tb_processor/uut/CU/p_state
/tb_processor/uut/CU/n_state
/tb_processor/uut/CU/opcode
/tb_processor/uut/CU/dest
/tb_processor/uut/CU/src1
/tb_processor/uut/CU/src2
/tb_processor/uut/CU/next_ready
/tb_processor/uut/CU/next_halted
Now
235 ps


Msgs
/tb_processor/uut/DP/mem/clk
/tb_processor/uut/DP/mem/we
/tb_processor/uut/DP/mem/re
/tb_processor/uut/DP/mem/address
/tb_processor/uut/DP/mem/write_data
/tb_processor/uut/DP/mem/read_data
/tb_processor/uut/DP/reg_file/reg0/clk
/tb_processor/uut/DP/reg_file/reg0/write_en
/tb_processor/uut/DP/reg_file/reg0/reset
/tb_processor/uut/DP/reg_file/reg0/data_in
/tb_processor/uut/DP/reg_file/reg0/data_out
/tb_processor/uut/DP/reg_file/reg1/clk
/tb_processor/uut/DP/reg_file/reg1/write_en
/tb_processor/uut/DP/reg_file/reg1/reset
/tb_processor/uut/DP/reg_file/reg1/data_in
/tb_processor/uut/DP/reg_file/reg1/data_out
/tb_processor/uut/DP/extender/in
/tb_processor/uut/DP/extender/out
/tb_processor/uut/DP/alu/opcode
/tb_processor/uut/DP/alu/A
/tb_processor/uut/DP/alu/B
/tb_processor/uut/DP/alu/result
/tb_processor/uut/DP/alu/B_sub
/tb_processor/uut/DP/alu/csa_sum_add
/tb_processor/uut/DP/alu/csa_sum_sub
/tb_processor/uut/DP/alu/csa_cout_add
/tb_processor/uut/DP/alu/csa_cout_sub
Now
235 ps


Msgs
/tb_processor/uut/DP/clk
/tb_processor/uut/DP/reset
/tb_processor/uut/DP/alu_op
/tb_processor/uut/DP/rd
/tb_processor/uut/DP/rs1
/tb_processor/uut/DP/rs2
/tb_processor/uut/DP/mem_read
/tb_processor/uut/DP/mem_write
/tb_processor/uut/DP/reg_write
/tb_processor/uut/DP/alu_src
/tb_processor/uut/DP/mem_to_reg
/tb_processor/uut/DP/current_state
/tb_processor/uut/DP/pc_in
/tb_processor/uut/DP/instruction_memory_out
/tb_processor/uut/DP/mem_read_data
/tb_processor/uut/DP/mem_address
/tb_processor/uut/DP/mem_write_data
/tb_processor/uut/DP/mem_read_enable
/tb_processor/uut/DP/reg_data1
/tb_processor/uut/DP/reg_data2
/tb_processor/uut/DP/write_back_data
/tb_processor/uut/DP/extended_offset
/tb_processor/uut/DP/alu_input_B
/tb_processor/uut/DP/alu_result
/tb_processor/uut/DP/alu_out
/tb_processor/uut/DP/instruction_reg
-No Data-
Now
235 ps
Cursor 1
575 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps