سوال 7 میانترم

Matin Mohammadi

401110329 جبرانی پایانترم

در این جا بعنوان پروژه جبرانی، سوال شماره 7 میانترم را پیاده سازی می‌کنیم.

در اینجا نیاز به یک رجیستر فایل با قابلیت ذخیره 4 آرایه 512 بیتی نیاز داریم و همچنین یک واحد پردازش ریاضیاتی که قابلیت جمع و ضرب دارد و همچنین یک حافظه با عمق 512 و عرض 32 بیت که امکان بارگزاری/ذخیره 16 خانه پشت سر هم از آن را داریم.

4 عملیات داریم که شامل ذخیره در حافظه، خواندن از حافظه، جمع و ضرب است.

ابتدا **واحد پردازش ریاضیاتی** را بررسی می‌کنیم:

توضیح ورودی و خروجی ماژول:

دو داده ورودی و یک سیگنال مشخص کننده عملیات داریم و حاصل عملیات که روی یک خروجی نوشته می‌شود (طول خروجی 16 تا 64 بیت و طول داده های ورودی 16 تا 32 بیت است و سیگنال عملیات نیز تک بیتی است)

در این واحد، دو ورودی (A1 و A2) را دریافت می‌کنیم و به آن ها بعنوان آرایه ای از 16 عدد 32 بیتی

نگاه می‌کنیم. سپس با توجه به عملیات مربوطه (جمع یا ضرب) این عملیات را روی این 32 بیتی ها

انجام می‌دهیم و در خروجی 1024 بیتی به صورت 64 بیت 64 بیت ذخیره می‌کنیم. (این خروجی 64

بیتی، حاصل الصاق A3 و A4 می‌باشد)

```
module ArithmeticProcessor (in1, in2, operation, out);
    input wire [511:0] in1, in2;
    input wire operation;
    output wire signed[1023:0] out;
    reg [1023 : 0] ans;
    reg [63:0] tmp1 = 64'b0,tmp2=64'b0;
    reg [63:0] tmp = 64'b0;
    integer i,j;
    always @(*) begin
        if(operation == 1)begin
            for (i = 0; i < 16; i = i + 1) begin
                for(j = 0; j < 32; j = j + 1) begin
                    tmp1[j] = in1[i*32+j];
                    tmp2[j] = in2[i*32+j];
                end
                for(j = 32; j < 64; j = j + 1) begin
```

```
                    tmp1[j] = tmp1[31];
                    tmp2[j] = tmp2[31];
                end
                tmp = $signed (tmp1) * $signed (tmp2);
                for (j = 0; j < 64; j = j + 1) begin
                    ans[64*i + j] = tmp[j];
                end
            end
        end
        else if (operation == 0)begin
            for (i = 0; i < 16; i = i + 1) begin
                for(j = 0; j < 32; j = j + 1) begin
                    tmp1[j] = in1[i*32+j];
                    tmp2[j] = in2[i*32+j];
                end
                for(j = 32; j < 64; j = j + 1) begin
                    tmp1[j] = tmp1[31];
                    tmp2[j] = tmp2[31];
                end
                tmp = $signed (tmp1) + $signed (tmp2);
                for (j = 0; j < 64; j = j + 1) begin
                    ans[64*i + j] = tmp[j];
                end
            end
        end
    end
    assign out = ans;
endmodule
```

در این واحد پردازشی، با توجه به بیت operation مشخص می‌شود که عملیات جمع لازم است یا ضرب.

به اینصورت که اگر operation = 0 آنگاه 32 بیت از هر ورودی را بر می‌داریم و با هم جمع می‌کنیم و حاصل را در 64 بیت از خروجی می‌ریزیم و این کار را برای همه 32 بیتی های متوالی انجام می‌دهیم $(\frac{512}{32} = 16)$

به طور مشابه با operation = 1 عملیات ضرب را انجام می‌دهیم و حاصل را در خروجی می‌ریزیم.

پس از آن نوبت به معرفی و توضیح **رجیستر فایل** می‌رسد.

```
module REGISTER_FILE (clk, reset, in1, in2, write_addr1, write_addr2, en1,
en2, read_addr, read_data, A1, A2, A3, A4);
    input wire clk;
    input wire reset;
    input wire [511 : 0] in1;
    input wire [511 : 0] in2;
    input wire [1 : 0] write_addr1;
    input wire [1 : 0] write_addr2;
    input wire en1;
    input wire en2;
    input wire [1 : 0] read_addr;
    output wire signed [511 : 0] read_data;
    output wire signed [511 : 0] A1;
    output wire signed [511 : 0] A2;
    output wire signed [511 : 0] A3;
    output wire signed [511 : 0] A4;
    reg signed [511 : 0] register_file [0 : 3];
    localparam zero = 512'b0;
    always @(negedge clk or posedge reset) begin
        if(reset) begin
            register_file[0] = zero;
            register_file[1] = zero;
            register_file[2] = zero;
            register_file[3] = zero;
        end else begin
            if (en1)
                register_file[write_addr1] <= (in1);
            if (en2)
                register_file[write_addr2] <= (in2);
        end
    end
    assign read_data = register_file[read_addr];
```

```
    assign A1 = register_file[0];
    assign A2 = register_file[1];
    assign A3 = register_file[2];
    assign A4 = register_file[3];
endmodule
```

توضیح ورودی و خروجی ماژول:

در ورودی سیگنال کلاک و ریست داریم برای رجیستر ها، دو سری داده ورودی، دو سری آدرس نوشتن و دو سری سیگنال نوشتن داریم. همچنین یک سری آدرس نوشتن داریم و در خروجی نیز داده خوانده شده و همچنین مقادیر A1 تا A4 را داریم.

در رجیستر فایل، نیاز به کلاک برای ورودی، ریست برای صفر کردن مقدار رجیستر ها در ابتدا، 2 ورودی و سیگنال های نوشتن و آدرس نوشتن برای نوشت روی رجیستر فایل و یک خروجی و آدرس خواندن برای خواندن از رجیستر فایل داریم.

در این ماژول یک حافظه register_file طراحی کردیم که شامل 4 آرایه 512 بیتی است. بر اساس سیگنال های reset و write و read مقادیر داخل حافظه و مقدار خروجی را مشخص می‌کنیم (بعلت اینکه می‌خواهیم نوشته شدن و خواندن از حافظه، سنکرون باشد و reset کردن، آسنکرون باشد فقط کلاک و ریست را در لیست حساسیت می‌آوریم.

پس از رجیسترفایل، نوبت به **حافظه** شامل 512 خانه 32 بیتی می‌رسد.

یک فایل به نام memory_init.hex داریم که در آن 512 عدد هگز 32 بیتی ذخیره شده است و به کمک دستور زیر، حافظه را مقدار دهی اولیه می‌کنیم:

```
$readmemh("memory_init.hex", memory);
```

که مقادیر آن نیز از یک کد پایتون بدست آمده اند (پس از ران کردن کد پایتون مقداری از داده ها را به صورت دستی تغییر می‌دهیم تا بعدا در تست کردن، حالات مرزی را بتوانیم داشته باشیم)

کد حافظه، به صورت زیر است:

```
module Memory(clk, reset, in, address, en, out_data);
    input wire clk;
    input wire reset;
    input wire signed [511 : 0] in;
    input wire [8 : 0] address;
    input wire en;
    output wire signed [511 : 0] out_data;
    reg signed [31 : 0] memory [0 : 511];
    reg signed [511 : 0] out;
    integer i, j;
    reg signed [31:0]tmp;
    always @(negedge clk or posedge reset) begin
            if (reset) begin
            $readmemh("memory_init.hex", memory);
        end
        else begin
            if (en) begin
                for (i = 0; i < 16; i = i + 1) begin
                    for (j = 0; j < 32; j = j + 1) begin
                        memory[(i + address) % 512][j] <= in[32 * i + j];
                    end
                end
            end
        end
    end
    always @(clk or reset or in or address or en) begin
        for (j = 0; j < 16; j = j + 1) begin
            tmp = memory[(j + address) % 512];
            for (i = 0; i < 32; i = i + 1) begin
                out[32 * j + i] <= tmp[i];
            end
```

```
        end
    end
    assign out_data = out;
endmodule
```

توضیح ورودی و خروجی ماژول:

در این ماژول سیگنال کلاک و ریست داریم، داده in و آدرس نوشتن و خواندن و سیگنال فعال ساز (en) همچنین یک دیتای خروجی که دیتایی است که می‌خوانیم.

ابتدا در صورت reset = 1 مقدار دهی اولیه انجام می‌شود و سپس در صورتی که en = 1 بود، عملیات خواندن 16 تایی انجام می‌شود. همچنین بلاکی داریم برای نوشتن بر خروجی.

نکته: آدرس به صورت یک عدد 9 بیتی است زیرا 512 خانه در حافظه داریم.

پس از حافظه، اکنون نوبت به این می‌رسد که این سه ماژول ساخته شده را در یک جا به هم متصل کنیم و پردازنده را بسازیم.

برای همین ماژول پردازنده را به شکل زیر طراحی می‌کنیم:

```
module PROCESSOR (clk, reset, instruction, A1, A2, A3, A4);
    input wire clk;
    input wire reset;
```

```
    input wire [15 : 0] instruction;
    output wire signed [511 : 0] A1;
    output wire signed [511 : 0] A2;
    output wire signed [511 : 0] A3;
    output wire signed [511 : 0] A4;
    integer i, j;

    reg signed [511 : 0] memory_in;
    reg [8 : 0] memory_addr;
    reg memory_write_en;
    wire signed [511 : 0] memory_out;
    Memory memory (.clk(clk), .reset(reset), .in(memory_in),
.address(memory_addr),
    .en(memory_write_en), .out_data(memory_out));

    reg [511 : 0] ap_in1;
    reg [511 : 0] ap_in2;
    reg operation;
    wire signed [1023 : 0] ap_out;

    ArithmeticProcessor ap (.in1(ap_in1), .in2(ap_in2), .operation(operation),
.out(ap_out));

    reg [511 : 0] RF_in_1, RF_in_2;
    reg [1 : 0] RF_write_address_1, RF_write_address_2, RF_read_address;
    reg RF_write_en1, RF_write_en2;
    wire signed [511 : 0] RF_out, RF_A1, RF_A2, RF_A3, RF_A4;

    REGISTER_FILE register_file (.clk(clk), .reset(reset), .in1(RF_in_1),
.in2(RF_in_2),
    .write_addr1(RF_write_address_1),.write_addr2(RF_write_address_2),
.en1(RF_write_en1),
    .en2(RF_write_en2),.read_addr(RF_read_address), .read_data(RF_out),
.A1(RF_A1),
    .A2(RF_A2), .A3(RF_A3), .A4(RF_A4));


    always @(posedge clk) begin
        #10
        if(instruction[15] == 0) begin
                memory_addr <= instruction[8 : 0];
                RF_write_en2 <= 0;
            if(instruction[14] == 0) begin
                memory_write_en <= 0;
                RF_write_en1 <= 1;
```

```
                RF_write_address_1 <= instruction[10 : 9];
                #10
                RF_in_1 <= memory_out;
            end
            else if(instruction[14] == 1)begin
                memory_write_en <= 1;
                RF_write_en1 <= 0;
                RF_read_address <= instruction[10 : 9];
                #10
                memory_in <= RF_out;
            end
        end
        else if (instruction[15] == 1) begin
            memory_write_en <= 0;
            RF_write_en1 <= 1;
            RF_write_en2 <= 1;
            RF_write_address_1 <= 2'b10;
            RF_write_address_2 <= 2'b11;
            ap_in1 <= RF_A1;
            ap_in2 <= RF_A2;
            if(instruction[14] == 0) begin
                operation = 1'b0;
                #10;
            end
            else if(instruction[14] == 1)begin
                operation = 1'b1;
                #10;
            end
            for(i = 0; i < 16; i = i + 1) begin
                    for(j  = 0; j < 32; j = j + 1) begin
                    RF_in_1[32 * i + j] <= ap_out[64 * i + j];
                    RF_in_2[32 * i + j] <= ap_out[64 * i + 32 + j];
                    end
                end
        end
    end
    assign A1 = RF_A1;
    assign A2 = RF_A2;
    assign A3 = RF_A3;
    assign A4 = RF_A4;
endmodule
```

توضیح ورودی و خروجی ماژول:

ورودی های ماژول، سیگنال کلاک و ریست و دستور 16 بیتی است و خروجی آن، 4 داده A1 تا A4

در پردازنده، ابتدا از 3 ماژول instance می‌گیریم و سپس کاری که انجام می‌دهیم به این صورت است که یک دستور 16 بیتی را در نظر گرفته و دو بیت سمت چپ آن را بعنوان opcode در نظر می‌گیریم. این دو بیت، 4 عملیات را پشتیبانی می‌کنند که در بلاک always مشخص کرده ایم. با دیدن هر دستوری، یک سری مقدار دهی اولیه داریم و پس از آن، مدت کوتاهی صبر می‌کنیم تا دستور انجام پذیرد و سپس خروجی ها را مقدار دهی می‌کنیم.

برای مثال دستور با opcode = 00 دستوری است که از حافظه بر روی رجیستر مشخص شده می‌خوانیم. آدرس رجیستر با بیت 10 و 9 و آدرس خانه حافظه نیز با بیت 8 تا 0 مشخص می‌شود.

در اصل، ما با داشتن 13 بیت، می‌توانیم تمامی دستورات را پوشش دهیم (دستورات جمع و ضرب که با 2 بیت نیز پوشش داده می‌شوند و دستورات لود و استور نیاز به opcode و address و source/destination دارند که 13 بیت می‌شود.) اما قالب دستورات را 16 بیتی در نظر می‌گیریم به 2 دلیل:

1. بتوانیم در آینده در صورت نیاز، پردازنده را ارتقاع دهیم و دستورات دیگری به آن اضافه کنیم.
2. طول قالب دستورات توانی از 2 باشد.

حال نوبت به این می‌رسد که برای ماژول پردازنده، یک تست طراحی کنیم به این شکل که برای ماژول، دستور را مشخص کنیم و منتظر باشیم و ببینیم که پس از اجرای دستور، روی A1 تا A4 چه چیزی نوشته می‌شود.

خلاصه دستورات: (دقت که بیت 13 تا 11 بلا استفاده اند)

00-> load (register_file [instruction [10:9]] = memory [instruction [8:0]])

01 -> store (memory [instruction [8:0]] = register_file [instruction [10:9]])

10 -> add (A4 [i]:A3 [i] = A1 [i] + A2 [i])

11 -> mul (A4 [i]:A3 [i] = A1 [i] * A2 [i])

تست بنچ زیر، دو تا عملیات لود، یک عملیات جمع و یک عملیات استور را دارد و سپس مقدار استور شده را دوباره لود می‌کند.

```
mmodule TB;

    reg clk;
    reg reset;
    reg [15 : 0] instruction;
    wire [511 : 0] A1;
    wire [511 : 0] A2;
    wire [511 : 0] A3;
```

```
    wire [511 : 0] A4;
    PROCESSOR processor (clk, reset, instruction, A1, A2, A3, A4);
    initial
        clk = 0;
    always
        #20 clk = ~clk;

    initial begin
        reset <= 1;
        #50
        reset <= 0;
        instruction <= 16'b00_000_00_000000000; //load first 16 to A1
        #500
        instruction <= 16'b00_000_01_000000001; // load next 16 to A2
        #500
        instruction <= 16'b10_000_00000000000; // add
        #500
        instruction <= 16'b01_000_10_000000000; // store A3 to the first part of
memory
        #500
        instruction <= 16'b00_000_00_000000000; //load first 16 to A1
        #500

        $stop;
    end
    initial
        $monitor($time, ":\nA1 = %h\nA2 = %h\nA3 = %h\nA4 = %h\n",
                    A1, A2, A3, A4);
endmodule
```

در خروجی، تغییرات را مشاهده می‌کنیم:

```
#                   0:
# A1 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                 520:
# A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                 560:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                1040:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A2 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                1080:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                1520:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A3 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
```

سپس:

```
#
#                1520:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A3 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
#                1560:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A3 = de38b20becb1e8f16fb47ad5e7a86b53249fa0e25278c417b298b230f243af86360b58bf9503c37743daa398f0f0f0ef1010100f10101011111111121111111
# A4 = ffffffffffffffff00000000ffffffffffffffffffffffffffffffffffffffff0000000000000000000000000ffffffff00000000000000000000000000000000
#
#                2520:
# A1 = de38b20becb1e8f16fb47ad5e7a86b53249fa0e25278c417b298b230f243af86360b58bf9503c37743daa398f0f0f0ef1010100f10101011111111121111111
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0ffffffff10101010000000011111111
# A3 = de38b20becb1e8f16fb47ad5e7a86b53249fa0e25278c417b298b230f243af86360b58bf9503c37743daa398f0f0f0ef1010100f10101011111111121111111
# A4 = ffffffffffffffff00000000ffffffffffffffffffffffffffffffffffffffff0000000000000000000000000ffffffff00000000000000000000000000000000
#
```

نکته: ظهور x در A1 و A4 تاثیر منفی در نتیجه محاسبات نمی‌گذارد و صرفا بعلت وجود تاخیر ایجاد می‌شود (همانطور که مشاهده می‌شود در فاصله زمانی خیلی کوتاهی این مشکل برطرف می‌شود)

در این تست به خوبی مشاهده می‌شود که لود و استور و جمع و لود دوباره کار می‌کنند. (مشخص است که در جمع، به آن صورت اورفلو نداریم و محتوای A4 صرفا علامت است)

برای نمونه، 8 رقم سمت چپ A1 که DE38B20A است را با 8 رقم سمت چپ A2 که 00000001 است جمع کرده ایم و حاصل DE38B20B پدید آمده است و چون این عدد منفی است، حاصل آن در A4، عدد منفی 1 را حاصل می‌کند (FFFFFFFF)

در تست بعدی، بجای جمع، ضرب می‌گذاریم تا از درستی کارکرد آن نیز اطمینان حاصل کنیم:

```
initial begin
        reset <= 1;
        #50
        reset <= 0;
        instruction <= 13'b00_00_000000000; //load first 16 to A1
        #500
        instruction <= 13'b00_01_000000001; // load next 16 to A2
        #500
        instruction <= 13'b11_00000000000; // mul

        #500
        instruction <= 13'b01_10_000000000; // store A3 to the first part of memory
        #500
        instruction <= 13'b00_00_000000000; //load first 16 to A1
        #500
        $stop;
    end
```

در اینجا بخش لود کردن که دقیقا مشابه قبل است، برای بخش حاصل ضرب داریم:

```
#                   0:
# A1 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                  80:
# A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                 120:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                 600:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                 640:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                1080:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A3 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

نکته: مجددا مانند بالا، تغییرات ناخواسته ای داریم که پس از 40 واحد زمانی اصلاح می‌شوند و در عملکرد کلی پردازنده ایرادی ایجاد نمی‌کنند.

```
#
#                1080:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A3 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
#                1120:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A3 = de38b20a2d70c3065c4c7fc2e4110ee642a4f951dda937329a87ac3c2e31b6a082f82b101b31f5d8d014fd800f0f0f10efefeff01010101011111111100000000
# A4 = fffffffffe171b4a057f4794d1d336d22e723e8b1cee293b007f134a00143bd8fce2fa421568b13dfb1f6df500000000ffffffff00000000000000000000000000
#
#                2080:
# A1 = de38b20a2d70c3065c4c7fc2e4110ee642a4f951dda937329a87ac3c2e31b6a082f82b101b31f5d8d014fd800f0f0f10efefeff01010101011111111100000000
# A2 = 00000001de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111
# A3 = de38b20a2d70c3065c4c7fc2e4110ee642a4f951dda937329a87ac3c2e31b6a082f82b101b31f5d8d014fd800f0f0f10efefeff01010101011111111100000000
# A4 = fffffffffe171b4a057f4794d1d336d22e723e8b1cee293b007f134a00143bd8fce2fa421568b13dfb1f6df500000000ffffffff00000000000000000000000000
#
```

که مشاهده می‌شود ضرب نیز به درستی محاسبه شده است (مثلا 8 رقم سمت چپ A1 و A2 را بررسی می‌کنیم) و همچنین می‌بینیم که اورفلو 32 بیتی ضرب ها اعمال شده است و در A4 مقادیر HIGH هر ضرب محاسبه شده و قرار داده شده اند.

8 رقم چپ A2 برابر با 00000001 و 8 رقم سمت چپ A1 برابر با DE38B20A است که ضرب آنها به وضوح DE38B20A است که درسمت چپ A3 مشاهده می‌شود.

در ادامه، تعدادی دستور متوالی جمع و ضرب و لود و استور می‌آوریم تا از صحت پردازنده اطمینان بیشتری حاصل شود.

```
reset <= 1;
        #50
        reset <= 0;
        instruction <= 13'b00_00_000000000; //load first 16 to A1
        #500
        instruction <= 13'b00_01_000100001; // load next 16 to A2
        #500
        instruction <= 13'b11_00000000000; // mul
        #500
        instruction <= 13'b10_00000000000; // add
        #500
        instruction <= 13'b01_11_000010000; // store A4
        #500
        instruction <= 13'b01_10_000000000; // store A3
        #500
        instruction <= 13'b11_10110011000; // mul
        #500
        instruction <= 13'b00_00_000000000; //load first 16 to A1
        #500
        instruction <= 13'b00_01_000000000; //load first 16 to A2
        #500
        $stop;
```

401110329 متین محمدی

قسمت زمانی اول:

```
#                  0:
# A1 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                 80:
# A1 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                120:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                600:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#                640:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# A4 = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
#
#               1080:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

قسمت زمانی دوم:

```
#               1080:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A4 = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
#               1120:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A4 = f9eb944e028977a1e4c1e27f3b5703f4ed60c206f1bdc1c4003aae67fb1e6cf719b6e0d5d703e3cc059b6d5effffffffff71bfc7000000000233f62400000000
#
#               1640:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = 0c4c76443b58abbc19812d47097920ebcef0c893e47a6e17db5a68e85b93c02ca5b052e3d45e7d00919eadac3d72e2120734f08f52d74267321c7d2f95cfef49
# A4 = 000000000000000000000000ffffffffffffffffffffffffffffffff0000000000000000ffffffffffffffff00000000000000000000000000000000ffffffff
#
#               3120:
# A1 = de38b20a0e7936e7613b43ee866d27659e32797db4464a9afe526796f3f147f0421a10cf52e9b2a8f0f0f0f0ffffffff10101010000000011111111100000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A4 = f9eb944e028977a1e4c1e27f3b5703f4ed60c206f1bdc1c4003aae67fb1e6cf719b6e0d5d703e3cc059b6d5effffffffff71bfc7000000000233f62400000000
#
#               3600:
# A1 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A4 = f9eb944e028977a1e4c1e27f3b5703f4ed60c206f1bdc1c4003aae67fb1e6cf719b6e0d5d703e3cc059b6d5effffffffff71bfc7000000000233f62400000000
#
#               3640:
# A1 = 0c4c76443b58abbc19812d47097920ebcef0c893e47a6e17db5a68e85b93c02ca5b052e3d45e7d00919eadac3d72e2120734f08f52d74267321c7d2f95cfef49
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A4 = f9eb944e028977a1e4c1e27f3b5703f4ed60c206f1bdc1c4003aae67fb1e6cf719b6e0d5d703e3cc059b6d5effffffffff71bfc7000000000233f62400000000
```

قسمت زمانی سوم:

```
#               3640:
# A1 = 0c4c76443b58abbc19812d47097920ebcef0c893e47a6e17db5a68e85b93c02ca5b052e3d45e7d00919eadac3d72e2120734f08f52d74267321c7d2f95cfef49
# A2 = 2e13c43a2cdf74d5b845e959830bf98630be4f163034237ddd08015267a2783c639642148174ca58a0adbcbc3d72e213f724e07f52d74266210b6c1e95cfef49
# A3 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A4 = f9eb944e028977a1e4c1e27f3b5703f4ed60c206f1bdc1c4003aae67fb1e6cf719b6e0d5d703e3cc059b6d5efffffffffff71bfc7000000000233f62400000000
#
#               4080:
# A1 = 0c4c76443b58abbc19812d47097920ebcef0c893e47a6e17db5a68e85b93c02ca5b052e3d45e7d00919eadac3d72e2120734f08f52d74267321c7d2f95cfef49
# A2 = 0c4c76443b58abbc19812d47097920ebcef0c893e47a6e17db5a68e85b93c02ca5b052e3d45e7d00919eadac3d72e2120734f08f52d74267321c7d2f95cfef49
# A3 = c3e5fe447ef45a331d943bbee8cadbdec69a03bea9cd7b32837cc40c5c255c40c8a8ae2c836df9c0b4113040c28d1dedb845f7f052d742661fee2bfe00000000
# A4 = f9eb944e028977a1e4c1e27f3b5703f4ed60c206f1bdc1c4003aae67fb1e6cf719b6e0d5d703e3cc059b6d5efffffffffff71bfc7000000000233f62400000000
#
```

بدین ترتیب از صحت عملکرد پردازنده اطمینان حاصل می‌کنیم و پردازنده مدنظر طراحی می‌شود.