# گزارش پروژه تمرین جبرانی پایانترم

## حل سوال 7

(در میانترم سوالات 6 و 5 حل شده بود)

این تمرین به صورت انفرادی انجام شده است.

سید محمد پویان شمس الدین

کد دانشجویی : 401110812

من برای ساخت این پردازنده از 4 ماژول ALU ، RegisterFile ، Memory و در نهایت vectorproccesor ساختم. در ادامه کد های ماژول هر بخش نمایش میدهم.

## ماژول ALU:

```
module ALU (
    input clock,
    input operation,
    input [511 : 0] input_data_1, input_data_2,
    output reg [511 : 0] output_data_1, output_data_2
);

    localparam ADD = 1'b0;
    localparam MULL = 1'b1;

    integer i;

    always @(posedge clock) begin
        #3;
        case (operation)
            ADD: begin
                for (i = 1; i < 17; i = i + 1) begin
                    {output_data_2[32*i-1 -: 32], output_data_1[32*i-1 -: 32]} <=
                    input_data_1[32*i-1 -: 32] + input_data_2[32*i-1 -: 32];
                end
            end

            MULL: begin
                for (i = 1; i < 17; i = i + 1) begin
                    {output_data_2[32*i-1 -: 32], output_data_1[32*i-1 -: 32]} <=
                    input_data_1[32*i-1 -: 32] * input_data_2[32*i-1 -: 32];
                end
            end
    endcase
    end

endmodule
```

ماژول RegisterFile :

```
1   module RegisterFile (
2       input clk,
3       input read, write,
4       input read_two, write_two,
5       input [1 : 0] address,
6       input [511 : 0] data_in,
7       output reg [511 : 0] data_out,
8       input [511 : 0] data_in_1, data_in_2,
9       output reg [511 : 0] data_out_1, data_out_2
10  );
11      reg [511 : 0] registers [0 : 3];
12
13      always @(posedge clk) begin
14          #1;
15          if (read) begin
16              data_out <= registers[address];
17          end
18
19          if (read_two) begin
20              data_out_1 <= registers[0];
21              data_out_2 <= registers[1];
22          end
23      end
24
25      always @(negedge clk) begin
26          if (write) begin
27              registers[address] <= data_in;
28          end
29
30          if (write_two) begin
31              registers[2] <= data_in_1;
32              registers[3] <= data_in_2;
33          end
34      end
35
36  endmodule
37
```

ماژول Memory :

```
module MainMemory (
    input clk,
    input read, write,
    input [4 : 0] address,
    input [511 : 0] data_in,
    output reg [511 : 0] data_out
);
    reg [31 : 0] mem [0 : 511];

    integer i;

    always @(posedge clk) begin
        #1;
        if (read) begin
            for (i = 1; i < 17; i = i + 1) begin
                data_out[32*i-1 -: 32] <= mem[16 * address + i - 1];
            end
        end
    end

    always @(negedge clk) begin
        if (write) begin
            for (i = 1; i < 17; i = i + 1) begin
                mem[16 * address + i - 1] <= data_in[32*i-1 -: 32];
            end
        end
    end

endmodule

```

ماژول پردازنده آرایه ای vectorProcessor:

در این ماژول اصلی از ماژول های قبلی استفاده کردیم. ورودی operation مشخص می کند کدام یک از عملیات load store sum product انجام شود.( فایل تمام ماژول ها قرار داده شده ) ولی تصویری از این ماژول در صفحه بعد قرار داده ام:

```
module VectorProcessor (
    input [1 : 0] opcode, rf_address,
    input [4 : 0] mem_address
);

    // Initializing upcodes:
    localparam ADDITION = 0;
    localparam MULTIPLY = 1;
    localparam LOAD = 2;
    localparam STORE = 3;

    // Control signals:
    reg clk, mem_read, mem_write, rf_read, rf_write, read_two_regs, write_two_regs;

    // Data :
    reg [511 : 0] rf_in_1, rf_in_2;
    wire [511 : 0] rf_out_1, rf_out_2;

    wire [511 : 0] mem_data_in, mem_data_out, rf_data_in, rf_data_out;

    // ALU Input & Outputs:
    reg [511 : 0] alu_in_1, alu_in_2;
    wire [511 : 0] alu_out_1, alu_out_2;

    // Instancing modules:
    ALU alu(clk, opcode[0], alu_in_1, alu_in_2, alu_out_1, alu_out_2);
    MainMemory main_memory (clk, mem_read, mem_write, mem_address, mem_data_in, mem_data_out);
    RegisterFile register_file (
        .clk(clk),
        .read(rf_read),
        .write(rf_write),
        .read_two(read_two_regs),
        .write_two(write_two_regs),
        .address(rf_address),
        .data_in(rf_data_in),
        .data_out(rf_data_out),
        .data_in_1(rf_in_1),
        .data_in_2(rf_in_2),
        .data_out_1(rf_out_1),
        .data_out_2(rf_out_2)
    );
```

```
    );

    // Doing task:
    assign mem_data_in = opcode[1] ? rf_data_out : {512{1'bz}};
    assign rf_data_in = opcode[1] ? mem_data_out : {512{1'bz}};

    always @(posedge clk) begin
        case (opcode)

            ADDITION, MULTIPLY : begin
                mem_read = 0; mem_write = 0;
                rf_read = 0; rf_write = 0;
                read_two_regs = 1; write_two_regs = 1;
                #2; // 2 units delay for register_file
                alu_in_1 <= rf_out_1;
                alu_in_2 <= rf_out_2;
                #2; // 2 units delay for alu
                rf_in_1 <= alu_out_1;
                rf_in_2 <= alu_out_2;
                // now the data is ready for write in registerfile before negedge of clock
            end

            LOAD: begin
                mem_read = 1; mem_write = 0;
                rf_read = 0; rf_write = 1;
                read_two_regs = 0; write_two_regs = 0;
            end

            STORE: begin
                mem_read = 0; mem_write = 1;
                rf_read = 1; rf_write = 0;
                read_two_regs = 0; write_two_regs = 0;
            end

        endcase
    end

    initial begin
        clk = 1;
        forever #5 clk = ~clk;
    end
```

ابتدا ماژول بستر آزمون را مطابق زیر تعریف می کنیم و مقدار دهی میکنیم. و حالت های خواسته شده سوال را بررسی می کنیم.

```
module TestBenchProcessor;

reg [1 : 0] opcode, rf_address;
reg [4 : 0] mem_address;

integer i;

VectorProcessor vectorProcessor(opcode, rf_address, mem_address);

initial begin
    for (i = 0; i < 32; i = i + 1) begin
        vectorProcessor.main_memory.mem[i] = i;
    end

    for (i = 32; i < 512; i = i + 1) begin
        vectorProcessor.main_memory.mem[i] = 2 ** (i % 32);
    end
    #100;

    $monitor("Time: %t | opcode = %b | rf_address = %d | mem_address = %d"
    , $time, opcode, rf_address, mem_address);

    $display("Doing sum on first two vectors and write Results on last two vectors :\n");

    opcode = 2; // Load
    rf_address = 0; mem_address = 0;
    #10;
    rf_address = 1; mem_address = 1;
    #10;

    opcode = 0; // Sum
    #10;

    opcode = 3; // Store
    rf_address = 2; mem_address = 31;
    #10;
    rf_address = 3; mem_address = 30;
    #10;

    $display("\nDoing multiply on second and third vectors and write Results on 15th and 16th vectors :\n");

    opcode = 2; // Load
    rf_address = 0; mem_address = 20;
    #10;
    rf_address = 1; mem_address = 21;
    #10;

    opcode = 1; // Multiply
    #10;

    opcode = 3; // Store
    rf_address = 2; mem_address = 16;
    #10;
    rf_address = 3; mem_address = 15;
    #10;

    $display("\n Results of msb sum :\n");

    for (i = 480; i < 496; i = i + 1) begin
        $display("Reading data from address %d: %d", i, vectorProcessor.main_memory.mem[i]);
    end

    $display("\n Results of lsb sum :\n");

    for (i = 496; i < 512; i = i + 1) begin
        $display("Reading data from address %d: %d", i, vectorProcessor.main_memory.mem[i]);
    end

    $display("\n Results of msb multiply :\n");

    for (i = 240; i < 256; i = i + 1) begin
        $display("Reading data from address %d: %b", i, vectorProcessor.main_memory.mem[i]);
    end

    $display("\n Results of lsb multiply :\n");

    for (i = 256; i < 272; i = i + 1) begin
        $display("Reading data from address %d: %b", i, vectorProcessor.main_memory.mem[i]);
    end
```

و نتیجه را مشاهده میکنیم:

```
# Doing sum on first two vectors and write Results on last two vectors :
#
# Time:                 100 | opcode = 10 | rf_address = 0 | mem_address =  0
# Time:                 110 | opcode = 10 | rf_address = 1 | mem_address =  1
# Time:                 120 | opcode = 00 | rf_address = 1 | mem_address =  1
# Time:                 130 | opcode = 11 | rf_address = 2 | mem_address = 31
# Time:                 140 | opcode = 11 | rf_address = 3 | mem_address = 30
#
# Doing multiply on second and third vectors and write Results on 15th and 16th vectors :
#
# Time:                 150 | opcode = 10 | rf_address = 0 | mem_address = 20
# Time:                 160 | opcode = 10 | rf_address = 1 | mem_address = 21
# Time:                 170 | opcode = 01 | rf_address = 1 | mem_address = 21
# Time:                 180 | opcode = 11 | rf_address = 2 | mem_address = 16
# Time:                 190 | opcode = 11 | rf_address = 3 | mem_address = 15
```

نتیجه جمع :

```
#  Results of msb sum :
#
# Reading data from address         480:          0
# Reading data from address         481:          0
# Reading data from address         482:          0
# Reading data from address         483:          0
# Reading data from address         484:          0
# Reading data from address         485:          0
# Reading data from address         486:          0
# Reading data from address         487:          0
# Reading data from address         488:          0
# Reading data from address         489:          0
# Reading data from address         490:          0
# Reading data from address         491:          0
# Reading data from address         492:          0
# Reading data from address         493:          0
# Reading data from address         494:          0
# Reading data from address         495:          0
#
#  Results of lsb sum :
#
# Reading data from address         496:         16
# Reading data from address         497:         18
# Reading data from address         498:         20
# Reading data from address         499:         22
# Reading data from address         500:         24
# Reading data from address         501:         26
# Reading data from address         502:         28
# Reading data from address         503:         30
# Reading data from address         504:         32
# Reading data from address         505:         34
# Reading data from address         506:         36
# Reading data from address         507:         38
# Reading data from address         508:         40
# Reading data from address         509:         42
# Reading data from address         510:         44
# Reading data from address         511:         46
#
```

نتیجه ضرب :

```
#  Results of msb multiply :
#
# Reading data from address          240: 00000000000000000000000000000000
# Reading data from address          241: 00000000000000000000000000000000
# Reading data from address          242: 00000000000000000000000000000000
# Reading data from address          243: 00000000000000000000000000000000
# Reading data from address          244: 00000000000000000000000000000000
# Reading data from address          245: 00000000000000000000000000000000
# Reading data from address          246: 00000000000000000000000000000000
# Reading data from address          247: 00000000000000000000000000000000
# Reading data from address          248: 00000000000000000000000000000001
# Reading data from address          249: 00000000000000000000000000000100
# Reading data from address          250: 00000000000000000000000000010000
# Reading data from address          251: 00000000000000000000000001000000
# Reading data from address          252: 00000000000000000000000100000000
# Reading data from address          253: 00000000000000000000010000000000
# Reading data from address          254: 00000000000000000001000000000000
# Reading data from address          255: 00000000000000000100000000000000
#
#  Results of lsb multiply :
#
# Reading data from address          256: 00000000000000010000000000000000
# Reading data from address          257: 00000000000001000000000000000000
# Reading data from address          258: 00000000000100000000000000000000
# Reading data from address          259: 00000000010000000000000000000000
# Reading data from address          260: 00000001000000000000000000000000
# Reading data from address          261: 00000100000000000000000000000000
# Reading data from address          262: 00010000000000000000000000000000
# Reading data from address          263: 01000000000000000000000000000000
# Reading data from address          264: 00000000000000000000000000000000
# Reading data from address          265: 00000000000000000000000000000000
# Reading data from address          266: 00000000000000000000000000000000
# Reading data from address          267: 00000000000000000000000000000000
# Reading data from address          268: 00000000000000000000000000000000
# Reading data from address          269: 00000000000000000000000000000000
# Reading data from address          270: 00000000000000000000000000000000
# Reading data from address          271: 00000000000000000000000000000000
```