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 Memory (
    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, regAddress,
    input [4 : 0] memoryAddress
);
    localparam ADD = 0; localparam MUL = 1; localparam LOAD = 2; localparam STORE = 3;
    reg clk, mem_read, mem_write, reg_read, reg_write, readTwo, writeTwo;

    reg [511 : 0] registerin1, registerin2; wire [511 : 0] registerout1, registerout2; wire [511 : 0] memoryDataIn, memoryDataOut, registerDataIn, registerDataOut;
    reg [511 : 0] AluIN1, AluIN2;
    wire [511 : 0] AluOut1, AluOut2;

    ALU alu(clk, opcode[0], AluIN1, AluIN2, AluOut1, AluOut2);
    Memory memory (clk, mem_read, mem_write, memoryAddress, memoryDataIn, memoryDataOut);
    RegisterFile register_file (
        .clk(clk), .read(reg_read), .write(reg_write),.read_two(readTwo),.write_two(writeTwo),
        .address(regAddress),.data_in(registerDataIn),.data_out(registerDataOut),.data_in_1(registerin1),.data_in_2(registerin2),
         .data_out_1(registerout1),.data_out_2(registerout2)
    );

    assign memoryDataIn = opcode[1] ? registerDataOut : {512{1'bz}};
    assign registerDataIn = opcode[1] ? memoryDataOut : {512{1'bz}};

    always @(posedge clk) begin
        case (opcode)

            ADD, MUL : begin
                mem_read = 0; mem_write = 0;
                reg_read = 0; reg_write = 0;
                readTwo = 1; writeTwo = 1;
                #2;
                AluIN1 <= registerout1;
                AluIN2 <= registerout2;
                #2;
                registerin1 <= AluOut1;
                registerin2 <= AluOut2;
            end

            LOAD: begin
                    mem_read = 1; mem_write = 0;
                    reg_read = 0; reg_write = 1;
                    readTwo = 0; writeTwo = 0;
                end
                STORE: begin
                    mem_read = 0; mem_write = 1;
                    reg_read = 1; reg_write = 0;
                    readTwo = 0; writeTwo = 0;
                end

            endcase
        end

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

endmodule
```

ابتدا برای اینکه مشاهده کنیم پردازنده به درستی کار میکند برای پردازنده یک ماژول بستر ازمون طراحی میکنیم که مقداری را در حافظه مقداردهی شود سپس با عملیات load از حافظه به رجیستر منتقل میکنیم و سپس عملیات جمع و ضرب را انجام می دهیم. در اخر نتیجه را store میکنیم و از روی آدرس حافظه نتیجه ضرب و جمع را می خوانیم. قسمتی از ماژول را مشاهده میکنیم:

(فایل کامل همه ماژول ها قرار داده شده است)

```
18      //load
19      opcode = 2;
20      regAddresss = 0; memoryAddress = 0;
21      #10;
22      regAddresss = 1; memoryAddress = 1;
23      #10;
24      //sum
25      opcode = 0;
26      #10;
27      //store
28      opcode = 3;
29      regAddresss = 2; memoryAddress = 31;
30      #10;
31      regAddresss = 3; memoryAddress = 30;
32      #10;
33      //load
34      opcode = 2;
35      regAddresss = 0; memoryAddress = 20;
36      #10;
37      regAddresss = 1; memoryAddress = 21;
38      #10;
39      //multiply
40      opcode = 1;
41      #10;
42      //store
43      opcode = 3;
44      regAddresss = 2; memoryAddress = 16;
45      #10;
46      regAddresss = 3; memoryAddress = 15;
47      #10;
48
```

و نتیجه هرکدام را مشاهده میکنیم:

نتیجه جمع:

```
#
# Reding the result of sum answer from adress:
#
# ADDRESS         496:         16
# ADDRESS         497:         18
# ADDRESS         498:         20
# ADDRESS         499:         22
# ADDRESS         500:         24
# ADDRESS         501:         26
# ADDRESS         502:         28
# ADDRESS         503:         30
# ADDRESS         504:         32
# ADDRESS         505:         34
# ADDRESS         506:         36
# ADDRESS         507:         38
# ADDRESS         508:         40
# ADDRESS         509:         42
# ADDRESS         510:         44
# ADDRESS         511:         46
```

نتیجه ضرب:

```
#
# Reading the result of multiplication answer from address :
#
# ADDRESS         240: 00000000000000000000000000000000
# ADDRESS         241: 00000000000000000000000000000000
# ADDRESS         242: 00000000000000000000000000000000
# ADDRESS         243: 00000000000000000000000000000000
# ADDRESS         244: 00000000000000000000000000000000
# ADDRESS         245: 00000000000000000000000000000000
# ADDRESS         246: 00000000000000000000000000000000
# ADDRESS         247: 00000000000000000000000000000000
# ADDRESS         248: 00000000000000000000000000000001
# ADDRESS         249: 00000000000000000000000000000100
# ADDRESS         250: 00000000000000000000000000010000
# ADDRESS         251: 00000000000000000000000001000000
# ADDRESS         252: 00000000000000000000000100000000
# ADDRESS         253: 00000000000000000000010000000000
# ADDRESS         254: 00000000000000000001000000000000
# ADDRESS         255: 00000000000000000100000000000000
# ADDRESS         256: 00000000000000010000000000000000
# ADDRESS         257: 00000000000001000000000000000000
# ADDRESS         258: 00000000000100000000000000000000
# ADDRESS         259: 00000000010000000000000000000000
# ADDRESS         260: 00000001000000000000000000000000
# ADDRESS         261: 00000100000000000000000000000000
# ADDRESS         262: 00010000000000000000000000000000
# ADDRESS         263: 01000000000000000000000000000000
# ADDRESS         264: 00000000000000000000000000000000
# ADDRESS         265: 00000000000000000000000000000000
# ADDRESS         266: 00000000000000000000000000000000
# ADDRESS         267: 00000000000000000000000000000000
# ADDRESS         268: 00000000000000000000000000000000
# ADDRESS         269: 00000000000000000000000000000000
# ADDRESS         270: 00000000000000000000000000000000
# ADDRESS         271: 00000000000000000000000000000000
# Break in Module TB at C:/Users/pouyn/OneDrive/Desktop/code-Q7.
```