پروژه پردازنده 512 بیتی – محمدمهدی عابدیان - 401110629

پروژه ما از چهار ماژول تشکیل شده است:

ماژول رجیسترفایل: چهار رجیستر 512 بیتی ما را نگه می‌دارد:

module RegisterFile (

    input clk,

    output reg [511:0] A1, A2, A3, A4

);

    reg [511:0] registers [3:0];

    always @(posedge clk) begin

        A1 <= registers[0];

        A2 <= registers[1];

        A3 <= registers[2];

        A4 <= registers[3];

    end

endmodule

و از هر رجیستر به بیرون ماژول خروجی دادیم.

ماژول مموری: 512 سطر 32 بیتی داده است.

module Memory (

    input clk,

    input [8:0] address,

    input [511:0] data\_in,

    input write\_enable,

    output reg [511:0] data\_out

);

    reg [31:0] memory [511:0];

    always @(posedge clk) begin

        if (write\_enable) begin

            for (integer i = 0; i < 16; i = i + 1) begin

                memory[address + i] <= data\_in[(i\*32) +: 32];

            end

        end

        for (integer i = 0; i < 16; i = i + 1) begin

            data\_out[(i\*32) +: 32] <= memory[address + i];

        end

    end

endmodule

می‌شود با فعال کردن write\_enable به صورت مستقیم محتویات data\_in که 512 بیت است را روی خانه با آدرس ورودی address نوشت و در هر زمان ورودی آدرس هر چه باشد، خروجی data\_out محتویات 512 بیت پس از آن آدرس را خواهد داشت.

ماژول ALU

module ALU (

    input [511:0] A1,

    input [511:0] A2,

    input control, // 1 برای جمع، 0 برای ضرب

    output reg [511:0] A3,

    output reg [511:0] A4

);

    always @(\*) begin

        if (control) begin

            {A4, A3} = A1 \* A2;

        end else begin

            {A4, A3} = A1 + A2;

        end

    end

endmodule

این ماژول خروجی A1,A2 رجیسترفایل را می‌گیرد با یک ورودی کنترلی control که در صورت خواستن جمع، صفر است و برای ضرب یک، عملیات مورد نظر را روی این دو انجام‌ می‌دهد و بخش کم‌ارزش خروجی روی A3 و بخش پرارزش آن روی A4 نوشته می‌شود.

ماژول پردازنده:

module Processor (

    input clk,

    input [1:0] reg\_select,

    input write\_enable,

    input [8:0] mem\_address,

    input [511:0] mem\_data\_in,

    input mem\_write\_enable,

    input ALU\_Control,

    input control, // ورودی کنترلی برای انتخاب بین جمع و ضرب

    input mem\_to\_reg\_enable,

    input mem\_to\_reg,

    output [511:0] A1, A2, A3, A4,

    output [511:0] mem\_data\_out

);

    wire [511:0] alu\_A3, alu\_A4;

    RegisterFile rf (

        .clk(clk),

        .A1(A1),

        .A2(A2),

        .A3(A3),

        .A4(A4)

    );

    ALU alu (

        .A1(A1),

        .A2(A2),

        .control(control), // ورودی کنترلی به ALU

        .A3(alu\_A3),

        .A4(alu\_A4)

    );

    Memory mem (

        .clk(clk),

        .address(mem\_address),

        .data\_in(mem\_data\_in),

        .write\_enable(mem\_write\_enable),

        .data\_out(mem\_data\_out)

    );

    // Write ALU results back to register file

    always @(posedge clk) begin

        if (ALU\_Control) begin

            rf.registers[2] <= alu\_A3; // نوشتن نتیجه کم ارزش به A3

            rf.registers[3] <= alu\_A4; // نوشتن نتیجه پر ارزش به A4

        end

        if (mem\_to\_reg\_enable) begin

            if (mem\_to\_reg) begin

            // Write from memory to register file

            rf.registers[reg\_select] <= mem\_data\_out;

            end else begin

            // Write from register file to memory

            for (integer i = 0; i < 16; i = i + 1) begin

                mem.memory[mem\_address + i] <= rf.registers[reg\_select][(i\*32) +: 32];

            end

            end

        end

    end

endmodule

دو سری سیم از خروجی A3 و A4 ALU به نام ALU\_A3 و ALU\_A4 می‌گذاریم که در صورت True بودن بیت کنترلی عملیات‌هایی که با ALU داریم (ALU\_Control)، آن را به رجیسترهای اصلی A3 و A4 رجیسترفایل متصل می‌کنیم.

همچنین درصورتی که بخواهیم داده‌ای را بین رجیسترفایل و مموری منتقل کنیم باید mem\_to\_reg\_enable 1 باشد و اگر بخواهیم از مموری به رجیسترفایل منتقل کنیم، mem\_to\_reg یک و در حالت برعکس صفر است.

همچنین اینکه از کدوم رجیستر بخوانیم یا به کدام بنویسیم با ورودی کنترلی reg\_select مشخص می‌شود.

همچنین یک ماژول تست بنچ داریم که صحت عملکرد را بسنجد:

module Processor\_tb;

    reg clk;

    reg [1:0] reg\_select;

    reg write\_enable;

    reg [8:0] mem\_address;

    reg [511:0] mem\_data\_in;

    reg mem\_write\_enable;

    reg control;

    wire [511:0] A1, A2, A3, A4;

    wire [511:0] mem\_data\_out;

    reg mem\_to\_reg\_enable;

    reg mem\_to\_reg;

    reg ALU\_Control;

    Processor uut (

        .clk(clk),

        .reg\_select(reg\_select),

        .write\_enable(write\_enable),

        .mem\_address(mem\_address),

        .mem\_data\_in(mem\_data\_in),

        .mem\_write\_enable(mem\_write\_enable),

        .control(control),

        .A1(A1),

        .A2(A2),

        .A3(A3),

        .A4(A4),

        .mem\_data\_out(mem\_data\_out),

        .mem\_to\_reg\_enable(mem\_to\_reg\_enable),

        .mem\_to\_reg(mem\_to\_reg),

        .ALU\_Control(ALU\_Control)

    );

    // Clock generation

    initial begin

        clk = 0;

        forever #5 clk = ~clk;

    end

    // Test sequence

    initial begin

        $monitor("ALU Control %b, Multiply Control %b, Mem\_to\_reg\_enable %b, Mem\_to\_reg %b\nmem\_data\_in %h", ALU\_Control, control, mem\_to\_reg\_enable, mem\_to\_reg, mem\_data\_in);

    end

    initial begin

        $display("Transfer data to memory");

        // Write to memory

        mem\_address = 9'h0;

        mem\_data\_in = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Example value

        mem\_write\_enable = 1;

        #10 mem\_write\_enable = 0;

        // From memory to register A1

        reg\_select = 2'b00;

        mem\_to\_reg\_enable = 1;

        mem\_to\_reg = 1;

        #20 mem\_to\_reg\_enable = 0;

        $display("A1: %h", mem\_data\_out);

        // Write to memory

        mem\_address = 9'h032;

        mem\_data\_in = 512'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // Example value

        mem\_write\_enable = 1;

        #10 mem\_write\_enable = 0;

        reg\_select = 2'b01;

        mem\_to\_reg\_enable = 1;

        // From memory to register A2

        mem\_to\_reg = 1;

        #20 mem\_to\_reg\_enable = 0;

        $display("A2: %h", mem\_data\_out);

        // Select Addition

        ALU\_Control = 1;

        control = 0;

        #30

        $display("A3 (Addition Result): %h", A3);

        $display("A4 (High Part, should be 0): %h", A4);

        control = 1; // Select multiplication

        #20;

        $display("A3 (Low Part of Multiplication Result): %h", A3);

        $display("A4 (High Part of Multiplication Result): %h", A4);

        ALU\_Control = 0;

        $display("Transfer data from A3 and A4 to memory");

        // read address 64 from memory

        mem\_address = 9'h64;

        reg\_select = 2'b10;

        mem\_to\_reg\_enable = 1;

        mem\_to\_reg = 0;

        #20 mem\_to\_reg\_enable = 0;

        $display("Memory Data Out from address %h: %h", mem\_address, mem\_data\_out);

        // read address 96 from memory

        mem\_address = 9'h96;

        reg\_select = 2'b11;

        mem\_to\_reg\_enable = 1;

        mem\_to\_reg = 0;

        #20 mem\_to\_reg\_enable = 0;

        $display("Memory Data Out from address %h: %h", mem\_address, mem\_data\_out);

        #10 $finish;

    end

endmodule

برای تست ابتدا یک مقدار را داخل ورودی mem\_data\_in مموری که همان data\_in آن است ی‌گذارد و با فعال کردن mem\_write\_enable آن را داخل مموری مینویسد. سپس reg\_select را برابر صفر می‌گذارد یعنی انتقال به رجیستر A1 و mem\_to\_reg را فعال می‌کند که داده از مموری به رجیستر نوشته شود. این عملیات یک بار دیگر با یک ورودی data\_in دیگر و reg\_select یک انجام می‌شود و حال A1 و A2 مقدار دارند.

سپس ورودی کنترلی عملیات ALU فعال می‌شود و یک بار control صفر و بار دیگر یک می‌شود که هم جمع و هم ضرب را ببینیم. سپس داده‌های رجیسترهای A3 و A4 به ترتیب در آدرس 64 و 96 مموری نوشته می‌شوند.

هر 4 خواسته پروژه انجام شده و در هر 4 عملیات این پردازنده موفق بوده.