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 ضرب ،0 برای جمع
    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 ALU A4 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'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // 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'hFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF; // 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 می‌گذارد 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.