گزارش کد سوال ۷

برای خواسته های سوال، ۴ ماژول طراحی شده است:

- Register file: این ماژول حاوی ۴ رجیستر ۵۱۲ بیتی به منظور انجام عملیات های ALU است و به آن متصل می شود.
- Memory: این ماژول یک ممووری با ۵۱۲ بیت عمق و ۳۲ بیت عرض به ما می دهد که قابلیت خواندن ۱۶ بلاک همزمان را دارد
- ALU: این ماژول، محاسبات را انجام می شود، ورودی هایش را از RF می گیرد و خروجی خود را نیز به آن برمیگرداند.
- CPU: این ماژول هم متصل کننده ی تمام این ماژول هاست و مسئول انجام عملیات های مورد نظر است با توجه به دستورات داده شده.

حال تک تک این ماژول ها را از نزدیک مقایسه می کنیم.

Register File

```
module Register_file #(
    parameter n=512
) (
    input wire clk,
    input wire [n-1:0] input_data1, input_data2,
    input wire [1:0] instruction, r_in,
    output reg [n-1:0] output_data, A1, A2, A3, A4
);
```

این ماژول شامل یک پارامتر ورودی که سایز ورودی ها را می گوید و همچنین، کلاک، ورودی ها و همچنین دستور مورد نظر و رجیستر مقصد را می گیرد و دیتای خروجی و رجیستر های A1~A4 را به عنوان خروجی در نظر می گیرد.

```
localparam LOADD = 2'b11,
 LOADD2 = 2'b10,
  LOADS = 2'b00,
   OUTR = 2'b01;
```

پارامتر های محلی نیز برای تمیز تر شدن کد تعریف کرده ایم که حالات مختلف instruction را پوشش می دهند.

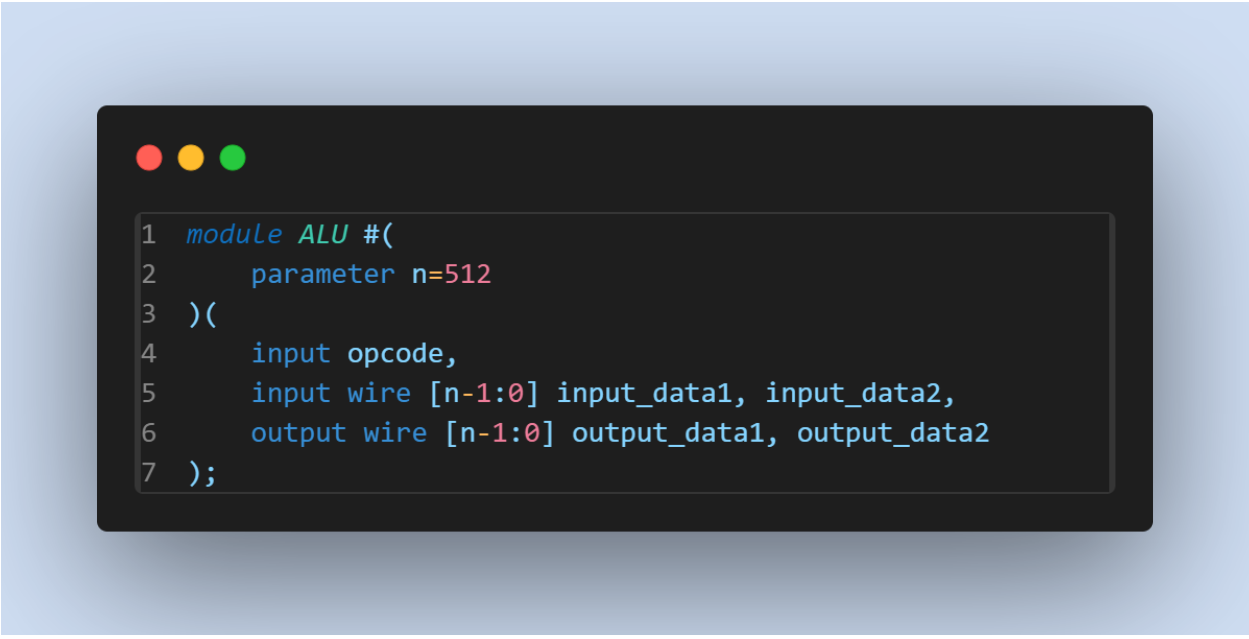
```
always @(negedge clk) begin
        if (instruction[1]) begin
            A3 <= input_data1;
            A4 <= input_data2;
        end
        else if (instruction == LOADS) begin
            case (r_in)
                2'b00:  A1 = input_data1;
                2'b01:  A2 = input_data1;
                2'b01:  A3 = input_data1;
                2'b11:  A4 = input_data1;
            endcase
        end
    end

```

در این ماژول دو حلقه ی اصلی داریم، یکی برای ورودی ها و یکی برای خروجی، این حلقه برای مقدار دهی رجیسترها بر اساس instruction و r_in است.

```
always @(posedge clk) begin
    if (instruction == OUTR) begin
        case  (r_in)
            2'b00:  output_data = A1;
            2'b01:  output_data = A2;
            2'b10:  output_data = A3;
            2'b11:  output_data = A4;
        endcase
    end
end
```

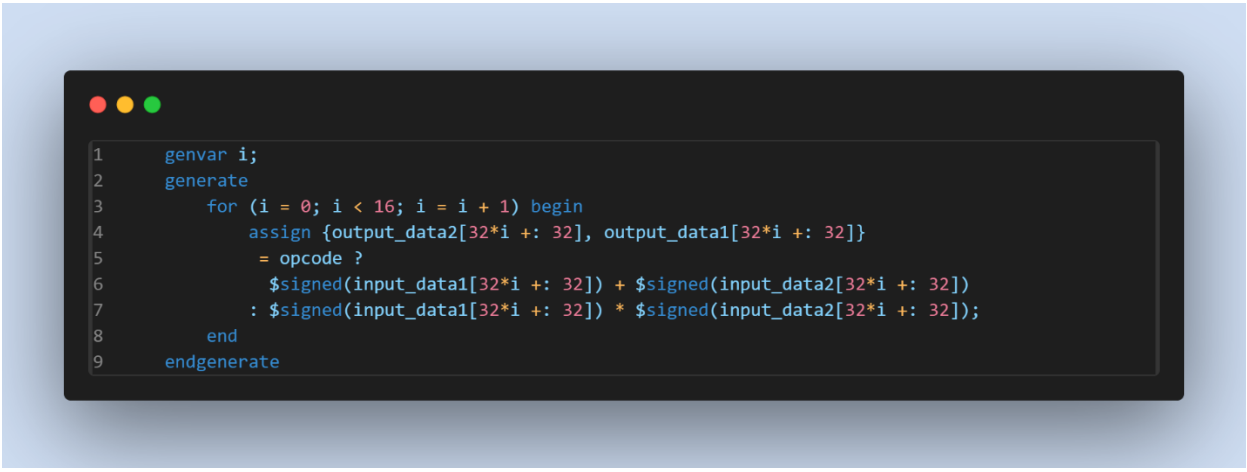
این نیز حلقه ی مربوط به مقدار دهی خروجی است.

ALU



```
module ALU #(
    parameter n=512
)(
    input opcode,
    input wire [n-1:0] input_data1, input_data2,
    output wire [n-1:0] output_data1, output_data2
);
```

این ماژول مسئولیت محاسبات را به عهده دارد، پارامتر n که سایز ورودی است را می گیرد، و سپس opcode که مشخص کننده ی عملیات است و همچنین ورودی ها را می گیرد و خروجی را به صورت high bit و low bit خروجی می دهد.



```
genvar i;
generate
    for (i = 0; i < 16; i = i + 1) begin
        assign {output_data2[32*i +: 32], output_data1[32*i +: 32]}
         = opcode ?
          $signed(input_data1[32*i +: 32]) + $signed(input_data2[32*i +: 32])
        : $signed(input_data1[32*i +: 32]) * $signed(input_data2[32*i +: 32]);
    end
endgenerate
```

در این ماژول از یک حلقه ی genvar استفاده شده است تا به ازای هر ۱۶ بیت ورودی، بخش پرارزش محاسبات را در output_data2 بریزد و بخش کم ارزش در output_data1

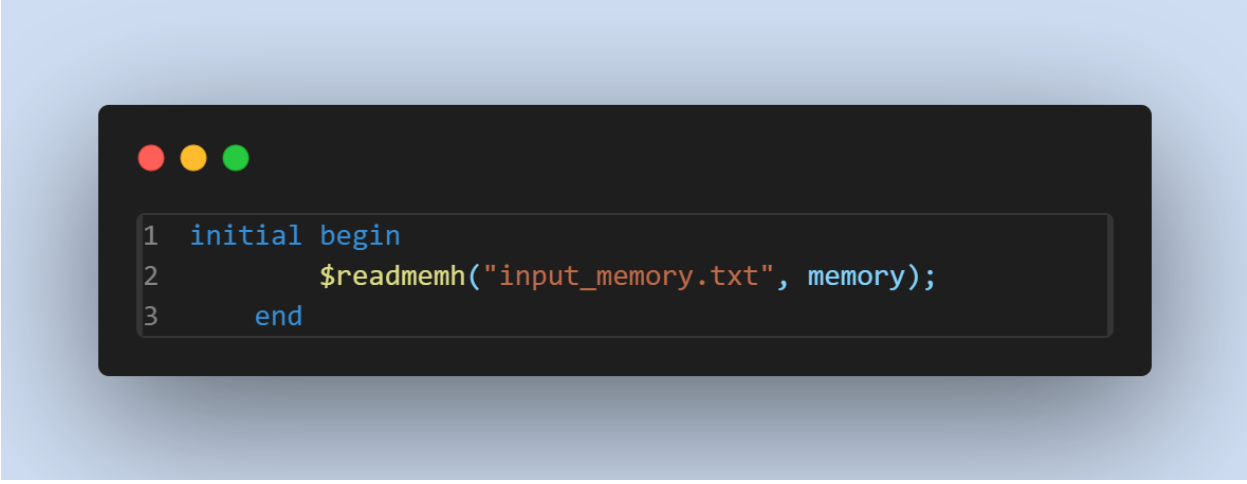
Memory

```
module Memory_Unit (
    input wire clk, rst, write_enable, output_enable,
    input wire [4:0] address,
    input wire [511:0] input_data,
    output reg [511:0] output_data
);
```

این ماژول از یک فایل مموری استفاده می کند تا حافظه ی ۵۱۲ در ۳۲ را برای ما نگهداری کند. ورودی های آن نیز به شرح تصویر است.

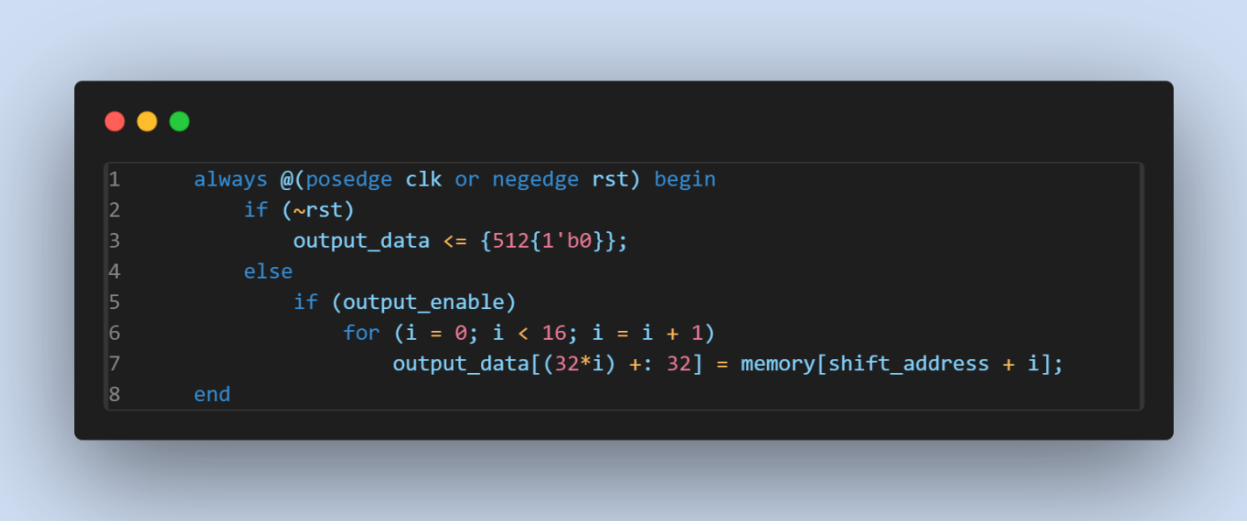
```
    reg [31:0] memory [0:511];
    wire [8:0] shift_address;
    integer i;

    assign shift_address = address << 4;
```

در این ماژول یک آرایه از خانه های ۳۲ بیتی داریم، به همراه آدرس های ۹ بیتی، با این تفاوت که فقط ۵ بیت از ورودی گرفته می شود زیرا ماژول این قابلیت را دارد که ۱۶ خانه ی متوالی را خروجی دهد، پس یک شیفت به چپ نیز مشاهده می شود.



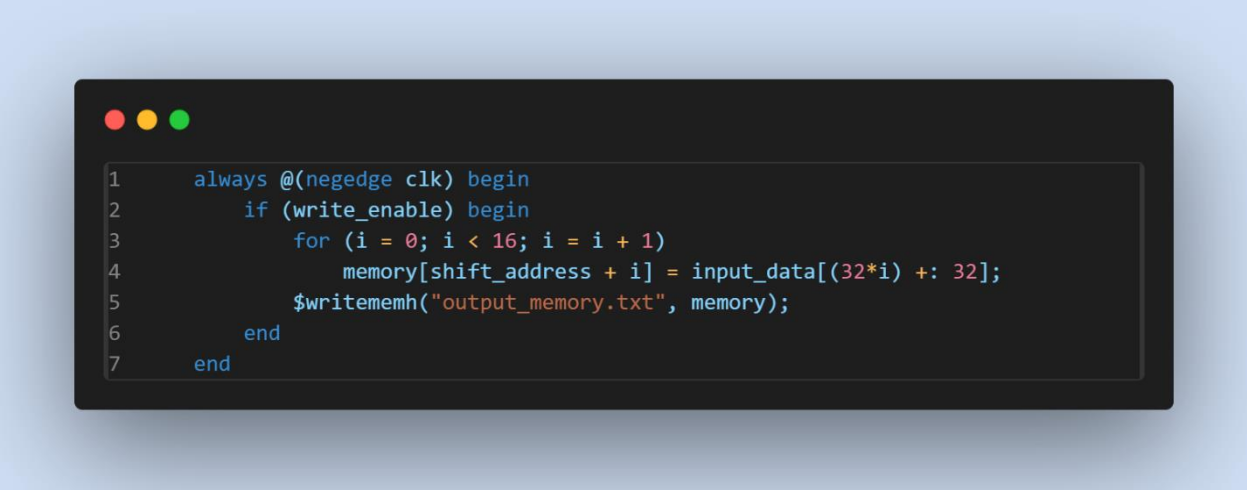
```
initial begin
        $readmemh("input_memory.txt", memory);
    end
```

این بلاک، مقدار دهی اولیه ی رجیستر ها را بر عهده دارد که از فایل می خواند.



```
always @(posedge clk or negedge rst) begin
    if (~rst)
        output_data <= {512{1'b0}};
    else
        if (output_enable)
            for (i = 0; i < 16; i = i + 1)
                output_data[(32*i) +: 32] = memory[shift_address + i];
end
```

در این حلقه، در لبه ی بالارونده ی کلاک، مقدار خروجی مشخص شده توسط آدرس در output_data قرار می گیرد.



```
always @(negedge clk) begin
    if (write_enable) begin
        for (i = 0; i < 16; i = i + 1)
            memory[shift_address + i] = input_data[(32*i) +: 32];
        $writememh("output_memory.txt", memory);
    end
end
```

و این بخش نیز در لبه ی پایین رونده ی کلاک عملیات نوشتن در مموری و فایل را انجام می دهد.

CPU

```
module CPU(
    input clk,
    input [8:0] instruction
);
```

ماژول CPU نیز، ماژولی است که تمام ماژول های بالاتر توضیح داده شده درون آن به همدیگر متصل شده اند.

```
    wire [511:0] A [0:3];
    wire [511:0] temp_array [0:6];
    wire [1:0] inst, r1;
    wire [4:0] addr;
    wire write_enable, output_enable;

    assign inst = instruction[8:7];
    assign r1 = instruction[6:5];
    assign addr = instruction[4:0];

    assign write_enable = ~instruction[8] & instruction[7];
    assign output_enable = ~instruction[8] & ~instruction[7];
    assign temp_array[0] = instruction[8] ? temp_array[2] : temp_array[5];
    assign temp_array[1] = instruction[8] ? temp_array[3] : temp_array[1];

    Register_file #(.n(512)) register_file (.clk(clk), .instruction(inst), .r_in(r1),
                                            .A1(A[0]), .A2(A[1]), .A3(A[2]), .A4(A[3]),
                                            .input_data1(temp_array[0]), .input_data2(temp_array[1]),
                                             .output_data(temp_array[4]));

    ALU #(.n(512)) alu (.input_data1(A[0]), .input_data2(A[1]),
                        .opcode(~instruction[7]), .output_data1(temp_array[2]), .output_data2(temp_array[3]));

    Memory_Unit memory_unit(.write_enable(write_enable), .output_enable(output_enable), .clk(clk),
                            .address(addr), .output_data(temp_array[5]), .input_data(temp_array[4]));
```

این ماژول مقدار زیادی سیم دارد برای اتصال های داخلی از مموری به RF و ALU، تمامی این سیم ها باید ۵۱۲ بیتی باشند، همچنین برای شکستن دستور به بخش های مختلف، سیم های inst, r1 و addr در نظر گرفته شده اند. سیم های werite_enable, output_enable نیز سیگنال های کنترلی مدار هستند که همینجا محاسبه می شوند و در آخر نیز بر اساس دستور تصمیم گرفته می شود که به سیم های temp_array چه سیمی متصل شود.

پایین تر نیز متصل شدن ماژول ها به یکدیگر مشاهده می شود.

مهدی علی نژاد، محمدامین عباسفر