در این بخش به حل سوال ۷ پرداخته شده است . پروژه ی مدنظر ساخت یک پردازنده آرایه ای ۵۱۲ بیتی می باشد که دارای ۳ بخش است : یک رجیستر فایل با قابلیت ذخیره سازی ۴ آرایه ۵۱۲ بیتی ، یک واحد ریاضی با قابلیت ضرب و جمع ، یک حافظه با عمق ۵۱۲ و عرض ۳۲ بیت است . مجموعه دستورات این پردازنده شامل بارگزاری از حافظه در یکی از ثبات ها ، ذخیره سازی از یکی ثبات ها به حافظه ، جمع واحد ریاضی و ضرب واحد ریاضی می باشد که در ادامه به بررسی دقیق تر آن پرداخته شده است.

ماژول RegisterFile

```
module RegisterFile (
    input clk,                  // Clock input
    input [1:0] rd_addr1,       // Read address 1
    input [1:0] rd_addr2,       // Read address 2
    input [1:0] wr_addr,        // Address of register to write
    input write_en,             // Write enable
    input [511:0] wr_data,      // Data to write
    output reg [511:0] rd_data1, // Data read from address 1
    output reg [511:0] rd_data2  // Data read from address 2
);

    reg [511:0] A1, A2, A3, A4; // Define four 512-bit arrays

    always @(posedge clk) begin
      if(write_en) begin
        // Write operation
        case (wr_addr)
            2'b00: A1 <= wr_data;
            2'b01: A2 <= wr_data;
            2'b10: A3 <= wr_data;
            2'b11: A4 <= wr_data;
            default: ; // No operation
        endcase

end
    else if(!write_en)begin
        // Read operation
        case (rd_addr1)
            2'b00: rd_data1 = A1;
            2'b01: rd_data1 = A2;
            2'b10: rd_data1 = A3;
            2'b11: rd_data1 = A4;
            default: rd_data1 = 512'h0; // Default value
        endcase

        case (rd_addr2)
            2'b00: rd_data2 = A1;
            2'b01: rd_data2 = A2;
            2'b10: rd_data2 = A3;
            2'b11: rd_data2 = A4;
            default: rd_data2 = 512'h0; // Default value
        endcase
end
end
endmodule
```

همانگونه که درتصویر بالا مشاهده می‌کنید ماژول ۶ ورودی دارد که ۲ ورودی برای انتخاب آدرس‌های خوانده‌شونده ، یک ورودی برای کلاک ، یک ورودی برای write enable و یک ورودی برای دریافت دیتایی که می‌خواهیم در آدرس مدنظر بنویسیم . همانگونه که در خط ۱۲ مشاهده می‌شود ۴ عدد آرایه ۵۱۲ بیتی خواسته ی سوال را

تعریف کرده ایم . حال به بررسی بلاک always میپردازیم که بخش ابتدایی آن به نوشتن در رجیسترفایل پرداخته شده است) در صورتی که write_en ، یک باشد یعنی قصد نوشتن را داریم که بعد از آن وارد بلاک switch caseمی‌شود که بسته به آدرس داده‌شده ، دیتا را دررجیستر مربوطه ذخیره می‌کند . (. در بخش بعد به خواندن از رجیستر فایل پرداخته‌شده‌است) دو ورودی rd_data1 و rd_data2 بررسی می‌شوند که این دو ورودی همانگونه که ذکرشد دارای آدرس‌هایی هستند که میخواهیم از رجیستر فایل بخوانیم و بسته به این آدرس‌های رجیسترها دیتاهای مدنظر وارد خروجی می‌شوند.).

ماژول MathUnit

```
module MathUnit(
    input [511:0] A1,        // First input
    input [511:0] A2,        // Second input
    input add,               // Addition enable
    input mul,               // Multiply enable
    output reg [511:0] low, // Low output
    output reg [511:0] high // High ouput
);
    always @(*) begin
        if (add) begin
            low = A1 + A2;  // Add two numbers
            high = 512'b0; // High not used in addition
        end else if (mul) begin
            {high, low} = A1 * A2; // Full 1024-bit multiplication
        end else begin
            //Default value
            high = 512'b0;
            low = 512'b0;
        end
    end
endmodule
```

همانگونه که در تصویر بالا مشاهده می‌کنید ماژول ۴ ورودی دارد که ۲ ورودی اول برای محاسبات هستند و یک ورودی add برای تشخیص جمع کردن و یک ورودی mul برای تشخیص ضرب دارد . در بلاک always اگر add، یک باشد عمل جمع را انجام داده و در low می ریزد و high را صفر می کند و اگر mul ، یک باشد عمل ضرب را انجام داده و باتوجه به سینتکس استفاده شده ۵۱۲ بیت کم ارزش را در low و ۵۱۲ بیت پرارزش را در high ریخته و خروجی می دهد و در غیر این صورت اگر هیچکدام از حالت های گفته شده نبود خروجی های ماژول را صفر می کند.

ماژول Memory

```
module Memory(
    input clk,                  // Clock input
    input [8:0] address,        // Address
    input [31:0] write_data,    // Data to write
    input write_enable,         // Write enable
    output reg [31:0] read_data // Data Output
);
    reg [31:0] memory [0:511];  // 512 x 32-bit memory

    always @(posedge clk) begin
        if (write_enable) begin
            // Write to Memory Address
            memory[address] <= write_data;
        end
        // Read from Memory Address
        else
        read_data <= memory[address];
    end
endmodule
```

همانگونه که در تصویر بالا مشاهده می کنید ماژول ۴ ورودی دارد که یک ورودی کلاک ، یک ورودی write enable، یک ورودی آدرس و یک ورودی دیتا می باشد . در خط ۸ Memory با عمق ۵۱۲ و عرض ۳۲ بیت مطابق خواسته ی سوال تعریف شده است . در بلاک always همانگونه که مشاهده می شود اگر write_enable، یک باشد یعنی عمل نوشتن در حافظه صورت بگیرد (دیتایی که به عنوان ورودی به ماژول دادیم را در آدرس خواسته شده در حافظه ذخیره می کند) و در غیر این صورت اگر write_enable صفر باشد ، آدرس داده شده از حافظه را خوانده و وارد خروجی می کند.

ماژول processor

```
module Processor(
    input clk,                      // Clock input
    input [8:0] mem_address,        // Memory address input
    input [31:0] mem_write_data,    // Data to write to memory
    input mem_write_enable,         // Memory write enable
    input [1:0] rd_addr1,           // Read address 1 for Register File
    input [1:0] rd_addr2,           // Read address 2 for Register File
    input [1:0] wr_addr,            // Write address for Register File
    input write_en,                 // Write enable for Register File
    input [511:0] wr_data,          // Data to write to Register File
    input add,                      // Addition control signal
    input mul,                      // Multiply control signal
    input load,                     // Load from memory to Register File
    input store,                    // Store from Register File to memory
    output [31:0] read_data,        // Data Output for Memory
    output [511:0] low,             // Low output of Math Unit
    output [511:0] high,            // High output of Math Unit
    output [511:0] rd_data1,        // Data read from address 1 in Register File
    output [511:0] rd_data2         // Data read from address 2 in Register File
);

    wire [31:0] mem_read_data;     // Data read from memory
    wire [511:0] data_to_write;    // Data to Write for RegisterFile
    // TriState-Buffer for Load enable
    assign data_to_write = load?read_data :wr_data;
    // TriState-Buffer for Store enable
    assign mem_read_data = store?rd_data2 :mem_write_data;
```

```
    RegisterFile RF (
        .clk(clk),
        .rd_addr1(rd_addr1),
        .rd_addr2(rd_addr2),
        .wr_addr(wr_addr),
        .wr_data(data_to_write),
        .write_en(write_en),
        .rd_data1(rd_data1),
        .rd_data2(rd_data2)
    );

    MathUnit mathunit (
        .A1(rd_data1),
        .A2(rd_data2),
        .low(low),
        .high(high),
        .add(add),
        .mul(mul)
    );

    Memory memory (
        .clk(clk),
        .address(mem_address),
        .write_data(mem_write_data),
        .write_enable(mem_write_enable),
        .read_data(read_data)
    );


endmodule
```

همانگونه که در تصویر مشاهده میکنید در این ماژول از سه ماژول دیگر که تعریف شده است instance گرفته و پورت های مربوطه را وصل کرده ایم . نکته ی قابل ذکر این است که برای دستورات بارگزاری از حافظه در یکی از ثبات ها و ذخیره سازی از یکی از ثبات ها به حافظه به عبارتی دستورات load و store از دو-TriState Bufferاستفاده کرده تا دستورات به درستی اجرا شوند.

ماژول tb_processor

```
module tb_Processor;

        reg clk;
        reg [8:0] mem_address;
        reg [31:0] mem_write_data;
        wire [31:0] read_data;
        reg mem_write_enable;
        reg [1:0] rd_addr1, rd_addr2, wr_addr;
        reg [511:0] wr_data;
        reg wr_en, add, mul, load, store;
        wire [511:0] low;
        wire [511:0] high;
        wire [511:0] rd_data1, rd_data2;

        Processor p (
            .clk(clk),
            .mem_address(mem_address),
            .mem_write_data(mem_write_data),
            .mem_write_enable(mem_write_enable),
            .rd_addr1(rd_addr1),
            .rd_addr2(rd_addr2),
            .wr_addr(wr_addr),
            .write_en(wr_en),
            .wr_data(wr_data),
            .add(add),
            .mul(mul),
            .load(load),
            .store(store),
            .low(low),
            .high(high),
            .rd_data1(rd_data1),
            .rd_data2(rd_data2),
            .read_data(read_data)
        );

        // Clock generation
        always #5 clk = ~clk;
```

```
    // Stimulus
    initial begin
        load = 0;
        clk = 0;
        wr_en = 1;
        wr_addr = 2'b10;
        wr_data = 512'h0000000100000001000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001;
        #10;
        wr_en = 1;
        wr_addr = 2'b00;
        wr_data = 512'h0000000300000002000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000;
        #10;
        wr_en = 0;
        rd_addr1 = 2'b10;
        rd_addr2 = 2'b00;
        #5;
        add = 1;
        mul = 0;
        #10;
        $display("data1 = %h", rd_data1);
        $display("data2 = %h", rd_data2);
        $display("low (add) = %h", low);
        $display("high (add) = %h", high);
        #10;
        add = 0;
        mul = 1;
        #10;
        $display("low (mul) = %h", low);
        $display("high (mul) = %h", high);

```

```
            // Test load/store
            mem_write_enable = 1;
            mem_address = 9'h0;
            mem_write_data = 32'h00010001;
            #10;
            mem_write_enable = 0;
            #5
            load = 1;

            wr_en = 1;
            wr_addr = 2'b01;
            #10;
            wr_en = 0;
            rd_addr1 = 2'b01;
            #10
            $display("Loaded data = %h into Register A2", rd_data1);
            load = 0;
            store = 1;
            mem_write_enable = 1;
            mem_address = 9'h0;
            #10;
            store = 0;
            mem_write_enable = 1;
            mem_address = 9'h0;
            $display("Stored data from Register A2 to memory address 0x1: %h",read_data);
            $finish;
        end

endmodule
```

همانگونه که مشاهده می شود از ماژول processor نمونه گرفته و پورت های مربوطه را تعریف کرده ایم . در بلاک always پورت مربوط به کلاک را تاگل کرده و در بلاک initial ، از خط ۴۱ تا ۵۰ ، دو عدد دیتا را در رجیستر های A1 و A3 رجیسترفایل نوشته و در خط ۵۰ تا ۵۳ آنها را خوانده و در خط ۵۸ و ۵۹ مقادیر رجیسترها را چاپ میکنیم . همانگونه در که در خط ۵۵ مشاهده میکنید ابتدا دستورا جمع را اجرا کرده و چاپ میکنیم و سپس دستور ضرب اجرا و چاپ می شود . برای بررسی دستور لود یا بارگزاری از حافظه در یکی از رجیستر ها ، مقداری را در حافظه نوشته و سپس آن را به رجیستر فایل متنقل کرده و برای بررسی صحت این بارگزاری آن را در خروجی پرینت می کنیم . برای بررسی دستور ذخیره سازی از یکی از ثبات ها به حافظه مقدار دیتا مدنظر را جا به جا کرده و آن را از خروجی حافظه می خوانیم.

ابتدا همه ی ماژول ها را compile کرده و پس از simulate کردن ماژول tb_processor وارد صفحه ی جدیدی شده و run-all را زده ، خروجی زیر را مشاهده می کنید:

```
VSIM 2> run -all
# data1 = 00000001000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001
# data2 = 00000003000000020000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# low (add) = 00000004000000030000000200000001000000010000000100000001000000010000000100000001000000010000000100000001000000010000000100000001
# high (add) = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# low (mul) = 00000006000000030000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
# high (mul) = 00000000000000030000000500000006000000060000000600000006000000060000000600000006000000060000000600000006000000060000000600000006
# Loaded data = 00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000010001 into Register A2
# Stored data from Register A2 to memory address 0x1: 00010001
# ** Note: $finish    : D:/University/DSD/project7/tb_Processor.v(94)
#    Time: 100 ps  Iteration: 0  Instance: /tb_Processor
```

اگر از ابتدا به بررسی خروجی بپردازیم ، همانگونه که مشاهده می کنید دو دیتای ذخیره شده به درستی نمایش داده شده اند و خروجی جمع نیز همانگونه که انتظار می رفت به درستی نمایش داده شده است . برای خروجی ضرب نیز ضرب دو عدد ذخیره شده در رجیستر فایل همانگونه که نمایش داده شده به درستی خروجی های lowو high را مشاهده می کنید . برای بررسی دستور load اگر ماژول تست را دوباره مشاهده کنید می بینید همان دیتایی که در حافظه ذخیره کردیم به درستی وارد رجیستر فایل شده و خروجی که از رجیستر فایل گرفتیم نیز با آن یکسان می باشد . درنهایت برای بررسی دستور store همانگونه که انتظار می رفت دیتا به درستی وارد حافظه شده و یکسان به خروجی حافظه می باشد ، در نتیجه به صحت کارکرد پردازنده ی آرایه ای که ساخته شده می رسیم.

پایان .