پیش گزارش آزمایش هشتم

روژین تقی زادگان ۴۰۱۱۰۵۷۷۵
رادین شاه دائی ۴۰۱۱۰۶۰۹۶
باربد شهرآبادی ۴۰۱۱۰۶۱۲۵

استاد محسن انصاری

توضیح آزمایش:

۹-۲ آزمایش هشتم: ALU اعداد مختلط

این آزمایش از چند بخش تشکیل شده‌است. در این آزمایش بایستی پیمانه‌های مختلف را پیاده‌سازی و تست کنید. سپس با کنار هم قرار دادن این پیمانه‌ها یک کامپیوتر پایه را پیاده‌سازی کنید. این ماشین فرضی دارای حافظه‌ی ۳۲ کلمه‌ای است.

الف- پیمانه جمع و تفریق اعداد مختلط

ب- ماژول ضرب اعداد مختلط

ج- یک واحد پایپ‌لاین که دستورات را از حافظه مذکور بخواند؛ سپس اجرای عملیات مختلط ورودی را به صورت پایپلاین انجام دهد.

در این آزمایش، نبایستی از واحدهای جمع کننده یا ضرب کننده بیش از یک بار استفاده شود.

شرح پیاده‌سازی: برای پیاده‌سازی این آزمایش، از ۴ ماژول ALU، RegisterFile، InstructionMemory و Processor بهره گرفتیم. تمام دستورات این پردازنده r-type هستند. رجیسترها شامل ۳۲ بیت بوده که ۱۶ بیت بالایی یا higher قسمت real عدد بوده و ۱۶ بیت پایینی یا lower قسمت imaginary عدد است. ALU ۳ ورودی opcode, operand1, operand2 را دارد که با توجه به opcode، result را مشخص می‌کند. این ماژول‌ها در Processor کنار هم آمده‌اند.

ماژول RegisterFile:

```
module RegisterFile(
    input wire clk,
    input wire [4:0] read_addr1,
    input wire [4:0] read_addr2,
    input wire [4:0] write_addr,
    input wire write_enable,
    input wire [31:0] write_data,
    output reg [31:0] read_data1,
    output reg [31:0] read_data2
);

reg [31:0] registers [15:0];

initial begin
    registers[0] = 32'h00010002; // Example: Higher 16 bits: 0001, Lower 16 bits: 0002
    registers[1] = 32'h00030004; // Example: Higher 16 bits: 0003, Lower 16 bits: 0004
end

always @(*)
begin
    read_data1 = registers[read_addr1];
    read_data2 = registers[read_addr2];
end

always @(posedge clk) begin
    if (write_enable) begin
        registers[write_addr] <= write_data;
    end
end

endmodule
```

این ماژول یک Register File عادی است که در درس معماری کامپیوتر با آن آشنا شدیم. این بانک ثبات با توجه به هر تغییر درون ورودی‌ها، خروجی‌های read_data1 و read_data2 را مشخص می‌کند (بدین معنی که آسنکرون هستند) و نیز در هر کلاک، اگر ورودی write_enable فعال بود، مقدار درون write_data را درون رجیستر مقصد قرار می‌دهد.

در این بانک ثبات، ۱۶ ثبات ۳۲ بیتی داریم که در هر کدام، یک عدد موهومی به صورت 16bit imaginary | 16bit real ذخیره شده‌است که در دستورات r-type این پردازنده استفاده خواهند شد.

با توجه به اینکه در ISA این پردازنده کوچک دستور load immediate را نداریم، در نتیجه با استفاده از بلوک initial begin مقدار درون رجیسترهای مختلف را مشخص می‌کنیم.

این ماژول در ماژول Processor کنار باقی ماژول‌ها می‌آید که در آینده آن را خواهیم دید.

ماژول InstructionMemory:

```
module InstructionMemory(
    input wire [4:0] address,
    output reg [15:0] instruction
);

reg [15:0] memory [31:0];

initial begin
    memory[0] = 16'h2012; // Instruction at address 0
    memory[1] = 16'h2012;
end

always @(*) begin
    instruction = memory[address];
end

endmodule
```

در این ماژول، دستورات را قرار می‌دهیم. این حافظه شامل ۳۲ دستور ۱۶ بیتی بوده و فرمت هر دستور به شکل زیر است:

4 bit opcode	4 bit src1	4 bit src2	4 bit dst

این حافظه نیز به صورت آسنکرون، از حافظه دستور می‌خواند. آدرس ورودی InstructionMemory در واقع همان pc بوده‌است که درون ماژول Processor از آن استفاده خواهیم کرد. در نظر داشته باشید که برای initialize کردن حافظه و قرار دادن دستورهای مختلف روی آن، از بلوک initial begin استفاده کرده‌ایم.

ماژول ALU:

```
module ComplexALU(
    input wire [31:0] operand1,
    input wire [31:0] operand2,
    input wire [3:0] opcode,
    output reg [31:0] result
);

always @(*) begin
    case(opcode)
        4'b0000: begin// Addition
            result[31:16] = operand1[31:16] + operand2[31:16];
            result[15:0] = operand1[15:0] + operand2[15:0];
        end
        4'b0001: begin// Addition
            result[31:16] = operand1[31:16] - operand2[31:16];
            result[15:0] = operand1[15:0] - operand2[15:0];
        end
        4'b0010: begin // Multiplication
            result[31:16] = (operand1[31:16] * operand2[31:16]) - (operand1[15:0] * operand2[15:0]);
            result[15:0] = (operand1[31:16] * operand2[15:0]) + (operand1[15:0] *operand2[31:16]);
        end
        default: // Default case
            result = 32'h0;
    endcase
end

endmodule
```

در این ماژول، با توجه به opcode های مختلف، مقدار result را به صورت آسنکرون محاسبه می‌کنیم. برای ضرب نیز باید مقداری محاسبه انجام دهیم که به حد کافی واضح می‌باشد.

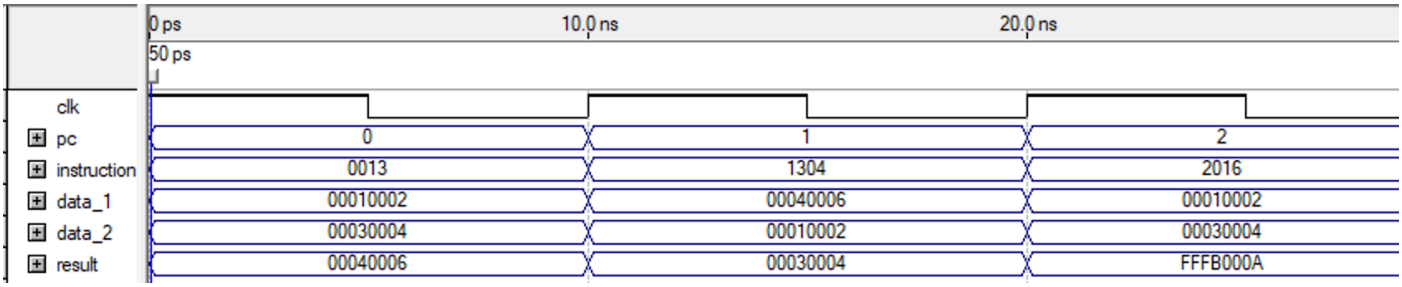
ماژول Processor:

```
module Processor(
    input wire clk,
    output reg [4:0] pc,
    output wire [31:0] result,
    output wire [15:0] instruction,
    output wire [31:0] data_1,
    output wire [31:0] data_2
);

// Instantiate Instruction Memory
InstructionMemory instr_memory(
    .clk(clk),
    .address(pc),
    .instruction(instruction)
);

// Instantiate Register File
RegisterFile reg_file_inst(
    .clk(clk),
    .read_addr1(instruction[11:8]),
    .read_addr2(instruction[7:4]),
    .write_addr(instruction[3:0]),
    .write_enable(1'b1),
    .write_data(result),
    .read_data1(data_1),
    .read_data2(data_2)
);

// Instantiate ALU
ComplexALU alu(
    .operand1(data_1),
    .operand2(data_2),
    .opcode(instruction[15:12]),
    .result(result)
);

always @(posedge clk) begin
    pc <= pc + 1;
end

endmodule
```

در این ماژول به سادگی، ماژول‌هایی که در بخش‌های قبل طراحی کردیم را کنار هم قرار می‌دهیم تا پردازنده کار کند. در صفحه‌ی بعد به تست این پردازنده می‌پردازیم:

برای تست این پردازنده، از سری دستورات زیر استفاده می‌کنیم:

```
r0 = 1+2i
r0 = 3+4i

add r0, r1, r3 : r3 = r1+r0 = 4 + 6i     = 0x0013
sub r3, r0, r4 : r4 = r3-r0 = 3 + 4i     = 0x1104
mul r0, r1, r6 : r6 = r0*r1 = 5 - 10i    = 0x2016
```



همانطور که مشاهده می‌شود، علاوه بر اینکه مقادیر درستی برای هر کدام از دستورات محسابه شد، مقدار درستی نیز درون register file (درون ثبات r3) ریخته می‌شود.