گزارش پروژه پردازنده ARM آزمایشگاه معماری کامپیوتر

آوا میرمحمدمهدی ۸۱۰۱۹۹۴۵۷ نسا عباسی مقدم ۸۱۰۱۹۹۴۵۷

جلسه اول: پیاده سازی پردازنده پایه ARM و ماژول IF

معماری پردازنده ARM به صورت Pipline است و شامل ۵ مرحله است :

۱. **Instruction Fetch:** دستور مورد نیاز از حافظه دستور خوانده شده و آدرس دستور بعدی در pc ذخیره میشود.

۲. **Instruction Decode:** دستور تفسیر شده و مشخص می‌شود چه فرآیندی برای اجرای دستور باید پیاده سازی شود.

۳. **Execution:** دستورات محاسباتی با توجه به سیگنالها اجرا می‌شود.

۴. **Memory:** برا دستوراتی که در آنها احتیاج به خواندن از مموری یا نوشتن در آن است، دسترسی به حافظه انجام می‌شود.

۵. **Write Back:** به روزرسانی رجیستر ها انجام می‌شود.

در جلسه اول برای هر کدام از مراحل گفته شده به همراه رجیسترهای بعد از آن، یک ماژول تشکیل داده و از تمامی این ۹ ماژول در ماژول top level به نام ARM، instance گرفتیم. در این مرحله تنها به طراحی IF پرداخته و باقی قسمت ها تنها وظیفه انتقال pc را برعهده دارند.

- **ماژول IF**

مرحله واکشی دستور (Instruction Fetch) با استفاده از ۴ ماژول پیاده سازی می‌شود. ورودی رجیستر pc توسط یک Mux انتخاب می‌شود بصورتیکه یکی از ورودی ها، خروجی ماژول Adder و دیگری آدرس branch است. از خروجی pc برای ورودی دو ماژول Instruction Memory و Adder استفاده می‌شود. ماژول Instruction Memory، pc را به عنوان آدرس میگیرد و دستور متناظر را خروجی می‌دهد و Adder، خروجی رجیستر pc را با ۴ جمع می‌کند.

```
module IF_Stage(clk, rst, freeze, branch_taken, branchAddr, pc, instruction);
    input clk, rst, freeze, branch_taken;
    input [31:0] branchAddr;
    output [31:0] pc, instruction;

    wire [31:0] adderOut;
    wire [31:0] PCout;
    wire [31:0] muxIFout;

    Adder adder(.a(PCout), .b(32'd4), .res(adderOut));
    InstructionMem instMem(.instruction(instruction), .address(PCout));
    Mux2To1 muxIF(.out(muxIFout),.in1(adderOut), .in2(32'd0), .sel(1'd0));
    PCRegister PCReg(.out(PCout), .in(muxIFout), .rst(rst), .freeze(1'd0), .clk(clk));

    assign pc = muxIFout;
endmodule
```

```
module Adder #(parameter bit = 32)(a, b, res);
    input [bit-1:0] a, b;
    output [bit-1:0] res;
    assign res = a + b;
endmodule
```

```
module InstructionMem #(parameter N = 32)(instruction, address);
    output reg[N-1:0] instruction;
    input [N-1:0] address;
    always @*
        case(address)
            32'd0: instruction = 32'b000000_00001_00010_00000_00000000000;
            32'd1: instruction = 32'b000000_00011_00100_00000_00000000000;
            32'd2: instruction = 32'b000000_00101_00110_00000_00000000000;
            32'd3: instruction = 32'b000000_00111_01000_00010_00000000000;
            32'd4: instruction = 32'b000000_01001_01010_00011_00000000000;
            32'd5: instruction = 32'b000000_01011_01100_00000_00000000000;
            32'd6: instruction = 32'b000000_01101_01110_00000_00000000000;
        endcase
   endmodule
```

```
module PCRegister # (parameter bit = 32)(out, in, rst, freeze, clk);
    input [bit - 1:0] in;
    input rst, freeze, clk;
    output reg [bit - 1:0] out;
    always@(posedge clk, posedge rst) begin
        if(rst)
            out <= {bit{1'b0}};
        else if(~freeze)
            out <= in;
        else
            out <= out;
    end
endmodule
```

```
module Mux2To1 # (parameter bit = 32)(out, in1, in2, sel);
    input [bit - 1:0] in1, in2;
    input sel;
    output [bit - 1:0] out;
    assign out = (sel == 1'b0) ? in1 : in2;
endmodule
```

- **ماژول IF_Register**

این رجیستر در ورودی سیگنالهای flush و freeze را به همراه ورودی ۳۲ بیتی pc_in و instruction_in دریافت کرده و pc_out و instruction را خروجی می‌دهد. freeze در واقع همان load رجیستر است که وقتی فعال باشد، ورودی را به خروجی منتقل می‌کند.

```
module IFIDreg #(parameter N = 32)(clk, rst, freeze, flush, pcIn, instructionIn, pcOut, instruction);
    input clk, rst, freeze, flush;
    input [N-1:0] pcIn;
    input [N-1:0] instructionIn;
    output reg [N-1:0] pcOut;
    output reg [N-1:0] instruction;

    always @(posedge clk or posedge rst) begin
        if(rst) begin
            pcOut <= 0;
            instruction <= 0;
        end
        else if(flush) begin
            pcOut <= 0;
            instruction <= 0;
        end
        else if(~freeze) begin
            pcOut <= pcIn;
            instruction <= instructionIn;
        end
    end
endmodule
```

- **باقی ماژول ها**

در این مرحله باقی ماژول تنها وظیفه انتقال pc را به عهده دارند و در قسمت posedge clk خروجی را برابر ورودی قرار می‌دهند.

```
module other_module #(parameter N = 32)(clk, rst, in, out);
    input clk, rst;
    input [N-1:0]in;
    output [N-1:0] out;
    assign out = in;
endmodule
```

```
module other_reg #(parameter N = 32)(clk, rst, in, out);
    input clk, rst;
    input [N-1:0]in;
    output reg [N-1:0] out;
    always @(posedge clk, posedge rst) begin
        if (rst)
            out <= {32'd0};
        else
            out <= in;
    end
endmodule
```

- **تست بنچ**

در نهایت با نوشتن تست بنچ عملکرد پردازنده را مورد بررسی قرار می‌دهیم.

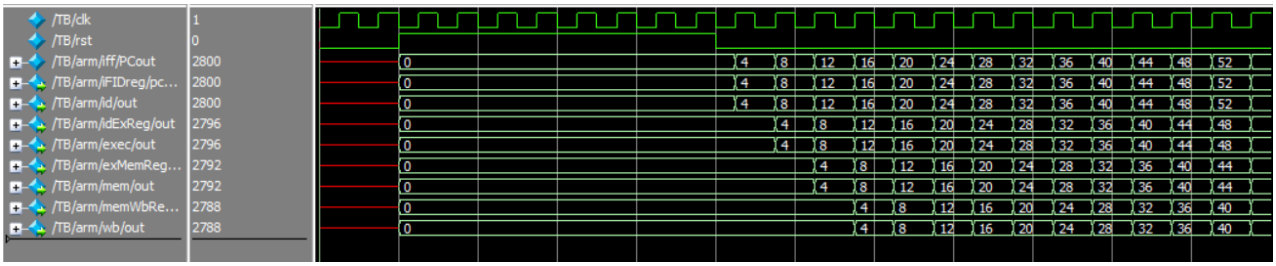
```
module TB();
    reg clk = 1'b0, rst = 1'b0;
    wire [31:0] out;
    ARM arm(clk, rst, out);
    always #5 clk = ~clk;
    initial begin
        #20 rst = 1'b1;
        #80 rst = 1'b0;
        #7000 $stop;
    end
endmodule
```

- Top module

```
module ARM(clk, rst, out);
    input clk, rst;
    output [31:0] out;
    wire freeze, branch_taken;
    wire [31:0] branchAddr, IFStagePcOut, IFRegPcOut, IFStageInstructionOut, IFRegInstructionOut;
    wire [31:0] IFout, IDout, IDExRegOut, execOut, exMemRegOut, memOut, memWbRegOut, IFIDregOut;

    IF iff(.clk(clk), .rst(rst), .freeze(freeze), .branch_taken(branch_taken), .branchAddr(branchAddr),
           .pc(IFStagePcOut), .instruction(IFStageInstructionOut));
    IFIDreg iFIDreg(.clk(clk), .rst(rst), .freeze(freeze), .flush(branch_taken), .pcIn(IFStagePcOut),
           .instructionIn(IFStageInstructionOut), .pcOut(IFRegPcOut), .instruction(IFRegInstructionOut));
    ID id(.clk(clk), .rst(rst), .in(IFRegPcOut), .out(IDout));
    IDExReg idExReg(.clk(clk), .rst(rst), .in(IDout), .out(IDExRegOut));
    Exec exec(.clk(clk), .rst(rst), .in(IDExRegOut), .out(execOut));
    ExMemReg exMemReg(.clk(clk), .rst(rst), .in(execOut), .out(exMemRegOut));
    Mem mem(.clk(clk), .rst(rst), .in(exMemRegOut), .out(memOut));
    MemWBReg memWbReg(.clk(clk), .rst(rst), .in(memOut), .out(memWbRegOut));
    WB wb(.clk(clk), .rst(rst), .in(memWbRegOut), .out(out));
endmodule
```

در شکل زیر حرکت موج گونه PC در مرحله های پایپ لاین مشخص است.



- **مشکلات و رفع آنها**

یکی از مشکلاتی که در زمان پروگرم کردن FPGA پیش آمد waiting شدن clk بود که که به دلیل import نکردن فایل pin assignment بود و پس از آن فراموشی کامپایل دوباره برنامه که با کمک تی ای ها و دیگر دانشجویان رفع شد.