به نام خدا

آزمایشگاه معماری کامپیوتر (CALab)

گزارش جلسه اول

ایمان رسولی پرتو 810199425 & پارسا حداد منفرد 810198380

← نصب Quartes II و driver setup و تست کردن بورد FPGA

این قسمت در آز منطقی انجام شده بود؛ با اتصال بورد و تست کردن switch ها برای روشن و خاموش شدن LED ها مراحل نصب برنامه، درایور و تست بورد verify میشه.

← پیاده سازی مرحله IF

مرحله IF شامل Fetch کردن دستور از حافظه Instruction هاست. این بخش شامل یک بدنه خواهد بود که PC رو آپدیت میکنه و بخش دیگه خوندن دستورها از INSTmem خواهد بود.(برای پیاده سازی نهایی، در پیاده سازی اولیه 5 دستور رو به شکل دستی وارد کردیم)

```
module IF_stage (
    input clk, rst, freeze, Branch_taken,
    input[31:0] BranchAddr,
    output reg[31:0] PC,Instruction
);

    reg [5:0] INSTmem [31:0];

    initial begin
        $readmemb("inst.txt", INSTmem);
    end

    always@(posedge clk,posedge rst)begin
        if(rst)
            PC <= 32'd0;
        else
            PC <= PC + 32'd4;
    end

    assign Instruction = INSTmem[PC];

    /*always@(PC)begin
      case(PC)
        0: Instruction = 32'b00000000001000100000000000000000;
        4: Instruction = 32'b00000000011001000000000000000000;
        8: Instruction = 32'b00000000101001100000000000000000;
        12:Instruction = 32'b00000000111010000001000000000000;
        16:Instruction = 32'b00000001001010100001100000000000;
        20:Instruction = 32'b00000001011011000000000000000000;
        24: Instruction = 32'b00000001101011100000000000000000;
        endcase

    end*/

endmodule
```

Fig1. کد نهایی قسمت IF – قسمت پیاده سازی اولیه کامنت شده

← پیاده سازی Stage های Pipeline و مشاهده Pipe شدن PC

پردازنده ARM به شکل Pipeline از پنج بخش IF, ID, EXE, MEM, WB تشکیل شده، برای پایپ کردن پردازنده بین هر دو مرحله باید یک Register قرار بدیم؛ در این جلسه بقیه مراحل غیر از IF فقط PC رو عبور میدن و کار دیگه‌ای انجام نمیدن. در نهایت این رجیسترها و قسمت های مختلف پردازنده رو بهم متصل میکنیم.

```
module ID_stage(input clk, rst, input [31:0] PC_in, output [31:0] PC);
  assign PC = PC_in;
endmodule

module ID_stage_reg(input clk, rst, input [31:0] PC_in, output reg [31:0] PC);
  always@(posedge clk)begin
    if(rst)
      PC <= 32'd0;
    else
      PC <= PC_in;
  end
endmodule
```

Fig2. پایپ لاین کردن ARM – بخش ID – بقیه بخش‌ها مشابه همین قسمت خواهد بود.

```
module ARM_ins(input clk, rst);

    wire[31:0] IF_out,IF_out_reg, ID_out, ID_out_reg, EXE_out,
               EXE_out_reg, MEM_out, MEM_out_reg, WB_out, WB_out_reg, Inst;
    wire inp;
    wire [31:0] init;

    assign inp = 1'b0;
    assign init = 32'd0;

    IF_stage IF1( .clk(clk), .rst(rst), .freeze(inp), .Branch_taken(inp),
                  .BranchAddr(init), .PC(IF_out), .Instruction(Inst));
    IF_stage_reg IFR1(.clk(clk), .rst(rst), .PC_in(IF_out), .PC(IF_out_reg));
    ID_stage ID1(.clk(clk), .rst(rst), .PC_in(IF_out_reg), .PC(ID_out));
    ID_stage_reg IDR1(.clk(clk), .rst(rst), .PC_in(ID_out), .PC(ID_out_reg));
    EXE_stage EXE1(.clk(clk), .rst(rst), .PC_in(ID_out_reg), .PC(EXE_out));
    EXE_stage_reg EXER1(.clk(clk), .rst(rst), .PC_in(EXE_out), .PC(EXE_out_reg));
    MEM_stage MEM1(.clk(clk), .rst(rst), .PC_in(EXE_out_reg), .PC(MEM_out));
    MEM_stage_reg MEMR1(.clk(clk), .rst(rst), .PC_in(MEM_out), .PC(MEM_out_reg));
    WB_stage WB1(.clk(clk), .rst(rst), .PC_in(MEM_out_reg), .PC(WB_out));
    WB_stage_reg WBR1(.clk(clk), .rst(rst), .PC_in(WB_out), .PC(WB_out_reg));

endmodule
```

Fig3. متصل کردن قسمت های مختلف به یکدیگر و ایجاد مسیر عبور PC

در نهایت از این module در قسمت کد instance arm گیری میکنیم و اون رو اجرا میکنیم. clk رو به Clock50 و rst رو به SW[2] متصل میکنیم.

← مشاهده سیگنال های خروجی در قسمت Signal Tap

در قسمت Signal Tap ابتدا Clock رو set میکنیم. سپس سیگنال‌هایی که میخواهیم ببینیم انتخاب میکنیم، در نهایت دوباره compile کرده و سیگنال‌ها رو مشاهده میکنیم (حساسیت به rst رو rising edge انتخاب میکنیم).

با انجام کارهای فوق شکل سیگنال PC بدین صورت خواهد بود:

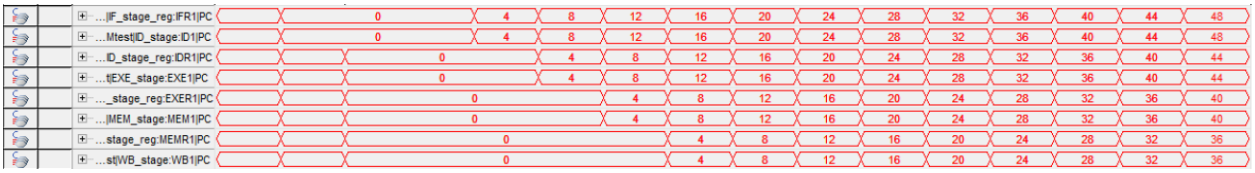

Fig4. شکل خروجی PC در Signal Tap

میبینیم که PC پایپ شده و جلو میره؛ بدین ترتیب بخش IF پردازنده ARM پیاده سازی میشه.