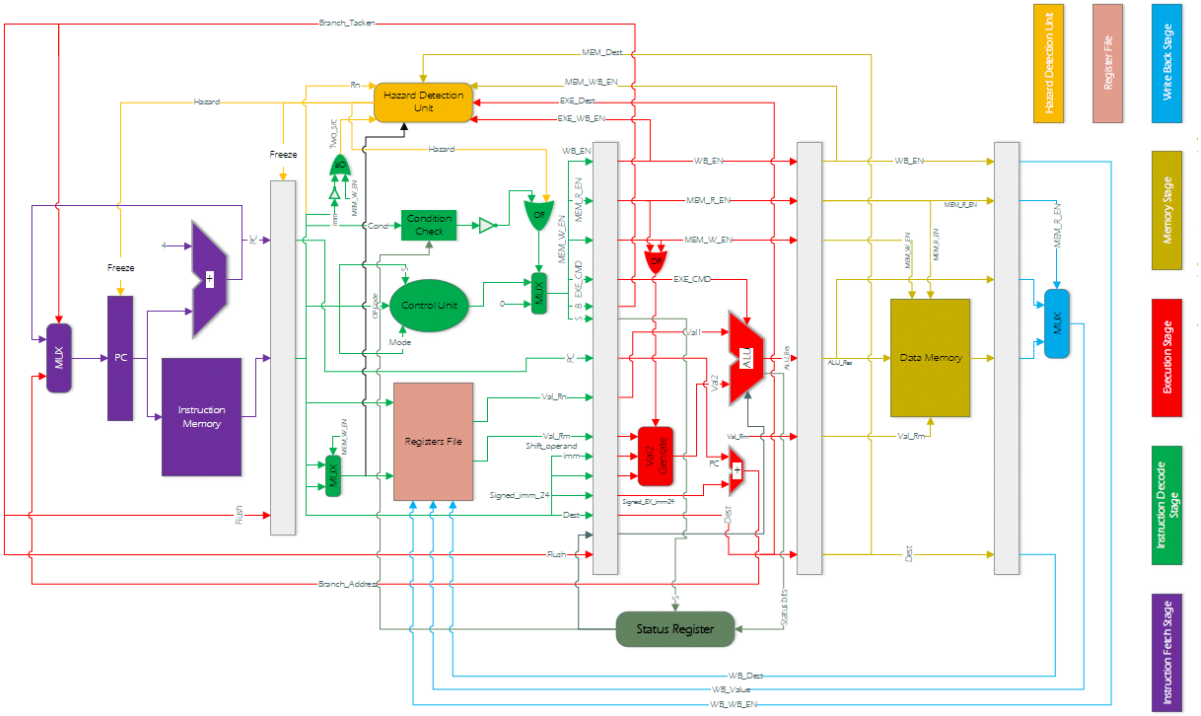
توضیحات آزمایش

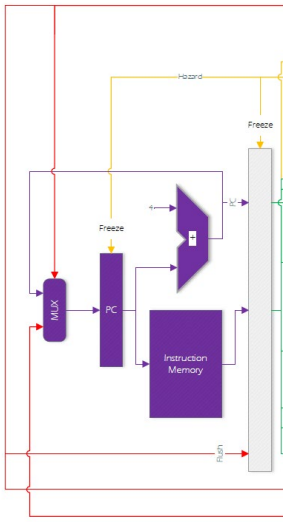
در این آزمایش باید پردازه ARM به صورت پایپ‌لاین پیاده‌سازی شود. دیاگرام این پردازنده به صورت زیر است:



این پردازنده دارای 13 دستور اصلی است. پیاده‌سازی باید در زبان وریلاگ باشد و در نهایت پس از شبیه‌سازی در نرم‌افزار ModelSim، با استفاده از نرم‌افزار Quartus سنتز می‌شود و روی FPGA قرار می‌گیرد. سپس، با استفاده از یک تست‌بنچ، پردازنده پیاده‌سازی شده تست می‌شود. از اهداف این آزمایش می‌توان به یادگیری نحوه عیب‌یابی و تست مدارهای سخت‌افزاری طراحی شده اشاره کرد.

جلسه اول

در این جلسه ماژول واکشی دستور (IF) پیاده‌سازی می‌شود:



1. ماژول‌های هر مرحله

به ازای هر مرحله پایپ‌لاین (5 مرحله) یک ماژول مشابه ماژول زیر ایجاد شده است:

```
module StageId(
    input clk, rst,
    input [31:0] pcIn,
    output [31:0] pcOut
);
```

همچنین برای رجیسترهای بین مراحل (4 تا) نیز یک ماژول مشابه ماژول زیر ایجاد شده است (4 تا):

```
module RegsIdEx(
    input clk, rst,
    input [31:0] pcIn,
    output [31:0] pcOut
);
```

2. ماژول IF و رجیستر بعد از آن

این 2 ماژول به صورت زیر ایجاد شده‌اند:

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

```
module RegsIfId(
    input clk, rst,
    input freeze, flush,
    input [31:0] pcIn, instructionIn,
    output [31:0] pcOut, instructionOut
);
```

درون `StageIf`، مطابق دیاگرام مسیر داده پردازنده، از یک عدد رجیستر برای PC، یک Adder برای جلو رفتن PC، Mux پشت سر PC و ماژول InstructionMemory استفاده شده است.

درون `RegsIfId`، یک ماژول Register قرار دارد که در هر کلاک PC را ذخیره می‌کند.

از آنجا که Quartus نمی‌تواند مستقیم فایل‌ای را توسط `$readmemb` بخواند و باید از ماژول‌های ROM خودش استفاده شود، برای پیاده‌سازی InstructionMemory، از case statement استفاده شد که بنا بر آدرس ورودی دستور متناظر در آن خانه حافظه را به صورت async خروجی می‌دهد:

```
module InstructionMemory #(
    parameter Count = 1024
)(
    input [31:0] pc,
    output reg [31:0] inst
);
    wire [31:0] adr;
    // Align address to the word boundary
    assign adr = {pc[31:2], 2'b00};

    always @(adr) begin
        case(adr)
          32'd0:  inst = 32'b000000_00001_00010_00000_00000000000;
          32'd4:  inst = 32'b000000_00011_00100_00000_00000000000;
          32'd8:  inst = 32'b000000_00101_00110_00000_00000000000;
          32'd12: inst = 32'b000000_00111_01000_00010_00000000000;
          32'd16: inst = 32'b000000_01001_01010_00011_00000000000;
          32'd20: inst = 32'b000000_01011_01100_00000_00000000000;
          32'd24: inst = 32'b000000_01101_01110_00000_00000000000;
        endcase
    end
endmodule
```

3. ماژول‌های مراحل دیگر

همانطور که در بخش اول ذکر شد، این ماژول‌ها با ورودی‌های ذکر شده ایجاد شدند، با این تفاوت که خروجی‌های ماژول‌های رجیستر بلاک به صورت `output reg` نیست زیرا رجیسترها توسط ماژول Register پیاده‌سازی می‌شوند و به صورت مستقیم در رجیستر بلاک پیاده‌سازی نمی‌شوند، بلکه از این رجیسترها نمونه گرفته می‌شود.

در همه این مراحل (یعنی به جز IF)، در Stageها، `assign pcOut = pcIn` شده و در Regsها (رجیسترهای میانی) از یک ماژول Register استفاده شده که هر کلاک `pcIn` را می‌گیرد و خروجی می‌دهد.

4. ماژول تاپ‌لول

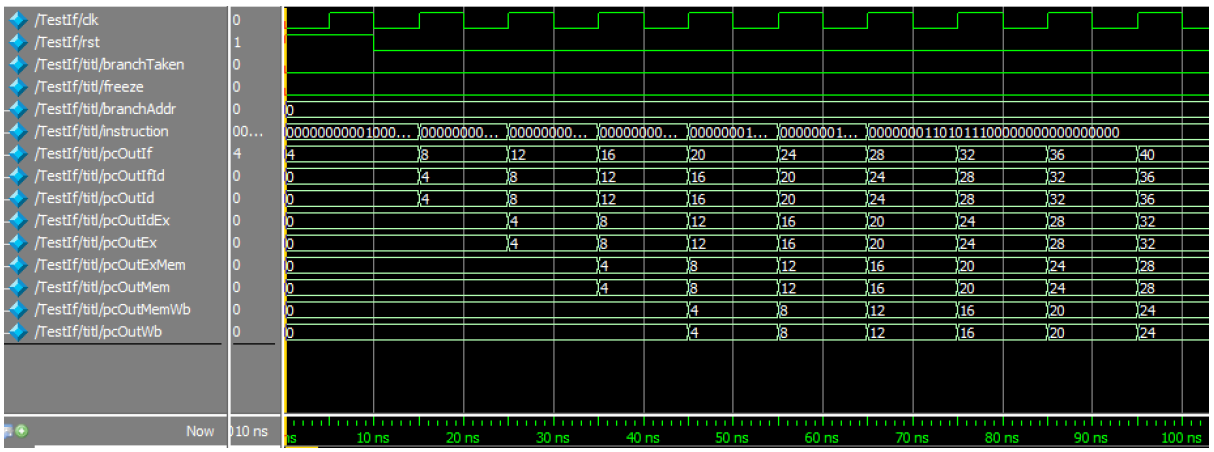
در این ماژول، 9 ماژول پایپ‌لاین به یکدیگر متصل شده‌اند:

```
module IfTopLevel(
    input clk, rst,
    output [31:0] pc
);
    wire branchTaken, freeze;
    wire [31:0] branchAddr;
    assign {branchTaken, freeze} = 2'b00;
    assign branchAddr = 32'd0;

    wire [31:0] instruction, instructionOut;
    wire [31:0] pcOutIfId, pcOutIdEx, pcOutExMem, pcOutMemWb;
    wire [31:0] pcOutIf, pcOutId, pcOutEx, pcOutMem, pcOutWb;

    StageIf stIf(clk, rst, branchTaken, freeze, branchAddr,
                 pcOutIf, instruction);
    RegsIfId regsIf(clk, rst, freeze, 1'b0, pcOutIf, instruction,
                    pcOutIfId, instructionOut);

    StageId stId(clk, rst, pcOutIfId, pcOutId);
    RegsIdEx regsId(clk, rst, pcOutId, pcOutIdEx);

    StageEx stEx(clk, rst, pcOutIdEx, pcOutEx);
    RegsExMem regsEx(clk, rst, pcOutEx, pcOutExMem);

    StageMem stMem(clk, rst, pcOutExMem, pcOutMem);
    RegsMemWb regsMem(clk, rst, pcOutMem, pcOutMemWb);

    StageWb stWb(clk, rst, pcOutMemWb, pcOutWb);

    assign pc = pcOutWb;
endmodule
```

در ماژول اصلی ARM که Quartus سنتز می‌کند، یک اینستنس از این ماژول تاپ‌لول گرفته شده است که کلاک آن به `CLOCK_50`، و rst آن به یکی از سوییچ‌های FPGA، `SW[0]` وصل شده است.

5. نتایج

خروجی ModelSim برای Test-bench نوشته شده به صورت زیر می‌باشد:



همانطور که مشاهده می‌کنیم، PC به درستی در هر کلاک به مرحله بعدی پایپ‌لاین می‌رود.

خروجی نتیجه کامپایل Quartus:

Flow Summary	
Flow Status	Successful - Fri Mar 10 22:53:37 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	arm11
Top-level Entity Name	ARM
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	4,108 / 33,216 (12 %)
Total combinational functions	1,319 / 33,216 (4 %)
Dedicated logic registers	3,889 / 33,216 (12 %)
Total registers	3889
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	36,992 / 483,840 (8 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

طبق گفته TA درس، نیازی به عکس گیری از اجرا بر روی FPGA و نتیجه SignalTap نبوده و در کلاس دیده شده است.

6. مشکلات و رفع آنها

یکی از مشکلاتی که در حین سنتز و استفاده از SignalTap به آن برخورد شد، خطای Waiting for clock پس از اجرای SignalTap بود.

پس از program کردن FPGA، لامپ‌های 7-Segment دستگاه فرم عجیبی داشتند و به طور رندم برخی از آنها روشن بود ولی برداشت خاصی از آن نکردیم.

در نهایت، فهمیدیم که مشکل، import نکردن فایل pin assignmentها بوده است.