به نام خدا

آزمایش پنجم _ Pipeline

امین ساوه دورودی – محمد رضا اسکینی – کامیاب عابدی

شرح اولیه

هدف از این آزمایش پیاده سازی پردازنده خط لوله با استفاده از پردازنده پیاده سازی شده تک چرخه ای در آزمایش قبلی میباشد. شماتیک کلی این نوع پردازنده ها شباهت زیادی با پردازنده تک چرخه ای دارد. در این پردازنده هدف انجام دستورات به صورت موازی میباشد. به طوری که در این پردازنده, Throughput به نسبت به پردازنده تک چرخه ای به طور محسوسی افزایش می یابد. برای پیاده سازی این نوع پردازنده, پردازنده تک چرخه ای را به 5 قسمت(فاز) تقسیم میکنیم. به طوری که همزمان پنج دستور (هر کدام در یک فاز) میتوانند اجرا شوند. در نتیجه Throughput 5 برابری نسبت به پردازنده تک چرخه ای خواهیم داشت. این 5 بخش عبارتند از :

Fetch, Decode, Execute, Memory, Writeback

در فاز Fetch, پردازنده دستور را از IM میخواند. در فاز Decode, پردازنده عملوند ها را از Regfile میخواند و همچنین دستورات را جدا سازی و decode میکند و به بخش کنترل میدهد. در فاز Execute, عملیات مورد نظر دستور توسط ALU انجام میشود. در فاز Memory, از روی data memory میخواند یا روی آن مینویسد. و در فاز آخر یعنی Writeback, حاصل بر روی regfile نوشته میشود. شماتیک کلی این پردازنده در تصویر(1) نشان داده شده است. همچنین رجیسترهایی نیز بین فازها اضافه میشوند تا بتوان راحت تر عملیات ها را مدیریت کرد.

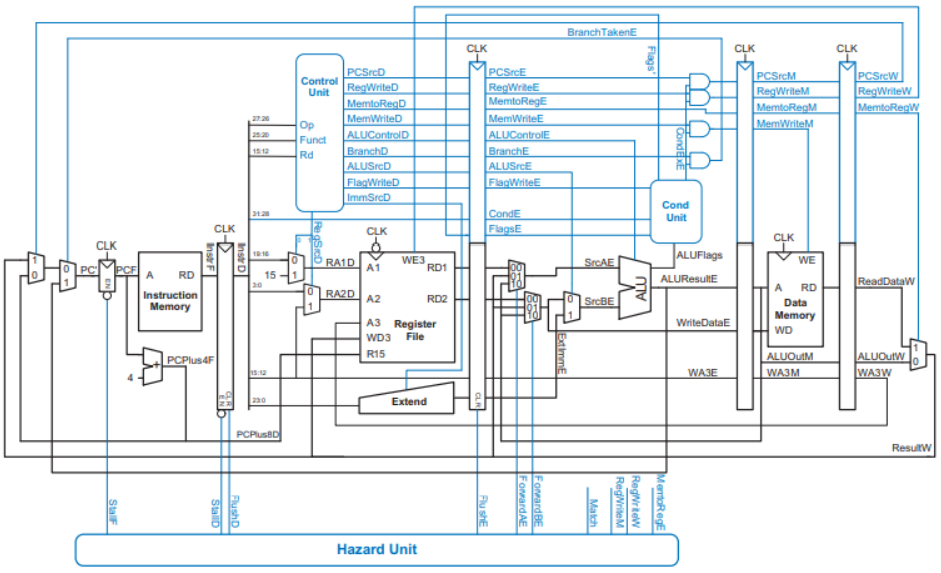


Figure 7.58 Pipelined processor with full hazard handling

تصویر 1 – شماتیک پردازنده خط لوله

در تصویر زیر(2) نیز میتوان تفاوت دیاگرام زمانی پردازنده خط لوله و تک چرخه ای را مشاهده کرد. در هر چرخه میتوان از تمامی فاز ها برای اجرای چند دستور استفاده کرد. به طوری که هر دستور در یک چرخه کامل اجرا نمیشود. اما به طور کلی در تعداد چرخه محدود تعداد دستورات بیشتری انجام میشود. همانطور که در تصویر(2) مشخص است, بعد از دو چرخه در پردازنده تک چرخه ای دو دستور به طور کامل اجرا شده است. اما در مدت زمانی کمتر در پردازنده خط لوله سه دستور به طور کامل اجرا شده است.

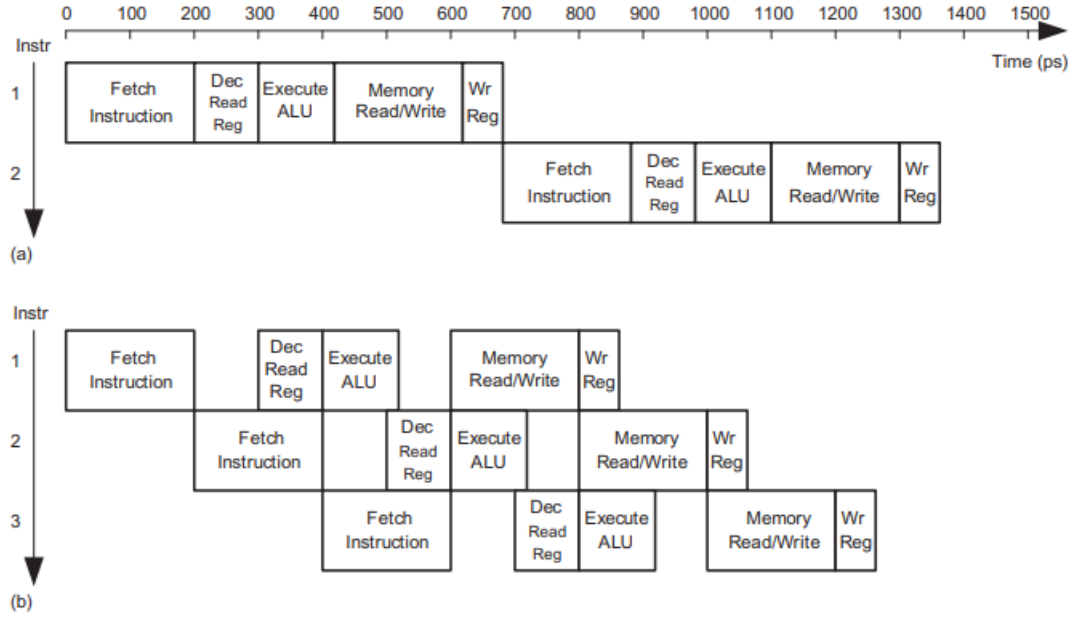


Figure 7.42 Timing diagrams: (a) single-cycle processor and (b) pipelined processor

تصویر 2 – دیاگرام زمانی دو پردازنده

امروزه, از پردازنده های خط لوله بسیار استفاده میشود. در مقایسه با پردازنده تک چرخه ای, پیچیده تر و سریعتر میباشد. به طوری که دستوراتی را به صورت موازی باهم اجرا میکند. به همین سبب, به صورت کلی, زمان کمتری برای اجرای تعداد مشخصی از دستورات صرف میکند. موازی اجرا کردن دستورات سبب ایجاد hazard هایی میشود که برای پیاده سازی این پردازنده نیاز است, واحدی برای کنترل این hazard ها پیاده سازی شود که باعث پیچیده تر شدن این پردازنده میشود.

در ادامه به بررسی تغییرات ایجاد شده در پردازنده تک چرخه ای برای پیاده سازی پردازنده خط لوله میپردازیم.

پیاده سازی

تنها ماژول هایی که در پیاده سازی این نوع پردازنده از روی پردازنده تک چرخه ای تغییر یا اضافه شده اند, توضیح داده میشوند.

در این پردازنده همانطور که در بالا ذکر شده است, 5 فاز وجود دارد و در بین هر فاز نیز رجیسترهایی برای مدیریت وضعیت وجود دارد. از آنجا که چند دستور همزمان اجرا میشوند, نیاز است که پردازنده مقدار های هر دستور را در انتهای هر فاز در رجیسترهای تعریف شده, ذخیره کنند. کاراکتر آخر نام هر رجیستر(F,D,E,M,W) نشان دهنده آن است که آن رجیستر مربوط به کدام فاز میباشد.

همانطور که در بالا ذکر شده است, به خاطر موازی بودن اجرای دستورات, نیاز به بخش جدیدی برای مدیریت و رفع hazard ها میباشد. پس این بخش جدید به ماژول cpu ما اضافه میشود. همچنین سیگنال های جدید و یا تغییرنام یافته, از تغییرات این ماژول میباشد.

```
controller c(clk, reset, FlushE, InstrD, ALUFlags, RegWrite1W, RegWrite2W, ALUSrcE, ALUControlE, PCSrcE, LMD,
ImmSrcD, LID,RegWrite1M, RegWrite2M, LME);

datapath dp(clk, reset, RegWrite1W, RegWrite2W, ALUSrcE, ALUControlE,
                PCSrcE, ALUFlags, PC, InstrF, InstrD, ALUResultE, ALUResult2E,
                LMD, DataAdrM, ReadDataM, ImmSrcD, SrcA , SrcB, LID,
                Match, ForwardAE, ForwardBE, FlushE, FlushD, StallD, StallF);

hazardunit hu(clk, reset, LME, PCSrcE, RegWrite1W, RegWrite1M, RegWrite2W, RegWrite2M,
                Match, ForwardAE, ForwardBE, FlushE, FlushD, StallD, StallF);
```

ماژول hazardunit

در هنگام اجرای موازی دستور ها ممکن است hazard هایی رخ بدهد که نیاز به ماژولی داریم تا آن ها را شناسایی کنیم و سیگنال هایی جهت رفع این hazard ها, مقداردهی کنیم و به بخش های مختلف برسانیم تا نحوه اجرای هر دستور در هر فاز پردازنده مشخص شود.به طور کلی در این ماژول, سه نوع سیگنال مقداردهی میشوند:

در صورتی که چند دستور در حال اجرا در پردازنده باشند و دستورها به مقداری موجود در دستور اول نیاز داشته باشند, دو رویکرد وجود دارد, میتوان آن مقدار را قبل از آنکه از تمامی فاز ها عبور کند, از یک فاز, آن مقدار را به دستور بعدی Forward کرد تا آن دستور نیز بتواند بدون توقف و اختلال به کار خود ادامه دهد. یا در بعضی از حالات نیاز است که دستورهای بعدی متوقف شوند (Stall) تا مقدار موردنظر در دستور قبلی آپدیت شود.

همچنین در مواقعی چند دستور در حال اجرا در پردازنده هستند و در دستور اول مشخص میشود که به طور مثال دستور Branch پذیرفته شده است و دیگر نیازی به دستورات در حال اجرا نیست, در این مواقع سیگنال های Flush مقداردهی میشوند تا جلوی ادامه روند دستور ها را بگیرند.

در ادامه کد این ماژول اورده شده است که با توجه به توضیحات بالا, سیگنال های مربوط به هر فاز مقدار دهی میشوند و به عنوان خروجی به دیگر بخش های cpu فرستاده میشوند تا بر اجرای موازی دستورات و عدم تداخل آن ها, مدیریت کنند.

```
module hazardunit(
		input clk, reset,
		input LME, JumpTaken,
		input RegWrite1W, RegWrite1M, RegWrite2W, RegWrite2M,
		input [7:0] Match,
		output reg [2:0] ForwardAE,
		output reg [2:0] ForwardBE,
		output FlushE, FlushD, StallD, StallF
		);
		wire LDRstall;
		assign LDRstall = (|Match) & LME;
		assign StallF = LDRstall ;
		assign StallD = LDRstall ;
		assign FlushE = (LDRstall | JumpTaken);
		assign FlushD = JumpTaken ;
		wire Match_1E_M_1, Match_2E_M_1, Match_2E_W_1, Match_1E_W_1,
		 Match_1E_M_2, Match_2E_M_2, Match_2E_W_2, Match_1E_W_2;
		assign {Match_1E_M_1, Match_2E_M_1, Match_1E_W_1, Match_2E_W_1,
		Match_1E_M_2, Match_2E_M_2, Match_2E_W_2, Match_1E_W_2} = Match;
		always @(*)
		begin
			if(Match_1E_M_1 & RegWrite1M) begin
				ForwardAE = 3'b001;
			end else if(Match_1E_W_1 & RegWrite1W) begin
				ForwardAE = 3'b010;
			end else if(Match_1E_M_2 & RegWrite2M) begin
				ForwardAE = 3'b011;
			end else if(Match_1E_W_2 & RegWrite2W) begin
				ForwardAE = 3'b100;
			end else begin
				ForwardAE =	3'b000;
			end
```

ماژول controller

```
decoder dec(InstrD[15], InstrD[14:6], FlagWD, PCSD, RegWrite1D, RegWrite2D, ALUSrcD,
                branchTypeD, LMD, ImmSrcD, LID);  ALUControlD,

pipeE pipe1(clk, FlushE, FlagWD, PCSD, RegWrite1D, RegWrite2D, ALUSrcD,  ALUControlD,
FlagWE, PCSE, RegWrite1E, RegWrite2E, ALUSrcE,  ALUControlE, branchTypeE,      branchTypeD, LMD,
  LME);

condlogic cl(clk, reset, ALUFlags, FlagWE, PCSE, branchTypeE, PCSrcE);

flopr #(2) pipe2(clk, 1'b0, {RegWrite1E, RegWrite2E}, {RegWrite1M, RegWrite2M});

flopr #(2) pipe3(clk, 1'b0, {RegWrite1M, RegWrite2M}, {RegWrite1W, RegWrite2W});
```

در این ماژول نیز با توجه به سیگنال دریافتی از بخش hazardunit, دو flopr و یک ماژول جدید به نام pipeE اضافه شده است. ماژول pipeE, با توجه به سیگنال FlushE, تعیین میکند که آیا باید رجیسترهای موجود در فاز Decoder به فاز Execute وارد شوند یا خیر.

همچنین دو flopr هم در انتهای فازها مشخص میکنند که آیا باید رجیستر ها وارد فاز بعدی شوند یا خیر.

دیگر ماژول های مورد استفاده شده در این ماژول, تغییری ندارند و صرفا ورودی و خروجی هاشون با توجه به اینکه مربوط به کدام فاز هستند, متفاوت میباشد.

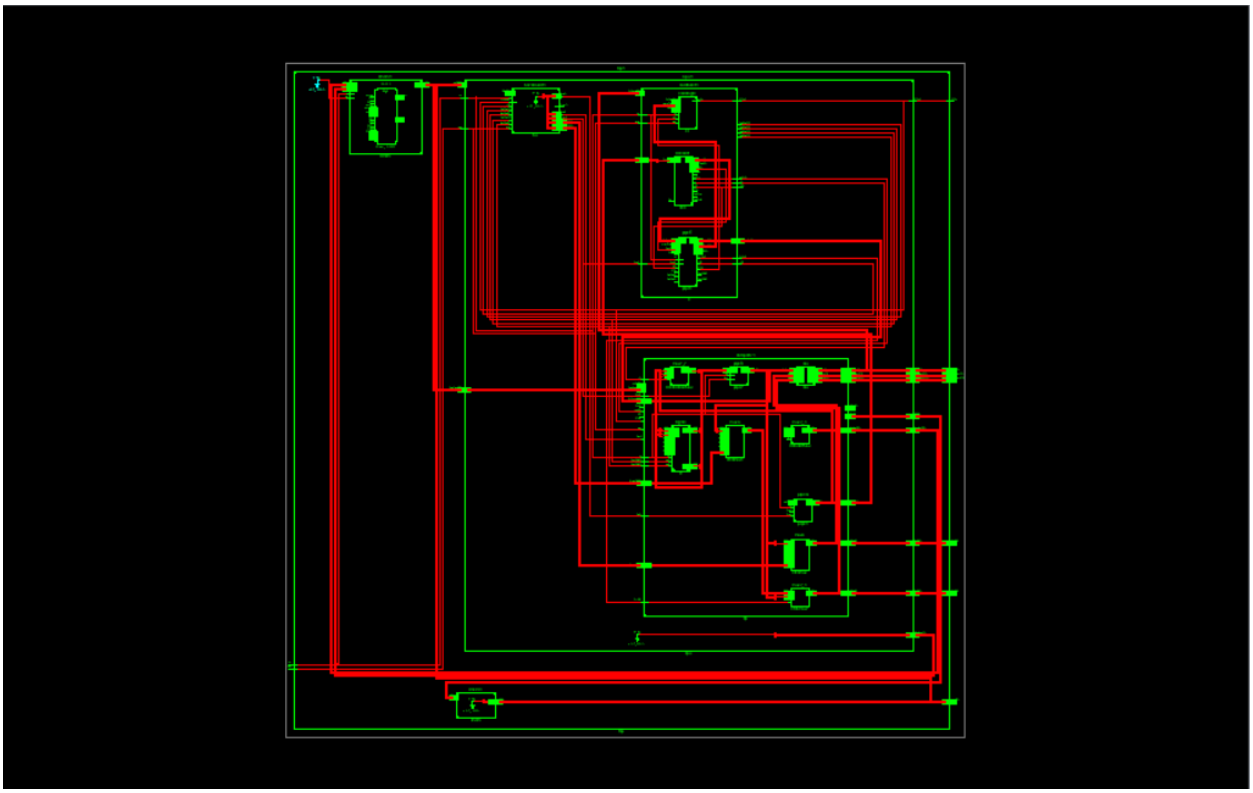
ماژول datapath

در این ماژول ابتدا, رجیسترهای هر فاز تعریف شده اند و ورودی ها و خروجی های ماژول ها نیز با توجه به اینکه در کدام فاز هستند, تغییر پیدا کرده است. همچنین در پایان هر فاز نیز ماژول هایی استفاده شده اند که با توجه به سیگنال های hazardunit, مشخص میکنند که آیا میتوان وارد فاز بعدی شد یا خیر. در انتهای فاز اول از ماژول pipeA استفاده شده است که با دریافت سیگنال FlushD و StallD مشخص میکند که آیا نیازی به اجرای بقیه دستور میباشد یا خیر (Flush) و همچنین آیا نیاز به توقف در این چرخه برای رسیدن مقادیر مورد نیاز از دستورات قبلی میباشد یا خیر.(Stall) در صورتی که منعی برای ورود به فاز بعدی نباشد مقدار InstrD را به فاز بعدی منتقل میکند. در انتهای فاز دوم از ماژول pipeB استفاده شده است که با توجه به سیگنال FlushE مشخص میکند که آیا میتوان وارد فاز بعدی با همین مقادیر شد یا تمامی مقادیر صفر شوند. در صورت صفر بودن این سیگنال رجیسترهای این فاز وارد فاز بعدی میشوند.در انتهای فاز سوم و چهارم همانطور که در تصویر(1) نمایان است, نیازی به بررسی سیگنال های hazardunit نمیباشد و تنها از flopr هایی برای عبور دادن رجیستر ها به فازهای بعدی استفاده میکنیم.در بخش پایانی این ماژول نیز, رجیستر Match مقدار دهی میشود. از این رجیستر در ماژول hazardunit برای تعیین سیگنال های مربوط به Forward استفاده میشود. با مقایسه رجیسترهای دستورات بررسی میکند که آیا دستورات جلوتر به دستورات قبلی وابسته هستند یا خیر. و اگر هستند امکان عملیات Forwarding هست یا خیر. باقی ماژول های این بخش نیز بدون تغییر می مانند.

```
mux2 #(8) pcmux(PCPlus4, InstrE[7:0], PCSrcE, PCNext);
                flopenr #(8) pcreg(clk, reset, ~StallF, PCNext, PC);
                adder #(8) pcadd1(PC, 8'b100, PCPlus4);
                ////Stage1//////
                pipeA pipe1(clk, StallD, FlushD, InstrF, InstrD);
                // register file logic
                mux2 #(3) destinationMux(InstrD[5:3], InstrD[10:8], LID, RA1D);
                assign RA2D = InstrD[2:0];
                regfile rf(clk, RegWrite1W, RegWrite2W, RA1D, RA2D, RA1W, RA2W, ResultW, ALUResult2W, RD1D, RD2D);
                extend ext(InstrD[2:0], InstrD[7:0], ImmSrcD, ExtImmD);
                mux2 #(8) resmux(ALUResultW, ReadDataW, LMW, ResultW);
                ////stage2/////
                pipeB #(39) pipe2(clk, FlushE, {RA1D, RA2D, RD1D, RD2D, ExtImmD, LMD, InstrD[7:0]},
                                  {RA1E, RA2E, RD1E, RD2E, ExtImmE, LME, InstrE[7:0]});
                // ALU logic
                mux5 #(8) srcamux (RD1E, ALUResultM, ResultW, ALUResult2M, ALUResult2W, ForwardAE, SrcA);
                mux5 #(8) srcbmux1(RD2E, ALUResultM, ResultW, ALUResult2M, ALUResult2W, ForwardBE, midSrcB);
                mux2 #(8) srcbmux2(midSrcB, ExtImmE, ALUSrcE, SrcB);
                alu alu(ALUControlE, SrcA, SrcB, ALUResultE, ALUResult2E, ALUFlags);
                ////stage3//////
                flopr #(31) pipe3(clk, 1'b0, {ALUResultE, ALUResult2E, RA1E, RA2E, LME, InstrE[7:0]},
                                  {ALUResultM, ALUResult2M, RA1M, RA2M, LMM, InstrM[7:0]});
                //For LM instruction
                mux2 #(8) DataAdrMux(8'b00000000, InstrM[7:0], LMM, DataAdrM);
                /////stage4//////
                flopr #(31) pipe4(clk, 1'b0, {ALUResultM, ALUResult2M, RA1M, RA2M, LMM, ReadDataM},
                                  {ALUResultW, ALUResult2W, RA1W, RA2W, LMW, ReadDataW});
                // Match Logic
                wire Match_1E_M_1, Match_2E_M_1, Match_2E_W_1, Match_1E_W_1,
                 Match_1E_M_2, Match_2E_M_2, Match_2E_W_2, Match_1E_W_2;
                assign Match_1E_M_1 = (RA1E == RA1M);
                assign Match_2E_M_1 = (RA2E == RA1M);
                assign Match_1E_W_1 = (RA1E == RA1W);
                assign Match_2E_W_1 = (RA2E == RA1W);
                assign Match_1E_M_2 = (RA1E == RA2M);
                assign Match_2E_M_2 = (RA2E == RA2M);
                assign Match_1E_W_2 = (RA1E == RA2W);
                assign Match_2E_W_2 = (RA2E == RA2W);
                assign Match = {Match_1E_M_1, Match_2E_M_1, Match_1E_W_1, Match_2E_W_1,
                 Match_1E_M_2, Match_2E_M_2, Match_2E_W_2, Match_1E_W_2};
endmodule
```

ماژول های دیگر بدون تغییر نسبت به پردازنده تک چرخه ای باقی می مانند.

تصویر(3) شماتیک RTL این پردازنده را نشان می دهد.



تصویر 3 – RTL

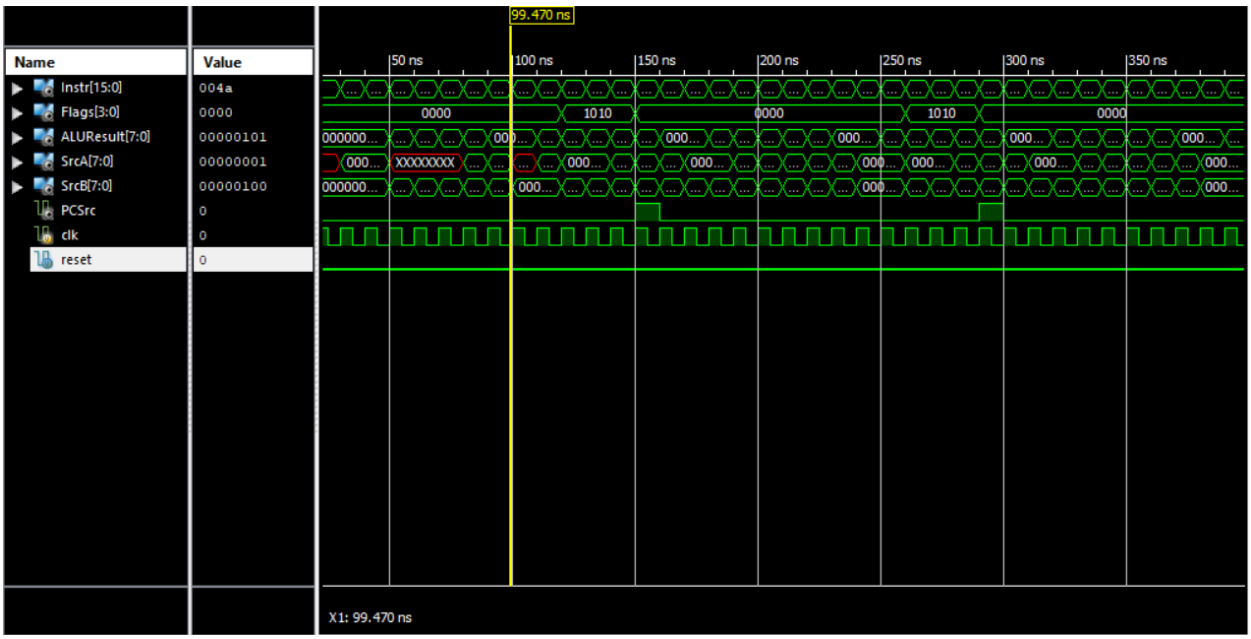
تست

برای تست صحت عملکرد پردازنده ابتدا دستوراتی را آماده میکنیم. جهت مقایسه بهتر این دستورات همان دستورات استفاده شده در آزمایش پردازنده تک چرخه ای میباشند.

```
LI R0 #3 //R0 = 3
LI R1 #2 //R1 = 2
LI R2 #1 //R2 = 1
LI R3 #4 //R3 = 4
OR R1 R0 //R1 = 3
XOR R2 R3 //R2 = 5
MOV R5 R2 //R5 = 5
ADD R1 R2 //R1 = 8
SUB R2 R1 //R2 = -3
XCHG R5 R0 // R0 = 5 R5 = 3
NOT R2 // R2 = 11111101 R2 = 00000010 = 2
JL to start //not accept
SAR R0 #2 //R0 = 1
SLR R3 #1 //R3 = 2
SAL R3 #4 //R3 = 32
SLL R1 #3 //R1 = 64
DEC R0 #1 // R0 = 0
DEC R0 #1 // R0 = -1
INC R0 #1 // R0 = 0
NOP
CMP R0 R1 //R0-R1 = 0-64=-64
LI R4 #5 // R4 = 5
LI R6 #15 // R6 = 15
ShowR R4
JA to start //not accept
ShowR R6
AND R6 R4 // R6 = 5
ShowRseg R4
ShowRseg R6
CMP R6 R4
JE to JMP0 //accept so R7 doesn't affect
LI R7 #7
JMP to 0
```

تصویر4 – دستورهای تست

دستورهای تست(تصویر(4)) را با استفاده از جدول موجود در دستورکار به هگزادسیمال تبدیل میکنیم و درون فایل memfile.dat قرار میدهیم تا حافظه دستور بتواند دستورات را از آن فایل بخواند.



تصویر5- نتیجه شبیه سازی

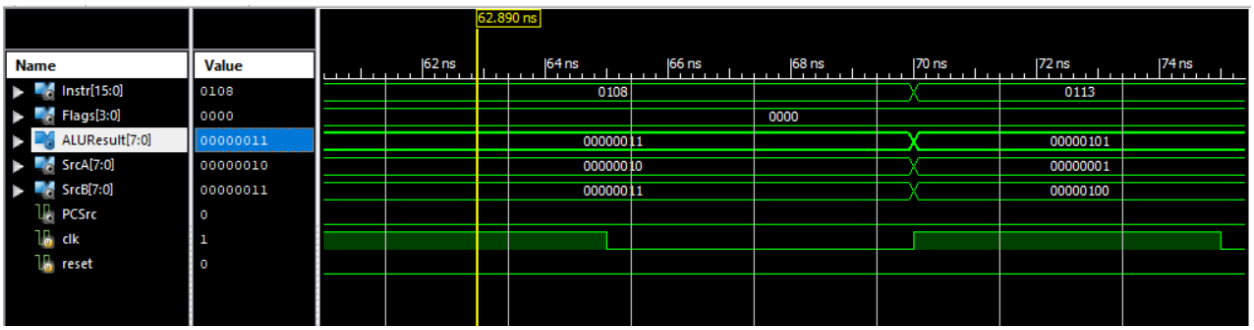
تصویر(5) نتیجه شبیه سازی پردازنده خط لوله را نشان میدهد که در ادامه به بررسی صحت عملکرد چند دستور میپردازیم.

نکته مهم و تفاوت بین دو پردازنده تک چرخه ای و خط لوله همانطور که در تصویر(2) مشخص است, این است که بر خلاف پردازنده تک جرخه ای , تمامی دستور در یک چرخه انجام نمیشود و از خواندن دستور از فایل(Fetch) تا رسیدن به جواب(Execute) دو چرخه اختلاف است. پس برای یافتن جواب های هر دستور (ALUResult) نیاز است که دو چرخه جلوتر از فاز Fetch دنبال آن ها باشیم.

```
OR R1 R0 //R1 = 3
XOR R2 R3 //R2 = 5
```

تصویر 6

برای تست عملکرد پردازنده ابتدا دو دستور بالا رو مورد بررسی قرار میدهیم.



تصویر1-6 – اجرا شده در پردازنده تک چرخه ای

تصویر6-2 – اجرا شده در پردازنده خط لوله – فاز Fetch

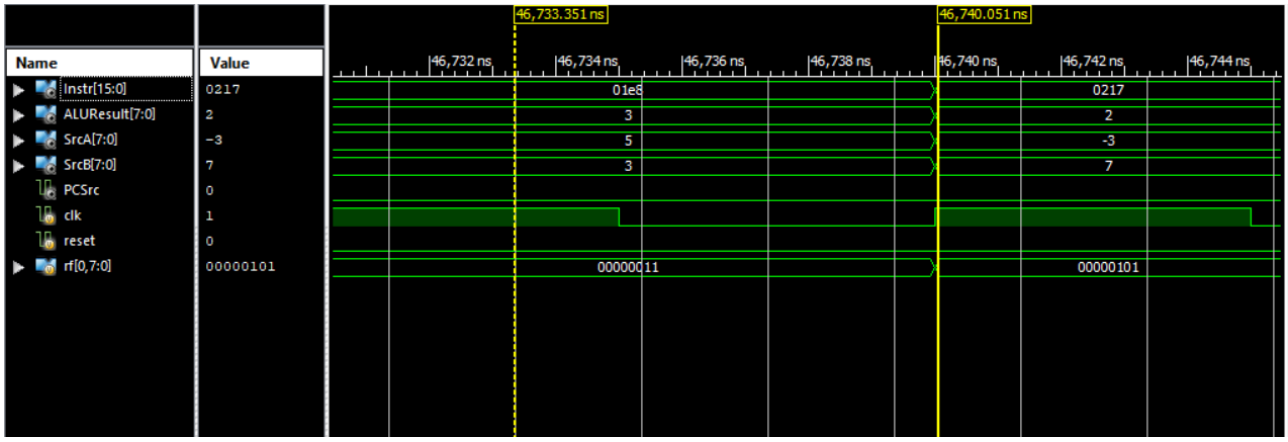
تصویر6-3 – اجرا شده در پردازنده خط لوله – فاز Execution

همانطور که در تصویر(6) مشاهده میکنیم انتظار داریم ALUResult ما در چرخه اول برابر با 3 باشد که همانطور که میبینیم عملیات or بر روی دو علموند انجام شده است و نتیجه 3 در خروجی ALU قرار گرفته است. همچنین در چرخه بعدی نیز بر روی دو عملوند عملیات xor انجام شده است که نشان دهنده این است هر دو بخش کنترل(تعیین نوع عملیات) و بخش مسیر داده(بارگذاری رجیستر ها) به درستی عمل کرده اند.

تصویر(1-6) در پردازنده خط لوله همانطور که در بالا اشاره شد, باید در دو چرخه بعد از Fetch به دنبال ALUResult باشیم. همانطور که در تصویر(3-6) مشخص است در دو چرخه بعد از فاز Fetch, مقدار ALUResult مطابق انتظارات ما بوده است. همچنین این شبیه سازی نشان میدهد که برای این عملیات, هر فاز به درستی وظیفه اش را انجام داده است.

```
XCHG R5 R0 // R0 = 5 R5 = 3
```

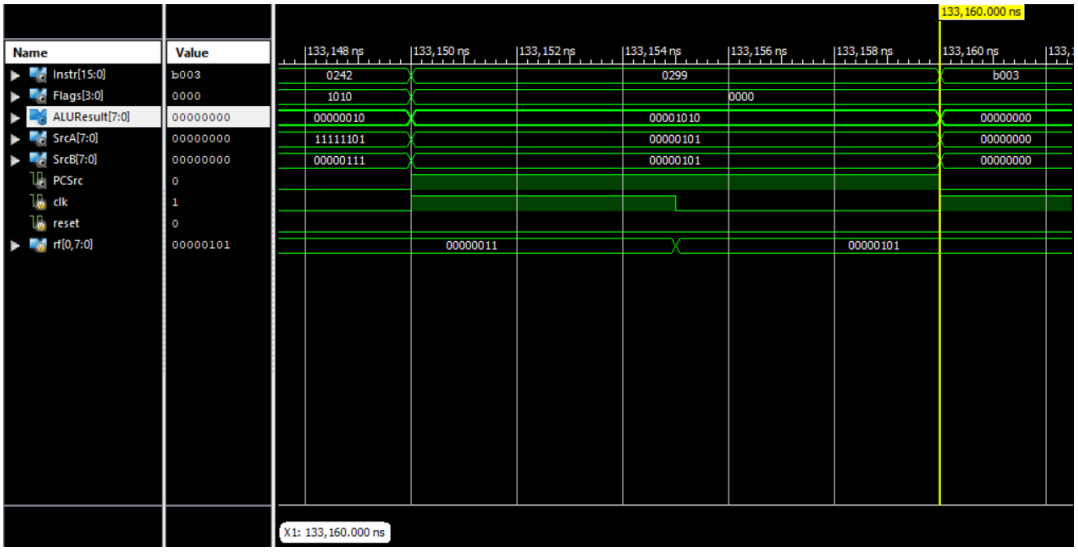
تصویر 7



تصویر1-7 - اجرا شده در پردازنده تک چرخه ای

دستور مشخص شده , مقدار دو رجیستر را با هم عوض میکند. همانطور که در تصویر(1-7) مشخص است, در لبه بالارونده کلاک عملیات نوشتن روی رجیستر انجام میشود و در اتمام چرخه مقدار 5 بر روی رجیستر R0 نوشته میشود.

تصویر2-7 – اجرا شده در پردازنده خط لوله – فاز Fetch



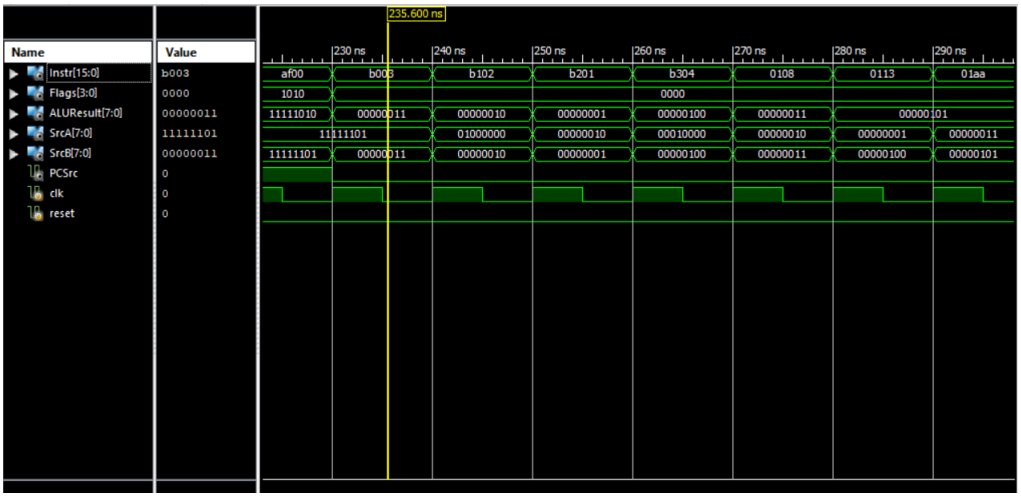
تصویر3-7 – اجرا شده در پردازنده خط لوله – فاز Writeback

در پردازنده خط لوله بین فاز خواندن دستور از فایل(Fetch) و فاز نوشتن روی regfile (Writeback) چهار فاز اختلاف است. پس برای مشاهده صحت عملکرد پردازنده در این دستور به چهار فاز بعد می رویم. همانطور که در تصویر (3-7) مشخص است, به درستی مقدار 3 درون رجیستر R0 قرار گرفته است. این صحت عملکرد نشان دهنده درستی کارکرد همه فازها در این مورد میباشد.

```
JMP to 0
```

تصویر 8

این دستور, Pc را مساوی صفر قرار میدهد به طوری که دستور اول حافظه خوانده میشود و اجرا میشود.



تصویر1-8 – اجرا شده در پردازنده تک چرخه ای

همانطور که در شبیه سازی مشخص است بعد از تمام شدن دستور JMP , دستور اول فایل اجرا میشود.

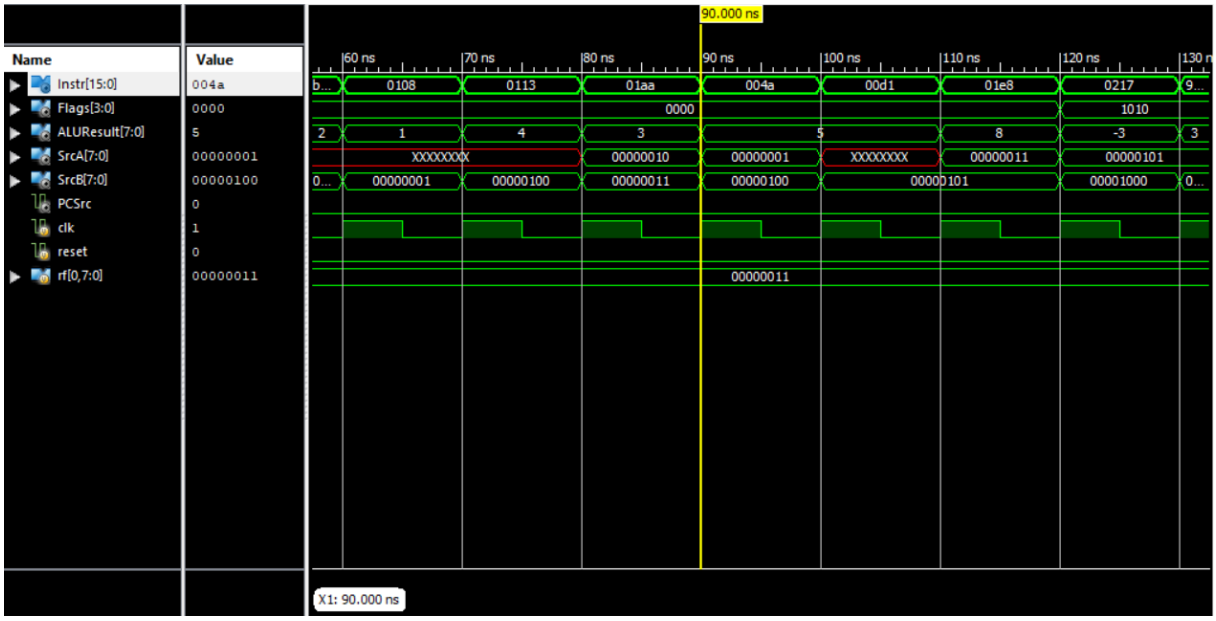
تصویر2-8 – اجرا شده در پردازنده خط لوله

دستورات از نوع Branch, حتما باید تمام 5 فاز را طی کنند تا مشخص شود که چه دستوری باید اجرا شوند و بعد از اینکه مشخص شد که Branch درست است, باید تمامی دستورات قبلی رها شوند (Flush). همانطور که در تصویر بالا مشاهده میشود بعد از 5 فاز مشخص میشود که باید دستور pc=0 اجرا شود و Flush نیز انجام شده است. این تست صحت عملکرد سیگنال های hazardunit را به ما نشان میدهد.

همچنین با مقایسه دو تصویر بالا میتوانیم بفهمیم که یک دور اجرای کد در پردازنده تک چرخه ای230ns طول میکشد در صورتی که در پردازنده خط لوله این تنها 160ns طول میکشد. پس میتوان به سریعتر بودن پردازنده خط لوله با استفاده از این تست نیز پی برد.

```
ADD R1 R2 //R1 = 8
SUB R2 R1 //R2 = -3
```

تصویر 9



تصویر1-9 – اجرا شده در پردازنده خط لوله

همانطور که در تصویر (9) مشخص است, دستور پایین به مقدار جدید رجیستر R1 برای انجام محاسبات نیاز دارد. در این مورد, ماژول hazardunit عملیات Forwarding را مدیریت میکند به طوری که مقدار این رجیستر را در فاز Execute به دستور پایین میدهد تا بتواند در عملیات های خود از آن استفاده کند. به همین خاطر دیگر هیچ مشکلی در اجرای دستور پایین به وجود نمی آید که نیاز به توقف داشته باشد. همانظور که مشاهده میشود ALUResult مطابق با انتظارات ما میباشد. پس با این شبیه سازی به صحت عملکرد واحد hazardunit در اجرای Forwarding پی میبریم.

نتیجه گیری

در این آزمایش, با تغییر و اضافه کردن ماژول هایی به پردازنده تک چرخه ای توانستیم پردازنده خط لوله را پیاده سازی کنیم. پردازنده خط لوله توانایی اجرا چند دستور همزمان را دارد که همین امر باعث اجرای سریعتر دستورات نسبت به پردازنده تک چرخه ای میباشد. موازی اجرا کردن دستورات در این پردازنده, در مواردی سبب ایجاد hazard میشود که با اضافه کردن واحد hazardunit توانستیم این مشکل را نیز رفع کنیم به طوری که هیچ اختلال و وقفه ای در اجرای دستورات به صورت موازی پیش نیاید. در آخر با انجام تست های مشابه با پردازنده تک چرخه ای و مقایسه خروجی ها, به صحت عملکرد پردازنده در اجرای دستورات و همچنین صحت عملکرد ماژول hazardunit پی بردیم.