

دانشگاه صنعتی شریف دانشکده مهندسی کامپیوتر

عنوان:

گزارش پروژه درسی

اعضای گروه

فرزام کوهی رونقی آریا همتی مهدی اکبری ثنا بابایان ونستان

نام درس معماری کامپیوتر

نیم سال دوم ۱۴۰۲ – ۱۴۰۳ نام استاد درس حسین اسدی

۱ مقدمه

افزودن قابلیت پیش بینی taken و یا taken not بودن دستورات پرشی شرطی به پردازنده پایپ لاین میپس.

۱-۱ تعریف مسئله

در این پروژه قصد داریم به پردازنده ی MIPS که در تمرین های عملی درس طراح شده بود، branch واحد prediction branch اضافه کنیم. همان طور که در درس نیز یاد گرفته بودید taken taken ها با توجه به کدی که در قبل اجرا شده است در خط لوله بیاورند که با فرض بودن آن اجرا می شوند یا اینکه دستور هایی را پیش بینی کنند که دستورهایی را در خط لوله بیاورند که با فرض not taken بودن اجرا می شوند.

۱-۲ اهمیت موضوع

انجام این کار باعث یک پیش بینی نسبتا درست درمورد پرش های شرطی می شود که با افزایش سرعت به عملکرد پردازنده ما کمک خواهد کرد.

۲ گام های پروژه

اهداف پروزه را به صورت یک سری گام نشان می دهیم:

گام ۱: اطمینان از صحت عملکرد پردازنده: براي اینکه در ادامه گام ها به مشکل نخوریم، در ابتدا پردازنده Mips pipeline خود را مجددا می آزماییم و در صورت داشتن مشکل، آن را اصلاح میکنیم.

گام ۲: افزودن قابلیت فلاش کردن پایپ لاین: برای اینکار، مکانیزمی طراحی میکنیم که در صورتی که یک سیگنال دلخواه، ۱ شد، pipe از تعدادی دستور پاك شود. سپس آن را تست میكنیم.

گام ۳: ساختن واحد predictor branch و آپدیت کرئن مقادیر بیت مورد نظر: با توجه به توضیحات گفته شده (۴ مدخل و ...) یک واحد predictor branch طراحی کرده و کارکرد آن را تست میکنیم.

گام ۴ : سیاست :FIFO برای جایگ ذاری branch های جدید، از سیا ست FIFO استفاده می کنیم.

گام ۵: سیاست :LRU یک نمونه دیگر از predictor branch خود اینبار با سیاست LRU براي جایگذاري میسازیم.

گام ۶: نوشتن برنامه اي شامل پرش شرطي و حلقه ها براي بررسي صحت عملکرد مدار: بخش ارزيابي و گزارش کار

گام ۷: برنامه اي بنويسيم که همه شرط ها را اشتباه در نظر بگيرد: براي بررسي نقاط ضعف branch او برنامه اي بنويسيم که همه شرط ها را اشتباه در نظر بگيرد: براي بررسي نقاط ضعف predictor

گام ۸ : افزودن سیاست LRU-Pseudo برای جایگذاری و همچنین طی کردن مراحل ارزیابی قبلی (گام ۶ و ۷)

۱-۲ گام ۱

با توجه به فایل insts.mif که در پیوست قرار داده شده است یک دور از صحت عملکرد تمام دستورات مدار اطمینان حاصل شود.خروجی های درست در فایل ذکر شده قرار دارد.شکل زیر نمایی از ویو فرم این دستورات می باشد که نشان دهنده درستی خروجی ها است.

	Name	Value at 0 p		320 ₁ 0 ns	480 ₁ 0 ns	540 ₁ 0 ns 800 ₁	0 ns 960 ₁ 0 ns	1.12 us	1.28 us	1.44 us	1.6 us	1.76 us	1.92 us	2.08 us 2.	24 us
	CLOCK B	- Po	10000	пппп							пппп		пппп		
	ALU E B			1000	10000	10000	70000	10000		<u> </u>		1 1 1 1 1 1	7 1 1 1 1 1		nnn
4	> PC OUT U		0 X 1 X 2 X	3 X 4 X	5 X 6 X 7	X 8 X 9 X	10 X 11 X 12	X 13 X 14 X	15 X 16 X 17	X 18 X 19	X 25 X 26	X 27 X 28	X 29 X 63	X 64 X 65 X	66 X 67
5	> Instruc 8		000000000000000000000000000000000000000										0000000000000	X11011100000011X	101000000
	> ALU re S	_	0	X 15 X 17		0 Y 6 Y 0		10 Y 0	X 25 X		0	X 13 X	0	X 45	
,	> Operan S	0	0		X 6 X 0 X	16 X 0 X 6	X 0 X 2 X	0 X 10 X	0 X	25 X	0 X 2	25 (0	X 13 X	
5 :	> Operan S	0	0 X 15	X 17 X 0	X 10 X 0 X	10 X 0 X 10	X 0 X 10 X	0 X 10 X 0	X 25 X	0	X	12 X 63 X	0	X 32 X 63	χ
5 :	> Read S	0	0	X14X 6 X										10	
\$:	> Reg1 S	0			X 6 X 0 X	16 X 0 X 6	X 0 X 2 X	0 X 10 X	0 X	25 X	0 X :	25	0	X 13 X	
j :	> Reg2 S	0	0		X 10 X 0 X	10 X 0 X 10	X 0 X 10 X	0 X 10 X 0	X 10 X	0	X :	25 (0	X 13 X	
5	> Write S	0	0	X 6 X	10 X 0 X 16	X 0 X 6 X	0 X 2 X 0	X 10 X	0 X 25	X	0	X 13	X	0 X	45 (65
9 :	> Write U	0	0	X 2 X	3 X 0 X 2	X 0 X 2 X	0 X 2 X 0	X 2 X 0 X	2 X 0 X 3	X	0	Х 3	X	0 X	3 X 7
	None	0 ps													
	Name	Value at	2.24 us	2.4 us	2.56 us	2.72 us	2.88 us	3.04 us	3,2 us	3.36 us	3.52 us	3.68 us	3.84 us	4.0 us	4.
-	CLOCK	B 0													
	ALU E														
-		B 1													
1	> PC_OUT		65 X 66 X 6	7 X 127 X	128 X 129 X 1	30 \ 131 \ 13	12 X 133 X 134	X 142 X 143	X 144 X 145	X 146 X 147	X 148 X 14	9 X 154 X	155 X 156	X 157 X 158	159
多马马	- 10 to -		65 X 66 X 6		128 X 129 X 1				X 144 X 145				155 X 156		159
- 多 多	> Instruc	U 0 B 00000000	0011 10100000			00000(111110)(101	100 1010000000		X100100X000000	X000100 X0000	00000000		000000000000000000000000000000000000000	((000000)	
- 多多多点	> Instruc > ALU_re	U 0 B 00000000 S 0		00000000	00000\(110110\(000	0000(\(11110\)(101 \(\sum_{90}\)\(\sum_{0}\)	1100 1010000000	0000000 X110110 0	X100100X000000	X000100 X0000	31 X		000000000000000000000000000000000000000	((000000)X 31 X 0 X	-31 \(\) 3
ラ ラ ラ ラ コ	> Instruc > ALU_re > Operan	U 0 B 00000000 S 0 S 0	0011X 10100000 X 45 X	0 00000000	00000(110110)(000 X 45	00000(111110)(101 X 90 X 0 X 0 X 90	110)\ 1010000000 -101 \ 0	0000000 X110110 0	X100100X000000	0(000100 X0000 15 X 0 X	31 X 0	0	000000X0000000 X	X(000000)X 31 X 0 X X 15	-31 \(\) 3 \(\) \(\) -3
5 5 5 5	> Instruc > ALU_re > Operan > Operan	U 0 B 00000000 S 0 S 0	0011 10100000	00000000	00000\(110110\(000	0000(\(11110\)(101 \(\sum_{90}\)\(\sum_{0}\)	110)\ 1010000000 -101 \ 0	0000000 X110110 0	X100100X000000	X000100 X0000	31 X	0	X 31 X	()(000000) 31	-31 \(\) 3 \(\) \(\) -3
555555	> Instruc > ALU_re > Operan > Operan > Read	U 0 B 00000000 S 0 S 0 S 0	0011X 10100000 X 45 X	0 0 0	00000\(\)110110\(\)000 \(\)\(\)45 \(\)\(\)45	00000(\(111110\)(101 \(\sum 90\) \(0\) \(\sum 90\) \(\sum 0\) \(\sum 90\)	110)X 1010000000 (-101 X 0 0 5 X 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0000000 X110110 0 X-1	(100100)(000000 X 101 X 101 X 15 X	0(000100 X0000 15 X 0 X	00000000000000000000000000000000000000	0	000000X0000000 X	X(000000)X 31 X 0 X X 15 0 X 15	X -3 X -3
5 5 5 5 5 5	> Instruc > ALU_re > Operan > Operan	U 0 B 00000000 S 0 S 0	0011X 10100000 X 45 X	0 00000000	X 45 X 45	00000(\(\frac{11110}{11110}\(\frac{101}{101}\) \(\frac{90}{0}\) \(\frac{0}{0}\) \(\frac{9}{0}\) \(\frac{0}{0}\) \(\frac{0}\)	110)\(\) 1010000000\(\) 101\(\) \(\) 0 5 \(\) 0	0000000 X110110 0 X -: 0 X -:	X100100X000000 X101 X 101 X 15 X	0000100 X0000 15 X 0 X 0 X 31 X	31 X 0	0	X 31 X	0(000000) 31	-31 X 3 X -3 X -2
5 5 5 5 5	> Instruc > ALU_re > Operan > Operan > Read	U 0 B 00000000 S 0 S 0 S 0	0011X 10100000 X 45 X	0 0 0	00000\(\)110110\(\)000 \(\)\(\)45 \(\)\(\)45	00000(\(111110\)(101 \(\sum 90\) \(0\) \(\sum 90\) \(\sum 0\) \(\sum 90\)	110)\(\) 1010000000\(\) 101\(\) \(\) 0 5 \(\) 0	0000000 X110110 0 X -: 0 X -:	(100100)(000000 X 101 X 101 X 15 X	0(000100 X0000 15 X 0 X	00000000000000000000000000000000000000	0	X 31 X	X(000000)X 31 X 0 X X 15 0 X 15	-31 X 3 X -3 X -2
5 5 5 5 5 5 5 5	> Instruc > ALU_re > Operan > Operan > Read > Reg1	U 0 B 00000000 S 0 S 0 S 0 S 0 S 0 S 0	0011X 10100000 X 45 X	0 0 0 0	X 45 X 45	00000(\(\frac{11110}{11110}\(\frac{101}{101}\) \(\frac{90}{0}\) \(\frac{0}{0}\) \(\frac{9}{0}\) \(\frac{0}{0}\) \(\frac{0}\)	1110)\ 1010000000 (-101 \ 0 5 \ 0 5 \ 0	0000000 X110110 0 X -: 0 X -:	X100100X000000 X101 X 101 X 15 X	0000100 X0000 15 X 0 X 0 X 31 X	00000000000000000000000000000000000000	0	X 31 X	0(000000) 31	-31 \(\) 3 \(\) \(\) -3

شكل ۱: كد تست در فولدر ۱Gam قرار دارد.

۲-۲ گام ۲

قبل از افزودن قابلیت فلاش کردن ابتدا به بررسی مشکل هایی که در صورت سه بار استفاده نکردن از operation no به صورت نرم افزاری بعد از دستورات پرش شرطی به وجود می آیند می پردازیم.

۲-۲-۲ باگ های دیتا

دستورات مربوط به بررسی این مشکل فایل instsDataBug.mif در پیوست قرار داده شده است. توجه تان را به خطوط ۱۳۲ و ۱۳۳ جلب می کنیم. این سه دستور بعد از یک دستور BNE قرار گرفته اند و این برنچ نیز taken است ولی چون از operation no بعد BNE استفاده نشده داده های نادرستی در مموری و رجیستر فایل مورد ویرایش قرار گرفته شده اند که این وضعیت برای ما مطلوب نمی باشد. نمای زیر نشان دهنده این مشکلاتات است.

Name	Value at 0 ps	us	2.56 us		2.64	us	2.	72 us	2.	B us	2.8	38 us	2.5	96 us	3.0	4 us	3.	12 us	3.	2 us	3.2	8 us	3,3	6 us	3.4	4 us	3.5	52 us	3.6	us	3.
CLOCK	8 0		П	П	ШП	шт	ш	ш	ш	ш	ш	ш	ш	ш	ш	Ш	Ш	ш	ш	ш		ШТ	Ш	ш	ш	ш	ш	ш	шП		
ALU_ENABLE	81																								1111					1111	
> PC_OUT	UO	128	X	129		130		131	X	132		133	\rightarrow	134		142		143		144	X	145		146		147	X	148		149	\mathcal{X}
> Instruction	в одоородоород	100000000000	000011	011011	00X	1010	0000000	000000	X00	0101110	0101 X010	0011011	10111 (01	10100100	111X01	00000000	0000 X 01	10110110	0001 010	001001000	111 X010	0000000	0000 (110	0000100	1111 00	00000000	0011X		0000	000000000	000
> ALU_result	S 0						90				0				73	X	-30		0		71		-15		0		31	X			ш
> Operand1	5 0			X	45				0			$\Box x$	90			0		$\perp \times$	73	X	-30									0	
> Operand2	S 0			X	45	X		0		$\perp \chi$	65	$\pm x$	-17		30	X	0		2	$\perp \chi$	15		0		31					0	
> Read_Data	S 0																														
> Reg1	S 0			X	45	X			0			\Box X	90			0			73	$\perp \chi$	-30	X								0	
> Reg2	S 0			X	45	X		0		X	65	-x	90	X		0		-x	73	X	-30	X	0	X	-15	X					
> Write_Data	S 0	0					$\perp x$	90	X			0				73	X	-30		0	X	71		-15		0	X	31	X		
> Write_Register	U 0		0				X	3	X		0		X	7	X	3	X	2	Υ	0	X	3	X	2	X	0	X	2	X		-

شكل ٢: باگ مربوط به ديتا

۲-۲-۲ باگ های برنچ

دستورات مربوط به این مشکلات در فایل instsBranchBug.mif در پیوست قرار داده شده موست. در خطوط ۱۳۲ و ۱۳۳ و ۱۳۳ یک دستور jump و یک دستور BEG و یک دستور ۱۳۲ و ۱۳۲ داریم که در صورت اجرا شدن اینها بعد از دستور BNE خط ۱۳۱ باعث بروز مشکلاتی خواهد شد و ما به صورت ناخواسته به آدرس هاس اشتباهی پرش خواهیم کرد. شکل زیر نمایی از ویو فرمی این دستورات است.

۲-۲-۳ افزودن سیگنال Flush و Flash Unit

برای از بین بردن تاثیر سه دستور بعد پرش شرطی ما نیاز به این داریم که در مراحل ،Write- MEM برای از بین بردن تاثیر سه دستور بعد پرش شرطی ما نیاز به این دو Back با استفاده از دو سیگنال ،flushjump flush از وقوع نتایج نامطلوب جلوگیری کنیم.این دو

N	arne Value at 0 ps	2.4 us	2.48 us	2.56	us	2.64 us	2.72	t us	2.8 us		2.88 us	2.5	6 us	3.0	4 us	3.12	us	3.2 _, u	5	3.2	B us	3.	36 us	3.44 (is	3.52 us
S. CLOCK	B 0			шп		ПП	$\Box\Box$			ПП	ш	ш	ш	ш	шП				\Box	ш	ш	ш	ш	шп		
S ALU_E	IABLE 8 1								11111																	
3 > PC_OU	r uo	127	128	\rightarrow	129	130		131		32	133		134	\Box X	142	\supset	20	\propto	148	\Box X	149	$\perp \times$	150	\perp	151	1:
25 > Instruc	ion B 000000000000000		X000000000	00000 00000	1101101100	X 10	10000000000	0000	(001010	11100101	110000000	01010	00000000	0111 01	000000000	000 0110	11011000	11)						000000	0000000	000
⇒ ALU_re	sult S 0		0			X	90	X					0						X	88	\perp X					
🥞 > Operar	d1 S 0				X	45 X					0						X	90	X							
🥞 > Operar	d2 S 0	0			X	45 X		0		X 65	\perp X	20	$\perp \chi$		0		X	2	X							0
> Read_0	lata S 0																									
⊜ > Reg1	S 0				X	45 X					0						X	90	X							
25 > Reg2	S 0				X	45 X		0		X 65	-x			0			X	90	X							
25 > Write_	Data S 0			0			X	90	X					0						X	88	X				
35 > Write_	Register U 0			0			X	3	X	0		X	7	X			0			X	3	-x				

شکل ۳: باگ مربوط به برتچ

سیگنال باید ۳ ملالک طول بکشد.برای مشاهده مکانیزم عملکرد این دو سیگنال می توان به مدار مراجعه کرد.تصاویر زیر گویای این امر هستند که بعد از به کارگیری این قابلیت دو مشکل مطرح شده در قسمت قبل را به کلی حل می کند.

۲-۳ گام ۳

۲-۲ طراحی اولیه ۴-۲

predictor branch ساخته شده حاوی ۴ block است به طوری که هر predictor branch بیت سمت راست مربوط به آدرس هست، بیت نهم بیت valid/invalid میباشد و به طور پیشفرض در ابتدا مقدار آن صفر است و بعد از آن با آمدن هر دستور بیت valid میشود. بیت بعدی مربوط به prediction هست که در صورتی که صفر باشد taken not و در صورتی که یک باشد prediction است. ۶ بیت بعدی که ۶ بیت آخر هست age میباشد که هیچ استفادهای از آن نمیکنیم (بعد از پیاده سازی های اولیه استراتژی پیاده سازی تغییر کرد و به علت اینکه هر که که بیت نمی شود.) توانی از ۲ بماند ۶ بیت را حذف نکریم ولی در واققع هیچ استفادهای از این ۶ بیت نمی شود.)

۵-۲ واحد FIFO branch predictor

پیاده سازی predictor branch fifo به این صورت است که هر وقت به دستور branch میرسیم آذرس آن را به عنوان ورودی به predictor branch ساخته شده می فرستیم. ورودی داده شده با ۴ آدرس آن را به عنوان ورودی به pridector branch ساخته شده سازی این بخش مشابه با پیاده سازی مقدار که در pridector branch هست چک می شود (پیاده سازی این بخش مشابه با پیاده سازی دعمه cache می باشد که به صورت fully عمل می کند) اگر ادرس مورد نظر بین آن ۴ مقدار بود سیگنال ۱ شود به این معنی بود سیگنال ۱ شود به این معنی branch در بین آن ۴ مقدار قرار دارد و در این صورت مقدار بیت branch که ادرس branch در بین آن ۴ مقدار قرار دارد و در این صورت مقدار بیت از ۴ مقدار بین آن ۴ مقدار و در این صورت مقدار بیت معنی

دهمین بیت از سمت راست هست را خروجی می دهد به این معنی که ،taken branch است. برای پیاده سازی FIFO اگر سیگنال branch is یک باشد ولی سیگنال hit صفر بود (به معنی وجود نداشتن در (predictor همه آن ۴ مقدار را به پایین یکی shift می دهیم و آدرس branch که taken نشد را در سطر اول predictor می نویسیم و بیت prediction آنرا هم یک می کنیم که taken باشد.

۲-۶ بررسی حالت های مختلف

taken predict ۱-۶-۲ شود و taken باشد

در این حالت همیشه هنگام operation no یک penalty miss خواهیم داشت.

taken not شود ولي taken predict ۲-۶-۲

در این حالت خط بعدی اجرا می شود بعد از آن به آدرس مورد نظر پرش می کند و در آنجا هم ۲ دستور اجرا می کند (مجموعا ۳ دستور اجرا می شود) و بعد از آن متوجه می شود که این ۳ دستور به اشتباه اجرا شده است و آنها را flush می کند و به آدرس بعد از آدرس branch برمیگردد و از آنجا اجرای دستورات را ادامه می دهد.

taken not predict ۳-۶-۲ شود ولی taken باشد

در این حالت ۳ دستور بعدی اجرا می شود و بعد از آن نتیجه شرط مشخص می گردد. با توجه به نتیجه بدست آمده باید ۳ دستوری که اجرا شدند را متوقف کنیم و عملیات هایی که در آن انجام شده است را reverse کنیم که flush پیاده سازی شده در پروژه این کار را انجام می دهد و در نهایت به آدرس درست می رود.

taken not predict ۴-۶-۲ شود و taken not باشد

۷-۲ واحد LRU branch predictor

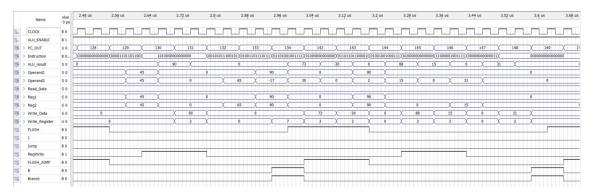
پیاده سازی این قسمت تقریبا با پیاده سازی predictor branch FIFO یکی است اما یکسری تفاوتهایی دارد که اصلی ترین قسمت آن در الگوریتمی است که بعد از hit شدن در اجرا می شود. الگوریتم به این صورت پیاده سازی شده که هر زمانی که آدرسی hit شد به معنی اینکه در یکی از ۴ block پیدا شده، بلاکی که hit شده هرجایی که باشد به سطر اول predictor انتقال داده می شود و اگر آن بلاک در سطر k ام باشد، - ۱۸ سطر بالای آن همگی یکی به سمت پایین شیفت می خورند. بوسیله این پیاده سازی هنگامی که یک miss branch شود پایینترین سطر از بین می رود و همه سطر ها یکی به سمت پایین شیفت می خورند و آدرس جدید در سطر اول نوشته می شود. با پیاده سازی که در قسمت hit انجام داده ایم همیشه سطری که حذف می شود و در سطر آخر قرار دارد کمترین استفاده را داشته است و به این ترتیب الگوریتم LRU را پیاده سازی کردیم. FIFO و FSM موجود در صورت پروژه برای الگوریتم های URU و و FIFO کردن predictor هم بر اساس FSM موجود در صورت پروژه برای الگوریتم های LRU و مشترک پیاده سازی شده به شکلی که اگر اشتباها taken not یا تنیجه قبل اعلام بشود یعنی اگر درس رو به شکلی علود این سری taken not اعلام میشود و برعکس آن هم به همین شکل. این تغییرات بوسیله بیت taken prediction اعمال می شود.

۸-۲ گام ششم

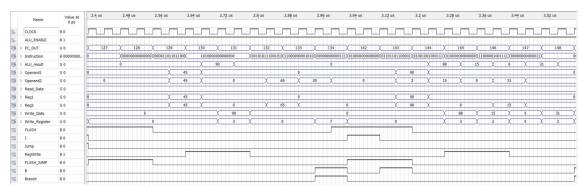
در این گام با استفاده از یک کد تست که در زمیمه قرار دارد، به مقایسه FIFO و LRU پرداخته ایم. مشاهده می شود زمان اجرا برای pipeline که بدون predictor هست، بیشتر از زمانی است که FIFO دارد. در این مثال که مشاهده می کنید زمان LRU بهتر از زمان predictor pipline، می شود.

۲-۹ گام هفتم

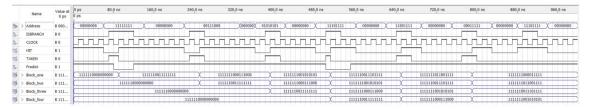
با ارائه یک کد که در آن تعدادی branch دائما اشتباه پیشبینی می شوند متوجه می شویم که اجرای برنامه با استفاده از pipeline بدون predictor بهینه تر بوده و در این مثال مجداد LRU نتیجه بهتری از FIFO ارائه می دهد.



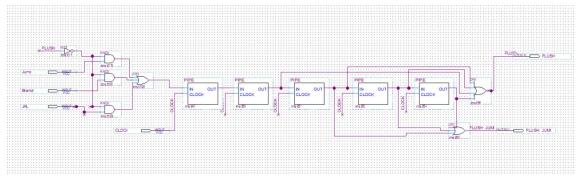
شکل ۴: رفع کردن مشکل باگ برای دیتا



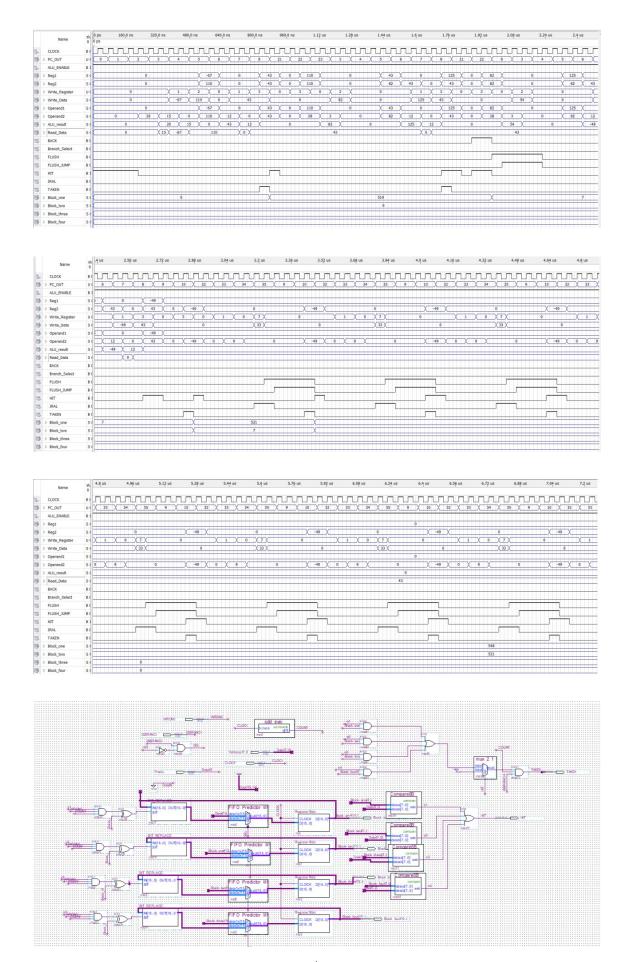
شکل ۵: رفع کردن مشکل باگ برای برنچ

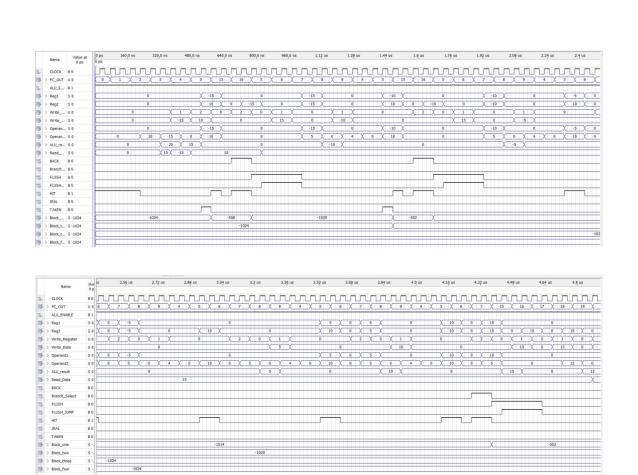


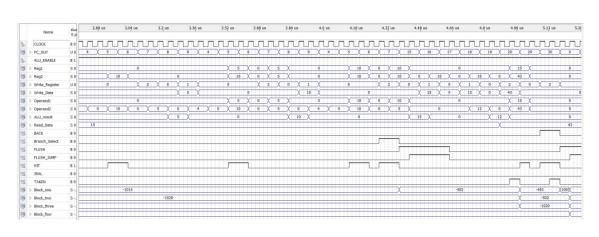
شکل ۶: اجرای برنامه با predictor FIFO

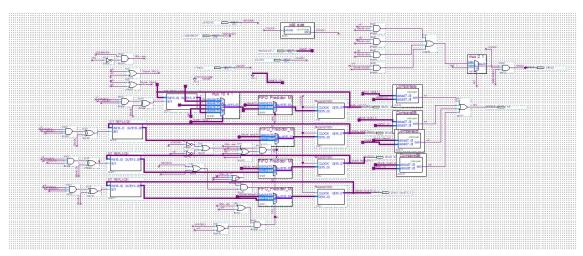


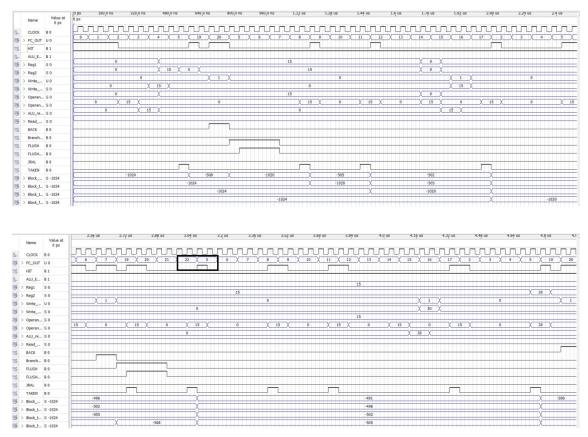
شکل ۷: Unit Flush



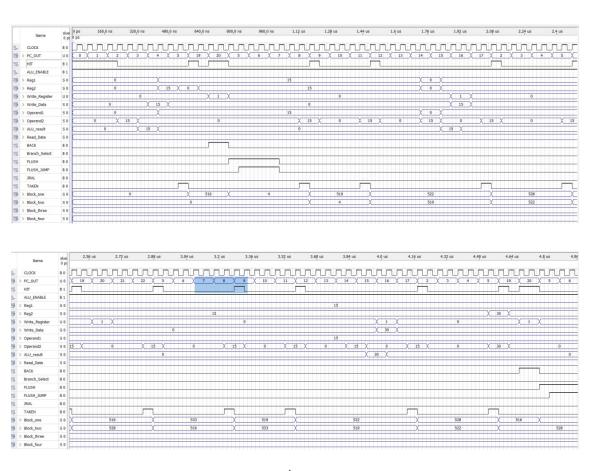


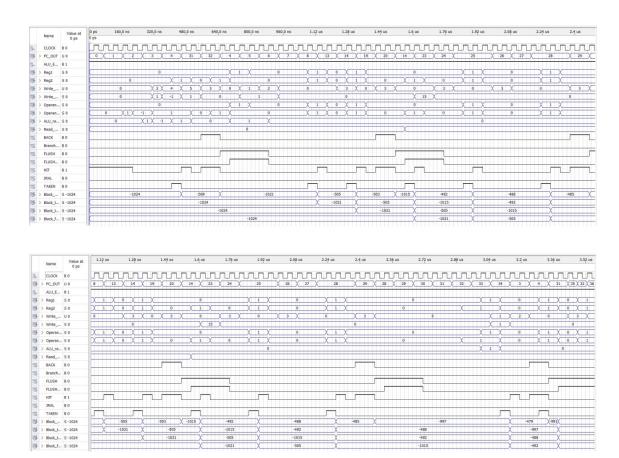




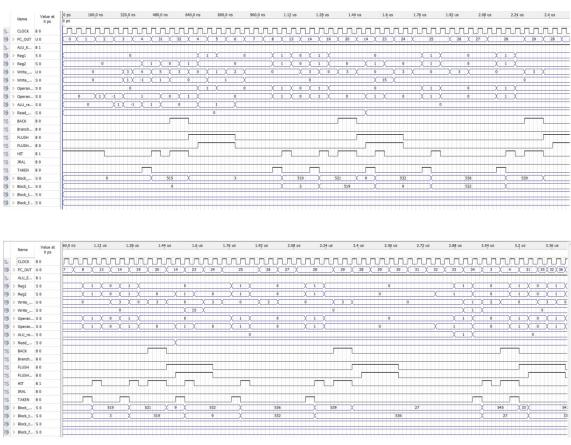


شکل ۸: دو عکس اول مربوط به FIFO و دو عکس دوم مربوط به LRU می باشد





شکل ۱۰: دو عکس اول مربوط به FIFO و دو عکس دوم مربوط به LRU می باشد



References

مطالب تكميلي

پیوستهای خود را در صورت وجود میتوانید در این قسمت قرار دهید.