به نام خدا

تمرین هفتم معماری کامپیوتر دکتر اسدی

اعضای تیم:

- فرزام کوهی رونقی 401106403
 - آريا همتي 401110523
 - ثنا بابایان 401105689

گزارش کار:

در این تمرین می خواهیم به پردازنده ای که در دو تمرین قبل ساخته ایم، خط لوله را اضافه کنیم که یعنی 5 مرحله خواهیم داشت و برای هر مرحله یک بلوک کنترلی داریم که سیگنال های مورد نیاز برای هر مرحله را تولید میکند.

در واحد کنترلی اول ما سیگنال zero_extend را می گیریم که شامل Instruction fetch می شود.

سیگنال های ALU_Op, ALU_Src را میخواهیم از واحد کنترلی دوم بگیریم که شامل ALU_Op, ALU_Src را میخواهیم از واحد کنترلی دوم بگیریم

سیگنال MemWrite را میخواهیم از واحد کنترلی سوم بگیریم که شامل Execute/address calculation می شود. (به دلیل تعدادی data hazard آن را به واحد کنترلی چهارم انتقال دادیم و واحد کنترلی سوم عملا سیگنالی خروجی نمی دهد)

سيگنال MemRead و JAL (و MemWrite) را ميخواهيم از واحد كنترلي چهارم بگيريم كه شامل JAL (و memory access/JAL مي شود.

سیگنال های Jump, RegDst, Branch, MemtoReg, RegWrite را میخواهیم از واحد کنترلی پنجم بگیریم که شامل write سیگنال های back/Jump/Branch می شود.

یایپ اول مقدار PC + 1 را لچ می کند.

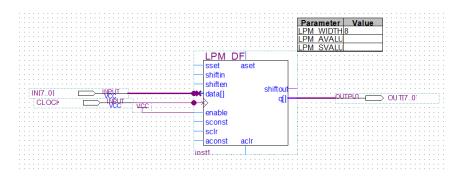
پایپ دوم مقدار , PC, instruction, immediate را لچ می کند.

پایپ سوم مقدار Reg1, Reg2, Instruction, Branch_dest, PC, Zero, ALU_Result , را لچ می کند.

پایپ چهارم مقدار Reg1, PC, Instruction, Branch_des, ZERO, ALU_result, Reg2, Address را لچ میکند.

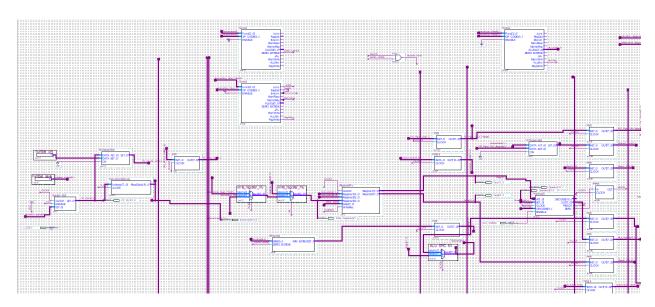
بلوکهای جدیدی که به مدار اضافه شده اند:

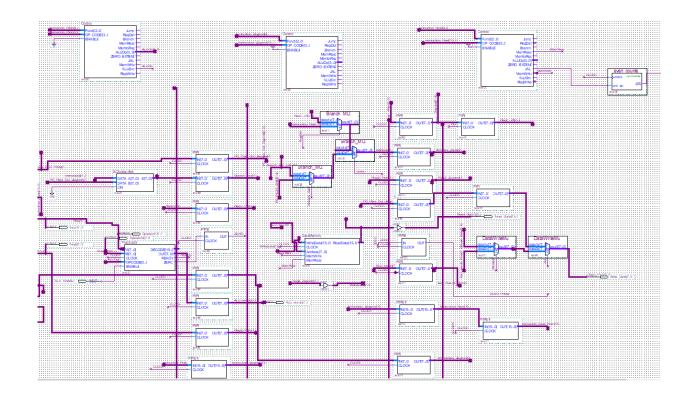
1- تعدادی بلوک به نام Pipe: این Pipe ها که 3 مدل PIPE1, PIPE, PIPE16 دارند، یک Lpm_dff هستند که تعداد بیتهای آن به ترتیب 1, 8, 16 است. تصویر PIPE در صفحه بعد موجود است.



2 – افزایش تعداد واحد کنترلی به 5 عدد: این واحد کنترلی همان ساختار واحد کنترلی قبل را دارد صرفا تعدادش افزایش یافته.

شکل کلی مدار MIPS_CPU که پایپ لاین است مانند زیر است: (تصویر را تکه تکه گذاشتم زیرا جا نمیشد!)





محتوای فایل دستورات آزمایش شده:

DEPTH = 256;

WIDTH = 16;

ADDRESS_RADIX = DEC;

DATA_RADIX = BIN;

CONTENT BEGIN

0:101000000000000; -- no-op

1:0111001010001111; -- LB r2, 15(r1) --> r2 = 6

2:0111000011010001; -- LB r3, 17(r0) --> r3 = 10

3:101000000000000; -- no-op

4 : 0000010011010000; -- ADD r2, r2, r3 --> r2 = 16

5:10100000000000; -- no-op

```
6:0000010011010001; -- SUB r2, r2, r3 --> r2 = 6
```

7:10100000000000; -- no-op

8:0000010011010010; -- AND r2, r2, r3 --> r2 = 2

9:10100000000000; -- no-op

10:0000010011010011; -- OR r2, r2, r3 --> r2 = 0010 | 1010 = 1010 = 10

11:101000000000000; -- no-op

12:0000010011010101; -- XOR r2, r2, r3 --> r2 = 0

13: 10100000000000; -- no-op

14:0010010011011001; -- ADDi r3, r2, 25 --> r3 = 25

15: 10100000000000; -- no-op

16:0000011000000111; -- JR r3 --> Go to 25

17: 101000000000000; -- no-op

18:10100000000000; -- no-op

19:10100000000000; -- no-op

25:0011011011001100; -- SUBi r3, r3, 12 --> r3 = 13

26:1110000000111111; -- J 63

27: 101000000000000; -- no-op

28:10100000000000; -- no-op

29:101000000000000; -- no-op

63:0101011011100000; -- ORi r3, r3, 1111111b --> r3 = 00100000 | 00001101 = 00101101 = 45

64: 11110000011111111; -- JAL 127 --> go to 127, r7 = 65

65:10100000000000; -- no-op

66: 10100000000000; -- no-op

67: 101000000000000; -- no-op

128:0000011011011000; -- ADD r3, r3, r3 --> r3 = 90

129: 101000000000000; -- no-op

130:0000011111011000; -- ADD r3, r3, r7 --> r3 = 155

131 : 10010101111001010; -- BNE r2, r7, 10 --> go to 142

132:101000000000000; -- no-op

133:101000000000000; -- no-op

134: 101000000000000; -- no-op

142:0000011011011001; -- SUB r3, r3, r3 --> r3 = 0

143:0010010010001111; -- ADDi r2, r2, 15 --> r2 = 15

144: 10100000000000; -- no-op

145:0110000010011111; -- SB r2, 31(r0) --> Mem[31] = 15

146: 100000000000111; -- BEQ r0, r0, 7 --> Go to 154

154: 0111000000011111; -- LB r0, 31(r0) --> r0 = 15

155: 10100000000000; -- no-op

156: 00000000000000100; -- MULT r0, r0, r0 --> r0 = 225

END;

همانگونه که مشاهده می شود به منظور جلوگیری از Data Hazard، میان برخی دستورات از no-operation استفاده شده که دستوری است که وجود ندارد (صرفا برای جلوگیری از Data Hazard است و کار خاصی انجام نمی دهد)

این فایل دستورات را در مدار Pipeline خود اجرا می کنیم:

		Value at	0 ps 160.0 ns	320,0 ns	480,0 ns 640,0	ns 800,0 ns	960,0 ns	1.12 us	1.28 us 1.44 us	1.6 us	1.76 us	1.92 us 2.0	08 us 2.24 us 2
	Name	0 ps	0 ps										
in_	CLOCK	В 0											
<u>is</u> _	ALU_E	B 1											
35	> PC_OUT		0 1 2	3 X 4 X 5	6 7 7	8 9 10	11 12	13 14	15 (16) 17 (18	19 (25)	26 27 28	29 (63)	64 (65) 66 (67)
***	> Instruc	В 00000000	(00000000)(00)(010100)(0	000110 0000000 1001	110 0000000 100110 00	0000(100110)(000000	(100110:\(0000000\)	00110 (0000000)1	00110:\(000000)\(110000)\(10000)\(10000)\(100000)\(100000)\(100000)\(100000)\(100000)\(100000)\(100000)\(100000)\(100000)\(100000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(1000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(10000000)\(100000000)\(100000000)\(100000000)\(100000000)\(100000000)\(100000000)\(100000000)\(100000000)\(100000000)\(1000000000)\(100000000)\(100000000)\(1000000000)\(1000000000)\(1000000000)\(1000000000)\(100000000)\(1000000000)\(10000000000	100000000000000 (11	101100000001 1	0100000000000000	110111100000111 101000000000
945	> ALU_re	S 0	0	15 17	0 16 0	(6 X 0 X	2 0 10	χ •	25 🗶	0	13	0	X 45 X
eut	> Operan	S 0	0	X	6 (0) 16	0 (6)	0	10	0 25 X	0	25	0	X 13 X 0
25	> Operan	S 0	0 15	X 17 X 0	10 0 10	0 10	0	X 10 X 0	X 25 X	0	12 (63)	0	X 32 X 63 X
25	> Read	S 0	0	X14 X 6									10
eut	> Reg1	S 0	0	X	6 0 16	0 6	0	(10)	0	0	25	0	X 13 X 0
25	> Reg2	S 0	0	X	10 0 10	0 10	0 X 10 X 0	X 10 X 0	X 10 X	0	X 25 X	0	X 13 X 0
94	> Write	S 0	0	X 6 X 10	0 (16 (0 X 6 X 0	X 2 X 0 X	10	0	0	X 13	X 0	X 45 X 65 X
<u>eut</u>	> Write	U 0	0	X 2 X 3	0 (2 (0	X 2 X 0 X	2 X 0 X	2 X 0 X 3 X	0	X 3	X 0	3 (7 (1)

	Nar	ime	Value at 0 ps	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 4.16 u
in	CLO	OCK E	во												
<u>is</u>	ALI	.U_E E	B 1												
**	> PC_	_OUT L	U 0	65 (66)	67 127 128	X 129 X 1	30 \(131 \) 1	32 133	134 🗶 142 🗶 1	43 (144)	145 🗶 146 🗶	147 🗶 148 🗶 14	9 154 1	155 156 15	7 158 159 16
**	> Ins	struc E	В 00000000	0011 1010000	000000000 (00000	0)(110110)(000	0000(111110)(10	11100 1010000	0000000000 (110	0110:\(100100\)	000000000000000000000000000000000000000	000000000	0000000 00	0000:\(0000000)\(0000	0000
**	> ALI	.U_re S	S 0	X 45 X	0		90 🚶 0	-101	0		15 0	31	0	31	0 31 30
**	> Op	eran S	S 0	X	0	X 45	X 0 X 90	X	0	X -101		0			15 31
545	> Op	eran S	S 0	X 63 X	0	X 45	X 0 X	65	0	X -101 15	0 31	χ ο		31 0	15 / -31
*	> Rea	ad 9	S 0											X14 X	
**	> Reg	g1 S	S 0	X	0	X 45	0 90	X	0	-101		0			15 (-31
545	> Reg	g2 S	S 0	X	0	45	X 0 X	65	0	-101	0 15	X	0		15 31
84	> Wr	rite S	S 0	45 (65	X 0		90	0 \ -101 \		0	15	0 (31)	0	X 1	5 (0 (-31 (3)
**	> Wr	rite l	U 0	3 (7	X 1 X	0	X 3 X	0 (3)	7 X	0 X	3 (2)	0 X 2 X			

با مشاهده PC_OUT به وضوح می توان خط دستور اجرا شده را مشاهده کرد و سیر مسیر Branch, JAL, Jump ها واضح است. از جهتی مقادیر حاصل شده از دستورات را می توان در Reg1 و Reg2 مشاهده کرد. دقت شود که مقداری که دور آن مستطیل مقادیر حاصل شده از دستورات را می توان در 251 است اما به دلیل overflow، برابر 31- شده است، همچنین مستطیل دیگر نیز 65 + 90 بوده که 155 می شود اما به دلیل overflow، برابر 101- شده. برای دیدن مقدار واقعی آنها، radix را به unsigned تغییر داده و مجددا عکس می گیرم:

	Name	Value at 0 ps	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 4.16 u
<u>_</u>	CLOCK	В 0												
S- /	ALU_E	B 1												
∰ > F	PC_OUT	U 0	65 (66) 6	7 127 12	28 129 13	0 131 132	133 1	34 \ 142 \ 14	3 144 1	45 146 1	47 (148) 14	9 154 15	5 156 15	7 158 159 16
≫ > 1	Instruc	В 00000000	0011 10100000	00000000 0000	000)(110110)(0000	000(111110)(1011	10100000	000000000 (110:	110 (100100)(000	000000001000000	000000000	0000000 (0000	000000000000000000000000000000000000000	00)(
勞 > /	ALU_re	U 0	45	0		90 🗶 0 🗶	155	0		15 0	31	0	31	0 225 30
⇒ > 0	Operan	U 0	X	0	X 45 X	0 X 90 X		0 X	155		0			15 X 225
⇒ > 0	Operan	U 0	63	0	X 45 X	0 (65	X	0 X	155 15	0 31	X 0	X	31 (0	15 (225
∰ > F	Read	S 0											(14)	
⇒ > F	Reg1	S 0	X	0	X 45 X	0 X 90 X		0 X	-101		0		X	15 (-31)
∰ > F	Reg2	S 0	X	0	X 45	0 (65	X	0 X	-101	0 (15	X	0	X	15 (-31)
∰ > \	Write	S 0	45 (65)		0	90 0	-101			15	0 31	0	15	0 31 31
⇒ > \	Write	U 0	3 (7)	(1X	0	X 3 X 0	X 3 X	7 X 0	X	3 X 2 X	0 X 2 X			

حال، همین دستورات را بدون no-operation و صرفا پشت هم به مدار مولتی سایکل میدهم و خروجی waveform آن را نیز قرار میدهم:

DEPTH = 256;

WIDTH = 16;

ADDRESS RADIX = DEC;

DATA_RADIX = BIN;

CONTENT BEGIN

0:10100000000000; -- no-op

1:0111001010001111; -- LB r2, 15(r1) --> r2 = 6

```
2:0111000011010001; -- LB r3, 17(r0) --> r3 = 10
```

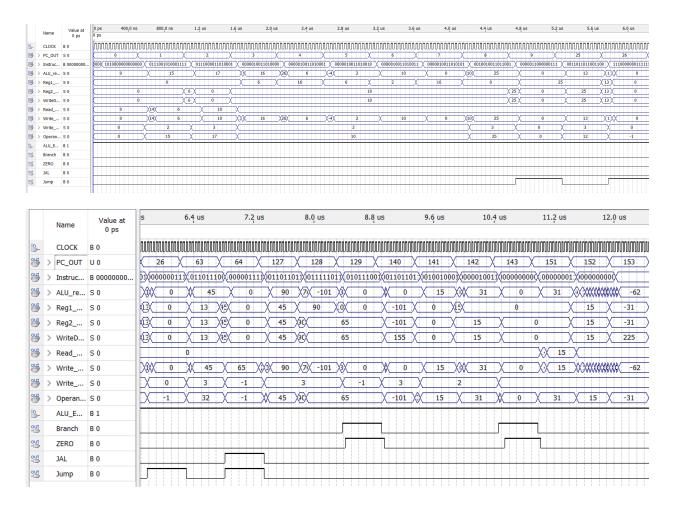
5:
$$0000010011010010$$
; -- AND r2, r2, r3 --> r2 = 2

$$63:0101011011100000$$
; -- ORi r3, r3, $1111111b$ --> r3 = $00100000 \mid 00001101 = 00101101 = 45$

$$142:01110000000111111;$$
 -- LB r0, $31(r0)$ --> $r0 = 15$

END;

این دستورات مشابه برنامه قبلی است صرفا no-op ها حذف شدند. حال waveform این دستورات را با CPU مولتی سایکل قرار میدهم:



نتایج: همانطور که مشاهده می شود، تعداد کلاکها در این حالت بسیار بسیار بیشتر از تعداد کلاکها در همین برنامه که در Pipeline اجرا شد، است. از آنجا که تعداد کلاکها در حالت Pipeline کمتر است و CLOCK time در هر دو یکسان، یعنی برنامه در pipeline سریعتر اجرا شده است. احتمالا در پیاده سازی واقعی و عملی مانند این شبیه سازی، سرعت اجرا در حالت pipeline بیشتر خواهد بود.