به نام خدا

تمرین هفتم معماری کامپیوتر دکتر اسدی

اعضای تیم :

- فرزام کوهی رونقی 401106403
- آریا همتی 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 Instruction decode/register file read می شود.

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 نمی‌دهد)

سیگنال MemRead و JAL (و MemWrite) را میخواهیم از واحد کنترلی چهارم بگیریم که شامل memory access/JAL می شود.

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

پایپ اول مقدار PC + 1 را لچ می کند.

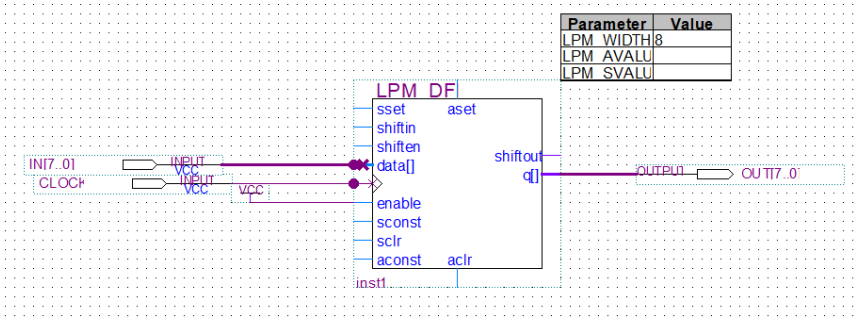
پایپ دوم مقدار PC, instruction, immediate, را لچ می کند.

پایپ سوم مقدارReg1, 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.

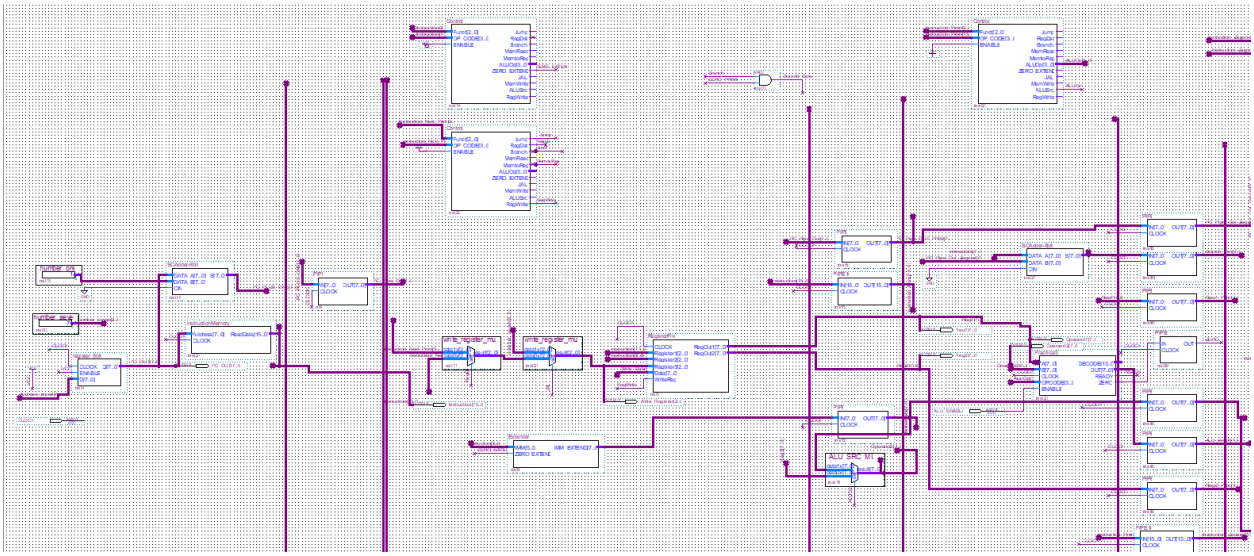
بلوک‌های جدیدی که به مدار اضافه شده اند:

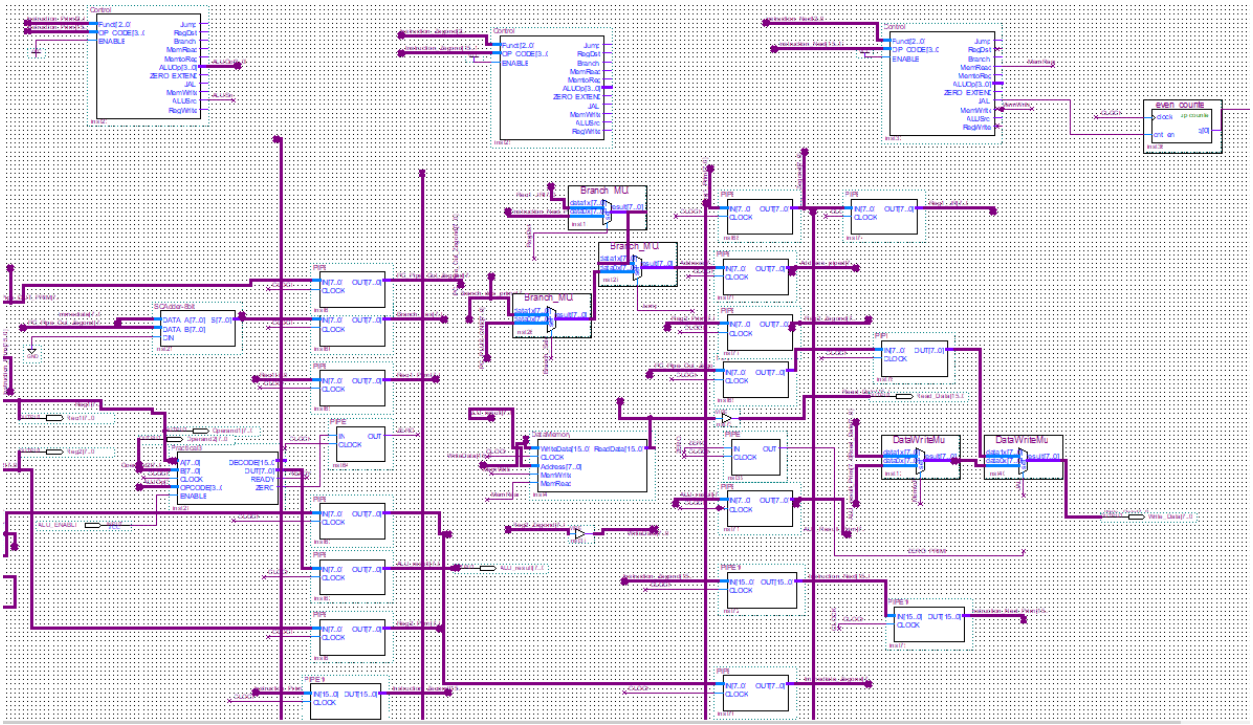
1- تعدادی بلوک به نام Pipe: این Pipe ها که 3 مدل PIPE1, PIPE, PIPE16 دارند، یک Lpm_dff هستند که تعداد بیت‌های آن به ترتیب 1, 8, 16 است. تصویر PIPE در صفحه بعد موجود است.



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 : 1010000000000000; -- no-op

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

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

3 : 1010000000000000; -- no-op

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

5 : 1010000000000000; -- no-op

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

7 : 1010000000000000; -- no-op

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

9 : 1010000000000000; -- no-op

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

11 : 1010000000000000; -- no-op

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

13 : 1010000000000000; -- no-op

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

15 : 1010000000000000; -- no-op

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

17 : 1010000000000000; -- no-op

18 : 1010000000000000; -- no-op

19 : 1010000000000000; -- no-op

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

26 : 1110000000111111; -- J 63

27 : 1010000000000000; -- no-op

28 : 1010000000000000; -- no-op

29 : 1010000000000000; -- no-op

63 : 0101011011100000; -- ORi r3, r3, 111111b --> r3 = 00100000 | 00001101 = 00101101 = 45

64 : 1111000001111111; -- JAL 127 --> go to 127, r7 = 65

65 : 1010000000000000; -- no-op

66 : 1010000000000000; -- no-op

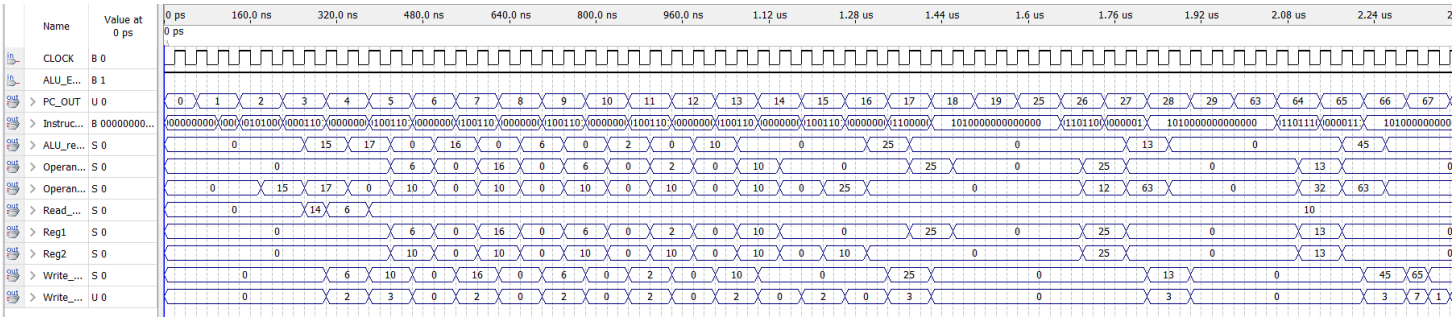
67 : 1010000000000000; -- no-op

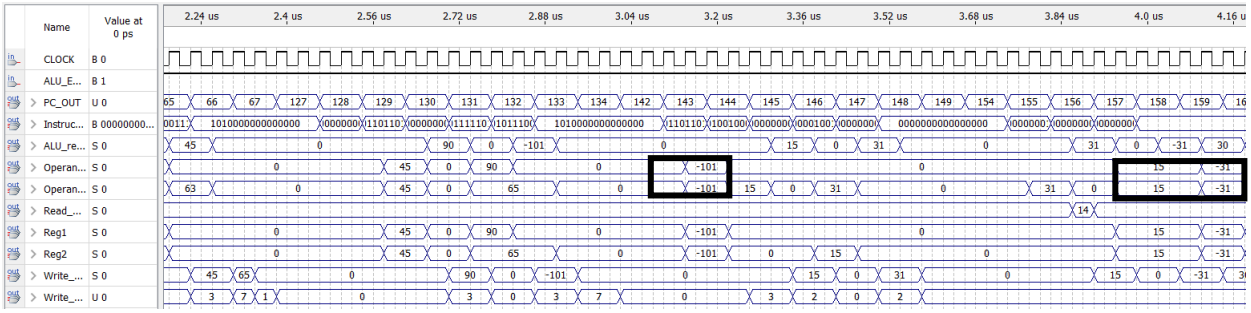
128 : 0000011011011000; -- ADD r3, r3, r3 --> r3 = 90

```
129 : 1010000000000000; -- no-op

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

131 : 1001010111001010; -- BNE r2, r7, 10 --> go to 142

132 : 1010000000000000; -- no-op

133 : 1010000000000000; -- no-op

134 : 1010000000000000; -- no-op

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

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

144 : 1010000000000000; -- no-op

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

146 : 1000000000000111; -- BEQ r0, r0, 7 --> Go to 154

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

155 : 1010000000000000; -- no-op

156 : 0000000000000100; -- MULT r0, r0, r0 --> r0 = 225

END;
```

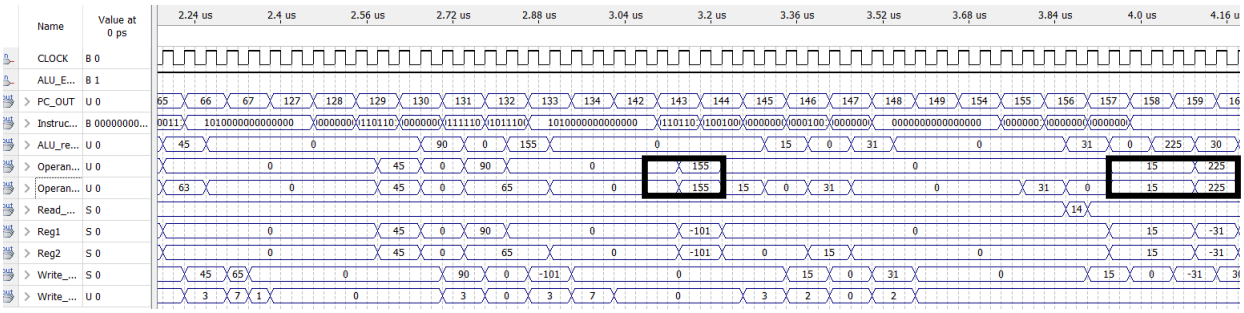
همانگونه که مشاهده می‌شود به منظور جلوگیری از Data Hazard، میان برخی دستورات از no-operation استفاده شده که دستوری است که وجود ندارد (صرفا برای جلوگیری از Data Hazard است و کار خاصی انجام نمی‌دهد)

این فایل دستورات را در مدار Pipeline خود اجرا می‌کنیم:





با مشاهده PC_OUT به وضوح می‌توان خط دستور اجرا شده را مشاهده کرد و سیر مسیر Branch, JAL, Jump ها واضح است. از جهتی مقادیر حاصل شده از دستورات را می‌توان در Reg1 و Reg2 و ALU_result مشاهده کرد. دقت شود که مقداری که دور آن مستطیل کشیده شده، در اصل 15 * 15 یعنی 225 است اما به دلیل overflow، برابر 31- شده است، همچنین مستطیل دیگر نیز 65 + 90 بوده که 155 می‌شود اما به دلیل overflow، برابر 101- شده. برای دیدن مقدار واقعی آنها، radix را به unsigned تغییر داده و مجددا عکس می‌گیرم:



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;

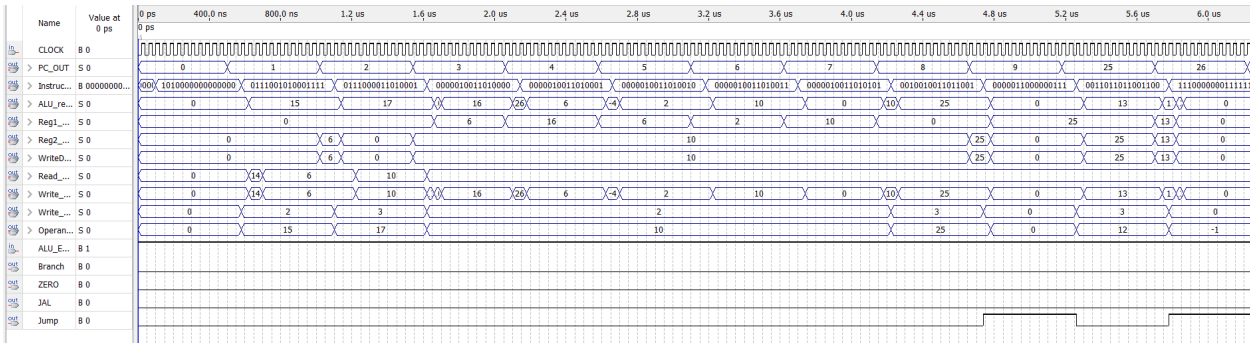
CONTENT BEGIN

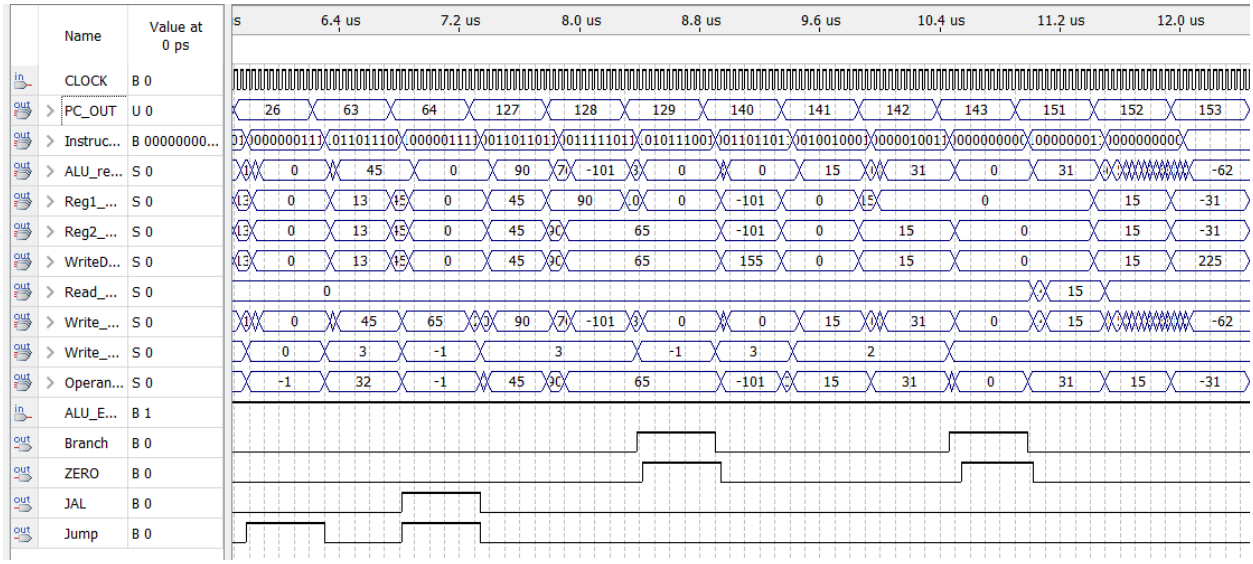
0 : 1010000000000000; -- no-op

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

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

3 : 0000010011010000; -- ADD r2, r2, r3 --> r2 = 16

4 : 0000010011010001; -- SUB r2, r2, r3 --> r2 = 6

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

6 : 0000010011010011; -- OR r2, r2, r3 --> r2 = 0010 | 1010 = 1010 = 10

7 : 0000010011010101; -- XOR r2, r2, r3 --> r2 = 0

8 : 0010010011011001; -- ADDi r3, r2, 25 --> r3 = 25

9 : 0000011000000111; -- JR r3 --> Go to 25

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

26 : 1110000000111111; -- J 63

63 : 0101011011100000; -- ORi r3, r3, 111111b --> r3 = 00100000 | 00001101 = 00101101 = 45

64 : 1111000001111111; -- JAL 127 --> go to 127, r7 = 128

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

129 : 0000011111011000; -- ADD r3, r3, r7 --> r3 = 218

130 : 1001010111001010; -- BNE r2, r7, 10 --> go to 131

131 : 0000011011011001; -- SUB r3, r3, r3 --> r3 = 0

132 : 0010010010001111; -- ADDi r2, r2, 15 --> r2 = 15

133 : 0110000010011111; -- SB r2, 31(r0) --> Mem[31] = 15

134 : 1000000000000111; -- BEQ r0, r0, 7 --> Go to 142

142 : 0111000000011111; -- LB r0, 31(r0) --> r0 = 15

143 : 0000000000000100; -- MULT r0, r0, r0 --> r0 = 225

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 می‌شود، 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.