معماری کامپیوتر - دکتر عطار زاده

تمرین سری ۷

سوال ۱

در هنگام اجرای خط اول برنامه، وضعیت پایپ‌لاین به صورت زیر است.

Cycles	1	2	3	4	5
`addi s1, zero, 51`	F	D	E	M	WB

با توجه به وابستگی دستور دوم به دستور اول، پیش‌فرستادن به صورت زیر انجام می‌شود.

Cycles	1	2	3	4	5	6
`addi s1, zero, 51`	F	D	E	M forward s1	WB	
`addi s0, s1, -4`		F	D	E	M	WB

در بالا، مقدار رجیستر s1 از مرحله Memory دستور اول، به مرحله Execute مرحله دوم پیش‌فرستاده می‌شود.

دستور سوم نیز به دستور دوم وابستگی دارد. بنابراین پیش‌فرستی به صورت زیر انجام می‌شود.

Cycles	1	2	3	4	5	6	7
`addi s1, zero, 51`	F	D	E	M forward s1	WB		
`addi s0, s1, -4`		F	D	E	M forward s0	WB	
`lw s3, 16(s0)`			F	D	E	M	WB

دستور lw باعث متوقف شدن پایپ‌لاین می‌شود. بنابراین اجرای دو دستور بعدی به صورت زیر خواهد بود (قسمت‌های قرمز شده توقف پایپ‌لاین را نشان می‌دهند).

Cycles	1	2	3	4	5	6	7	8	9	10
`addi s1, zero, 51`	F	D	E	M forward s1	WB					
`addi s0, s1, -4`		F	D	E	M forward s0	WB				
`lw s3, 16(s0)`			F	D	E	M	WB forward s3			
`sw s3, 20(s0)`				F	D	D	E	M	WB	
`xor s2, s0, s3`					F	F	D	E	M	WB

همچنین به دلیل وابستگی دستور sw به lw قبل از آن، پیش‌فرستی رخ می‌دهد که در شکل مشخص شده است.

با اضافه شدن دستور آخر، شیوه اجرای پایپ‌لاین به صورت زیر خواهد بود (صفحه بعد).

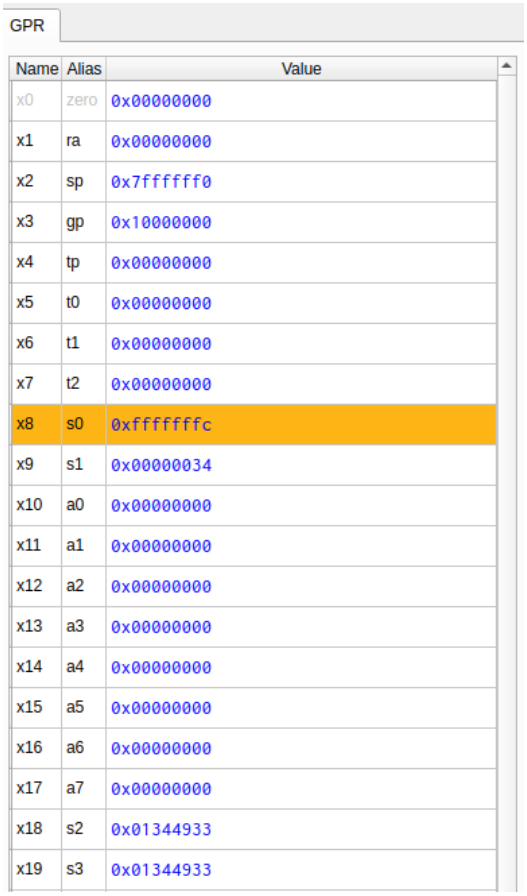
Cycles	1	2	3	4	5	6	7	8	9	10	11
`addi s1, zero, 51`	F	D	E	M forward s1	WB						
`addi s0, s1, -4`		F	D	E	M forward s0	WB					
`lw s3, 16(s0)`			F	D	E	M	WB forward s3				
`sw s3, 20(s0)`				F	D	D	E	M	WB		
`xor s2, s0, s3`					F	F	D	E	M forward s2	WB	
`or s2, s2, s3`							F	D	E	M	WB

نتیجه شبیه‌سازی کد داده شده با پردازنده دارای hazard detection به صورت زیر است.

GPR

Name	Alias	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7ffffff0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0	0x00000030
x9	s1	0x00000034
x10	a0	0x00000000
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000
x18	s2	0x00000030
x19	s3	0x00000000

همچنین نتیجه شبیه‌سازی با پردازنده بدون پشتیبانی از hazard detection به صورت زیر است.



به دلیل عدم پشتیبانی از forwarding، در هنگام انجام دستور خط ۲، از مقدار اولیه s1 به جای مقدار تغییر داده شده s1 توسط دستور خط ۱ استفاده شده است که باعث شده مقادیر اشتباه محاسبه شوند.

سوال ۳

برای جابه‌جایی منطق branch به مرحله decode، تغییرات زیر را انجام می‌دهیم.

- یک ALU در این قسمت از پایپ‌لاین اضافه می‌کنیم. از این ALU برای تصمیم‌گیری برای انجام branching استفاده می‌شود. ورودی‌های این ALU همان RD1 و RD2 است که همان خروجی‌های register file هستند. خروجی حاصل به جایی وصل نخواهد بود.

- به جای ALU در مرحله execute، از ALU اضافه شده در مرحله قبل خروجی zero را به واحد کنترل می‌بریم (در واقع سیگنال ZeroD را می‌سازیم و جایگزین ZeroE می‌کنیم). همچنین سیگنال‌های JumpD و BranchD را برای محاسبه سیگنال PCSrc استفاده می‌کنیم (در واقع سیگنال PCSrcD را می‌سازیم که ورودی‌های آن JumpD و BranchD و ZeroD هستند).

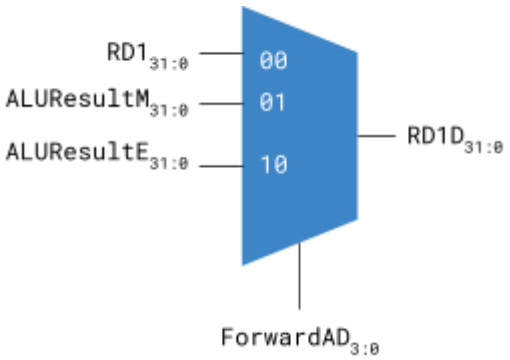
- با توجه به این که از register file مقادیری را می‌خوانیم، امکان وقوع خطای RAW در مرحله decode وجود دارد. برای حل مشکل، امکان forwarding را به این مرحله اضافه می‌کنیم. به همین منظور سیگنال ForwardAD و ForwardBD را می‌سازیم که مقدار آن به صورت زیر مشخص می‌شود.

```
If ((Rs1D == RdE) & RegWriteE) & (Rs1D != 0) then
      ForwardAD = 10
Else If ((Rs1D == RdM) & RegWriteM) & (Rs1M != 0) then
      ForwardAD = 01
Else
      ForwardAD = 00
```

لازم به ذکر است که نیازی به forward کردن از مرحله writeback نیست. چون مقدار رجیستر در نیمه اول کلاک نوشته می‌شود و در نیمه دوم خوانده می‌شود. پس بدون نیاز به forward، مقدار درست خوانده می‌شود.

مقدار سیگنال ForwardBD نیز مشابه بالا است، با این تفاوت که از سیگنال‌های Rs2 استفاده می‌شود.

همچنین به ورودی‌های ALU ای که در مرحله decode اضافه کردیم، یک mux با ورودی‌های زیر وصل می‌کنیم تا forwarding به درستی انجام شود.



مشابه mux بالا، برای RD2 نیز یک mux اضافه می‌کنیم.

- برای محاسبه آدرس هدف هنگام branching، ماژول adder ای که در قسمت execute داشتیم را به قسمت decode منتقل می‌کنیم. سپس ورودی‌های PCD و ImmExtD را به آن می‌دهیم.
همچنین خروجی آن را (که آن را PCTargetD می‌نامیم) به mux ای که قبل از PC register قرار دارد وصل می‌کنیم.

- سیگنال‌های forward قبلی و stall تغییری نمی‌کنند. نمی‌کند. تنها سیگنال‌های Flush تغییر می‌کنند:

```
FlushD = PCSrcD
FlushE = lwStall
```

درواقع با توجه به این که branching به مرحله decode منتقل شده است و تصمیم برای branch کردن زودتر گرفته می‌شود، وقتی branch رخ می‌دهد، نباید رجیستر مربوط به مرحله execute را flush کرد.

همچنین مقدار CPI دستورات branch هنگامی که misprediction رخ می‌دهد (یا در این حالت خاص، وقتی branch رخ می‌دهد) از ۳ به ۲ کاهش می‌یابد. به طور مشابه، دستورات jump نیز با CPI برابر ۲ اجرا خواهند شد.

(مثال ۷.۹) برای محاسبه CPI میانگین، میانگین CPI در دستورات branch را محاسبه می‌کنیم.

$$Branch\ Average\ CPI\ =\ (0.5)(1)\ +\ (0.5)(2)\ =\ 1.5$$

در نهایت، میانگین کل CPI را حساب می‌کنیم.

$$Average\ CPI\ =\ (0.25)(1.4)\ +\ (0.1)(1)\ +\ (0.11)(1.5)\ +\ (0.02)(2)\ +\ (0.52)(1)\ =\ 1.175$$

(مثال ۷.۱۰) با اضافه شدن موارد جدید، مسیر بحرانی در مرحله decode، از Register file، یکی از mux ها، ALU، و یک گیت and و or و همچنین از mux قبل از PC Register خواهد گذشت. همچنین چون این مسیر بحرانی باید در نصف سیکل طی شود، در نتیجه مقدار t_{C_decode} برابر خواهد بود با

$$t_{C\_decode} = 2(t_{RFread} + t_{setup} + 2t_{mux} + t_{ALU} + t_{AND\_OR} + t_{pcq})$$

با توجه به جدول ۷.۷ کتاب

$$t_{C\_decode} = 2(100 + 50 + 2 \times 30 + 120 + 20 + 40) = 780ps$$

در نتیجه

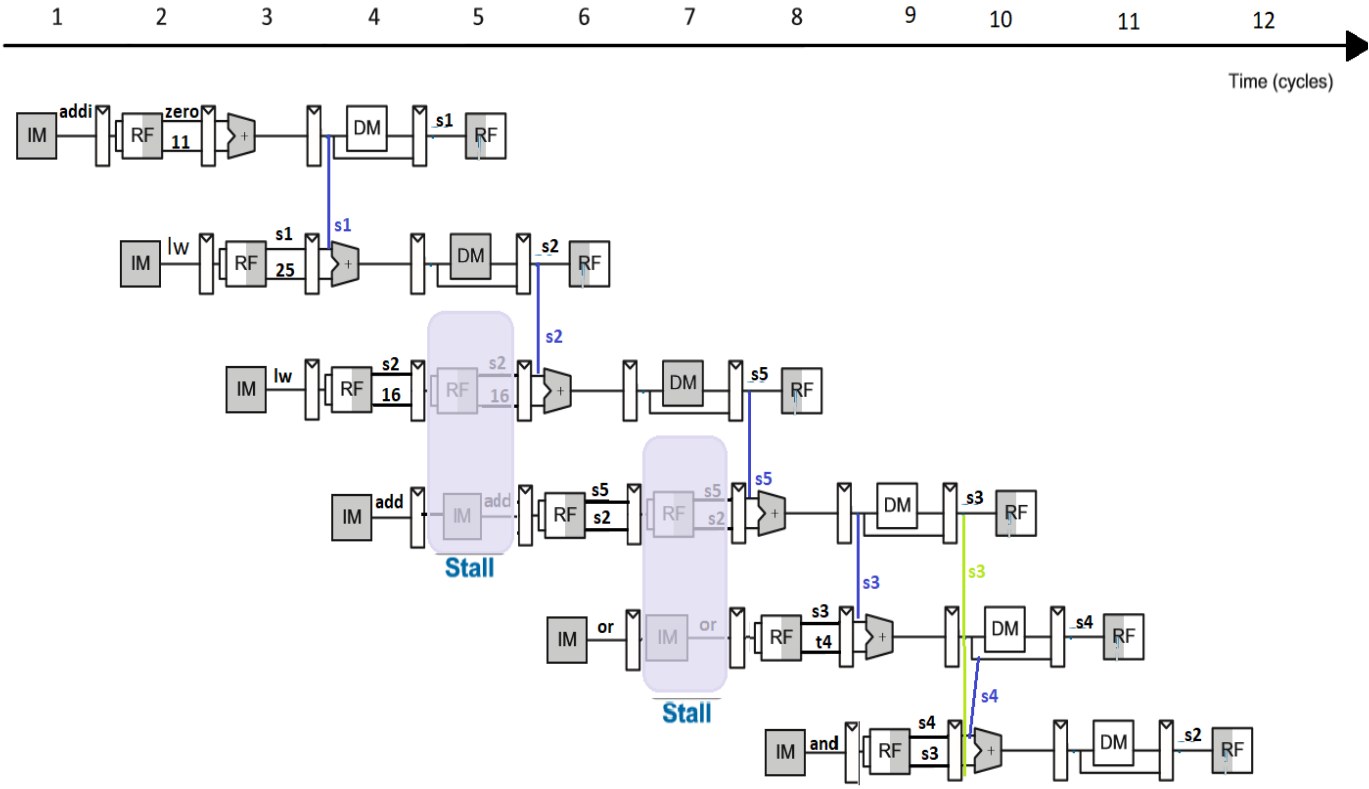
$$t_C = 780ps$$

و زمان اجرای کل خط‌های SPECINT2000 برابر است با

$$Total\ time\ =\ (100 \times 10^{9})(1.175)(780 \times 10^{-12})\ =\ 91.65$$

سوال۲) این سوال با فرض این که طبق کتاب pipeline ما دارای ۵ استیج است حل شده است.

دستور اول در ۵ سیکل انجام میشود. در دستور دوم به S1 نیاز است که در سیکل چهارم آماده استفاده است لذا نیازمند به stall کردن نیست و s1 با forwarding به دستور lw اول میدهد و این دستور نیز در ۵ سیکل انجام میشود.



در دستور سوم برای محاسبات alu در سیکل نیازمند S2 هستیم که هنوز مقدار دهی نشده است. لذا باید stall داشته باشیم و چون مرحله decode مشغول است د رمرحله fetchو stall میشود. وقتی دستور چهارم وارد مرحله decode میشود هنوز یکی از رجیستر های مورد نیازش(s5) مقدار دهی نشده است لذا دوباره باید در مرحله decode در حالت stall قرار بگیرد و در این مرحله چون decode مشغول است دستور پنجم در استیت fetch در حالت stall قرار میگیرد. دستور چهارم با تکنیک forwarding رجیستر s5 را دریافت میکند. دستور پنجم نیز رجیستر s3 و دستور ششم رجیستر های s3 و s4 را با forwarding دریافت میکنند.

لذا قابل توجه است که RAW data hazards با استفاده از تکنیک forwarding و stall برطرف شدند.

ب)همانطور که در شکل نیز مشخص است اجرای این کد ۱۲ سیکل طول میکشد و با توجه به اینکه ۶ دستور داریم CPI برابر با ۲ است:

CPI =num of cycles/ num of instructions = 12 / 6 = 2