درس معماری کامپیوتر

تمرین سری هفتم **۱۴۰۴/۰۳/۱۶**

متین باقری **۴۰۲۱۰۵۷۲۷**

---

۱) آ) در اینجا وقتی گفته می‌شود میان A و B data forwarding داریم، یعنی می‌توان خروجی A را به ورودی B متصل کرد. (به فلش های درون تصویر توجه کنید)

میان E3 و E1 Data Forwarding وجود دارد چون در کلاک 6 به مقدار R1 نیاز داریم.

میان E3 و E1 Data Forwarding وجود دارد چون در کلاک 7 به مقدار R2 نیاز داریم.

میان E2 و D Data Forwarding وجود دارد چون در کلاک 6 به مقدار R2 نیاز داریم. (optional)

میان E3 و E1 Data Forwarding وجود دارد چون در کلاک 11 به مقدار R4 نیاز داریم.

میان W و M Data Forwarding وجود دارد چون در کلاک 11 به مقدار R1 نیاز داریم. (استفاده از latch در RF)

Instructions	Cycles															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
MOVI R1, X	F	D	E1	E2	E3	M	W									
MOVI R2, Y		F	D	E1	E2	E3	M	W								
MUL R4, R1, R1			F	D	-	E1	E2	E3	M	W						
MUL R1, R1, R2				F	-	D	E1	E2	E3	M	W					
ADD R4, R5, R6						F	D	E1	E2	E3	M	W				
ADD R5, R2, R4							F	D	-	-	E1	E2	E3	M	W	
SUBI R3, R1, 2048								F	-	-	D	E1	E2	E3	M	W
JNZ L1											F	D	-	-	E1	...
...	...															

در کلاک ۴ مرحله D دستور سوم انجام می‌شود، در حالی که هنوز مقدار R1 حاصل از دستور اول در RF نوشته نشده است. و این یعنی در مرحله E1 ورودی ها میتوانند علاوه بر RF از حاصل مراحل بعدی به این قسمت بیایند. E1 دستور سوم در

کلاک ۶ انجام می‌شود، یعنی ورودی آن از حاصل انجام E3 دستور اول در کلاک پنجم می‌آید و در اینجا data forwarding داریم. (البته که واقع گرایانه نیست، چراکه دستور MOVI معادل li است که در آن مقدار imm در دستور وجود دارد و پس از مرحله F آماده است، نهایتا نیاز به یک مرحله محاسبه برای sign extend کردن آن داشته باشیم؛ در هر صورت نتیجه زودتر از مرحله E3 آماده است. از آنجا که تصویر جدول بالا صرفا یک مثال آموزشی است، تمام حالات ممکن برای وجود data forwarding را بررسی می‌کنیم.)

در کلاک ۶ مرحله D دستور ۴ انجام می‌شود، **میتوان** فرض کرد که نتیجه دستور دوم پس از مرحله E2 حاضر است و از E2 به D data forwarding داریم. (برای اینکه stall شدن دستور ۴ام پیش از مرحله D را توجیه کنیم) توجه داریم که بدون در نظر گرفتن این data forwarding و صرفا با وجود data forwardingیی که در قسمت قبل ذکر کردیم نیز، مشکلی به وجود نمی‌آید.

در کلاک 14 فلگ ALU zero تعیین شده و در کلاک 15 در مرحله E1 دستور JNZ از آن استفاده می‌شود و در اینجا data forwarding نداریم.

ب) سخت افزاری. چون به صورت خودکار، در صورت وجود data hazard/dependency فرایند اجرای دستور را stall میکند. در حالی که در حالت نرم افزاری، nop وارد پردازنده می‌شود.

ج) $T = 8$

نحوه اجرای کد به این صورت است:

MOVI R1, X # R1 <- X = (1) 4	
MOVI R2, Y # R2 <- Y = (2) 2	
L1:	
MUL R4, R1, R1 # R4 <- R1 × R1	R4 = (3) 16, (9) 64, …, (45)
MUL R1, R1, R2 # R1 <- R1 × R2	R1 = (4) 8, (10) 16, 32, 64, 128, 256, 512, (46) 1024, (52) 2048
ADD R4, R5, R6 # R4 <- R5 + R6	R4 = (5) 0, (11) 2, …, (53)
ADD R5, R2, R4 # R5 <- R2 + R4	R5 = (6) 2, (12) 4, …, (54)
SUBI R3, R1, 2048 # R3 <- R1 - 2048	R3 = (7) -2040, (13) -2032, …, (55) 0
JNZ L1	does_jump = (8) Yes, (14) Yes, …, (56) No
MUL R1, R1, R2 # R1 <- R1 × R2	R1 = (15) 4096, , …, (57)

مقداری که هر ثبات مقصد در هر دستور می‌گیرد جلوی آن دستور نوشته شده. مقادیر آبی رنگ درون پرانتز که قبل از هر مقدار آمده، شماره کلاکی است که دستور در آن اجرا می‌شود. مقادیر T و N را با توجه به کد بالا به دست آوردیم.

د) $N = 45$

ه) در کل 57 دستور اجرا شده و چرخه ی L1 ۹ مرتبه اجرا می‌شود.

۲) در طراحی خط لوله، در هر لحظه n دستور به طور موازی در حال اجرا اند. با گذشت هر T_p انجام یک دستور به پایان می‌رسد. بنابراین باید تابع T_p را کمینه کنیم. به ازای $n = 20$ مقدار تابع برابر با 2 است که کمترین مقدار آن در دامنه اعداد طبیعی است. پس $n = 20$ و $T_p = 2ns$ و $throughput = 1\ instruction / 2ns = 0.5\ G\ Hz$

---

۳)

ابتدا، اجرای پایپ‌لاین را در حالتی که هیچ گونه دور زدن [۵] وجود ندارد، نشان دهید:

Instructions	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲
sub $2, $3, $1	f	D	X	M	W							
lw $5, 0($2)		f		d*	D	X	M	W				
addi $4, $5, 1			f		d*	d*	d*	D	X	M	W	
add $5, $3, $1				f		d*		d*	D	X	M	W

اکنون نشان دهید اگر پایپ‌لاین دارای **دور زدن کامل** باشد، چه تفاقی می‌افتد:

Instructions	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲
sub $2, $3, $1	f	D	X	M	W							
lw $5, 0($2)		f	D	X	M	W						
addi $4, $5, 1			f	d*	D	X	M	W				
add $5, $3, $1				f	d*	D	X	M	W			

در نهایت، عملکرد این قطعه کد در صورت وجود دور زدن چقدر بهبود یافته است؟ (به صورت درصدی بیان شود)

این ۴ دستور به جای ۱۲ کلاک در ۹ کلاک انجام شدند که بهبود 25٪ را نشان می‌دهد. البته باید توجه کنیم که این دستورات، در شروع کار قرار دارند و پر شدن pipeline و رسیدن به حالت steady state نیازمند صرف چندین کلاک بوده است. نگرش دیگر این است که در حالت اول $3 + 3 + 1 = 7$ کلاک برای اتمام 3 دستور نیاز بوده و در حالت دوم $1 + 2 + 1 = 4$ کلاک برای اتمام 3 دستور نیاز بوده است. در این صورت بهبود برابر 43% بوده است.

۴) **a.** آ) با در نظر گرفتن اینکه:

“in most MIPS pipeline implementations, new independent instructions can be fetched and started even while a previous instruction is stalled—as long as there’s no conflict.”

```
Label1:
        LW R2,0(R2)
        BEQ R2,R0,Label1 # Taken once, then not taken
        OR R2,R2,R3
        SW R2,0(R5)
```

instr\clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw	F	D	X	M	W									
beq		F	D	_	X	M	W							
lw			F	_	D	X	M	W						
beq				F	_	_	D	X	M	W				
lw					F	_	_	D	~					
beq						F	_	_	~					
lw							F	_	~					
beq								F	~					
or									F	D	X	M	W	
sw										F	D	X	M	W

~ means instruction is dropped and not continued anymore. (due to branch misprediction)

a. ب) از آنجا که در متن سوال مشخص نشده که delay slot دقیقا چه دستوری است، فرض می‌کنیم nop است.

```
Label1:
        LW R2,0(R2)
        BEQ R2,R0,Label1 # Taken once, then not taken
        OR R2,R2,R3
        SW R2,0(R5)
```

instr\clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw	F	D	X	M	W									
beq		F	D	_	X	M	W							
nop			nop	nop	nop	nop	nop							
lw				F	D	X	M	W						
beq					F	D	_	X	M	W				
nop						nop	nop	nop	nop	nop				
lw							F	D	~					
beq								F	~					
or									F	D	X	M	W	
sw										F	D	X	M	W

(ĩ .b

```
LW R2,0(R1)
Label1: BEQ R2,R0,Label2 # Not taken once, then taken
LW R3,0(R2)
BEQ R3,R0,Label1 # Taken
ADD R1,R3,R1
Label2: SW R1,0(R2)
```

instr\clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14
lw	F	D	X	M	W									
beq		F	D	_	X	M	W							
sw			F	_	D	~								
lw				F	_	D	X	M	W					
beq					F	_	D	_	X	M	W			
add						F	_	D	_	~				
beq							F	_	D	X	M	W		
sw								F	_	D	X	M	W	

b.ب)

```
LW R2,0(R1)
Label1: BEQ R2,R0,Label2 # Not taken once, then taken
LW R3,0(R2)
BEQ R3,R0,Label1 # Taken
ADD R1,R3,R1
Label2: SW R1,0(R2)
```

instr\clk	1	2	3	4	5	6	7	8	9	10	11	12	13
lw	F	D	X	M	W								
beq		F	D	_	X	M	W						
nop			nop	nop	nop	nop	nop						
lw				F	_	D	X	M	W				
beq					F	_	D	_	X	M	W		
nop						nop	nop	nop	nop	nop			
beq							F	_	D	X	M	W	
nop								nop	nop	nop	nop	nop	
sw									F	D	X	M	W

۵) آ)

8*100 + 2 + 1 = 803 instruction

ب) CPI = 1

803 * 8 ns = 6,424 ns = 6.424 micro seconds

ج)

instr \ clk	1	2	3	4	5	6	7	8	9	10
lb $t2, 0($a0)	F	D	X	M	W					
beq $t2, $0, exit		F	_	D	X	M	W			
blt $t2, 97, next				F	D	X	M	W		
bgt $t2, 122, next					F	D	X	M	W	
sub $t2, $t2, 32						F	D	X	M	W

~= 1039.1 clock cycle

د) با اختلاف عملکرد بهتری دارد.

2 * 1039.1 = 2078.2 ns

۶) آ) در پیش‌بینی‌کننده‌ی ۲-بیتی اشباعی، برای هر دستور پرش (مانند beq در MIPS)، یک شمارنده‌ی ۲ بیتی نگهداری می‌شود. این شمارنده می‌تواند یکی از چهار حالت زیر را داشته باشد:

00 Strongly Not Taken پرش انجام نمی‌شود (پیش‌بینی منفی قوی)

01 Weakly Not Taken پرش انجام نمی‌شود (پیش‌بینی منفی ضعیف)

10 Weakly Taken پرش انجام می‌شود (پیش‌بینی مثبت ضعیف)

11 Strongly Taken پرش انجام می‌شود (پیش‌بینی مثبت قوی)

هنگام اجرای دستور پرش ، پردازنده به مقدار این شمارنده نگاه می‌کند. اگر مقدار 2 بیت بالا باشد (10 یا 11)، پرش پیش‌بینی‌شده انجام می‌شود. اگر مقدار پایین باشد (00 یا 01)، انشعاب پیش‌بینی‌شده انجام نمی‌شود. اگر پیش‌بینی درست بود، مقدار شمارنده به همان سمت حفظ می‌شود یا تقویت می‌شود (مثلاً از 10 به 11 می‌رود). اگر پیش‌بینی اشتباه بود، شمارنده به سمت مخالف حرکت می‌کند (مثلاً از 11 به 10 یا از 01 به 10). اشباعی بودن به این معنی است که مقدار شمارنده از محدوده 00 تا 11 خارج نمی‌شود. اگر فقط ۱ بیت داشته باشیم، پیش‌بینی‌کننده ممکن است در پرش هایی که رفتار متناوب دارند (مثل یک‌بار yes، یک‌بار no) خیلی زود تغییر کند. اما با ۲ بیت، پیش‌بینی باید دو بار اشتباه شود تا جهت آن تغییر کند و پایداری بیشتری دارد.