Computer Architecture

HW6 – G14

Mohammad Bahrami & Navid Raeeszadeh

گزارش سوال اول:

طبق جدول 7.7 کتاب مقدار تاخیر120 پیکوثانیه است.

الف)کاهش بیست درصدی: 96 پیکوثانیه .

Table 7.7 Delay of circuit elements

Element	Parameter	Delay (ps)
Register clk-to-Q	t_{pcq}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	30
AND-OR gate	$t_{AND\text{-}OR}$	20
ALU	t_{ALU}	120
Decoder (control unit)	t_{dec}	25
Extend unit	t_{ext}	35
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

فرمول محاسبه کلاک پردازنده پایپ لاین:

$$T_{c\_pipelined} = max\begin{bmatrix} t_{pcq} + t_{mem} + t_{setup} & Fetch \\ 2(t_{RFread} + t_{setup}) & Decode \\ t_{pcq} + 4t_{mux} + t_{ALU} + t_{AND-OR} + t_{setup} & Execute \\ t_{pcq} + t_{mem} + t_{setup} & Memory \\ 2(t_{pcq} + t_{mux} + t_{RFsetup}) & Writeback \end{bmatrix} \quad (7.5)$$

مقدار کلاک سایکل پردازنده پایپ لاین به اندازه ی طولانی ترین استیج پایپ لاین است.

Fetch = 40 + 200 + 50 = 290 ps

Decode = 2*(100 + 50) = 300 ps

Execute = 40 + 4*30 + 96 + 20 + 50 = 326 ps

Memory = 40 + 200 + 50 = 290 ps

Writeback = 2*(40 + 30 + 60) = 260 ps

طولانی ترین استیج همان استیج Execute باقی ماند ولی تاخیر آن 24 تا کاهش پیدا کرد و به 326 پیکوثانیه رسید که این مقدار تاخیر برابر Tc_pipeline خواهد بود.

ب) افزایش 20 درصدی: 144 پیکوثانیه

تنها استیجی که تغییر خواهد کرد Execute خواهد بود چون مقدار ALU تنها در این استیج تاثیر گذار است.

Execute = 40 + 4*30 + 144 + 20 + 50 = 374 ps

مقدار Tc_pipeline برابر با 374 پیکوثانیه خواهد بود.

گزارش سوال دوم:

```
Addi   s0, zero, 24
addi   s1, zero, 16
sub    t0, s0, s1
lw     t1, 2(t0)
ori    t2, t1, 63
lw     s2, 0(t2)
xori   t1, s2, 27
sw     t1, 0(s1)
```

از سیکل پنجم شروع می‌شود، هر کدام از دستورات addi و sub و ori و xori در طی یک سیکل ساعت انجام میشوند ولی دستور lw به دلیل stall در طی دو سیکل انجام میشود. پس مجموعا در 10 سیکل ساعت انجام میشود (هر خط یک سیکل، lw ها در دو سیکل) پس مجموعا 14 سیکل طول خواهد کشید که cpi برنامه با هشت دستور می‌شود (8 / 14). Cpi = 1.75

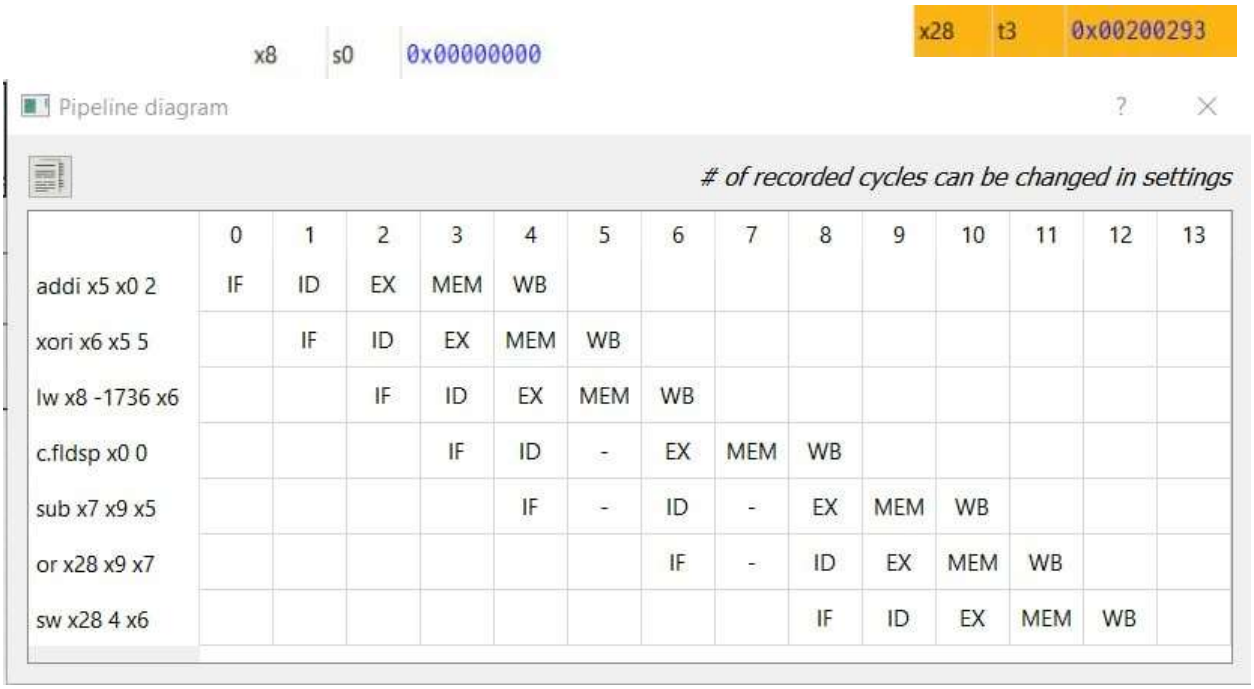
گزارش سوال سوم:

هنگامی که hazard را هندل نمی‌کنیم کلا دستور اشتباه فهمیده می‌شود و خروجی‌ها تماما نادرست می‌شوند. هزارد های بوجود آمده هنگام RAW رخ خواهند داد. بدون کنترل هزارد با فورواردینگ، در هنگام کامپایل عبارت nop قرار داده نمی‌شود، بنابراین خروجی‌های ما نادرست خواهند بود.

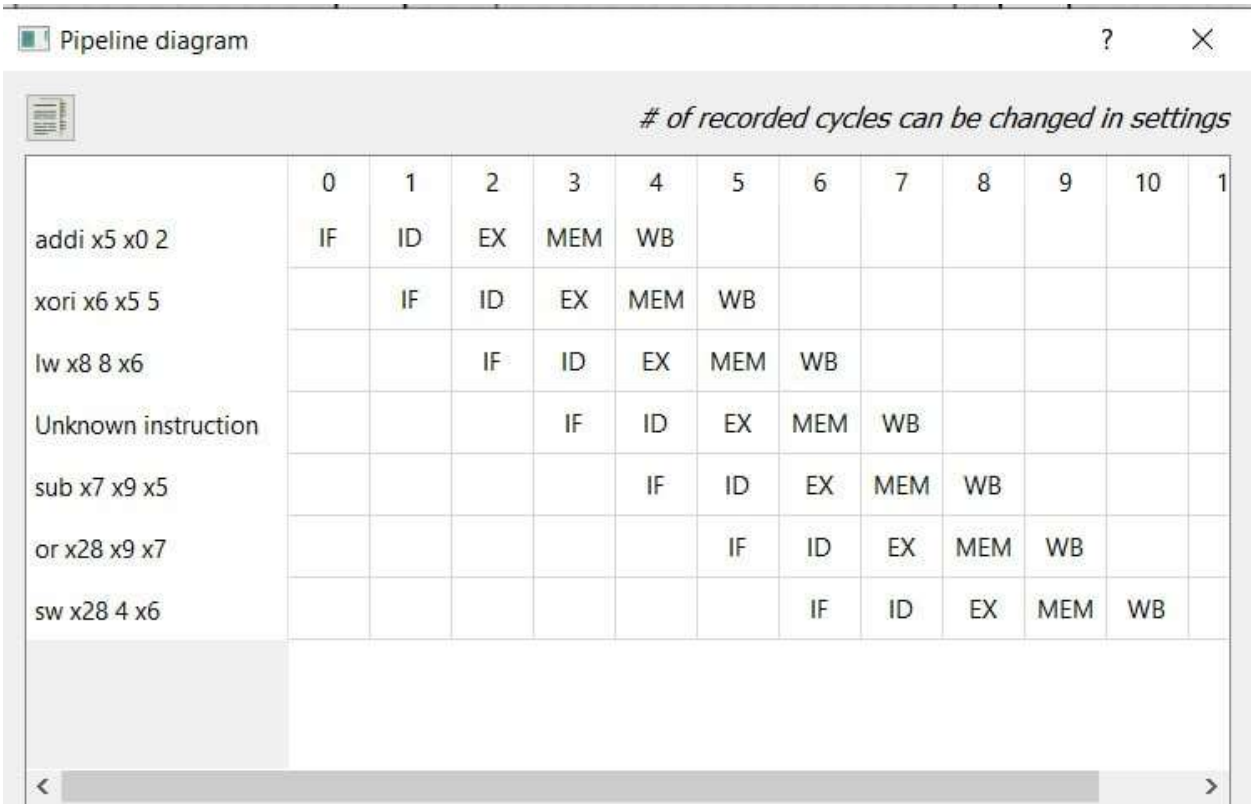
جهت هندل کردن هزارد از روش forwading استفاده کرد، به این صورت که دیتاها را از باس‌های درونی به استیج‌های اجرا میفرستیم. برای دستوری مانند lw هم که استفاده از روش forwarding امکان پذیر نیست از stalling استفاده میکنیم و بدین ترتیب latency آن دو سیکل خواهد شد.

در تصاویر زیر از خروجی‌ها، با قابلیت رفع مخاطره و حالت درست آن با forwarding نمایش داده می‌شود.

مقادیر درست و مطابق انتظار s0 و t3 با forwardingو رفع مخاطره



	0	1	2	3	4	5	6	7	8	9	10	11	12	13
addi x5 x0 2	IF	ID	EX	MEM	WB									
xori x6 x5 5		IF	ID	EX	MEM	WB								
lw x8 -1736 x6			IF	ID	EX	MEM	WB							
c.fldsp x0 0				IF	ID	-	EX	MEM	WB					
sub x7 x9 x5					IF	-	ID	-	EX	MEM	WB			
or x28 x9 x7							IF	-	ID	EX	MEM	WB		
sw x28 4 x6									IF	ID	EX	MEM	WB	

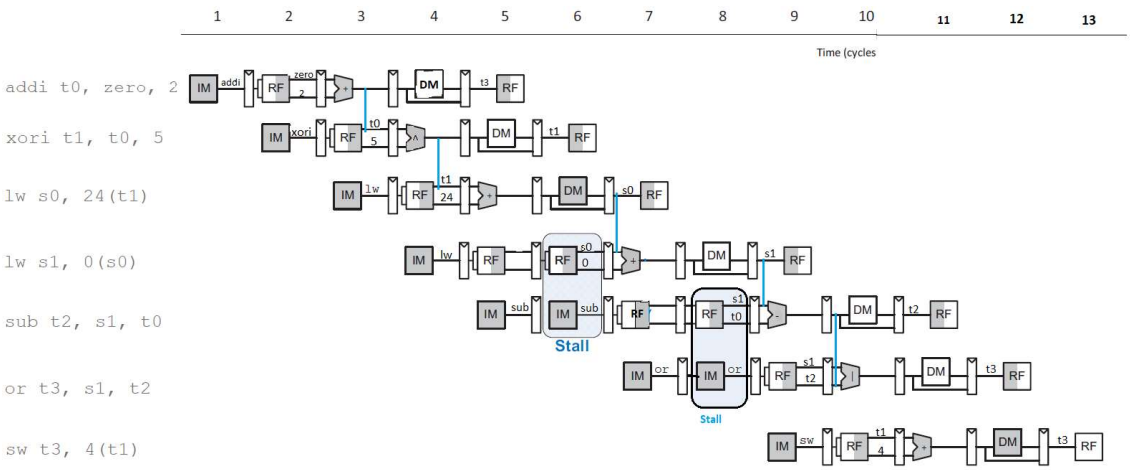


	0	1	2	3	4	5	6	7	8	9	10	1
addi x5 x0 2	IF	ID	EX	MEM	WB							
xori x6 x5 5		IF	ID	EX	MEM	WB						
lw x8 8 x6			IF	ID	EX	MEM	WB					
Unknown instruction				IF	ID	EX	MEM	WB				
sub x7 x9 x5					IF	ID	EX	MEM	WB			
or x28 x9 x7						IF	ID	EX	MEM	WB		
sw x28 4 x6							IF	ID	EX	MEM	WB	

x8	s0	0x00000000	x28	t3	0x0000001d

مقدار نادرست ذخیره شده در s0 و t3 بدون کنترل هزارد رخ داده

برای بخش ب، همان‌گونه که در شکل مشخص شده در 5 جا به forwarding نیاز داشتیم، در دستورات (addi – xori – sub – or) و از آنجایی که فورواردینگ برای lw امکان‌پذیر نمی‌باشد، برای lw هم از stall استفاده میکنیم و چون دو دستور lw داریم بنابراین به دو دستور lw احتیاج داریم.


1
2
3
4
5
6
7
8
9
10
11
12
13
Time (cycles
addi t0, zero, 2
xori t1, t0, 5
lw s0, 24(t1)
lw s1, 0(s0)
sub t2, s1, t0
or t3, s1, t2
sw t3, 4(t1)
IM
RF
DM
Stall