حل تمرین امتیازی **مهلت تحویل ساعت ۲۴ روز چهارشنبه ۶ دی**

به موارد زیر توجه کنید:

۱- حتما نام و شماره دانشجویی خود را روی پاسخ‌نامه بنویسید.

۲- کل پاسخ تمرینات را در قالب یک فایل pdf با شماره دانشجویی خود نام‌گذاری کرده در سامانه CW بارگذاری کنید.

۳- این تمرین ۱۰۰ نمره دارد که معادل ۱ نمره اضافه بر ۲۰ نمرهٔ درس است.

۴- در صورت مشاهده هر گونه مشابهت نامتعارف هر دو (یا چند) نفر کل نمره این تمرین را از دست خواهند داد.

۱- (۳۰ نمره) با استفاده از حداقل تعداد گذرگاه و مالتی‌پلکسر مداری برای ثبات‌های R1، R2، R3 و R4 طراحی کنید که سه دستور زیر را انجام دهد.

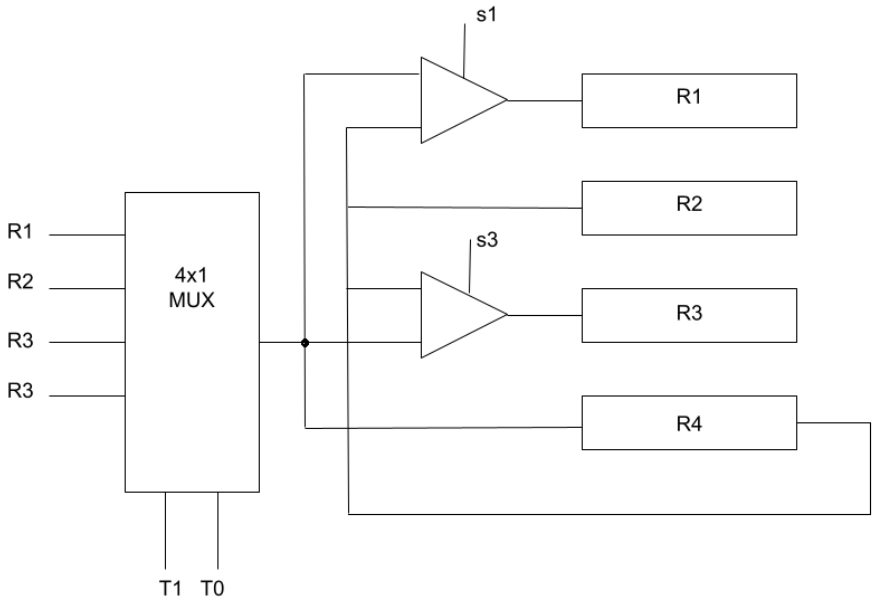
فرض کنید ثبات‌ها دارای enable هستند.

```
 T0.~T1: R2 ← R4, R4 ← R2
~T0.~T1: R1 ← R4, R2 ← R4, R3 ← R1
     T1: R1 ← R3, R3 ← R4
```

پاسخ:

می‌بینیم که در هر خط دو ثبات مبدا داریم، بنابراین باید دو گذرگاه داشته باشیم. یکی از این ثبات‌ها که در همه خط‌ها مشترک است R4 است، بنابراین یکی از گذرگاه‌ها فقط از R4 ورودی می‌گیرد. ورودی گذرگاه دیگر بسته به شرایط متغیر است، بنابراین به یک مولتی‌پلکسر ۴ به ۱ نیاز داریم که مقادیر یکی از ثبات‌های R1 و R2 و R3 را روی گذرگاه قرار بدهد.

از طرف دیگر می‌بینیم که R2 فقط از R4 ورودی می‌گیرد و R4 هم فقط از R2 ورودی می‌گیرد، بنابراین ورودی‌های این دو ثبات نیازی به mux ندارند. اما دو ثبات R1 و R3 از دو جا ورودی می‌گیرند، پس آنها در ورودی خود نیاز به mux دارند.



مقادیر کنترلی در شکل عبارتند از:

S1=T0’T1’

S3=T1

Load(R1)=T0’T1’+T1=T1+T0’

Load(R2)=T0T1’+T0’T1’=T1’

Load(R3)=T0’T1’+T1=T1+T0’

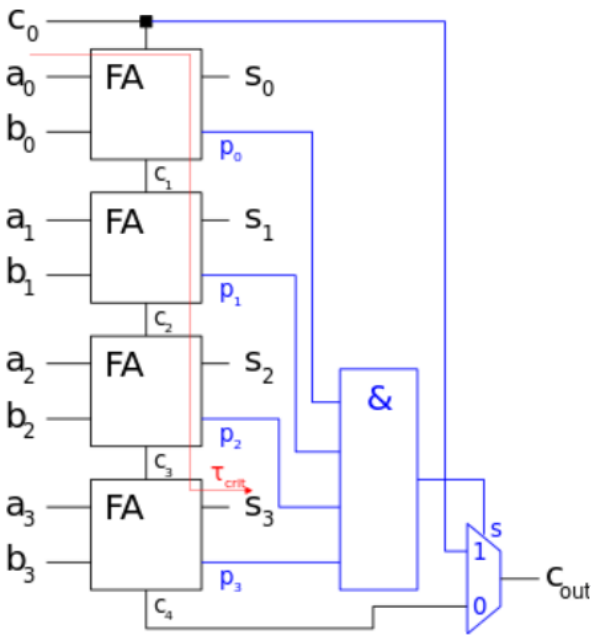
Load(R4)=T0T1’+T0’T1’=T1’

۲- (۲۰ نمره) شکل زیر یک جمع‌کنندهٔ چهار بیتی از نوع Carry Skip Adder را نشان می‌دهد.

الف- توضیح دهید این جمع‌کننده چطور کار می‌کند و چرا نتیجهٔ به دست آمده درست است.

ب- زمان لازم برای تولید بیت نقلی خروجی (DCout) بسته به مقدار ورودی‌های A و B متفاوت است. کمترین و بیشترین مقدار DCout را بر حسب تاخیر یک تمام‌افزا (DFA)، یک گیت AND چهار ورودی (DA4) و یک MUX دو به یک (DMUX) حساب کنید.

ج- می‌خواهیم با کنار هم گذاشتن چهار جمع‌کننده از همین نوع دو عدد ۱۶ بیتی را با هم جمع کنیم. کمترین و بیشترین مقدار زمان لازم برای آماده شدن حاصل‌جمع را محاسبه کنید. فرض کنید ورودی C0 هر جمع‌کنندهٔ چهاربیتی به خروجی Cout جمع‌کنندهٔ قبلی متصل است.



پاسخ:

الف- در این روش، اگر s=1 خروجی Cout برابر با ورودی Cin است. از طرفی s وقتی یک می‌شود که همهٔ piها یک باشند. می‌دانیم که هر یک از piها حاصل XOR دو بیت ai و bi هستند و تنها در صورتی pi=1 می‌شود که دو بیت ai و bi نامساوی باشند، یعنی یکی ۰ و دیگری ۱ باشد. در چنین شرایطی جمع این دو بیت carry نخواهد داشت. بنابراین، s در صورتی یک می‌شود که جمع دو به دوی هر کدام از بیت‌های ورودی carry نداشته باشد، پس Cout برابر با Cin خواهد بود. در غیر این‌صورت، s=0 شده و Cout از روش معمول به دست می‌آید.

یک راه دیگر برای توجیه کارکرد این مدار، استفاده از رابطه‌ای است که Cout را بر حسب piها بیان می‌کند. این رابطه را در جمع‌کننده‌های CLA دیده‌ایم:

$$C_{out} = G + PC_0 \quad P = p_0.p_1.p_2.p_3 \quad G = g_3 + p_3g_2 + p_3p_2g_1 + p_3p_2p_1g_0$$

طبق این رابطه، اگر P=1، Cout=C0 خواهد بود (چون در این صورت G هم صفر خواهد بود) و به همین دلیل مدار بالا درست کار می‌کند.

ب- استفاده از این روش فقط Cout را آن هم در شرایط خاص زودتر حاضر می‌کند. اگر نه، آماده شدنِ حاصل‌جمع در هر حال به اندازه $4D_{FA}$ زمان می‌برد، $3D_{FA}$ برای حاضر شدن C_3 و $1D_{FA}$ برای حاضر شدن S_3.

در مورد C_{out}کمترین تاخیر زمانی رخ می‌دهد که s=1 و بیشترین تاخیر زمانی رخ می‌دهد که s=0، بنابراین:

$$Delay_{min}(C_{out}) = Delay(p_i) + Delay(AND) + Delay(MUX) = D_{FA} + D_{A4} + D_{mux}$$
$$Delay_{max}(C_{out}) = 4D_{FA} + D_{MUX}$$

ج- اگر چهارتا جمع‌کننده از این نوع را به دنبال هم قرار بدهیم، وقتی کمترین تاخیر را داریم که در هر کدام از سه واحد جمع‌کنندهٔ چهاربیتی اول (مربوط به بیت‌های ۰ تا ۱۱) C_{out} در کمترین زمان ممکن حاضر شود:

$$Delay_{min}(C_{out1}) = D_{FA} + D_{A4} + D_{mux}$$
$$Delay_{min}(C_{out2}) = Delay_{min}(C_{out1}) + D_{mux}$$
$$Delay_{min}(C_{out3}) = Delay_{min}(C_{out2}) + D_{mux} = D_{FA} + D_{A4} + 3D_{mux}$$
$$Delay_{min}(S_{15}) = Delay_{min}(C_{out3}) + 4D_{FA}$$

بیشترین تاخیر هم وقتی رخ می‌دهد که در هر کدام از سه واحد جمع‌کنندهٔ اول Cout در بیشترین زمان ممکن حاضر شود:

$$Delay_{max}(S_{15}) = 16D_{FA} + 3D_{mux}$$

۳- اکثر کامپیوترها هنگام فراخوانی زیرروال، پارامترها را با استفاده از پشته‌ای که در حافظه قرار دارد ارسال می‌کنند. این بدان معنی است که هر فراخوانی و بازگشتی به چندین دسترسی حافظه نیاز دارد. یک معماری جایگزین به نام Berkely RISC از پنجرهٔ ثبات (register windows) استفاده می‌کند. با استفاده از این ثبات‌ها متغیرهای محلی که به عنوان پارامتر ارسال می شوند در مجموعه‌ای از ثبات ها قرار می‌گیرند که این ثبات ها بین ثبات‌های تابع فراخوان (caller) و تابع فراخوانی‌شده (callee) همپوشانی دارند. این همپوشانی اجازه می‌دهد پارامترها بدون استفاده از حافظه ارسال شوند. در این حالت پنجرهٔ ثبات جایگزین عملیات دسترسی حافظه شده است، پس تعداد دستورات ثابت مانده اما CPI تغییر می‌کند چون عملیات ثبات نسبت به عملیات حافظه به چرخه‌های کمتری نیاز دارد.

فرض کنید در یک پردازنده، تعداد چرخه‌های انواع دستورات به شرح زیر است:

loads/stores: 4, ALU & unconditional branches: 2,
conditional branches: 3, procedure calls & returns: 15

متخصصین معماری می‌خواهند بین افزایش تعداد ثبات‌ها و یا به‌کارگیری پنجرهٔ ثبات یکی را انتخاب کنند. افزایش تعداد ثبات‌ها تعداد دستورات load و store را به ترتیب ۴۰٪ و ۳۰٪ کاهش خواهد داد. از طرفی به‌کارگیری پنجرهٔ ثبات CPI را برای procedure calls به ۴/۵ و return را به ۳ می‌رساند. حال با یک برنامهٔ نمونه با توزیع دستورات به صورت زیر، کدام انتخاب بهتر است؟

40% load, 13% store, 31% ALU, 8% cond. branches, 2% uncond. branches, 3% procedure call, 3% return

پاسخ:

اگر تعداد ثبات‌ها را افزایش دهیم، تعداد دستورات load و store کم می‌شود. اگر فرض کنیم در مجموع ۱۰۰ دستور داریم، زمان اجرای این ۱۰۰ دستور در پردازنده‌ای که تعداد ثبات‌ها افزایش یافته این طور محاسبه می‌شود:

$$execCycles_1 = (40 \times 0.6 + 13 \times 0.7) \times 4 + 8 \times 3 + 33 \times 2 + 6 \times 15 = 312.4$$
$$execCycles_2 = (40 + 13) \times 4 + 8 \times 3 + 33 \times 2 + 3 \times 4.5 + 3 \times 3 = 324.5$$

بنابراین، اضافه کردن ثبات‌ها بهتر است.

۴- (۲۰ نمره) در خط لولهٔ یک پردازنده، اجرای هر دستور بدون وابستگی یک چرخه طول می‌کشد. فرض کنید درصد دستوراتی که وابستگی داده دارند طبق جدول زیر است. این جدول شامل دستوراتی است که نتیجه‌ای که در یک مرحله از خط لوله تولید می‌شود در یک دستور پس از آن، یا در دو دستور پس از آن و یا در هر دو دستور پس از آن موردنیاز خواهند بود. دربارهٔ این پردازنده به سوالات زیر پاسخ دهید.

	EX to 1st Only	MEM to 1st only	Ex to 2nd Only	MEM to 2nd Only	EX to 1st and EX to 2nd
Stalls	5%	20%	5%	10%	10%
with no forwarding	2	2	1	1	2
with full forwarding	0	1	0	0	0
with EX/MEM forwarding	0	2	1	1	1
with MEM/WB forwarding	1	1	0	0	1

الف- اگر هیچ سازوکاری برای هدایت به جلو (forwarding) نداشته باشیم. چه کسری از کل چرخه‌ها صرف تعلیق ناشی از وابستگی‌های داده‌ای خواهد شد؟

پاسخ: در این صورت تعداد تعلیق موردنیاز طبق اعداد ردیف دوم جدول خواهد بود، بنابراین:

$$stalls = 2 \times (0.05 + 0.2 + 0.1) + 1 \times (0.05 + 0.1) = 0.85$$

$$stallPercentage = 100 \times \frac{0.85}{1.85} \cong \%46$$

ب- اگر سازوکار هدایت به جلو به طور کامل استفاده شود (یعنی داده‌ها را در صورت امکان به صورت زودهنگام در اختیار بگیریم) چه کسری از چرخه‌ها صرف تعلیق ناشی از وابستگی‌های داده‌ای خواهد شد؟

پاسخ: در این صورت تعداد تعلیق موردنیاز طبق اعداد ردیف سوم جدول خواهد بود، بنابراین:

$$stalls = 1 \times 0.2 = 0.2$$

$$stallPercentage = 100 \times \frac{0.2}{1.2} \cong \%16.7$$

ج- فرض کنید به هر دلیلی امکان به‌کارگیری مولتی‌پلکسرهای سه‌ورودی برای پیاده‌سازی کامل هدایت رو به جلو نداریم. بنابراین باید میان یکی از این دو روش انتخاب کنیم؛ این که داده‌ها را ثبات EX/MEM یا از ثبات MEM/WB به ورودی ALU برسانیم. کدام‌یک از این دو گزینه به تعداد کمتری چرخهٔ تعلیق می‌انجامد؟

پاسخ: در این صورت تعداد تعلیق موردنیاز طبق اعداد ردیف‌های چهارم و پنجم جدول خواهد بود، بنابراین:

$$stallsEX/MEM = 2 \times (0.2) + 1 \times (0.25) = 0.65$$
$$stallsMEM/WB = 1 \times (0.05 + 0.2 + 0.1) = 0.35$$

پس مورد دوم بهتر است.

جدول زیر تاخیر مراحل مختلف خط لوله را مشخص می‌کنند. توجه دارید که تاخیر مرحلهٔ EX به نوع روش هدایت رو به جلویی که به‌کارگرفته شده بستگی دارد.

IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
150 ps	100 ps	120 ps	150 ps	140 ps	130 ps	120 ps	100ps

د- براساس احتمالات بروز وابستگی‌های داده‌ای و مقادیر تاخیر هر مرحله از خط لوله، افزودن مدارهای هدایت رو به جلو در قیاس با نبود این مدارها، چه میزان سرعت اجرا را افزایش خواهد داد؟

پاسخ: در هر دو حالت هر مرحله ۱۵۰ پیکوثانیه طول می‌کشد. در صورت نبود مدارهای هدایت رو به جلو، هر دستور به طور متوسط ۱,۸۵ مرحله طول می‌کشد و در صورت وجود این مدارها، هر دستور ۱,۲ مرحله طول می‌کشد.

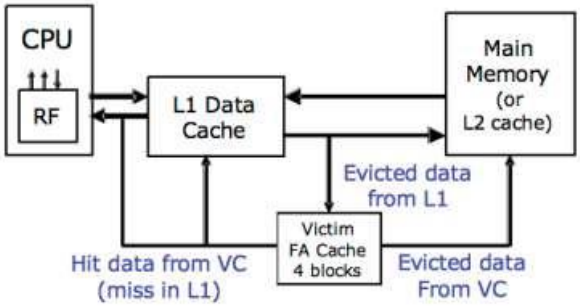
$$speedup = \frac{1.85}{1.2} = 1.54$$

ه- اگر میشد با سفر در زمان همه موانع داده‌ای را از بین ببریم و با فرض این که افزودن این امکان تاخیر مرحلهٔ EX را به ۲۵۰ پیکوثانیه می‌رساند، سرعت اجرای برنامه‌ها چقدر افزایش می‌یافت؟

پاسخ: در این صورت اجرای هر دستور یک مرحلهٔ ۲۵۰ پیکوثانیه‌ای طول می‌کشد.

$$speedup = \frac{1.2 \times 150}{250} = 0.72 \text{ (compared with full-forwarding)}$$

۵- (۱۵ نمره) هر چند زمانِ دسترسی (access time) در حافظه‌های نهان با دسترسی مستقیم (direct map) کمتر از حافظه‌های شبه‌انجمنی (set-associative) است، تعدادِ فقدان‌های ناشی از تضاد (conflict misses) در آنها بیشتر است. برای کاهشِ این فقدان‌ها روشی پیشنهاد شده است[1] با عنوانِ به‌کارگیری حافظهٔ نهان قربانی (Victim cache). چگونگی کارکردِ حافظهٔ نهانِ قربانی در شکلِ زیر دیده می‌شود.



در این ساختار دسترسی به هر داده طی مراحل زیر انجام می‌شود:

۱- حافظهٔ L1 بررسی می‌شود. اگر حاویِ دادهٔ موردنظر باشد، داده به پردازنده انتقال می‌یابد.

۲- اگر داده در L1 نباشد، حافظهٔ قربانی بررسی می‌شود. اگر حاویِ داده باشد، داده به L1 منتقل شده و از آنجا به پردازنده انتقال می‌یابد. اگر لازم باشد این داده روی داده‌ای قدیمی در L1 نوشته شود، دادهٔ جایگزین‌شده به حافظهٔ قربانی منتقل می‌شود، به این ترتیب که این داده در انتهای یک صف FIFO در حافظهٔ قربانی قرار می‌گیرد.

۳- اگر دادهٔ موردنظر نه در L1 باشد و نه در حافظهٔ قربانی، داده از حافظهٔ اصلی بازیابی شده و در L1 قرار می‌گیرد. این بار نیز اگر این داده به جای دادهٔ دیگری نوشته شود، دادهٔ قدیمی به انتهای صف حافظهٔ قربانی افزوده می‌شود و اگر صف پر بود، یک داده از ابتدای صف حذف می‌شود. اگر در مدتی که این داده در حافظه نبوده تغییری کرده باشد، نسخهٔ جدید آن روی حافظهٔ اصلی نوشته می‌شود.

[1] Jouppi, N. P. (1990-05-01). Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. 17th Annual International Symposium on Computer Architecture, 1990. Proceedings. pp. 364–373. doi:10.1109/ISCA.1990.134547. ISBN 0-8186-2047-1.

توجه کنید داده‌ای که روی L1 ذخیره شده، روی حافظهٔ قربانی نیست و برعکس، داده‌ای که روی حافظهٔ قربانی ذخیره شده، روی L1 نیست، به عبارت دیگر L1 و حافظهٔ قربانی هیچ دادهٔ مشترکی ندارند.

با توجه به توضیحات بالا، به سوال زیر پاسخ دهید.

فرض کنید L1 یک حافظهٔ نهان با نگاشت مستقیم با ظرفیت ۱۲۸ بایت است. حافظهٔ قربانی از نوع انجمنی با ظرفیت ۳۲ بایت است و سیاست جایگزینی آن FIFO است. اندازهٔ بلوک در هر دو حافظه ۱۶ بایت است و دسترسی به آنها به صورت کلمهٔ ۴ بایتی است. برای سادگی فرض کرده‌ایم آدرس‌ها ۸ بیتی هستند، بنابراین تقسیم‌بندی هر آدرس به صورت زیر خواهد بود.

7	6	4	3	2	1	0
TAG	INDEX		WORD SELECT		BYTE SELECT	

الف- جدول زیر را که یک ردیابی از دسترسی‌های حافظه را نشان می‌دهد، تکمیل کنید. آدرس‌ها ۸ بیتی فرض شده‌اند و مقدار 'inv' به معنای نامعتبر بودن محتوای ورودی است. برای سادگی، فقط وقتی عناصری را در جدول پر کنید که مقداری تغییر کرده است. سه خط اول جدول برای شما پر شده‌اند.

Input Address	Main Cache (tag)									Victim Cache (tag)		
	L0	L1	L2	L3	L4	L5	L6	L7	Hit?	Way0	Way1	Hit?
	inv	inv	inv	inv	inv	inv	inv	inv	-	inv	inv	-
0	0								N			N
80	1								N	0		N
4	0								N	8		Y
A0			1						N			N
10		0							N			N
C0					1				N			N
18		0							Y			
20			0						N		A	N
8C	1								N			Y
28			0						Y			
AC			1						N		2	Y
38				0					N			N
C4					1				Y			
3C				0					Y			
48					0				N	C		N
0C	0								N		8	N
24			0						N	A		N

ب) فرض کنید ۱۵٪ از دسترسی‌های حافظه در حافظهٔ قربانی حل می‌شوند. اگر بازیابی داده از حافظهٔ قربانی ۴ چرخه و بازیابی داده از حافظه اصلی ۵۰ چرخه طول بکشد، حافظهٔ قربانی به صورت میانگین، زمان دسترسی به حافظه را چند چرخه کمتر می‌کند؟ فرض کنید نرخ فقدان (miss rate) در L1 ۱۰٪ باشد.

پاسخ:

$$AMATnoVictim = hitTime + 0.1 \times 50 = hitTime + 5$$
$$AMATwithVictim = hitTime + 0.1 \times (0.15 \times 4 + 0.85 \times 50) = hitTime + 4.31 =$$
$$\boldsymbol{Diff = 0.69\ cycle}$$