دانشکده مهندسی کامپیوتر

معماری کامپیوتر

آزمون میان‌ترم

نام و نام خانوادگی:

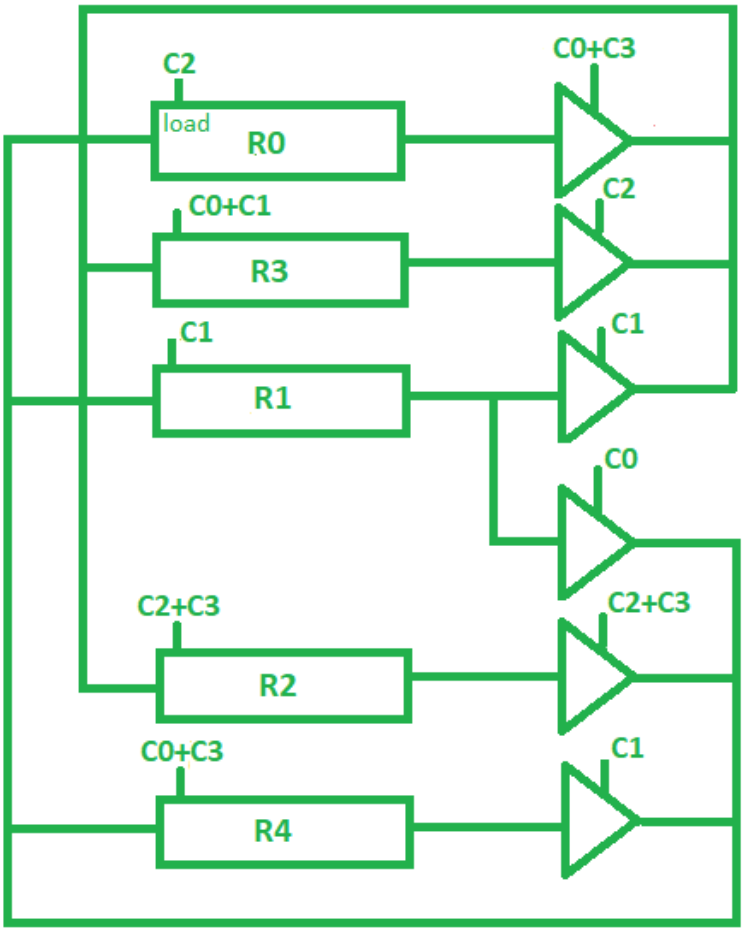
شماره دانشجویی:

آذر ۱۴۰۲ | زمان آزمون: ۹۰ دقیقه | مدرس: لاله ارشدی

۱- (۱۰ نمره) توصیف RTL زیر را به کمک بافرهای سه‌حالته پیاده‌سازی کنید.. فرض کنید ثبات‌ها n بیتی باشند و هر کدام یک ورودی load دارند. همچنین فرض کنید سیگنال‌های C0 تا C3 هرگز همزمان یک نمی‌شوند.

```
C0: R3 ← R0, R4 ← R1
C1: R3 ← R1, R1 ← R4
C2: R2 ← R3, R0 ← R2
C3: R2 ← R0, R4 ← R2
```

پاسخ:



۲- (۸ نمره) یک برنامهٔ کامپیوتری در اختیار داریم که ۹۰٪ زمانِ اجرای آن قابلِ موازی‌سازی است.

الف- با این فرض که در صورتِ اجرای موازی، برنامه روی همهٔ هسته‌ها با سرعت یکسان اجرا می‌شود و هیچ سربارِی ایجاد نمی‌شود، مشخص کنید که تسریع (speedup) حاصل از به‌کارگیری n هسته چقدر خواهد بود؟

ب- سوالِ قبل را با این فرض پاسخ دهید که موازی‌سازی بر روی n هسته $0.001 \times n \times T_0$ سربار اضافه کند. (T_0 زمان اجرای برنامه پیش از موازی‌سازی است) در این صورت n چند باشد که تسریع به‌دست آمده ماگزیمم باشد؟ تسریع ماگزیمم چند است؟

پاسخ:

الف- زمان اجرای ۱۰٪ برنامه هیچ تغییری نمی‌کند و زمان اجرای بقیهٔ برنامه بر n تقسیم می‌شود، بنابراین:

$$speedup_a = \frac{T_0}{0.1T_0 + 0.9T_0/n} = \frac{n}{0.1n + 0.9}$$

ب- در این صورت، به اندازهٔ سربار گفته‌شده به زمان اجرای برنامه روی n هسته افزوده می‌شود، بنابراین:

$$speedup_b = \frac{T_0}{0.1T_0 + 0.9T_0/n + 0.001nT_0} = \frac{n}{0.1n + 0.9 + 0.001\mathrm{n}^2}$$

برای این‌که تسریع بیشینه باشد، باید نسبت به n از آن مشتق گرفته و برابر با صفر قرار دهیم:

$$0.1n + 0.9 + 0.001n^2 - n(0.1 + 0.002n) = 0 \quad \Longrightarrow \quad 0.9 = 0.001n^2 \quad \Rightarrow \quad n = 30$$

$$speedup_{b\,\max} = \frac{30}{3 + 0.9 + 0.9} = 6.25$$

بارم‌بندی:

هر پاسخ ۲ نمره

۳- (۷ نمره) در یک پردازنده اجرای دستورات صحیح، ممیز شناور، و باقی دستورات به ترتیب ۲، ۵ و ۴ چرخه طول می‌کشد. می‌دانیم ۵۰٪ دستورات از نوع اعداد صحیح و ۳۰٪ از نوع ممیز شناور است. می‌توانیم با اعمال تغییراتی، CPI دستورات صحیح و ممیز شناور را به ترتیب به ۱ و ۳ برسانیم، اما ناچار خواهیم بود فرکانس clock را کاهش دهیم. حساب کنید فرکانس clock باید حداقل چه کسری از فرکانس اولیه باشد که این تغییرات موجب افزایش کارایی شود؟

پاسخ:

برای پاسخ به این سوال، ابتدا زمان اجرای هر دو حالت را به دست می‌آوریم و آنها را با هم برابر قرار می‌دهیم تا نسبت فرکانس clockها به دست آید.

$$ExecTime_1 = \frac{0.5 \times 2 + 0.3 \times 5 + 0.2 \times 4}{CR_1} = 3.3/CR_1$$

$$ExecTime_2 = \frac{0.5 \times 1 + 0.3 \times 3 + 0.2 \times 4}{CR_2} = 2.2/CR_2$$

$$ExecTime_1 = ExecTime_2 \quad \Rightarrow \quad \frac{3.3}{CR_1} = \frac{2.2}{CR_2} \quad \Rightarrow \quad CR_2 = \frac{2}{3}CR_1$$

بارم‌بندی:

معادلات اول و دوم هر کدام ۳ نمره، پاسخ نهایی یک نمره

۴- (۱۵ نمره) یک جمع‌کنندهٔ چهاربیتی از نوع CLA (Carry-Look-Ahead) در نظر بگیرید. فرض می‌کنیم این جمع‌کننده دو عدد a_0-a_3 و b_0-b_3 را با هم جمع می‌کند. بیت نقلی ورودی را c_0 می‌نامیم.

الف- رابطهٔ بیت نقلی c_{i+1} را برحسب p_i، g_i و c_i بنویسید. (p_i و g_i بیت‌های propagate و generate هستند.)

ب- رابطهٔ بیت نقلی خروجی c_4 را برحسب c_0، p_0-p_3 و g_0-g_3 بنویسید.

ج- چهار جمع‌کنندهٔ چهار بیتی CLA را در کنار هم قرار می‌دهیم تا یک جمع‌کنندهٔ ۱۶ بیتی بسازیم. بیت‌های نقلی ورودی و خروجی هر واحد چهار بیتی را به ترتیب Cj و Cj+1 می‌نامیم. با توجه به رابطه‌ای که در بند ب به‌دست آوردید، Pj و Gj را طوری تعریف کنید که بتوانیم Cj+1 را برحسب Cj، Pj و Gj بنویسیم.

اگر تاخیر گیت‌های AND، OR و XOR (با هر تعداد ورودی) به ترتیب d، d و 2d باشد، به سوالات زیر پاسخ دهید:

د- نتیجهٔ یک جمع چهاربیتی با استفاده از یک جمع‌کنندهٔ CLA بعد از چه مدت آماده می‌شود؟ چرا؟

ه- نتیجهٔ یک جمع‌کنندهٔ ۱۶ بیتی با استفاده از دو لایه جمع CLA بعد از چه مدت آماده می‌شود؟ چرا؟

و- نتیجهٔ یک جمع چهار بیتی با جمع‌کنندهٔ عادی (Ripple-Carry) بعد از چه مدت آماده می‌شود؟ چرا؟

ز- نتیجهٔ یک جمع ۱۶ بیتی با جمع‌کنندهٔ عادی (Ripple-Carry) بعد از چه مدت آماده می‌شود؟ چرا؟

ط- اگر برای جمع ۱۶ بیت، از چهار جمع‌کنندهٔ چهاربیتی CLA استفاده کنیم که به صورت ripple به دنبال هم بسته شده‌اند استفاده کنیم، نتیجه نهایی بعد از چه مدت آماده می‌شود؟ چرا؟

الف / $c_{i+1} = g_i + p_i c_i$ ۰/۱ نمره ۳

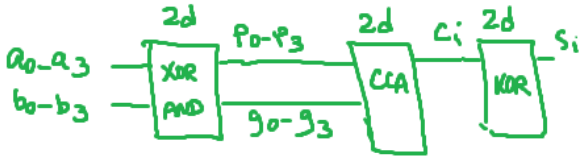
ب / $c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$

ج / $c_{j+1} = G_j + P_j c_j$ نمره ۳

$P_j = p_3 p_2 p_1 p_0$

$G_j = g_3 + \cdots + p_3 p_2 p_1 g_1$

د / 6d نمره ۲



۰/۲

ه / a_0-a_3, b_0-b_3 —XOR, AND (2d)→ p_0-p_3, g_0-g_3 —AND-OR (2d)→ P_j, G_j —AND-OR (2d)→ C_j —CLA (2d)→ c_i —XOR (2d)→ s_i 10d

و / ripple-carry 4bit : 8d نمره ۱

ز / ripple-carry 16bit : 32d نمره ۱

ط / بعد از 4d بیت نقلی جمع کننده ۴ بیتی اول ظاهر شود، در این فاصله بیت‌های p_i و g_i بقیه جمع کننده هم ظاهر شده. بنابراین بعد از 2d بیت نقلی جمع کننده دوم هم ظاهر می‌شود (C_2) و بعد از 2d دیگر هم C_3 ظاهر می‌شود. در جمع کننده چهارم باید 4d صبر کنیم تا نتیجه ظاهر شود، پس نتیجه کل بعد از $4d + 2d + 2d + 4d = 12d$ ظاهر می‌شود نمره ۲

۵- (۵ نمره) در استاندارد IEEE754 برای نمایش اعداد ممیز شناور ۶۴ بیتی (دقت مضاعف) طول بخش‌های علامت (Sign)، نما (Exponent) و کسری (Fraction) آن به ترتیب یک، ۱۱ و ۵۲ بیت است.

الف- کوچک‌ترین و بزرگ‌ترین عددِ مثبتِ نرمالِ قابلِ نمایش در این ساختار چند است؟

ب- کوچک‌ترین و بزرگ‌ترین عددِ مثبتِ غیرنرمالِ قابلِ نمایش در این ساختار چند است؟

پاسخ:

الف- چون بخش نما ۱۱ بیت دارد، مقدار نما بین ۱۰۲۳- و ۱۰۲۴+ است و البته مقادیر بیشینه و کمینهٔ گفته‌شده را اتخاذ نمی‌کند. بنابراین کمترین نما برابر با ۱۰۲۲- و بیشترین نما برابر با ۱۰۲۳ است. کمترین و بیشترین اعداد نرمال قابل نمایش عبارتند از:

$$max_{normal} = 1.11\ldots \times 2^{1023} = (2 - 2^{-52}) \times 2^{1023} = 2^{1024} - 2^{971}$$

$$min_{normal} = 1.0 \times 2^{-1022} = 2^{-1022}$$

ب- در حالت غیرنرمال نما همیشه ۱۰۲۲- است و مقدار کسری بیشینه و کمینه را تعینن می‌کند:

$$max_{nonnormal} = 0.11\ldots \times 2^{-1022} = (1 - 2^{-52}) \times 2^{-1022} = 2^{-1022} - 2^{-1074}$$

$$min_{nonnormal} = 0.0\ldots1 \times 2^{-1022} = 2^{-52} \times 2^{-1022} = 2^{-1074}$$

بارم‌بندی:

هر پاسخ نادرست کسر یک نمره، نمای نادرست در حالت غیرنرمال کسر یک نمره

۶- (۱۵ نمره) بلوک دیاگرامِ مسیر داده و کنترل پردازندهٔ ساده‌شدهٔ MIPS را در **Error! Reference source not found.** می‌بینید. عملیات کنترل در این شکل توسط تعدادی سیگنال کنترلی انجام می‌شود که در جداول ۱ و ۲ آمده است. می‌خواهیم دو دستورِ زیر را به مجموعهٔ دستوراتِ این پردازنده اضافه کنیم:

```
andm (rd),rs,rt          ; Mem[rd] ← rs AND rt
```

100000	rs	rt	rd	0	100100

```
andmi (rt),rs,cnst       ; Mem[rt] ← rs AND zero_extend(cnst)
```

011100	rs	rt	16 bits cnst

چه تغییراتی باید در شکل و جداول بدهیم؟ مسیر دادهٔ اجرای این دستور را برمبنای شکل توضیح دهید و مشخص کنید برای اجرای این دستور، مقادیر هر یک از سیگنال‌های کنترلی زیر چه باید باشد؟

```
RegDst, Branch, MemRead, MemtoReg, MemWrite, ALUSrc, RegWrite, ALUOp
```

(به خاطر داشته باشید که در دستوراتِ R-type جدول۱ شش بیتِ پرارزشِ دستور همه صفر هستند.)

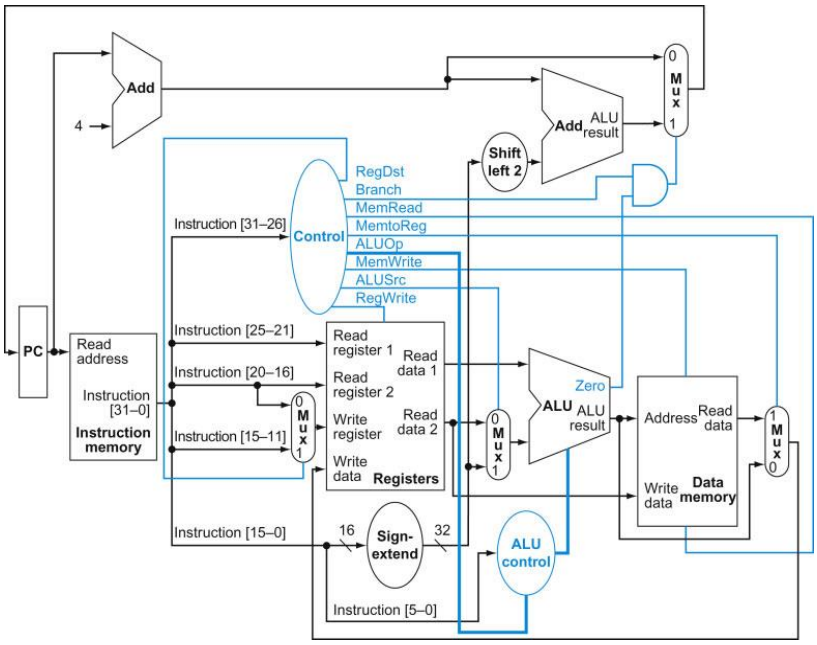
پاسخ:

برای اجرای دستور andm باید محتویات دو ثباتی را که شمارهٔ آنها در بیت‌های ۲۵-۲۱ و ۲۰-۱۶ قرار دارند با هم and کنیم و در خانه‌ای از حافظه بنویسیم که ثبات شمارهٔ ۱۵-۱۱ مشخص می‌کند. فایل ثبات‌های ما قابلیت خواندن همزمان دو ثبات را دارد، بنابراین باید امکان خواندن ثبات سوم را هم به آن اضافه کنیم. برای این که کمترین تغییرات را بدهیم می‌توانیم فایل ثبات را این طور تغییر دهیم که اگر ورودی RegWrite=1 باشد، فایل ثبات مثل قبل عمل کند ولی اگر

RegWrite=0، آنگاه محتوای ثباتی که توسط بیت‌های WriteRegister مشخص می‌شود، در خروجی قابل خواندن باشد، (مثلا با عنوان Readdata3). همین بیت‌ها را می‌توانیم برای آدرس‌دهی حافظه استفاده کنیم، به این ترتیب که در ورودی حافظه یک مالتی‌پلکسر قرار دهم که آدرس را یا از خروجی ALU بگیرد، یا از Readdata3 و سیگنال کنترلی MemAddrSrc را هم برای کنترل آن درنظربگیریم. ورودی دادهٔ حافظه هم در شکل از خروجی فایل ثبات می‌آید، در حالی که در این دستور باید از خروجی ALU بیاید، بنابراین در ورودی دادهٔ حافظه هم به یک مالتی‌پلکسر دیگر نیاز داریم که با یک سیگنال کنترلی (مثلا به نام MemData) مشخص کند که داده از ثبات گرفته شود یا از خروجی ALU. اجرای دستور andmi تفاوت چندانی ندارد، جز موارد زیر. اول این که ثبات سوم فایل ثبات باید Rt باشد (بیت‌های ۲۰-۱۶)، بنابراین باید RegDst=0. دوم این که محتویات Rs باید با مقدار ثابت AND شود، پس باید ALUSrc=1. سوم این که مقدار ثابت باید zero-extend شود، در حالی که در شکل، این مقدار sign-extend می‌شود. بنابراین باید یک مالتی‌پلکسر هم پیش از مالتی‌پلکسر متصل به ورودی دوم ALU قرار بدهیم که بین sign-extend و zero-extend انتخاب کند. ورودی کنترل این مالتی‌پلکسر را هم SignExt می‌نامیم که همیشه یک است، مگر در دستور andmi. و مورد آخر این که باید یک کد جدید برای ALUOp در نظر بگیریم (مثلا ۱۱ که استفاده نشده است) و در آن به ALU فرمان AND کردن بدهیم. البته می‌توانیم بیت‌های op را هم به ALU بدهیم که اگر کد ورودی ۱۱ بود، خود ALU از روی این بیت‌ها تصمیم بگیرد چه عملی انجام بدهد و این کد را برای همه دستورات immediate استفاده کنیم.

مقدار سیگنال‌های کنترلی برای اجرای هر کدام از دو دستور عبارتند از:

andm: RegDst=1, Branch=0, MemRead=0, MemtoReg=0, MemWrite=1, ALUSrc=0, RegWrite=0, ALUOp=10
andmi: RegDst=0, Branch=0, MemRead=0, MemtoReg=0, MemWrite=1, ALUSrc=1, RegWrite=0, ALUOp=11



شکل ۱- بلوک دیاگرام مسیر داده و کنترل پردازنده ساده MIPS

جدول ۱- شرح ارتباط سیگنال‌های واحد ALU Control در **Error! Reference source not found.**

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

جدول ۲- شرح ارتباط سیگنال‌های واحد Control در **Error! Reference source not found.**

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

موفق باشید