معماری کامپیوتری

نیم‌سال دوم ۰۳-۰۲

دانشکده‌ی مهندسی کامپیوتر
استاد: دکتر لاله ارشدی
تمرین تئوری چهارم
پاسخ‌دهنده: معین آعلی - ۴۰۱۱۰۵۵۶۱

پاسخ مسئله‌ی ۱.

$a : \text{۰}xAC\text{۶۲۰۰۱۴}$

$b : \text{۰}x\text{۰۰۸۲۰۸۲}A$

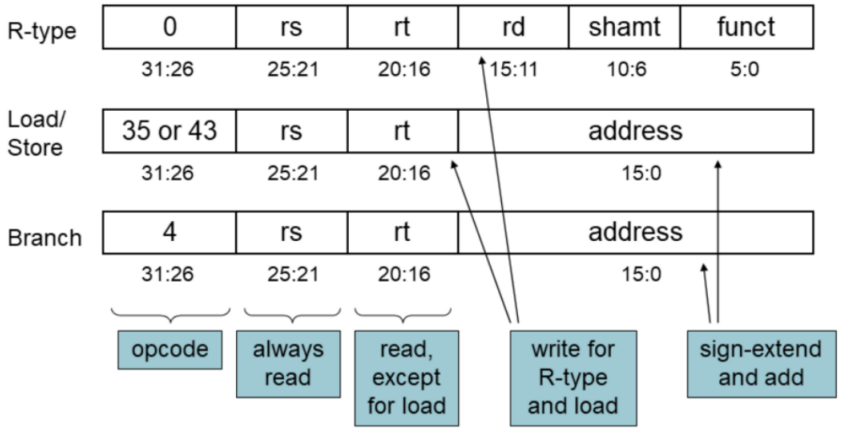
الف

در این باید کد اسمبلی معادل با هر دستور را بنویسیم. برای این کار ابتدا اعداد را به باینری تبدیل می‌کنیم:

$a = (\text{۱۰۱۰۱۱۰۰۰۱۱۰۰۰۱۰۰۰۰۰۰۰۰۰۰۱۰۱۰۰})_{\text{۲}}$

$b = (\text{۰۰۰۰۰۰۰۰۱۰۰۰۰۰۱۰۰۰۰۰۱۰۰۰۰۰۱۰۱۰۱۰})_{\text{۲}}$

حال از این تصویر استفاده کرده و دستورات را شناسایی می‌کنیم:



$opcode_a = a[\text{۳۱} : \text{۲۶}] = (\text{۱۰۱۰۱۱})_{\text{۲}} = (\text{۴۳})_{\text{۱۰}} \longrightarrow store$

$$\begin{cases} r_s = a[\text{۲۵} : \text{۲۱}] = (\text{۰۰۰۱۱})_{\text{۲}} = (\text{۳})_{\text{۱۰}} \longrightarrow v_{\text{۱}} \\ r_t = a[\text{۲۰} : \text{۱۶}] = (\text{۰۰۰۱۰})_{\text{۲}} = (\text{۲})_{\text{۱۰}} \longrightarrow v_{\text{۰}} \\ immediate = a[\text{۱۵} : \text{۰}] = (\text{۰۰۰۰۰۰۰۰۰۰۰۱۰۱۰۰})_{\text{۲}} = (\text{۲۰})_{\text{۱۰}} \end{cases} \longrightarrow sw \ \ \$v_{\text{۰}}, \text{۲۰}(\$v_{\text{۱}})$$

$opcode_b = b[\text{۳۱} : \text{۲۶}] = (\text{۰۰۰۰۰۰})_{\text{۲}} = (\text{۰})_{\text{۱۰}} \longrightarrow R - Type$

$$\begin{cases} r_s = b[\text{۲۵} : \text{۲۱}] = (\text{۰۰۱۰۰})_{\text{۲}} = (\text{۴})_{\text{۱۰}} \longrightarrow a_{\text{۰}} \\ r_t = b[\text{۲۰} : \text{۱۶}] = (\text{۰۰۰۱۰})_{\text{۲}} = (\text{۲})_{\text{۱۰}} \longrightarrow v_{\text{۰}} \\ r_d = b[\text{۱۵} : \text{۱۱}] = (\text{۰۰۰۰۱})_{\text{۲}} = (\text{۱})_{\text{۱۰}} \longrightarrow a_t \\ function = b[\text{۵} : \text{۰}] = (\text{۱۰۱۰۱۰})_{\text{۲}} \longrightarrow set\ on\ less\ than \end{cases} \longrightarrow slt \ \ \$a_t \ , \ \$a_{\text{۰}} \ , \ \$v_{\text{۰}}$$

ب

خروجی sign extend دستور a ، ۱۶ بیت اول دستور را به یک عدد ۳۲ بیتی تبدیل می‌کند با حفظ علامت:

$$output = (\text{۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۱۰۱۰۰})_{۲} = (۱۴)_{۱۶}$$

خروجی واحد shift left که توسط دستور b اجرای می‌شود، خروجی واحد sign extend را می‌گیرد و ۲ بیت به چپ شیفت می‌دهد.

$$output = (\text{۰۱۰۰۰۰۰۱۰۰۰۰۰۱۰۰۰۰۰۱۰۱۰۱۰۰۰})_{۲} = (۲۰۸۲۰A۸)_{۱۶}$$

ج

در این بخش باید مقادیر واحد کنترل ALU را به ازای هر دستور محسابه کنیم. از تصویر زیر کمک می‌گیریم:

opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

$$a \rightarrow lw \rightarrow \begin{cases} ALUOp = ۰۰ \\ funct = a[۵:۰] = \times \end{cases}$$

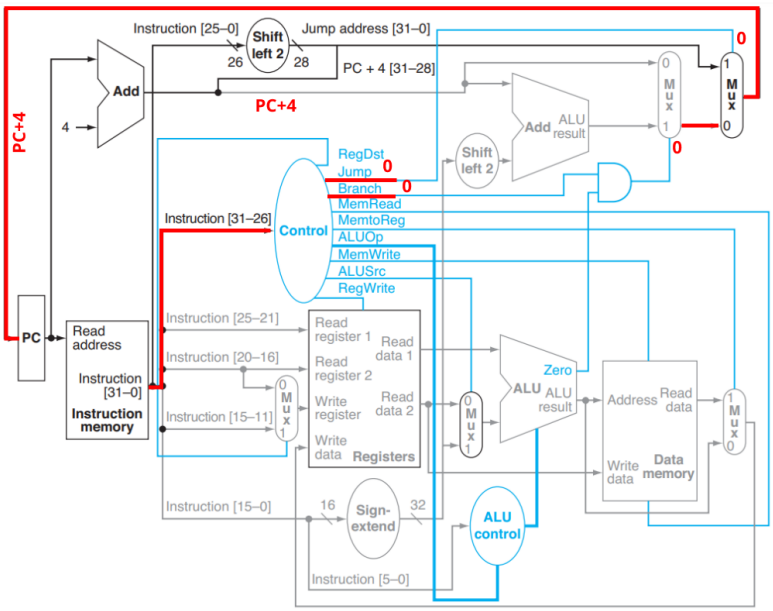
$$b \rightarrow RType \rightarrow \begin{cases} ALUOp = ۱۰ \\ funct = b[۵:۰] = ۱۰۱۰۱۰ \end{cases}$$

د

با توجه به این‌که هیچ کدام از دستورات مربوط به دستورات jump و یا branch نیستند، پس در هر دو حالت مقدار برابر است با:

$$PC \leftarrow PC + ۴$$

در شکل زیر مسیرهایی که از آن‌ها این مقادیر به‌دست می‌آیند را می‌بینید:

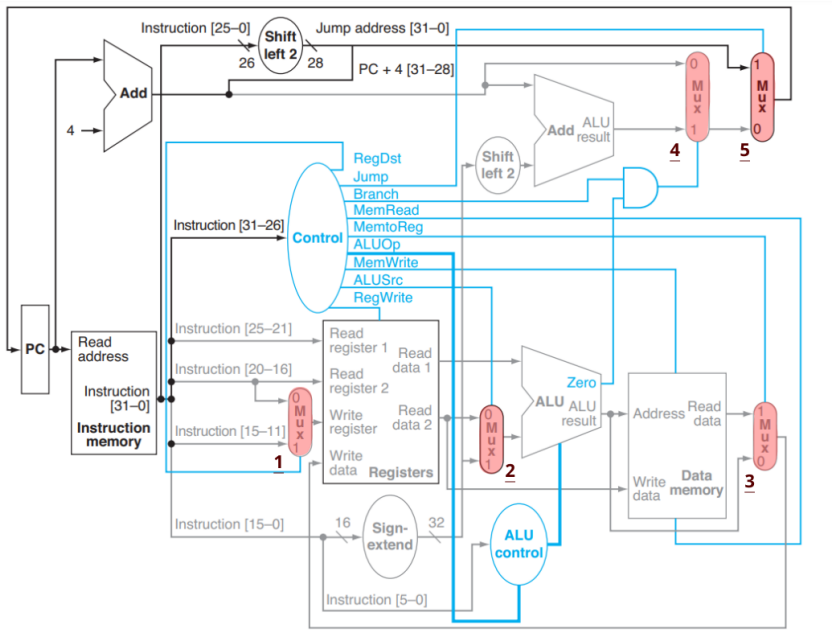


مقدار موجود در هر یک از ثبات‌ها به شرح زیر است:

	R0	R1	R2	R3	R4	R5	R6	R8	R12	R31
a	0	-1	2	-3	-4	10	6	8	2	-16
b	0	256	-128	19	-32	13	-6	-1	16	-2

۵

در این بخش باید مقادیر خروجی هر یک از MUX ها را مشخص کنیم. ابتدا به سراغ دستور a می‌رویم:



۱. چون دستور ما sw است و RegDst برابر صفر است، پس خروجی این مالتیپلکسر برابر است با:

$$MUX_{۱} = Instruction[۲۰ : ۱۶] = (۰۰۰۱۰)_{۲} = (۲)_{۱۰} \rightarrow \$R۲$$

۲. چون سیم مربوط به ALUSrc برابر یک است، پس حاصل واحد Sign Extend مقدار خروجی آن است:

$$MUX_{۲} = SignExtend(Instruction[۱۵ : ۰]) = (۱۴)_{۱۶} = (۲۰)_{۱۰} \rightarrow \$R۲۰$$

۳. خروجی این مالتیپلکسر برای ما فاقد اهمیت است.

$$MUX_{۳} = \times$$

۴. سیم مربوط به Branch صفر است، پس خروجی این مالتیپلکسر برابر است با:

$$MUX_{۴} = PC + ۴$$

۵. چون این دستور مربوط به Jump نیست و سیم مربوط به آن صفر است، پس خروجی ما برابر است با:

$$MUX_{۵} = PC + ۴$$

حال به سراغ دستور b می‌رویم:

۱. مقدار سیم مربوط به RegDst برابر یک است، پس rd خروجی داده می‌شود:

$$MUX_{۱} = Instruction[۱۵ : ۱۱] = (۰۰۰۰۱)_{۲} = (۱)_{۱۰}$$

۲. چون سیم مربوط به ALUSrc برابر صفر است، مقدار ReadData۲ خروجی داده می‌شود:

$$MUX_{۲} = Instruction[۲۰ : ۱۶] = \$rt = \$R۲$$

۳. چون سیم مربوط به MemtoReg قطع است، پس خروجی این مالتیپلکسر برابر است با:

$MUX_{۳} = ALUResult = slt\ \ \$a_t\ ,\ \$a_۰\ ,\ \$v_۰ = ۰$

۴. بیت کنترلی Branch صفر چون دستور ما مربوط به Branch یا Jump نیست، پس خروجی برابر است با:

$MUX_{۴} = PC + ۴$

۵. به همان دلیلی که در مورد قبل گفته شد، خروجی برابر است با:

$MUX_{۵} = PC + ۴$

و

در این بخش باید ورودی‌های ALU و واحدهای Add موجود در این معماری را محاسبه کنیم. ابتدا به سراغ دستور a می‌رویم:

۱. ورودی‌های واحد ALU معماری:

$$\begin{cases} Read۱ = Instruction[۲۵ - ۲۱] = \$rs = \$R_{۳} = (-۳)_{۱۰} \\ Read۲ = Instruction[۲۰ - ۱۶] = address = (۲۰)_{۱۰} \end{cases}$$

۲. ورودی‌های واحد Add سمت چپ معماری:

$$\begin{cases} input۱ = PC \\ input۲ = ۴ \end{cases}$$

۳. ورودی‌های واحد Add سمت راست معماری:

$$\begin{cases} input۱ = PC + ۴ \\ input۲ = ۴ \times Address = ۸۰ \end{cases}$$

حال به سراغ دستور b می‌رویم:

۱. ورودی‌های واحد ALU معماری:

$$\begin{cases} Read۱ = Instruction[۲۵ - ۲۱] = \$rs = \$R_۲ = -۱۲۸ \\ Read۲ = Instruction[۲۰ - ۱۶] = \$rt = \$R_۴ = -۳۲ \end{cases}$$

۲. ورودی‌های واحد Add سمت چپ معماری:

$$\begin{cases} input۱ = PC \\ input۲ = ۴ \end{cases}$$

۳. ورودی‌های واحد Add سمت راست معماری:

$$\begin{cases} input۱ = PC + ۴ \\ input۲ = ۴ \times Address = ۸۳۶۰ \end{cases}$$

ز

در این بخش باید مقادیر مختلف Register File را به ازای هر دو دستور مشخص کنیم. به ازای دستور a مقادیر این ثبات‌ها برابر هستند با:

- ReadRegister۱ = ۳
- ReadRegister۲ = ۲
- WriteRegister = ۲
- WriteData = ×
- RegWrite = ۰

حال به سراغ دستور b می‌رویم:

- ReadRegister۱ = ۴
- ReadRegister۲ = ۲
- WriteRegister = ۱
- WriteData = ۰
- RegWrite = ۱

پاسخ مسئله‌ی ۲.

الف

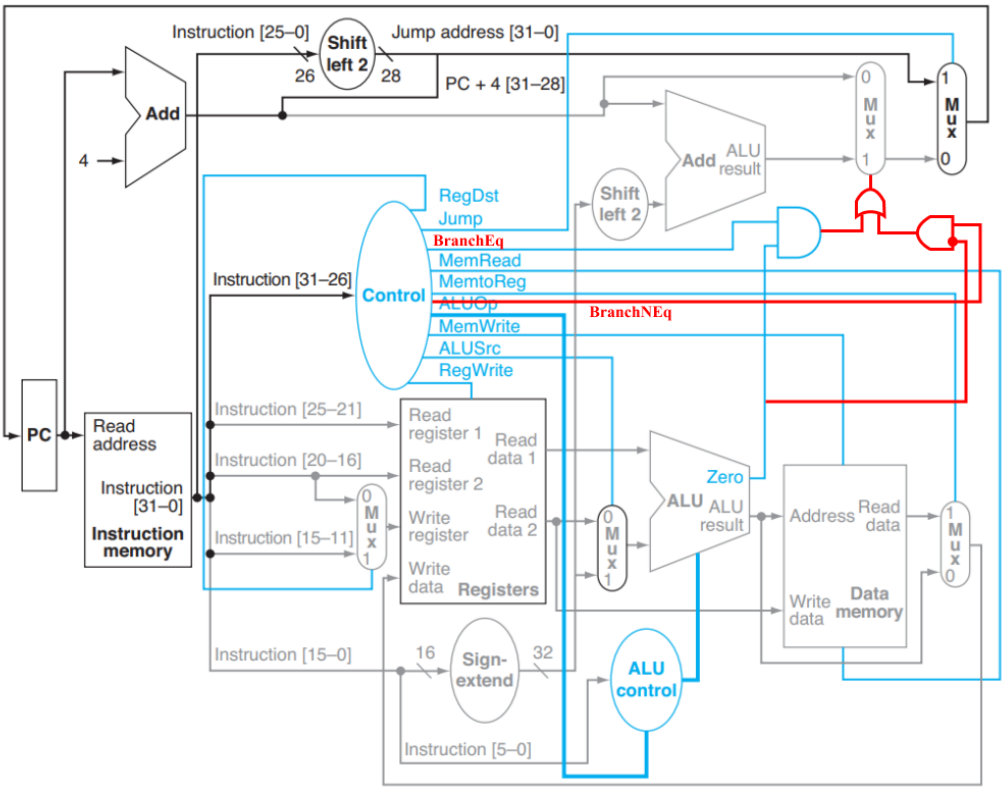
این دستور I Type و از نوع Branch است. برای اضافه کردن این دستور به معماری خود، کافیست ALUOp آن را با دستور Branch برابر قرار دهیم و مقدار Funct های آن‌ها را متفاوت در نظر بگیریم.

ب

جداول و معماری فوق به این صورت تغییر خواهند کرد:

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
Branch equal	**01**	**Branch equal**	**xxxxx0**	**subtract**	**0110**
forr	**01**	**Branch on not equal and subtract**	**xxxxx1**	**subtract**	**0110**

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1
forr	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**



پاسخ مسئله‌ی ۳.

در این سوال باید توضیح دهیم که با گیر کردن سیگنال، اجرای کدام دستورات دچار مشکل خواهند شد.

الف

اگر بیت RegDst روی صفر گیر کند، آنگاه رجیستر مقصد هم روی rt گیر می‌کند. در این حالت فقط دستوراتی دچار مشکل می‌شوند که نیاز دارند چیزی را روی rd بنویسند، پس فقط **R Type** دچار مشکل می‌شوند.

ب

اگر بیت صفرم ALUOp روی مقدار یک گیر کند، دستورات **lw** و **sw** و **R Type** که در آن‌ها بیت صفرم ALUOp برابر صفر است دچار مشکل می‌شوند.

البته اگر در دستورات R Type بیت صفرم ALUOp را روی حالت Dont Care در نظر گرفته باشیم، برای این عملیات‌ها مشکلی به وجود نخواهد آمد، زیرا مقدار ۱۱ و ۱۰ برای ما فرقی ندارد.

ج

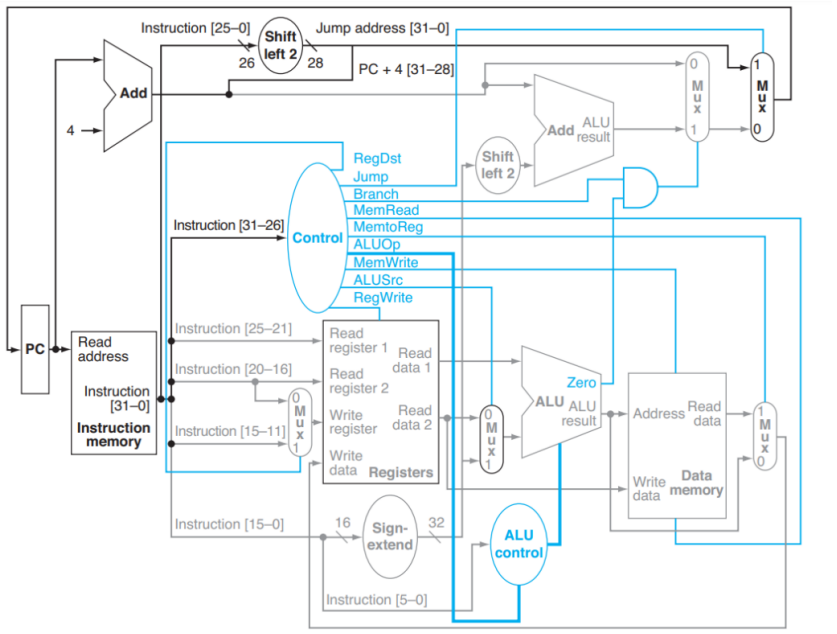
اگر سیگنال RegWrite روی صفر گیر کند، آنگاه روی هیچ رجیستری چیزی نوشته نمی‌شود و درنتیجه دستوراتی مانند **R Type** و **lw** به مشکل می‌خورند. به عنوان مثال در دستورات R Type باید نتیجه عملیات را روی یک رجیستر مقصد نوشت، که این کار انجام نخواهد شد، و یا در دستور lw باید مقداری از حافظه بخوانیم و در یک رجیستر بنویسیم که این کار انجام نخواهد شد.

همچنین در بعضی از دستورات Immediate که نوشتن روی رجیستر و یا حافظه داریم، به مشکل خواهیم خورد.

پاسخ مسئله‌ی ۴.

تاخیرهای زیر را برای بخش‌های مختلف شکل زیر در نظر می‌گیریم:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Control
220ps	75ps	20ps	95ps	70ps	240ps	45ps



ابتدا تاخیرهای هر بخش را محاسبه می‌کنیم:

- پرش غیر شرطی:

$$\max(IMem, Add) + MUX + Control = ۲۲۰ + ۲۰ + ۴۵ = ۲۸۵^{ps}$$

- پرش شرطی:

$$\max(IMem, Add) + \max(Control, Regs) + MUX + \max(ALU, Add) + ۲ \times MUX = ۲۲۰ + ۷۰ + ۲۰ + ۹۵ + ۲ \times ۴۰ = ۴۴۵^{ps}$$

- Load:

$$\max(IMem, Add) + \max(Control, Regs) + MUX + ALU + DMem + MUX + Regs = ۲۲۰ + ۷۰ + ۲۰ + ۹۵ + ۲۴۰ + ۲۰ + ۷۰ = ۷۳۵^{ps}$$

- Store:

$$\max(IMem, Add) + \max(Control, Regs) + MUX + ALU + DMem = ۲۲۰ + ۷۰ + ۲۰ + ۹۵ + ۲۴۰ = ۶۴۵^{ps}$$

- RType:

$$\max(IMem, Add) + \max(Control, Regs) + MUX + ALU + MUX + Regs = ۲۲۰ + ۷۰ + ۲۰ + ۹۵ + ۲۰ + ۷۰ = ۴۹۵^{ps}$$

$$T_۱ = \frac{۸}{۱۰۰} \times ۲۸۵ + \frac{۱۱}{۱۰۰} \times ۴۴۵ + \frac{۱۶}{۱۰۰} \times ۷۳۵ + \frac{۱۵}{۱۰۰} \times ۶۴۵ + \frac{۵۰}{۱۰۰} \times ۵۱۵ = ۵۴۳/۶^{ps}$$

$$T_۲ = \frac{۸}{۱۰۰} \times ۲۸۵ + \frac{۱۱}{۱۰۰} \times ۴۳۵ + \frac{۱۶}{۱۰۰} \times ۶۶۵ + \frac{۱۵}{۱۰۰} \times ۵۷۵ + \frac{۵۰}{۱۰۰} \times ۵۰۵ = ۵۱۵/۸^{ps}$$

$$SpeedUP = \frac{T_۱}{T_۲} = \frac{۵۴۳/۶}{۵۱۵/۸} \approx ۱/۰۵$$

پس ما ۵% افزایش سرعت داشته‌ایم!