فصل ۵

طراحی یک پردازنده ساده

۵-۱- مقدمه

در فصل دوم دیدم که کارایی یک پردازنده به سه عامل کلیدی وابسته بود: تعداد دستورات، پریود کلاک و تعداد کلاک متوسط لازم برای اجرای هر دستور یا همان CPI. در فصل سوم و چهارم همان طور که دیدیم کامپایلر و معماری مجموعه دستورات (ISA)، تعداد دستورات یک برنامه را تعیین می‌کنند. اما پریود کلاک و تعداد کلاک لازم برای اجرای هر دستور توسط سخت‌افزار تعیین می‌شود. در این فصل یک پردازنده ساده برای مجموعه دستورات پردازنده MIPS طراحی خواهیم نمود. پردازنده‌ای که ما طراحی می‌کنیم، تعداد محدودی از دستورات را پیاده سازی خواهد نمود و شامل همهٔ دستورات پردازنده MIPS نخواهد بود. به عبارتی پردازنده طراحی شده یک زیر مجموعه[1] از دستورات پردازنده MIPS را پشتیبانی خواهد کرد. دستوراتی که پشتیبانی خواهد شد به شرح زیر است:

- دستورات مراجعه به حافظه مشتمل بر lw و sw
- دستورات محاسباتی و منطقی مشتمل بر add، sub، and، or و slt
- دستورات پرش شرطی و غیر شرطی مشستمل بر beq و j یا همان jump

در زیر مجموعه انتخاب شده سایر دستورات اعداد صحیح مانند دستورات ضرب و تقسیم و یا دستورات ممیز شناور وجود ندارند و پردازنده‌ای که در این فصل طراحی می‌کنیم آنها را پشتیبانی نخواهد نمود. با وجود این، اصول اولیه و کلیدی طراحی یک پردازنده در این فصل توضیح داده خواهد شد و می‌توان دستورات دیگر را به روش مشابه، به این مجموعه اضافه و پیاده سازی نمود. به هنگام پیاده سازی ما این شانس را خواهیم داشت که ببینیم چگونه معماری مجموعه دستورات جنبه-های مختلف طراحی سخت‌افزار را تحت تأثیر قرار می‌دهد و همچنین انتخاب استراتژی‌های مختلف پیاده‌سازی چگونه پریود کلاک و CPI یک ماشین را تعیین می‌کند. بسیاری از اصول کلیدی طراحی که در فصل ۳ مطرح شدند همانند اصول «موارد پر استفاده را سریعتر کن» و «منظم‌تر بودن ساده‌تر می-کند» را در این فصل به هنگام طراحی سخت‌افزار لمس خواهیم نمود. علاوه بر این مباحثی که در این فصل مطرح خواهد شد همانند ایده‌هایی است که امروزه در طراحی انواع مختلف پردازنده استفاده می‌شود.

---

[1] - Sub set

۵-۱-۱- مروری بر پیاده‌سازی

در فصل‌های ۳ و۴ ما دستورات مختلف پردازنده MIPS، مشتمل بر دستورات محاسباتی و منطقی، دستورات مراجعه به حافظه و دستورات پرش را بررسی کردیم. برای پیاده‌سازی سخت‌افزاری این دستورات، اکثر کارهایی که لازم است انجام گیرد مشابه هم بوده و مستقل ازکلاس واقعی دستورات می‌باشد. برای همهٔ دستورات دو مرحله اول یکسان است:

۱. فرستادن شمارنده برنامه (PC) به حافظه دستورات که برنامه را نگهداری می‌کند و واکشی[1] (خواندن) دستور از حافظه.

۲. خواندن محتوای یک یا دو رجیستر با استفاده از فیلدهای موجود در دستور. به کمک این فیلدها می‌توانیم عمل انتخاب بین رجیسترها را انجام دهیم و تصمیم بگیریم که کدام یک از آنها باید خوانده شوند. برای دستور lw ما نیاز داریم که فقط یک رجیستر را بخوانیم ولی برای اکثر دستورات دیگر دو رجیستر لازم است.

بعد از این دو مرحله، کاری که لازم است انجام شود تا اینکه دستور اجرا شود، به نوع دستور و به کلاس آن بستگی دارد. خوشبختانه برای هر کدام از سه کلاس دستورات (مراجعه به حافظه، دستورات محاسباتی و منطقی و پرش‌ها)، کاری که لازم است انجام شود یکی است و بستگی به دستورات داخل آن کلاس ندارد.

حتی با وجود تفاوتهایی که بین کلاس‌های مختلف دستورات وجود دارد، برخی تشابهات هم بین آنها وجود دارد. به طور مثال همهٔ کلاس‌های دستورات بعد از خواندن محتوای رجیسترها، از واحد ALU استفاده می‌کنند. دستورات مراجعه به حافظه از ALU برای محاسبه آدرس حافظه، دستورات محاسباتی و منطقی برای انجام عملیات و دستورات پرش برای انجام عمل مقایسه استفاده می‌کنند. همان طور که می‌بینیم، سادگی و منظم بودن معماری مجموعه دستورات، باعث می‌شود تعداد زیادی دستور همانند هم اجرا شوند و این باعث می‌شود طراحی سخت‌افزار نیز ساده‌تر شود.

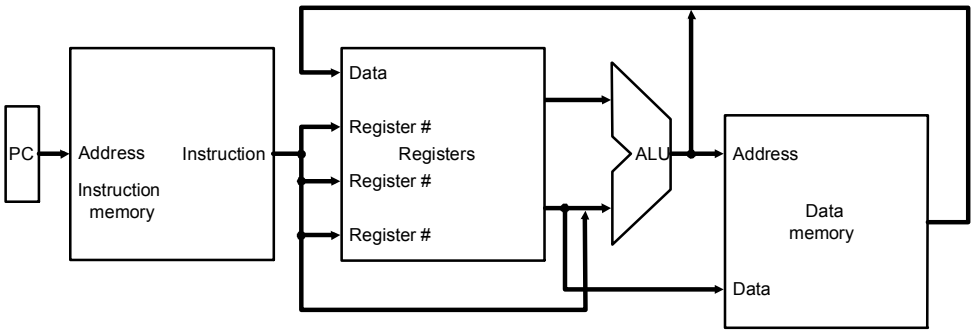
بعد از اینکه از ALU استفاده کردیم، کارهایی که لازم است انجام گیرد تا دستور خاتمه پیدا کند، بین کلاس‌های مختلف متفاوت است. یک دستور مراجعه به حافظه نیاز به دسترسی به حافظه جهت انجام عمل ذخیره سازی برای دستورsw و یا خواندن برای دستورlw دارد. یک دستور محاسباتی و منطقی باید نتیجه حاصل شده از ALU را داخل یکی از رجیسترها بنویسد. در نهایت برای یک دستور پرش، ممکن است لازم شود که آدرس دستور بعدی را بر اساس نتیجه مقایسه عوض کنیم.

---

[1] - Fetch

در شکل ۱ یک مرور سطح بالا از پیاده‌سازی سخت‌افزار پردازنده MIPS نشان داده شده است. در ادامه این فصل ما این شکل را با جزئیات بیشتری توضیح خواهیم داد. ما به این شکل واحدهای عملیاتی بیشتری اضافه خواهیم نمود و تعداد خطوط ارتباطی بین واحدهای مختلف را نیز افزایش خواهیم داد.

همچنین یک واحد کنترل[1] که کار انجام شده برای کلاسهای مختلف دستورات را کنترل می‌کند، نیز به این شکل اضافه خواهد شد. قبل از اینکه ما شروع کنیم و یک سخت‌افزار کامل طراحی کنیم، نیار به یک سری مفاهیم مطرح شده در مدار منطقی خواهیم داشت به همین دلیل در ابتدا مروری خواهیم داشت بر این مفاهیم.



شکل ۱: یک مرور کلی بر پیاده سازی سخت افزاری زیر مجموعه‌ای از دستورات پردازنده MIPS که در آن فقط ماجولهای عملیاتی اصلی و ارتباطات اصلی نشان داده شده است

۵-۱-۲- مروری بر مدارهای منطقی و مفهوم کلاک

مدارهای منطقی به دو دسته کلی مدارهای ترکیبی[2] و ترتیبی[3] تقسیم‌بندی می‌شوند. مدارهای ترکیبی مدارهایی هستند که در آنها عناصر حافظه وجود ندارد و خروجی‌های مدار در هر لحظه فقط به ورودیهای مدار وابسته‌اند. به طور مثال ALU یک مدار ترکیبی است. بر عکس مدارهای ترتیبی مدارهایی هستند که دارای عناصر حافظه‌اند و در آنها خروجی مدار در هر لحظه علاوه بر ورودیهای مدار به حالتهای داخلی یا همان مقادیر ذخیره شده در داخل حافظه‌ها نیز بستگی دارد. از جمله مدارهای ترتیبی می‌توانیم به رجیسترها و حافظه‌ها اشاره کنیم.

هر عنصری از مدار ترتیبی که دارای حافظه باشد، یک عنصر حالت[4] نامیده می‌شود. هر عنصر حالت حداقل دارای دو ورودی و یک خروجی می‌باشد. یکی از ورودیها داده‌ای است که داخل عنصر حالت ذخیره خواهد شد و ورودی دیگر کلاک می‌باشد که مشخص می‌کند عمل ذخیره‌سازی چه

[1] - Control unit
[2] - Combinational
[3] - Sequential
[4] - State element

موقع انجام خواهد شد. خروجی عنصر حالت هم داده‌ای است که در کلاک قبلی در داخل آن ذخیره شده است. به طور مثال فلیپ فلاپ نوع D یکی از ساده‌ترین عناصر حالت می‌باشد که دارای دو ورودیD وکلاک و یک خروجیQ می‌باشد. علاوه بر فلیپ فلاپ نوعD، سخت‌افزار پردازنده MIPS که در این فصل پیاده‌سازی خواهیم کرد، دارای دو عنصر حالت دیگر نیز می‌باشد: حافظه‌ها و رجیسترها. ماژول مربوط به حافظه‌ها و رجیسترها به همراه چند ماژول دیگر در شکل ۱ نشان داده شده است.

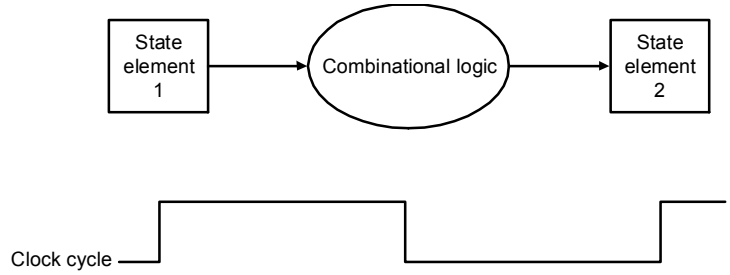
عملیات نوشتن در داخل عنصر حالت با توجه به کلاک انجام خواهد شد اما عملیات خواندن از عنصر حالت در هر زمانی امکان‌پذیر است.

۵-۱-۳- متودولوژی کلاک

متودولوژی کلاک[1] تعیین می‌کند که سیگنالها چه موقع می‌توانند خوانده شوند و چه موقع می‌توانند نوشته شوند. مشخص کردن زمانبندی خواندن‌ها و نوشتن‌ها امر مهمی است، به دلیل اینکه وقتی یک سیگنال در یک زمان هم نوشته شود و هم خوانده شود در این صورت مقداری که خوانده می‌شود، می‌تواند مقدار قبلی نوشته شده در زمانهای قبل، مقدار فعلی که نوشته می‌شود و یا ترکیبی از این دو باشد. بدیهی است که قابل پیش بینی نبودن در طراحی کامپیوتر پذیرفتنی نیست. یک متودولوژی کلاک برای از بین بردن این عدم قطعیت طراحی می‌شود. برای سادگی، ما یک متودولوژی کلاک حساس به لبه[2] را به کار خواهیم کرد. یک متودولوژی کلاک حساس به لبه به این معنی است که مقادیر ذخیره شده در داخل ماشین فقط در لبه‌های کلاک می‌توانند تغییرکنند. بنابراین عناصر حالت مقدارشان فقط در لبه‌های کلاک می‌توانند تغییرکند و بروز شود[3]. به دلیل اینکه فقط عناصر حالت می‌توانند مقدار داده[4] را داخل خود نگهداری کنند، بنابراین مدارهای ترکیبی باید ورودی‌های خود را از عناصر حالت دریافت کنند و خروجی را به عناصر حالت تحویل دهند. مقادیر ورودی مدارهای ترکیبی همان مقادیری هستند که در کلاک قبلی داخل عناصر حالت ذخیره شده‌اند و خروجی آنها مقادیری است که باید در کلاک بعدی داخل عناصر حالت ذخیره شوند. شکل ۲ دو عنصر حالت را نشان می‌دهد که یک مدار ترکیبی را در میان گرفته‌اند و با یک کلاک واحد کار می‌کنند. همهٔ سیگنالها باید از عنصر حالت شماره ۱ شروع شده، از مدار ترکیبی رد شده و به طرف عنصر حالت شماره ۲ منتشر شوند. مدت زمان لازم برای انتشار نباید از یک پریود کلاک بیشتر شود. بنابراین تأخیر انتشار از عنصر حالت ۱ تا عنصر حالت ۲ پریود کلاک را تعیین می‌کند.

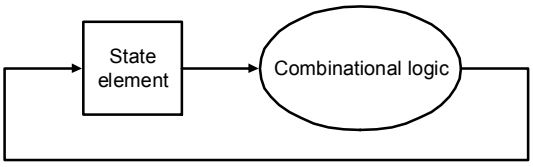
---

[1] - Clocking methodology
[2] - Edge triggered
[3] - Update
[4] - Data value



شکل ۲: مدارهای ترکیبی، عناصر حالت و کلاک، ارتباط نزدیکی به هم دارند

برای سادگی، ما سیگنال کنترل عملیات نوشتن (write) را به هنگامی که عنصر حالت در هر لبهٔ کلاک نوشته می‌شود را نشان نمی‌دهیم. اما اگر یک عنصر حالت در هر لبهٔ کلاک مقدارش بروز نمی‌شود، در این صورت یک خط کنترلی نوشتن مورد نیاز خواهد بود. سیگنالهای کلاک و write ورودی هستند و عنصر حالت فقط زمانی تغییر پیدا می‌کند که به هنگام اتفاق افتادن لبهٔ کلاک، ورودی write فعال باشد. متودولوژی حساس به لبه بودن این امکان را در اختیار ما قرار می‌دهد که محتوای یک رجیستر را بخوانیم و مقدار خوانده شده را از یک مدار ترکیبی رد کنیم و نتیجه بدست آمده را در لبهٔ کلاک بعدی در همان رجیستر بنویسیم. این امر در شکل ۳ نشان داده شده است. فرض اینکه عملیات نوشتن در لبهٔ مثبت کلاک اتفاق بیافتد یا در لبهٔ منفی آن، مهم نیست، به دلیل اینکه ورودی‌های مدار ترکیبی که خروجی عناصر حالت هستند تا لبه انتخاب شده کلاک نمی‌توانند تغییر کنند. با استفاده از متودولوژی زمان‌بندی حساس به لبه، هیچ نوع مسیر feedback و یا Loop ای در داخل یک سیکل کلاک وجود نخواهد داشت و مدار به درستی کار خواهد کرد.



شکل ۳: یک متودولوژی حساس به لبه این امکان را در اختیار قرار می‌دهد که بتوانیم در یک سیکل کلاک عملیات خواندن و نوشتن در داخل یک عنصر حالت را به درستی انجام دهیم

تقریباً همهٔ عناصر حالت و مدارهای ترکیبی در پردازنده MIPS دارای ورودی‌ها و خروجی‌های ۳۲ بیتی خواهند بود، به دلیل اینکه اکثر داده‌هایی که توسط پردازنده MIPS دستکاری می‌شوند دارای طول ۳۲ بیت هستند. در جاهایی که هرکدام از این ورودی‌ها و یا خروجی‌ها دارای طولی غیر از ۳۲ بیت باشند، مشخصاً اعلام خواهیم کرد. در شکل‌هایی که رسم خواهد شد اگر طول سیگنال بیشتر از ۱ بیت باشد در این صورت خط مربوط به آن سیگنال ضخیم‌تر رسم خواهد شد که نشان دهندهٔ طول بیش از یک است. سیگنالی که طول آن بیشتر از ۱ بیت است اصطلاحاً باس[1] یا گذرگاه نامیده می‌شود.

[1] - Bus

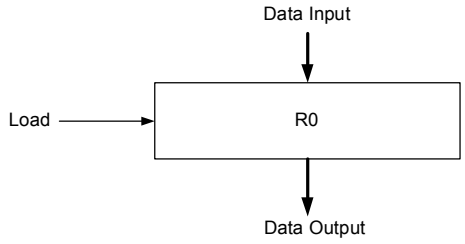
بعضی مواقع ما چند تا باس را با هم ترکیب نموده و یک باس ضخیم‌تر ایجاد خواهیم کرد. به طور مثال یک باس ۳۲ بیتی را از ترکیب دو باس ۱۶ بیتی ایجاد خواهیم کرد. در این صورت برچسب‌هایی که روی خطوط قرار می‌دهیم به وضوح نشان خواهد داد که دو تا باس به هم متصل شده و یک باس بزرگتر را ایجاد کرده‌اند پیکانهایی نیز بر روی خطوط شکل‌ها رسم می‌شود که نشان دهنده جهت جریان داده‌ها بین عناصر حالت خواهد بود.

۵-۱-۴- پیاده‌سازی زیر مجموعه‌ای از دستورات MIPS

ما چند پیاده‌سازی مختلف از سخت‌افزار ارائه خواهیم کرد. در این فصل یک پردازنده ساده طراحی خواهد شد که در آن هر دستوری در یک کلاک انجام می‌گیرد ولی پریود این کلاک بزرگ می‌باشد. این پیاده‌سازی ساده همانند شکل ۱ خواهد بود. در این پردازنده ساده، هر دستوری در یک لبهٔ کلاک شروع به اجرا می‌کند و در لبهٔ‌کلاک بعدی اجرای آن خاتمه پیدا می‌کند. به دلیل اینکه پردازنده ساده طراحی شده در این فصل سرعت پایین‌تری دارد، ما یک پردازنده سریعتر را در فصل ۶ ارائه خواهیم کرد که از تکنیک پایپلاین استفاده می‌کند.

۵-۲- رجیسترها

به دلیل استفاده زیاد و کاربردهای بسیار رجیسترها در طراحی پردازنده، این بخش از فصل به مجموعه رجیسترهای پردازنده، اختصاص پیدا کرده است. رجیسترها در داخل یک پردازنده معمولاً به عنوان فضای ذخیره سازی موقت مورد استفاده قرار می‌گیرند. رجیسترها از حافظه‌های اصلی سریعترند و استفاده از آنها راحت‌تر است. هر رجیستری به تعداد بیت‌هایی که می‌تواند ذخیره کند در داخل خود فلیپ فلاپ دارد. به طور مثال یک رجیستر ۴ بیتی دارای ۴ عدد فلیپ‌فلاپ می‌باشد. هر رجیستری دارای ۲ ورودی به نام‌های clock و reset می‌باشد که بین همه فلیپ فلاپها مشترک است. ورودی کلاک، پالس ساعت را تأمین می‌کند و ورودی reset، اگر فعال شود همه فلیپ فلاپ‌ها و در نتیجه رجیستر را پاک می‌کند. بلاک دیاگرام یک رجیستر در شکل ۴ نشان داده شده است. ورودی‌های کلاک و reset به دلیل سادگی در این شکل نشان داده نشده‌اند، ولی به صورت ضمنی می‌دانیم که این ورودی‌ها وجود دارند.



شکل ۴: بلاک دیاگرام رجیستر

با توجه به بلاک دیاگرام، هر رجیستر دارای دو ورودی به نام‌های Load و Data Input (ورودی داده) می‌باشد. هر موقع که Load=1 باشد، داده ورودی در داخل رجیستر ذخیره خواهد شد. و هر موقع که Load=0 باشد، رجیستر محتوای فعلی خود را حفظ خواهد نمود. محتوای داخلی رجیستر همیشه از طریق خروجی Data Output (خروجی داده)، بدون توجه به ورودی Load در دسترس خواهد بود.

همان‌طور که گفته شد، یک رجیستر n بیتی دارای n عدد فلیپ فلاپ است. بنابراین تعداد بیت-هایی که می‌توان داخل رجیستر ذخیره نمود، n بیت است. پس ورودی Data Input شامل n خط است. همچنین طول داده‌ای که می‌توان از داخل رجیستر خواند، n بیت است. بنابراین خروجی Data Output نیز باید دارای n خط باشد. به دلیل اینکه تعداد خطوط Data Input و Data Output بیشتر از یک خط است، این خطوط در دیاگرام ضخیم‌تر رسم شده‌اند.

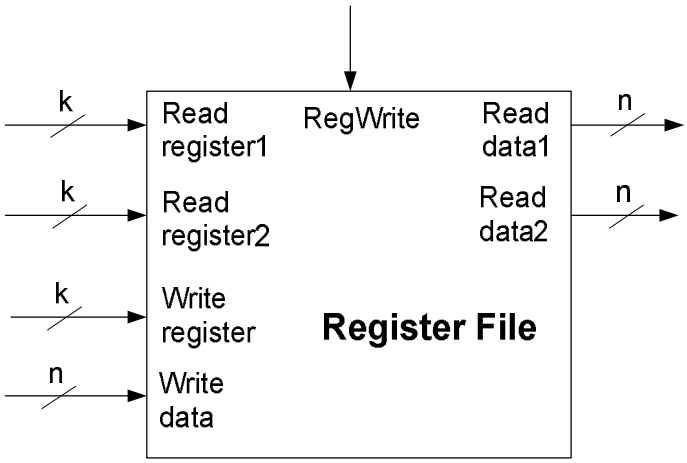
۵-۲-۱- رجیسترهای عمومی پردازنده

هر پردازنده‌ای در داخل خود تعدادی رجیستر برای کاربردهای عمومی دارد که معمولاً در دستورات محاسباتی و منطقی از آنها استفاده می‌کند. در پردازنده‌های مدرن، برای افزایش کارآیی و منظم‌تر شدن طراحی، رجیسترهای عمومی به صورت خاصی گروه‌بندی شده و در داخل یک ماژول به نام بانک رجیستر[1] قرار داده می‌شوند. در این بخش بانک رجیستر را مورد بررسی دقیق قرار می‌دهیم.

۵-۲-۱-۱- بانک رجیستر

بانک رجیستر بسیار شبیه به حافظه RAM می‌باشد که کلمات ذخیره شده در آن همان داده‌های داخل رجیسترهای بانک رجیستر می‌باشد. برای دسترسی به محتویات هر کدام از رجیسترهای داخل بانک رجیستر، باید آدرس آن رجیستر را مشخص کنیم. بلاک دیاگرام یک نمونه بانک رجیستر در شکل ۵ نشان داده شده است

[1] - Register File



شکل ۵: بانک رجیستر

این بانک رجیستر شامل مشخصات زیر است:

- این بانک رجیستر دارای دو پورت خواندن و یک پورت نوشتن است. یعنی در هر زمان می‌توان محتوای دو رجیستر مختلف از مجموعه رجیسترها را خواند و استفاده نمود و در همان حین یک عملیات نوشتن به داخل یکی از رجیسترها انجام داد.
- پورتهای Read register1، Read register2 و Write register برای آدرس‌دهی مورد استفاده قرار می‌گیرند. یعنی با هر کدام از این خطوط می‌توان یک رجیستر را آدرس‌دهی نمود. تعداد رجیسترها مساوی 2^k است در نتیجه تعداد خطوط آدرس می‌شود k.
- داده‌های خوانده شده از دو رجیستر بر روی پورتهای Read register1 و Read register2 خارج می‌شود. داده‌ای هم که قرار است داخل یکی از رجیسترهای بانک رجیستر نوشته شود، بر روی پورت Write data قرار داده می‌شود. هر رجیستر قادر است n بیت اطلاعات را ذخیره نماید. بنابراین طول هر کدام از پورتهای مربوط به داده مساوی n می‌باشد.
- به دلیل اینکه تعداد خطوط آدرس k و تعداد خطوط داده n می‌باشد، در نتیجه اندازه این بانک رجیستر مساوی $2^k \times n$ می‌باشد.

۵-۲-۱-۲- عملیات خواندن از بانک رجیستر

همان طور که ذکر شد، می‌توان در یک حین از دو رجیستر متفاوت عملیات خواندن را انجام داد. برای انجام این کار کافی است که آدرس دو رجیستر را به ترتیب بر روی پورتهای Read

register1 و Read register2 قرار داد. در این صورت داده مربوط به این رجیسترها به ترتیب بر روی پورتهای Read data1 و Read data2 ظاهر خواهد شد. به طور مثال فرض کنید برای دستوری مانند add $3,$2,$5 را که عملیات $3=$2+$5 را انجام می‌دهد، تصمیم گرفته باشیم محتوای دو رجیستر با شماره های 2 و 5 را از بانک رجیستر بخوانیم، در این صورت باید بر روی پورت Read register1 عدد 2 و بر روی پورت Read register2 عدد 5 قرار داده شود. با انجام این کار محتوای رجیستر 2 بر روی پورت Read data1 و محتوای رجیستر 5 بر روی پورت Read data2 به بیرون فرستاده خواهد شد.

۵-۲-۱-۳- عملیات نوشتن در داخل بانک رجیستر

برای انجام عملیات نوشتن، آدرس رجیستری که قرار است نوشته شود بر روی پورت Write register و داده‌ای که قرار است نوشته شود بر روی پورت Write data قرار داده می‌شود و همچنین پورت RegWrite نیز 1 می‌شود. با انجام این کار در لبه مثبت کلاک داده مورد نظر در آن رجیستر نوشته خواهد شد. به طور مثال فرض کنید بخواهیم عدد 398 را در داخل رجیستر شماره 8 بنویسیم. برای این کار باید مقدار 398 را بر روی پورت Write data و مقدار 8 را بر روی پورت Write register قرار دهیم و پورت ورودی RegWrite را نیز 1 نمائیم.

توجه: همان طور که گفته شد بانک رجیستر شامل رجیسترهای عمومی پردازنده است و چون هر رجیستری به کلاک نیاز دارد، باید بانک رجیستر ورودی کلاک هم داشته باشد. در بلاک دیاگرام بانک رجیستر ورودی کلاک نشان داده نشده است ولی به طور ضمنی این ورودی وجود دارد. ما هر موقعی که لازم باشد می‌توانیم از بانک رجیستر بخوانیم ولی عملیات نوشتن فقط در لبه مثبت کلاک انجام خواهد شد.

۵-۲-۱-۴- طراحی سخت افزار بانک رجیستر

پرازنده MIPS دارای ۳۲ رجیستر ۳۲ بیتی است. بنابراین برای بانک رجیستر آن تعداد خطوط آدرس و خطوط داده باید به ترتیب مساوی ۵ و ۳۲ باشد. وقتی که می‌خواهیم از بین ۳۲ رجیستر، دو مورد را انتخاب کنیم و محتویات آنها را بخوانیم، حتماً باید برای هر کدام از انتخابها از یک مالتی-پلکسر استفاده کنیم چون فقط با مالتی‌پلکسر است که می‌توانیم عمل انتخاب را انجام دهیم. از طرفی برای عملیات نوشتن باید فقط خط Load یکی از رجیسترها را فعال کنیم چون در عملیات نوشتن فقط یکی از رجیسترها نوشته خواهد شد. همان طور که می‌دانیم دیکدر مداری است که در هر زمان فقط یکی از خروجی‌های آن فعال می‌شود و بقیه خروجی‌ها غیر فعال‌اند. بنابراین به هنگام نوشتن داخل

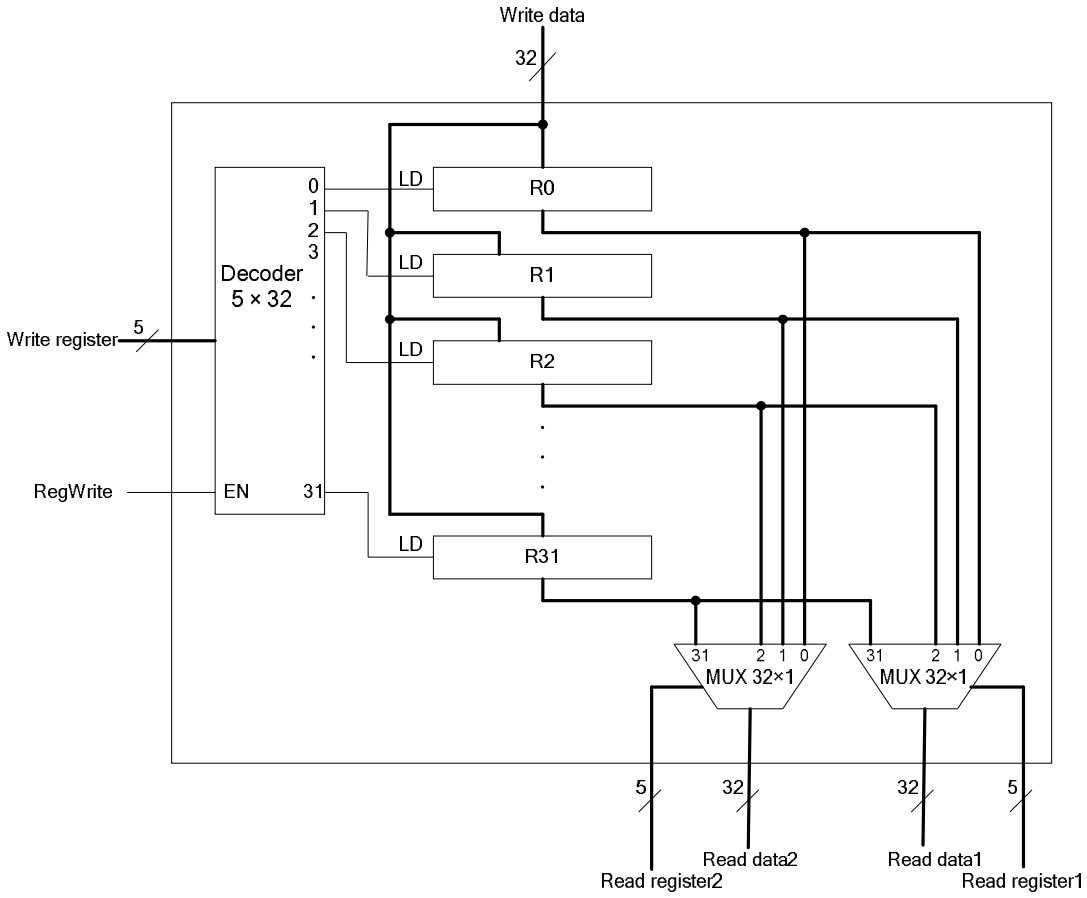
بانک رجیستر می‌توان از یک دیکدر برای فعال کردن خط Load یکی و فقط یکی از رجیسترها استفاده نمود.

شکل ۶ مدار داخلی بانک رجیستر پردازنده MIPS را نشان می‌دهد. خطوط پر رنگ در این شکل نشان دهنده این است که این خط شامل چند خط (چند بیت) است به طور مثال خط write data که پر رنگ است شامل ۳۲ خط است. همان طور که در این شکل دیده می‌شود، برای انتخاب یک رجیستر از بین ۳۲ رجیستر از یک مالتی پلکسر ۳۲ به ۱ استفاده می‌شود و چون مالتی پلکسر دارای ۳۲ خط ورودی است، پس تعداد خطوط انتخاب مالتی‌پلکسر باید ۵ باشد. از آنجایی که باید این امکان وجود داشته باشد که ما بتوانیم محتوای دو رجیستر را در یک زمان بخوانیم، به همین دلیل در این شکل دو مالتی پلکسر تعبیه شده است. ورودی‌های Read register1 و Read register2 برای انتخاب دو رجیستر از بین ۳۲ رجیستر در عملیات خواندن استفاده می‌شوند، به همین دلیل باید این دو ورودی را به خطوط انتخاب مالتی پلکسرها وصل کنیم (هر کدام به یک مالتی پلکسر)، و از آنجا که هر کدام از مالتی پلکسرها از بین ۳۲ رجیستر یکی را انتخاب می‌کنند بنابراین هر کدام از این ورودی‌ها باید شامل ۵ بیت باشند. خروجی هر کدام از مالتی پلکسرها محتوای یکی از ۳۲ رجیستر موجود است و چون هر رجیستری ۳۲ بیتی است بنابراین خروجی مالتی پلکسرها نیز ۳۲ بیتی است. خروجی دو مالتی پلکسر به ترتیب به خروجی‌های Read data1 و Read data2 وصل شده است، بنابراین هر کدام از این خروجی‌ها باید ۳۲ بیتی باشد.

همان طور که ذکر شد از دیکدر می‌توان برای عملیات نوشتن داخل بانک رجیستر و فعال کردن خط Load رجیسترها استفاده نمود. در مدار بانک رجیستر پردازنده MIPS به دلیل وجود ۳۲ رجیستر، از یک دیکدر ۵ به ۳۲ استفاده شده است. ورودی Write register برای انتخاب رجیستری که نوشته خواهد شد به کار می‌رود، به همین دلیل این ورودی را باید به خطوط ورودی دیکدر وصل کنیم و تعداد بیت‌های آن باید مساوی ۵ باشد. همچنین دیکدر استفاده شده دارای یک خط فعال ساز[1] به نام EN است که هر موقع فعال شود (۱ شود)، یکی از خروجی‌های دیکدر فعال خواهد شد و بنابراین عملیات نوشتن داخل یکی از ۳۲ رجیستر انجام خواهد گرفت. همان طور که می‌دانیم در پردازنده MIPS همه دستورات نتیجه خود را داخل رجیستر نمی‌نویسند. به طور مثال دستوراتی مانند add و sub نتیجه خود را داخل یک رجیستر می‌نویسند ولی دستوراتی مانند sw و beq داخل رجیسترها چیزی نمی‌نویسند. بنابراین باید بتوانیم عملیات نوشتن داخل بانک رجیستر را به صورت

[1] - Enable

کنترل شده انجام دهیم. ورودی WriteReg که به خط فعال ساز دیکدر وصل می‌شود برای این منظور استفاده می‌شود.

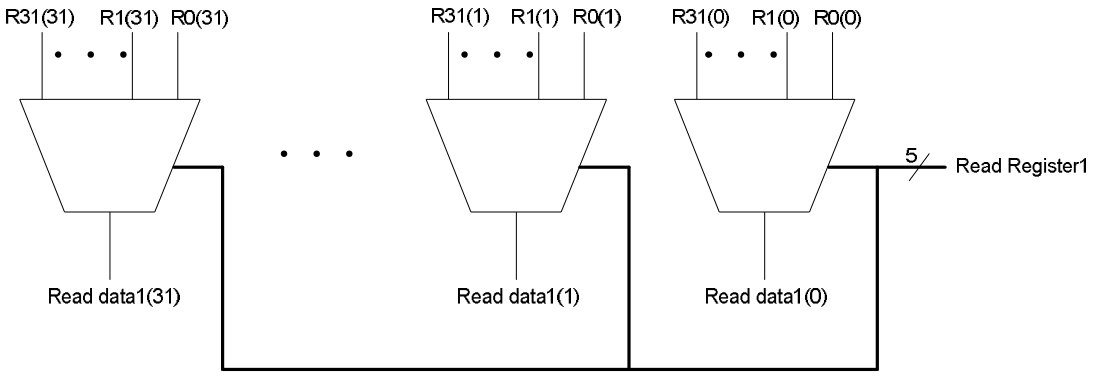


شکل ۶: مدار داخلی بانک رجیستر پردازنده MIPS

خط Write data به ورودی داده همه رجیسترهای بانک رجیستر وصل شده است. وقتی عملیات نوشتن انجام می‌شود، این داده ورودی داخل یکی از رجیسترها نوشته خواهد شد. به دلیل اینکه هر رجیستر شامل ۳۲ بیت است، این خط ورودی نیز باید شامل ۳۲ بیت باشد.

نکته‌ای که در اینجا باید ذکر شود این است که هر کدام از مالتی پلکسرهای بانک رجیستر نشان داده شده از چند مالتی پلکسر تشکیل شده‌اند: یک مالتی پلکسر برای انتخاب از بین بیت‌های 0 ، یک مالتی پلکسر برای انتخاب از بین بیت‌های ۱ و به همین ترتیب. در کل چون ۳۲ بیت داریم، هر کدام از مالتی پلکسرها شامل ۳۲ مالتی پلکسر هستند که خط انتخاب همه این مالتی پلکسرها مشترک است. برای روشنتر شدن مطلب، شکل ۷ را مشاهده کنید. در این شکل مالتی پلکسر مربوط به Read data1 نشان داده شده که شامل ۳۲ مالتی پلکسر می‌باشد. مالتی پلکسر سمت راست برای انتخاب از بین بیت-

های 0 ، مالتی پلکسر بعدی برای انتخاب از بین بیت‌های ۱ و به همین ترتیب تا اینکه مالتی پلکسر آخری هم برای انتخاب از بین بیت‌های ۳۱ رجیسترهای بانک رجیستر به کار می‌رود. همان طور که مشاهده می‌شود خط انتخاب همه این مالتی پلکسرها مشترک است بنابراین در همه مالتی پلکسرها، بیت‌های مربوط به یک رجیستر انتخاب خواهد شد. به طور مثال اگر مالتی پلکسر اولی بیت 0 مربوط به رجیستر R1 را انتخاب کند (Read Register = 00001)، حتماً رجیستر بعدی بیت ۱ رجیستر R1 را انتخاب خواهد کرد، و به همین ترتیب. بنابراین در نهایت همه بیت‌های مربوط به R1 انتخاب خواهند شد.



شکل ۷: مدار هر کدام از مالتی پلکسرهای بانک رجیستر

۵-۲-۲- رجیسترهای مخصوص

در داخل هر پردازنده تعدادی رجیستر خارج از مجموعه رجیسترهای عمومی یا عام منظوره وجود دارند. این رجیسترها استفاده خاصی دارند و نمی‌توان در دستورات پردازنده از آنها استفاده نمود. به عبارتی توسط کاربر قابل دسترسی نیستند. در پردازنده MIPS، تعدادی رجیستر وجود دارد که استفاده خاص داشته و خارج از مجموعه رجیسترهای عمومی یا بانک رجیستر قرار گرفته‌اند. یکی از این رجیسترها، رجیستر شمارنده برنامه یا PC[1] می‌باشد. رجیستر PC برای دسترسی به حافظه دستورات مورد استفاه قرار می‌گیرد و آن را آدرس دهی می‌کند. به عبارتی دستور بعدی که در داخل پردازنده اجرا خواهد شد از آدرس PC خوانده خواهد شد. در پردازنده MIPS دو رجیستر مخصوص دیگر به نام‌های Hi و Lo وجود دارد که فقط برای عملیات ضرب و تقسیم از آنها استفاده می‌شود.

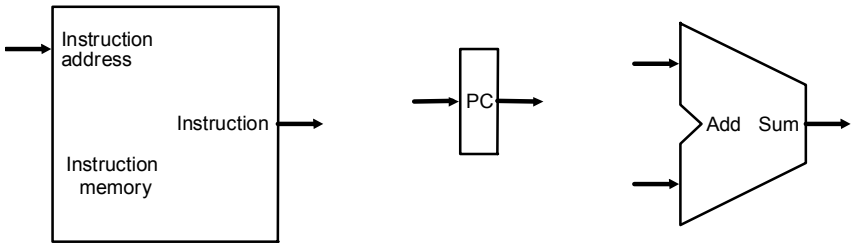
تعریف: به رجیسترهایی که استفاده مخصوص دارند رجیسترهای خاص منظوره یا رجیسترهای با کاربرد خاص نیز گفته می‌شود. و به رجیسترهای عمومی رجیسترهای با عام منظوره یا رجیسترهای با کاربرد عام نیز گفته می‌شود.

[1] - Program Counter

۵-۳- ساختن یک مسیر داده

مسیر داده یا data path واحدی از پردازنده است که همهٔ عملیات محاسباتی و منطقی مورد نیاز همهٔ دستورات را انجام می‌دهد. یک روش ساده برای طراحی مسیر داده، پیدا کردن ماژولهای اصلی مورد نیاز برای اجرای هر کدام از کلاس‌های دستورات می‌باشد. بنابراین ما برای هر کدام از کلاس-های دستورات، ماژولهای مورد نیاز را پیدا کرده و بخش‌هایی از مسیر داده که برای اجرای این کلاس دستور مورد نیاز ایت را طراحی خواهیم کرد. در کنار عناصر مسیر داده که برای هر بخش نشان خواهیم داد، سیگنالهای کنترلی مربوطه را نیز در برخی موارد توضیح خواهیم داد.

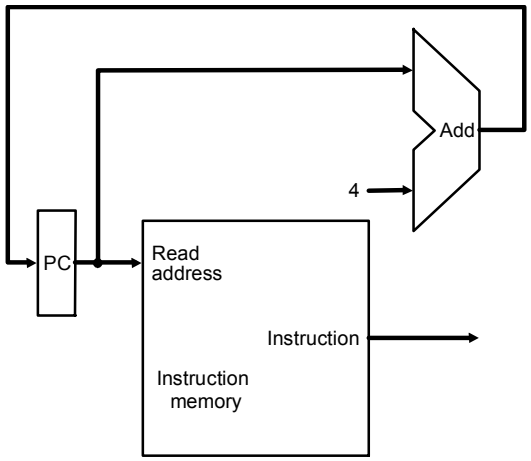
اولین ماژولی که مورد نیاز است یک محل ذخیره‌سازی برای نگهداری دستورات یک برنامه است. یک واحد حافظه که یک عنصر حالت می‌باشد برای نگهداری دستورات به کار می‌رود و به کمک یک خط آدرس می‌توانیم این حافظه را آدرس دهی کرده و دستورات را از داخل آن بخوانیم. عنصر حافظه مورد نیاز در شکل ۸ نشان داده شده است. آدرس دستور نیز باید در داخل یک عنصر حالت نگهداری شود که به آن شمارنده برنامه[1] یا PC گفته می‌شود. شمارنده برنامه نیز در شکل ۸ نشان داده شده است. و در نهایت ما نیاز به این داریم که PC را اضافه کنیم تا اینکه آدرس دستور بعدی بدست آید بنابراین نیاز به یک جمع کننده نیز داریم. این جمع کننده یک مدار ترکیبی است که می‌تواند از همان ALU که در فصل ۴ طراحی شد بدست آید، برای این منظور کافی است که خط کنترلی ALU را طوری مقداردهی کنیم که همیشه عملیات جمع را انجام دهد، به عبارتی باید به این صورت مقداردهی کنیم: $\text{ALU Operation} = 010$. ما این ALU خاص را که فقط عملیات جمع انجام می‌دهد، با برچسب Add نشان خواهیم داد. جمع کننده آدرس نیز در شکل ۸ نشان داده شده است.



شکل ۸: برای ذخیره سازی و دسترسی به دستورات، دو عنصر حالت مورد نیاز است. همچنین یک جمع کننده نیز برای محاسبه آدرس دستور بعدی مورد نیاز است.

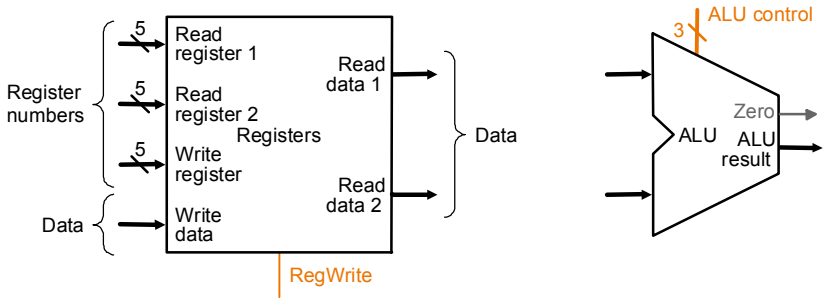
[1] - Program counter

برای اجرای هر دستور، در ابتدا باید دستور از حافظه خوانده شود. به این عمل، واکشی دستور یا Fetch گفته می‌شود. برای اینکه آمادگی این را داشته باشیم که پس از اجرای این دستور، دستور بعدی را نیز اجرا کنیم، ما باید شمارنده برنامه را طوری اضافه کنیم تا به ۴ بایت بعدتر اشاره کند چون هر دستوری در پردازنده MIPS دارای طول ۴ بایت است. بنابراین برای اینکه PC به دستور بعدی اشاره کند باید به آن ۴ واحد اضافه نمود. مسیر داده مورد نیاز برای این کار در شکل ۹ نشان داده شده است. همان طور که مشاهده می‌شود، این شکل از ۳ عنصر نشان داده شده در شکل ۸ استفاده می‌نماید.



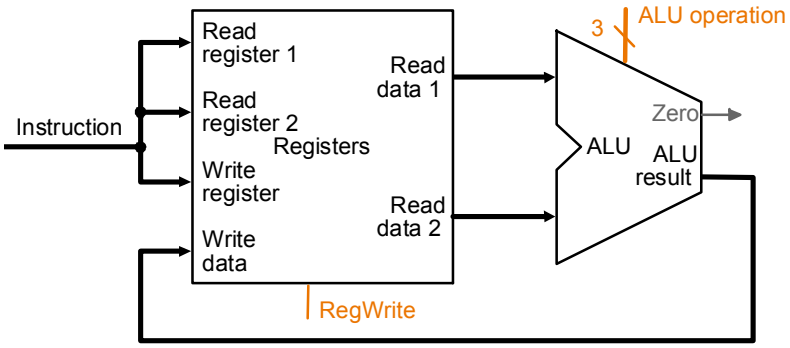
شکل ۹: قسمتی از مسیر داده که برای واکشی دستور از حافظه و همچنین افزایش PC به کار می‌رود

حال بیائید دستورات نوع R را در نظر بگیریم. همهٔ این دستورات محتوای دو رجیستر را می-خوانند، یک عملیات ALU بر روی محتوای رجیسترهای خوانده شده انجام می‌دهند و نتیجه حاصل شده را در داخل یک رجیستر می‌نویسد. دستورات این کلاس شامل add، sub، slt، and و or می-باشند. به طور مثال دستور add $t1, $t2, $t3 را در نظر بگیرید. این دستور محتوای دو رجیستر$t2 و $t3 را خوانده و با هم جمع کرده و نتیجه را در $t1 می‌نویسد. برای اجرای دستورات نوع R ما نیاز به یک بانک رجیستر داریم تا از طریق آن محتوای دو رجیستر را بخوانیم. همچنین برای انجام عملیات، نیاز به یک ALU داریم. در نهایت که عملیات انجام گرفت نتیجه باید در داخل بانک رجیستر ذخیره گردد. ماژولهای لازم برای دستورات نوع R در شکل ۱۰ نشان داده شده‌اند. ALU نشان داده شده در این شکل همان ALU طراحی شده در فصل ۴ می‌باشد. این ALU دو ورودی ۳۲ بیتی دریافت نموده و یک خروجی ۳۲ بیتی را به عنوان نتیجه تولید می‌کند. بانک رجیستر نشان داده شده در این شکل همان بانک رجیستر طراحی شده در این فصل است.



شکل ۱۰: دو عنصر مورد نیاز برای پیاده سازی دستورات نوع R عبارتند از: بانک رجیستر و ALU

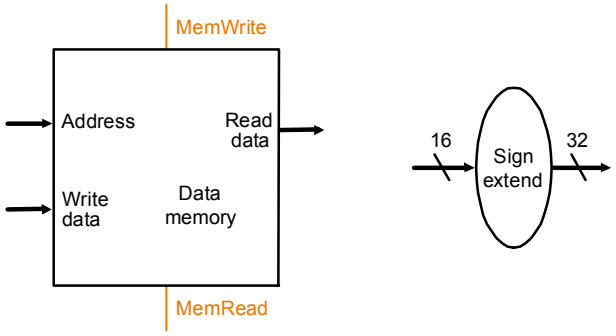
مسیر داده لازم برای اجرای دستورات نوع R که از ماژولهای شکل ۱۰ استفاده می‌کند در شکل ۱۱ نشان داده شده است. به دلیل اینکه شماره رجیسترهایی که باید خوانده شوند و یا نوشته شوند از فیلدهای مربوطه در دستوری که اجرا می‌شود، استخراج می‌شوند، ما برای این شکل یک ورودی به نام Instruction در نظر گرفته‌ایم. این ورودی باید به خروجی حافظه در شکل ۹ وصل گردد چون این حافظه، دستورات برنامه را نگهداری می‌کند و خروجی آن همان دستوری است که اجرا خواهد شد.



شکل ۱۱: مسیر داده دستورات نوع R

حال بیائید دستورات مراجعه به حافظه lw و sw را که دارای شکل کلی lw $t1, offset_value($t2) و sw $t1, offset_value($t2) می‌باشند را بررسی نمائیم. این دستورات با جمع کردن رجیستر پایه که در اینجا t2 بوده با یک مقدار ثابت ۱۶ بیتی علامت دار که همان Offset می‌باشد، یک آدرس برای حافظه محاسبه می‌کنند. اگر دستور sw باشد، مقداری که در داخل حافظه ذخیره خواهد شد باید از بانک رجیستر خوانده شود. به دلیل اینکه محتوای یک رجیستر در داخل حافظه ذخیره خواهد شد که باید آن را از داخل بانک رجیستر خواند (در اینجا رجیستر t1). اگر دستور از نوع lw باشد، مقداری که از حافظه خوانده می‌شود باید در داخل بانک رجیستر ذخیره شود (در داخل رجیستر t1 بانک رجیستر). بنابراین برای دستورات مراجعه به حافظه، ما به ALU و بانک رجیستر نشان داده شده در شکل ۱۰ نیاز خواهیم داشت.

علاوه بر این، ما به یک واحد به نام واحد گسترش بیت علامت که مقدار ثابت ۱۶ بیتی فیلد Offset را به ۳۲ بیت تبدیل می‌کند نیاز داریم. این واحد ۱۶ بیت بالا را با علامت عدد پر می‌کند. همچنین ما به یک ماژول حافظه برای خواندن یا نوشتن داده‌ها نیاز داریم. این ماژول حافظه چون برای داده‌ها استفاده می‌شود، حافظه داده[1] نامیده می‌شود. چون حافظه داده، هم می‌تواند خوانده شود و هم می‌تواند نوشته شود، بنابراین باید دارای خطوط کنترلی برای عملیات نوشتن و خواندن باشد. همچنین این حافظه باید یک ورودی برای آدرس و یک ورودی برای داده‌ای که نوشته خواهد شد داشته باشد. شکل ۱۲ ماژولهای حافظه داده و گسترش بیت علامت را نشان می‌دهد.

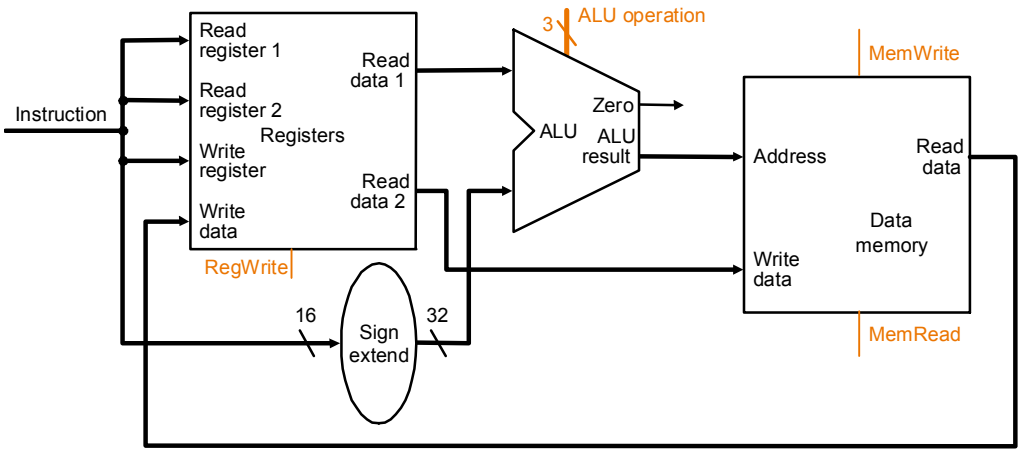


a. Data memory unit b. Sign-extension unit

شکل ۱۲: ماژول‌های حافظه داده و گسترش بیت علامت که در دستورهای مراجعه به حافظه از آنها استفاده می‌شود

شکل ۱۳ مسیر داده مربوط به دستورات lw و sw را نشان می‌دهد. همان طور که دیده می‌شود در این شکل ماژولهای ALU، بانک رجیستر، گسترش بیت علامت و حافظه داده‌ها وجود دارند. در این شکل ALU یک عدد ثابت ۱۶ بیت که تبدیل به ۳۲ بیت شده را با محتوای یک رجیستر که از بانک رجیستر خوانده می‌شود با هم جمع کرده و آدرس حافظه را نتیجه می‌دهد. شماره رجیستری که محتوای آن از بانک رجیستر خوانده می‌شود و همچنین عدد ثابت ۱۶ بیتی در فیلدهای مربوطه در دستورهای lw و sw قرار دارند. بنابراین یکی از ورودیهای این مسیر داده دستور خوانده شده از حافظه است.

[1] - Data memory



شکل ۱۳: مسیر داده دستورات مراجعه به حافظه

دستور beq دارای سه عملوند[1] می‌باشد: دو رجیستر که با هم مقایسه می‌شوند و یک مقدار ثابت ۱۶ بیتی که برای بدست آوردن آدرس پرش استفاده می‌شود. به آدرس پرش، آدرس مقصد پرش یا branch target address نیز گفته می‌شود. شکل دستور beq به صورت beq $t1,$t2,offset. برای پیاده‌سازی این دستور ما باید مقدار ثابت ۱۶ بیتی را با محتوای شمارنده برنامه جمع کنیم.

در مورد دستورات پرش باید به دو نکته مهم توجه کنیم:

- معماری مجموعه دستورات بیان می‌کند که آدرس base برای دستورات پرش، آدرس دستور بعد از دستور پرش می‌باشد. به دلیل اینکه ما محاسبه PC+4 (آدرس دستور بعدی) را در مسیر داده مربوط به واکشی دستور انجام داده‌ایم، راحت‌تر این است که ما این مقدار را برای محاسبه آدرس دستور پرش نیز استفاده کنیم.
- معماری مجموعه دستورات همچنین بیان می‌کند که فیلد offset باید دو مرتبه به سمت چپ شیفت پیدا کند. به دلیل این که این offset، آفست یک کلمه ۳۲ بیتی است. همان طور که در فصل ۳ نیز توضیح داده شد، این میزان شیفت باعث می‌شود که محدودهٔ مؤثر آفست با ضریب ۴ افزایش پیدا کند.

برای اینکه مورد دوم را در سخت‌افزار به درستی انجام دهیم باید فیلد آفست را دو مرتبه به سمت چپ شیفت دهیم.

علاوه بر محاسبهٔ آدرس پرش، ما باید این مورد را نیز مشخص کنیم که آیا دستور beq بعد از اجرا واقعاً به آدرس مقصد پرش، پرش را انجام خواهد داد و یا اینکه پرش انجام نخواهد شد و

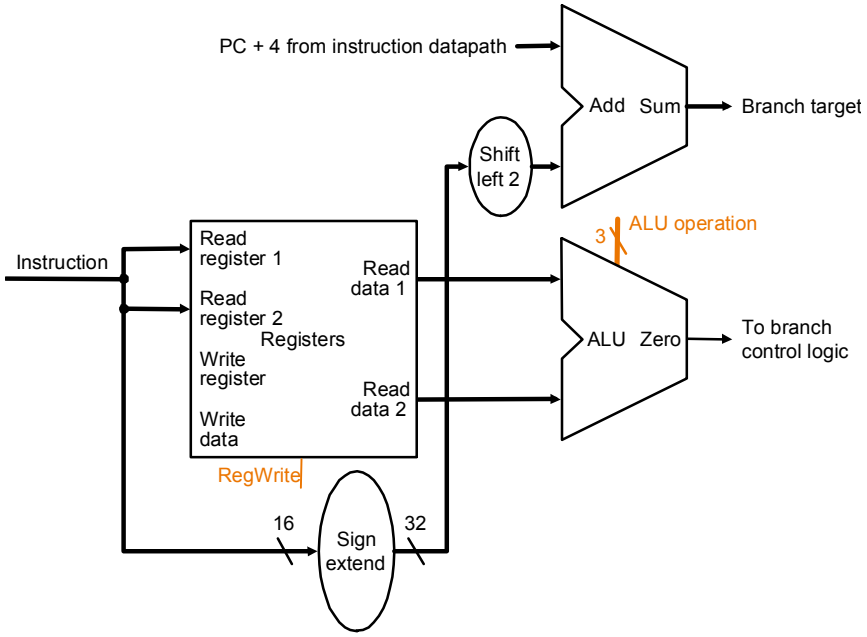
[1] - Operand

بعد از دستور beq دستور پشت سر آن اجرا خواهد شد. موقعی که شرط مقایسه برقرار می‌شود (یعنی دو عملوند با هم مساوی می‌شوند)، مقدار جدید pc، آدرس محاسبه شده برای مقصد پرش خواهد بود و در این حالت می‌گوییم که پرش انجام می‌شود و یا اصطلاحاً گفته می‌شود Branch is taken.

اگر دو عملوند با هم مساوی نباشند، مقدار جدید PC مساوی PC+4 خواهد بود (آدرس دستور پشت سر دستور پرش). در این حالت گفته می‌شود که پرش انجام نمی‌شود و یا اصطلاحاً Branch is not taken..

بنابراین مسیر داده دستور پرش باید دو عملیات را انجام دهد: محاسبه آدرس مقصد پرش و مقایسه محتوای رجیسترها. (دستورهای پرش برای انجام شدن درست عملیات نیاز دارند که مسیر داده مربوط به واکشی دستور را مقداری تغییر دهند که این مورد را بعداً توضیح خواهیم داد.)

شکل ۱۴ مسیر داده دستور beq را نشان می‌دهد. برای محاسبهٔ آدرس مقصد پرش، مسیر داده دستورbeq باید شامل یک واحد گسترش بیت علامت و یک واحد ALU باشد. برای انجام مقایسه ما باید از بانک رجیستر استفاده کنیم که محتوای دو رجیستر را در اختیار ما قرار دهد و همچنین از یک ALU نظیر ALU فصل ۴ که عملیات مقایسه این دو رجیستر را انجام دهد. به دلیل اینکه این ALU یک خروجی به نام Zero دارد که نشان می‌دهد نتیجه عملیات ALU صفر شده است یا خیر، ما می‌توانیم به کمک ALU عملیات تفریق را بر روی محتوای دو رجیستر انجام دهیم (خط کنترل ALU را طوری مقداردهی کنیم که عملیات Sub را انجام دهد) و بعد خروجی Zero را بررسی کنیم اگر این خروجی یک شود نشان دهنده این است که دو رجیستر با هم مساوی هستند و در غیر این صورت مساوی نیستند. در ادامه ما نحوه اتصال خطوط کنترلی را توضیح خواهیم داد.



شکل ۱۴: مسیر داده دستور beq

دستور jump به این صورت انجام می‌شودکه ۲۸ بیت پایین PC را با ۲۶ بیت از بیت‌های دستورالعمل j که ۲ بیت به سمت چپ شیفت داده می‌شود، پر می‌کنیم. این شیفت به سادگی با قرار دادن دو بیت (00) در سمت راست ۲۶ بیت انجام می‌شود. نتیجه شیفت ۲۸ بیت می‌شود که این ۲۸ بیت به جای ۲۸ بیت پایین PC قرار می‌گیرد.

حال که ما مسیر داده مورد نیاز برای اجرای کلاس‌های مختلف دستورات را توضیح دادیم، می‌توانیم این مسیر داده‌ها را با هم ترکیب نموده و یک مسیر داده کلی ایجاد کنیم و خطوط کنترلی را نیز به آن اضافه کنیم تا اینکه طراحی ما کامل شود. در بخش‌های بعدی ما یک پردازنده ساده به کمک مسیر داده‌هایی که توضیح داده شد، طراحی خواهیم نمود. این پردازنده هر دستوری را در یک کلاک انجام خواهد داد. چون هر دستور در این پردازنده در یک کلاک انجام می‌شود، نامی که به آن اختصاص خواهیم داد، پردازنده تک سیکلی یا Single cycle processor خواهد بود.

۵-۴- طراحی یک پردازنده ساده

در این بخش ما ساده‌ترین پیاده‌سازی ممکن از زیر مجموعه انتخاب شده پردازنده MIPS را ارائه خواهیم کرد. این پیاده‌سازی شامل یک مسیر داده ساده و یک بخش کنترل ساده است. مسیر داده پردازنده ساده از ترکیب مسیر داده‌های معرفی شده در بخش قبلی ایجاد خواهد شد. پردازنده‌ای که ما طراحی خواهیم کرد دستورات lw، sw، beq، add، sub، and، or و slt را پشتیبانی خواهد کرد.

۵-۴-۱- طراحی یک مسیر داده ساده

فرض کنید که ما بخواهیم یک مسیر داده را از قطعه‌هایی که در شکل ۹، شکل ۱۱، شکل ۱۳ و شکل ۱۴ نشان داده شده‌اند بسازیم. این مسیر داده ساده هر دستوری را در یک کلاک اجرا خواهد کرد. اگر هر دستوری در یک کلاک اجرا شود، در این صورت هیچ کدام از مسیرهای داده مرجع نمی‌توانند بیش از یک بار در یک دستور استفاده شوند، بنابراین اگر ماژولی بیش از یک بار استفاده شود باید به تعداد مورد نیاز از آن ماژول قرار دهیم.

در مسیر داده ساده‌ای که طراحی می‌شود ممکن است یک ماژول در بیش از چند دستور استفاده شود. برای اینکه از یک ماژول در بیش از چند دستور استفاده شود، ما باید امکان اتصال چند ورودی را به ورودی آن ماژول فراهم کنیم و یک سیگنال کنترلی داشته باشیم که از بین آنها یکی را انتخاب کند. عمل انتخاب معمولاً به کمک مالتی پلکسر صورت می‌گیرد.

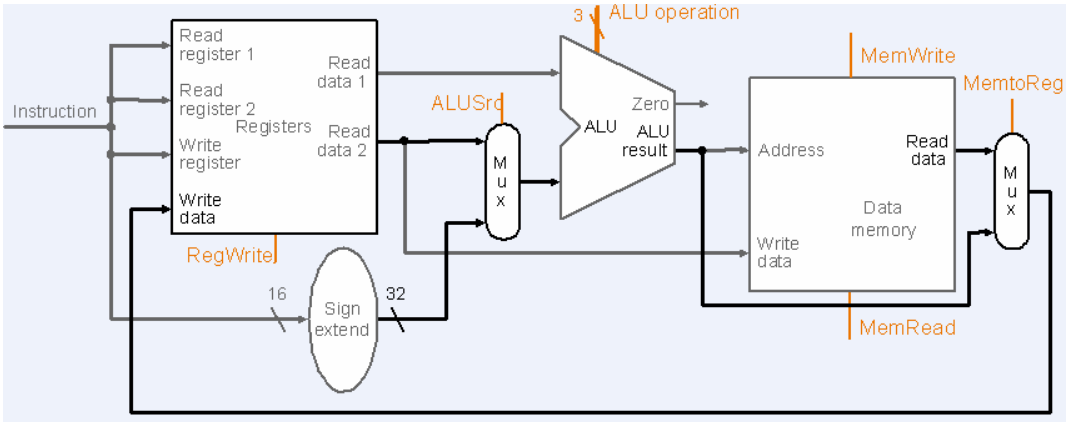
مثال: مسیر داده دستورات نوع R نشان داده شده در شکل ۱۱ و مسیر داده دستورات مراجعه به حافظه نشان داده شده در شکل ۱۳، شباهت زیادی به هم دارند.

تفاوتهای مهمی که بین آنها وجود دارد به شرح زیر است:

- ورودی دومی که وارد ALU می‌شود یا یک رجیستر است (اگر دستور از نوع R باشد) و یا داده‌ای است که گسترش بیت علامت داده شده است (در دستورات مراجعه به حافظه)
- مقداری که در داخل یک رجیستر ذخیره خواهد شد یا از ALU می‌آید (دستورات نوع R) و یا از حافظه(دستور load)

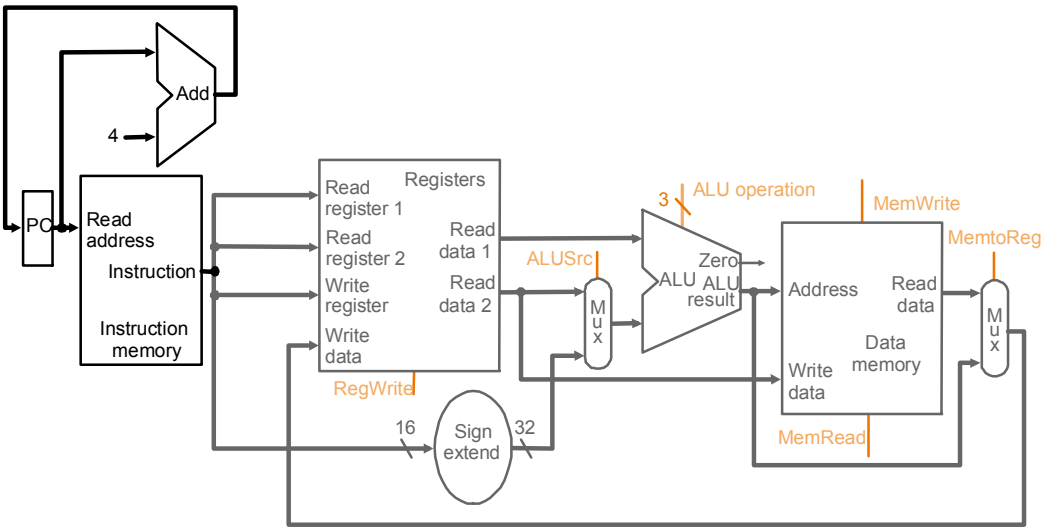
توضیح دهید که چگونه می‌توانیم این دو مسیر داده را با هم ترکیب کنیم به شرطی که از ماژولهای مشترکی که در این دو شکل وجود دارند فقط یکبار استفاده کنیم؟ (به طور مثال ماژول بانک رجیستر در هر دو شکل وجود دارد و در ترکیب دو مسیر داده فقط باید از یک بانک رجیستر استفاده کنیم).

جواب: برای ترکیب این دو مسیر داده و استفاده از فقط یک ALU و یک بانک رجیستر، ما باید برای ورودی دوم ALU این امکان را فراهم کنیم که از دو منبع مختلف داده دریافت کند، و همین طور برای داده‌ای که داخل بانک رجیستر ذخیره خواهد شد باید از دو منبع مختلف داده استفاده کنیم. بنابراین ما به یک مالتی‌پلکسر در ورودی دوم ALU و یک مالتی‌پلکسر دیگر در ورودی دوم بانک رجیستر نیاز خواهیم داشت. شکل ۱۵ مسیر داده ترکیب شده را نشان می‌دهد.



شکل ۱۵: ترکیب مسیر داده دستورات مراجعه به حافظه و دستورات نوع R

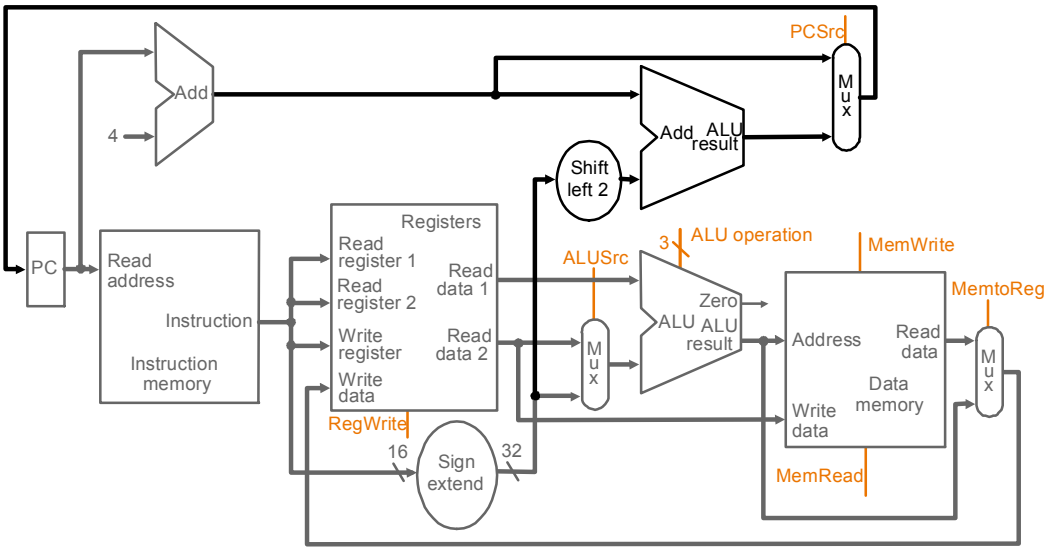
مسیر داده مربوط به واکشی دستور که در شکل ۹ نشان داده شده است به راحتی می‌تواند به مسیر داده شکل ۱۵ اضافه گردد. شکل ۱۶ نتیجه را نشان می‌دهد. این مسیر داده دارای دو حافظه جدا یکی برای دستورات و دیگری برای داده‌ها است. همچنین این مسیر داده به دو ALU احتیاج دارد به دلیل اینکه یکی از ALU ها برای اضافه کردن PC و دیگری در همان زمان برای اجرای دستور به کار می‌رود.



شکل ۱۶: ترکیب مسیرهای داده دستورات مراجعه به حافظه، دستورات نوع R و واکشی دستور

حال ما می‌توانیم با اضافه کردن مسیر داده دستورات پرش نشان داده شده در شکل ۱۴ به شکل ۱۶ یک مسیر داده ساده برای معماری MIPS ایجاد کنیم. شکل ۵-۱۳ مسیر داده‌ای را که ما با ترکیب همهٔ تکه مسیرهای داده جداگانه بدست آورده‌ایم نشان می‌دهد. دستور پرش از ALU اصلی برای مقایسه دو عملوند رجیستری استفاده می‌کنید، بنابراین ما باید جمع کننده‌ای که در شکل ۵-۱۰ برای بدست آوردن آدرس پرش استفاده می‌شود را در مسیر داده نهایی حفظ کنیم.

همان طور که در شکل ۵-۱۳ نشان داده شده است یک مالتی‌پلکسر دیگر نیز مورد نیاز خواهد بود تا از بین آدرس بعدی (PC+4) و آدرس مقصد پرش یکی را برای نوشته شدن در داخل PC یکی را انتخاب کنیم.



شکل ۱۷: مسیر داده ساده برای معماری MIPS که از ترکیب همه قطعه مسیرهای قبلی بدست آمده است

حال که ما طراحی مسیر داده ساده را کامل کردیم، می‌توانیم واحد کنترل را به آن اضافه کنیم و بدین ترتیب، کار طراحی پردازنده ساده را کامل کنیم. کار واحد کنترل همان طور که از نام آن پیداست کنترل عملکرد بخش‌های مختلف پردازنده است. واحد کنترل است که بسته به دستوری که اجرا می‌شود، به مسیر داده فرمان می‌دهد که عملیات مشخصی را انجام دهد مثلاً اگر دستور add اجرا می‌شود، عملیات جمع را انجام دهد. واحد کنترل باید توانایی دریافت ورودیها را داشته باشد و سیگنالهای write مورد نیاز برای همهٔ عناصر حالت، خطوط انتخاب همهٔ مالتی‌پلکسرها و خطوط کنترلی ALU را تولید نماید. از آنجایی که کنترل ALU تا اندازه‌ای با کنترل عناصر دیگر متفاوت است، بنابراین بهتر است قبل از طراحی بقیه قسمتهای واحد کنترل، در ابتدا طراحی واحد کنترل ALU را انجام دهیم.

۵-۴-۲- طراحی واحد کنترل ALU

اگر به یاد داشته باشید ALU طراحی شده در فصل ۴ دارای سه ورودی کنترلی بود. فقط ۵ ترکیب از ۸ ترکیب ممکن ورودیها در این ALU استفاده می‌شد و عملکرد ALU مطابق با جدول زیر بود:

ورودی‌های کنترل ALU	عملی که ALU انجام می‌دهد
000	AND
001	OR
010	add
110	subtract
111	Set on less than (slt)

بسته به کلاس دستور، ALU باید یکی از ۵ عملیات این جدول را انجام دهد. برای دستورات lw و sw ما از ALU برای محاسبه آدرس حافظه استفاده می‌کنیم که در این حالت ALU عملیات Add را انجام می‌دهد. برای دستورات نوع R، ALU باید یکی از ۵ عملیات add، sub، and، or یا slt را انجام دهد (بسته به اینکه فیلد ۶ بیتی Funct در فرمت دستور چه مقداری داشته باشد.) برای دستور beq نیز باید عملیات تفریق توسط ALU انجام شود.

ما می‌توانیم خطوط کنترلی ALU را توسط یک واحد کنترل کوچک تولید نمائیم. این واحد کنترلی کوچک، در ورودی خود فیلد ۶ بیتی Funct و دو بیت را که ما آن را ALUOp نام‌گذاری می‌کنیم دریافت می‌کند. ALUOp مشخص می‌کند که عملیاتی که باید انجام گیرد، add (00) برای دستورهای lw و sw، یا sub (10) برای beq و یا توسط فیلد Funct دستور مشخص شود (10). خروجی این واحد کنترل کوچک سه بیت کنترلی مربوط به خطوط کنترل ALU می‌باشد.

شکل ۱۸ یک جدول صحت را نشان می‌دهد که در آن خروجی‌های واحد کنترل کوچک که همان خطوط کنترل ALU می‌باشند بر اساس ورودیها که همان فیلد Funct و دو بیت ALUOp می‌باشند، مقداردهی شده‌اند. برای کامل‌تر شدن شکل، ارتباط بین بیت‌های ALUOp و opcode دستور نیز نشان داده شده است. در ادامه این فصل خواهیم دید که بیت‌های ALUOp توسط واحد کنترل اصلی تولید خواهند شد.

opcode دستور	ALUOp	عملیاتی که دستور انجام می‌دهد	فیلد Funct	عملیاتی که ALU انجام می‌دهد	خطوط کنترلی ALU
LW	00	Load word	xxxxxx	add	010
SW	00	Store word	xxxxxx	add	010
branch equal	01	Branch equal	xxxxxx	subtract	110
R-type	10	add	100000	add	010
R-type	10	subtract	100010	subtract	110

R-type	10	AND	100100	and	000
R-type	10	OR	100101	or	001
R-type	10	Set on less than	101010	Set on less than	111

شکل ۱۸: نحوه مقدارگیری خطوط کنترل ALU بر اساس بیت‌ها ورودی ALUOp و Funct (فیلد Funct)

استفاده از چند سطح برای عملیات دیکدینگ (به این معنی که واحد کنترل اصلی بیت‌های ALUOp را تولید نماید و پس از آن ALUOp به عنوان ورودی به واحد کنترل ALU که سیگنالهای کنترلی ALU را تولید می‌کند فرستاده شود)، یک تکنیک معمول، برای پیاده‌سازی است. استفاده از چند سطح کنترل می‌تواند موجب کوچکتر شدن اندازه واحد کنترل اصلی شود. استفاده از چند واحد کنترل کوچک به جای یک واحد کنترل بزرگ ممکن است باعث بهبود سرعت واحد کنترل نیز بشود. بهینه‌سازی‌هایی از این دست، امر مهمی است به دلیل اینکه واحد کنترل معمولاً در مسیر بحرانی پردازنده قرار داشته و باعث کاهش فرکانس می‌شود.

روشهای زیادی برای طراحی این مدار که دارای دو بیت ورودی ALUOp و ۶ بیت ورودی Funct بوده و سه بیت خروجی برای کنترل ALU را تولید می‌کند، وجود دارد. به دلیل اینکه فقط تعداد کمی از ۶۴ حالت ممکن فیلد Funct در اینجا استفاده می‌شود و فیلد Funct فقط زمانی استفاده می‌شود که دو بیت ALUOp مساوی 10 باشند، بنابراین ما می‌توانیم یک مدار کوچک طراحی کنیم که این حالتهای ممکن را تشخیص داده و بیت‌های کنترلی ALU را تولید نماید.

به عنوان یک گام از طراحی این مدار، بهتر است که یک جدول صحت برای ترکیب‌های ممکن از فیلد Funct و بیت‌های ALUOp ایجاد کنیم. شکل ۱۹ این جدول را نشان می‌دهد و در آن مقادیر خطوط کنترلی ALU (operation) بر اساس ورودیها مشخص شده‌اند. به دلیل اینکه جدول صحت کامل برای این مثال خیلی بزرگ است (۲۵۶=$۲^۸$ سطر جدول) و همچنین برای ما مهم نیست که به ازای بعضی از این ترکیبات ورودی، بیت‌های کنترلی ALU چه مقداری داشته باشند، بنابراین ما در این جدول فقط سطرهایی را نشان داده‌ایم که به ازای آن خطوط کنترل ALU باید مقدار مشخص داشته باشند. در تمامی این فصل ما به این صورت عمل خواهیم کرد و فقط سطرهایی را نشان خواهیم داد که مورد نیاز می‌شوند و سطرهایی را که مقدار آنها برای ما مهم نیست را نشان نخواهیم داد. در جدول نشان داده شده در شکل ۱۹، در برخی از این سطرها مقدار بعضی از ورودیها X است. این X به معنی این است که خروجی موجود در این سطرها به مقدار این ورودی بستگی ندارد و ورودی می‌تواند هر مقداری داشته باشد. به طور مثال وقتی که بیتهای ALUOp مساوی 00 هستند (سطر اول جدول)، ما همیشه خروجی‌ها (خطوط کنترل ALU) را بدون توجه به فیلد Funct مساوی 010 قرار می‌دهیم. در این حالت بیت‌های ورودی Funct در این سطر جدول اهمیتی نخواهند داشت به همین دلیل مقدار آنها

را X قرار داده‌ایم. به محض اینکه جدول صحت آماده شد، ما می‌توانیم این جدول را ساده‌سازی نموده (به طور مثال ممکن است از جدول کارنو استفاده کنیم) و آن را تبدیل به مدار کنیم.

ALUOp		فیلد Funct						Operation (خطوط کنترل ALU)
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
X	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

شکل ۱۹: جدول صحت برای سه بیت کنترلی ALU

ما قصد داریم که این مدار (مدار واحد کنترل ALU) را طراحی کنیم. فرض کنید این مدار دارای سه خروجی جداگانه به نام‌های Operation0، Operation1 و Operation2 باشد که هر کدام از آنها متناظر با یکی از بیت‌های خروجی ستون آخر جدول شکل ۱۹ باشد. مدار لازم برای هر کدام از این خروجی‌ها با ترکیب سطرهایی از جدول که در آنها این خروجی خاص ۱ شده است، بدست می-آید. به طور مثال بیت کم ارزش خطوط کنترلی ALU (Operation0) فقط در دو سطر آخر جدول شکل ۱۹ دارای مقدار ۱ می‌باشد بنابراین جدول صحت برای Operation0 فقط این دو سطر را خواهد داشت. شکل ۲۰ جدول صحت را برای هر سه بیت کنترلی ALU نشان می‌دهد. ما از ساختار مشترک در هر جدول صحت استفاده کردیم تا اینکه حالتهای بی اهمیت بیشتری داشته باشیم. به طور مثال، ۵ سطر از جدول شکل ۱۹ که خط خروجی Operation1 را ۱ می‌کنند، فقط به دو سطر در شکل ۲۰ کاهش پیدا کرده است. مطلبی که در اینجا لازم است ذکر شود این است که حالت‌های بی‌اهمیت در مدار به این منظور استفاده می‌شوند که تعداد گیت‌های مدار و تعداد ورودی‌های لازم برای هر گیت را کاهش دهند و باعث بهبود مدار شوند. با استفاده از جدول ساده شده شکل ۲۰، ما می‌توانیم مدار شکل ۲۱ را بدست بیاوریم. ما این مدار را واحد کنترل ALU نام‌گذاری می‌کنیم. همان طور که مشاهده شد، مدار کنترل ALU، به دلیل اینکه فقط سه خروجی تولید می‌کند و برای طراحی آن فقط تعداد کمی از سطرهای جدول مورد استفاده قرار می‌گیرد، مدار ساده‌ای است. اگر تعداد ورودی‌ها و خروجی‌ها بیشتر شود طراحی واحد کنترل نیز سخت‌تر می‌شود.

ALUOp		Funct فیلد					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
X	1	X	X	X	X	X	X
1	X	X	X	X	X	1	X

(الف) جدول درستی برای Operation2=1

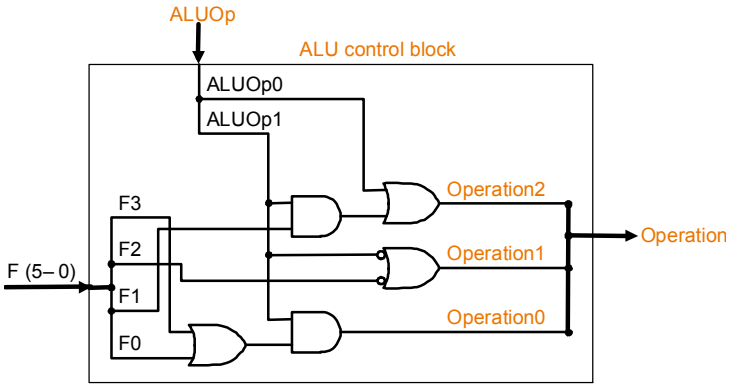
ALUOp		Funct فیلد					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
0	X	X	X	X	X	X	X
X	X	X	X	X	0	X	X

(ب) جدول درستی برای Operation1=1

ALUOp		Funct فیلد					
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0
1	X	X	X	X	X	X	1
1	X	X	X	1	X	X	X

(ج) جدول درستی برای Operation0=1

شکل ۲۰: جدول درستی برای بیت‌های کنترلی ALU



شکل ۲۱: مدار واحد کنترل ALU

۵-۴-۳- طراحی واحد کنترل اصلی

حال که طراحی واحد کنترل ALU را انجام دادیم، در این بخش قصد داریم که طراحی واحد کنترل اصلی که بقیه قسمتهای پردازنده را کنترل می‌کند را طراحی کنیم. برای انجام این کار اجازه دهید که فیلدهای دستور و خطوط کنترلی مسیر داده نهایی را که طراحی کردیم، دوباره بررسی کنیم. برای درک این مطلب که چگونه فیلدهای دستور را به خطوط کنترل مسیر داده وصل کنیم، بهتر است که سه فرمت دستور گفته شده برای سه نوع دستور را دوباره مرور کنیم. این سه فرمت در شکل ۲۲ نشان داده شده‌اند.

0	rs	rt	rd	shamt	funct
31-26	25-21	20-16	15-11	10-6	5-0

(الف) فرمت دستورات نوع R

35 or 43	rs	rt	address
31-26	25-21	20-16	15-0

(ب) فرمت دستورات Load و store

4	rs	rt	address
31-26	25-21	20-16	15-0

(ج) فرمت دستور beq

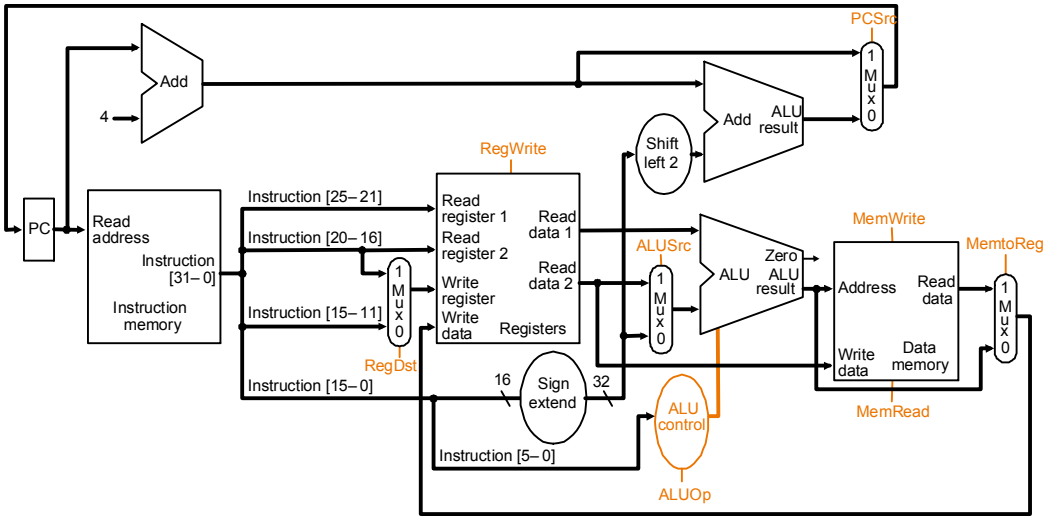
شکل ۲۲: فرمت دستورات نوع R (الف)، مراجعه به حافظه (ب) و beq (ج)

از این شکل می‌توان نکات زیر را استخراج نمود:

- فیلد OP که opcode نیز نامیده می‌شود همیشه در بیت‌های ۳۱ تا ۲۶ قرار دارد. ما به این فیلد تحت عنوان OP[5-0] مراجعه خواهیم کرد.
- دو رجیستری که باید خوانده شوند همیشه با فیلدهای rs و rt که در مکانهای ۲۵ تا ۲۱ و ۲۰ تا ۱۶ قرار دارند، مشخص می‌شود. این مطلب برای دستورهای نوع R، beq و sw درست است.
- رجیستر base برای دستورات lw و sw همیشه در مکانهای ۲۵ تا ۲۱ (rs) قرار می‌گیرد.
- عدد ثابت ۱۶ بیتی (offset) برای دستورات beq، lw و sw همیشه در مکانهای ۱۵ تا ۰ قرار داده می‌شود.

- رجیستر مقصد در یکی از دو محل قرار دارد: برای دستور lw رجیستر مقصد در مکانهای ۲۰ تا ۱۶ (rt) قرار دارد، در حالی که برای دستورهای نوع R در مکانهای ۱۵ تا ۱۱ (rd) قرار دارد. بنابراین ما نیاز داریم که از یک مالتی‌پلکسر استفاده کنیم تا مشخص کنیم که کدام فیلد دستور، شماره رجیستری را که نوشته خواهد شد، مشخص کند.

با استفاده از این اطلاعات ما می‌توانیم به مسیر داده کلی طراحی شده در شکل ۱۷، برچسب‌های دستور و یک مالتی‌پلکسر (به ورودی Write register از بانک رجیستر) اضافه کنیم. شکل ۲۳ این تغییرات را نشان می‌دهد. بعلاوه در این شکل واحد کنترل ALU، سیگنال write برای عناصر حالت، سیگنال read حافظه، و سیگنالهای کنترلی مالتی‌پلکسرها نیز نشان داده شده‌اند. به دلیل اینکه همهٔ مالتی‌پلکسرها دارای دو ورودی هستند، همهٔ آنها دارای یک خط کنترل (خط انتخاب) می‌باشند.



شکل ۲۳: مسیر داده‌ای که به آن واحد کنترل ALU، برچسب‌های دستور و یک مالتی پلکسر اضافه شده است

شکل ۲۳ دارای ۷ سیگنال کنترلی یک بیتی و یک سیگنال کنترلی دو بیتی به نام ALUOp می‌باشد ما قبلاً توضیح دادیم که سیگنال کنترلی ALUOp چگونه کار می‌کند. قبل از هر کاری بهتر است کار ۷ سیگنال کنترلی دیگر را توضیح بدهیم و پس از آن به نحوه مقدار گرفتن این سیگنالها به هنگام اجرای دستور بپردازیم. شکل ۲۴ کار این ۷ سیگنال کنترلی را توضیح داده است.

اسم سیگنال کنترلی	تأثیر آن وقتی که غیرفعال می‌شود	تأثیر آن وقتی که فعال می‌شود
RegDst	شماره رجیستر مقصد که به write register می‌رسد از فیلد	شماره رجیستر مقصد که به write register می‌رسد از فیلد

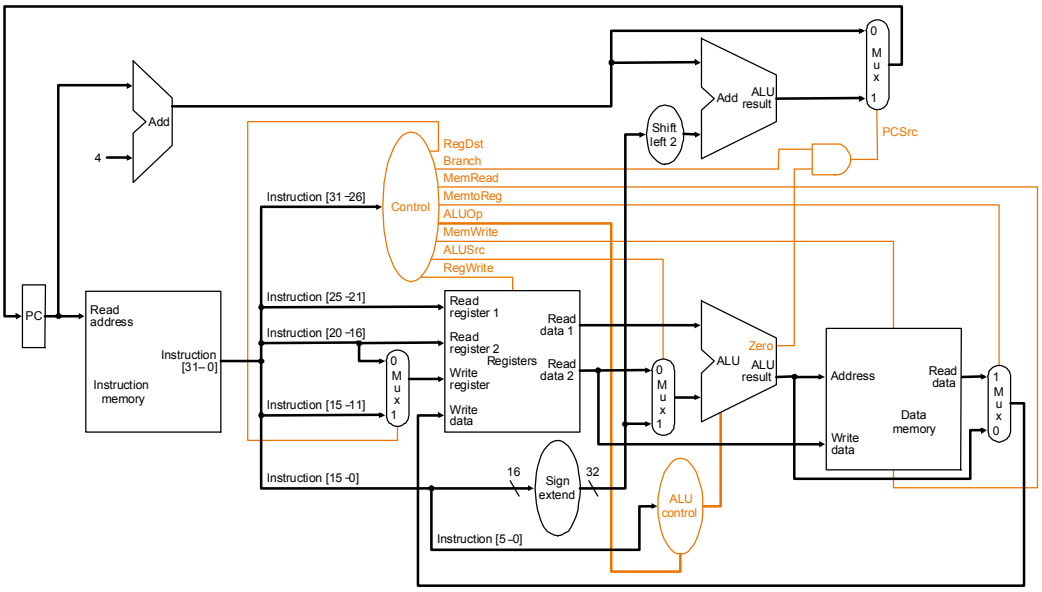
	rt (بیت‌های ۲۰ تا ۱۶) می‌آید	rd (بیت‌های ۱۵ تا ۱۱) می‌آید
RegWrite	هیچ کار (None)	در رجیستر مشخص شده با آدرس write register مقدار داده قرار گرفته بر روی خط write data نوشته می‌شود
ALUSrc	ورودی دوم ALU از خروجی دوم بانک رجیستر (Read data2) می‌آید	ورودی دوم ALU ، مقدار گسترش بیت علامت داده شده عدد ثابت ۱۶ بیتی است
PCSrc	خروجی جمع کننده‌ای که جمع PC+4 را انجام می‌دهد به PC منتقل می‌شود	خروجی جمع کننده‌ای که آدرس پرش را حساب می‌کند به PC منتقل می‌شود
MemRead	هیچ کار (None)	محتوای خانه‌ای از حافظه که آدرس آن هم‌اکنون بر روی خطوط آدرس قرار دارد، خوانده شده و بر روی خط خروجی حافظه قرار می‌گیرد
MemWrite	هیچ کار (None)	داده موجود بر روی خط write data حافظه در آدرس مشخص شده نوشته می‌شود
MemtoReg	داده‌ای که بر روی خط write data بانک رجیستر قرار می‌گیرد از ALU می‌آید	داده‌ای که بر روی خط write data بانک رجیستر قرار می‌گیرد از حافظه داده می‌آید

شکل ۲۴: تأثیر فعال شدن ۷ سیگنال کنترلی بر روی مسیر داده

حال که ما عملکرد همهٔ سیگنالهای کنترلی را مشاهده کردیم، می‌توانیم به نحوه مقداردهی آنها بپردازیم. واحد کنترل می‌تواند مقدار همهٔ این سیگنالهای کنترلی را بر اساس فیلد opcode دستور مشخص نماید. تنها استثنائی که در این زمینه وجود دارد سیگنال کنترلی PCSrc است. این سیگنال کنترلی باید زمانی یک شود که دستور beq اجرا شده و خروجی Zero از ALU که برای مقایسه

تساوی به کار می‌رود یک گردد. برای تولید سیگنال PCSrc ما نیاز داریم سیگنال Zero را با یک سیگنال از خروجی‌های واحد کنترل به نام Branch ، AND کنیم.

این ۹ سیگنال کنترلی (۷ سیگنال شکل ۲۴ و دو سیگنال ALUOp) می‌توانند بر اساس ۶ بیت ورودی واحد کنترل که همان ۶ بیت opcode است، مقداردهی شوند. شکل ۲۵ مسیر داده طراحی شده را به همراه واحد کنترل و سیگنالهای کنترلی نشان می‌دهد. قبل از اینکه ما تلاش کنیم که برای این کنترل معادله‌ای بنویسیم یا جدول صحتی تشکیل دهیم، بهتر است که عملیات کنترلی را توضیح دهیم. به دلیل اینکه مقداردهی خطوط کنترلی فقط به opcode وابسته است، ما بر اساس مقدار opcode تعیین می‌کنیم که مقدار سیگنالهای کنترلی باید ۰ ، ۱ یا X باشد. شکل ۲۶ مشخص می‌کند که چگونه سیگنالهای کنترلی بر اساس فیلد opcode مقدار می‌گیرند.



شکل ۲۵: مسیر داده ساده همرا با واحد کنترل

دستور	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
نوع R	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

شکل ۲۶: مقدار سیگنالهای کنترلی توسط فیلد opcode دستور مشخص می‌شود

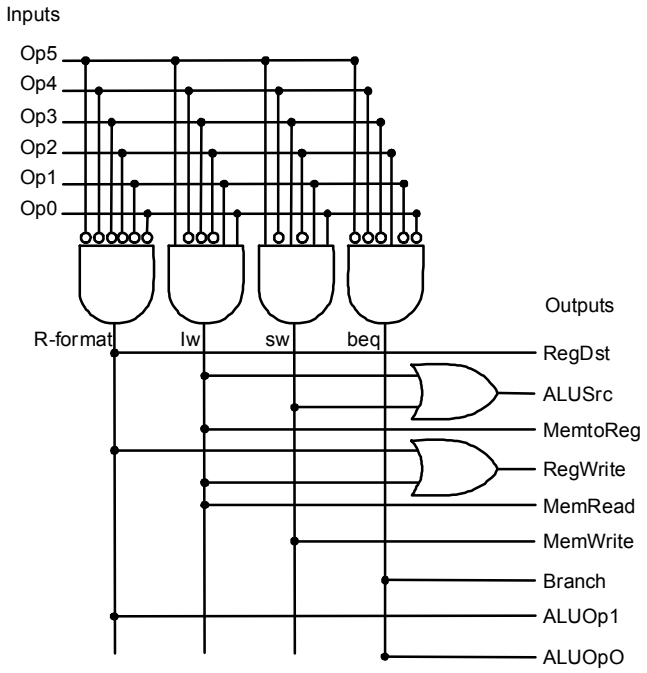
واحد کنترل می‌تواند با استفاده از محتویات شکل ۲۶ به طور دقیق طراحی شود. واحد کنترلی که ما طراحی می‌کنیم، مداری است که دارای ۶ بیت ورودی (فیلد opcode) بوده و در خروجی خود سیگنالهای کنترلی مورد نیاز را تولید می‌نماید. برای طراحی این مدار باید جدول صحت را برای هر کدام از خروجی‌ها تشکیل دهیم. قبل از رسم جدول صحت بهتر است که opcode دستورهای پیاده-سازی شده را شرح دهیم. جدول زیر مقدار فیلد opcode را برای هر کدام از دستورات هم به صورت دهدهی و هم به صورت باینری نشان می‌دهد:

اسم دستور	opcode دستور به صورت دهدهی	opcode دستور به صورت باینری					
		Op5	Op4	Op3	Op2	Op1	Op0
نوع R	0	0	0	0	0	0	0
lw	35	1	0	0	0	1	1
sw	43	1	0	1	0	1	1
beq	4	0	0	0	1	0	0

با استفاده از اطلاعات این جدول ما می‌توانیم عملکرد واحد کنترل را با یک جدول صحت بزرگ که در شکل ۲۷ نشان داده شده است، توضیح دهیم. این شکل عملکرد واحد کنترل را به طور کامل شرح می‌دهد و ما می‌توانیم با استفاده از آن مدار واحد کنترل را با گیت‌های منطقی طراحی کنیم. اگر از همان روشی که برای طراحی واحد کنترل ALU استفاده کردیم، در اینجا نیز استفاده کنیم، مدار شکل ۲۸ برای واحد کنترل اصلی بدست می‌آید. از آنجایی که هر کدام از خروجی‌ها فقط با استفاده از گیت‌های AND ، OR و به صورت دو طبقه طراحی شده‌اند، می‌توانیم در طراحی واحد کنترل اصلی از آرایه‌های منطقی قابل برنامه‌ریزی یا اصطلاحاً PLA استفاده کنیم. مدار شکل ۲۸ با استفاده از PLA طراحی شده است.

ورودی و خروجی	اسم سیگنال	نوع R	lw	sw	beq
ورودی‌ها	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
خروجی‌ها	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

شکل ۲۷: عملکرد واحد کنترل پردازنده single cycle که به طور کامل با جدول صحت نشان داده شده است



شکل ۲۸: مدار واحد کنترل پردازنده single cycle