فصل ۶

بهبود کارآیی به کمک پایپلاین کردن

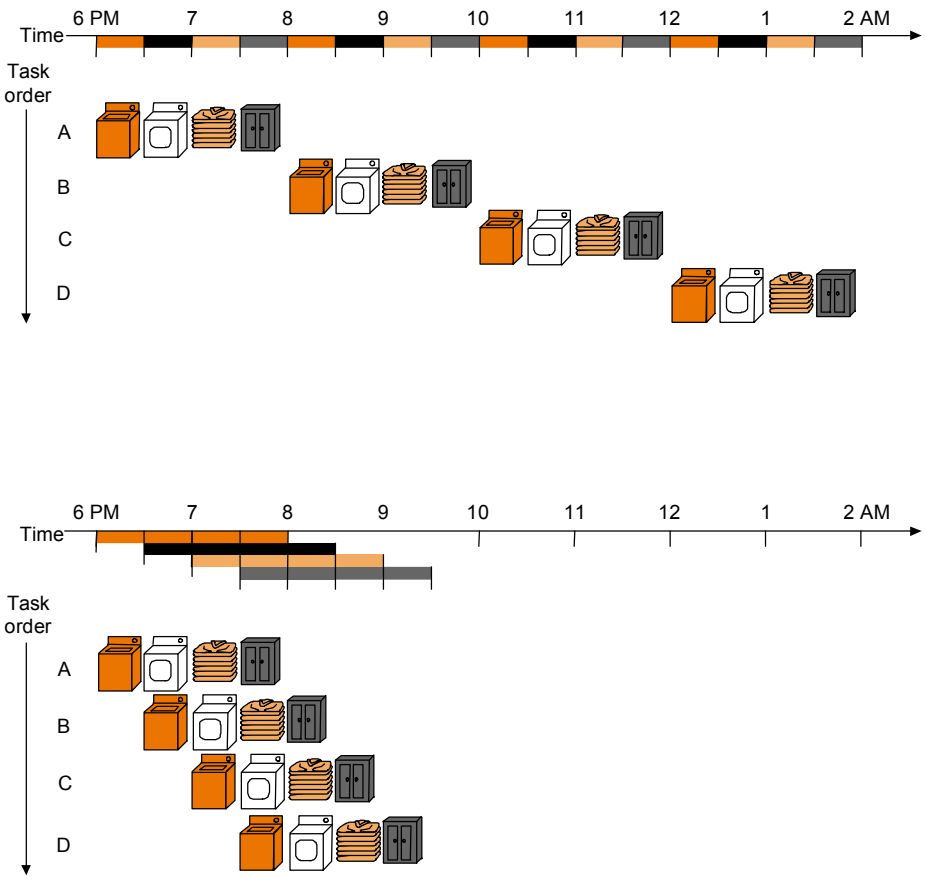
۶-۱- مروری بر تکنیک پایپلاین کردن

پایپلاین کردن یا pipelining تکنیکی است که در آن اجرای چند دستور با هم همپوشانی پیدا می‌کنند. امروزه پایپلاین کردن یک امر حیاتی برای افزایش سرعت پردازنده‌ها به حساب می‌آید. در این بخش برای توضیح اصطلاحات و مفاهیم پایپلاین کردن از یک مقایسه با دنیای بیرون استفاده خواهیم کرد. اگر شما علاقمند باشید که فقط یک نگاه کلی به پایپلاین داشته باشید که ببینید پایپلاین چگونه کار می‌کند و تأثیر آن بر کارآیی پردازنده چیست، در این صورت بهتر است که این بخش را مطالعه نموده و سپس بخش ۶-۸ را مطالعه نمایید. و اگر شما علاقمند باشید که یک کامپیوتر پایپلاین شده را کالبد شکافی کنید می‌توانید بعد از مطالعه این بخش، بخش‌های ۶-۲ تا ۶-۷ را مطالعه نمائید.

کسانی که در اتاق رختشوئی کار می‌کنند و تعداد زیادی لباس را شستشو می‌کنند، به طور ناخودآگاه از تکنیک پایپلاین کردن استفاده می‌کنند. یک روش غیر پایپلاین برای شستن این تعداد لباس به صورت زیر است:

۱. تعدادی لباس کثیف را داخل شوینده (washer) قرار دهید.
۲. بعد از تمام شدن عملیات شستن، لباس‌های خیس را داخل خشک کننده (dryer) قرار دهید.
۳. وقتی که کار خشک‌کننده تمام شد، لباس‌های خشک را بر روی یک میز قرار داده و تا کنید.
۴. وقتی که کار تا کردن تمام شد، لباس‌های تا شده را داخل کمد قرار دهید و یا از همکارتان بخواهید که آنها را بیرون ببرد.

وقتی که لباس‌های تا شده را داخل کمد قرار دادید، دوباره یک کار جدید را با شستن دسته دیگری از لباس‌های کثیف، شروع کنید. شکل ۱ طریقه شستن لباسها را هم به صورت غیرپایپلاین و هم به صورت پایپلاین شده نشان می‌دهد. همان طور که این شکل نشان می‌دهد، راهکار پایپلاین شده خیلی سریعتر و در زمان کمتری انجام می‌شود. در این روش، به محض اینکه شوینده شستن اولین دسته از لباس‌های کثیف را تمام کرد، لباس‌های شسته شده داخل خشک‌کن قرار گرفته و همزمان با آن دسته دیگری از لباس‌های کثیف داخل شوینده قرار داده می‌شوند. وقتی که دسته اول لباس‌ها خشک شد، آنها را بر روی میز قرار داده تا اینکه تا کنید همزمان با تا کردن این لباسها، دسته دوم لباسها را که داخل شوینده قرار داشته و شسته شده‌اند را به داخل خشک‌کن منتقل کنید و همچنین دسته سوم لباسها را در داخل شوینده که هم اکنون خالی شده است قرار دهید. وقتی که عمل تا کردن دسته اول لباسها تمام شد، آنها را داخل کمد بچینید و یا به همکارتان بگوئید آنها را به بیرون منتقل کند. همزمان با چیدن این لباسها دسته دوم لباسها را که خشک شده‌اند به روی میز منتقل کنید تا اینکه تا شوند، حال که خشک‌کن خالی شده است دسته سوم لباسها را که هم اکنون داخل شوینده قرار دارند و شسته شده‌اند به داخل خشک‌کن منتقل کنید تا خشک شوند. همچنین دسته چهارم لباسها را داخل شوینده که هم اکنون خالی است منتقل کنید تا شسته شوند. در این لحظه همهٔ گامها (Steps) که مراحل (stager) پایپلاین نیز نامیده می‌شوند، به طور همزمان (concurrent) در حال کار می‌باشند. تا زمانیکه منابع جداگانه برای هر مرحله داشته باشیم، می‌توانیم کارها را به صورت پایپلاین انجام دهیم.



شکل ۱: قیاس اتاق رختشوئی برای پایپلاین کردن. قسمت بالای شکل حالت غیر پایپلاین و قسمت پایین شکل حالت پایپلاین را نشان می‌دهد. با فرض اینکه شستن یک دست لباس را در چهار مرحله انجام دهیم و هر کدام از مراحل ۳۰ دقیقه طول بکشند، در این صورت برای شستن ۴ دست لباس در حالت غیر پایپلاین، ۸ ساعت و در حالت پایپلاین شده ۳ ساعت و ۳۰ دقیقه زمان لازم است. ما مراحل پایپلاین را در یک دیاگرام زمانی دو بعدی، در طول زمان به این صورت نشان داده‌ایم که برای هر دست لباس، منابع مورد نیاز را تکرار کرده‌ایم اما می‌دانیم که در واقعیت از هر منبعی فقط یک نسخه در اختیار داریم. به چنین دیاگرامی، دیاگرام پایپلاین گفته می‌شود.

نکته‌ای که در مورد پایپلاین باید متذکر شویم این است که: زمان بین قرار دادن یک جوراب تکی کثیف داخل شوینده تا خشک شدن، تا شدن و قرار دادن آن داخل کمد، در پایپلاین کردن کاهش پیدا نمی‌کند. دلیل اینکه پایپلاین سریعتر است این است که برای تعداد زیادی لباسهای کثیف، می‌توان کارهای مختلف را به صورت موازی انجام داد و در نتیجه تعداد زیادی لباس را در واحد زمان شستشو داد.

اگر همهٔ مراحل پایپلاین به یک اندازه طول بکشند و تعداد زیادی کار برای انجام وجود داشته باشد، در این صورت میزان افزایش سرعت (speedup) در پایپلاین کردن نسبت به حالت غیر پایپلاین مساوی با تعداد مراحل پایپلاین خواهد بود.

روش پایپلاین شده برای شستن لباسها به دلیل داشتن ۴ مرحله، پتانسیل این را دارد که ۴ مرتبه سریعتر از حالت غیر پایپلاین شده عمل کند. به طور مثال اگر ۲۰ دسته لباس داشته باشیم، زمان لازم جهت عملیات شستشوی آنها در حالت پایپلاین شده ۵ برابر زمان لازم برای شستشوی یک دسته لباس است و این در حالی است که در حالت غیر

پایپلاین شده زمان لازم برای شستشوی ۲۰ دسته لباس، ۲۰ برابر زمان لازم برای شستشوی یک دسته لباس خواهد بود. اما همان طور که در شکل ۱ نشان داده شده است، شستن ۴ دسته لباس، در حالت پایپلاین نشده ۸ ساعت طول کشیده است ولی در حالت پایپلاین شده سه ساعت ونیم طول کشیده است. بنابراین میزان افزایش سرعت مساوی با speedup 2.3 = 8/3.5 = خواهد بود. دلیل اینکه میزان افزایش سرعت ۴ برابر نشده، این است که تعداد دسته لباسهای کثیف کم است. اگر تعداد دسته لباسهای کثیف به سمت بی‌نهایت میل کند، در این صورت میزان افزایش سرعت مساوی با ۴ خواهد بود.

مفاهیم مشابهی در ارتباط با پردازنده‌ها نیز می‌تواند اعمال گردد. در پردازنده‌ها نیز اجرای دستورات می‌تواند به صورت پایپلاین شده انجام شود. دستورات MIPS به طور کلاسیک در ۵ گام انجام می‌شوند:

۱. واکشی یا خواندن دستور از حافظه دستورات (فاز fetch)
۲. خواندن رجیسترها و دیکد کردن دستور (فرمت دستورات MIPS این اجازه را می‌دهد که خواندن و دیکد کردن به صورت همزمان انجام شوند.) (فاز decode)
۳. اجرای عملیات یا محاسبه یک آدرس (فاز execute)
۴. دسترسی به یک عملوند در حافظه داده‌ها (فاز memory)
۵. نوشتن نتیجه در یک رجیستر (فاز writeback)

بنابراین پایپلاین MIPS که ما در این فصل آن را بررسی خواهیم کرد دارای ۵ مرحله است. مثال زیر نشان می-دهد که پایپلاین کردن سرعت اجرای دستورات را افزایش می‌دهد.

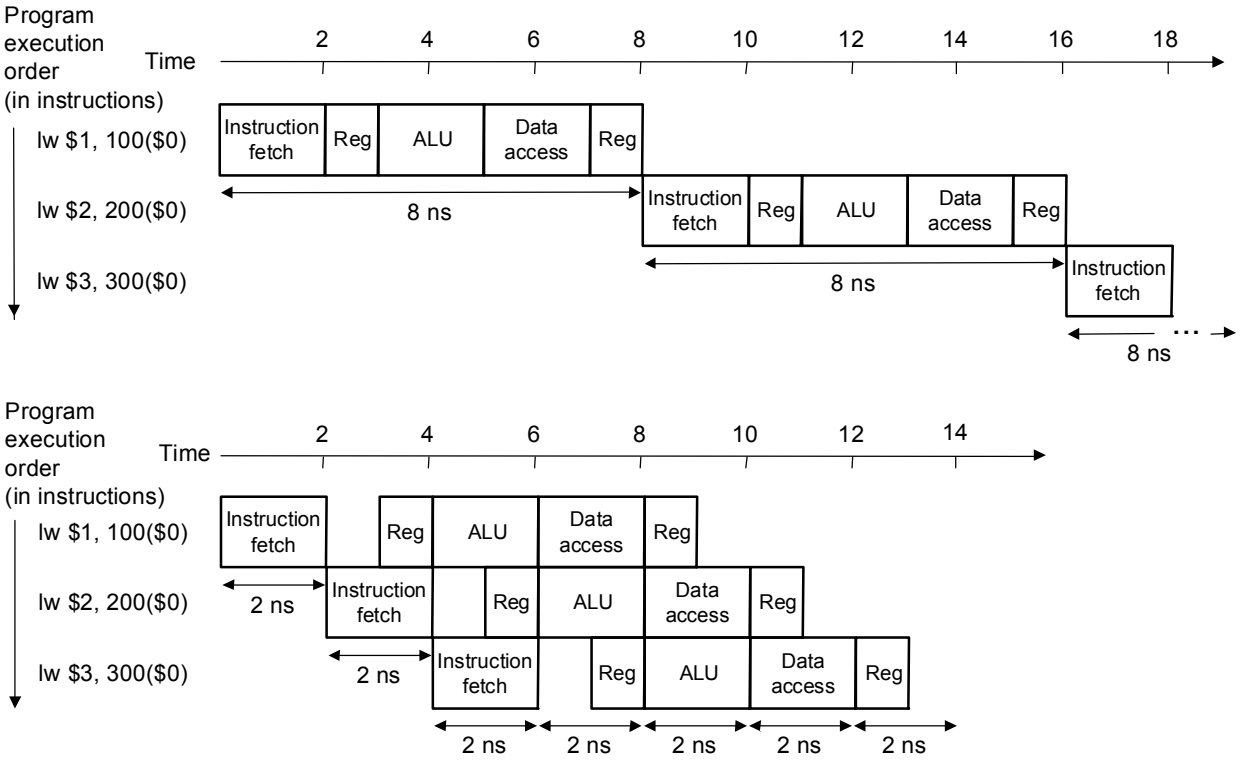
مثال: مقایسه کارایی پردازنده single cycle و پردازنده پایپلاین شده

متوسط زمان بین اجرای دو دستور در پردازنده single cycle که در آن هر دستور در یک کلاک انجام می‌شود را نسبت به پیاده‌سازی پایپلاین شده مقایسه کنید. زمان اجرا برای واحدهای عملیاتی در این مثال به این صورت است: ۲ نانوثانیه برای دسترسی به حافظه، ۲ نانوثانیه برای عملیات ALU، و ۱ نانوثانیه برای خواندن یا نوشتن بانک رجیستر. (همان طور که در فصل ۵ گفته شد، در مدل single cycle، هر دستوری در یک کلاک انجام می‌شود، بنابراین پریود کلاک باید طوری انتخاب شود که مطابق با کندترین دستور باشد.)

جواب: برای واضح‌تر شدن بحث باید یک پایپلاین ایجاد کنیم. در این مثال و برای قسمتهای باقیمانده این فصل، ما توجه خود را به ۸ دستور معطوف خواهیم کرد: lw , sw , add , sub , and , or , slt , beq. زمان لازم برای هر کدام از ۸ دستور در شکل ۲ نشان داده شده است. پریود کلاک پردازنده single cycle باید طوری باشد که کندترین دستور بتواند در یک کلاک اجرا شود. همان طور که در شکل ۲ دیده می‌شود، کندترین دستور، دستور lw است که زمان اجرای آن ۸ نانوثانیه است. بنابراین زمان اجرای بقیه دستورات نیز ۸ نانوثانیه خواهد بود، به دلیل اینکه هر دستور در یک کلاک انجام می‌شود و پریود کلاک در این مثال مساوی ۸ نانوثانیه (T=8ns) می‌باشد.

کلاس دستور	خوانده از حافظه دستورات	خواندن رجیستر	عملیات ALU	دسترسی به حافظه داده‌ها	نوشتن در رجیستر	زمان کل
Load word (lw)	2 ns	1 ns	2 ns	2 ns	1 ns	8 ns
Store word (sw)	2 ns	1 ns	2 ns	2 ns		7 ns
R-format (add ,sub, and, or, slt)	2 ns	1 ns	2 ns		1 ns	6 ns
Branch (beq)	2 ns	1 ns	2 ns			5 ns

شکل ۲: زمان لازم برای اجرای هر کدام از ۸ دستور که از زمان لازم برای هر کدام از ماژولهای سخت افزاری بدست آمده است. در این محاسبات فرض شده است که تأخیر مالتی‌پلکسرها، واحد کنترل، دسترسی به PC و واحد گسترش بیت علامت، صفر است.



شکل ۳: مقایسه پردازنده غیر پایپلاین (قسمت بالای شکل) و پردازنده پایپلاین شده (قسمت پایین شکل). در این شکل فرض شده است که نوشتن در بانک رجیستر، در نیم کلاک اول و خواندن از بانک رجیستر در نیم کلاک دوم انجام می‌گیرد به همین دلیل قسمت مربوط به بانک رجیستر به صورت نصفه نشان داده شده است.

مشابه با شکل ۱، شکل ۳ اجرای ۳ دستور lw را به صورت پایپلاین نشده و پایپلاین شده مقایسه می‌نماید. زمان اجرای بین اولین و چهارمین دستور در طراحی غیر پایپلاین مساوی $3\times 8ns$ یا ۲۴ نانو ثانیه است.

هر مرحله از پایپلاین یک کلاک طول می‌کشد، بنابراین پریود کلاک باید به اندازه کافی طولانی باشد که مطابق با کندترین مرحله باشد. مشابه با پردازنده single cycle که پریود کلاک باید مساوی ۸ نانوثانیه می‌شد در حالی که بعضی از دستورات می‌توانستند سریعتر هم اجرا شوند (به طور مثال دستور beq در ۵ نانوثانیه هم می‌توانست اجرا شود)، پردازنده پایپلاین شده هم پریود کلاکش باید حداقل مساوی بدترین تأخیر در مراحل پایپلاین باشد که در اینجا ۲ نانوثانیه است، هرچند که بعضی مراحل در کمتر از ۲ نانو ثانیه هم اجرا می‌شوند (به طور مثال مراحل خواندن یا نوشتن بانک رجیستر در ۱ نانوثانیه هم قابل اجرا هستند). زمان اجرای بین اولین و چهارمین دستور در پردازنده پایپلاین شده مساوی با $3 \times 2ns$ یا ۶ نانوثانیه است. اگر با حالت غیر پایپلاین مقایسه کنیم (۲۴ نانوثانیه) می‌بینیم که کارآیی پردازنده پایپلاین به غیر پایپلاین مساوی ۴ خواهد بود. (یعنی ۴ مرتبه افزایش سرعت داشته‌ایم). می‌توانیم این میزان افزایش سرعت را به این صورت هم بیان کنیم که در پردازنده single cycle هر دستور در ۸ نانوثانیه اجرا می‌شود و حال آنکه در پردازنده پایپلاین شده در ۲ نانوثانیه، بنابراین پردازنده پایپلاین شده ۴ برابر سریعتر است.

ما می‌توانیم میزان افزایش سرعت (speedup) پردازنده پایپلاین شده به پردازنده غیر پایپلاین را که در بالا صحبت کردیم در قالب فرمول بیان کنیم. اگر همهٔ مراحل کاملاً بالانس باشند (تأخیرشان یکسان باشد)، در این صورت زمان بین دستورات در پردازنده پایپلاین شده (با فرض شرایط ایده‌آل) برابر خواهد بود با:

$$\text{زمان بین دستورات در پردازنده پایپلاین شده} = \frac{\text{زمان بین دستورات در پردازنده غیر پایپلاین}}{\text{تعداد مراحل پایپلاین}}$$

و یا

$$\text{زمان اجرای یک دستور در پردازنده پایپلاین شده} = \frac{\text{زمان اجرای یک دستور در پردازنده غیر پایپلاین}}{\text{تعداد مراحل پایپلاین}}$$

در شرایط ایده‌آل میزان افزایش سرعت پایپلاین کردن مساوی تعداد مراحل پایپلاین خواهد بود، یک پایپلاین ۵ مرحله‌ای ۵ برابر سریعتر خواهد بود. این فرمول بیان می‌کند که یک پایپلاین ۵ مرحله‌ای برای پردازنده باید زمان اجرای ۸ نانوثانیه‌ای پردازندهٔ غیر پایپلاین را به اندازه ۵ برابر بهبود دهد، به عبارتی پریود کلاک پردازنده پایپلاین شده مساوی ۱/۶ نانوثانیه باشد. اما مثالی که توضیح داده شد، نشان می‌دهد که مراحل پایپلاین ممکن است بالانس نباشند. بعلاوه پایپلاین کردن مقداری سربار (overhead) نیز دارد. بنابراین زمان اجرای هر دستور در پردازنده پایپلاین شده از کمترین میزان ممکن (۱/۶ نانوثانیه در این مثال) بیشتر خواهد شد و میزان افزایش سرعت کمتر از تعداد مراحل پایپلاین خواهد شد.

بعلاوه ادعای بهبود ۴ برابری در مثالی که بیان شد، در زمان اجرای کل ۳ دستور نیز مشاهده نمی‌شود: در پردازنده پایپلاین شده زمان اجرای سه دستور مساوی ۱۴ نانوثانیه است و در غیر پایپلاین مساوی ۲۴ نانوثانیه. برای اینکه ببینیم زمان اجرای کل زیاد اهمیت ندارد، بیایید حالتی را در نظر بگیریم که تعداد دستورات خود را افزایش می‌دهیم. فرض کنید که شکل ۳ را به ۱۰۰۳ دستور گسترش دهیم. وقتی که ما ۱۰۰۰ دستور به مثال پایپلاین اضافه می‌کنیم

هر دستوری ۲ نانوثانیه به زمان کل اضافه خواهد کرد. بنابراین زمان اجرای کل برای حالت پایپلاین شده مساوی خواهد بود با $1000 \times 2n + 14\,ns$ یا ۲۰۱۴ نانوثانیه.

در مثال غیر پایپلاین، ۱۰۰۰ دستور اضافه می‌کنیم که هر کدام ۸ نانوثانیه طول می‌کشند، بنابراین زمان اجرای کل مساوی خواهد بود با $1000 \times 8n + 24n$ یا ۸۰۲۴ نانوثانیه. اگر کارآیی پردازنده پایپلاین شده را به غیر پایپلاین حساب کنیم چنین بدست می‌آید:

$$\frac{8024ns}{2014ns} = 3.98 \approx 4 = \frac{8ns}{2ns}$$

به عبارتی نسبت زمانهای اجرای کل نزدیک به نسبت زمان بین دو دستور است.

نکته: پایپلاین کردن زمان اجرای تک تک دستورات را کاهش نمی‌دهد بلکه تعداد دستورات انجام شده در واحد زمان را افزایش می‌دهد. به عبارتی پایپلاین کردن با افزایش تعداد دستورات انجام شده در واحد زمان کارآیی را بهبود می‌دهد. امروز برنامه‌های واقعی دارای میلیاردها دستور می‌باشند، بنابراین throughput در این برنامه‌ها معیار بسیار مهمی به حساب می‌آید.

۶-۱-۱- طراحی مجموعه دستورات برای پایپلاین کردن

در این قسمت یک نگاه کلی به معماری مجموعه دستورات پردازنده MIPS خواهیم داشت و خواهیم دید که چگونه این معماری برای پایپلاین کردن مناسب می‌باشد.

اول اینکه، همهٔ دستورات MIPS دارای طول ثابت هستند. این محدودیت باعث می‌شود که واکشی دستورات در مرحله اول پایپلاین و دیکد کردن آن در مرحله دوم پایپلاین بسیار ساده‌تر شود. در مجموعه دستوراتی نظیر $80x86$ که در آن طول دستورات از یک تا ۱۷ بایت متغیر است، پایپلاین کردن کار واقعاً مشکلی است.

دوم اینکه، MIPS تعداد فرمت‌های کمتری دارد که در آنها فیلدهای مربوط به رجیسترهای مبدأ (source)، در یک مکان یکسان از دستور قرار دارند. این تقارن باعث می‌شود که در مرحله دوم پایپلاین در همان زمانی که دستور دیکد می‌شود، رجیسترهای مورد نظر نیز از بانک رجیستر خوانده شوند. اگر این تقارن و تشابه وجود نداشت، ما مجبود بودیم که مرحله دو را به دو مرحله جداگانه بشکنیم که باعث می‌شد پایپلاین ۶ مرحله‌ای داشته باشیم. (ما در آینده اشاره‌ای مختصر به مضرات پایپلاین‌های طولانی‌تر خواهیم داشت.)

سوم اینکه، عملوندهای حافظه درMIPS فقط در دستورات Load و Store ظاهر می‌شوند. این محدودیت به این معنی است که ما می‌توانیم از مرحلهٔ execute (مرحلهٔ سوم) برای محاسبه آدرس حافظه استفاده کنیم و در مرحله بعدی به حافظه دسترسی داشته باشیم. اگر ما به مانند $80x86$ می‌توانستیم بر روی عملوندهای حافظه عملیات انجام دهیم، در این صورت مرحله ۳ و۴ باید شامل ۳ مرحله محاسبه آدرس، دسترسی به حافظه و اجرای عملیات می‌شد.

چهارم اینکه، اپراندها در حافظه باید به صورت تراز شده باشند. بنابراین برای هر عملیات نوشتن یا خواندن حافظه فقط یک دسترسی به حافظه لازم است. بنابراین داده مورد نظر می‌تواند در یک مرحله از پایپلاین بین حافظه و پردازنده منتقل شود.

۶-۱-۲- مخاطرات پایپلاین

در پایپلاین ممکن است شرایطی به وجود آید که دستور بعدی نتواند در کلاک بعدی اجرا شود. این شرایط مخاطره یا hazard نامیده می‌شود. مخاطرات پایپلاین به سه دسته تقسیم می‌شوند:

۱. مخاطرات ساختاری یا structural hazards
۲. مخاطرات کنترلی یا control hazards
۳. مخاطرات داده‌ای یا data hazards

ما سه نوع مخاطره را در ابتدا در مثال اتاق رختشوئی بررسی خواهیم کرد و سپس مسأله معادل آن را در کامپیوتر مطرح نموده و برای رفع کردن آن راه حل ارائه خواهیم کرد.

۶-۱-۳- مخاطرات ساختاری

اولین مخاطره، مخاطره ساختاری نامیده می‌شود و به این معنی است که سخت‌افزار نمی‌تواند ترکیبی از دستورات را که می‌خواهیم اجرا کنیم، پشتیبانی نماید. یک مخاطره ساختاری در اتاق رختشویی موقعی می‌تواند به وجود آید که ما فقط از یک ماشین برای ترکیب عملیات شستن و خشک‌کردن استفاده کنیم و دو ماشین جداگانه برای این دو کار نداشته باشیم، یا اینکه همکار ما سرش شلوغ باشد و مشغول انجام کاری باشد و نتواند لباس‌های تاکرده را به بیرون ببرد. در این حالت زمانبندی دقیق پایپلاین ما با شکست مواجه خواهد شد.

همان طور که در بالا نیز توضیح داده شد، مجموعه دستورات پردازنده MIPS طوری طراحی شده‌اند که مناسب پایپلاین کردن باشند و باعث شده‌اند که طراحان به هنگام طراحی پایپلاین به راحتی از مخاطره ساختاری جلوگیری کنند. فرض کنید ما به جای دو حافظه جداگانه برای دستورات و داده‌ها، فقط یک حافظه می‌داشتیم. در این صورت اگر پایپلاین شکل ۳ دستور چهارمی هم داشت که می‌خواست اجرا شود، آن وقت می‌دیدیم که در یک پریود کلاک، دستور اول به حافظه جهت خواندن یا نوشتن داده‌ها مراجعه می‌نمود و دستور چهارمی هم به حافظه جهت واکشی دستور مراجعه می‌کرد. حال آنکه ما نمی‌توانیم در یک زمان دو مراجعه به حافظه انجام دهیم و بنابراین دستور چهارمی نمی‌تواند اجرا شود و باید یک کلاک منتظر بماند تا کار دستور اول تمام شود. بنابراین بدون داشتن دو حافظه جداگانه، پایپلاین ما دارای مخاطره ساختاری می‌شد.

۶-۱-۴- مخاطره کنترلی

مخاطره دوم، مخاطره کنترلی نامیده می‌شود و از آنجا ناشی می‌شود که بر اساس نتایج یک دستور تصمیم‌گیری می‌کنیم (دستور پرش شرطی) در حالی که دستورات دیگری نیز در داخل پایپلاین در حال اجرا هستند.

فرض کنید به خدمه اتاق رختشوئی کار شستوشوی لباس‌های یک تیم فوتبال محول شده است. بسته به اینکه لباس‌ها چقدرکثیف باشند، لازم است میزان پودر یا درجهٔ حرارت آب را طوری تنظیم کنیم تا اینکه اولاً لباس‌ها به خوبی تمیز شوند و ثانیاً لباس‌ها آسیب نبینند. در پایپلاین اتاق رختشوئی برای اینکه به فرمول صحیح دست پیدا کنیم، ما باید تا آخر مرحله ۲ صبر کنیم تا اینکه لباس‌های خشک را بررسی نموده و تشخیص دهیم که آیا لازم است تنظیمات شوینده را عوض کنیم یا نه. به عبارتی تصمیم بگیریم که برای آینده چکار کنیم؟

در اینجا دو راه حل برای رفع مخاطره کنترلی در اتاق رختشوئی و دو راه حل معادل در کامپیوتر ارائه می‌کنیم.

۶-۱-۴-۱- متوقف کردن یا Stall دادن

برای رفع مخاطره کنترلی در پایپلاین اتاق رختشوئی به این صورت عمل کنید: اولین دسته لباسها را وارد شوینده کنید و بعد از اینکه شسته شدند آنها را وارد خشک کننده کنید تا خشک شوند. وقتی که این دسته از لباسها در حال خشک شدن هستند، دسته دیگری از لباسها را به داخل شوینده وارد نکنید و این کار را متوقف کنید تا زمانیکه دسته اول لباسها خشک شوند. لباس‌های خشک را چک کنید و یک فرمول جدید پیشنهاد کنید. دسته دوم لباسها را وارد شوینده کنید و با فرمول جدید آنها را بشویید و سپس خشک کرده و بررسی کنید و فرمول جدیدی ارائه کنید. این کار را آنقدر ادامه دهید تا اینکه فرمول درست را بدست آورید. این یک راه‌حل ملاحظه کارانه است و قطعاً جواب می‌دهد ولی سرعت آن پایین است.

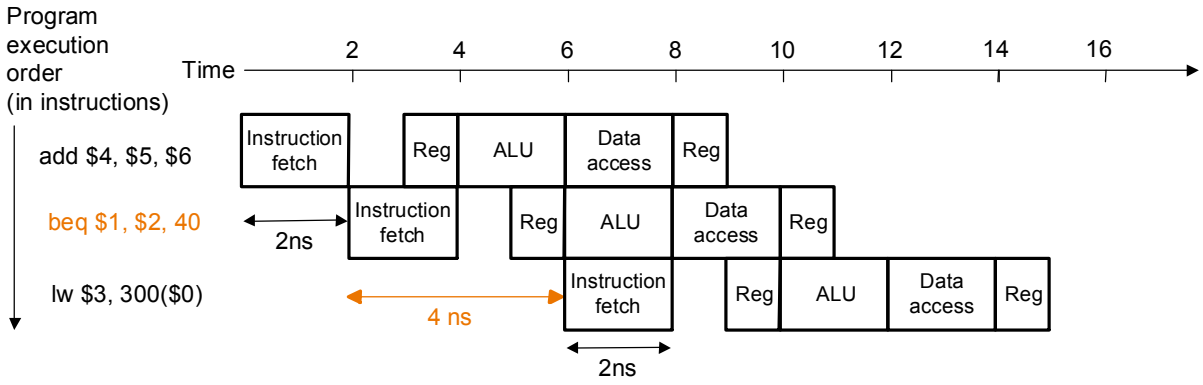
کار تصمیم گیری معادل عملیات فوق در یک کامپیوتر دستور پرش شرطی می‌باشد. وقتی که یک دستور پرش اجرا می‌شود ممکن است پرش را انجام دهد (branch taken) و یا اینکه انجام ندهد (branch not taken). تصمیم اینکه پرش انجام بشود یا نشود بستگی به برقرار شدن یا نشدن شرط پرش دارد. طبق توضیحات پردازنده‌ای که در فصل ۵ طراحی شد، عمل مقایسه و بررسی شرط دستور پرش در مرحله ۳ یا مرحله execute انجام می‌شود. اگر در آخر فاز execute تصمیم گرفته شود که پرش انجام بشود در این صورت دستورات پشت سر دستور پرش نباید اجرا شوند و دستور بعدی باید از آدرس مقصد پرش اجرا شود. به عبارتی اگر پرش انجام بشود و دستورات پشت سر آن وارد پایپلاین شده باشند، باید از پایپلاین خارج شوند و دور ریخته شوند تا اینکه جلوی اجرای آنها گرفته شود. به عمل پاک کردن و دور ریختن دستورات از پایپلاین flush کردن پایپلاین گفته می‌شود.

اگر یک کامپیوتر بخواهد منتظر نتیجه دستور پرش بماند و اجرای دستورات پشت سر آن را متوقف نماید، در این صورت هر موقع که دستور پرش شرطی وارد پایپلاین می‌شود، باید جلوی ورود دستورات پشت سر آن به داخل پایپلاین تا پایان مشخص شدن تصمیم پرش گرفته شود. به عمل جلوگیری از اجرای دستورات در پایپلاین، stall دادن یا توقف پایپلاین گفته می‌شود.

فرض کنید که ما مقداری سخت‌افزار اضافه قرار دهیم تا اینکه بتوانیم در مرحلهٔ ۲ رجیسترها را تست نموده، آدرس پرش را حساب کرده و شمارنده برنامه را بر اساس نتیجه تست مقداردهی کنیم. حتی با وجود این سخت‌افزار اضافه، پایپلاین شامل دستور پرش شرطی، شبیه به شکل ۴ خواهد شد. دستور lw که پس از دستور beq قرار دارد، اگر

پرش انجام نشود، اجرا خواهد شد. این دستور lw وارد پایپلاین نمی‌شود و متوقف می‌ماند تا زمانیکه نتیجه beq مشخص گردد. بنابراین دستور lw بعد از beq باید تا اندازه‌ای صبر کند که دستور beq به آخر مرحله ۲ برسد و پس از آن وارد پایپلاین می‌شود.

در شکل ۴ پایپلاین به اندازه یک گام stall می‌خورد و متوقف می‌ماند به stall های پایپلاین، اغلب bubble یا حباب نیز گفته می‌شود.



شکل ۴: متوقف کردن پایپلاین با اجرای دستور پرش شرطی به عنوان راه حلی برای مخاطره کنترلی

مثال: کارآیی توقف پایپلاین به هنگام اجرا شدن دستور شرطی

فرض کنید ۱۷ درصد دستورات یک برنامه، دستورات پرش شرطی باشند. با فرض اینکه تصمیم گیری دستور پرش در مرحله دوم پایپلاین صورت گیرد، تأثیر توقف پایپلاین به هنگام اجرای دستورات پرش شرطی را بر روی CPI بدست آورید. فرض کنید همهٔ دستورات با فرض اینکه اصلاً توقف در پایپلاین نداشته باشیم یک کلاک طول بکشند.

جواب: چون CPI دستورات دیگر مساوی یک می‌باشد و دستورات branch با متوقف‌کردن پایپلاین یک کلاک بیشتر اضافه می‌کنند، CPI جدید به صورت زیر بدست می‌آید.

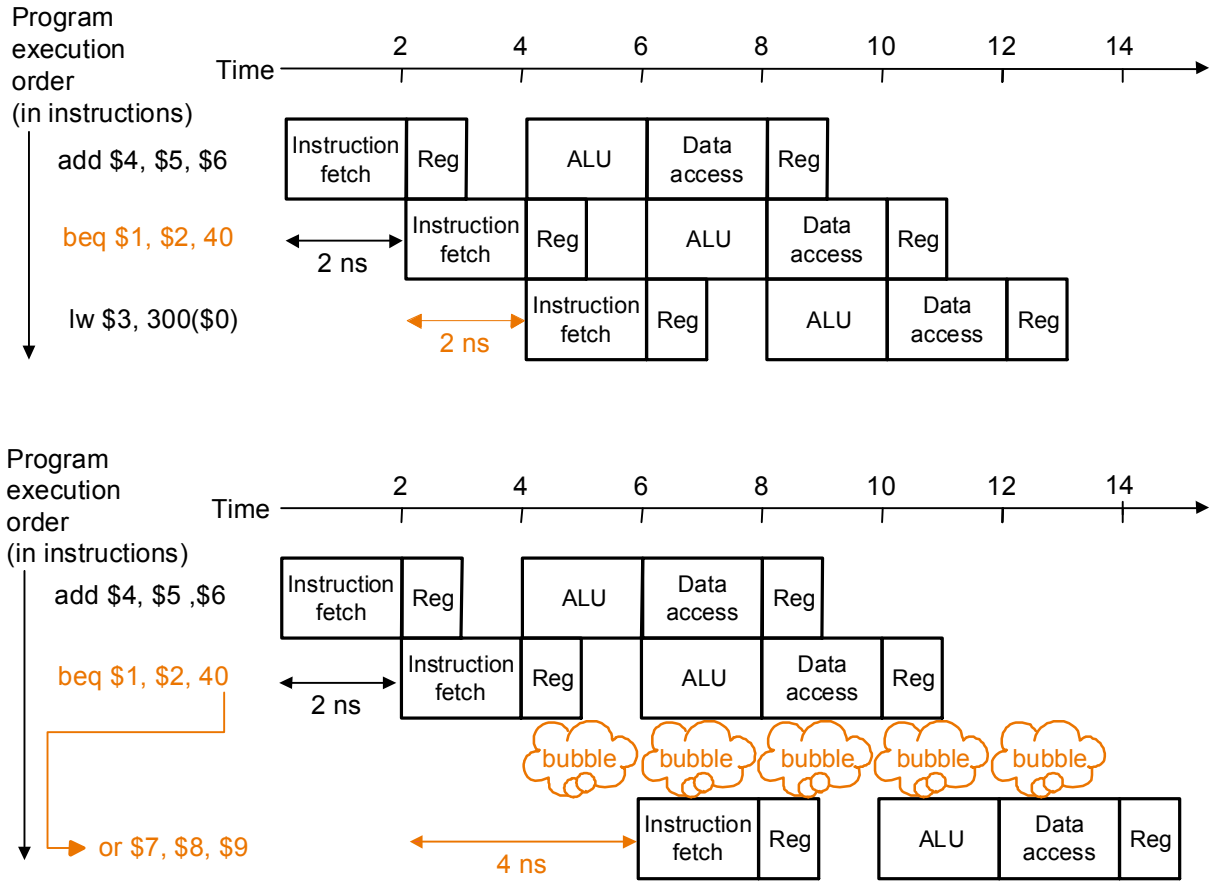
$$CPI_{new} = \frac{\sum C_i \times CPI}{I} = \frac{0.83 \times I \times 1 + 0.17 \times I \times 2}{I} = 1.17$$

CPI که از یک به ۱/۱۷ تغییر پیدا کرده باعث می‌شود که سرعت پردازنده به اندازه ۱/۱۷ مرتبه کندتر شود. اگر ما نتوانیم که تصمیم مربوط به دستور پرش را در مرحله ۲ اتخاذ کنیم (این امر اغلب در پایپلاین‌های طولانی‌تر اتفاق می‌افتد)، در این صورت با stall دادن در دستورات پرش شرطی سرعت پردازنده بیشتر پایین خواهد آمد. هزینه این کار برای اکثر کامپیوترها بالا است و بنابراین ما باید دنبال یک راه‌حل دیگر برای رفع مخاطره کنترلی باشیم. بنابراین دومین راه‌حل که پیش‌بینی (Predict) است به صورت زیر ارائه می‌شود.

۶-۱-۴-۲- پیش بینی پرش یا Branch prediction

اگر شما کاملاً مطمئن باشید که فرمول مناسب برای شستن لباسهای تیم فوتبال را می‌دانید، سپس پیش‌بینی کنید که این فرمول به دسته دوم لباسها نیز قابل اعمال است، در این صورت به هنگامی که دسته اول لباسها در حال خشک شدن هستند، دسته دوم لباسها را وارد شوینده کرده و با فرمول پیش‌بینی شده آنها را خواهید شست. در این راه حل اگر شما درست پیش‌بینی کرده باشید، پایپلاین با کندی مواجه نخواهد شد و سرعت آن پایین نخواهد آمد. ولی اگر اشتباه کرده باشید، شما باید دوباره لباسهای شسته شده را، با فرمول درست بشوئید.

کامپیوترها نیز از پیش‌بینی کردن برای دستورات پرش شرطی استفاده می‌کنند. یک راهکار ساده این است که همیشه فرض کنیم پرش انجام نخواهد شد. اگر درست گفته باشیم در این صورت پایپلاین با حداکثر سرعت خود اجرا خواهد شد. در این راهکار پایپلاین فقط زمانی متوقف می‌شود که پرش انجام شود (branch taken). شکل ۵ یک مثال برای راه‌حل پیش‌بینی انجام نشدن پرش ارائه نموده است. قسمت بالایی شکل حالتی را نشان می‌دهد که پیش‌بینی درست بوده و پرش انجام نشده است ولی قسمت پایین شکل برای حالتی است که پیش‌بینی غلط بوده و پرش انجام شده است.



شکل ۵: راه حل پیش بینی انجام نشدن پرش

یک نوع پیش‌بینی خیلی هوشمندانه می‌تواند به این صورت باشد که بعضی از پرش‌های شرطی را فرض کنیم که انجام می‌شوند (Branch taken) و بعضی‌ها را فرض کنیم که انجام نمی‌شوند(branch untaken). در مقام مقایسه با

پایپلاین اتاق رختشوئی، ما می‌توانیم برای لباس‌های تیره یک فرمول داشته باشیم و برای لباهای روشن یک فرمول دیگر داشته باشیم. به عنوان یک مثال کامپیوتری، در پایین حلقه‌ها معمولاً دستورات پرش قرار دارند که به اول حلقه پرش می‌کند. چون معمولاً این پرش‌ها انجام می‌شوند و به عقب برمی‌گردند، بنابراین ما می‌توانیم همیشه دستورهای پرش شرطی را که به آدرس‌های پائین‌تر در برنامه پرش می‌کنند به این صورت پیش‌بینی کنیم که این پرش‌ها انجام می‌شوند(branch taken).

روشهای پیش‌بینی که در بالا توضیح دادیم، روشهای خیلی کلی و کلیشه‌ای هستند و رفتار و شخصیت مستقل تک تک دستورهای پرش را در نظر نمی‌گیرند. سخت‌افزارهای پیش‌بینی کننده دینامیک[1] در مقایسه با روش‌های فوق، پیش‌بینی‌های خود را بر روی تک تک دستورات پرش جداگانه انجام می‌دهند و ممکن است پیش‌بینی برای یک دستور پرش در طول حیات یک برنامه تغییر کند. اگر بخواهیم مثال اتاق رختشوئی را در نظر بگیریم، در پیش‌بینی دینامیک، یک نفر ممکن است به میزان کثیف بودن لباس‌ها نگاه کرده و یک فرمول پیش‌بینی کند. پیش‌بینی بعدی که انجام خواهد شد به موفقیت پیش‌بینی قبلی بستگی خواهد داشت. یک راهکار عمومی برای پیش‌بینی دینامیک در کامپیوترها این است که یک سابقه‌ای از هر دستور پرش همانند taken می‌شود یا untaken، را نگهداری کنیم و از گذشته استفاده نموده و آینده را پیش‌بینی کنیم. چنین سخت‌افزاری در حدود ۹۰ درصد دقیق خواهد بود. (بخش ۶-۶ را مشاهده کنید.)

موقعی که حدس یا پیش‌بینی اشتباه در می‌آید، در این صورت مدار کنترل پایپلاین باید دستوری را که بعد از دستور پرش اشتباه پیش‌بینی شده، قرار دارد، از پایپلاین خارج نموده و جلوی اجرای آن را بگیرد. و همچنین مدار کنترل پایپلاین باید دستوری را که در آدرس مقصد پرش قرار دارد وارد پایپلاین نماید.

نکته: در پایپلاین‌های طولانی‌تر، هزینه‌ای که در حالت پیش‌بینی غلط پرداخت می‌شود، بیشتر است چون در این حالت دستورهای بیشتری باید از پایپلاین دور ریخته شوند. در بخش ۶-۶ راه‌حل‌های مختلفی برای رفع مخاطرات کنترلی با جزئیات بیشتر ارائه شده است.

۶-۱-۴-۳- راه‌حل delayed decision

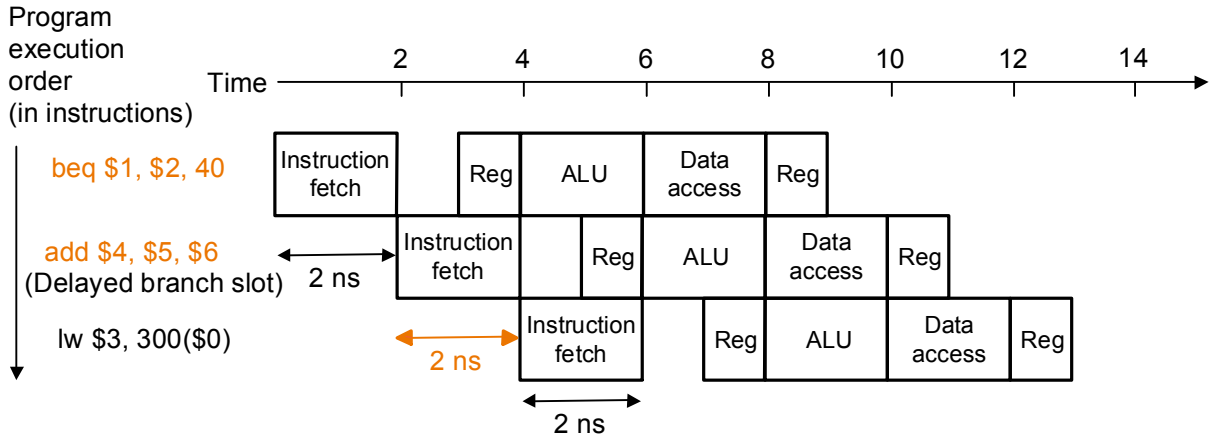
در پایپلاین اتاق رختشوئی، هر موقع که شما می‌خواهید در مورد فرمول شوینده تصمیم بگیرید، در موقعی که منتظر هستید تا اولین دسته از لباسهای تیم فوتبال خشک شود، در این لحظه می‌توانید یک دسته از لباسهای غیر فوتبالی را که فرمول آنها را می‌دانید، در داخل شوینده قرار دهید تا شسته شوند. تا وقتی که شما به اندازه کافی لباس کثیف داشته باشید که هیچ ارتباطی به لباسهای فوتبالی نداشته باشند، این روش به خوبی کار خواهد کرد.

در کامپیوترها این راه حل delayed branch نامیده می‌شود. در delayed branch دستور بعد ازbranch همیشه اجرا می‌شود. این مطلب از دید برنامه‌نویسی که به زبان اسمبلی ماشین MIPS برنامه می‌نویسد، پنهان است، به دلیل اینکه اسمبلر دستورها را arrange نموده و دستورات مناسبی را بعد از branch ها قرار می‌دهد. کامپایلرهای MIPS بعد

[1] - Dynamic branch predictors

از هر دستور branch دستور مناسبی قرار می‌دهند که این دستور همیشه اجرا می‌شود چه پرش انجام شود و چه انجام نشود. انجام شدن پرش و یا انجام نشدن آن، دستور بعد از دستور پشت سر branch را تعیین خواهد کرد که آیا دستور پشت سر آن را وارد پایپ کند و یا اینکه از محل پرش دستوری را وارد نماید.

در مثال شکل ۴ دستور add که قبل از دستور branch قرار گرفته است، دستور branch را تحت تأثیر قرار نمی‌دهد، بنابراین همان طور که در شکل ۶ نشان داده شده است، ما آن را به پشت دستور branch منتقل کرده‌ایم. به مکان پشت سر دستور پرش، delayed branch slot گفته می‌شود.



شکل ۶: راه حل delayed branch به عنوان راه حلی برای رفع مخاطره کنترلی. در این شکل حباب پایپلاین یا bubble با دستور add جایگزین شده است.

کامپایلرها معمولاً ۵۰ در صد این slot ها را با دستورات مناسب پر می‌کنند. اما اگر تعداد مراحل پایپلاین بیش از ۵ مرحله باشد، در این صورت تعداد slot های دستور پرش نیز بیشتر خواهد شد و بنابراین پر کردن آنها مشکل خواهد بود.

۶-۱-۵- مخاطره داده‌ای

برگردیم به مثال اتاق رختشوئی، فرض کنید که شما در حال تا کردن لباسهایی هستید که اکثراً جوراب هستند. با بدشانسی متوجه می‌شوید که جفت همهٔ جورابهایی که هم اکنون دارید تا می‌کنید، داخل دسته‌ای از لباسها قرار دارند که هم اکنون داخل شوینده هستند. بنابراین شما تا زمانیکه آن دسته از لباسها تمام نشده باشند، نمی‌توانید کاری انجام دهید. بنابراین باید پایپلاین را متوقف کنید. این مسأله در کامپیوترها، مخاطره داده‌ای نامیده می‌شود: یک دستور به نتایج دستور قبلی که هم اکنون داخل پایپلاین است و تمام نشده، وابسته است. به عنوان مثال، فرض کنید ما یک دستور add داشته باشیم که دستوری که پشت سر آن قرار گرفته یک دستور sub باشد که از نتایج add استفاده می‌کند:

```
add $s0,$t0,$t1
sub $t2,$s0,$t3
```

اگر راه حلی برای این مسأله ارائه ندهیم، یک مخاطره داده‌ای می‌تواند پایپلاین را با تأخیر زیادی مواجه کند. دستور add تا آخر مرحله ۵ نتایج خود را نخواهد نوشت و این بدان معنی است که ما باید سه کلاک اجرای دستور sub را متوقف کنیم تا دستور add به آخر مرحله ۵ برسد و یا به عبارتی باید سه حباب به پایپلاین اضافه کنیم.

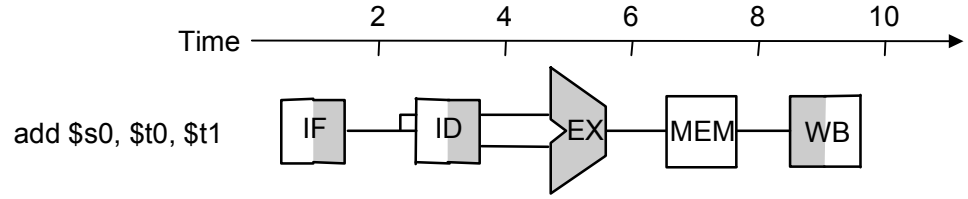
هر چند ما می‌توانیم به کامپایلر تکیه کنیم که جلوی بروز مخاطرات داده‌ای را بگیرد، اما این روش همیشه جواب نخواهد داد. به دلیل اینکه وابستگی‌های داده‌ای در برنامه‌ها خیلی زیادند و تأخیرها آنقدر زیادند که نمی‌توانیم از کامپایلر بخواهیم که ما را از شر این مسأله برهاند.

راه حل این مسأله به این صورت است که ما نیاز نداریم تا آخر اجرای دستور صبر کنیم تا مسأله وابستگی داده-ای را حذف کنیم و برای رشته کد بالایی، به محض اینکه ALU نتیجه عملیات add را آماده کرد، ما می‌توانیم آن نتیجه را در اختیار دستور sub قرار دهیم. گرفتن نتایج از منابع داخلی و قبل از تمام شدن دستوری که به نتایج آن نیاز داریم، forwarding یا bypassing نامیده می‌شود.

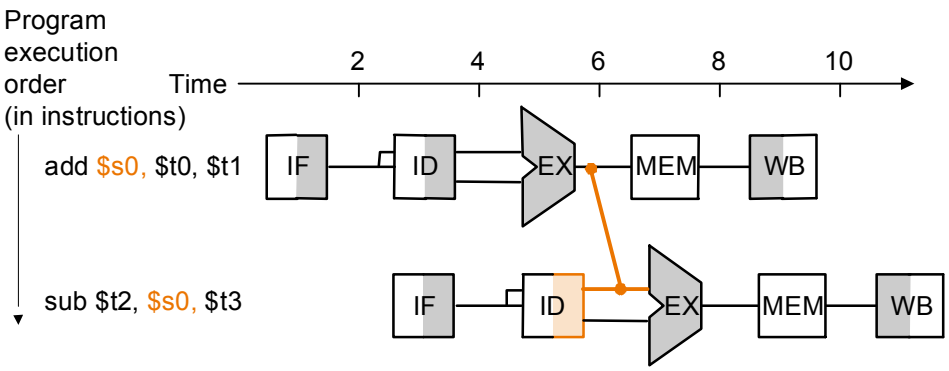
مثال: راه حل forwarding برای حل مسأله وابستگی داده‌ای

برای دو دستور بالایی نشان دهید که کدام یک از مرحله‌های پایپلاین باید با forwarding به هم وصل شوند. از نمایش شکل ۷ برای نمایش مسیر داده (datapath) در طول ۵ مرحله پایپلاین استفاده کنید. برای هر دستور یک کپی ازdatapath را قرار دهید (به همان ترتیبی که در شکل ۱ برای مثال اتاق رختشوئی نیز این کار را انجام دادیم).

جواب: شکل ۸ اتصال لازم را نشان می‌دهد که مقدار محاسبه شده برای رجیستر $S0 را بعد از مرحله execute دستورadd به عنوان ورودی برای مرحله execute دستور sub ، به آن forward می‌کند. توجه کنید که مقدار forward شده برای $s0 جایگزین مقداری می‌شود که در در مرحله ID از بانک رجیستر خوانده شده است.



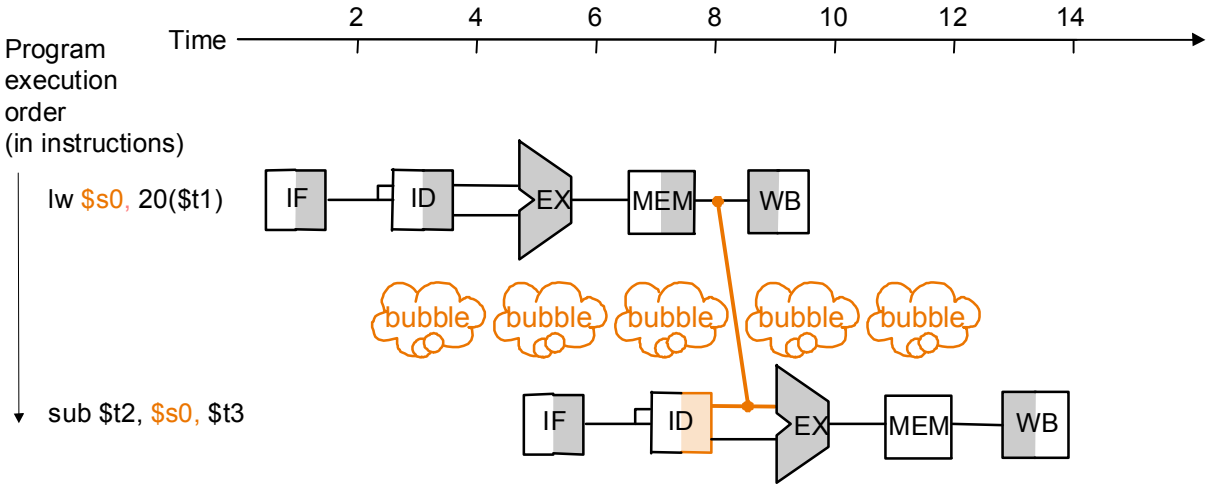
شکل ۷: نمایش گرافیکی پایپلاین دستورات. در این نمایش ما از سمبولهایی برای نشان دادن منابع فیزیکی استفاده کرده‌ایم که در داخل هرکدام از این سمبولها یک علامت اختصاری برای مراحل پایپلاین نوشته شده است که در سرتاسر این فصل از این علامتها استفاده خواهد شد. IF یعنی Instruction Fetch و به مرحله‌ای گفته می‌شود که دستور از حافظه دستورات خوانده می‌شود. ID یعنی Instruction Decode و به مرحله دیکد و خواندن رجیسترها اشاره دارد. EX یعنی Execute و به مرحله‌ای گفته می‌شود که عملیات مورد نیاز دستور در داخل ALU اجرا می‌شود و به همین دلیل از علامت ALU برای آن استفاده شده است. MEM یعنی Memory و به مرحله‌ای اشاره دارد که دستور به حافظه داده‌ها دسترسی پیدا می‌کند. WB یعنی Write Back و مرحله‌ای است که نتایج داخل بانک رجیستر نوشته خواهند شد. سایه‌ای که بر روی سمبولها قرار گرفته نشان دهنده استفاده شدن آنها در این دستور است. به طور مثال برای دستور add که در این شکل نشان داده شده است IF ، ID ، EX و WB استفاده می‌شوند ولی MEM استفاده نمی‌شود چون دستور add به حافظه داده‌ها نیازی ندارد. اگر سایه در سمت راست سمبول قرار گرفته باشد نشان دهنده خوانده شدن است و اگر سایه در سمت چپ سمبول قرار گرفته باشد نشان دهنده نوشته شدن است. طبق این توضیحات و با توجه به سایه‌ها در IF دستور خوانده می‌شود، در ID رجیسترها خوانده می‌شوند و در WB رجیسترها نوشته می‌شوند.



شکل ۸: نمایش گرافیکی forwarding

در نمایش گرافیکی برای forwarding، مسیر forwarding فقط وقتی معتبر است که مرحلهٔ مقصد که به آن forward می‌شود، از نظر زمانی بعد از مرحلهٔ مبدأ که از آن forward می‌شود قرار داشته باشد. به طور مثال از خروجی مرحله دسترسی به حافظه در دستور اول به ورودی مرحله execute دستور دوم نمی‌تواند یک مسیر forwarding معتبر وجود داشته باشد. به دلیل این که معنی این کار این است که در زمان به عقب برگردیم و این نشدنی است.

Forwarding به خوبی stallها را از بین می‌برد و اجرای برنامه را سریعتر می‌کند. در بخش ۶-۴ forwarding را با جزئیات بیشتری بررسی خواهیم کرد. اما باید متذکر شویم که forwarding نمی‌تواند همهٔ stall ها را بر طرف کند. به طور مثال فرض کنید دستوراول به جای add یک دستور lw به داخل رجیستر $s0 باشد. همان گونه که از شکل ۸ می‌توانیم ببینیم، داده مورد نظر دستور اول بعد از مرحله ۴ در دسترس خواهد بود که برای مرحله ۳ دستور دوم خیلی دیر خواهد بود. بنابراین حتی با forwarding ما باید یک stall به پایپلاین بدهیم و اجرای دستور بعد از lw را با تأخیر مواجه کنیم تا داده‌ای که نیاز دارد آماده شود. این مطلب در شکل ۹ نشان داده شده شده است. بخش ۶-۵ نشان خواهد داد که چگونه سخت‌افزار پایپلاین شده با مواردی نظیر این مسأله برخورد خواهد نمود.



شکل ۹: در صورت استفاده از forwarding ممکن است دوباره به stall نیاز داشته باشیم و آن زمانی است که بعد از دستور lw دستورات نوع R وجود داشته باشند که به نتایج lw نیاز داشته باشند.

مثال: عوض کردن ترتیب دستورات برنامه برای جلوگیری از stall

قطعه برنامه زیر کارش عوض کردن (swap کردن) محتوای دو خانه از حافظه است. دستورات این برنامه را طوری جا بجا کنید که stall اتفاق نیافتد. (فرض کنید که آرایه‌ای از کلمات به نام V داریم و می‌خواهیم V[k] را با v[k+1] عوض کنیم یعنی محتوای دو خانه از آرایه را عوض کنیم. آدرس V[k] داخل رجیستر $t1 است.)

```
lw $t0,0($t1)  #  t0 = V[k]
lw $t2,4($t1)  #  t2 = V[k+1]
sw $t2,0($t1)  # V[k] = t2
sw $t0,4($t1)  # V[k+1] = t0
```

جواب: مخاطره بر روی رجیستر $t2 و بین lw دوم و sw اول اتفاق می‌افتد. و جابجا کردن جای دو دستور sw باعث رفع این مخاطره خواهد شد:

```
lw $t0,0($t1)  #  t0 = V[k]
lw $t2,4($t1)  #  t2 = V[k+1]
sw $t0,4($t1)  # V[k+1] = t0
sw $t2,0($t1)  # V[k] = t2
```

توجه کنیدکه ما با این کار مخاطره جدیدی ایجاد نکردیم به دلیل اینکه بین نوشتن رجیستر t0 توسط دستور lw و خواندن از آن توسط دستور sw یک دستور فاصله است. بنابراین بر روی یک ماشین که از تکنیک forwarding استفاده می‌کند این کد جدیدی که ما با جابجایی دستورات تولید کردیم، ۴ کلاک طول خواهد کشید.

توجه: اگر دستور بعد از دستور load دستوری باشد که از نتایج load استفاده کند، در این صورت حتماً نیاز به یک stall داریم تا مخاطره داده‌ای را به کمک forwarding رفع کنیم. می‌توان از کامپایلر کمک گرفت و از او خواست که بلافاصله بعد از دستور load دستوری قرار ندهد که به نتایج آن نیاز داشته باشد و به جای آن دستوراتی قرار دهد که وابسته به load نیستند. به این ترتیب نیازی به stall دادن نخواهد بود. به دستورات load ای که دستور بعد از آن به نتیجه load وابسته نیست dalayed loads گفته می‌شود.

Forwarding یکی دیگر از ویژگیهای خوب معماری MIPS را که مناسب برای پایپلاین کردن می‌باشد را نشان می‌دهد. (چهار ویژگی را قبلاً ذکر کرده‌ایم) هر دستور MIPS فقط یک نتیجه را در خاتمه اجرای دستور می‌نویسد. اگر یک دستور بیش از چند نتیجه را بخواهد بنویسد در این صورت forwarding مشکل‌تر خواهد شد. به طور مثال در پردازنده PowerPC، دستورات load ممکن است از شیوه آدرس‌دهی update addressing استفاده کنند که این دستورات دو نتیجه برای نوشتن داخل بانک رجیستر دارند، بنابراین پردازنده باید قابلیت forward کردن دو نتیجه را برای هر دستور داشته باشد.

۶-۱-۶- خلاصه مرور پایپلاین

- پایپلاین کردن یک تکنیکی است که بین یک رشته‌ای از دستورات که باید به ترتیب اجرا شوند، موازی‌سازی می‌کند و باعث می‌شود که چند دستور به صورت همزمان در سخت افزار اجرا شوند. پایپلاین کردن، یک مزیت خیلی خوبی که دارد این است که بر خلاف تکنیکهای دیگری که برای افزایش سرعت استفاده می‌شود (فصل ۹ را مشاهده کنید)، از دید کاربر کاملاً پنهان است.
- پایپلاین کردن باعث می‌شود که تعداد دستوراتی که در واحد زمان انجام می‌شوند، بیشتر شود به عبارتی بازدهی یا throughput را افزایش می‌دهد. در پایپلاین زمان اجرای یک دستور کاهش پیدا نمی‌کند بلکه زمان اجرای کل برنامه کاهش پیدا می‌کند. در پایپلاین ۵ مرحله‌ای که ما در این بخش معرفی کردیم هر دستوری در ۵ مرحله ودر ۵ کلاک اجرا می‌شود.
- معماری مجموعه دستورات می‌تواند پایپلاین کردن را برای طراحان راحت‌تر کند و یا اینکه آن را خیلی مشکل نماید. معماری مجموعه دستورات MIPS دارای پنج ویژگی خیلی خوب است که آن را برای پایپلاین کردن مناسب‌تر کرده است.
- در پایپلاین سه نوع مخاطره وجود دارد: مخاطره ساختاری، مخاطره کنترلی و مخاطره داده‌ای
- پیش‌بینی پرش، forwarding و stall دادن راههایی هستند که باعث می‌شوند در حالی که جواب درست بدست می‌آید و برنامه به درستی کار می‌کند، سرعت برنامه نیز افزایش پیدا کند.
- اصطلاح forwarding از آن جایی ناشی می‌شود که نتایج از یک دستور قبلی به یک دستور بعدی فرستاده می‌شود، یعنی به جلو فرستاده می‌شود و یا forward می‌شود. اصطلاح bypassing نیز از pass کردن یا فرستادن نتایجی که باید داخل بانک رجیستر نوشته شود، به مراحل دلخواه در پایپلاین ناشی شده است.

در بخش بعدی این فصل ما یک پردازنده پایپلاین شده برای زیر مجموعه‌ای از دستوراتMIPS مشتمل بر دستورات lw ، sw ، add ، sub ، and ، or ، slt ، beq طراحی خواهیم کرد و مفاهیم پایپلاین کردن را در آن پوشش خواهیم داد. سپس مشکلاتی را که پایپلاین کردن ایجاد می‌کند را خواهیم دید و در برخی موارد کارآیی را نیز بررسی خواهیم کرد.

اگر شما می‌خواهید که خیلی گذرا پایپلاین را بررسی کنید ما باور داریم که بعد از خاتمه این بخش شما به اندازه کافی پیش زمینه خواهید داشت و می‌توانید بعد از این بخش به سراغ بخش‌های ۶-۸ و ۶-۹ بروید تا اینکه با مفاهیم پیشرفته پایپلاین کردن همانند superscalar و پایپلاین کردن دینامیک (dynamic pipelining) آشنا شوید و همچنین ببینید که در پردازنده‌هایی که اخیراً طراحی شده‌اند پایپلاین چگونه کار می‌کند.

ولی اگر می‌خواهید که پایپلاین کردن را با جزئیات مطالعه نمائید، بعد از پایان این بخش و فصل ۵ شما قادر خواهید بود که یک مسیر داده را تغییر دهید تا به صورت پایپلاین کار کند (بخش ۶-۲) و بخش کنترل را برای این

پردازنده پایپلاین شده طراحی کنید (بخش ۶-۳). همچنین قادر خواهید بود که مسیر داده و بخش کنترل را برای پیاده سازی forwarding عوض کنید (بخش ۶-۴)، و تغییرات مشابهی را برای stall هایی که بعد از دستور load لازم است، ایجاد کنید (بخش ۶-۵). بعد از آن شما می‌توانید مطالب زیادی در مورد راه حل‌های مخاطره و دستور پرش، در بخش۶-۶ مطالعه کنید و در بخش ۶-۷ نیز چگونگی برخورد با exception را ببینید.