استیج های pipeline : fetch, decode, exe, memory, writeback

fetch : دستور از حافظه خوانده می شود و

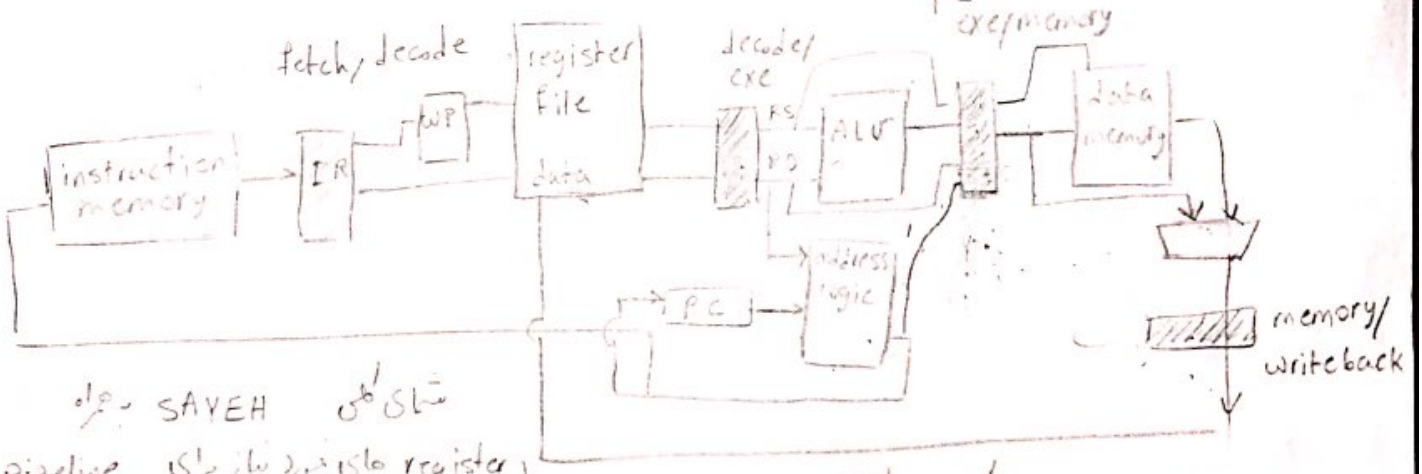
decode : مقادیر از registerfile خوانده می شوند . سیگنال های کنترلی تولید می شود

exe : مقادیر توسط ALU محاسبه می شود

memory : اگر نیاز به خواندن یا نوشتن memory بود می خوانیم یا می نویسیم

writeback : در صورت نیاز مقادیر register ها نوشته می شود

برای دستیابی به Pipline باید بین سخت افزار هر ۲ استیج رجیسترهای قرار دهیم تا بتوانند جدا از هم کار کنند . از طرفی چون باید سخت افزار هر استیج تا حد امکان جدا باشد و چون مرحله fetch و memory سخت افزار مشترک دارند (هر دو به memory) و کار این ۲ استیج با هم تداخل دارد memory دستور و داده را جدا می کنیم



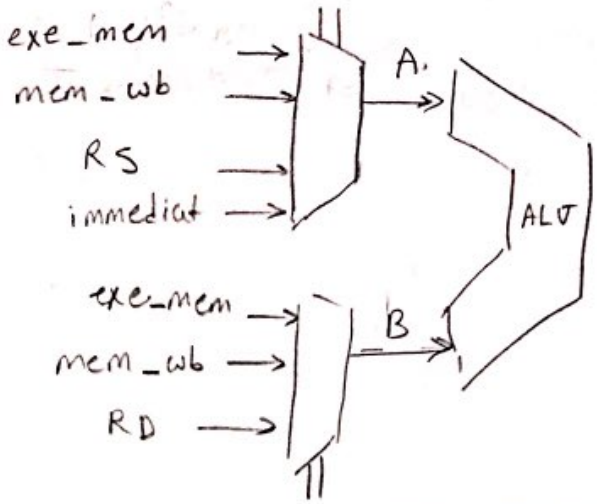
نام های کلی به همراه SAVEH های register های مورد نیاز برای Pipeline هرکدام از رجیسترهای بین ۲ استیج ممکن است بیش از یک رجیستر باشد و در شکل های بعدی دقیق تر رجیسترها مشخص می شود

حال hazard های ممکن را آورده و پس از رفع آن ها datapath نهایی را می کشیم :

۱. structral hazard : می شود داده و دستور را همزمان از حافظه خواند برای همین حافظه ی داده را از حافظه دستور جدا کردیم

۲. Data hazard : ۱. read after write : زمانی پیش می آید که مرحله ی writeback دستور نوشتن در رجیستری تمام نشده باشد و بخواهیم از آن بخوانیم (از آن رجیستر) . برای حل آن از forwarding استفاده می کنیم

دیتایی که قرار است در رجیسترها نوشته شوند می توانند PC+1 ، memout ، ALUout (data memory) باشد که این data می تواند در رجیسترهای exe-mem ، mem_wb باشد و کنترلر این تصمیم را می گیرد :



---

۲. خواندن و استفاده از Z و V (توسط ALU یا کنترلر) و نوشتن نتایج آن ها Z و V در کنترلر تا پایان مرحله exe نوشته می شوند. در مواردی که ALU می خواهد از Z و V استفاده کند مشکلی پیش می آید زیرا در مرحله ی exe هست و قطعاً exe بقیه ی مرحله ها تمام شده . در زمانی که کنترلر بخواهد از Z و V استفاده کند باید توجه داشته باشیم که exe دستورهایی که Z و V را تغییر می دهند تمام شده باشد

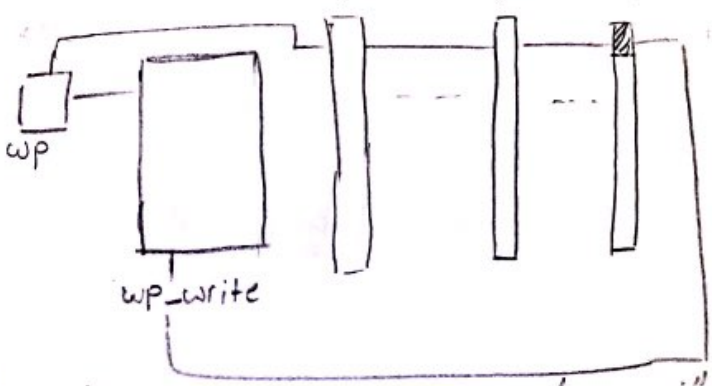
control hazard : زمانی اتفاق می افتد که بخواهیم PC را مقدار دهیم (دستورهای branch و Jump) اوّل باید حواسمان باشد که اگر کنترلر Z و V را چک می کند دستوری که آن ها را set می کند مرحله ی exe اش تمام شده باشد . ⟸ در آخر مرحله ی exe مقدار PC مشخص می شود ⟸ ۲ دستور بعدی fetch می شود (محتمل است) . اگر دستور jump بوده که باید ۲ دستور را flushing کنیم و اگر branch انجام شود . اگر انجام نشود نیازی به flushing نیست .

Z و V را در مرحله ی exe می خواند و بررسی می کند branch باید انجام شود یا نه . (که تماماً Z و V set شده باشند)

مشکل دیگری که وجود دارد این است که ممکن است در مرحله‌ی writeback بخواهیم در رجیستر ۵ بنویسیم و همزمان در decode دستور دیگری بخواهیم از رجیستر دیگری بخوانیم. از آن جایی که wp و irs و ird آدرس این رجیسترها را مشخص می‌کنند اگر این آدرس‌ها یکی شوند با مشکل برمی‌خوریم.

برای حل این مشکل wp را دوتا می‌کنیم. به این صورت که wp و irs و ird برای خواندن به صورت قبل عمل می‌کنند ولی یک wp_write نیز اضافه می‌شود و یک ird مخصوص write کردن. wp_write و ird_write آدرس خانه‌ی مقصد نوشته شدن را تعیین می‌کنند.

تغییر wp_write و wp مثل هم خواهد بود. و در صورت آمدن دستور addp هر ۲ در انتهای مرحله‌ی decode مقدارشان تغییر می‌کند اما مقدار wp_write از مرحله‌ی decode به همراه دستور پاس داده می‌شود تا اگر زمانی که دستور به writeback رسید مقدار wp تغییر کرده بود به مشکل نخوریم.



ird و irs مقادیر ۲ بیتی هستند که با wp جمع می‌شوند

این یک structural hazard بوده و با تغییر سخت افزاری حل شد.

توجه شود که در این پایپ لاین سیگنال‌های کنترلی در decode ست می‌شوند پس اگر دستور addp داشتیم با توجه به این که wp خودش جمع کننده دارد در decode مقدار wp + 1 آماده است و در پایان decode در wp نوشته می‌شود ← به دیتا هازارد به این معنی که wp هنوز ست نشده باشد و بخواهیم از regist file بخوانیم برمی‌خوریم.

Pipeline controller

برای flushing ← تشخیص: دستور توی exe ای branch باشد که انجام شود

یا jump - عملیات: reset رجیسترهای همه decode, fetch

برای forwarding ← تشخیص: در exe اگر دستور مبداء خواندن با دستور مقصد نوشتن دستور موجود در mem یکی بود آن داده forward می‌شود در غیر این صورت اگر با دستور مقصد دستور موجود در wb یکی بود آن داده forward می‌شود