پروژه معماری کامپیوتر

"مخاطرات pipeline و روش های رفع آنها"

گروه ۲

دانیال غریب ۴۰۰۱۰۵۱۵۵

مهدی اکبر ۴۰۰۱۰۴۷۲۶

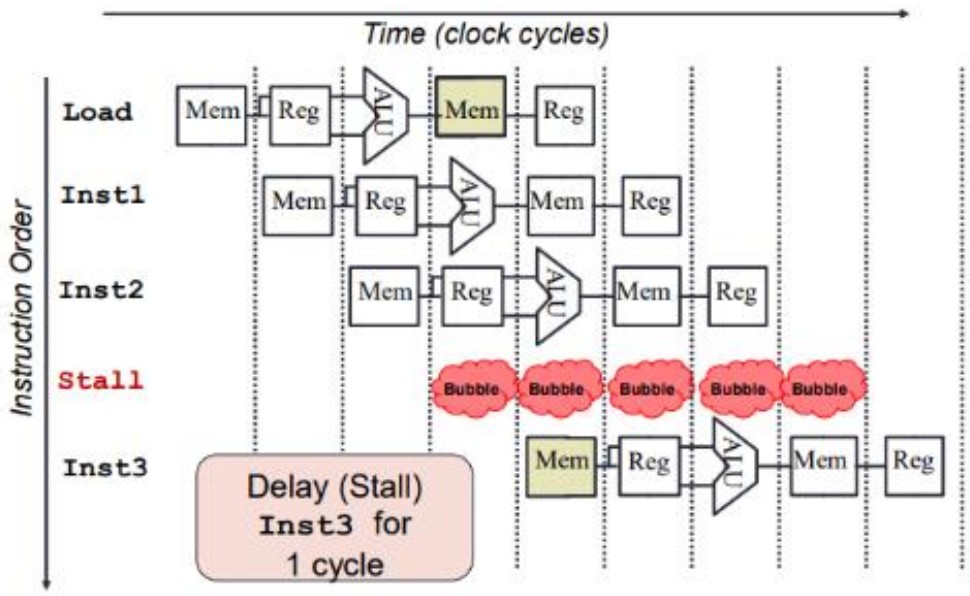
مانا عباس زاده ۴۰۰۱۰۹۶۳۸

مخاطرات در pipeline :

به طور کلی در pipeline، ممکن است سه حالت مخاطره رخ دهد:

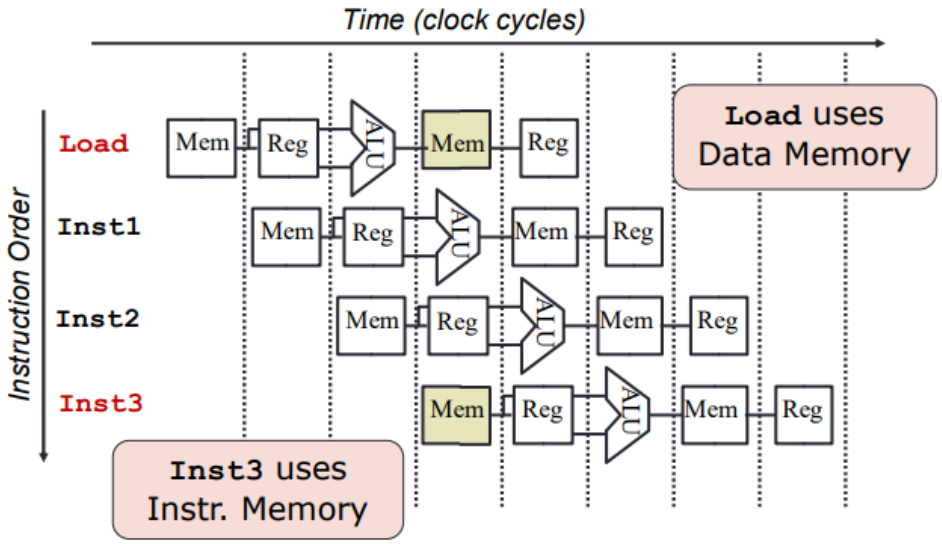
- **Structural hazards** :

این نوع مخاطره زمانی اتفاق می افتد که از یک منبع سخت افزاری، به صورت همزمان در دو جای مختلف استفاده شود. یکی از راه ها برای رفع این مشکل، متوقف کردن (Stall) pipeline است. یعنی با به تاخیر انداختن یکی از دستورات و انجام آن در مرحله بعد، دیگر از سخت افزار (مانند memory) در دو جای مختلف به صورت همزمان استفاده نمی‌شود، مانند مثال زیر:



یکی دیگر از روش های رفع این مخاطره، جدا کردن سخت افزار به دو سخت افزار مجزا است. به طور مثال اگر سخت افزار موردنظر memory باشد، می‌توانیم آن را به دو memory جدا که یکی data memory و دیگری instruction memory است

تقسیم کنیم. در این صورت می‌توانیم به راحتی در یک زمان واحد، از این دو memory استفاده کنیم بدون اینکه مخاطره ای صورت بگیرد، مانند مثال زیر:



ما در مدارمان ، مانند روش دوم، دو memory (data memory و instruction memory) طراحی کرده ایم که از این نوع مخاطره در مدار جلوگیری می‌کند. بنابراین structural hazard نداریم.

- **Data hazards** :

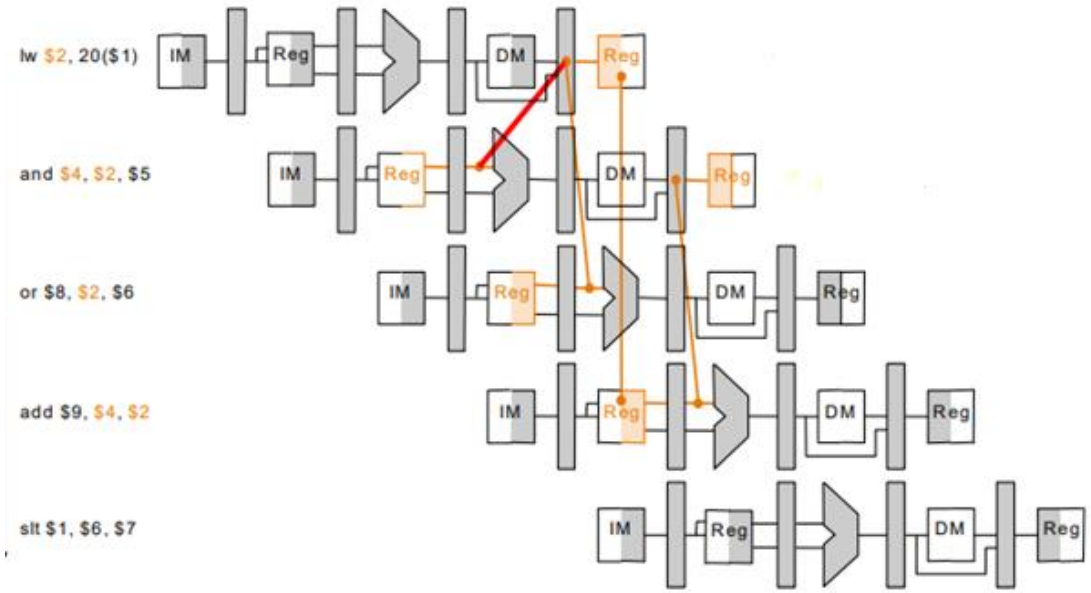
این نوع مخاطرات ممکن است در کار کردن با دیتاهای مرتبط و وابسته به هم رخ دهند که حالات مختلفی نیز دارند:

- **RAW** (**read after write**) : این مخاطره زمانی اتفاق می افتد که در یک مرحله دیتایی را در رجیستر ذخیره می‌کنیم و در عملیات بعد، به مقدار جدید آن رجیستر نیاز داریم، اما مقدار قبلی آن رجیستر به ما داده می‌شود. کد زیر، مثالی از این مخاطره است:

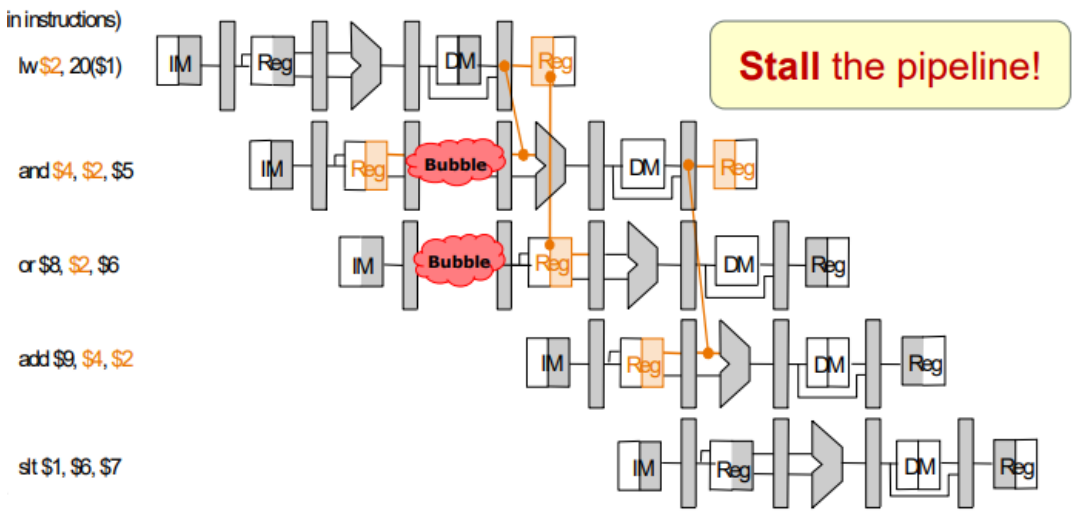
```
i1: add $1, $2, $3 #writes to $1
i2: sub $4, $1, $5 #reads from $1
```

در این مثال، اگر در i2، رجیستر ۱ زودتر از اینکه در i1 مقداردهی شود، خوانده شود، یک data hazard خواهیم داشت. برای رفع این مشکل، می‌توانیم نتیجه را forward کنیم، به این صورت که نتیجه را به instruction های بعدی، قبل از اینکه در register file ذخیره شود، ارسال (forward) کنیم. روش دیگر، bypass یا جایگزینی داده های خوانده شده از register file است. ما این نوع مخاطره (RAW) را در مدارمان داریم.

- **WAW** (**write after write**) و **WAR** (**write after read**) : این دو نوع مخاطره هم جزو data hazard ها هستند که در ساختار پایپلاین MIPS، هیچگاه رخ نمی‌دهند، بنابراین این دو مخاطره را ما در مدارمان نداریم.
- **Load instruction** : این نوع مخاطره مربوط به load کردن دیتا در رجیستر است، یعنی زمانی که به دیتایی، قبل از اینکه تولید و load شود، نیاز داریم. ما این مخاطره را در مدارمان داریم. کد زیر مثالی از این نوع مخاطره است :



در مثال بالا، در مرحله دوم، ما زمانی که به رجیستر ۲ نیاز داریم، هنوز در مرحله قبل، دیتا در آن لود نشده است و این مورد یک مخاطره ایجاد می‌کند. برای رفع این مخاطره، دیگر نمی‌توان از روش forwarding که قبل تر گفته شد، استفاده کرد. روشی که برای جلوگیری از این نوع مخاطره باید انجام داد، ایجاد stall در pipeline است. مثلا اگر در مثال قبل bubble ایجاد کنیم، دیگر مخاطره ای صورت نمی‌گیرد و زمانی که به دیتای موردنظر نیاز داشته باشیم، مقدار آن آماده خواهد بود:



- **Control hazards** :

این نوع مخاطره مربوط به زمانی است که اجرا شدن یا نشدن یک دستور، به دستور قبلی‌اش وابسته باشد و معمولا در دستورات branch رخ می‌دهد. مدار ما در فاز ۳، این مخاطره را دارد، اما در فاز ۴، با استفاده از branch prediction این مخاطره را برای دستورات branch برطرف کرده‌ایم. کد زیر، مثالی از این مخاطره است:

```
i1: beq $3, $5, label    # branch
i2: add $1, $2, $4       # depends on i1
...   ...   ...
```

در مثال بالا، ممکن است با توجه به i1، مجبور باشیم به label برویم. اما اگر i2 قبل از تصمیم گیری و اجرای نهایی i1 اجرا شود، ممکن است رجیستر ۱ اشتباه مقداردهی شود. برای رفع این نوع مخاطره، باید در پایپلاین stall ایجاد کنیم، که ممکن است مقدار آن زیاد باشد (control hazard penalty). برای کاهش این مقدار ۳ روش وجود دارد:

- **Early branch resolution** : محاسبه branch decision را به مرحله قبلی pipeline انتقال دهیم.
- **Branch prediction** : مقدار نهایی خروجی را قبل از اینکه تولید شود حدس بزنیم. (این روش برای رفع مخاطرات مربوط به دستورات branch است.)
- **Delayed branching** : زمانی که منتظر خروجی هستیم، کار مفید دیگری انجام دهیم.