QUIZ#3

INSTRUCTOR: SAM, X. NGUYEN

STUDENT ID: 20110243 | FULL IN NAME: Lê Hải Đăng

Câu 1: Why do we need pipeline?

- Cải thiện nội dung: Hiển thị nội dung mà khách hàng muốn xem trong tương lai. Bằng cách này, nội dung có thể được nâng cao.

- Quản lý ứng dụng hiệu quả: Để theo dõi tất cả các hoạt động trong ứng dụng và lưu trữ dữ liệu trong cơ sở dữ liệu hiện có hơn là lưu trữ dữ liệu trong cơ sở dữ liệu mới.

- Nhanh hơn: Để cải thiện doanh nghiệp nhanh hơn nhưng với giá rẻ hơn.

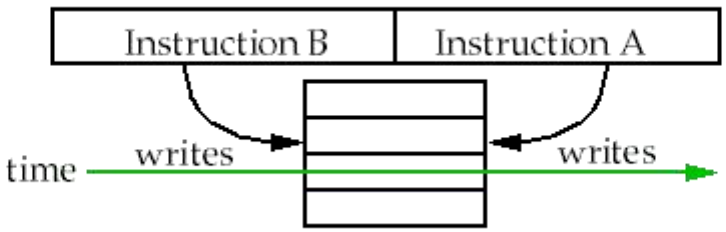
Để đạt được các mục tiêu trên có thể là một nhiệm vụ khó khăn vì một lượng lớn dữ liệu được lưu trữ ở các định dạng khác nhau, do đó việc phân tích, lưu trữ và xử lý dữ liệu trở nên rất phức tạp. Các công cụ khác nhau được sử dụng để lưu trữ các định dạng dữ liệu khác nhau. Giải pháp khả thi cho tình huống như vậy là sử dụng Data Pipeline. Data Pipeline tích hợp dữ liệu được trải rộng trên các nguồn dữ liệu khác nhau và nó cũng xử lý dữ liệu trên cùng một vị trí.

Câu 2: Examples for data hazards

*Read after write (RAW)

```
Add     R1, R2, R3      #R1 is result register
Sub     R4, R1, R2      #confict with R1
Add     R3, R5, R1      #confict with R1
Or      R6, R1, R2      #confict with R1
                        #but can be OK
Add     R5,R2,R1        #R1 OK now
```

*Write after write (WAW)

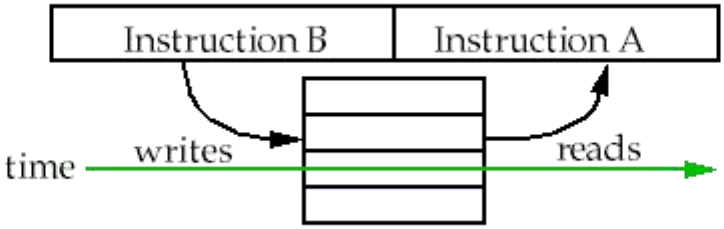


- Sau khi lệnh B được thực thi, giá trị của thanh ghi phải là kết quả của B, nhưng kết quả của A được lưu trữ thay thế.

- Điều này chỉ có thể xảy ra với các pipeline ghi giá trị trong nhiều giai đoạn hoặc trong các đường ống có độ dài thay đổi

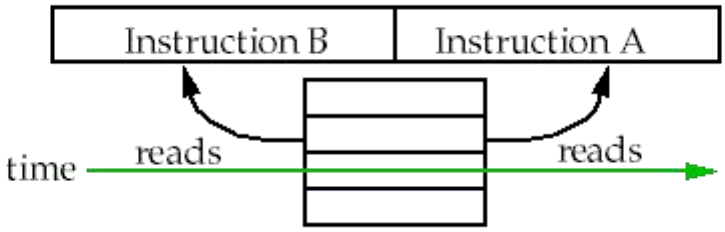
- Nó không xảy ra trong phiên bản đường dẫn DLX nhưng một phiên bản sửa đổi có thể cho phép nó

* Write after read (WAR)



- B cố gắng viết một thanh ghi trước khi A đọc nó.
- Trong trường hợp này, A sử dụng giá trị mới (không chính xác).
- Loại nguy hiểm này hiếm khi xảy ra vì hầu hết các đường ống đọc giá trị sớm và ghi kết quả muộn.
- Tuy nhiên, nó có thể xảy ra đối với một CPU có các chế độ địa chỉ phức tạp. tức là tự động gia tăng.

* Read after read (RAR)



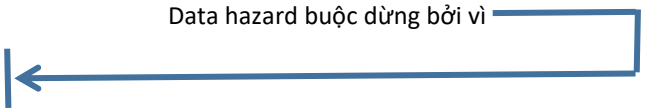
- Đây KHÔNG phải là một mối nguy vì giá trị thanh ghi KHÔNG thay đổi.
- Thứ tự của hai lần đọc không quan trọng.

Câu 3: Examples for control hazards

Có thông tin chính xác về hành vi của nhánh tại thời điểm biên dịch cũng rất hữu ích cho việc lập lịch các nguy cơ dữ liệu:

```
        Lw      R1, 0(R2)
        Sub     R1, R1, R3
        Beq     R1, L
        Or      R4, R5, R6
        Add     R10, R4, R3
L:      Add     R7, R8, R9
```

Data hazard buộc dừng bởi vì



Giả sử chúng ta biết rằng nhánh hầu như luôn được sử dụng và giá trị trong R7 là không cần thiết: Trình biên dịch có thể di chuyển ADD R7, R8, R9 sau lệnh tải.

Giả sử chúng ta biết rằng nhánh hiếm khi được lấy và giá trị trong R4 là không cần thiết trên đường dẫn được thực hiện: Trình biên dịch có thể di chuyển OR R4, R5, R6 sau lệnh tải.

*Another example:

Giả sử chúng ta có một CPU có một khe trễ nhánh duy nhất. Khoảng thời gian này có thể được lấp đầy bởi một chỉ dẫn hữu ích trong 65% thời gian.

Ngoài ra, điều kiện rẽ nhánh không được biết đến trong hai chu kỳ ngoài khe trễ. Nếu những điều này được dự đoán đúng, sẽ không bị lỗi. Nếu chúng bị dự đoán sai, hai chỉ thị can thiệp phải được hủy bỏ.

Các nhánh xuôi luôn được dự đoán là không lấy, trong khi các nhánh lùi luôn được dự đoán lấy. Các nhánh chuyển tiếp chiếm 75% tất cả các nhánh và các nhánh rẽ chiếm 20% trong tất cả các lệnh.

*Another example:

```
Add     $t4, $t5, $t6
Beq     $t1, $t2, label
Lw      $t3, 300($s0)
```

Trong ví dụ trên, nếu áp dụng pipeline thông thường, tại chu kỳ thứ 3 của pipeline, khi beq đang thực thi thì lệnh lw sẽ được nạp vào. Nhưng nếu điều kiện bằng của lệnh beq xảy ra thì lệnh thực hiện tiếp sau đó không phải là lw mà là lệnh được gán nhãn label, lúc này xảy ra xung đột điều khiển