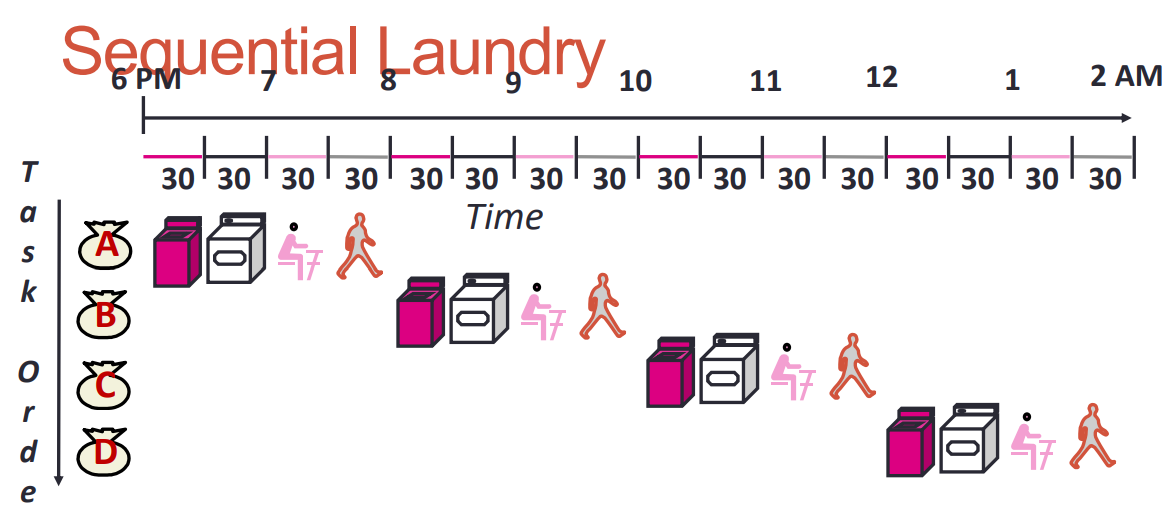
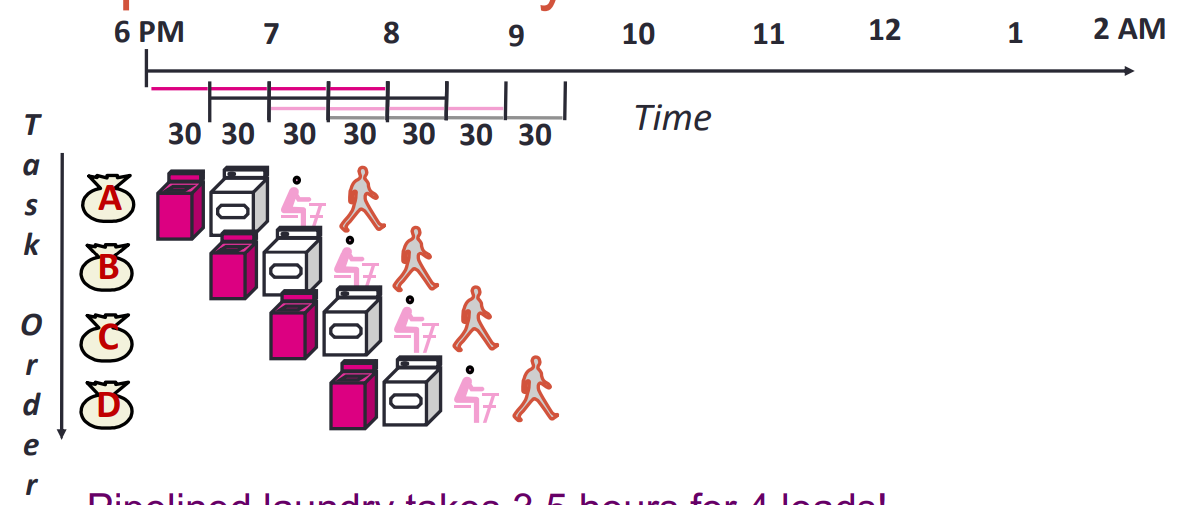
PipeLine

1. *Vấn đề trước khi có Pipeline( Nỗi đau tuần tự) và khi dùng pipeline.*

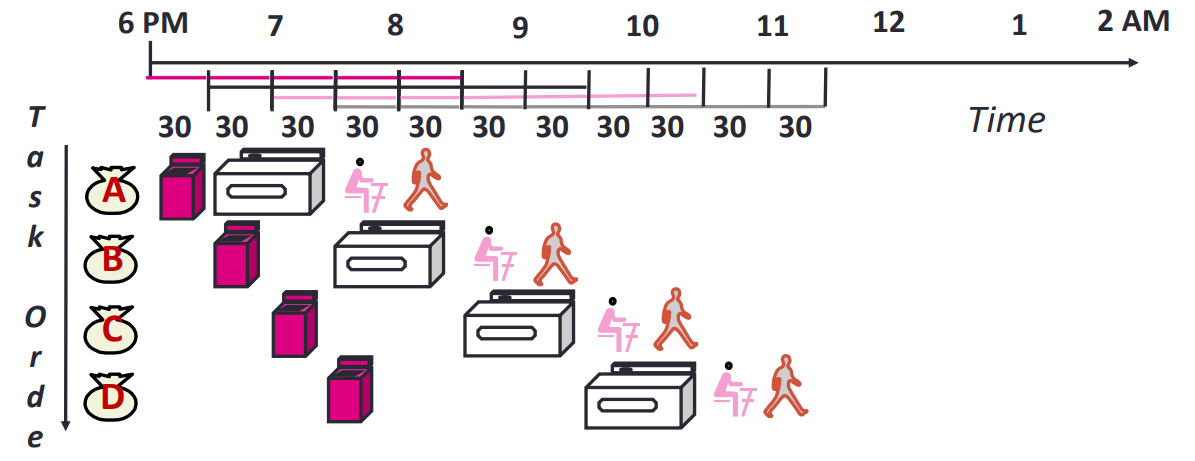
\* Trước khi có Pipeline thì những task lặp lại thực hiện 1 cách tuần tự thì rất mất thời gian



\* Khi có Pipeline thì các công đoạn sẽ được chia để có thể thực hiện đồng thời khi tài nguyên đó không được dùng tới

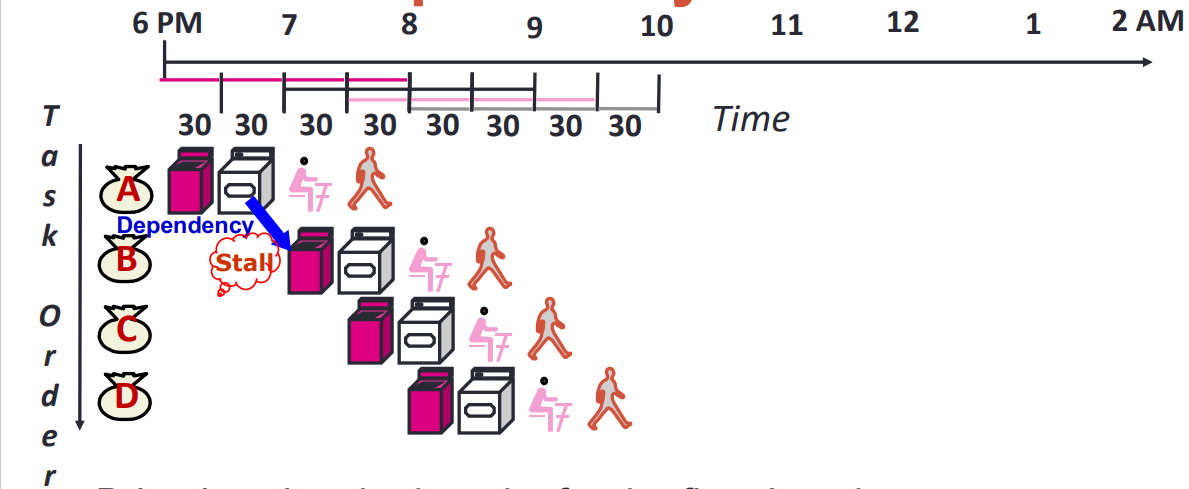


\* Tuy vậy khi dùng Pipeline thì nếu các bước không đồng bộ thời gian với nhau thì sẽ tăng thời gian lên đáng kể và gây ra thời gian chết => Thời gian không được tận dụng.



1. *Dependency*

\*Ví dụ về Dependency:



1. *Mô hình chung Pipeline trong Datapath*

IF: Instruction Fetch

ID: Instruction Decode and Register Read

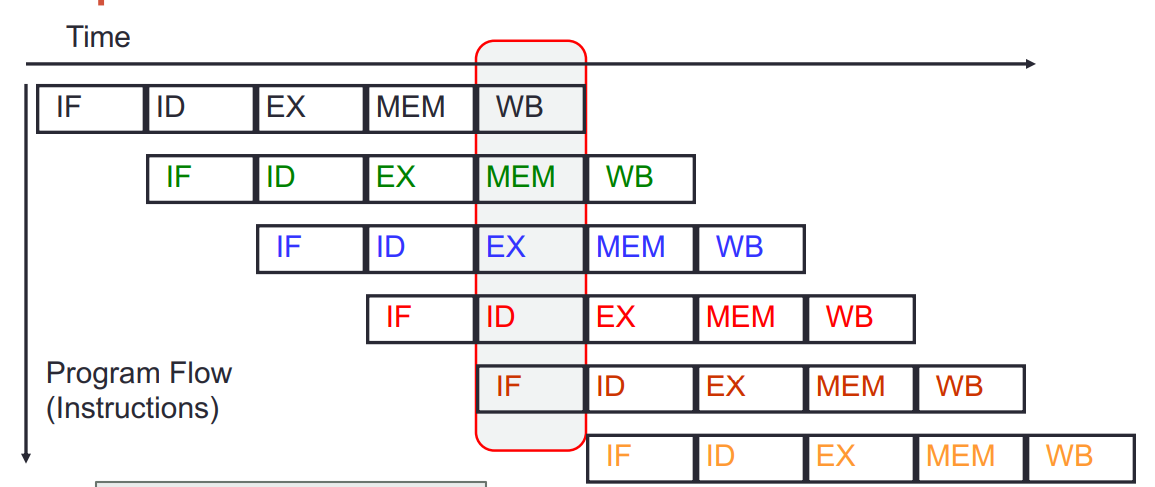
EX: Execute an operation or calculate an address

MEM: Access an operand in date memory

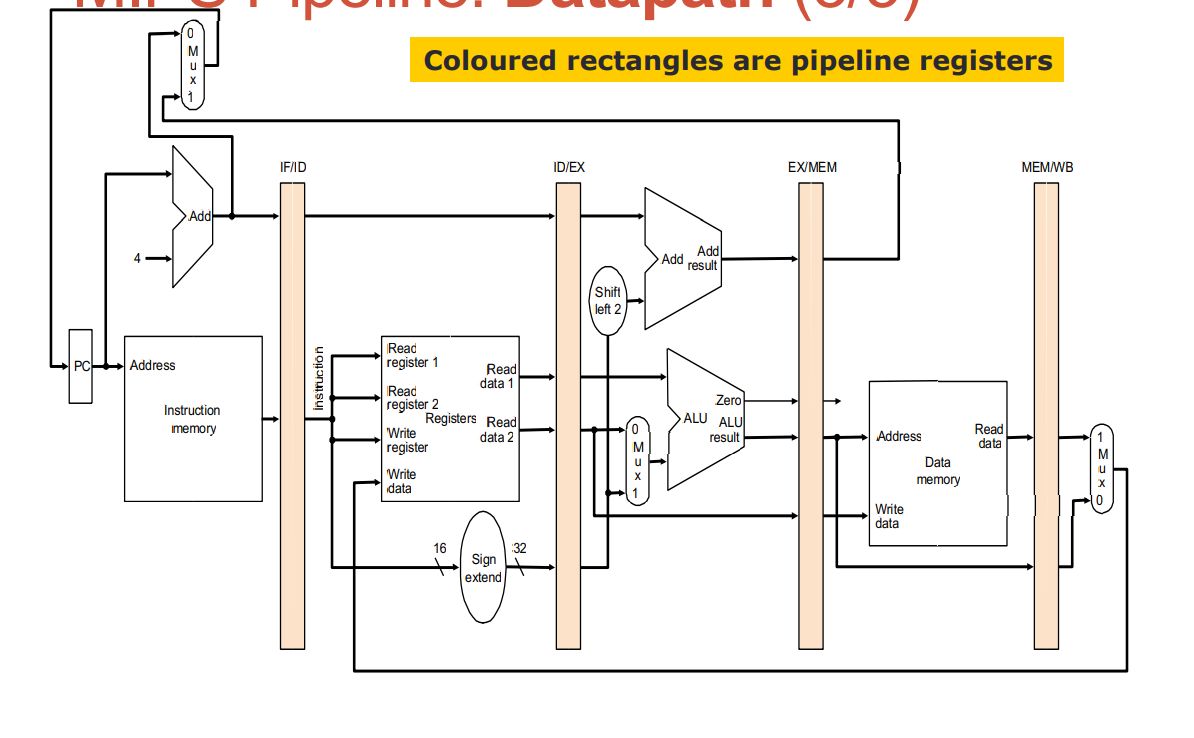
WB: Write back the result into a register

Mỗi công đoạn chiếm 1 chu kì xung clock

\*Pipeline khi áp dụng vào trong MIPS:

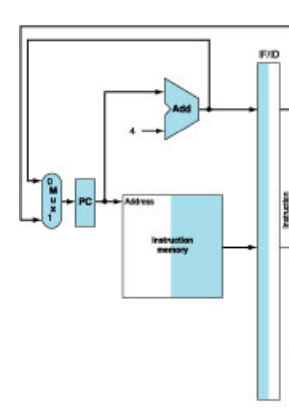


\* Hình ảnh của các thanh ghi giữa các bước trong DataPath ( Thanh ghi trong đường ống lệnh)



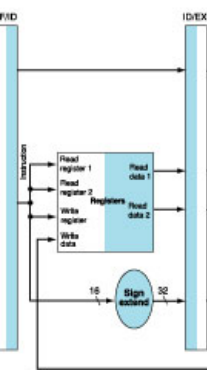
1. *Khảo sát rõ Pipeline trong MIPS DataPath*
2. *Bước IF*

\* Hình ảnh minh họa

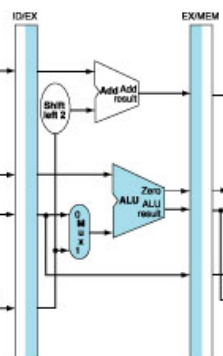


1. *Bước ID*

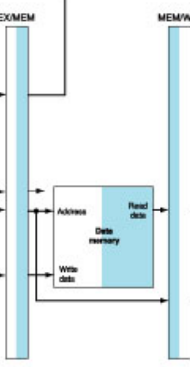
\* Hình ảnh minh họa



1. *Bước ALU*

**

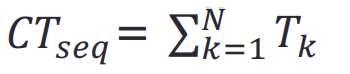
1. *Bước Memory Stage*

**

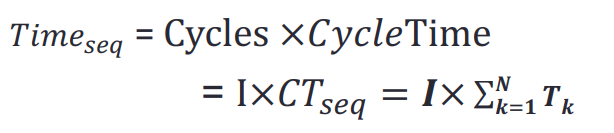
*Pipeline Performance*

1. *Single Cycle Processor*

\*Công thức tính cycle time ( Thời gian thực hiện chu kỳ):



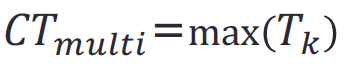
\* Công thức tính thời gian thực thi:



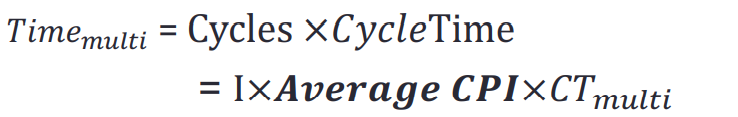
=> Do đó người ta sẽ nghĩ thêm ra cách dùng Multi Cycle Processor

1. *Multi Cycle Processor*

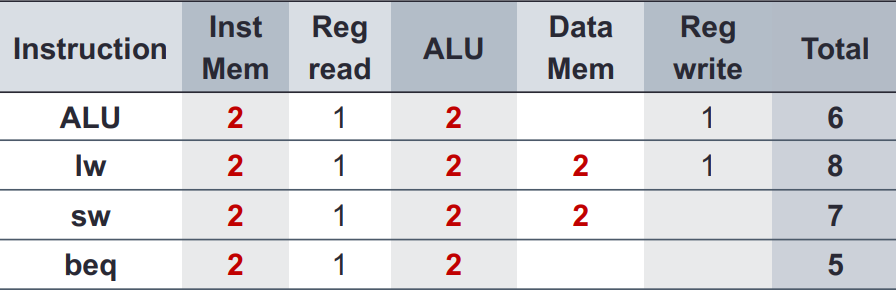
\*Công thức tính cycle time:



\*Công Thức tính thời gian thực thi:



\* Ví dụ cách chọn Cycle Time cho cả I và II

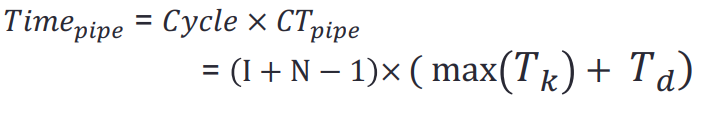


1. *Cách tính theo Pipeline Performance (Như II)*

\*Công Thức tính Cycle:

image (23)

\* Công thức tính thời gian



Pipeline Hazard

1. *Khái niệm*

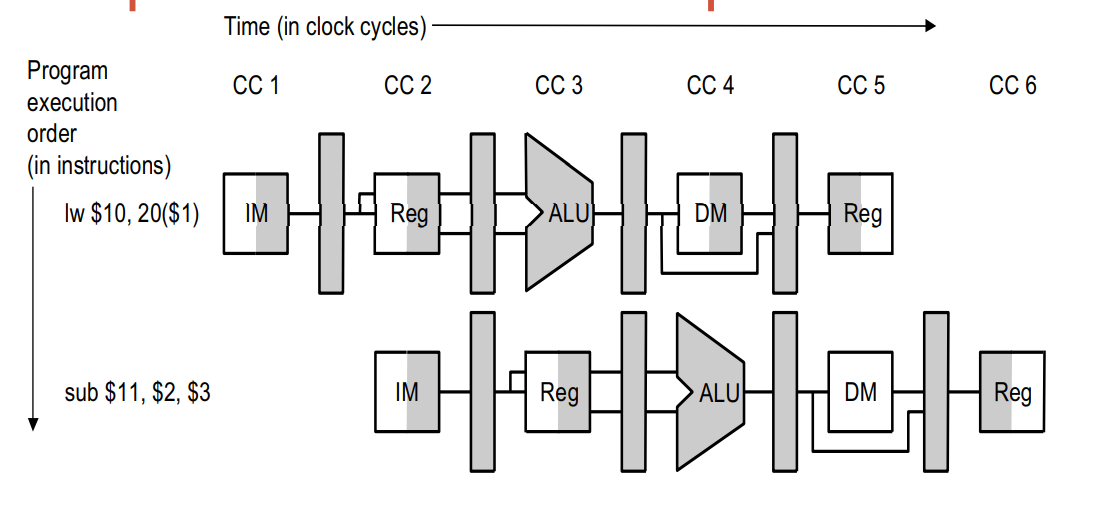
**Vấn đề mà ngăn cản các lệnh kế tiếp được thực hiện** sau cái lệnh trước đó gồm 3 loại:

+ Structure Hazard(Hazard về cấu trúc) khi sử dụng đồng thời nguồn tài nguyên phần cứng

+ Data Hazard(Data Dependency ấy)

+ Control Hazard khi có lệnh rẽ nhánh

\*Hình thể hiện Pipeline đầy đủ trong MIPS:



1. *Các loại Instruction Dependency*
2. *Data Dependency*

- Khi mà các lệnh khác nhau truy xuất cùng 1 thanh ghi cùng 1 thời điểm

**⇒ True Data Dependency**

+ RAW(Read After Write):Là việc một lệnh sau đọc một thanh ghi mà đã được ghi ở lệnh trước

⇒Có thể giá trị được đọc lệnh 2 bị sai

**⇒Tuy vậy có những Dependency không gây ra Hazard (Không phải là True Data Dependency)**

+ WAR:Write after Read

+ WAW:Write after Write

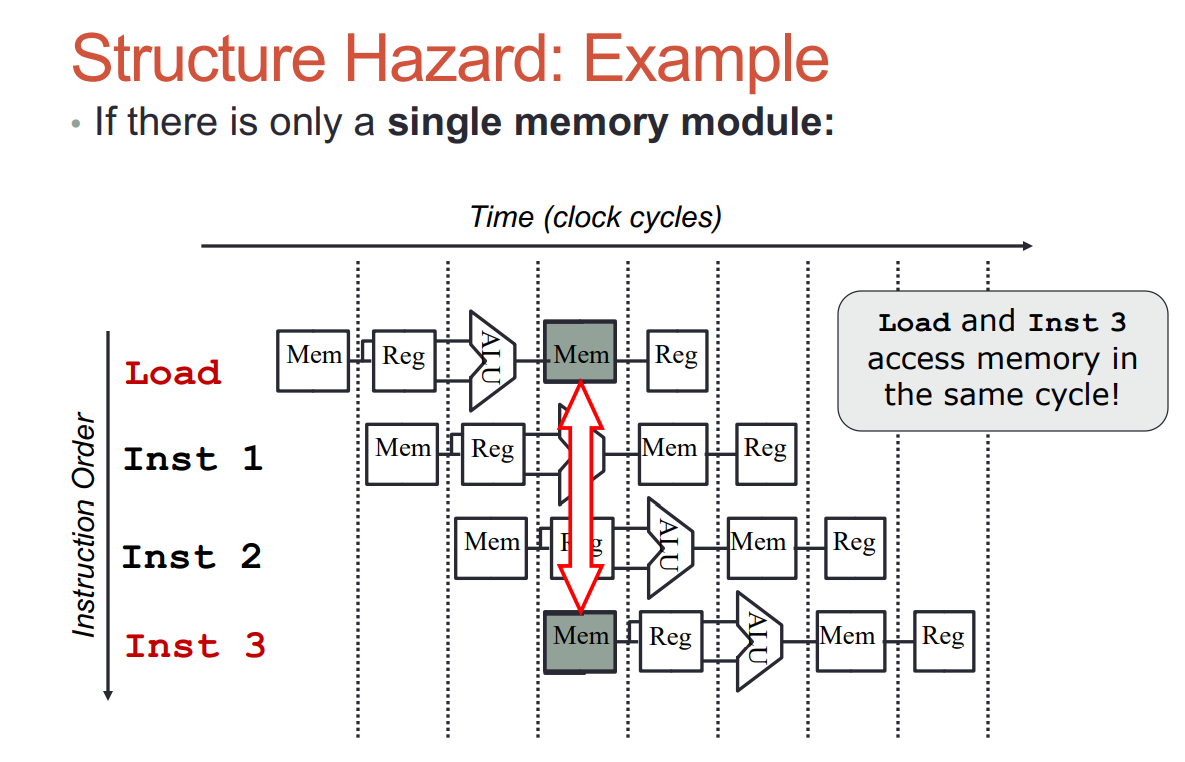
1. *Control Dependency*

- Khi mà sự thực thi lệnh này phụ thuộc lệnh khác

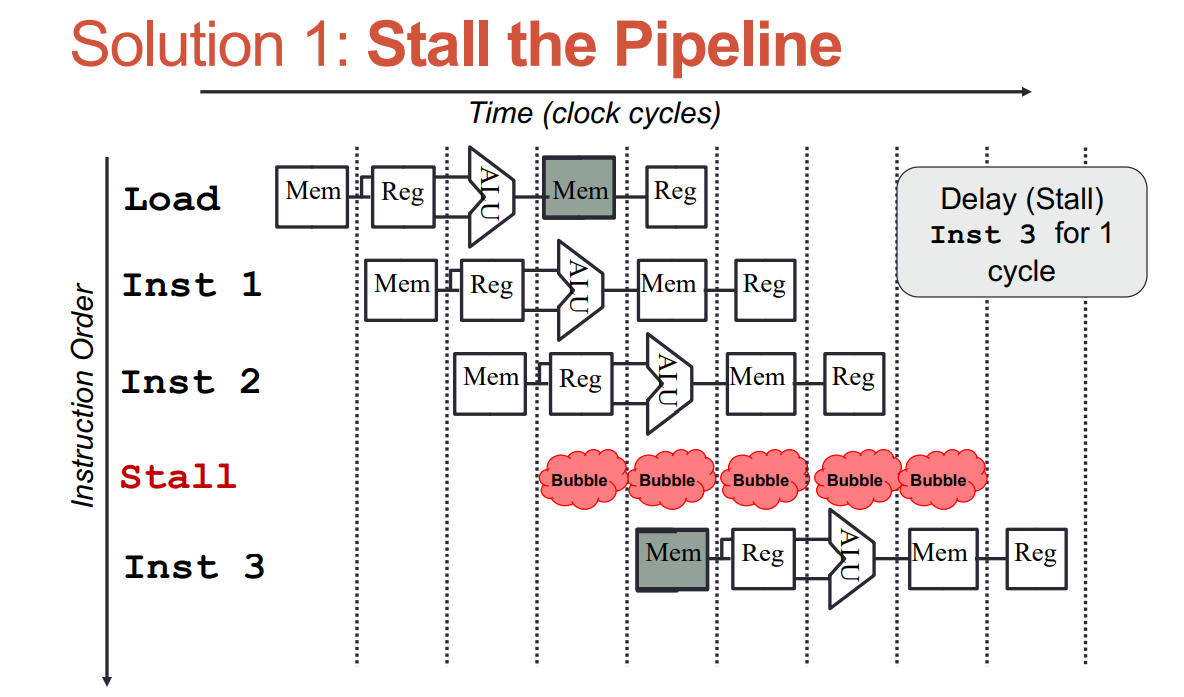
⇒ Không giải quyết các loại Dependency thì ảnh hưởng tới tính đúng đắn chương trình

1. *Các loại Hazard*
2. *Hazard dạng cấu trúc*

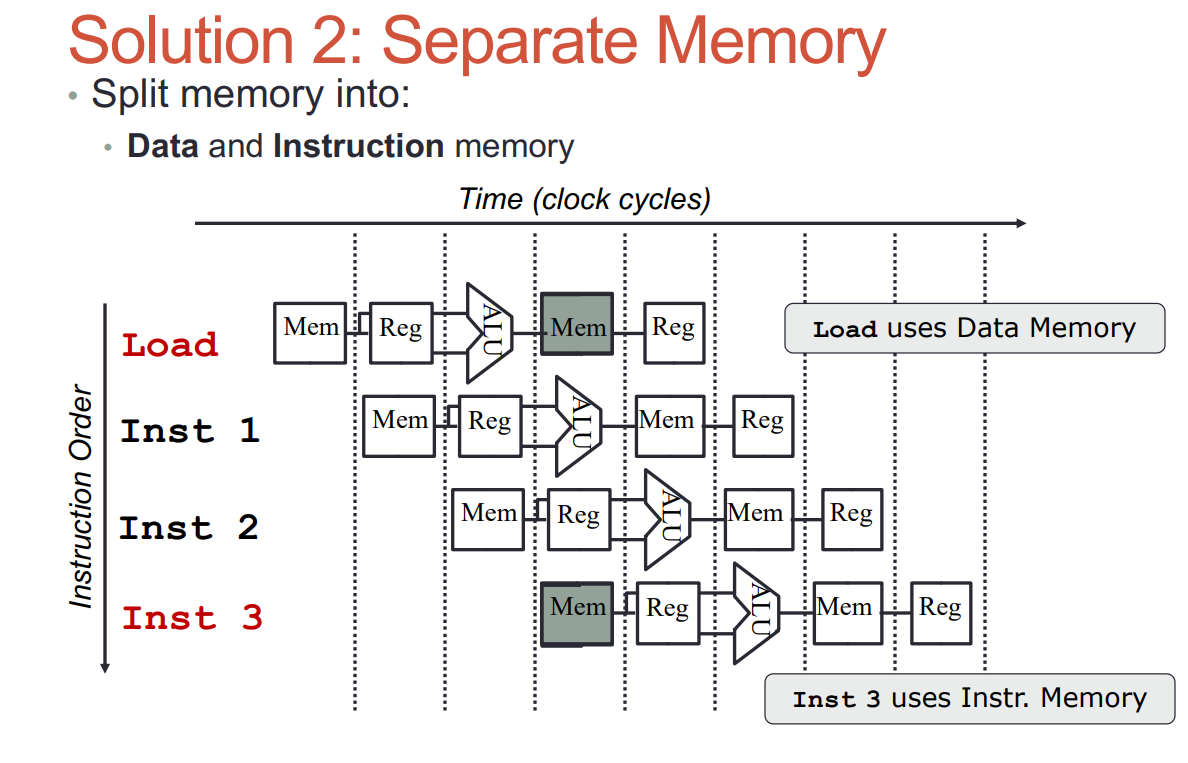
\* Hình ảnh về Structurw Hazard



\*Giải pháp 1: Stall the Pipeline (Làm trễ Pipeline)

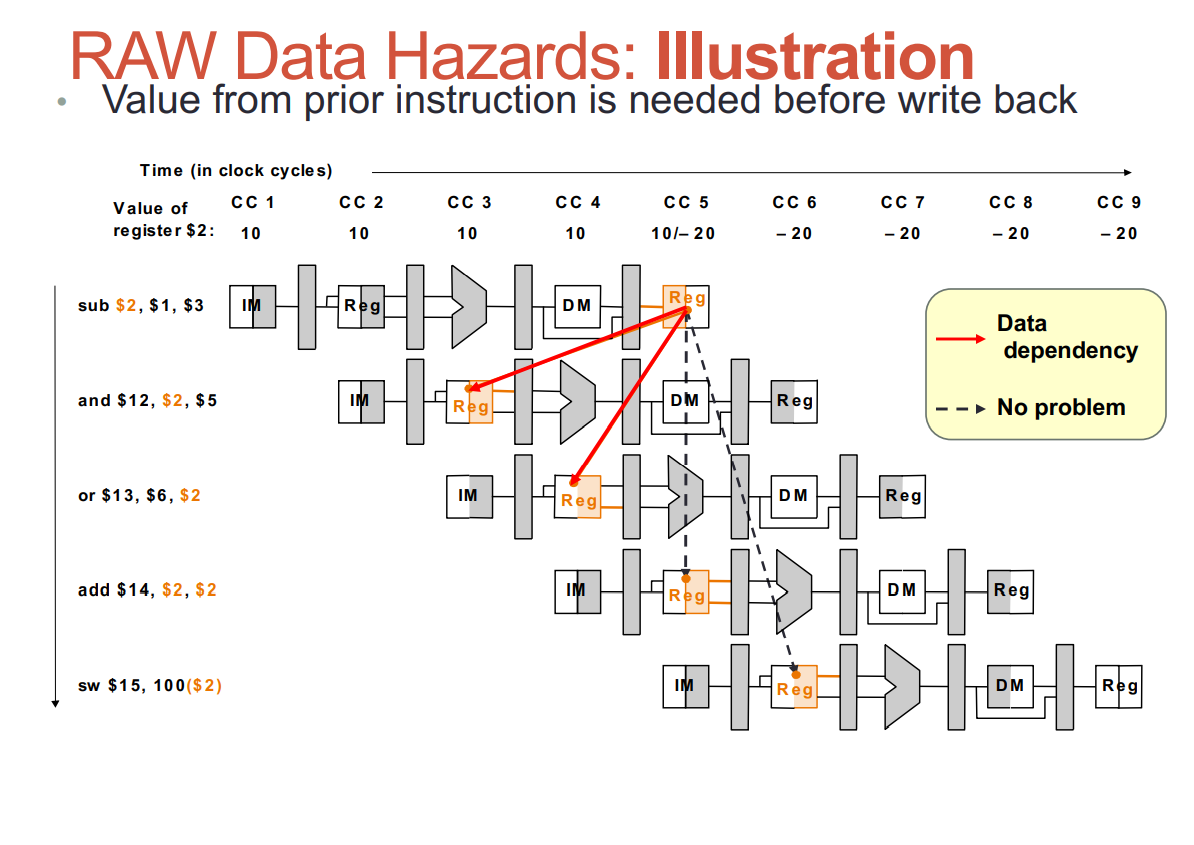


\*Giải pháp 2:Seperate Memory(Chia bộ nhớ thành bộ nhớ lệnh và bộ nhớ dữ liệu)

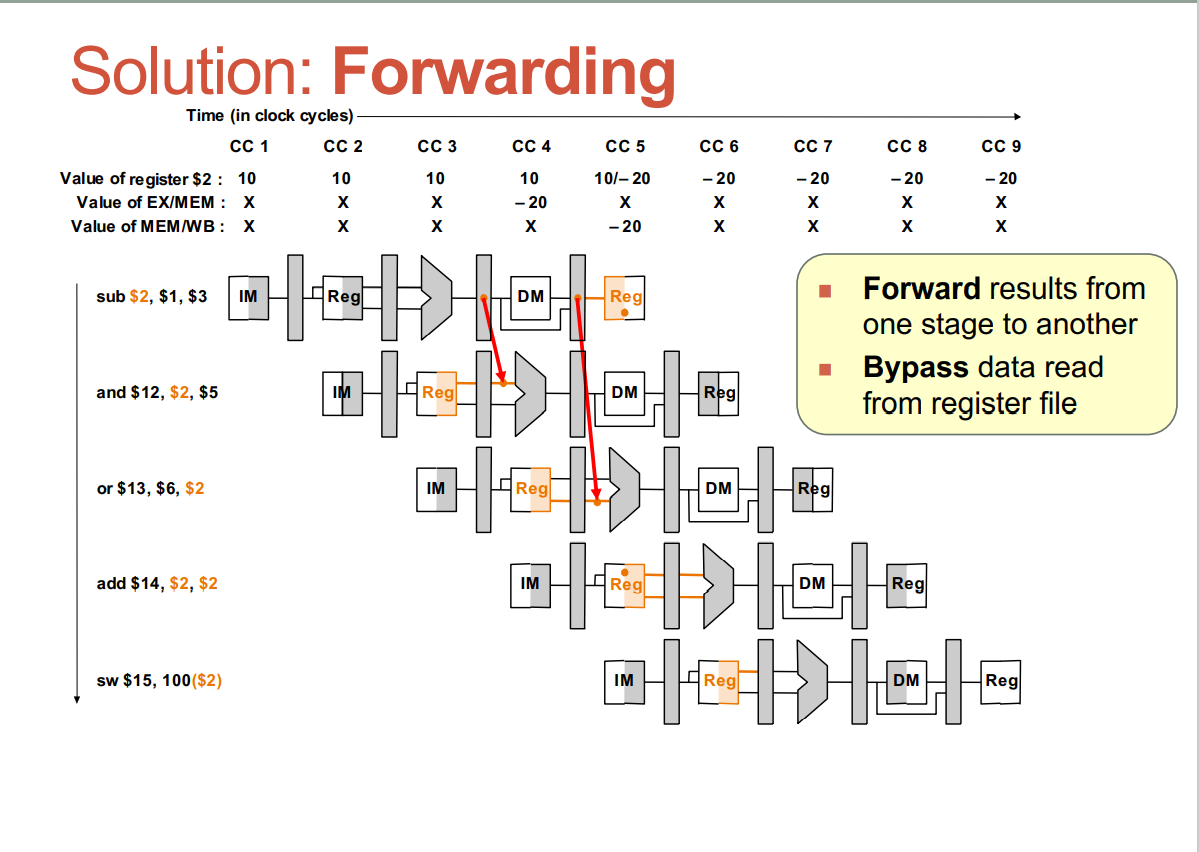


1. *Data Hazard*

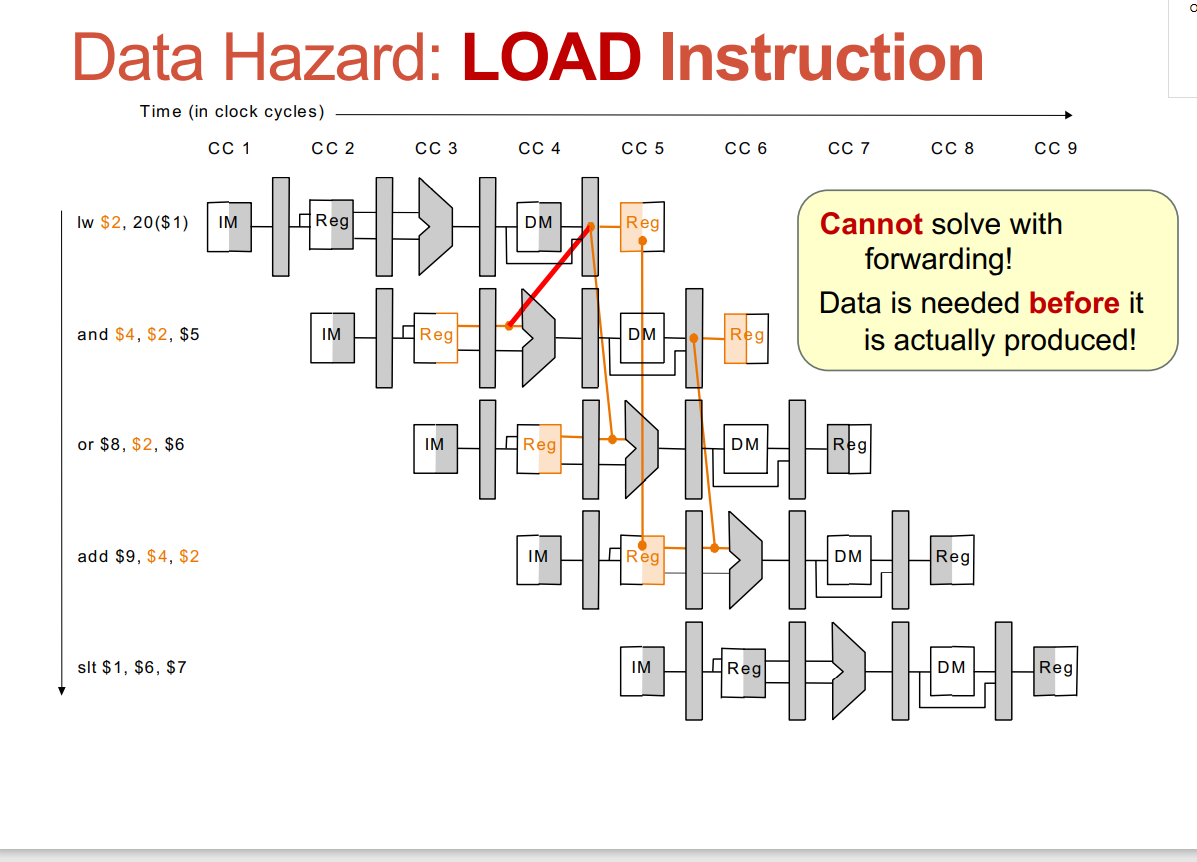
\* Hình ảnh về vấn đề RAW



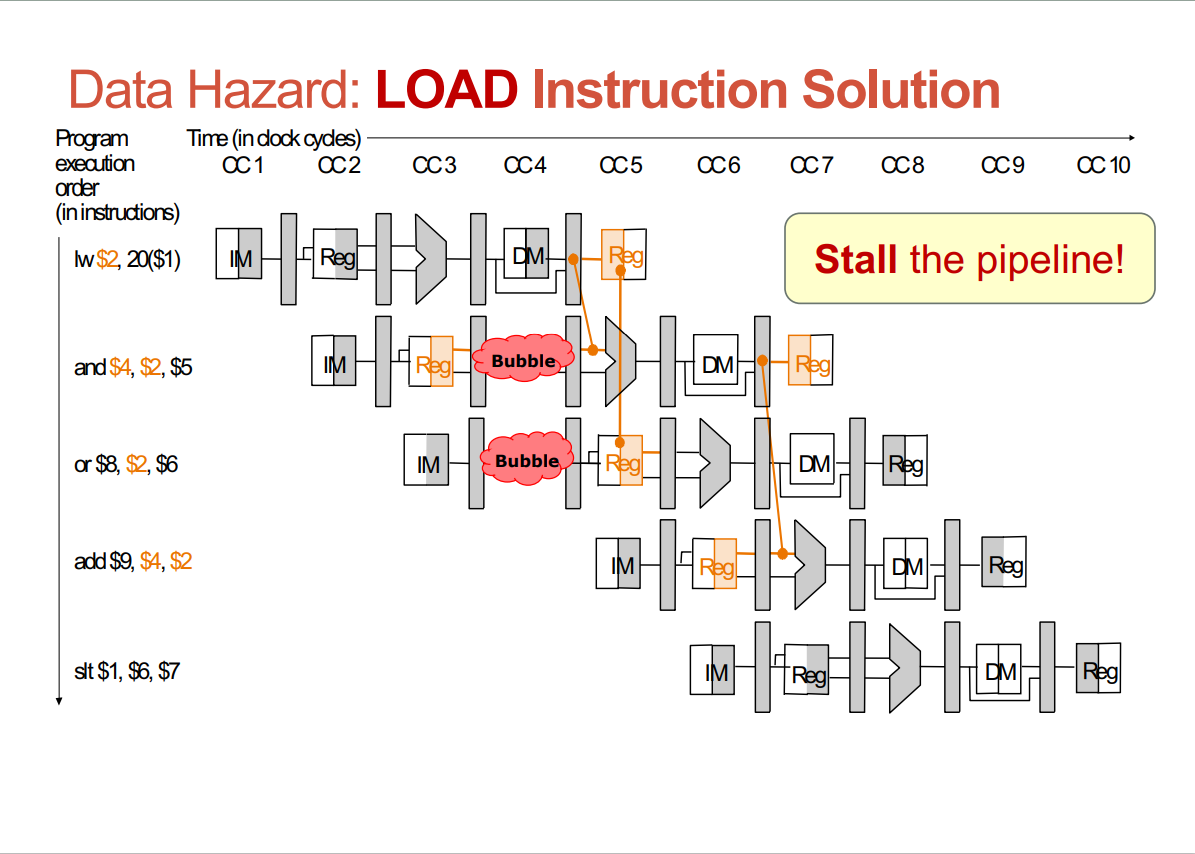
\* Cách giải quyết vấn đề đầu tiên (Forwarding):



\* Tuy vậy không thể sử dụng cách trên cho lệnh lw:

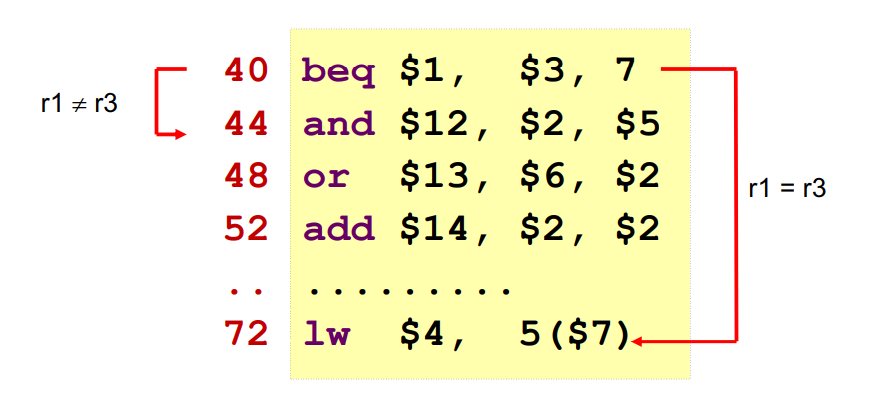


\* Cách giải quyết thứ hai(Stall The Pipeline) sẽ giải quyết vấn đề này



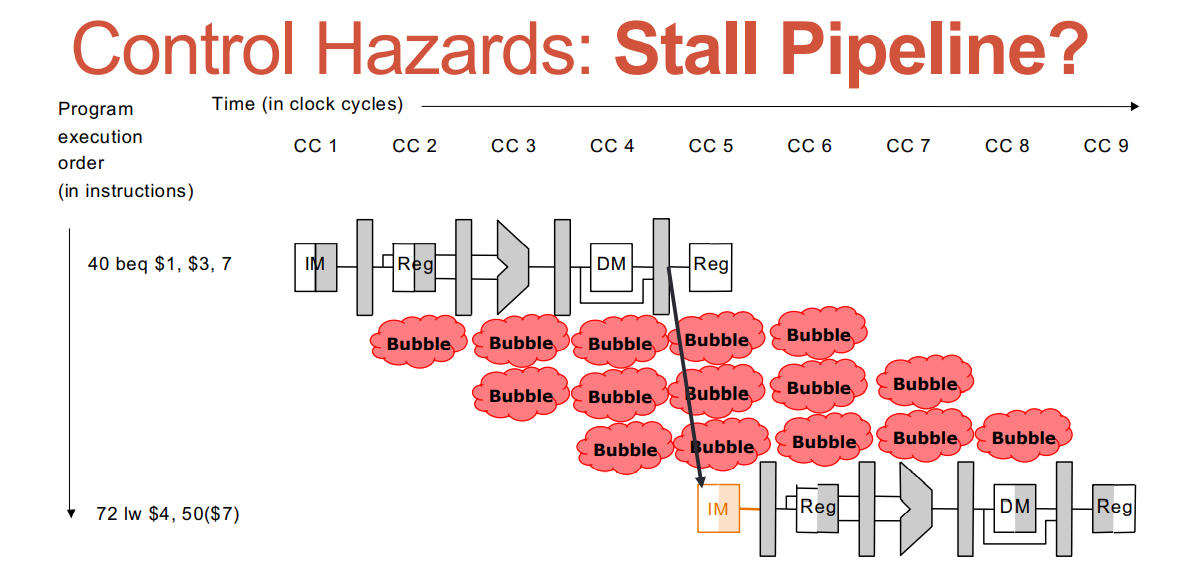
1. *Controll Hazard*

\* Hình ảnh lệnh rẽ nhánh đơn giản



\* Vấn đề nếu sắp xếp tuần tự:

Có thể tạm thời delay 3 chu kì xung clock cho mỗi lệnh để sau khi đợi hoàn thành việc check rẽ nhánh hay không rồi quyết định như hình:



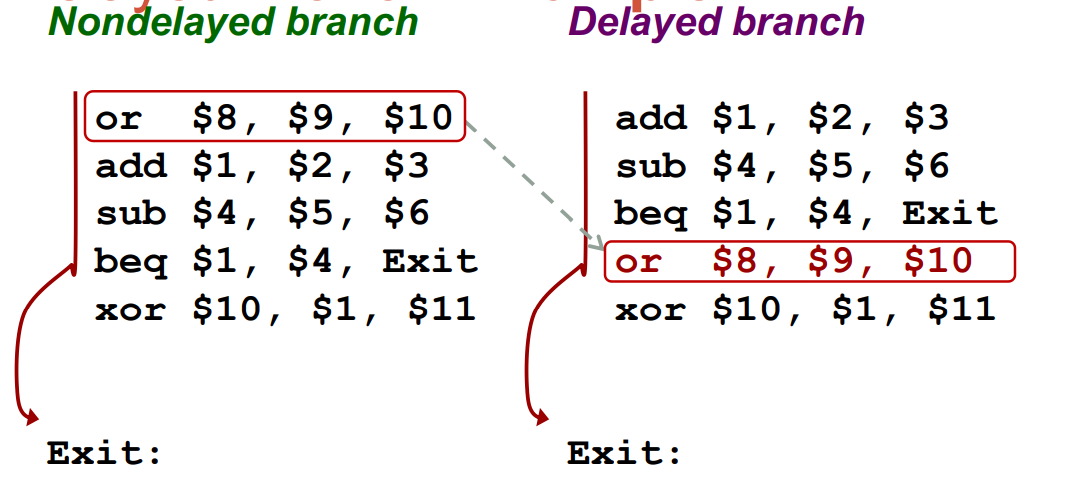
Nhưng nếu như vậy chương trình:

* Sẽ nặng(Do cứ mỗi lần rẽ nhánh cứ đợi 3cc)

⇒**Có 3 giải pháp thay thế:Early Branch(Rẽ nhánh sớm hơn),Branch Prediction(Dự đoán rẽ nhánh),Delayed Branching(Trong khi chờ đợi quyết định rẽ nhánh thì làm gì đó hữu ích)**

**\* Kỹ thuật Delayed Branching**

* Nguyên lý:Sử dụng thời gian chờ 3 chu kì xung clock để thực hiện 1 công việc hữu ích khác mà không liên quan tới các thanh ghi rẽ nhánh
* Ý tưởng:Đưa những lệnh không phụ thuộc vào việc rẽ nhánh để đưa vào sau cái thời gian/lệnh rẽ nhánh

****

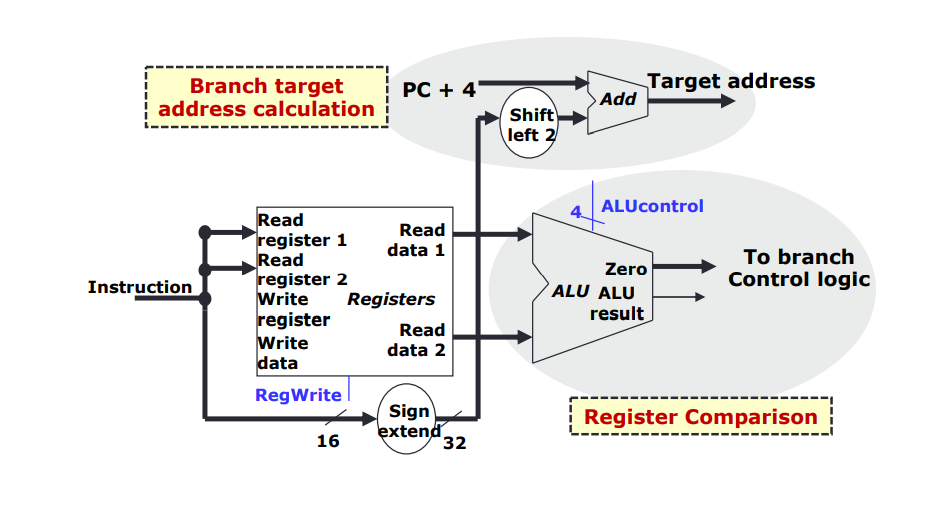
\* Kỹ thuật Early Branch

Lệnh quyết định rẽ nhánh sẽ bắt đầu thực hiện ở bước MEM khi tín hiệu IsZero được giao xong với branch tại cổng AND tạo ra tín hiệu PCSrc điều khiển cái MUX chọn giá trị thanh PC

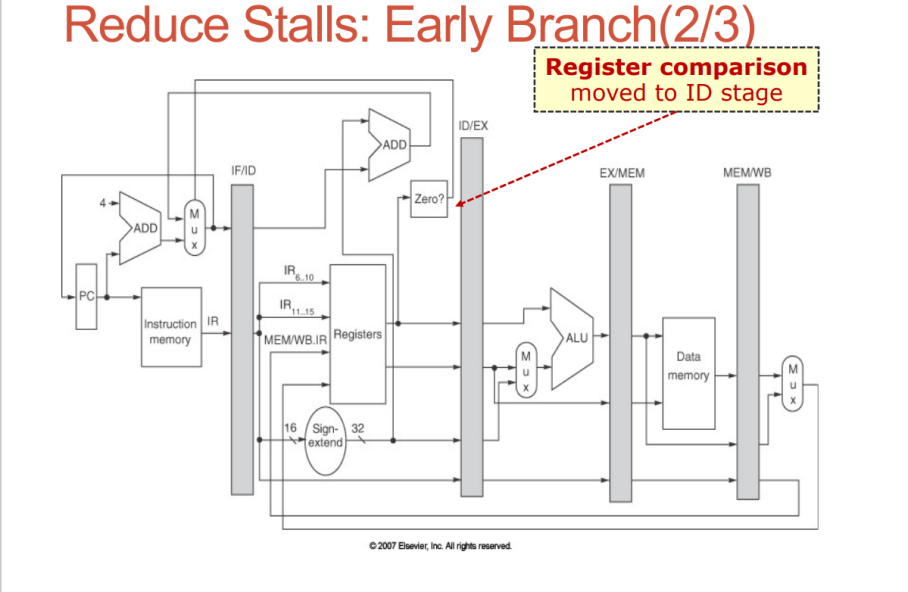
Nguyên lý:

* Chuyển bước quyết định rẽ nhánh về bước ID thay vì MEM ⇒ So sánh sẽ ở khúc Register chứ không so sánh ALU nữa ⇒ALU không làm bước so sánh nữa\*\*(Lưu ý là lúc này phần isZero sẽ được đưa về thành 1 bộ mạch riêng ở bước ID và sẽ thay thế chức năng của PCSrc làm tín hiệu cho bộ MUX chọn giá trị thanh ghi PC khi rẽ nhánh)\*\*

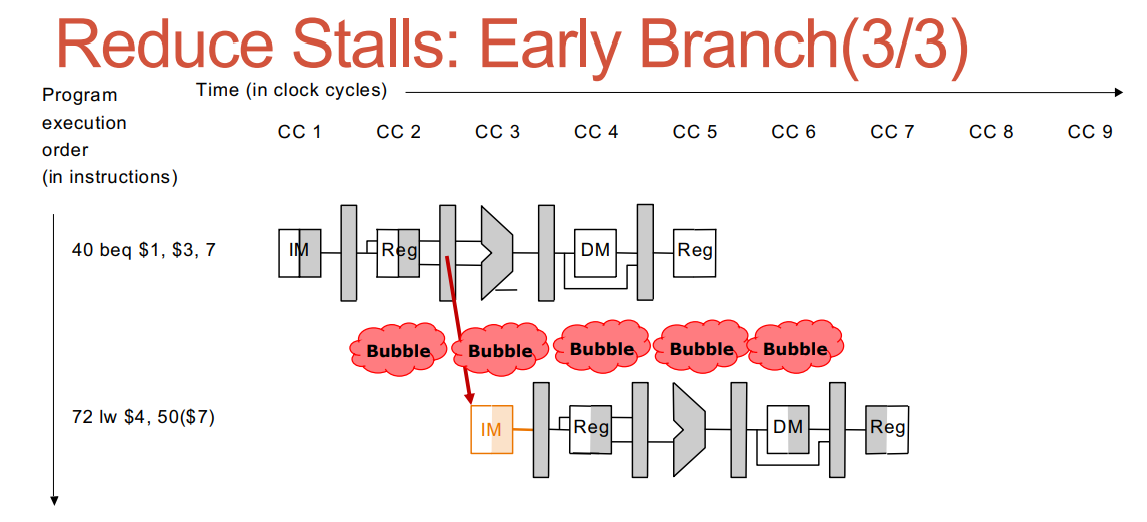
+ hình ảnh minh họa:



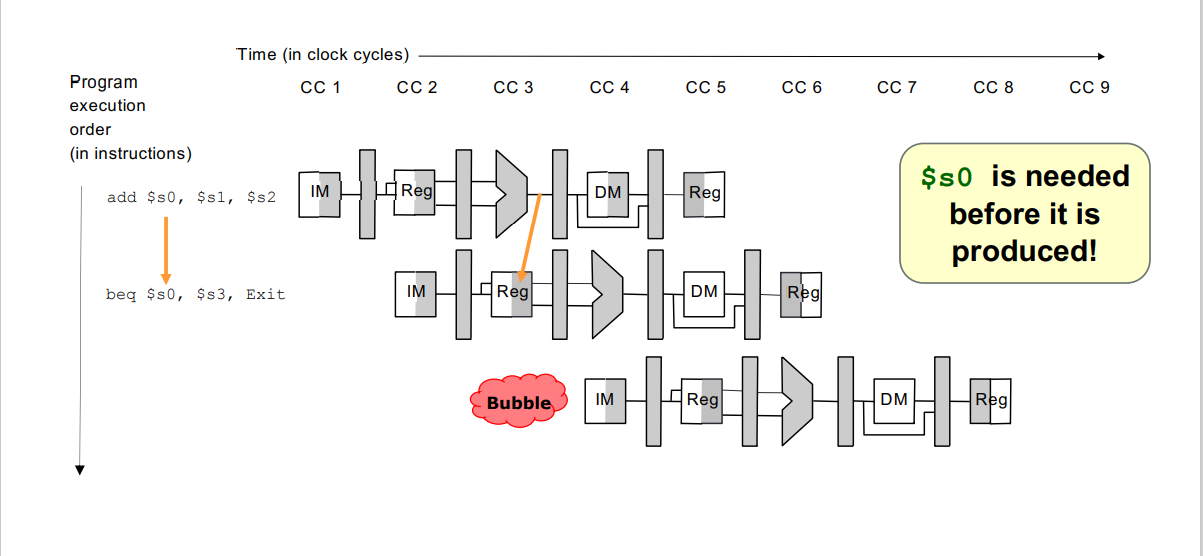
+ Hình ảnh trên Datapath được thay đổi sau khi dời về bước ID



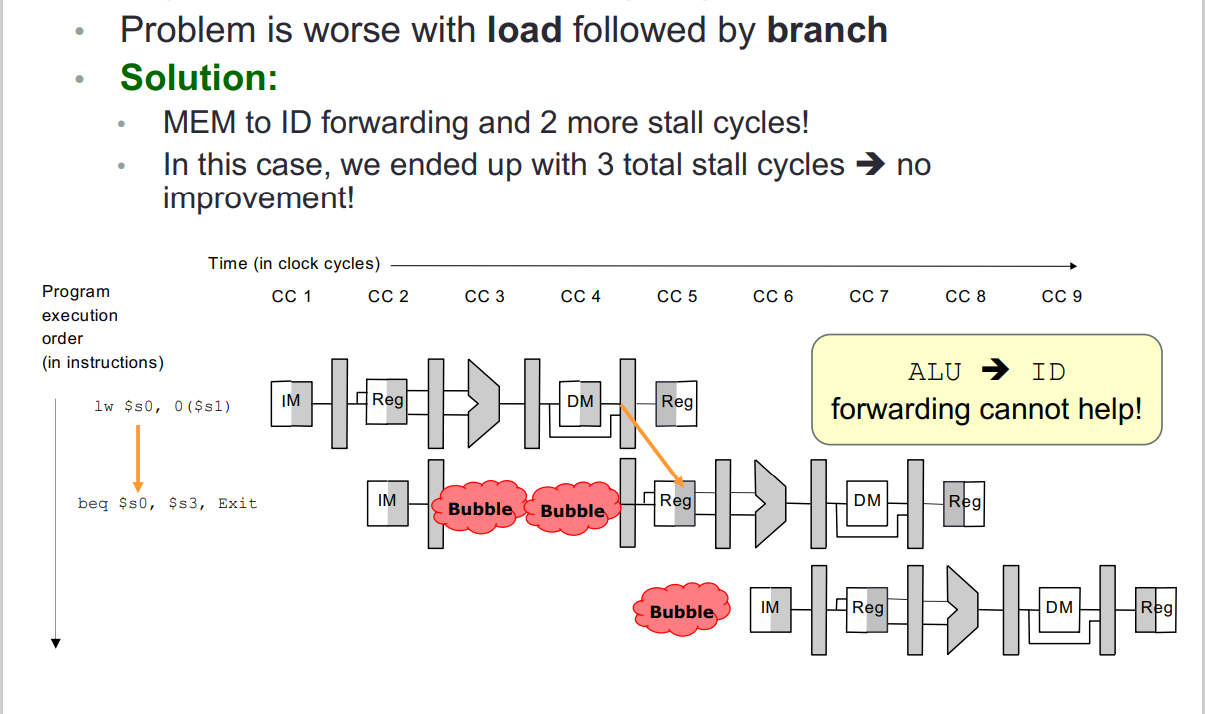
+ Kết quả:



+ Tuy vậy nếu có một lệnh nào đó tác động vào 1 thanh ghi được so sánh thì sẽ xảy ra tình trạng Data Hazard



\* Chúng ta có thể dùng tạm việc Stall the Pipeline để giải quyết tạm thời nhưng lúc này sẽ quay về việc delay 3 chu kì xung clock(Đối với lệnh Lw)⇒Dùng Branch Prediction



**\*Kĩ thuật Branch Prediction**

* Nguyên lý: Giả sử điều kiện rẽ nhánh không thỏa ⇒ Lấy lệnh kế tiếp thực hiện
* Nếu dự đoán đúng thì thôi
* Nếu dự đoán sai thì bỏ lệnh đó đi trong Pipeline

+ Hình minh họa về dự đoán sai

