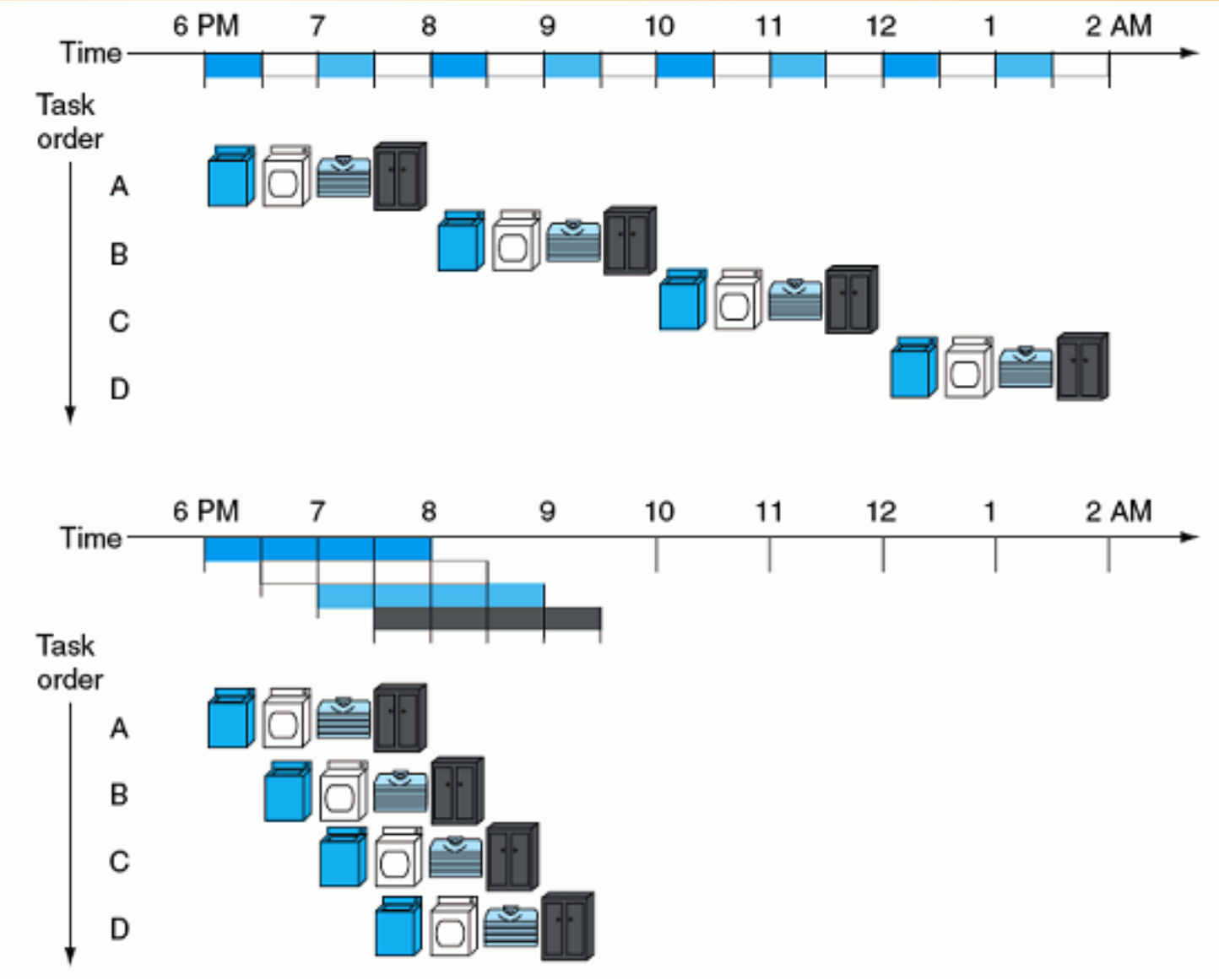
Pipeline

Môn học: Kiến trúc máy tính & Hợp ngữ

Cải thiện tốc độ?

- Có 2 cách tiếp cận phổ biến:
  - Latency: Thời gian hoàn thành 1 công việc nhất định
    - Ví dụ: Thời gian để đọc 1 sector từ đĩa gọi là disk access time hoặc disk latency
  - Throughput: Số lượng công việc có thể hoàn thành trong 1 khoảng thời gian nhất định

Giải pháp giặt ủi



Pipeline

- Pipeline không phải là giải pháp giúp tăng tốc theo kiểu Latency, mà là Throughput trên toàn bộ công việc được giao
  - Trên cùng 1 lượng tài nguyên không đổi, các công việc sẽ được tiến hành song song thay vì tuần tự, mỗi công việc chạy trong 1 pipepline (đường ống)

- *Pipelining là một kỹ thuật thực hiện lệnh trong đó các lệnh thực hiện theo kiểu "gối đầu" nhau (overlap) nhằm tận dụng những khoảng thời gian rỗi giữa các công đoạn, qua đó làm tăng tốc độ xử lý lệnh*

Pipeline

- Khả năng tăng tốc phụ thuộc vào số lượng đường ống (pipeline) sử dụng
- Thời gian để cho chảy đầy (fill) đường ống và Thời gian để làm khô (drain) sẽ làm giảm khả năng tăng tốc
  - Ví dụ giặt ủi trên nếu không tính thời gian fill và drain thì tăng tốc 4 lần, còn nếu tính thì chỉ tăng tốc được 2.3 lần

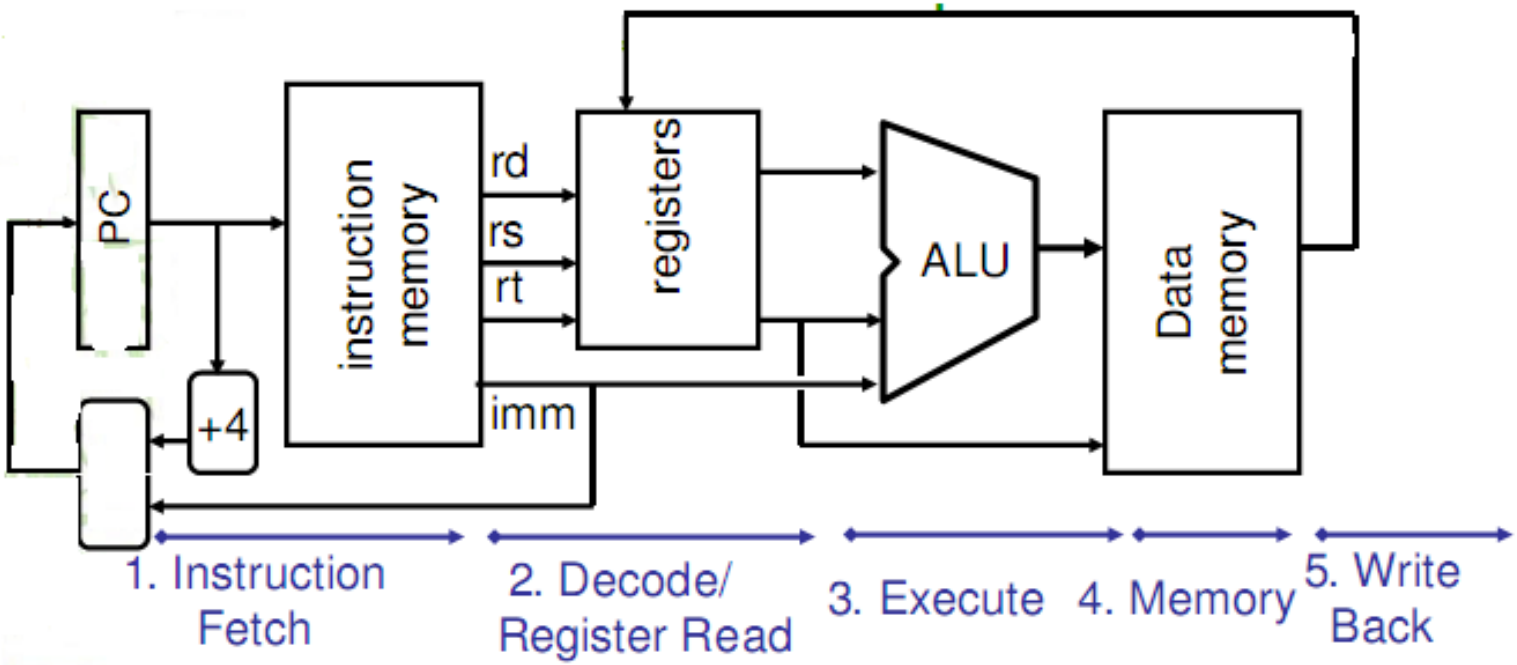
Pipeline

- Giả sử một máy giặt giặt mất 20 phút, gấp đồ mất 20 phút. Vậy khi dùng giải pháp pipeline sẽ nhanh hơn bình thường bao nhiêu?
- Tổng thời gian cho giải pháp pipeline sẽ bị giới hạn bởi thời gian thực thi của đường ống chậm nhất
- Độ dài không cân bằng giữa các đường ống sẽ làm giảm khả năng tăng tốc

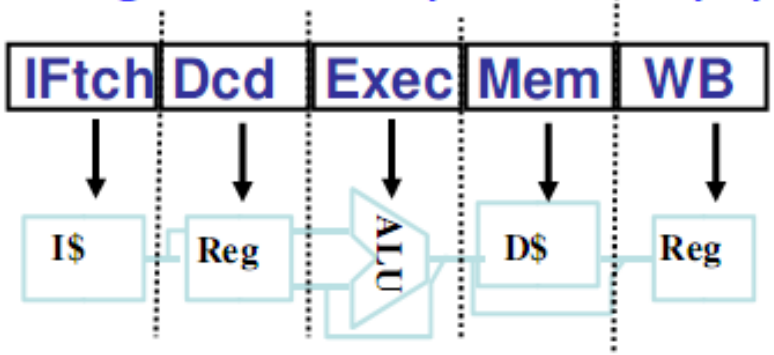
Các bước thực thi lệnh trong MIPS

- IFtch: Instruction Fetch, Increment PC (Nạp lệnh)
- Dcd: Instruction Decode, Read Registers (Giải mã lệnh)
- Exec: (Thực thi)
  - Mem-ref: Calculate Address (Tính toán địa chỉ toán hạng)
  - Arith-log: Perform Operation (Tính toán số học, luận lý)
- Mem: (Lưu chuyển với bộ nhớ)
  - Load: Read Data from Memory
  - Store: Write Data to Memory
- WB: Write Data Back to Register (Lưu dữ liệu vào thanh ghi)

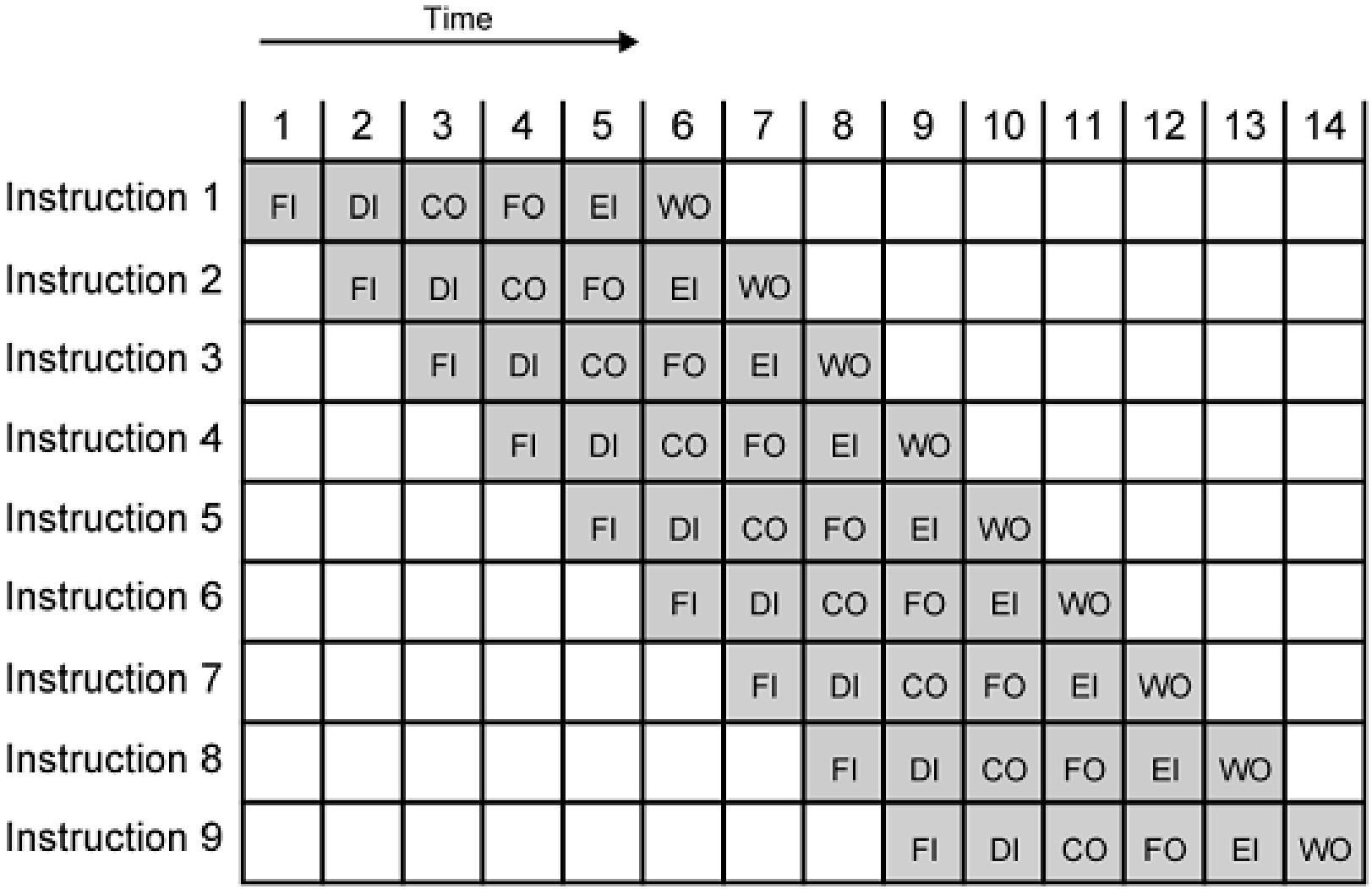
Datapath



- Use datapath figure to represent pipeline

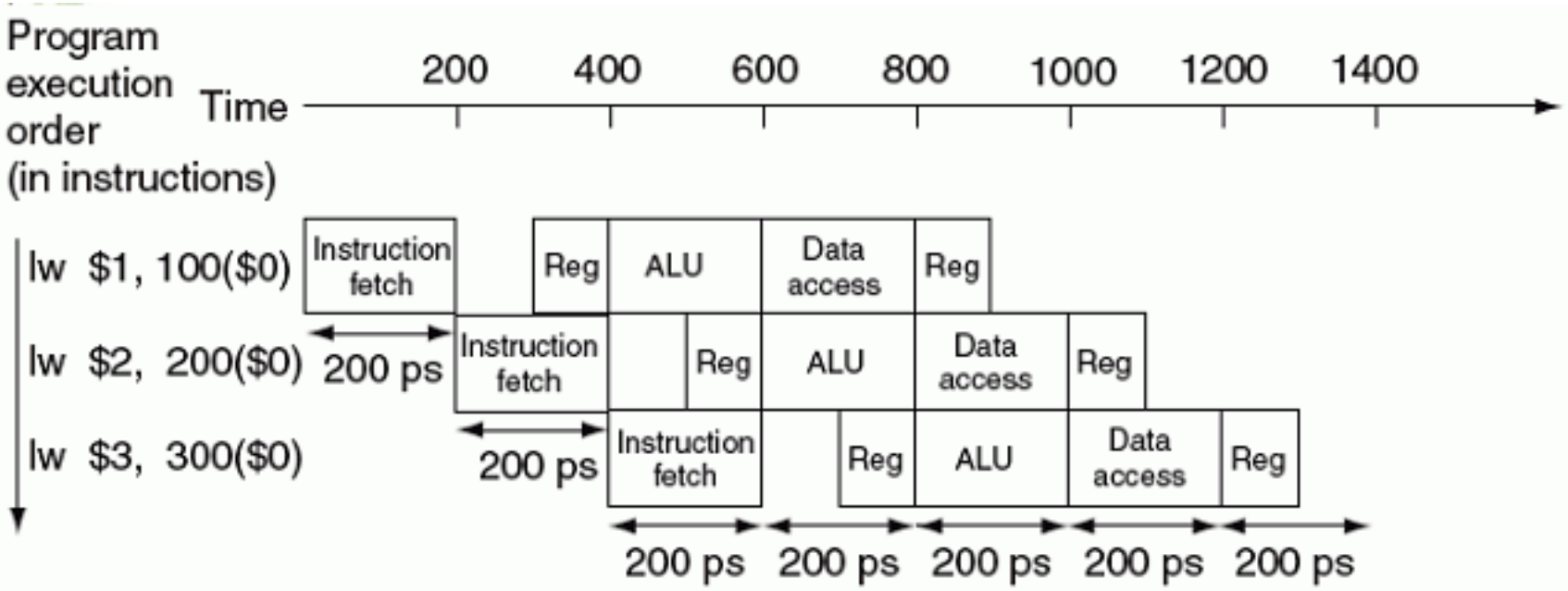


Ý tưởng Pipeline



Ví dụ

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

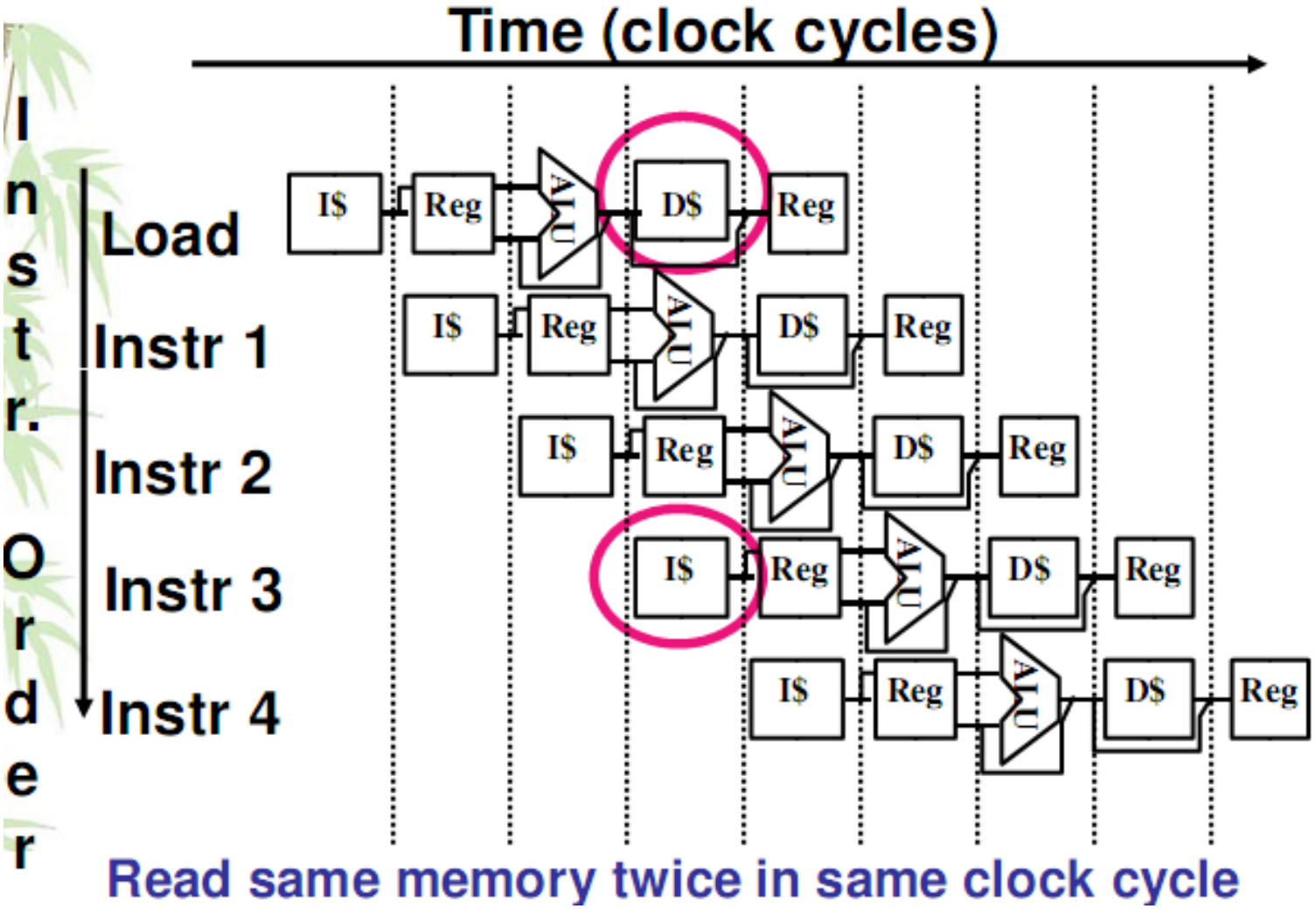


Các trở ngại (Hazards) của pipeline

- Structural hazards: do nhiều lệnh dùng chung một tài nguyên tại 1 thời điểm
- Data hazard: lệnh sau sử dụng dữ liệu kết quả của lệnh trước
- Control hazard: do rẽ nhánh gây ra, lệnh sau phải đợi kết quả rẽ nhánh của lệnh trước

→ Gây ra hiện tượng "stalls" hoặc "bubbles" trong pipeline

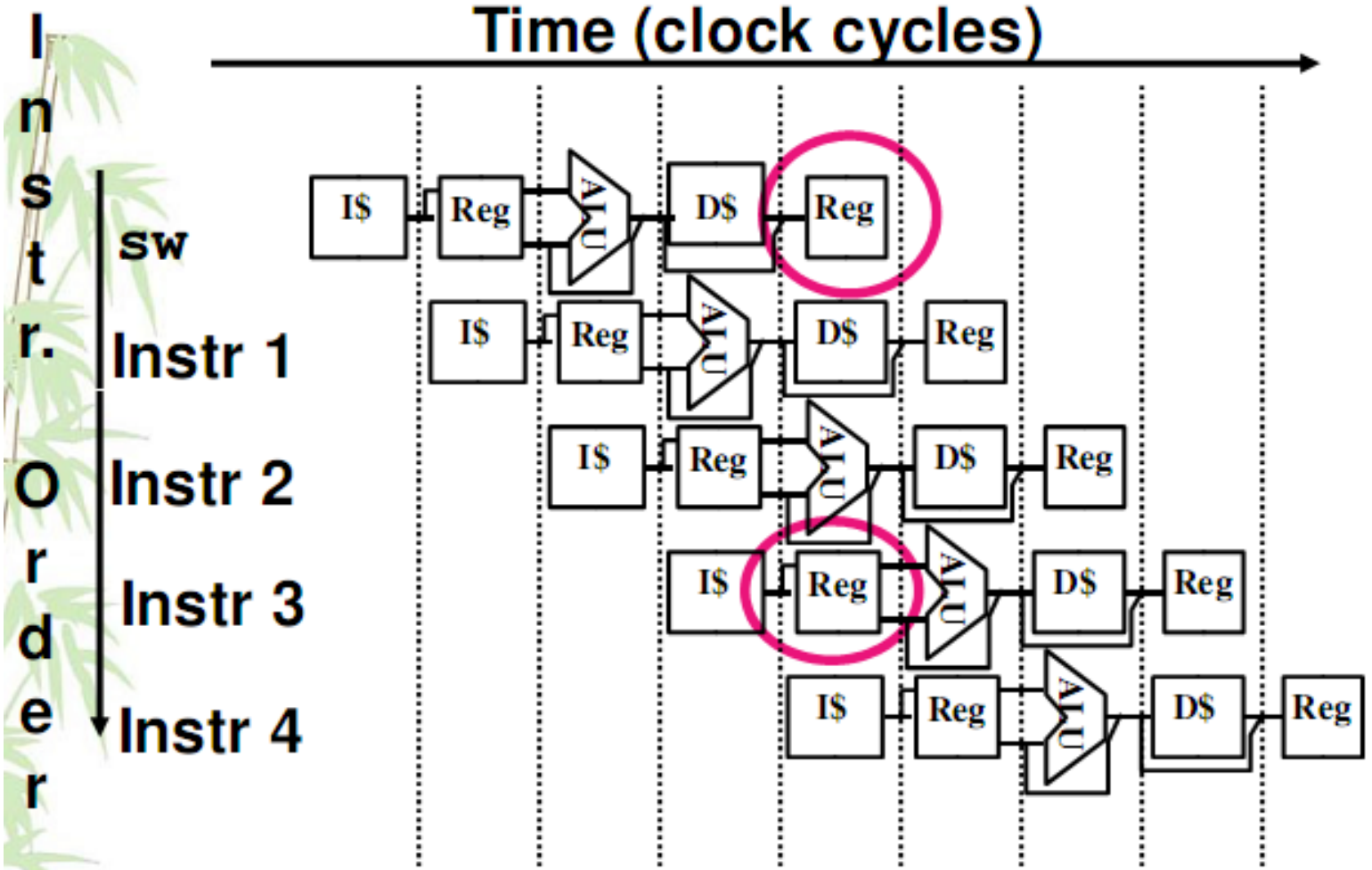
Structural hazards #1: Single memory



Structural hazards #1: Single memory

- Giải pháp:
  - Tạo 2 bộ nhớ đệm Cache Level 1 trên CPU
    - L1 Instruction Cache và L1 Data Cache
  - Cần những phần cứng phức tạp hơn để điều khiển khi không có cả 2 bộ nhớ đệm này

Structural hazards #2: Registers

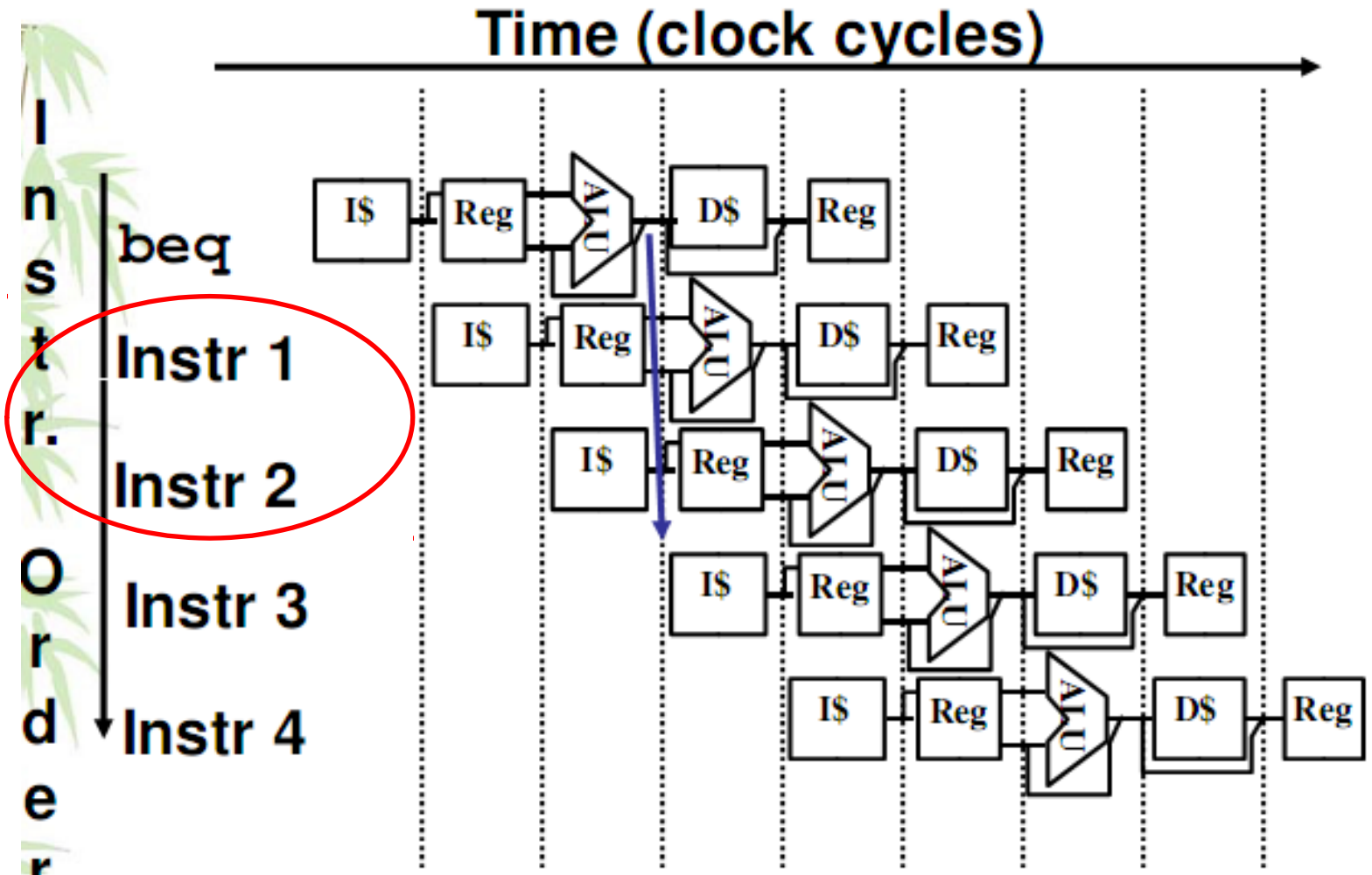


Can we read and write to registers simultaneously?

Structural hazards #2: Registers

- Có 2 giải pháp khác nhau được dùng:
  - RegFile có tốc độ truy cập rất nhanh (thường ít hơn 1 nửa thời gian thực thi trên ALU tính trên 1 chu kỳ clock)
    - Write vào RegFile trong suốt nửa đầu chu kỳ clock
    - Read từ RegFile trong nửa chu kỳ clock còn lại
  - Tạo RegFile với 2 ngõ Read và Write độc lập

Control hazard: Rẽ nhánh



Where do we do the compare for the branch?

Control hazard: Rẽ nhánh

- Chúng ta phải đặt điều kiện rẽ nhánh vào trong ALU
  - Do vậy sẽ có ít nhất 2 lệnh sau phần rẽ nhánh sẽ được fetch, bất kể điều kiện rẽ nhánh có thực hiện hay không
- Nếu chúng ta không thực hiện rẽ nhánh → Cứ thực thi theo trình tự bình thường
- Ngược lại, đừng thực thi bất kỳ lệnh nào sau điều kiện rẽ nhánh, cứ nhảy đến label tương ứng

Control hazard: Rẽ nhánh

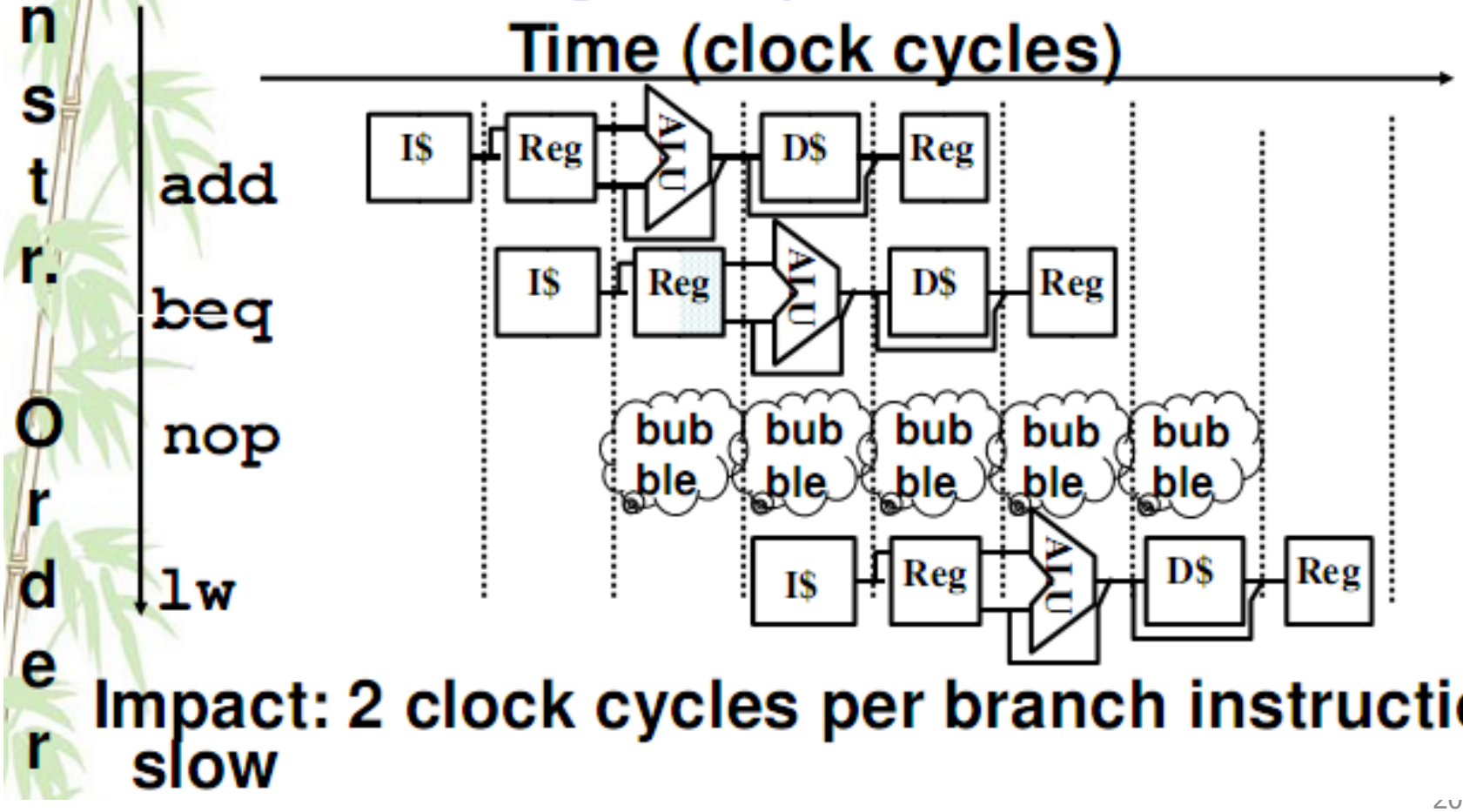
- Giải pháp ban đầu: Trì hoãn (stall) cho đến khi điều kiện rẽ nhánh được thực hiện
  - Chèn những lệnh rác "no-op" (chẳng thực hiện việc gì, chỉ để trì hoãn thời gian) hoặc hoãn việc nạp (fetch) sang lệnh kế (trong 2 chu kỳ clock)
  - Nhược điểm: Điều kiện rẽ nhánh phải làm đến 3 chu kỳ clock

Control hazard: Optimization 1

- Giải pháp tối ưu hoá 1:
  - Chèn thêm các phép so sánh rẽ nhánh đặc biệt tại Stage 2 (decode)
  - Ngay sau khi lệnh được decode, lập tức quyết định giá trị mới cho thanh ghi PC
  - Lợi ích: Bởi vì điều kiện rẽ nhánh đã làm xong trong stage 2, nên chỉ có 1 lệnh không cần thiết được nạp → chỉ cần 1 no-op là đủ

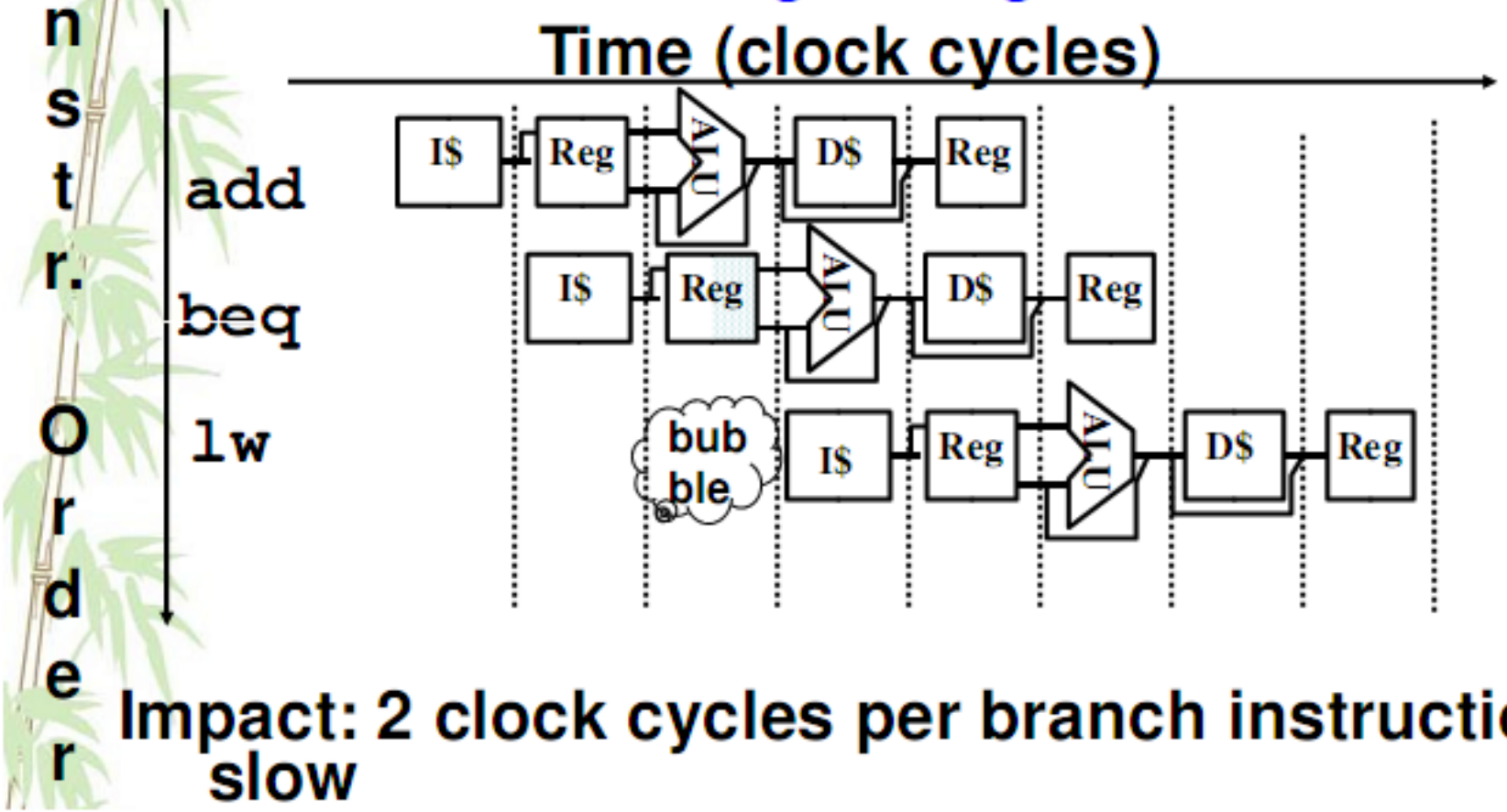
Minh hoạ

- User inserting no-op instruction



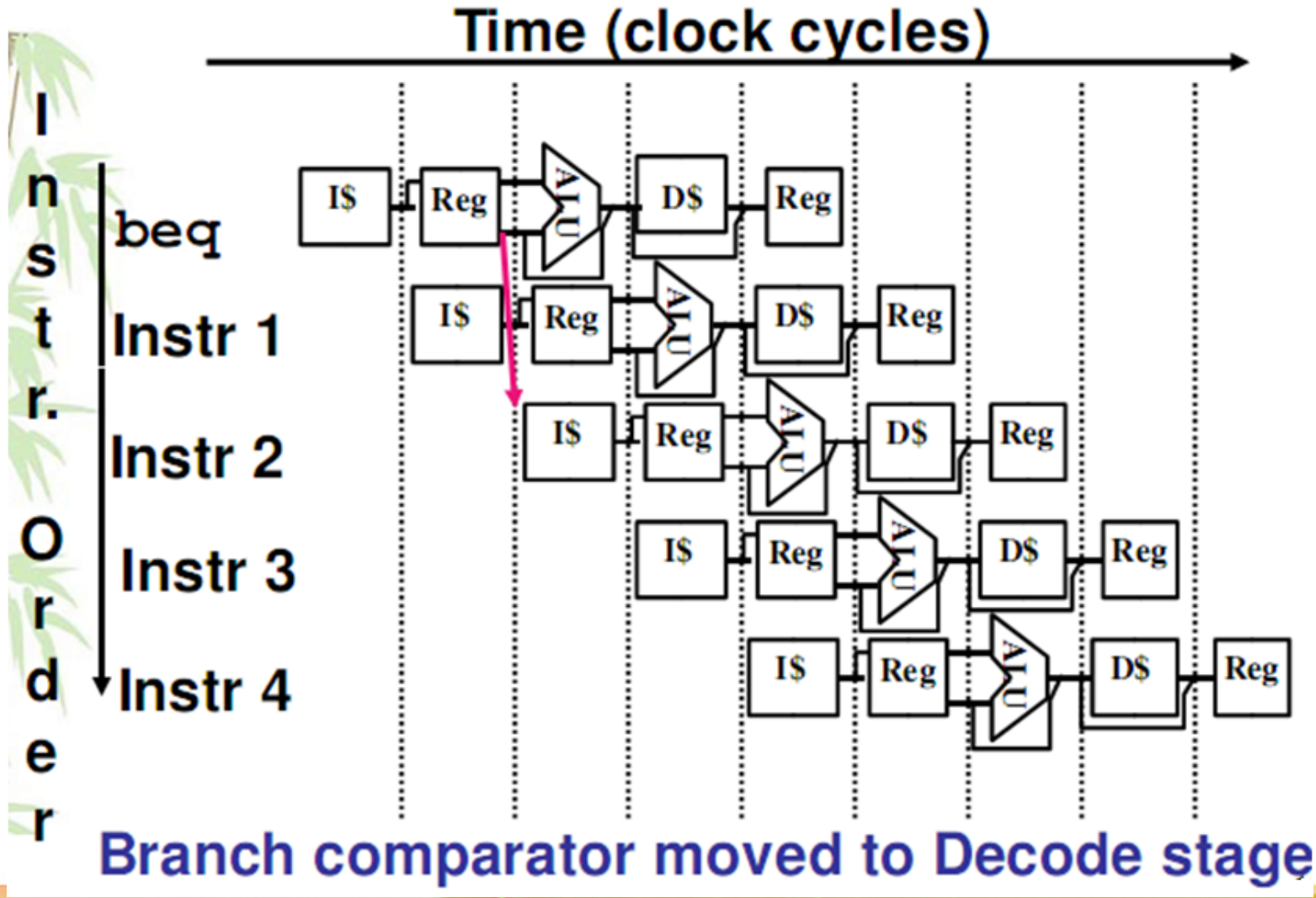
Minh hoạ

- Controller inserting a single bubble



Impact: 2 clock cycles per branch instruction slow

Minh hoạ


Time (clock cycles)
Instr. Order
beq
Instr 1
Instr 2
Instr 3
Instr 4
I$
Reg
ALU
D$
Reg
Branch comparator moved to Decode stage

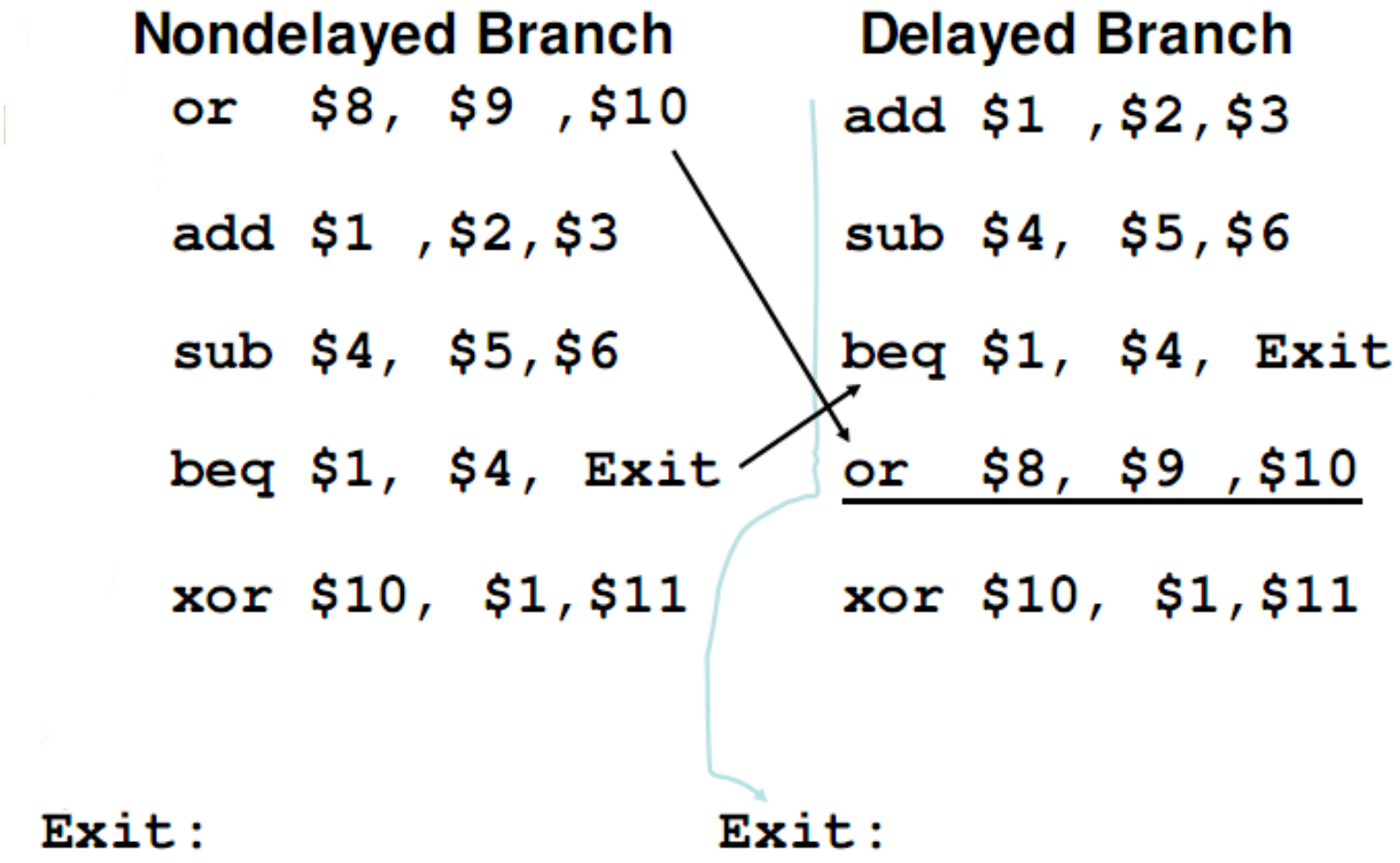
Control hazard: Optimization 2

- Giải pháp tối ưu hoá 2: Tái định nghĩa rẽ nhánh
  - Định nghĩa cũ: Nếu chúng ta thực hiện rẽ nhánh thì sẽ không có bất kỳ lệnh nào sau lệnh rẽ nhánh được làm một cách "vô tình" (không mong muốn)
  - Định nghĩa mới: Bất cứ khi nào thực hiện rẽ nhánh, một lệnh ngay sau lệnh rẽ nhánh sẽ lập tức được thực thi (gọi là branch-delay slot)
  - Ý nghĩa: Chúng ta luôn thực thi 1 lệnh ngay phía sau lệnh rẽ nhánh

Control hazard: Optimization 2

- Lưu ý về Branch-Delay Slot:
  - Trường hợp xấu nhất: có thể luôn phải đặt 1 lệnh no-op vào trong branch-delay slot
  - Trường hợp tốt hơn: có thể tìm được 1 lệnh trước lệnh rẽ nhánh để đặt trong branch-delay slot mà vẫn không làm ảnh hưởng chương trình
    - Thủ công: Tái cấu trúc thứ tự lệnh là cách làm phổ biến
    - Tự động: Compiler phải rất thông minh để tìm lệnh làm điều này

Nondelayed vs. Delayed

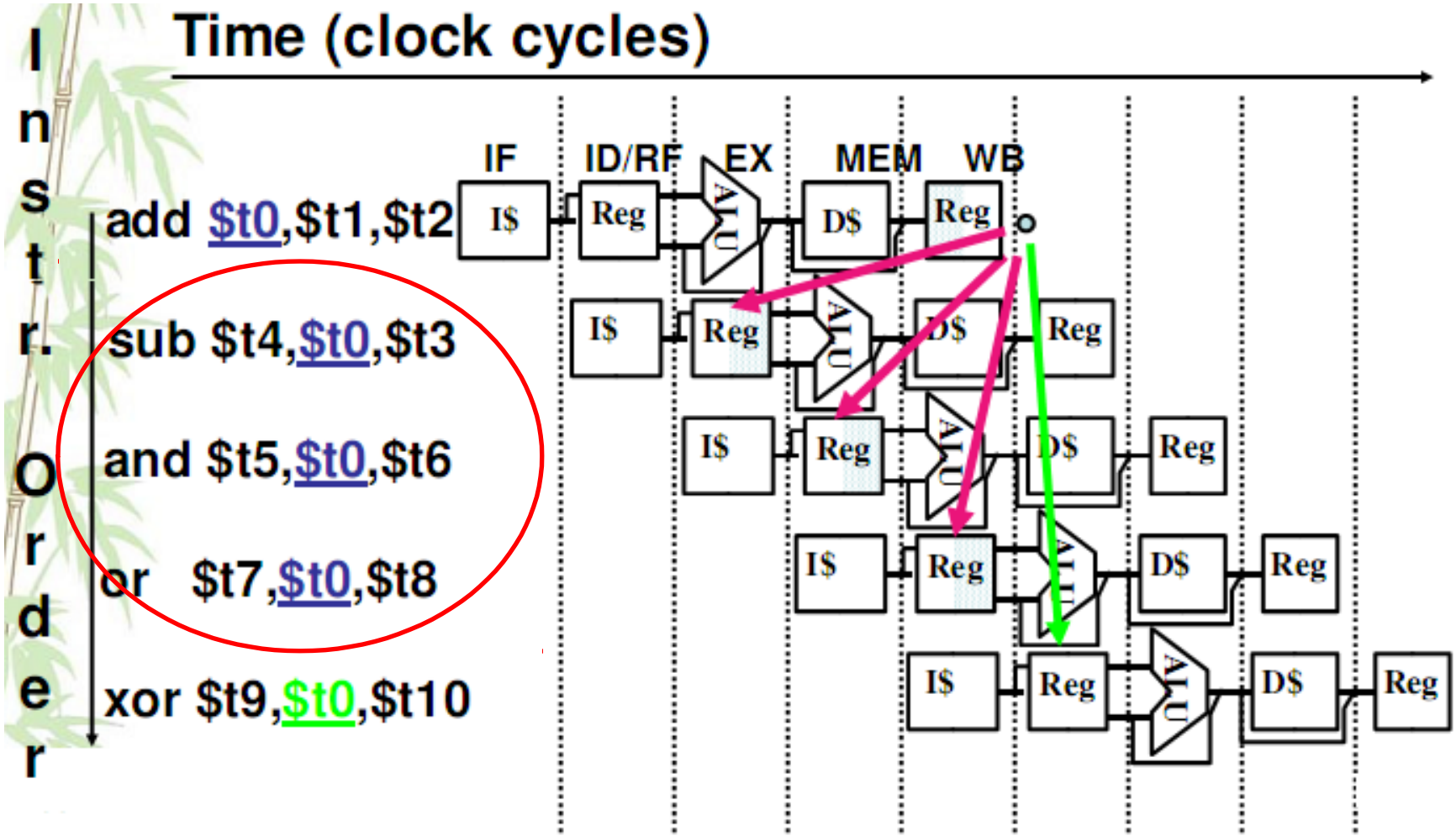


Data hazards

- Xem xét dãy lệnh sau:

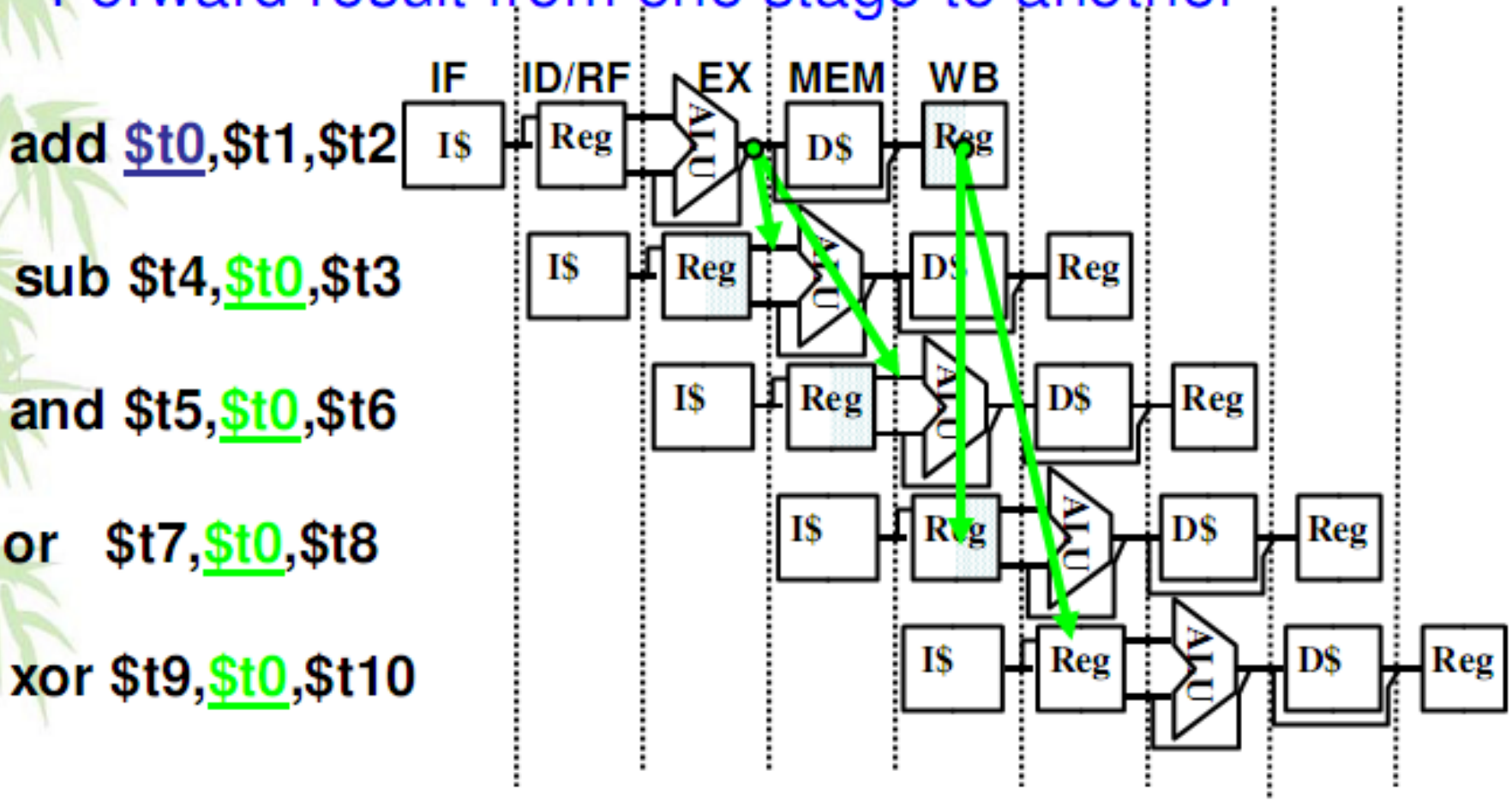
```
add  $t0,  $t1,  $t2
sub  $t4,  $t0,  $t3
and  $t5,  $t0,  $t6
or   $t7,  $t0,  $t8
xor  $t9,  $t0,  $t10
```

Data hazards

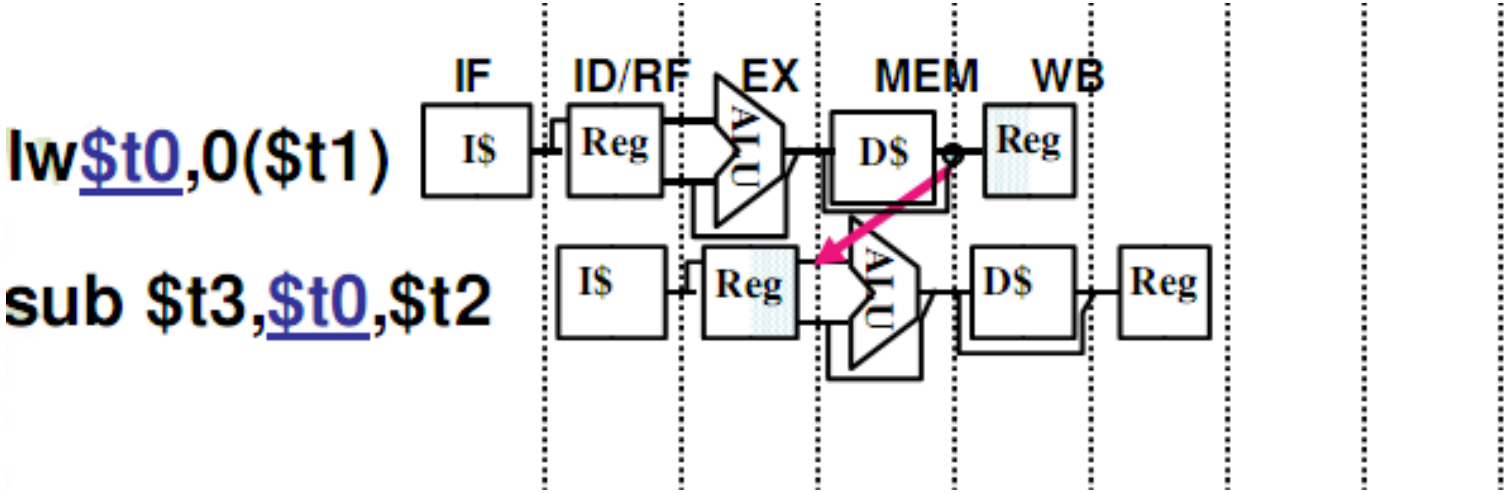


Giải pháp Data hazards: Forwarding

- Forward result from one stage to another



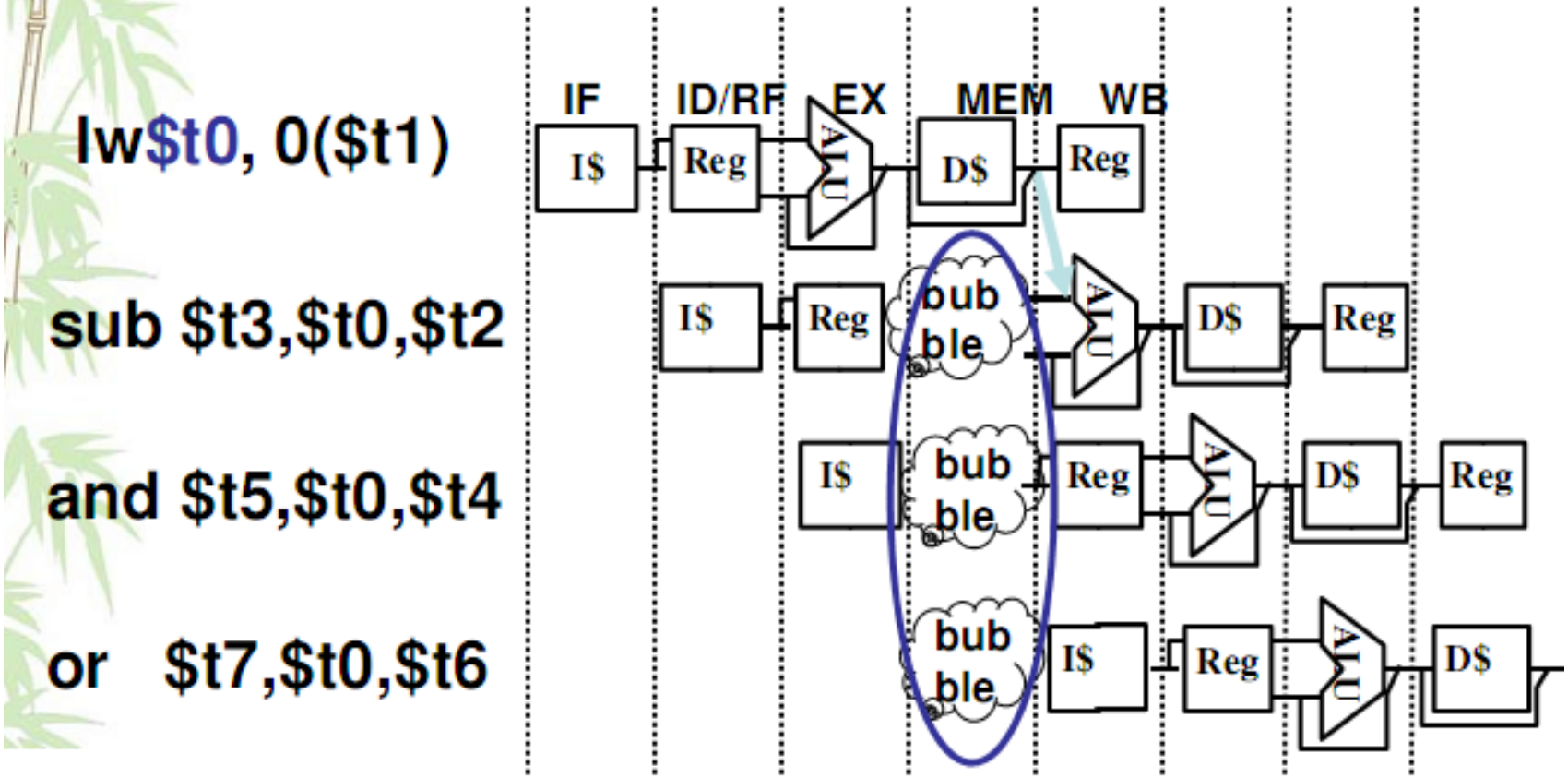
Forwarding không giải quyết được...



- Giải pháp: Phải trì hoãn lệnh sub lại (stall) sau đó mới dùng Forwarding được

Data hazards: Loads

- Hardware stalls pipeline
  - Called "interlock"



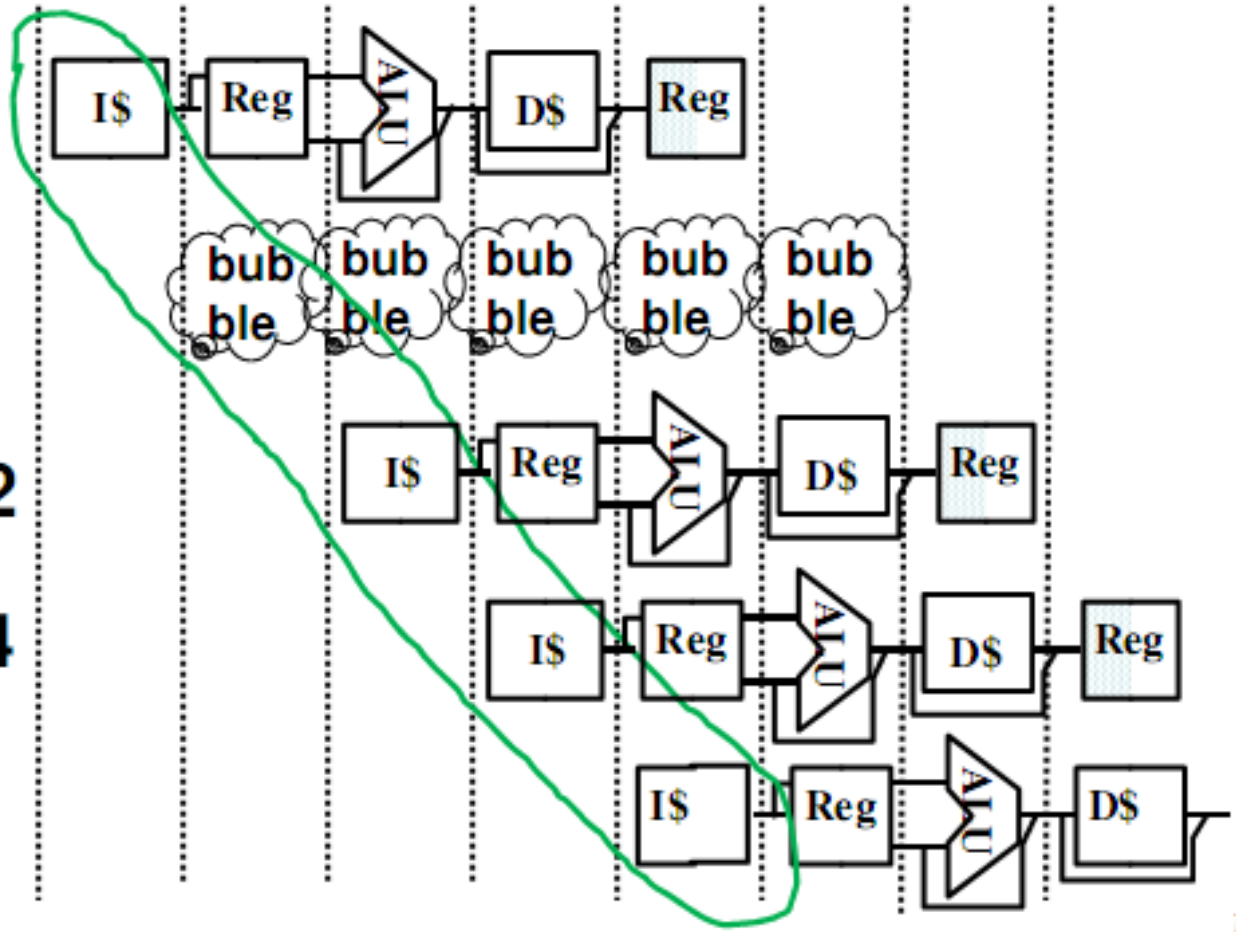
Data hazards: Loads

- Vị trí lệnh (instruction slot) sau một load được gọi là "load delay slot"
- Nếu lệnh đó dùng kết quả của load, thì hardware interlock có thể sẽ hoãn (stall) nó đúng 1 chu kỳ clock
- Nếu sau load là 1 lệnh không liên quan, thì không cần trì hoãn (stall) lệnh đó

Data hazards: Loads

- Stall is equivalent to nop

```
lw$t0, 0($t1)
nop
sub $t3,$t0,$t2
and $t5,$t0,$t4
or  $t7,$t0,$t6
```



Homework

- Sách Petterson & Hennessy: Đọc 6.1