

Bộ xử lý đường ống

Processor Pipelining

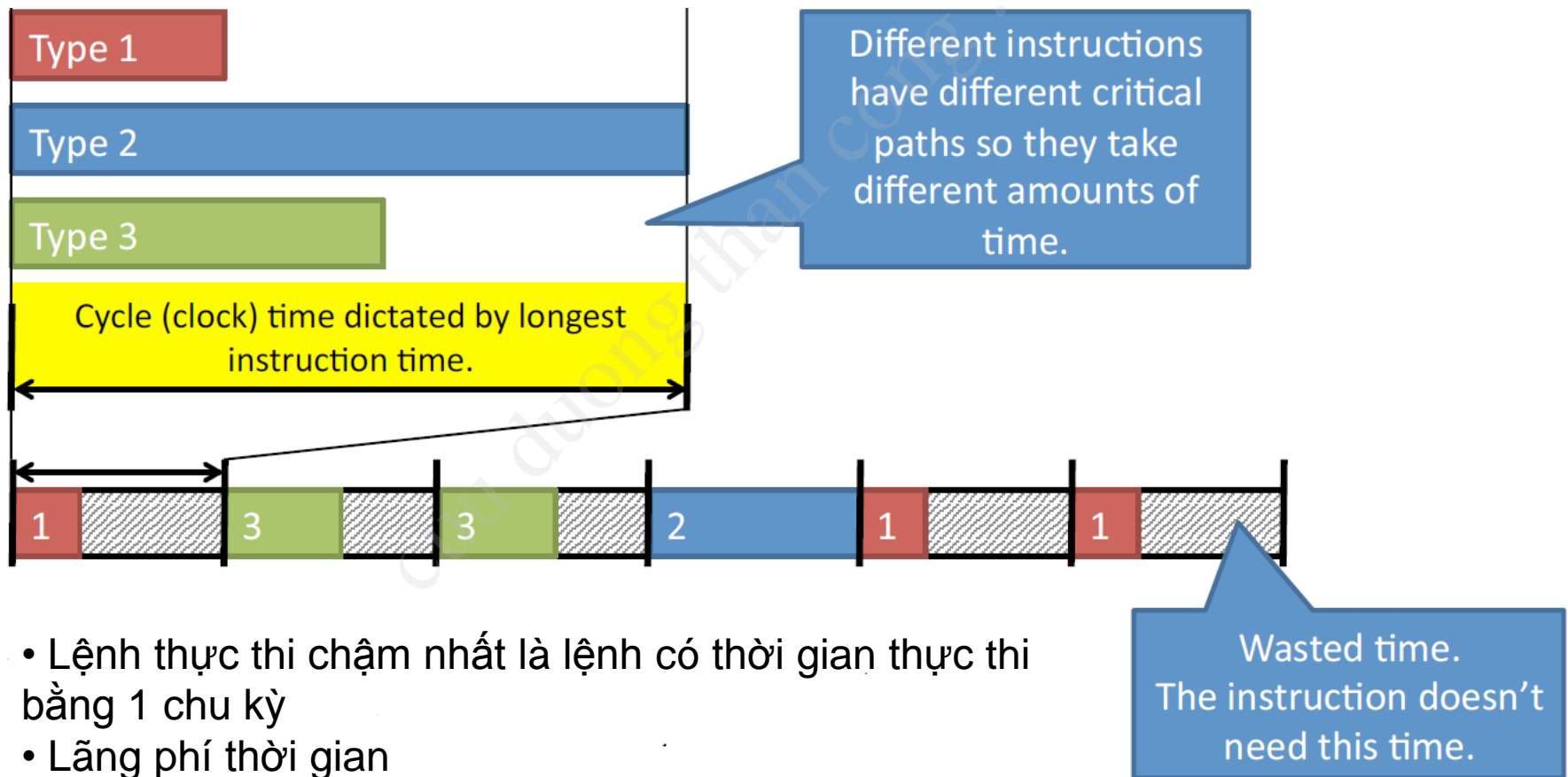
Nội dung

- **So sánh tốc độ xử lý**
 - Single - cycle datapath (from the previous lecture)
 - Multi-cycle
 - Pipelining
- **Kỹ thuật đường ống**
 - Kỹ thuật đường ống là gì?
 - Tại sao lại sử dụng kỹ thuật đường ống?
- **Xây dựng bộ xử lý đường ống**
 - Chia cắt từ bộ xử lý đơn xung nhịp
 - Hoạt động của MIPS pipeline
 - Điều khiển Pipeline

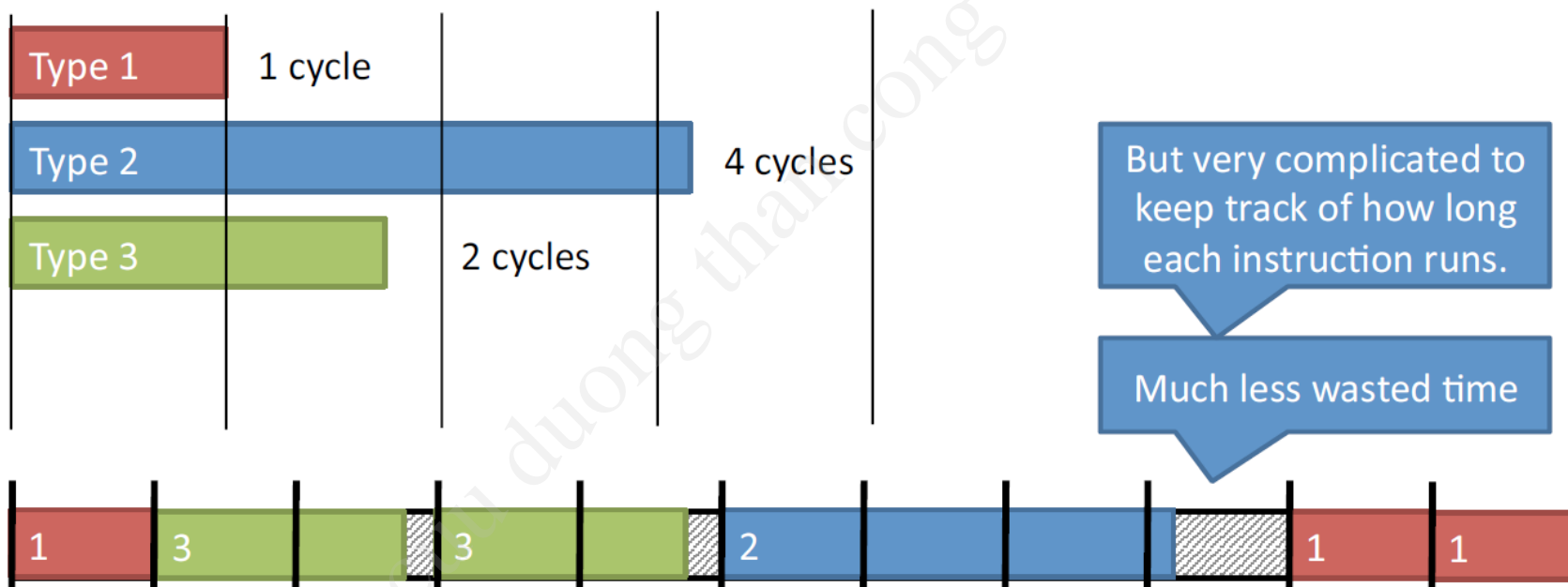
Tốc độ xử lý

(What limits our clock?)

Thời gian thực thi đơn xung nhịp



Giải pháp: Bộ xử lý đa xung nhịp



- Lệnh nhanh nhất xác định tương ứng với 1 chu kỳ
- Lệnh chậm hơn sẽ chiếm nhiều chu kỳ

Cách nào tốt hơn?

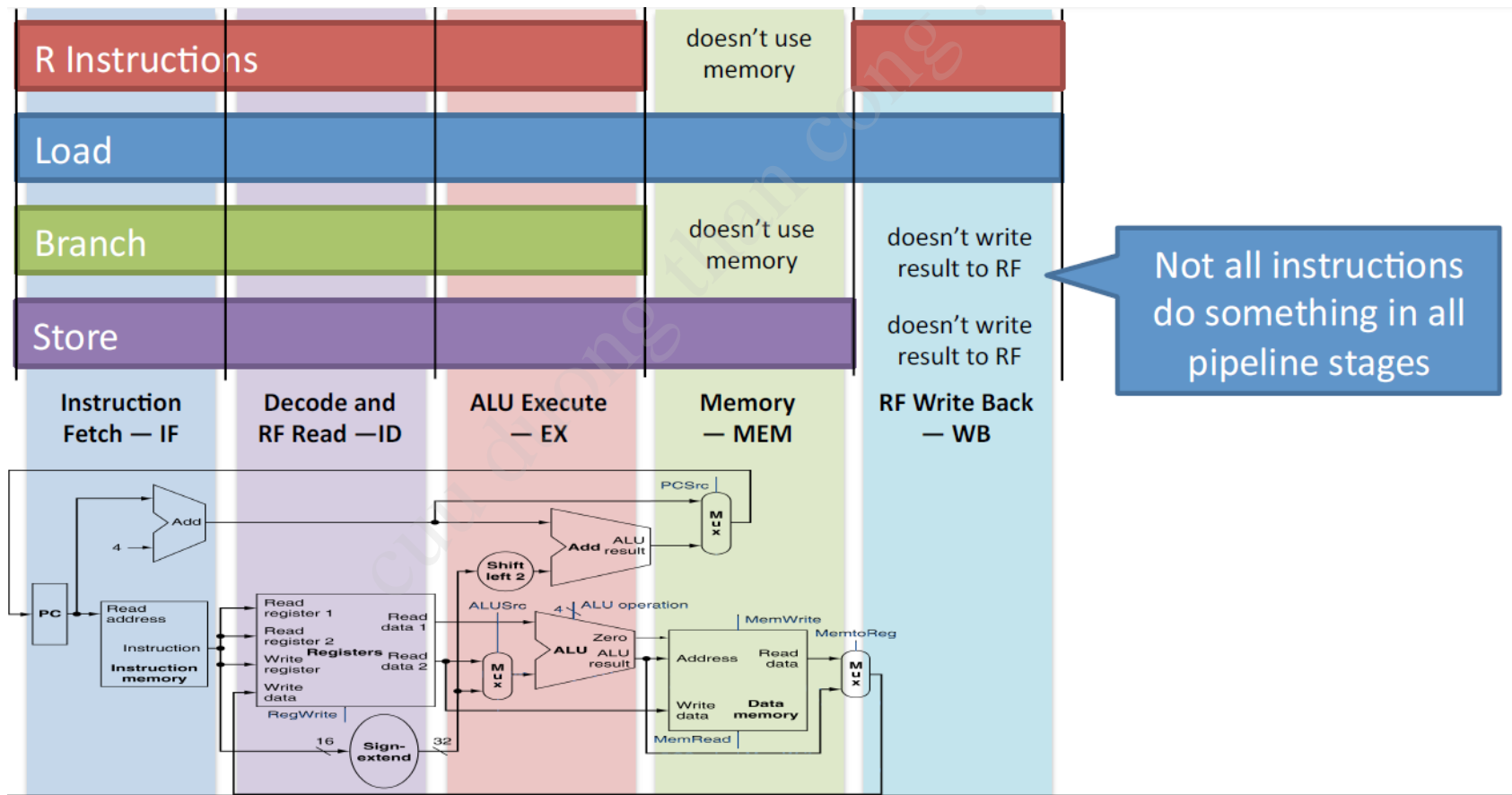
- Chia lệnh thành các giai đoạn khác nhau
- Giai đoạn dài nhất sẽ xác định tốc độ xử lý



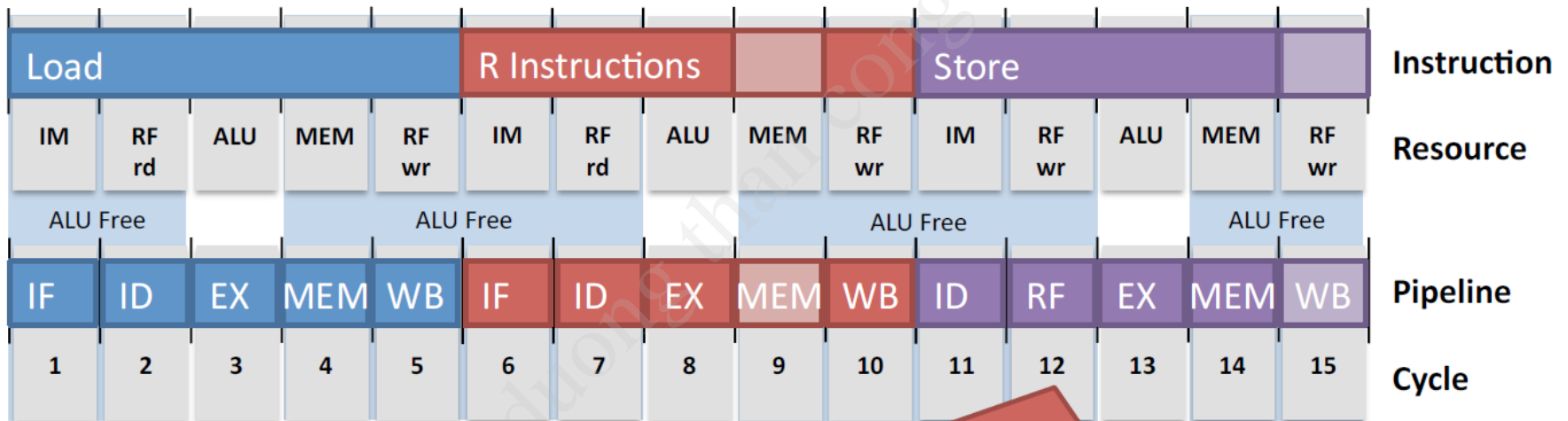
...Cần nhiều chu kỳ cho một lệnh!

Chia thành 5 parts → đồng hồ nhanh hơn 5x lần → nhưng cần nhiều hơn 5x chu kỳ cho một lệnh

Ví dụ MIPS: 5 giai đoạn đường ống



Kỹ thuật này có tốt hơn không?



Can we take advantage of the ALU being unused in cycle 12 to do other useful work?

(and in 1, 2, 4, 5, 6, 7, 8, 10, 11, 14, 15)

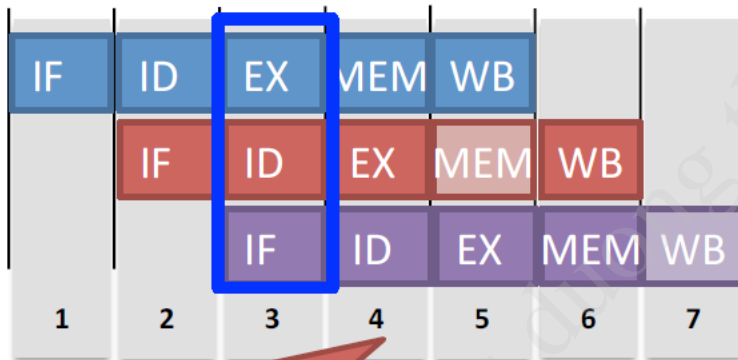
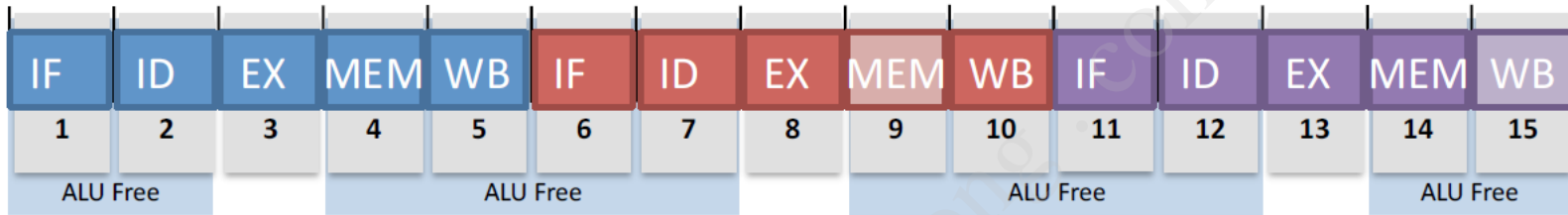
Q: What is the ALU doing in cycle 12?

- ☐ Nothing
- ☐ Accessing the register file
- ☐ Calculating the store address

A: Nothing

In cycle 12 the store instruction is only reading from the register file.

Hoạt động trong đường ống



Load

R-Inst.

Store

Đây là điều chúng ta cần từ đường ống: sử dụng tất cả các phần của bộ xử lý đối với các lệnh khác nhau tại cùng một thời điểm

Q: What is the ALU doing when the load is accessing the memory?

- ☐ Nothing
- ☐ Accessing the register file
- ☐ Processing the R instruction's ALU op

A: Processing the R instruction's ALU op

In cycle 4:

- load is using the memory (MEM)
- R instruction is using the ALU (EX)
- store is using the RF (ID)

Kỹ thuật đường ống là gì?

Một số ví dụ trong đời sống

Kỹ thuật đường ống 1: quy trình giặt là (serial - pipeline)

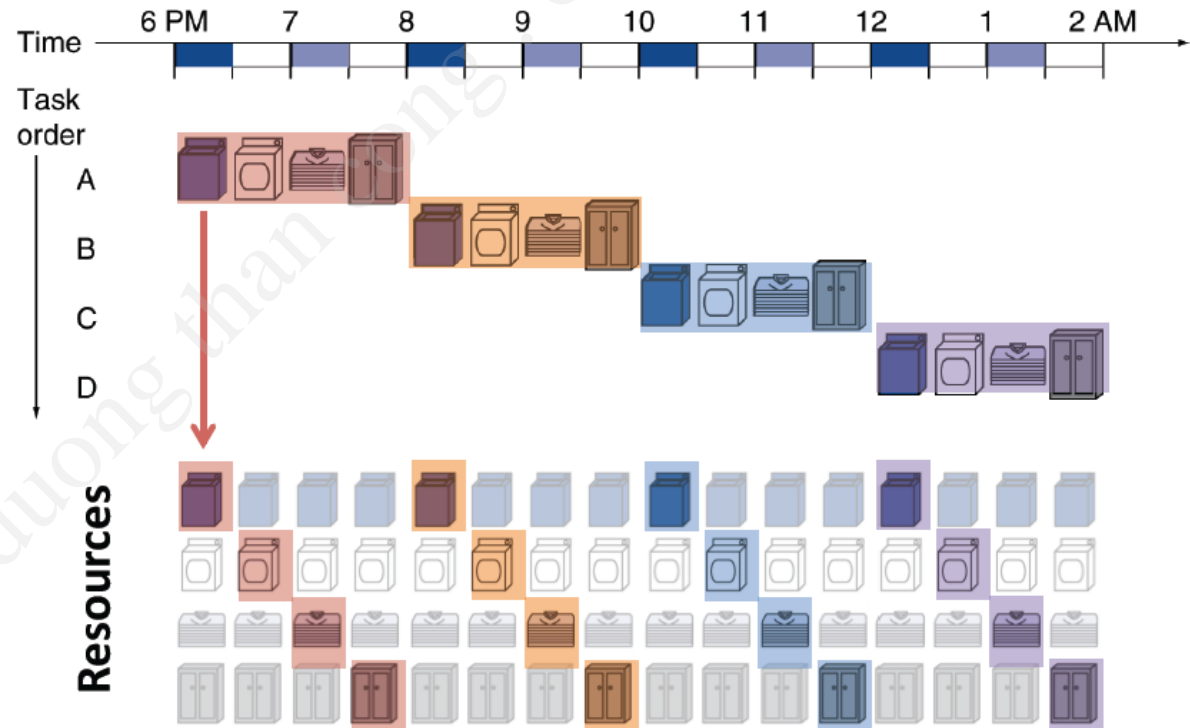
- 4 hoạt động cho một tải:
 - Wash (1h), Dry (1h), Fold (1h), Put away (1h)
- 4 tải mất bao lâu?
 - Wash + Dry + Fold + Put away = 4h
 - **4 loads * 4h/load = 16h**

Q: Bao nhiêu phần trăm tài nguyên đã sử dụng?

- 100%
- 50%
- 25%

A: 25%

Chỉ sử dụng một pha: wash, dry, fold, và put away ở mỗi thời điểm. Còn 3 pha khác là nhàn rỗi.



How can pipelining help?

Ví dụ: Quy trình giặt là (pipelined)

- Hãy thử xếp chồng các hoạt động
- Bao lâu cho 4 lần tải?
 - 4 lần tải trong 7 giờ (mỗi lần tải trong 4h)
 - 7h vs. 16h nhanh hơn 2.3x!

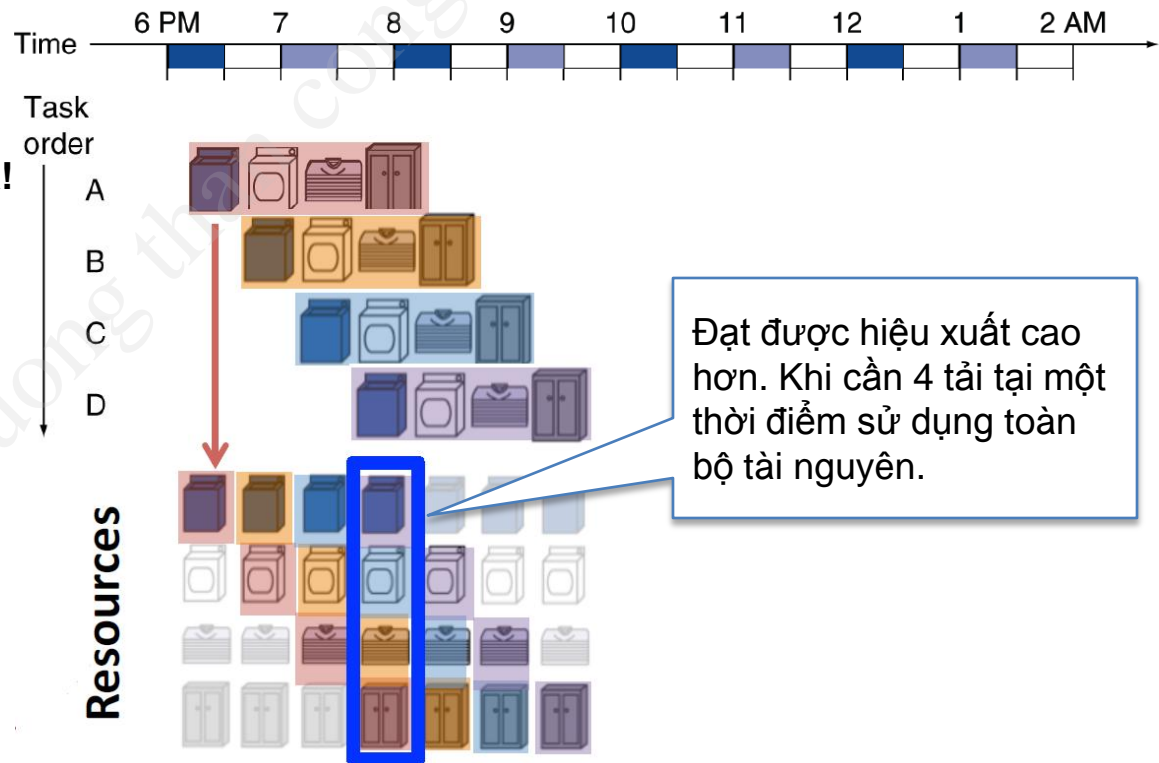
Q: Cần bao nhiêu người để thực hiện 4 hoạt động trong cùng một thời điểm ?

- 1.
- 2.
- 4.

A: 4

Để thực hiện 4 thao tác một lúc cần 4 người.

Tương đương với việc cần điều khiển logic cho 4 lệnh tại một thời điểm.



Đường ống hóa giúp cho việc sử dụng tất cả các tài nguyên tại cùng một thời điểm khi thực hiện nhiều hoạt động khác nhau.

Ví dụ về kỹ thuật đường ống 2: lắp ráp xe (serial)

- Công nghệ của Henry Ford
- Sản xuất theo đường ống



Non-pipelined: 1 car/4 hours

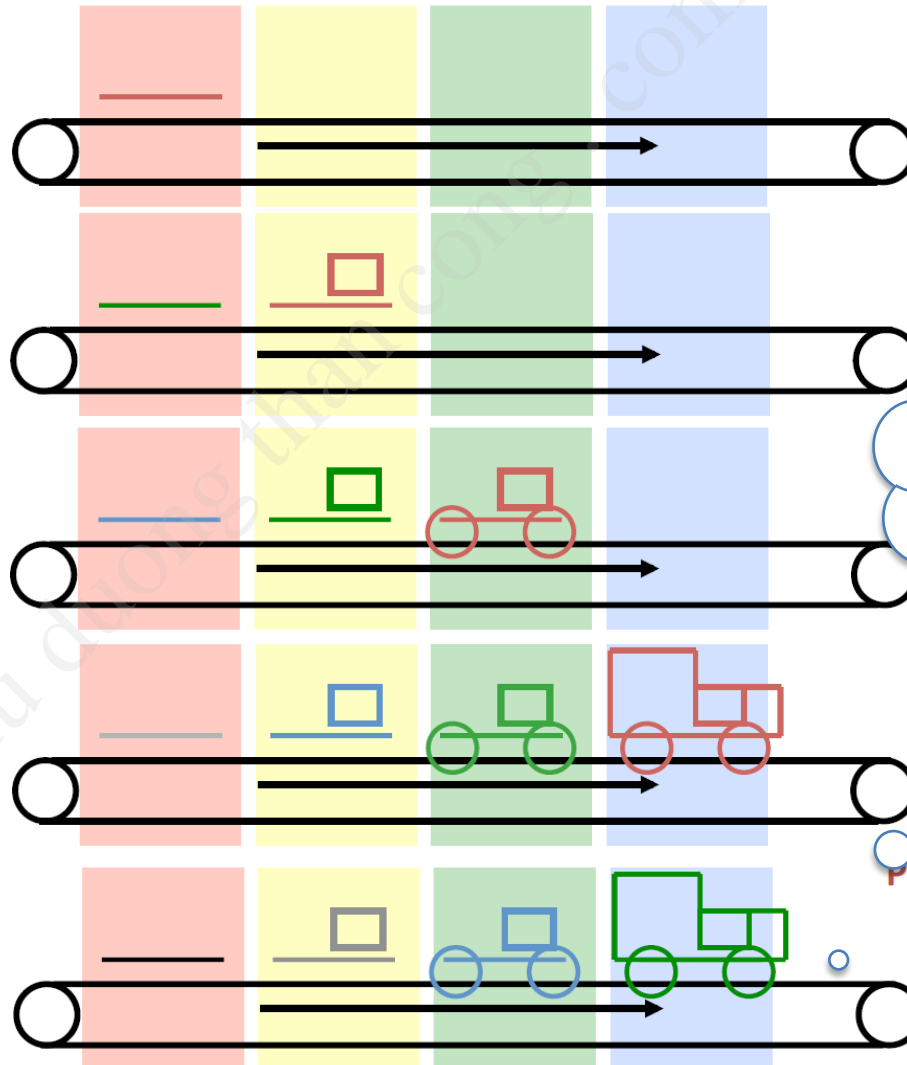
Pipelining example 2: car assembly (serial)

Q: Hiệu suất hoạt động sẽ như thế nào nếu đường ống không đầy?

1. Goes up
2. Stays the same
3. Goes down

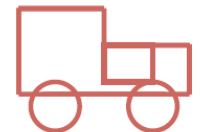
A: Goes down

Nếu đường ống không đầy, sẽ không sử dụng hết tài nguyên làm hiệu năng giảm xuống.



Đường ống đầy.
Hiệu năng tối ưu
bởi vì tất cả tài
nguyên đều được
sử dụng trong
cùng một khoảng
thời gian.

Pipelined: 1 car/hour



Tại sao lại sử dụng Pipeline ? (Hint: performance)

Tại sao lựa chọn pipeline?

- Nếu có thể giữ cho đường ống luôn đầy sẽ có **throughput (số công việc thực hiện được trong một khoảng thời gian)** tốt hơn.
 - Laundry: 1 load of laundry/hour
 - Car: 1 car/hour
 - MIPS: 1 instruction/cycle
- Xuất hiện trễ (total time per)
 - Laundry: 4 giờ cho mỗi lần giặt là
 - Car: 4 giờ cho một xe ô tô
 - MIPS: 5 chu kỳ cho mỗi lệnh
- Pipelining nhanh hơn bởi vì sử dụng tất cả tài nguyên tại cùng một thời điểm
 - Laundry: máy giặt, máy sấy, gấp, cất vào tủ
 - Car: lắp đế, lắp giáp động cơ, lắp lốp, lắp buồng lái
 - MIPS: Nạp lệnh, đọc thanh ghi, ALU, Truy cập bộ nhớ và ghi vào thanh ghi. (Instruction fetch, register read, ALU, memory, and register write).

Hiệu năng đường ống hóa trong bộ xử lý

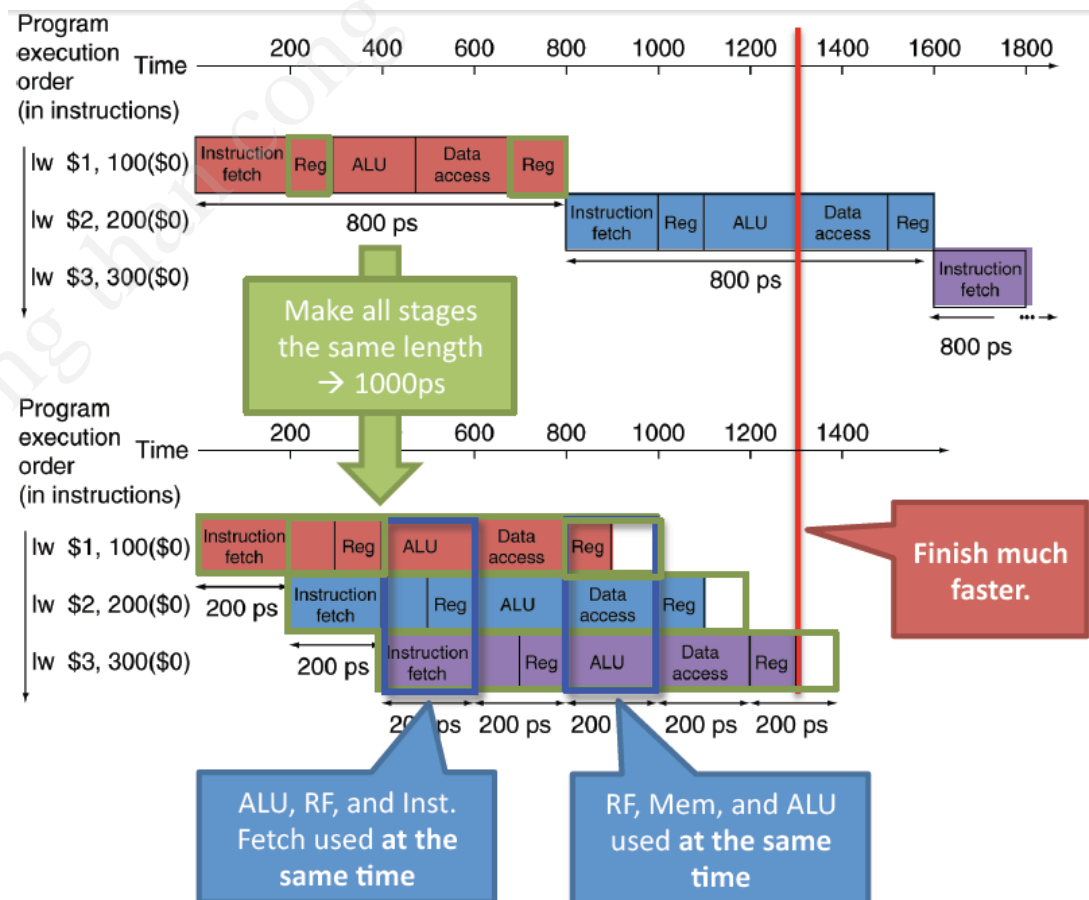
- Chương trình tải 3 lệnh mỗi lệnh cần 800ps (0.8ns)
- Nếu đường ống hóa và xếp chồng sẽ sử dụng được tất cả tài nguyên một cách song song và thực hiện 3 lệnh trên nhanh hơn.

Q: Thông lượng tăng lên bao nhiêu lần trong đường ống 5 giai đoạn?

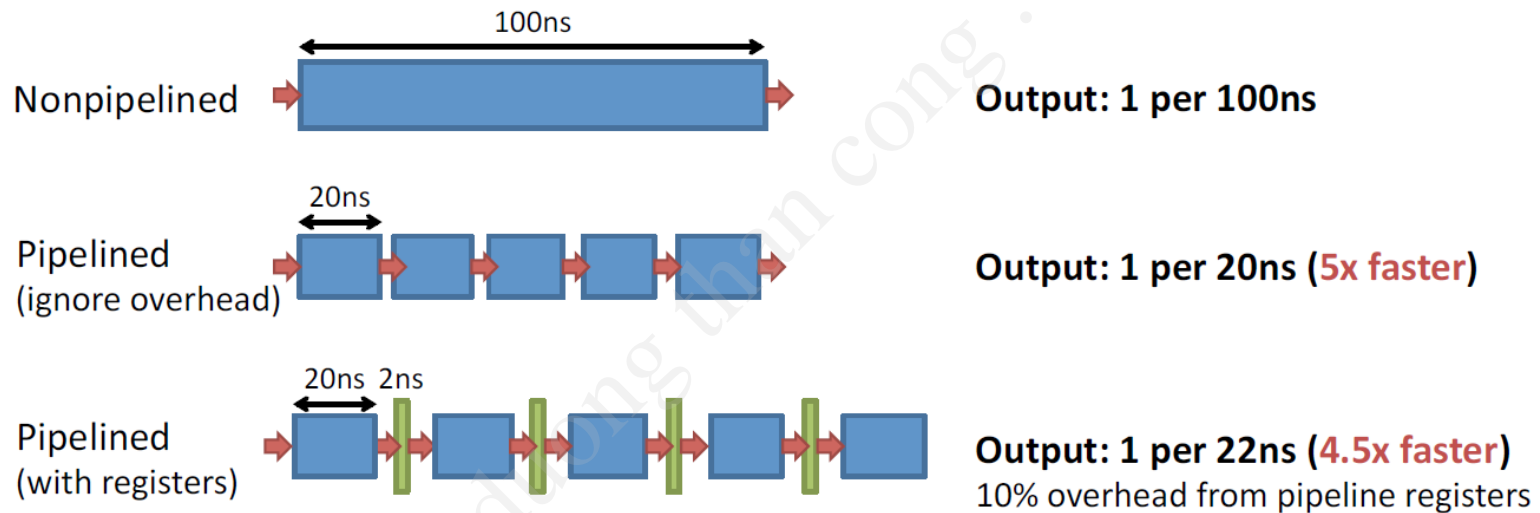
1. 1.7 lần
2. 4 lần
3. 5 lần

A: 1.7 lần

Đối với đường ống, **throughput** là một lệnh trong mỗi 200ps và 800ps không có không đường ống hóa. Tuy nhiên phải tăng độ trễ lên tới 1000ps trên một lệnh để cân bằng 5 pha đường ống. Tốc độ tuyệt đối cho 3 lệnh riêng biệt là 1.7x (1400ps/2400ps).



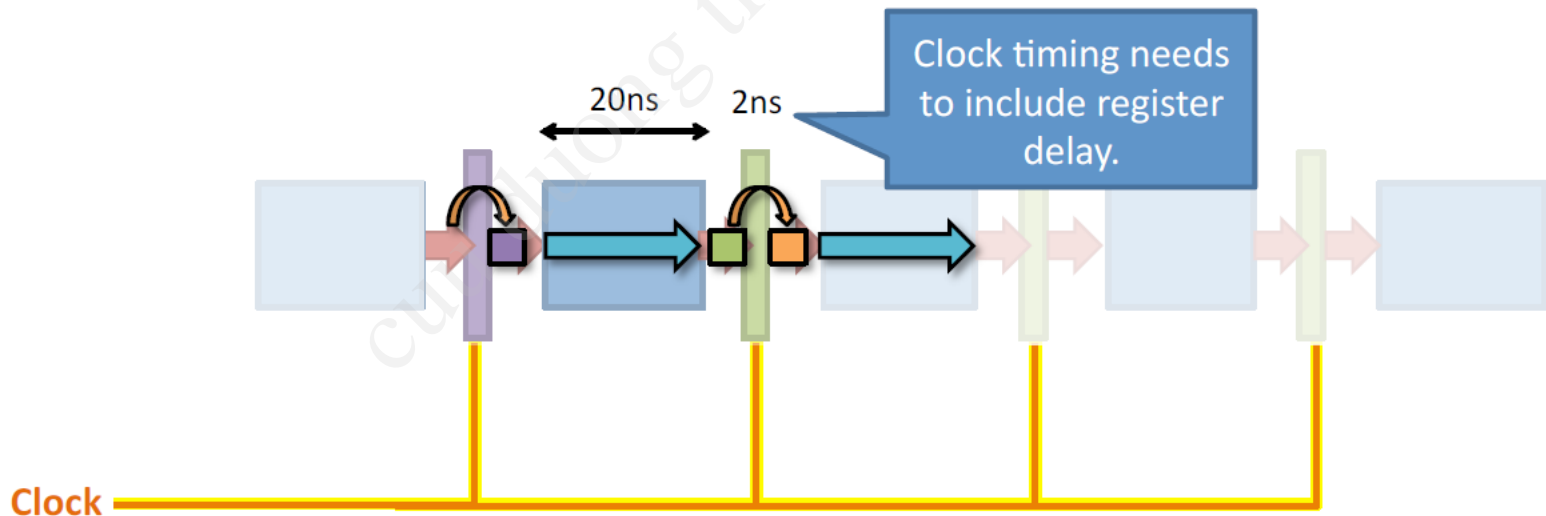
Các thanh ghi Pipeline và mào đầu (phí tổn điều khiển)



- Mỗi trạng thái đường ống là một tổ hợp logic (ALU, sign extension)
- Cần lưu trữ trạng thái các pha (which instruction)
- Cần các thanh ghi **pipeline** giữa các pha để lưu trữ lệnh cho các pha.

Đồng hồ trong bộ xử lý Pipeline

- Tốc độ đồng hồ xác định bởi **register** → **stage** → **register**
 - **Clock dịch chuyển dữ liệu đi đến thanh ghi đầu tiên**
 - Dữ liệu tính toán trong các trạng thái (combinational: think an adder)
 - Dữ liệu cần đến thanh ghi tiếp theo đúng giờ tương ứng với xung đồng hồ tiếp theo



Hiệu năng của việc đường ống hóa MIPS

Instr	IF Instruction Fetch	ID Decode & RF Read	EX Execute	MEM Access Memory	WB Write back to RF	Total time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
R-format	200ps	100ps	200ps		100ps	600ps
beq	200ps	100ps	200ps			500ps

- Thiết kế đơn xung nhịp (**Single-cycle**):
 - Đồng hồ đặt cho lệnh chậm nhất: **800ps clock time**
- Thiết kế đường ống hóa **Pipelined**:
 - Đồng hồ được đặt cho pha chậm nhất: **200ps**
- Chú ý rằng một vài lệnh không sử dụng hết các pha.
 - Cần điều khiển để chắc chắn rằng các pha hoạt động đồng bộ

Xây dựng bộ xử lý đường ống

Cắt ra từ bộ xử lý đơn xung nhịp

Làm thế nào để chia các lệnh MIPS?

(You've already seen it...)

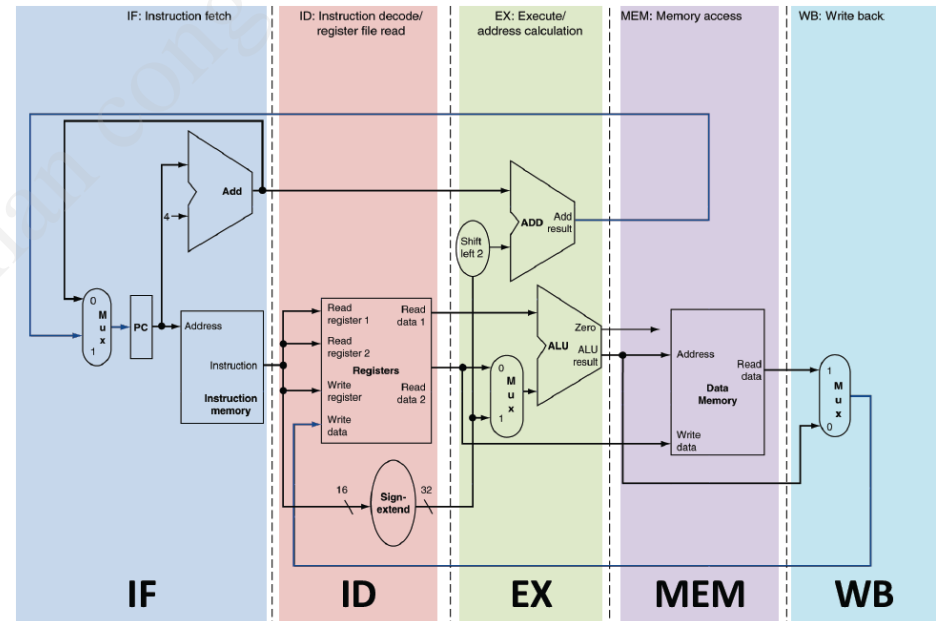
1. IF: Instruction fetch from memory
2. ID: Instruction decode and register read
3. EX: Execute operation or calculate address
4. MEM: Access memory
5. WB: Write result back to register

Q: Thiếu cái gì trong hình vẽ?

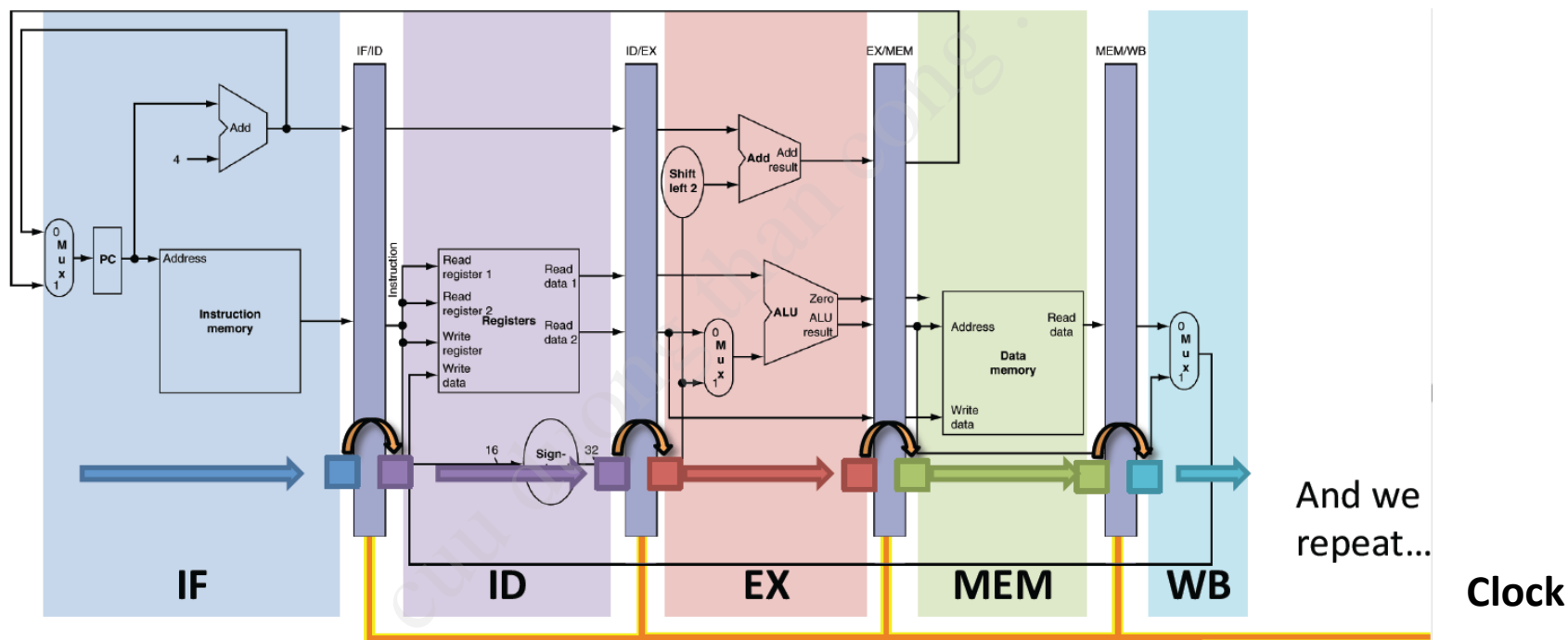
1. Balanced stages
2. Pipeline registers
3. Write back for the RF

A: Pipeline registers

Cần chúng để lưu trạng thái (lệnh và kết quả) giữa các pha.

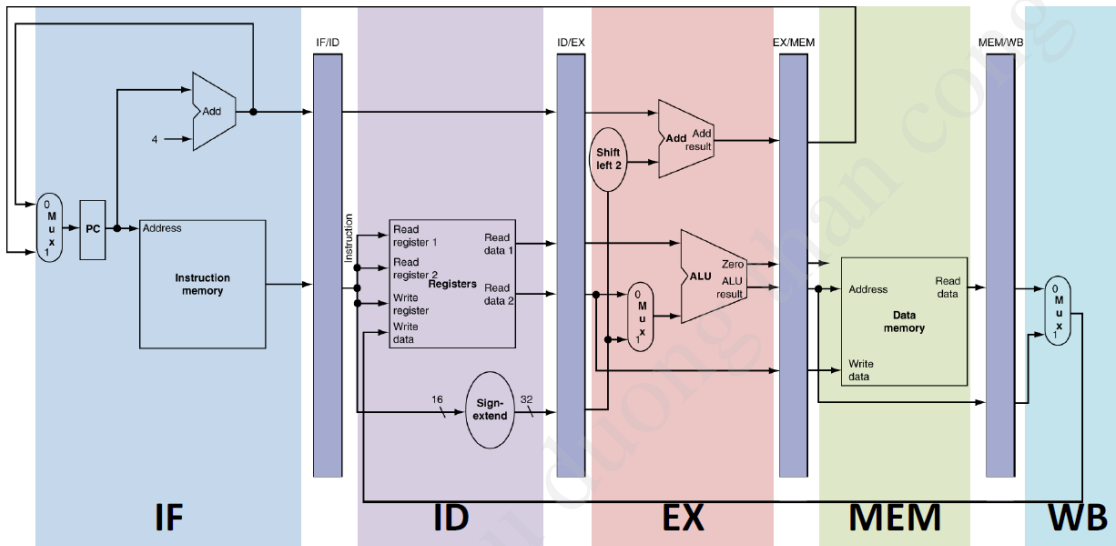


Các thanh ghi pipeline.



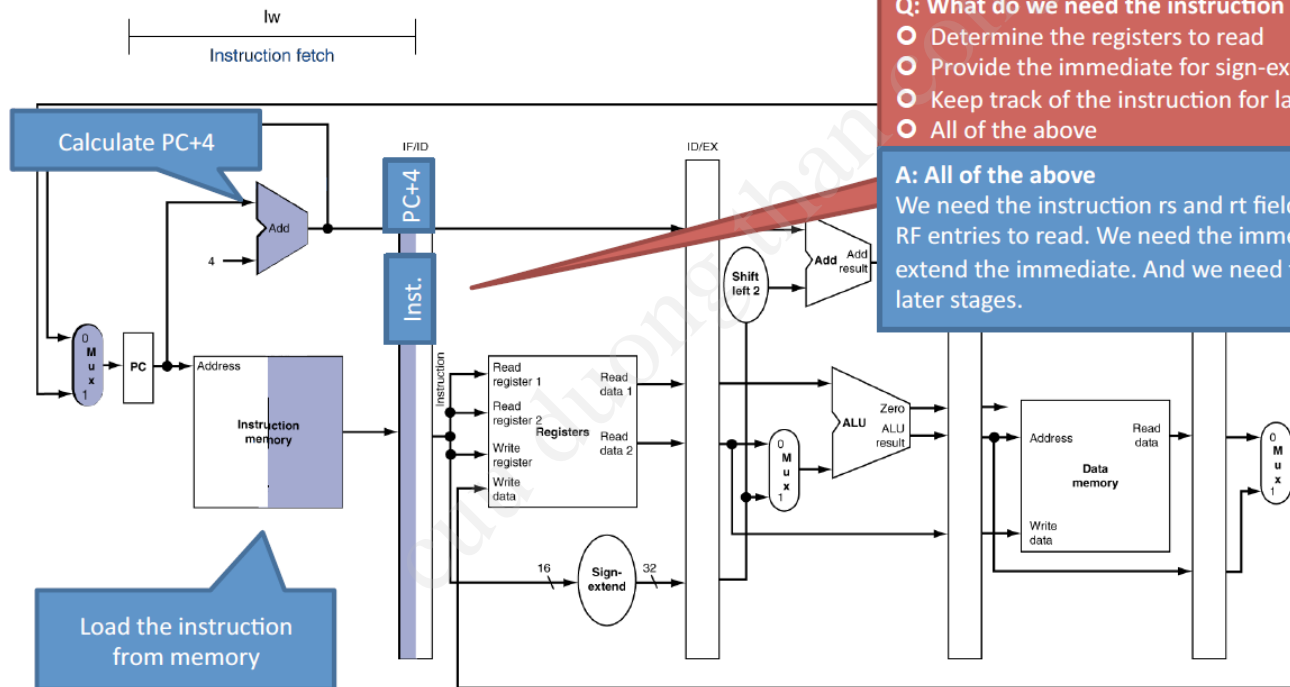
- Các thanh ghi lưu giữ thông tin thủ tục giữa các pha.
- Dịch chuyển dữ liệu đến các pha tiếp kế tiếp theo xung đồng hồ

Chiều chuyển động của đường ống trong MIPS.



- Làm thế nào để tải lệnh đi trong pipeline
- Chú ý:
 - Cái gì kết nối trong mỗi giai đoạn?(combinational)
 - Cái gì được lưu trữ trong thanh ghi? (state)

IF for load



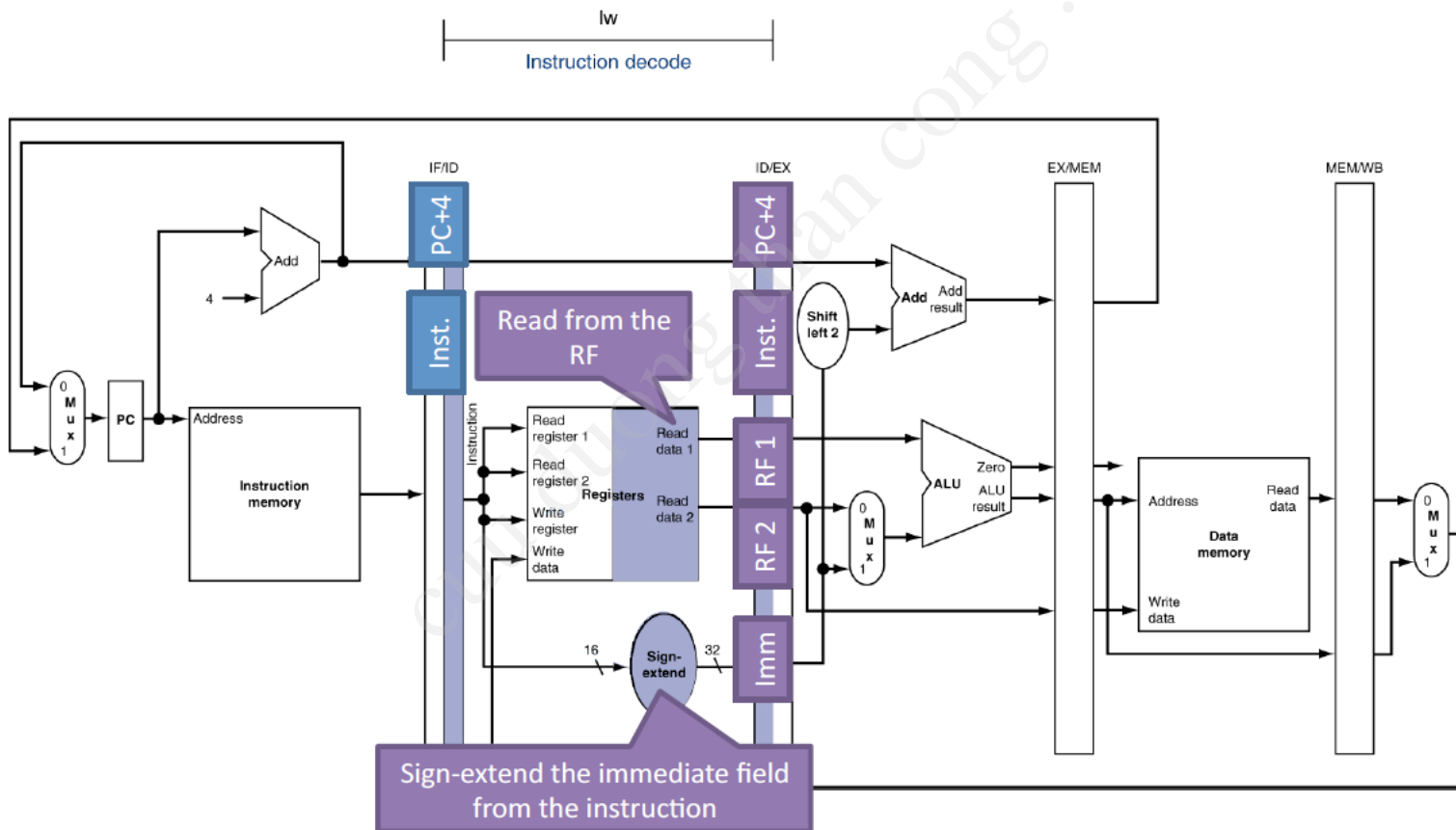
Q: What do we need the instruction for in the next stage?

- Determine the registers to read
- Provide the immediate for sign-extending
- Keep track of the instruction for later stages
- All of the above

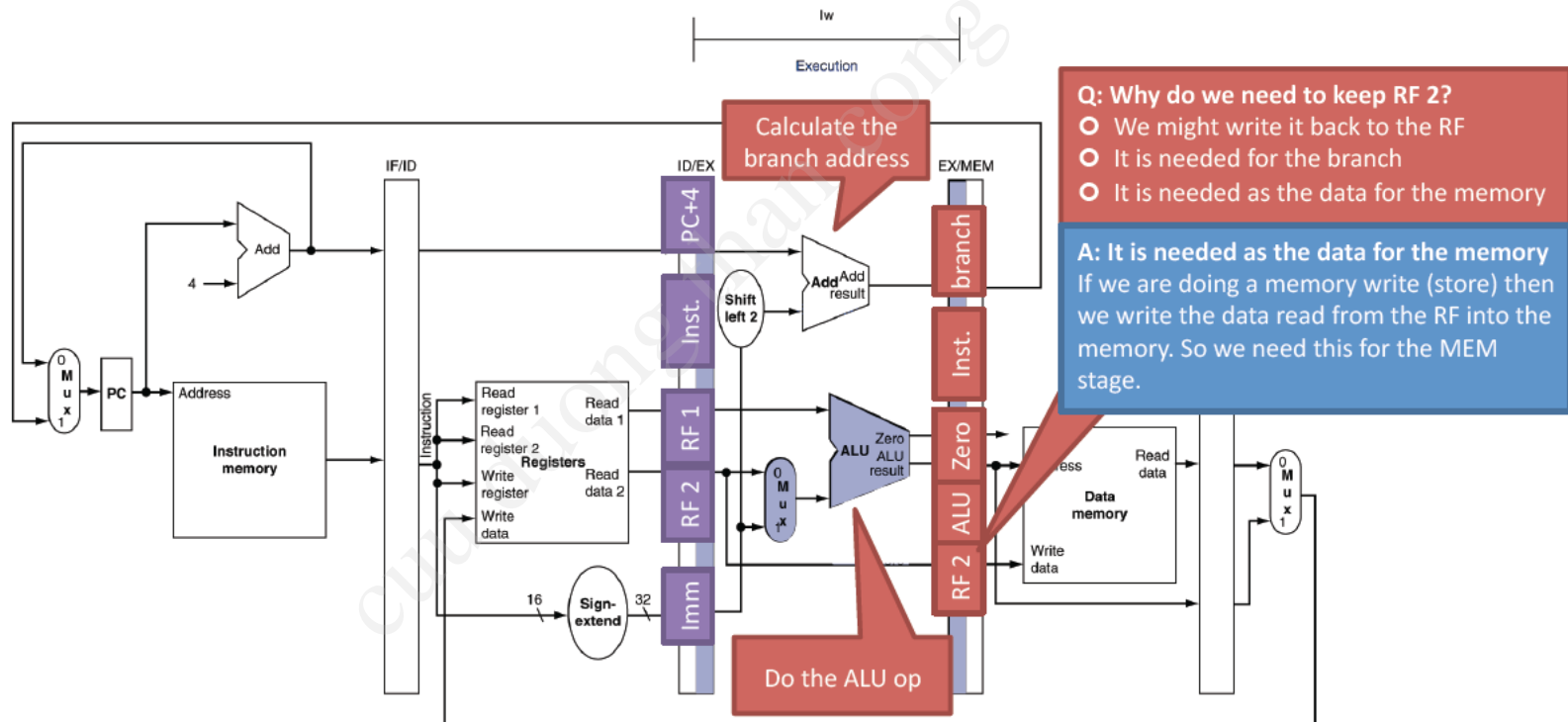
A: All of the above

We need the instruction rs and rt fields to figure out which RF entries to read. We need the immediate field to sign-extend the immediate. And we need the instruction for later stages.

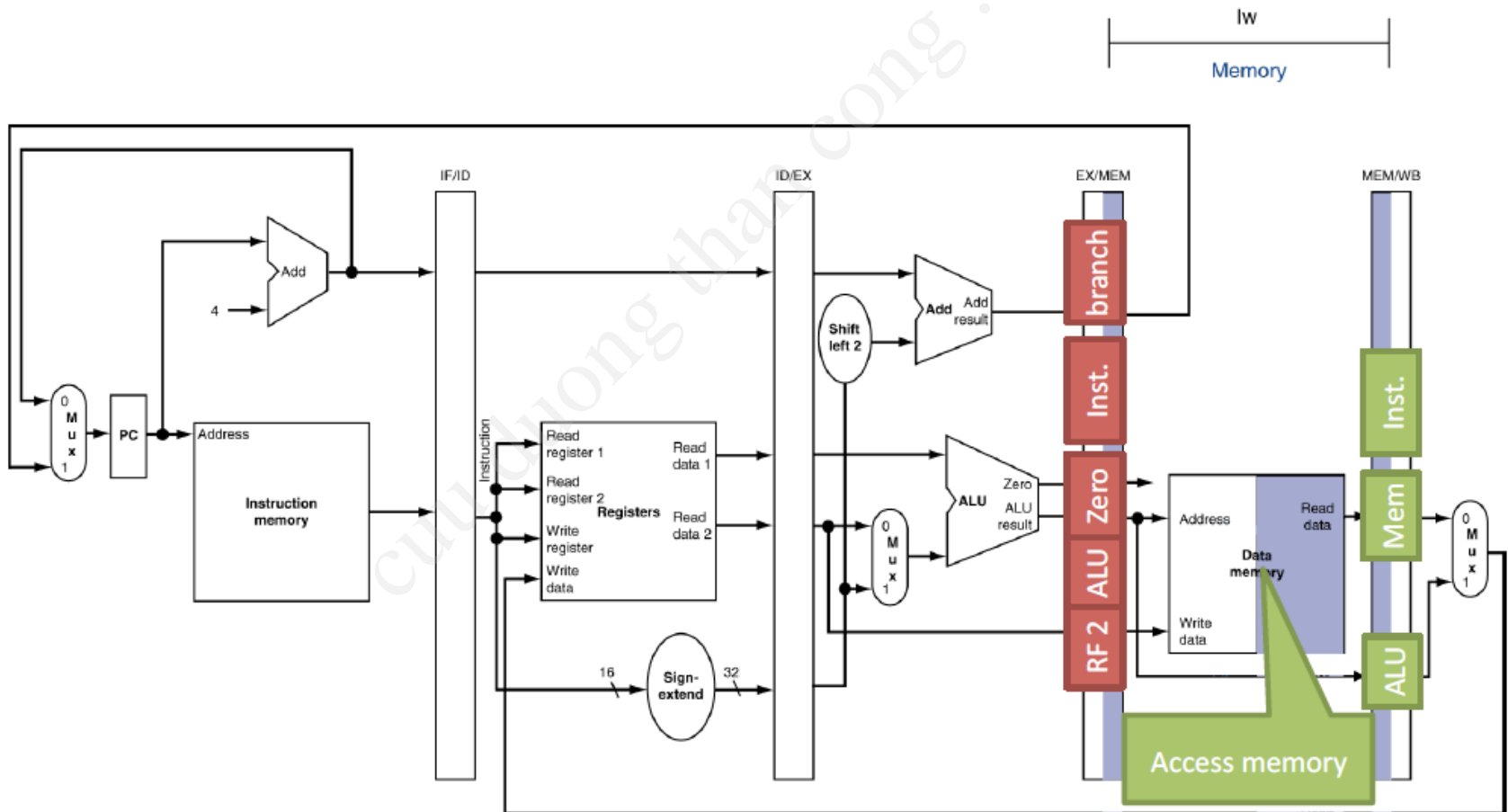
ID for load



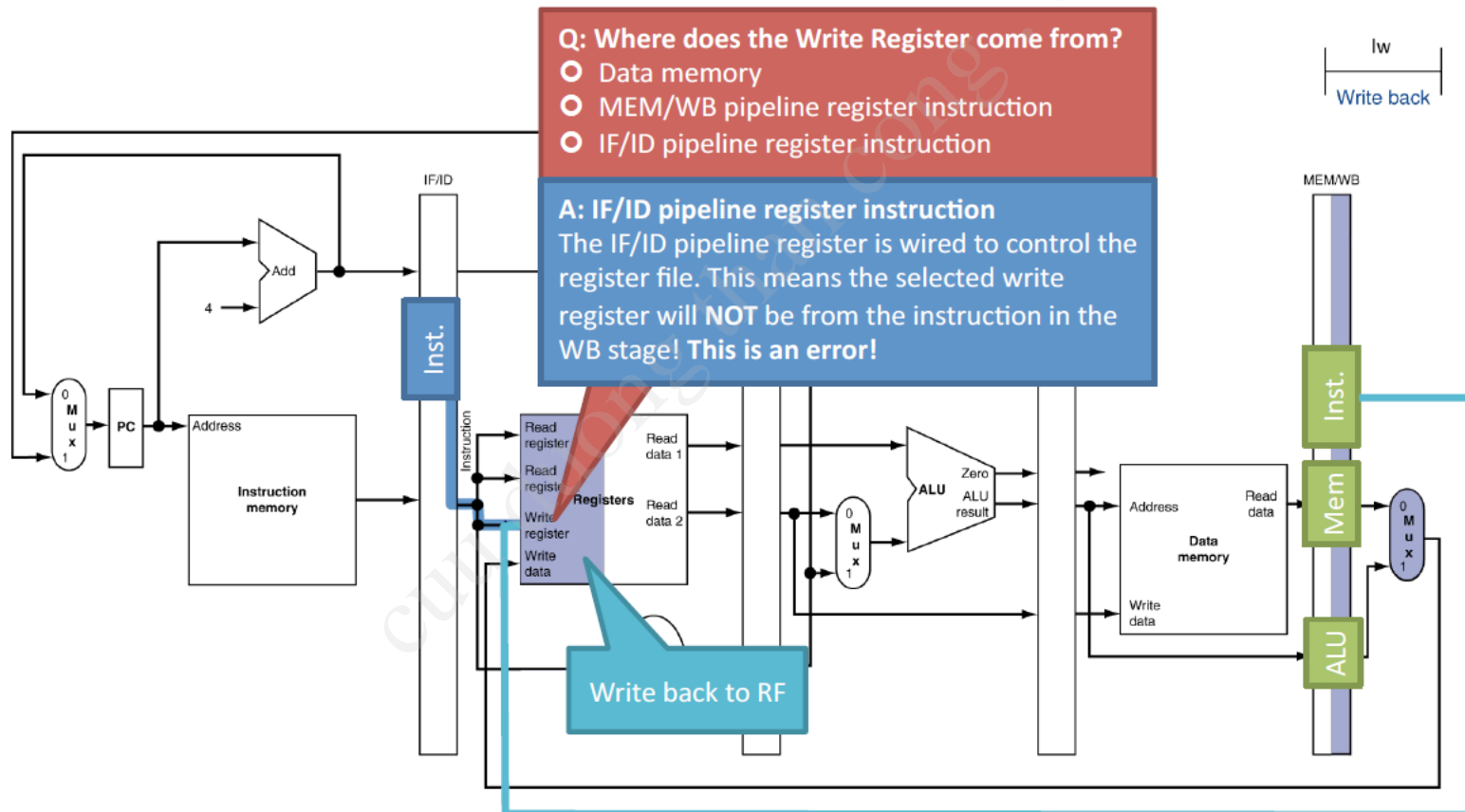
EX for load



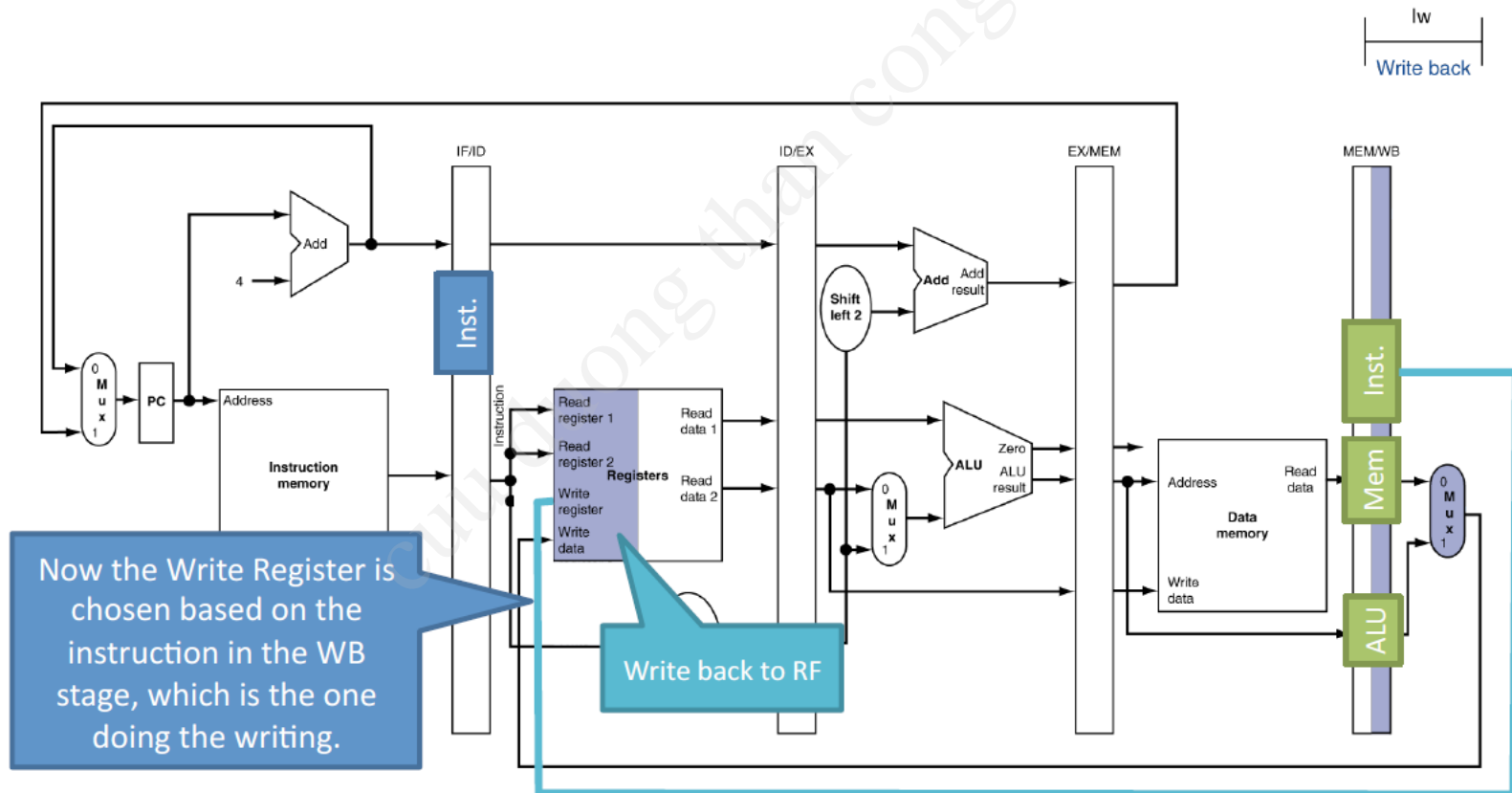
MEM for load



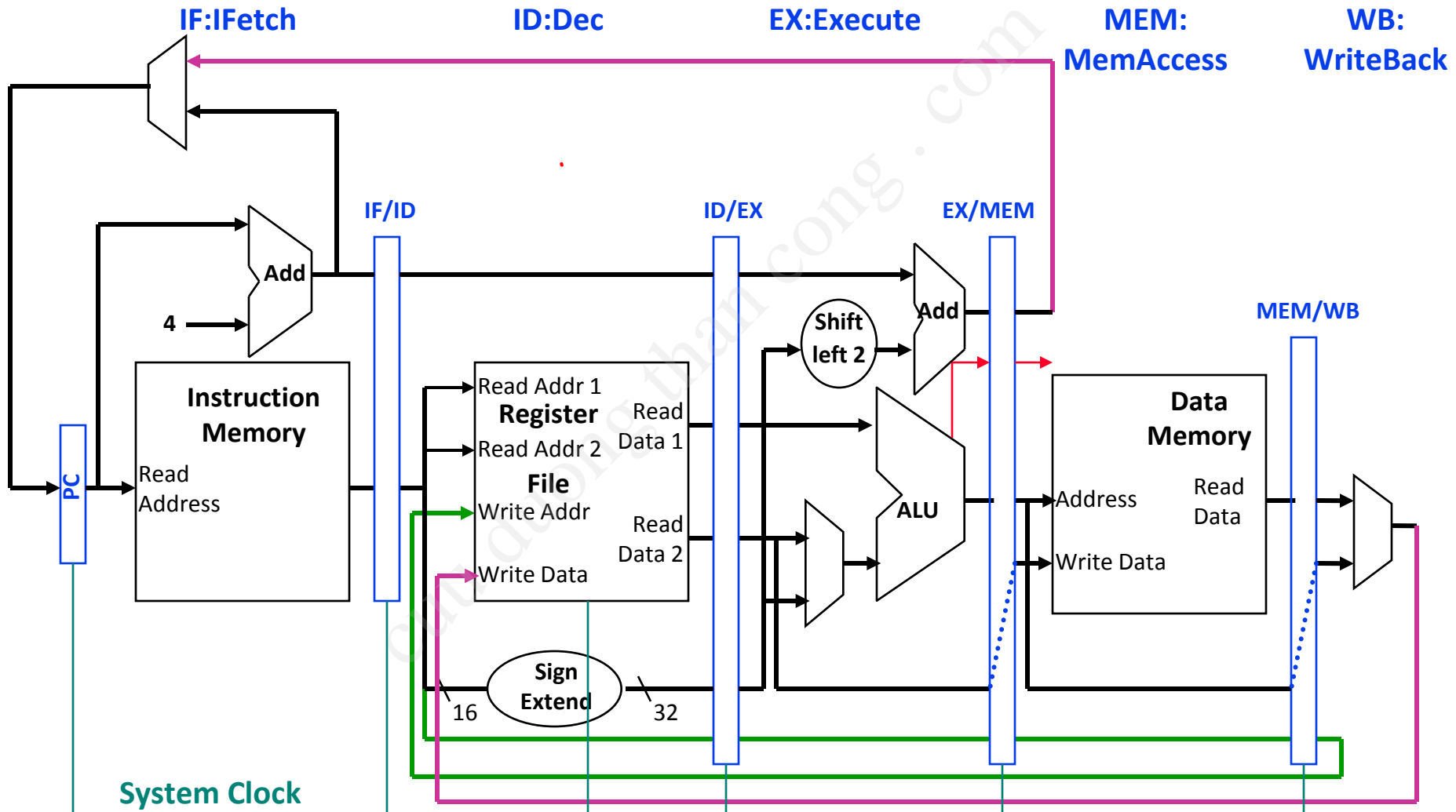
WB for load



Fixing the WB stage



Đường dữ liệu MIPS pipeline



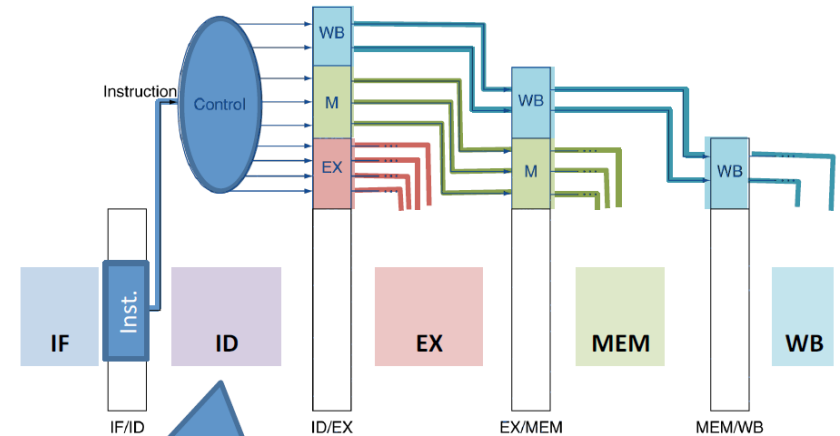
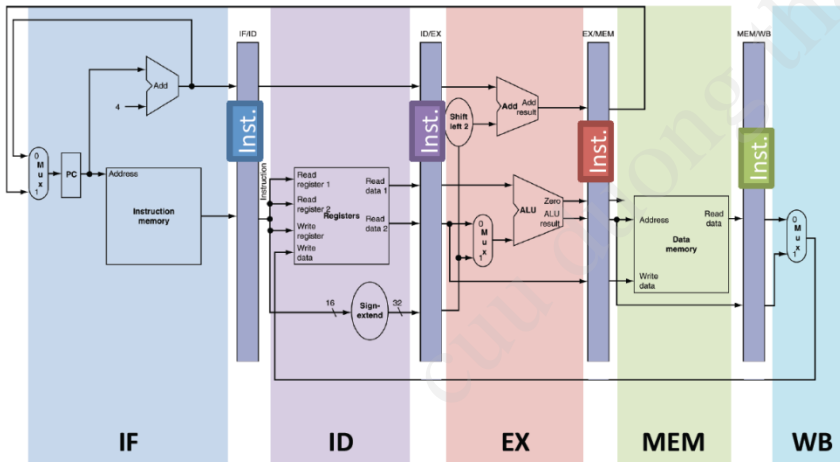
□ Thanh ghi trạng thái giữa các giai đoạn thực hiện lệnh để phân cách

Điều khiển logic trong Pipeline

(Làm thế nào để giải mã các lệnh trong đường ống?)

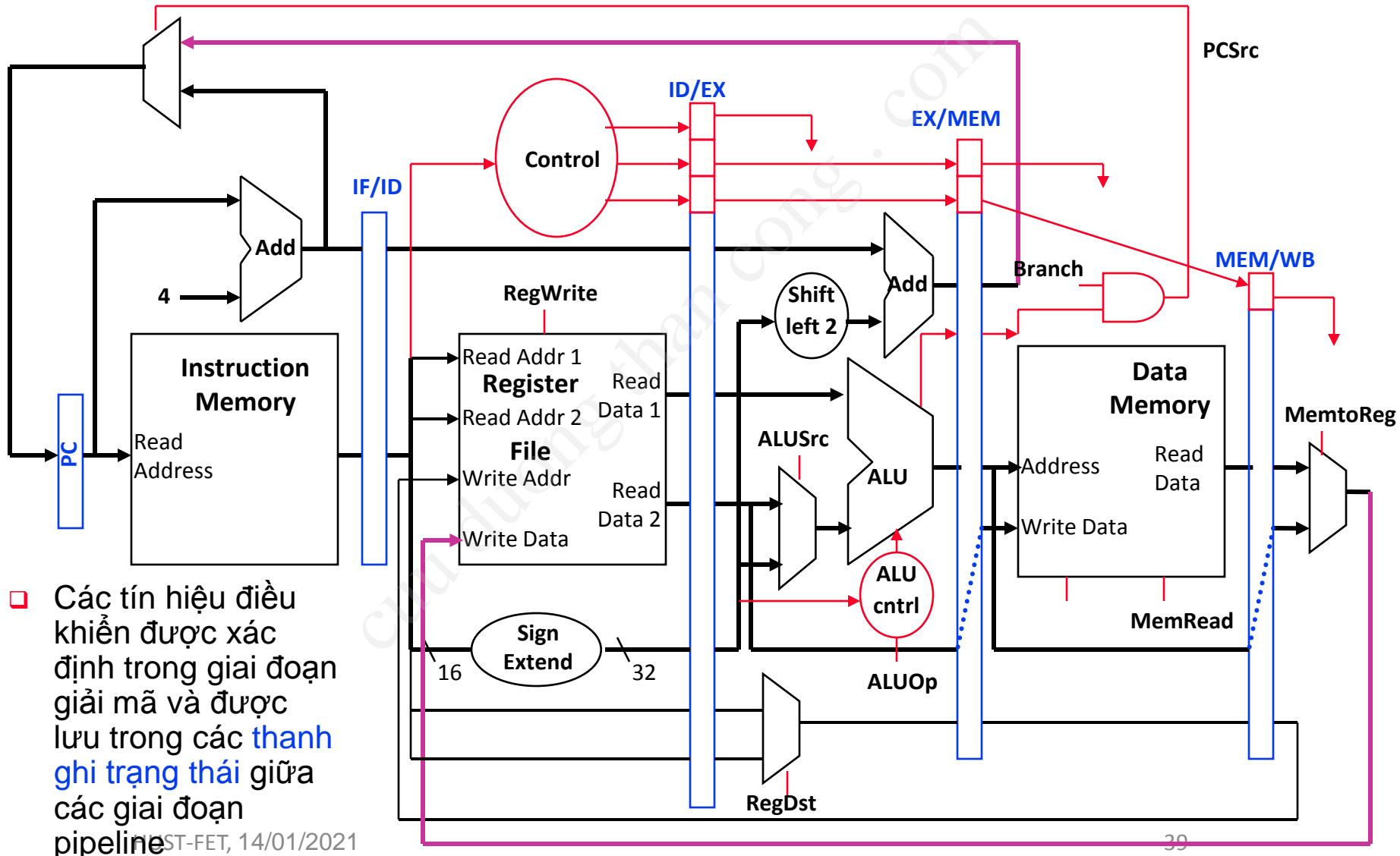
Điều khiển Pipeline

- Có cần toàn bộ thanh ghi lệnh trong các giai đoạn ống ?
- Không, chỉ cần một vài bit cho mỗi pha.



This is why it is called **Decode**: we decode the instruction into control signals for the pipeline.

Điều khiển MIPS pipeline

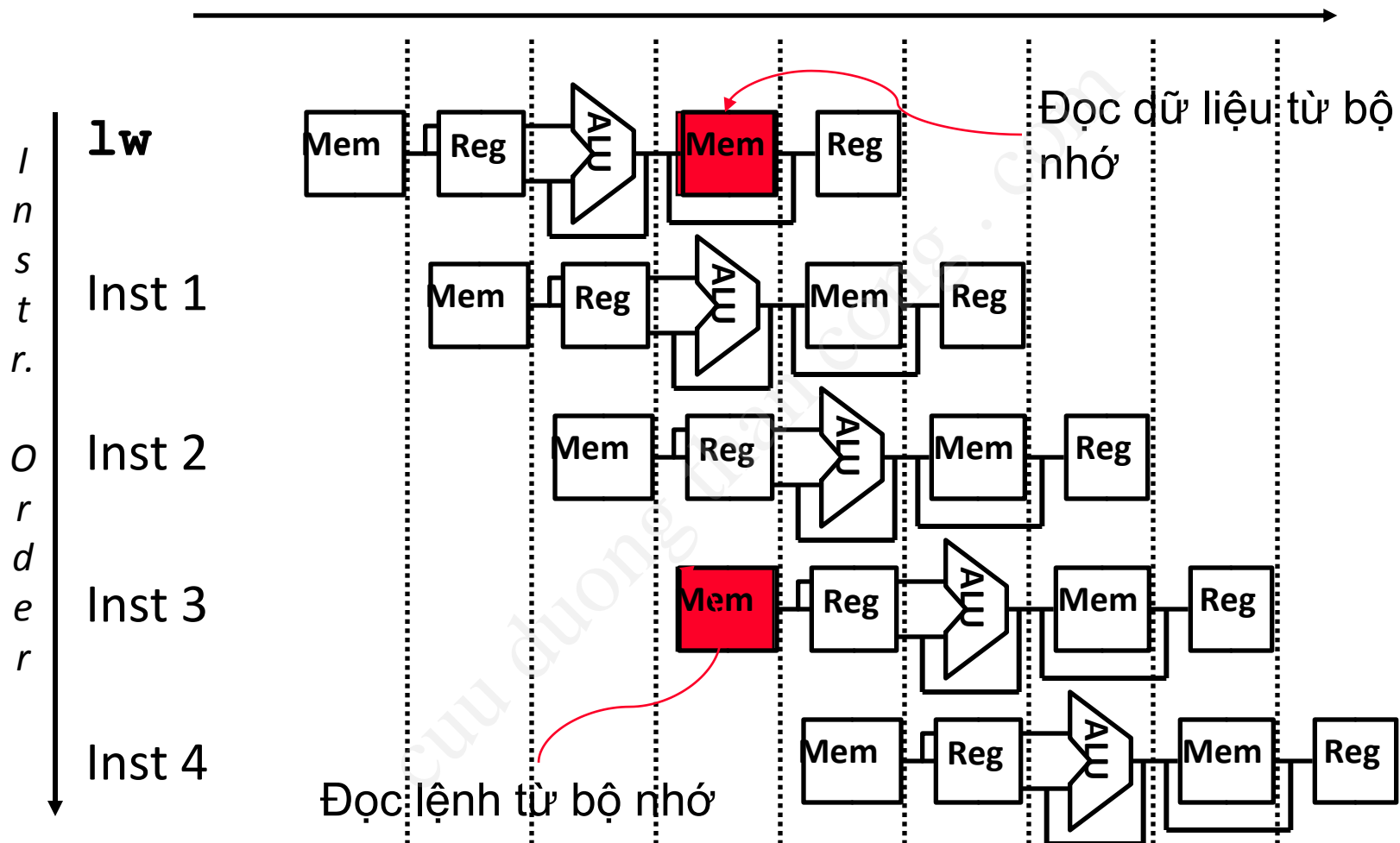


Xung đột Pipeline

- ❑ **Xung đột cấu trúc:** yêu cầu sử dụng cùng một tài nguyên cho 2 lệnh khác nhau tại cùng 1 thời điểm
- ❑ **Xung đột dữ liệu:** yêu cầu sử dụng dữ liệu trước khi nó sẵn sàng
 - ❑ Các toán hạng nguồn của 1 lệnh được tạo ra bởi lệnh phía trước vẫn đang nằm trong pipeline
- ❑ **Xung đột điều khiển:** yêu cầu quyết định điều khiển dòng chương trình trước khi điều kiện rẽ nhánh và giá trị PC mới được tính toán
 - ❑ Các lệnh rẽ nhánh, nhảy và ngắt
- ❑ **Giải quyết xung đột bằng cách chờ đợi**
 - ❑ Khối điều khiển pipeline cần phát hiện xung đột
 - ❑ Và hành động để giải quyết xung đột

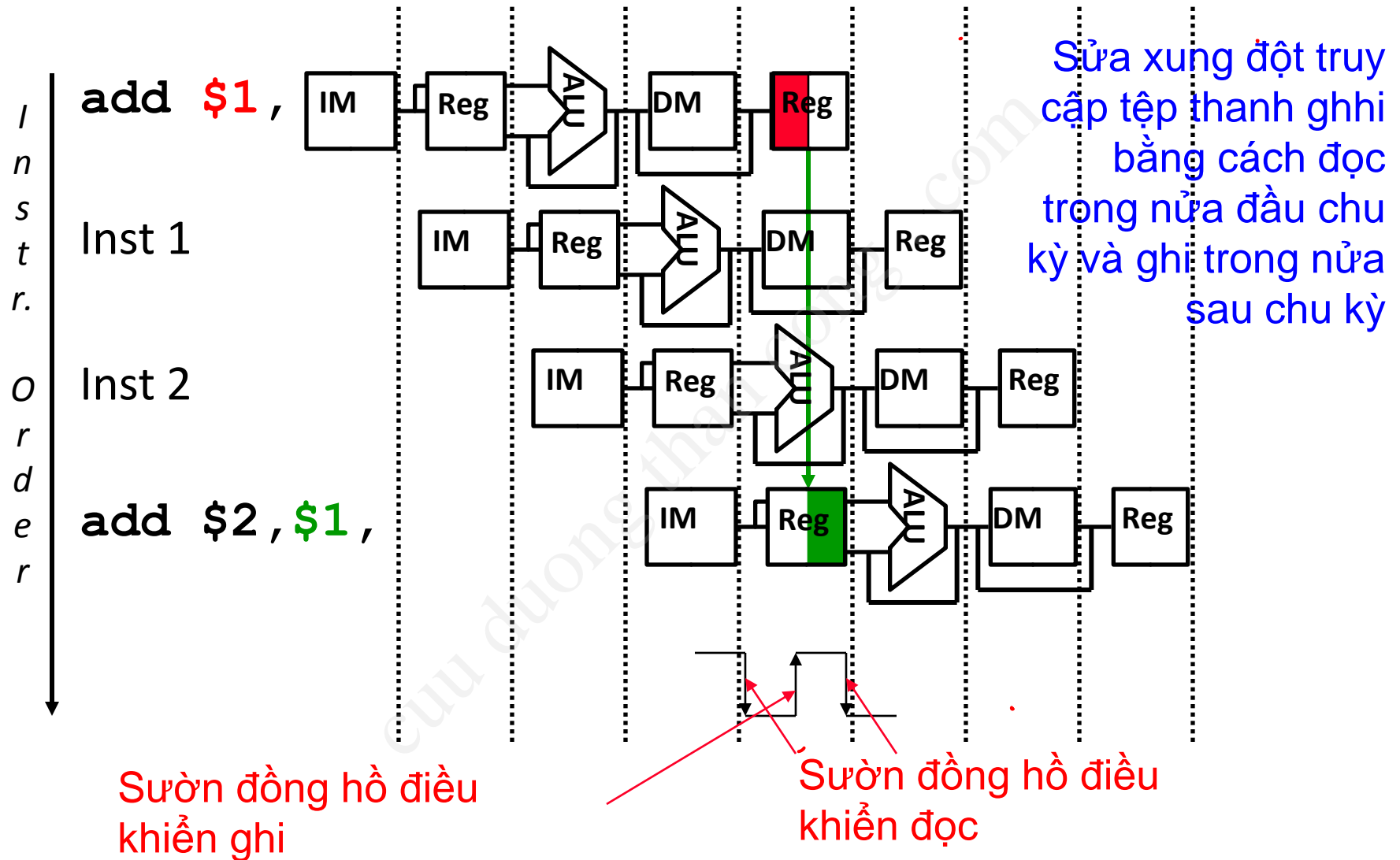
Bộ nhớ đơn: Xung đột cấu trúc

Time (clock cycles)

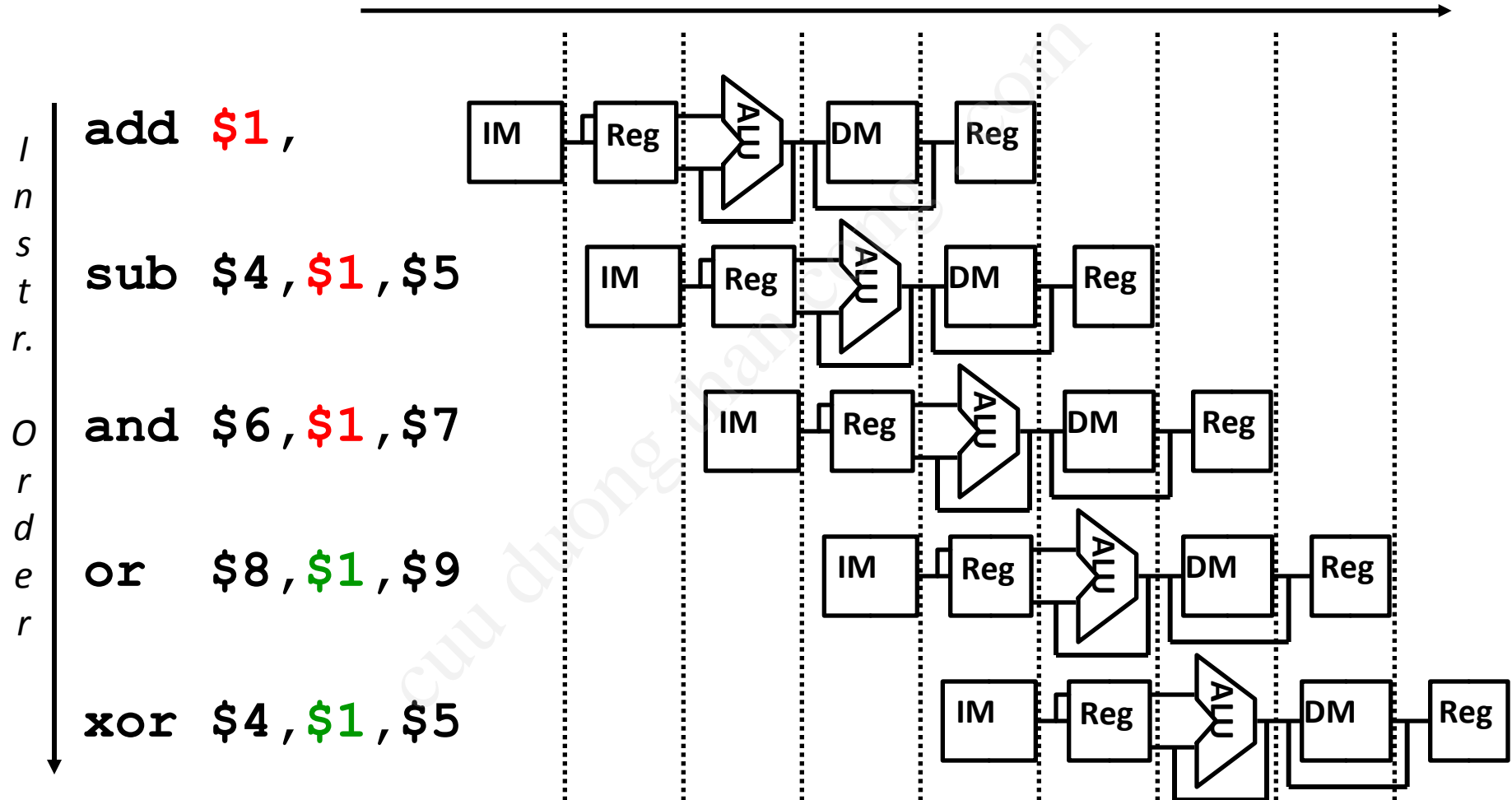


❑ Sửa: Bộ nhớ dữ liệu và lệnh riêng rẽ (Instr. and Data)

Truy cập tệp thanh ghi



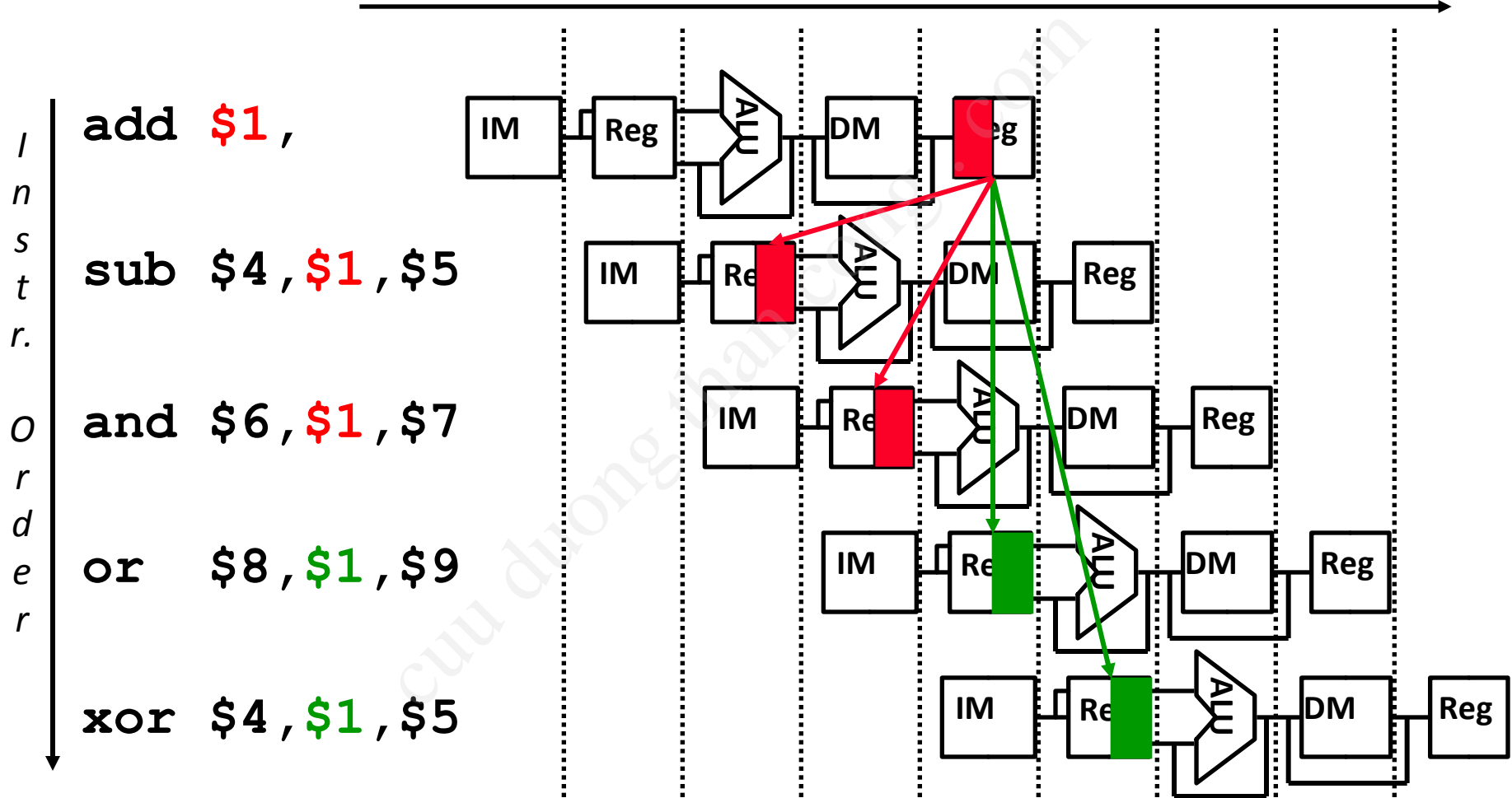
Sử dụng thanh ghi: Xung đột dữ liệu



❑ Xung đột **đọc trước khi ghi (Read before write)**

❑ Phụ thuộc dữ liệu ngược theo thời gian gây ra **xung đột**

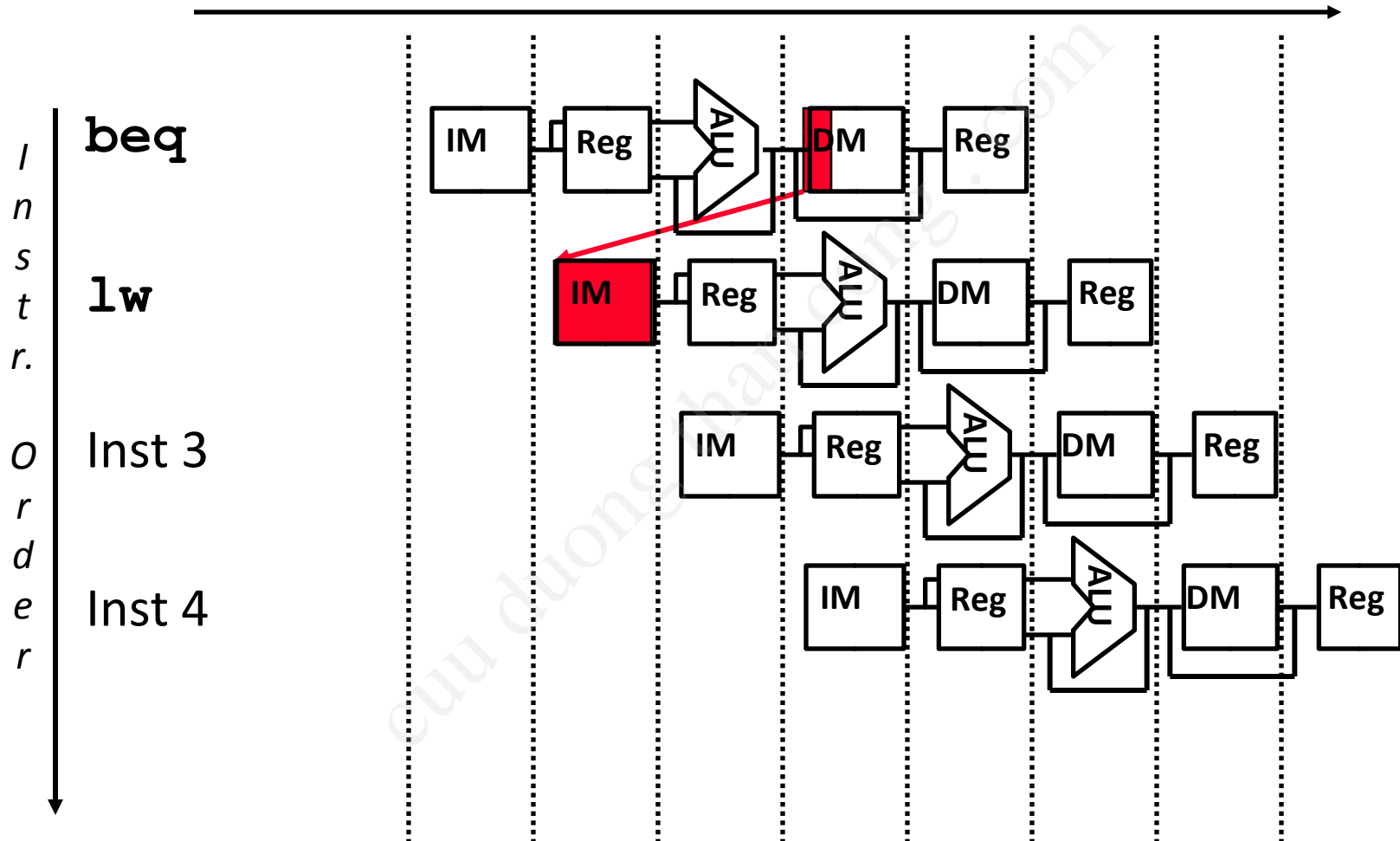
Sử dụng thanh ghi: Xung đột dữ liệu



❑ Xung đột **đọc trước khi ghi** (Read before write)

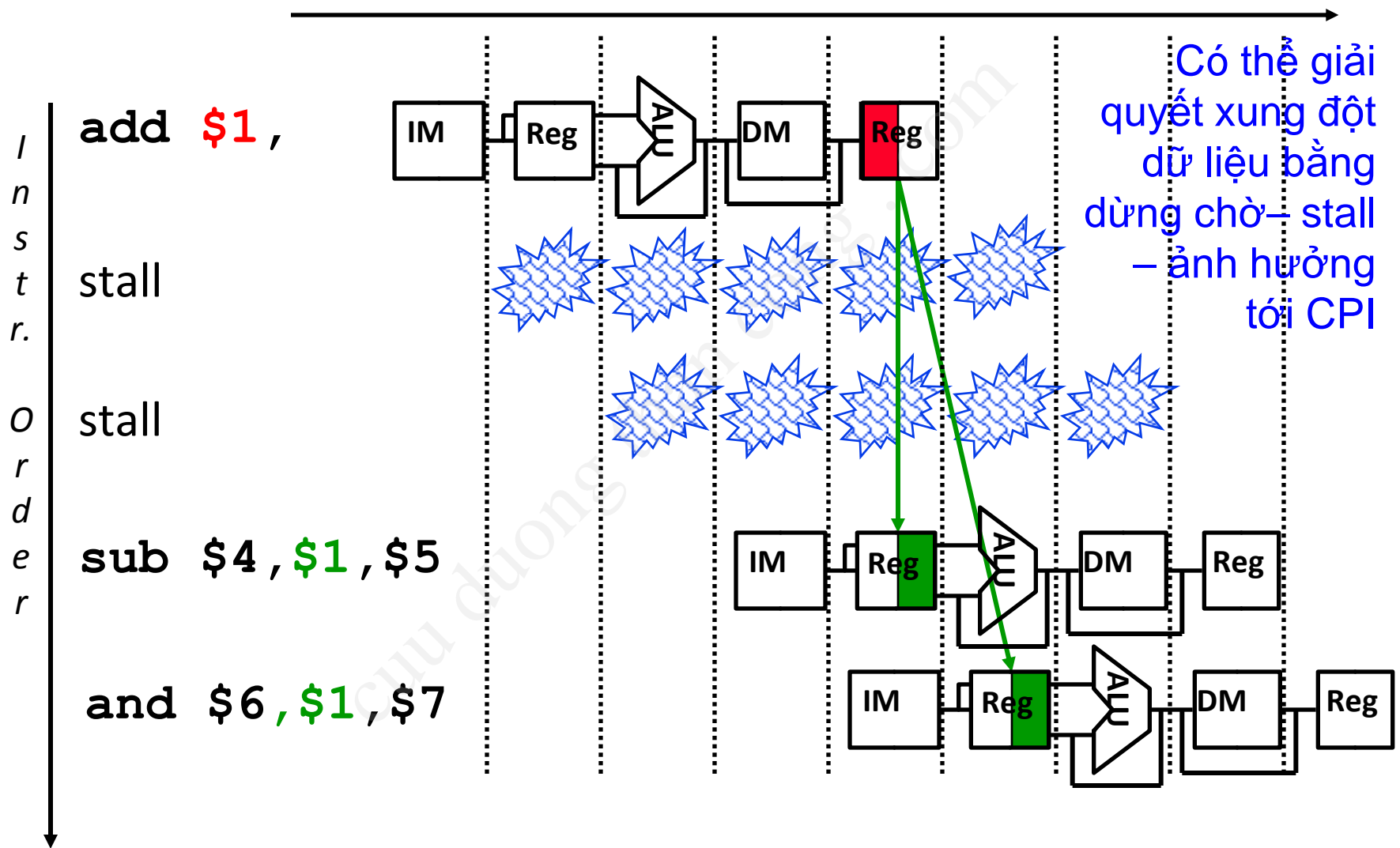
❑ Phụ thuộc dữ liệu ngược theo thời gian gây ra **xung đột**

Độc từ bộ nhớ gây xung đột dữ liệu



Dependencies backward in time cause hazards

Giải quyết xung đột: Tạm dừng



Ví dụ:

1: addi \$s0, \$zero, 10
 2: addi \$s1, \$zero, 0
 L1:
 3: add \$t0, \$t0, \$s1
 4: addi \$s1, \$s1, 1
 5: bne \$s1, \$s0, L1

Tính CPI cho chương trình

	1	2	3	4	5	6	7	8	9	10	11	12			
1	IF	ID	EX	MEM	WB										
2		IF	ID	EX	MEM	WB									
3			IF	ID	EX	MEM	WB								
4				IF	ID	EX	MEM	WB							
5					IF	ID	EX	MEM	WB						
3						*	IF	ID	EX	MEM	WB				
4								IF	ID	EX	MEM	WB			
5									IF	ID	EX	MEM	WB		

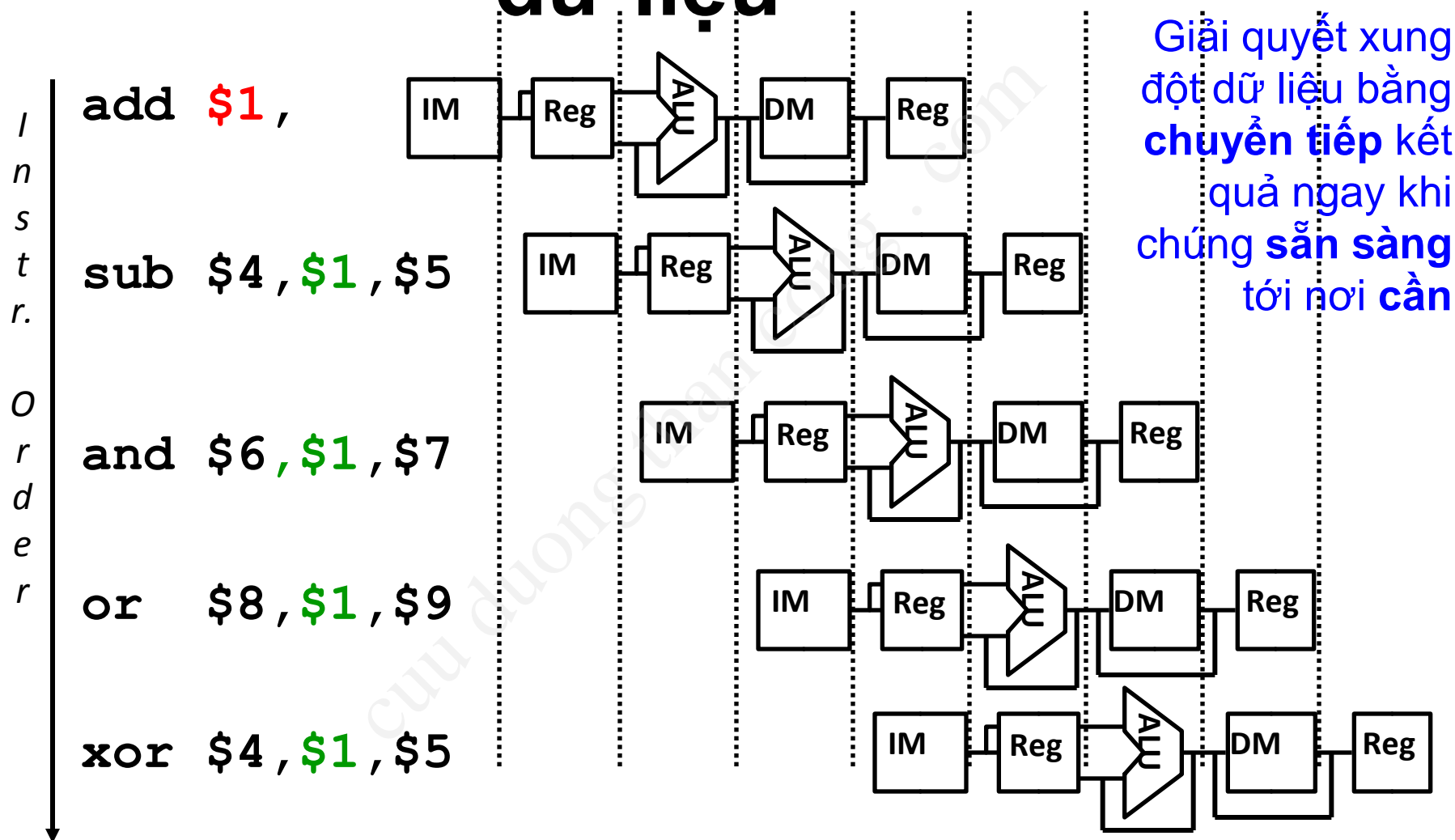
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	IF	ID	EX	MEM	WB										
2		IF	ID	EX	MEM	WB									
3			*	*	IF	ID	EX	MEM	WB						
4						IF	ID	EX	MEM	WB					
5							*	*	IF	ID	EX	MEM	WB		
3										*	*	*	*	IF	ID
4															IF
5															

Chuyển tiếp dữ liệu

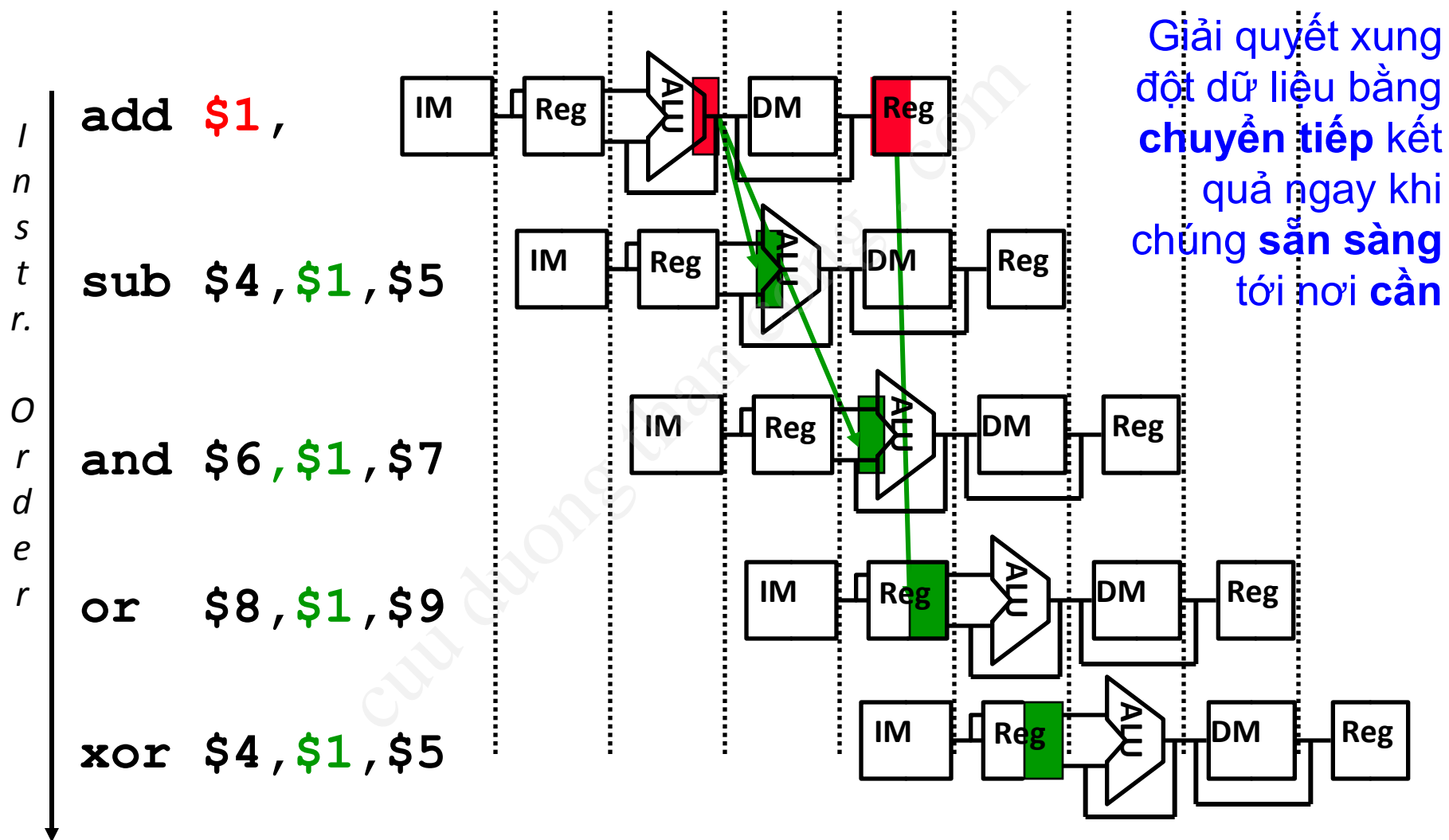
- ❑ Lấy kết quả ở thời điểm nó xuất hiện **sớm nhất** trong **bất kỳ** thanh ghi pipeline nào, và chuyển tiếp nó đến khối chức năng (VD. ALU) mà cần kết quả tại chu kỳ đồng hồ đó
- ❑ Với khối chức năng ALU: đầu vào có thể từ **bất kỳ** thanh ghi pipeline nào chứ không cần từ ID/EX bằng cách
 - Thêm bộ chọn vào trước đầu vào của ALU
 - Nối dữ liệu ghi Rd ở EX/MEM hoặc MEM/WB tới một trong 2 hoặc cả 2 thanh ghi pipeline Rs và Rt thuộc giai đoạn EX.
 - Thêm phần điều khiển phần cứng để điều khiển bộ chọn
- ❑ Các khối chức năng khác cũng cần được thêm tương tự (VD. DM)
- ❑ Với chuyển tiếp có thể đạt được $CPI = 1$ ngay khi có sự phụ thuộc dữ liệu

Giải quyết xung đột: Chuyển tiếp

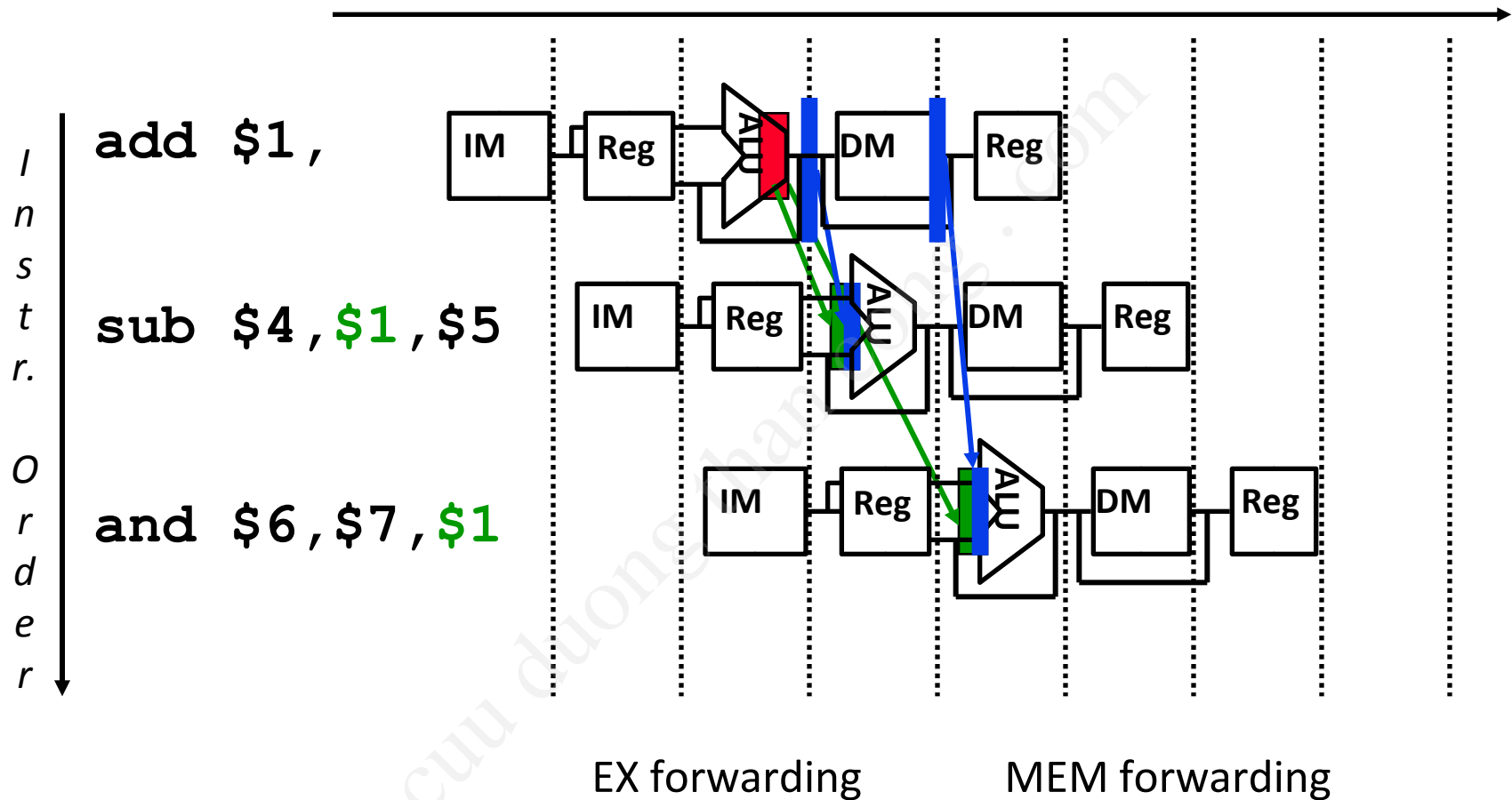
dữ liệu



Giải quyết xung đột: Chuyển tiếp dữ liệu

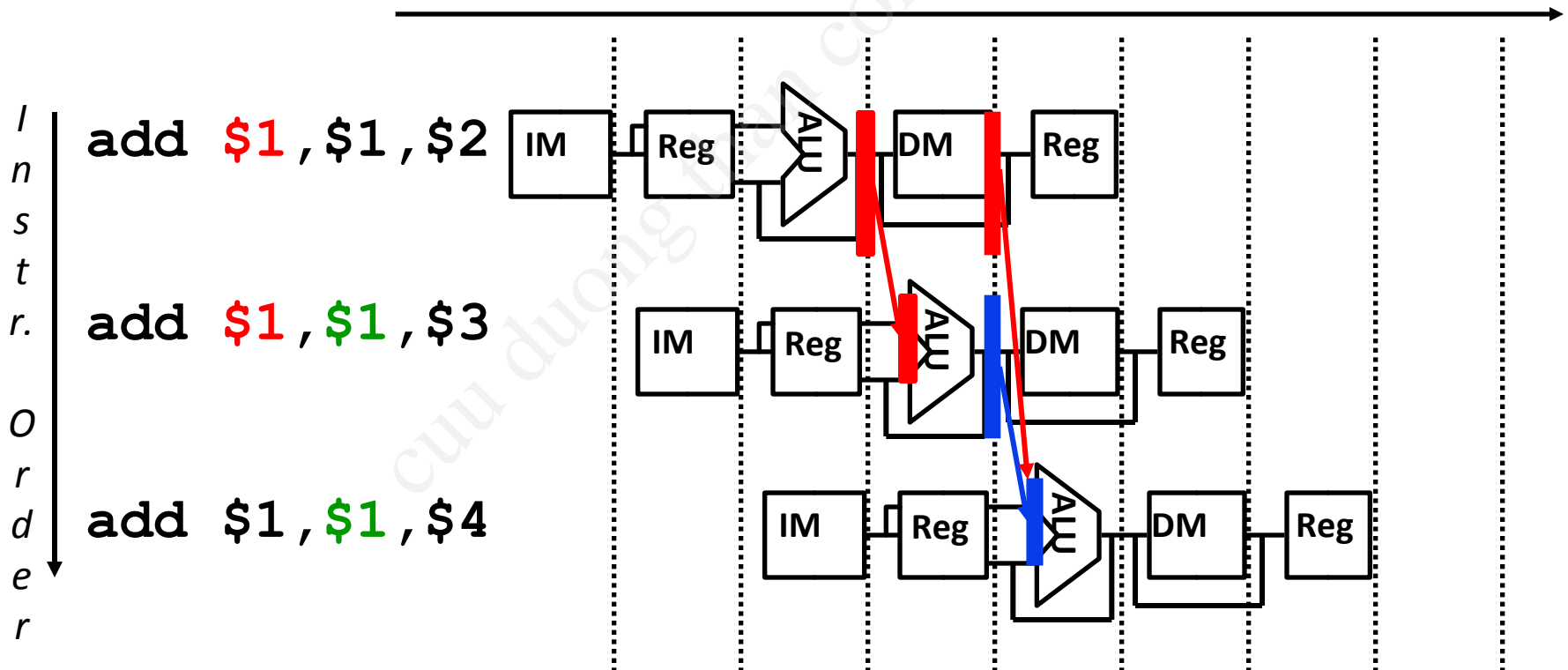


Minh họa triển khai chuyển tiếp

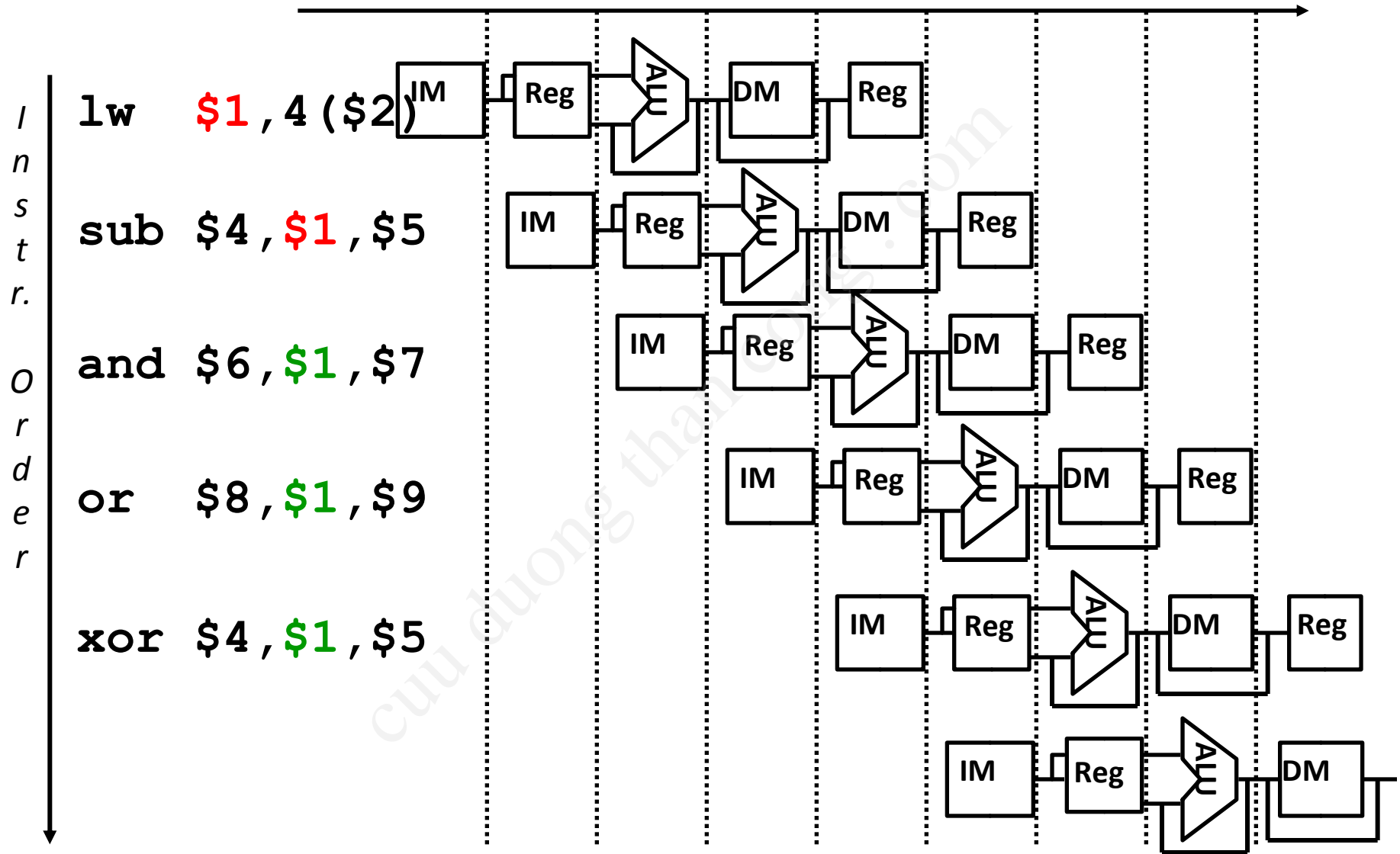


Xung đột dữ liệu khi chuyển tiếp

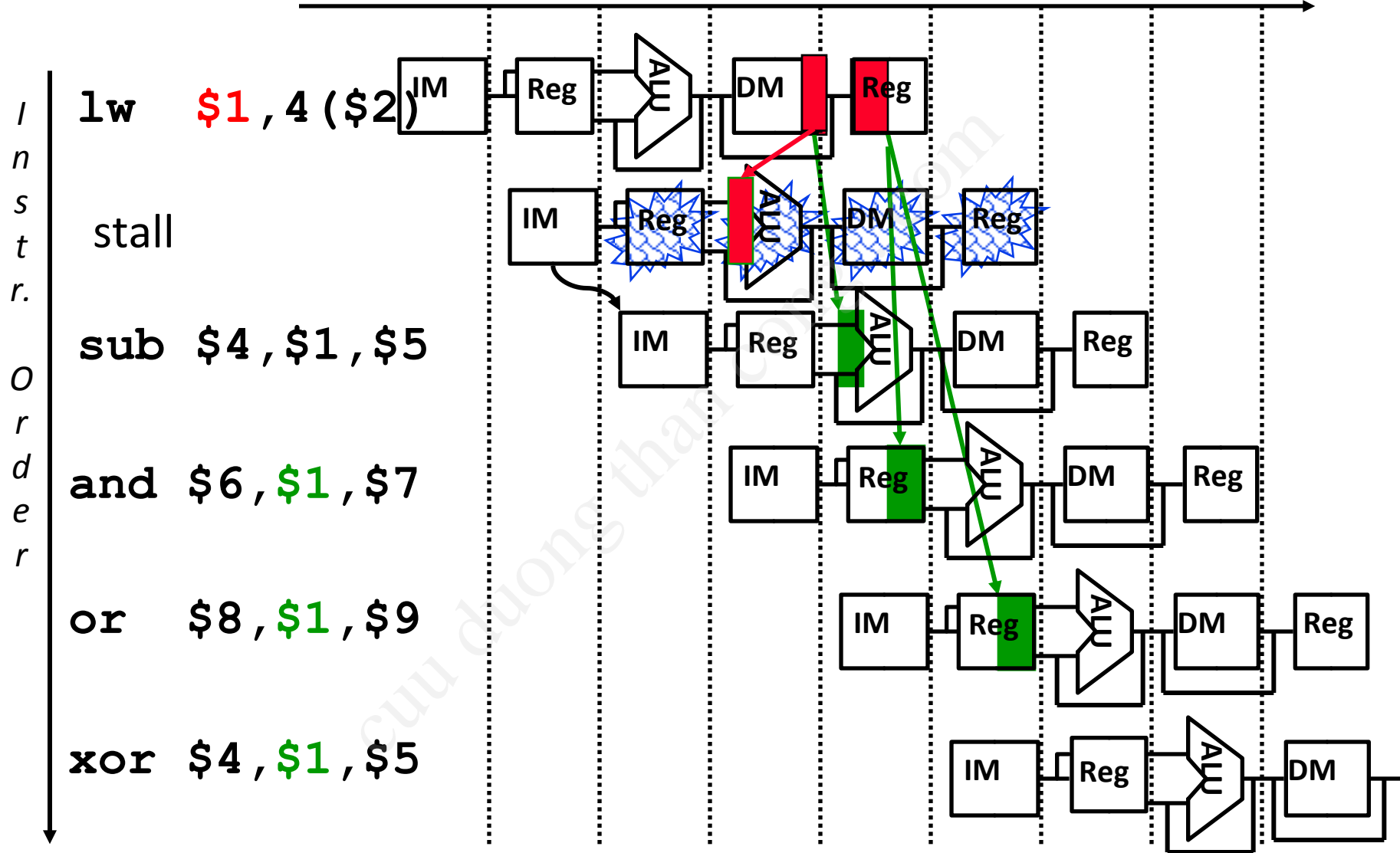
- ❑ Một loại xung đột dữ liệu xuất hiện khi chuyển tiếp: Xung đột giữa kết quả của lệnh đang ở giai đoạn WB và lệnh đang ở giai đoạn MEM – kết quả nào cần được chuyển tiếp?



Xung đột dữ liệu khi có lệnh lw



Xung đột dữ liệu khi có lệnh lw

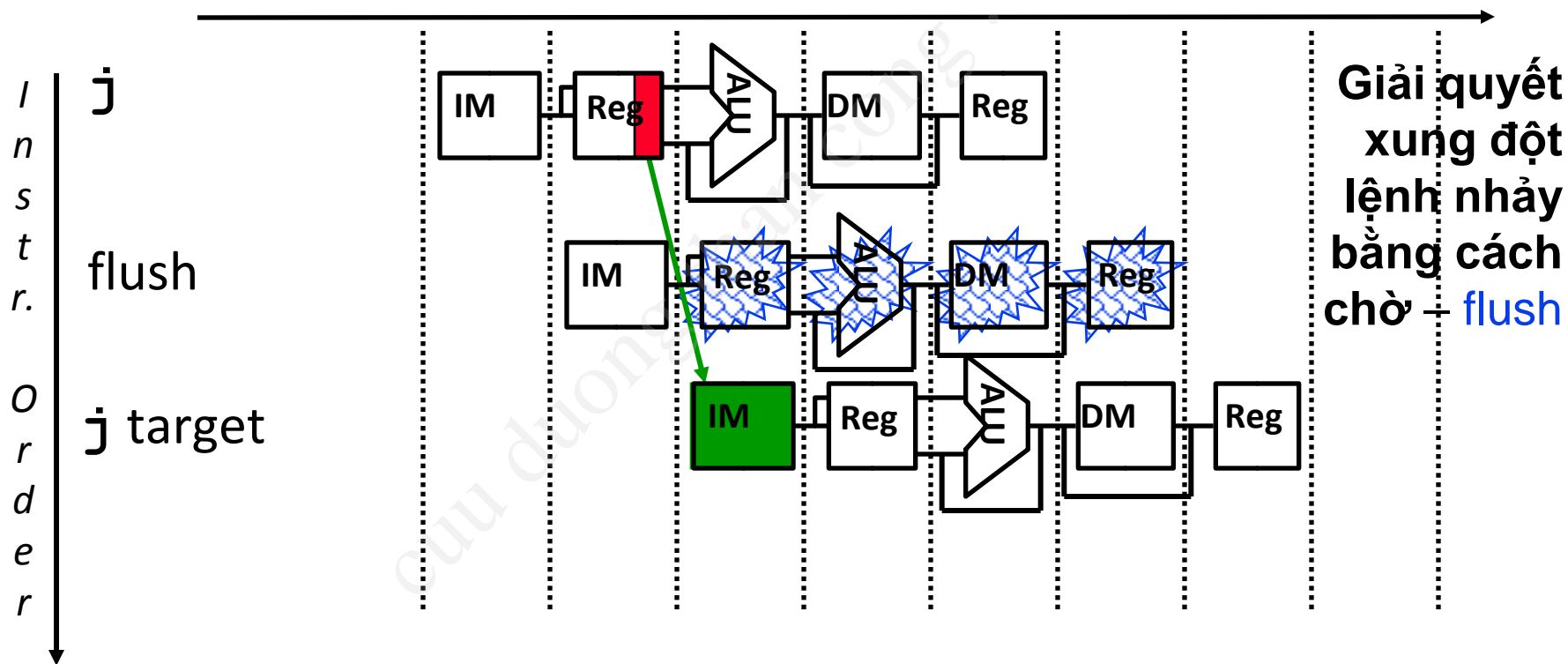


❑ Sẽ vẫn cần **một chu kỳ chờ** ngay cả khi có chuyển tiếp

Xung đột điều khiển

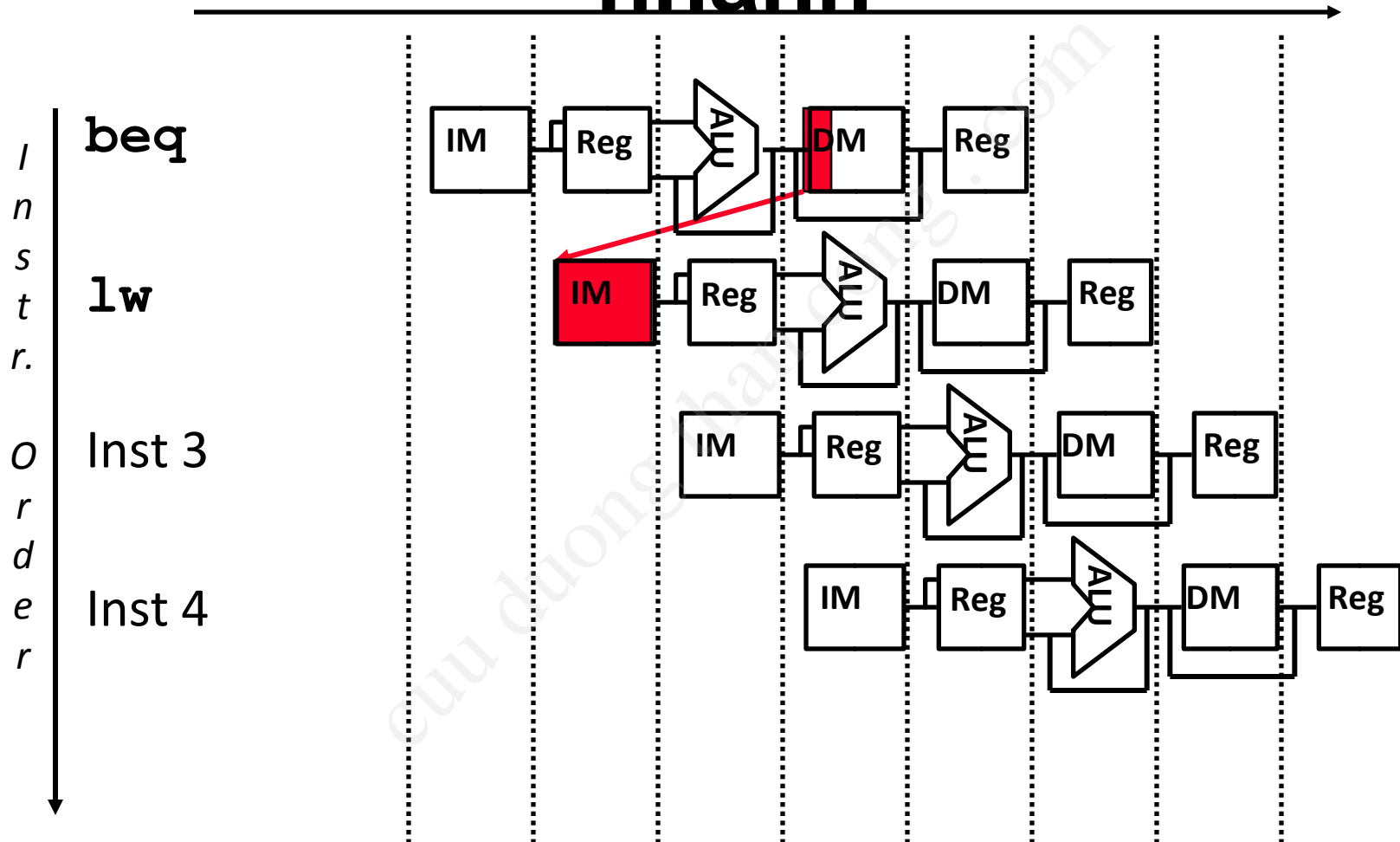
- ❑ Khi địa chỉ các lệnh không tuần tự (i.e., $PC = PC + 4$); xuất hiện khi có các lệnh thay đổi dòng chương trình
 - Lệnh rẽ nhánh không điều kiện (j, jal, jr)
 - Lệnh rẽ nhánh có điều kiện (beq, bne)
 - Ngắt, Exceptions
- ❑ Giải pháp
 - Tạm dừng (ảnh hưởng CPI)
 - Tính toán điều kiện rẽ nhánh càng sớm càng tốt trong giai đoạn pipeline giảm số chu kỳ phải dừng
 - Rẽ nhánh chậm (Delayed branches - Cần hỗ trợ của trình dịch)
 - Dự đoán và hy vọng điều tốt nhất!
- ❑ Xung đột điều khiển ít xảy ra, nhưng không có giải pháp giải quyết hiệu quả như chuyển tiếp đối với xung đột dữ liệu

Lệnh nhảy: Cần một chu kỳ dừng

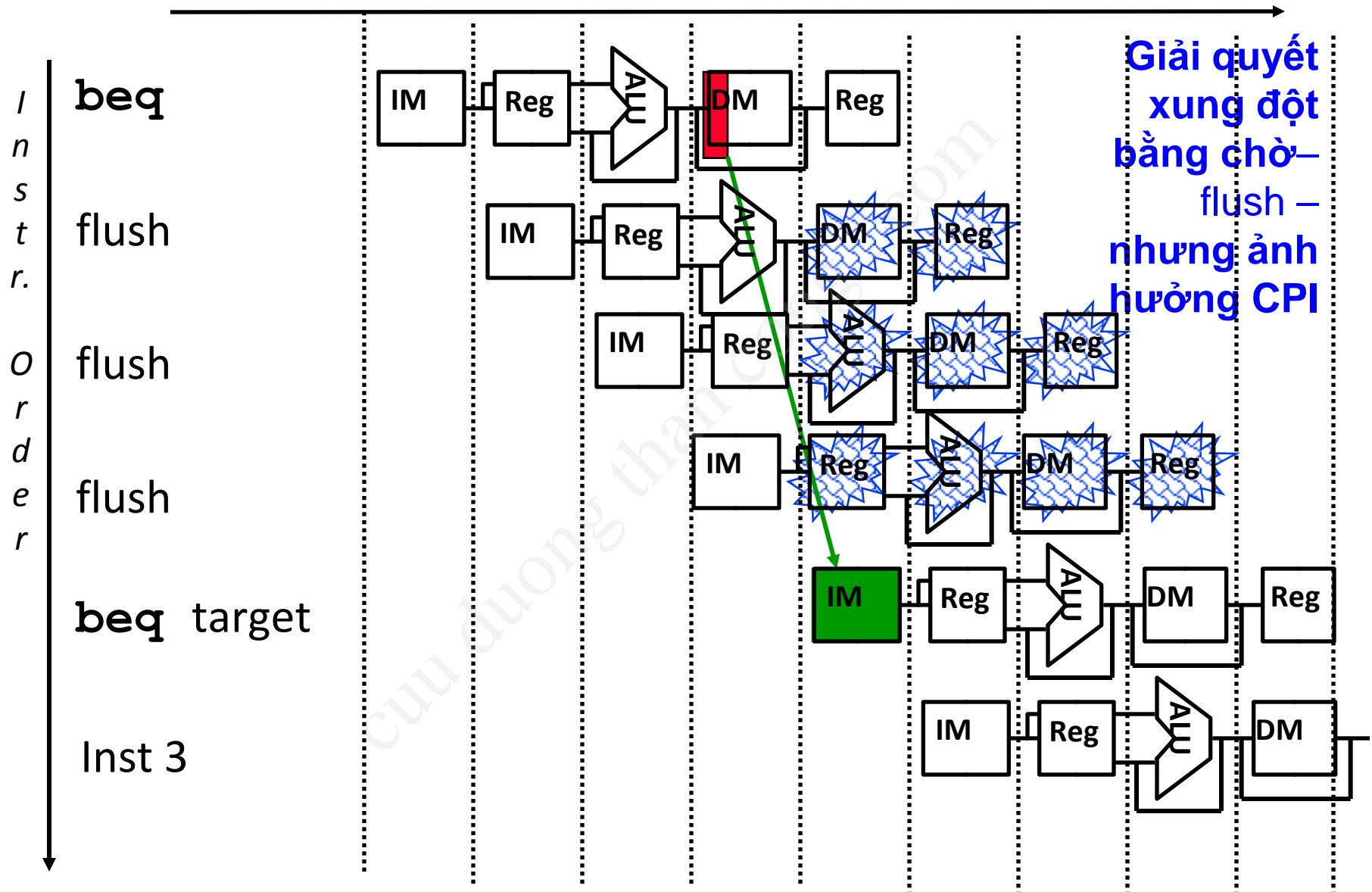


- ❑ Lệnh nhảy không được giải mã cho đến giai đoạn ID, cần một lệnh xóa (flush)
 - Để xóa, đặt trường mã lệnh của thanh ghi pipeline IF/ID bằng 0 (làm nó trở thành 1 lệnh noop)
- ❑ Lệnh nhảy rất ít xuất hiện – chỉ chiếm 3% số lệnh trong SPECint

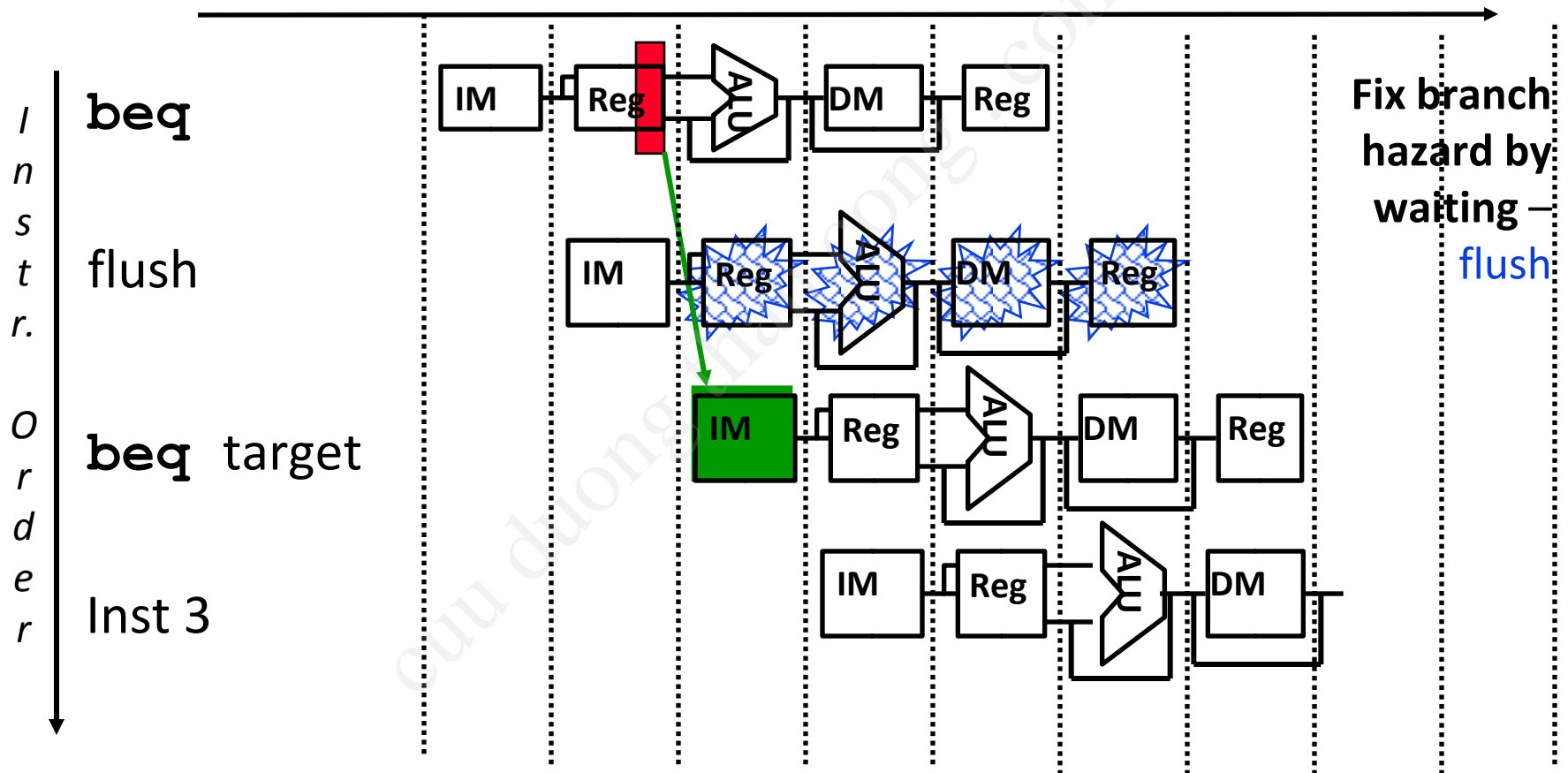
Xung đột điều khiển lệnh rẽ nhánh



Giải quyết xung đột điều khiển lệnh rẽ nhánh



Giải quyết xung đột điều khiển lệnh rẽ nhánh



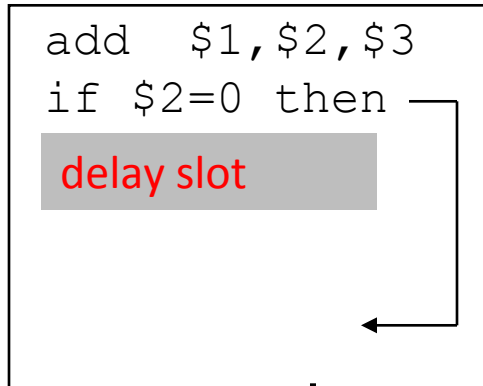
- ❑ Tính toán điều kiện rẽ nhánh càng sớm càng tốt, tức là trong giai đoạn giải mã → chỉ cần 1 chu kỳ chờ

Rẽ nhánh chậm

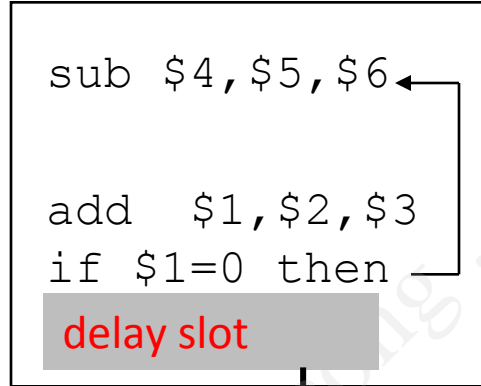
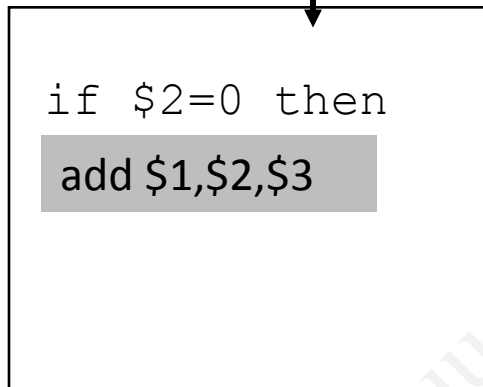
- ❑ Nếu phần cứng cho rẽ nhánh nằm ở giai đoạn ID, ta có thể loại bỏ các chu kỳ chờ rẽ nhánh bằng cách sử dụng **rẽ nhánh chậm (delayed branches)** – luôn thực hiện lệnh theo sau lệnh rẽ nhánh – rẽ nhánh có tác dụng **sau** lệnh kế tiếp nó
 - Trình dịch MIPS compiler chuyển 1 **lệnh an toàn** (không bị ảnh hưởng bởi lệnh rẽ nhánh) tới sau lệnh rẽ nhánh (vào khe trễ). Vì vậy sẽ **dầu** được sự rẽ nhánh chậm
- ❑ Với pipeline sâu (nhiều giai đoạn), trễ rẽ nhánh tăng cần nhiều lệnh được chèn vào sau lệnh rẽ nhánh
 - Rẽ nhánh chậm đang được thay thế bởi các phương pháp khác tốn kém hơn nhưng mềm dẻo (động) hơn như dự đoán rẽ nhánh
 - Sự phát triển của IC cho phép có bộ dự đoán rẽ nhánh ít tốn kém hơn

Sắp xếp lệnh trong rẽ nhánh chậm

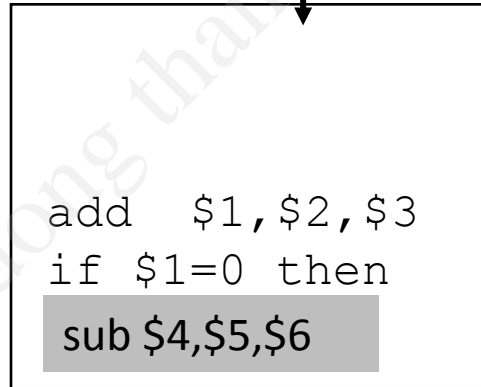
A. Từ trước lệnh rẽ nhánh B. Từ đích lệnh rẽ nhánh



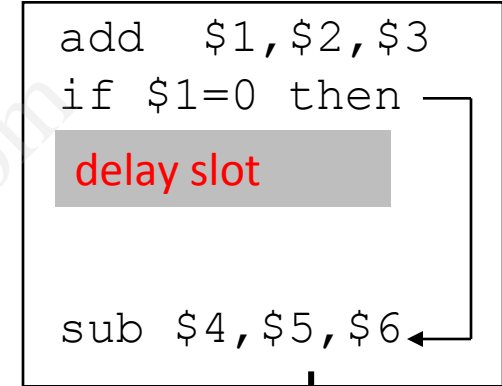
becomes



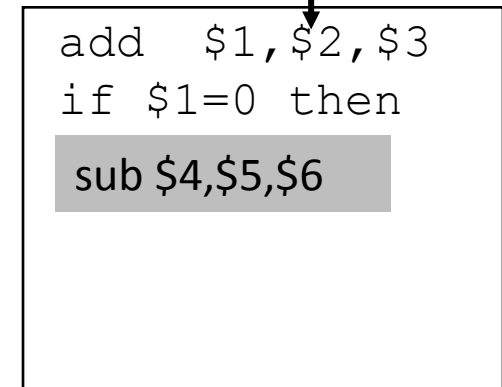
becomes



C. Từ nhánh sai



becomes



- ❑ TH A là lựa chọn tốt nhất, điền được khe trễ và giảm I
- ❑ TH B , lệnh sub cần sao chép lại, tăng I
- ❑ TH B và C, phải đảm bảo thực hiện lệnh sub không ảnh hưởng khi không rẽ nhánh

Tóm tắt

- ❑ Các bộ xử lý hiện đại đều dùng kỹ thuật pipeline
- ❑ Pipelining không làm giảm **độ trễ** của 1 nhiệm vụ đơn lẻ, nó giúp tăng **thông lượng** của toàn bộ
- ❑ Tăng tốc tiềm năng: $CPI = 1$ và đồng hồ nhanh, T_c nhỏ
- ❑ Tốc độ đồng hồ bị hạn chế bởi giai đoạn pipeline **chậm nhất**
 - Các giai đoạn pipeline không cân bằng làm giảm hiệu suất
 - Thời gian “**làm đầy**” pipeline và thời gian “**làm trống**” pipeline ảnh hưởng đến độ tăng tốc khi pipeline sâu () và đoạn mã ngắn
- ❑ Cần phát hiện và giải quyết xung đột
 - Dừng ảnh hưởng xấu tới CPI (làm CPI lớn hơn giá trị lý tưởng 1)