KIẾN TRÚC MÁY TÍNH

ET4270

TS. Nguyễn Đức Minh

[Adapted from Computer Organization and Design, 4th Edition, Patterson & Hennessy, © 2008, MK]
[Adapted from Computer Architecture lecture slides, Mary Jane Irwin, © 2008, PennState University]

Tổ chức lớp

Số tín chỉ	3 (3-1-1-6)
Giảng viên	TS. Nguyễn Đức Minh
Văn phòng	C9-401
Email	minhnd1@gmail,com
Website	https://sites.google.com/site/fethutca/home • Username: ca.fet.hut@gmail.com • Pass: dungkhoiminh
Sách	*Computer Org and Design*, 3rd Ed., Patterson &Hennessy, ©2007 *Digital Design and Computer Architecture*, David Money Harris
Thí nghiệm	3 bài
Bài tập	Theo chương, đề bài xem trên trang web

Điểm số

Điều kiện thi **Lab**

Bài thi giữa kỳ **30%**

Bài tập	20%	(Tối đa 100 điểm)
Tiến trình	10%	

Tối đa: 100 điểm,

Bắt đầu: 50 điểm

Tích lũy, trừ qua trả lời câu hỏi trên lớp và đóng góp tổ chức lớp

Bài thi cuối kỳ **70%**

Lịch học

- Thời gian:
  - Từ 14h00 đến 17h20
  - Lý thuyết: 11 buổi x 135 phút / 1 buổi
  - Bài tập: 4 buổi x 135 phút / 1 buổi
  - Thay đổi lịch (nghỉ, học bù) sẽ được thông báo trên website trước 2 ngày

Kết luận chương 2

- **Dữ liệu và chỉ thị** cho máy tính được biểu diễn bằng các **chuỗi bit**. Giá trị của dữ liệu, ý nghĩa của chỉ thị máy được quy định trong **phương pháp mã hóa**.
- **Thiết kế kiến trúc tập lệnh**:
  - Kích thước và kiểu dữ liệu
  - Phép toán: loại nào được hỗ trợ
  - Định dạng và mã hóa chỉ thị: Chỉ thị được giải mã thế nào?
  - Vị trí toán hạng và kết quả
    - Số lượng toán hạng?
    - Giá trị toán hạng được lưu ở đâu?
    - Kết quả được lưu ở vị trí nào?
    - Các toán hạng bộ nhớ được định vị thế nào?
- **Kiến trúc tập lệnh MIPS(RISC)** được thiết kế dựa trên 4 nguyên tắc cơ bản.
- **Bộ cộng trừ nhân chia** được triển khai bằng các phần tử logic hay bằng thuật toán.

Nguyên tắc thiết kế MIPS (RISC)

- Tính đơn giản quan trọng hơn tính quy tắc(Simplicity favors regularity)
  - Chỉ thị kích thước cố định (32 bit)
  - Ít định dạng chỉ thị (3 loại định dạng)
  - Mã lệnh ở vị trí cố định (6 bit đầu)
- Nhỏ hơn thì nhanh hơn
  - Số chỉ thị giới hạn
  - Số thanh ghi giới hạn
  - Số chế độ địa chỉ giới hạn
- Tăng tốc các trường hợp thông dụng
  - Các toán hạng số học lấy từ thanh ghi (máy tính dựa trên cơ chế load-store)
  - Các chỉ thị có thể chứa toán hạng trực tiếp
- Thiết kế tốt đòi hỏi sự thỏa hiệp
  - 3 loại định dạng chỉ thị

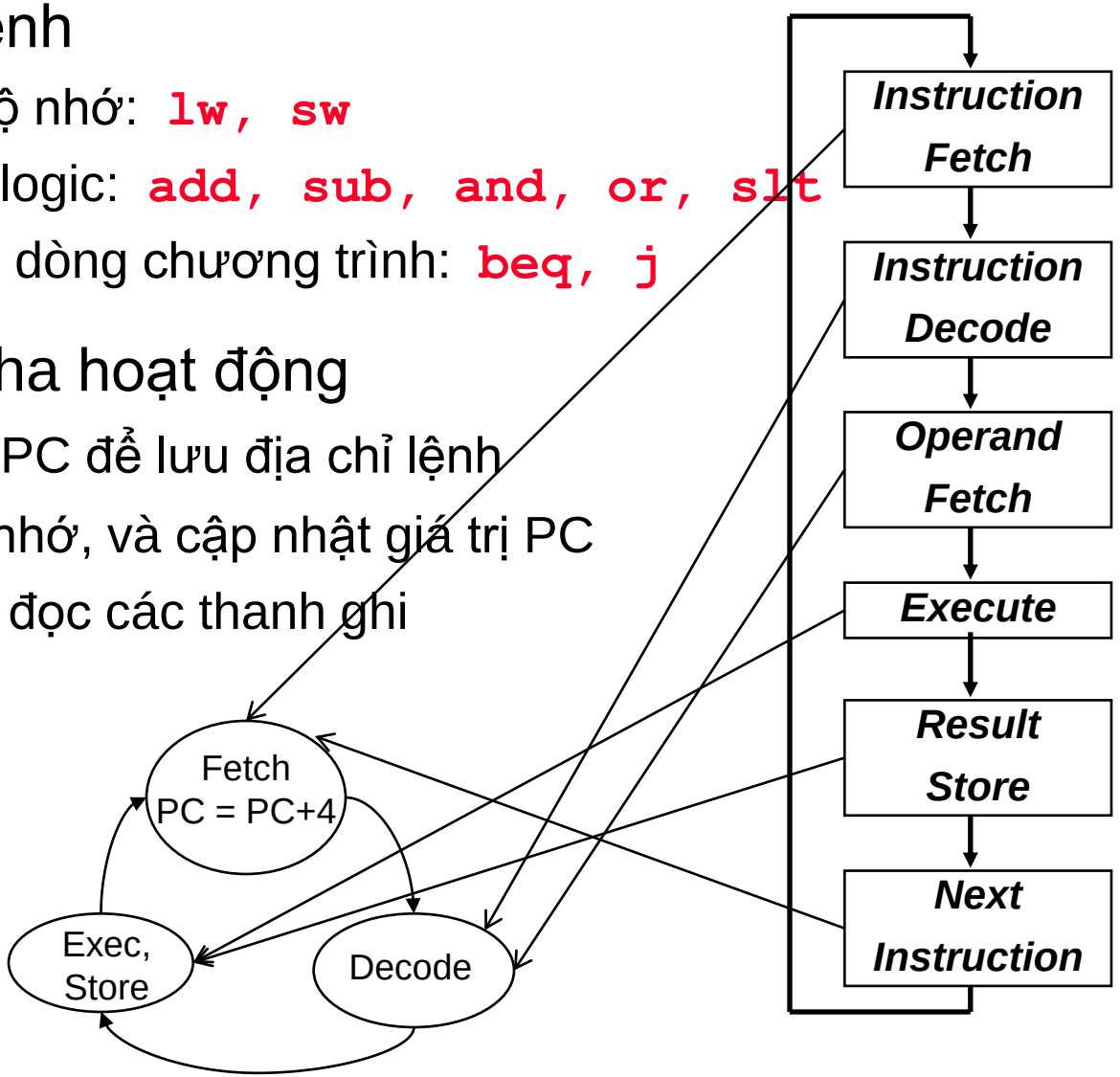
Nội dung

- Đường dữ liệu bộ xử lý MIPS
  - Đơn xung nhịp
  - Đa xung nhịp
  - Hiệu năng
- Kỹ thuật đường ống
  - Nguyên tắc hoạt động
  - Hiệu năng
  - Xung đột trong đường ống

Bộ xử lý: Đường dữ liệu và điều khiển

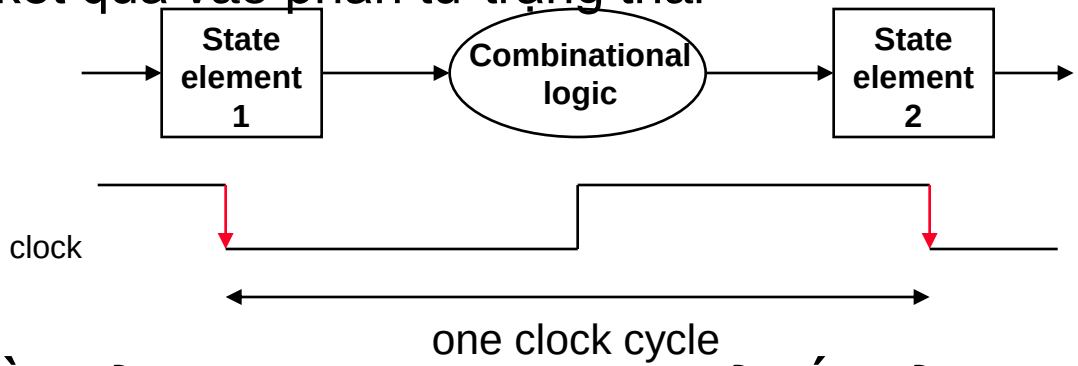
- Triển khai các lệnh
  - Lệnh truy cập bộ nhớ: `lw, sw`
  - Lệnh số học và logic: `add, sub, and, or, slt`
  - Lệnh điều khiển dòng chương trình: `beq, j`
- Triển khai các pha hoạt động
  - Dùng thanh ghi PC để lưu địa chỉ lệnh

    Đọc lệnh từ bộ nhớ, và cập nhật giá trị PC
  - Giải mã lệnh và đọc các thanh ghi
  - Thực hiện lệnh
  - Lưu kết quả



Thiết kế đồng bộ theo đồng hồ

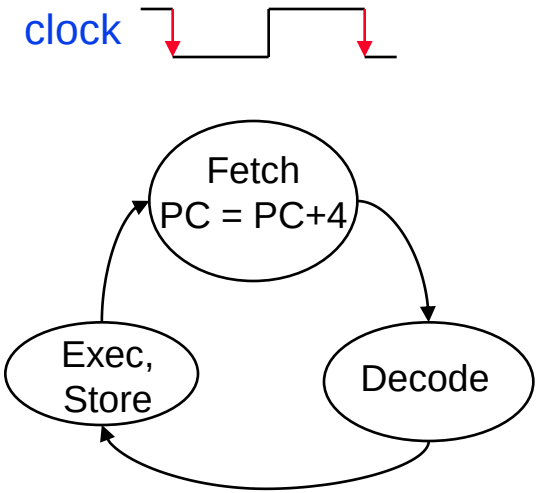
- Mạch đồng bộ theo đồng hồ: thời điểm dữ liệu trong 1 phần tử trạng thái là hợp lệ và ổn định được quy định bởi xung đồng hồ
  - Phần tử trạng thái - phần tử nhớ - VD. thanh ghi, FF
  - Kích hoạt theo sườn – các trạng thái thay đổi khi có xườn xung
- Hoạt động thông thường:
  - đọc nội dung của phần tử trạng thái -> tính giá trị bằng logic tổ hợp -> ghi kết quả vào phần tử trạng thái

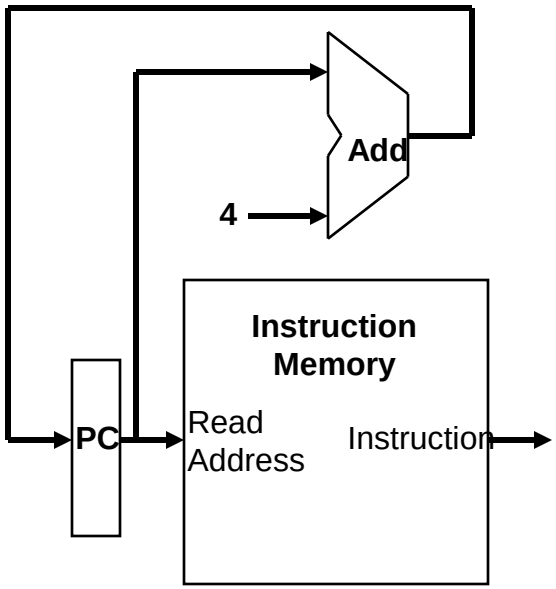


- Các phần tử trạng thái được ghi ở tất cả các chu kỳ đồng hồ. Nếu không: cần tín hiệu điều khiển việc ghi

Nạp lệnh

- Đọc lệnh tại địa chỉ (lưu trong) PC từ bộ nhớ lệnh (*eng. Instruction Memory*)
- Cập nhật giá trị PC tới địa chỉ của lệnh kế tiếp

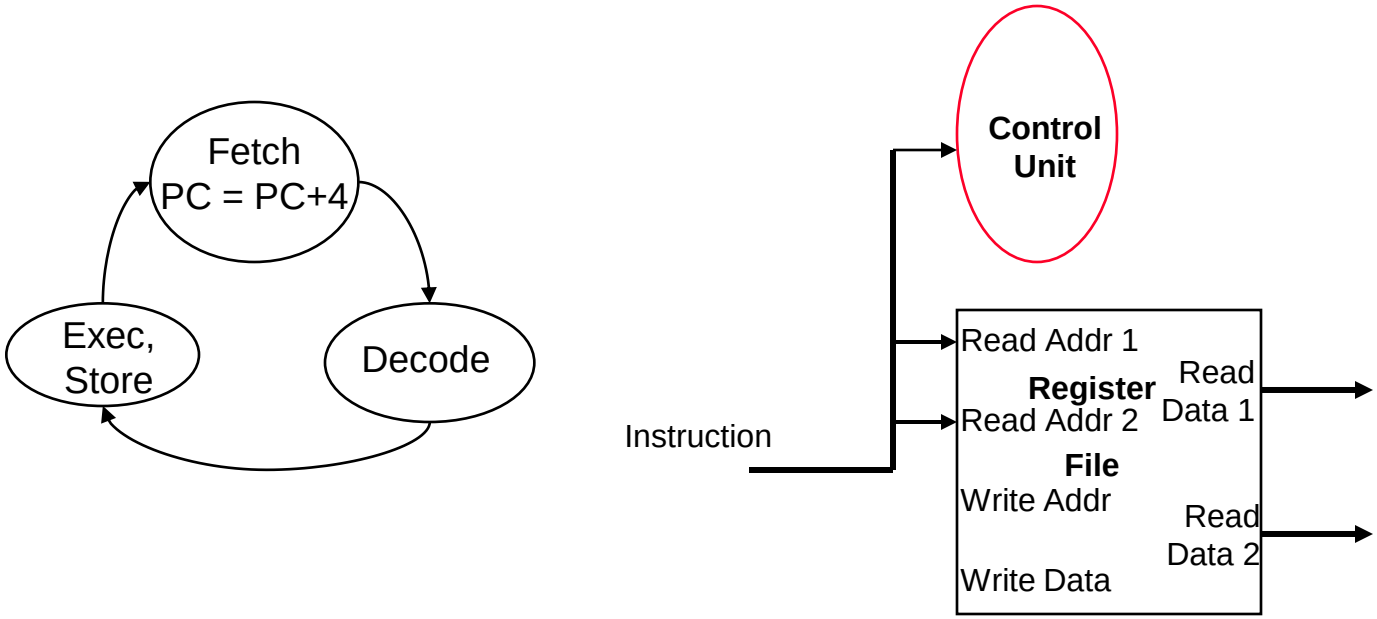




- PC được cập nhật ở mọi chu kỳ ➔ không cần tín hiệu điều khiển ghi PC.
- Đọc từ bộ nhớ lệnh được thực hiện bằng logic tổ hợp

Giải mã lệnh

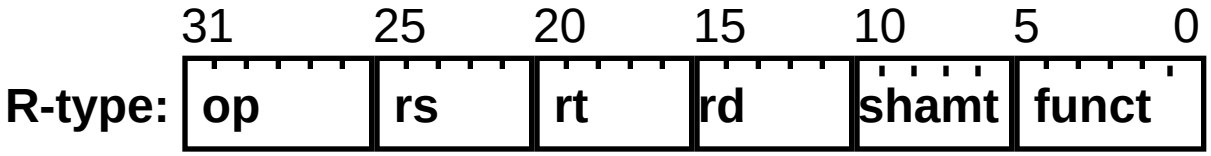
- Chuyển các bit thuộc trường mã lệnh và trường mã chức năng tới khối điều khiển



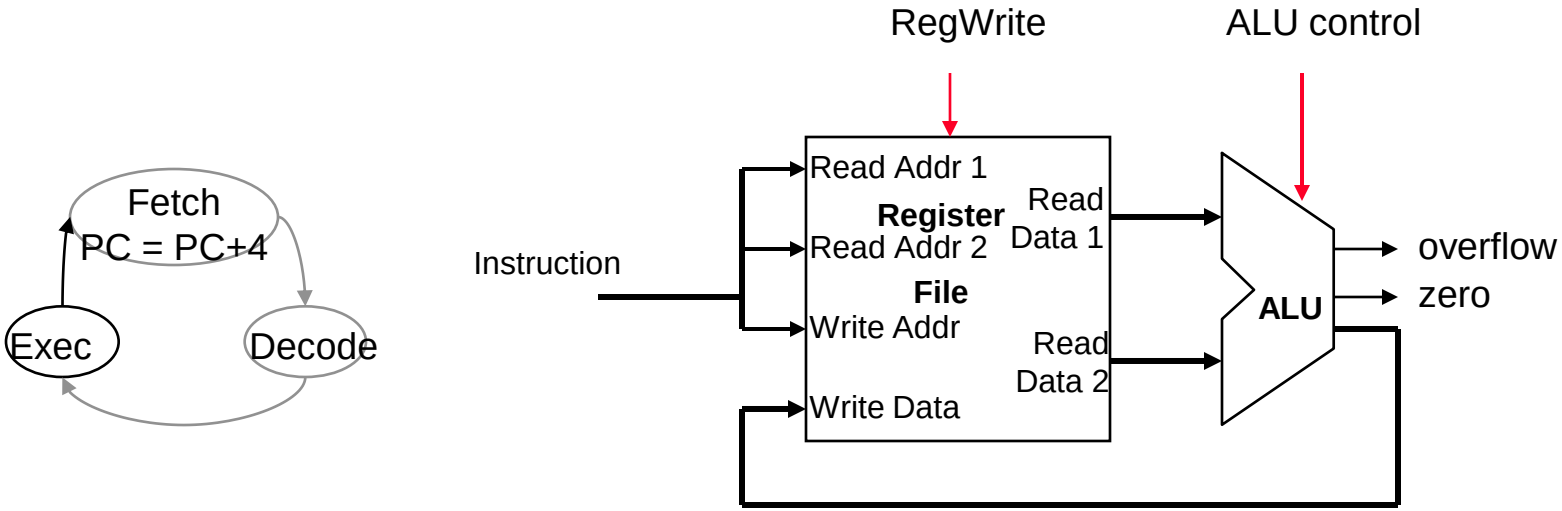
- Đọc 2 giá trị toán hạng nguồn từ tệp thanh ghi
  - Chỉ số các thanh ghi nằm trong lệnh

Thực hiện lệnh loại R

- Lệnh định dạng R (**`add, sub, slt, and, or`**)



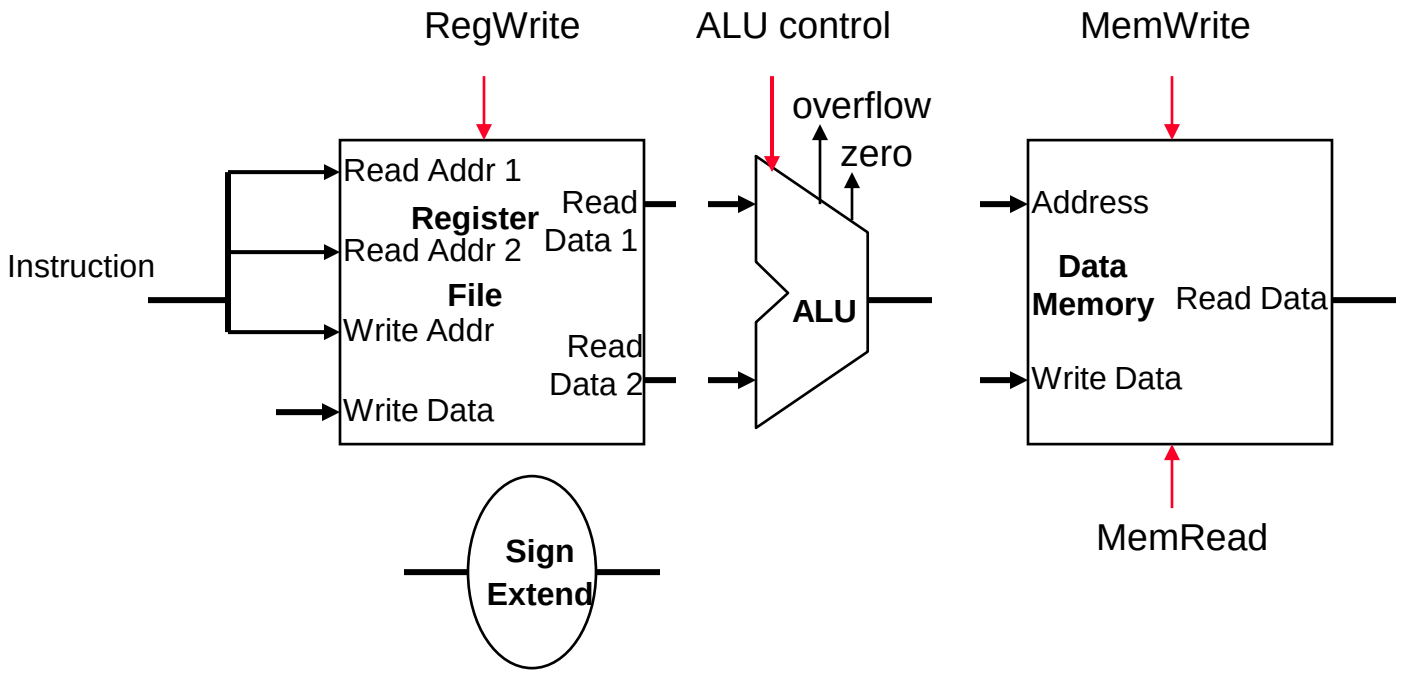
- Thực hiện phép toán (mã hóa bởi op và funct) trên giá trị toạn hạng trong rs và rt
- Ghi kết quả vào tệp thanh ghi (tại vị trí rd)



- Tệp thanh ghi không được ghi ở mọi chu kỳ ➔ cần tín hiệu điều khiển ghi riêng biệt.

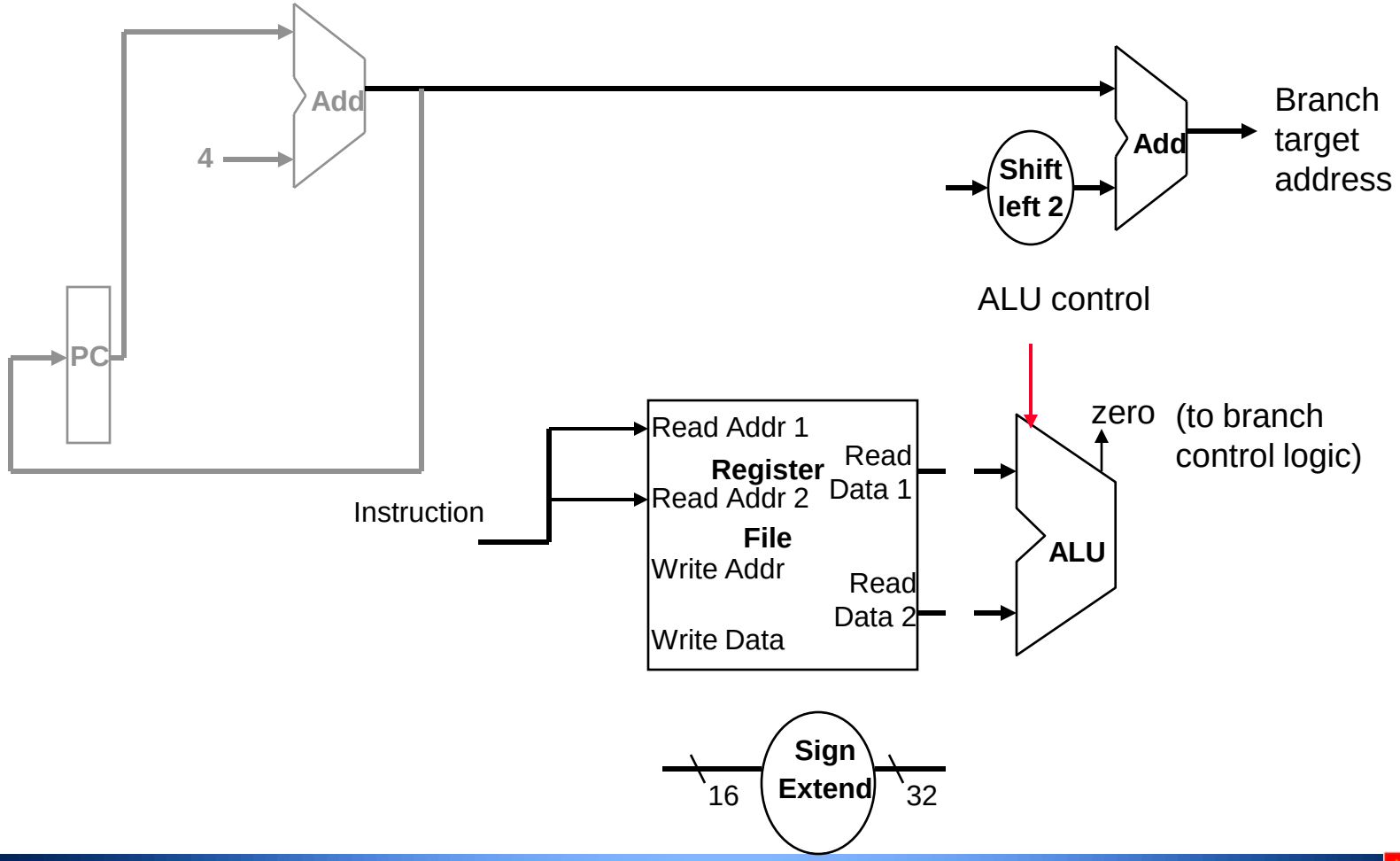
Thực hiện lệnh đọc ghi bộ nhớ

- Tính địa chỉ bộ nhớ bằng cách cộng thanh ghi cơ sở (đọc từ tệp thanh ghi khi giải mã lệnh) với giá trị offset
- ghi (sw) giá trị (được đọc từ tệp thanh ghi khi giải mã lệnh) vào bộ nhớ dữ liệu
- đọc (lw) giá trị từ bộ nhớ dữ liệu vào tệp thanh ghi



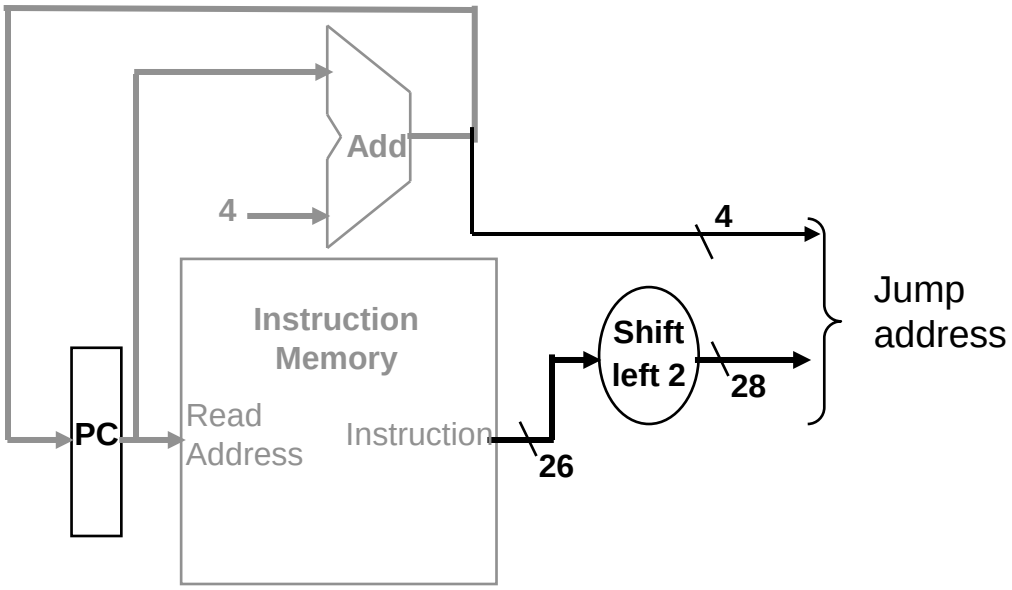
Thực hiện lệnh rẽ nhánh có điều kiện

- so sánh toán hạng đọc từ tệp thanh ghi khi giải mã
- tính địa chỉ đích bằng cách cộng giá trị PC (sau khi cập nhật) với trường offset 16 bit đã được mở rộng dấu.



Thực hiện lệnh nhảy không điều kiện

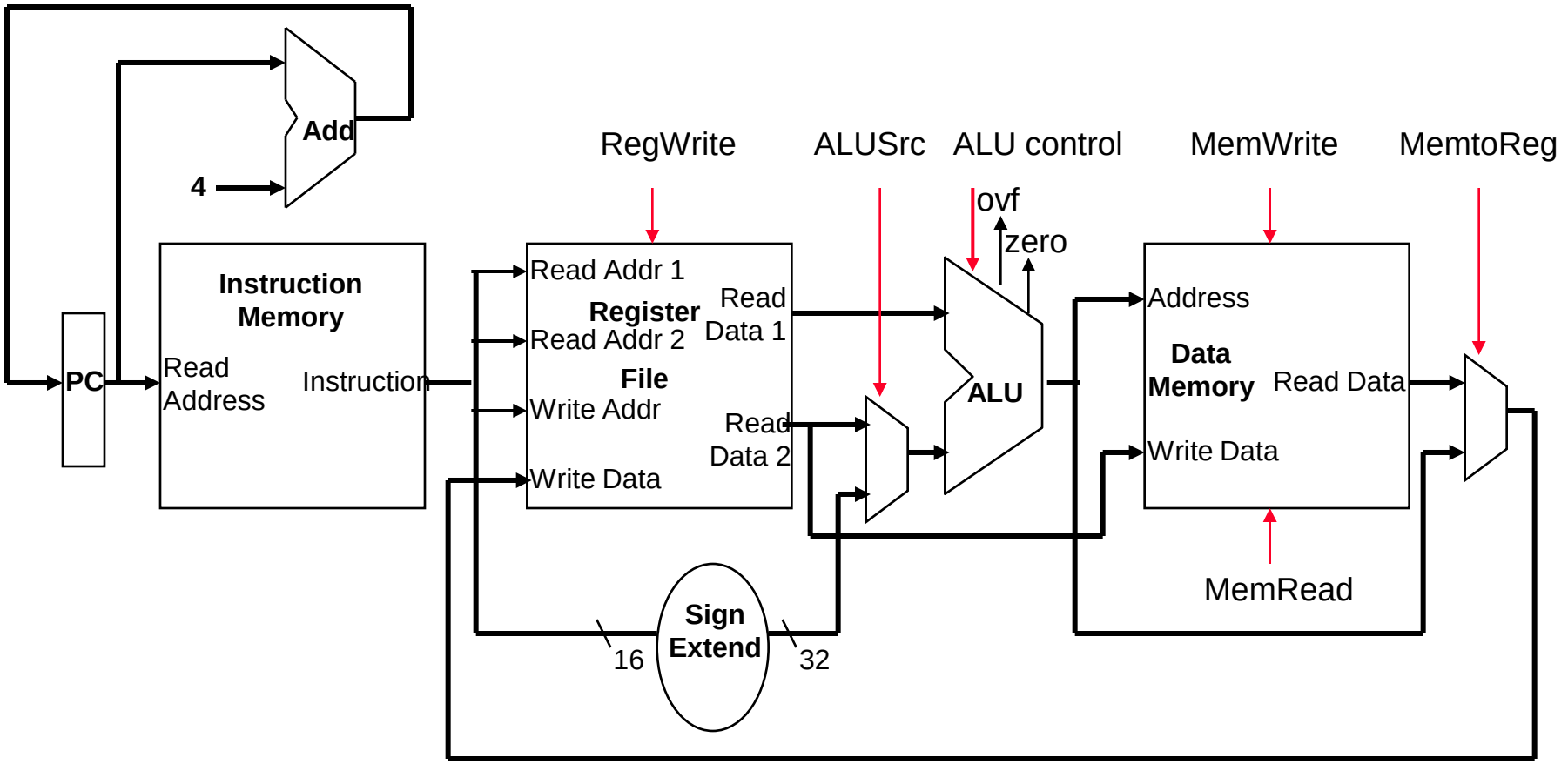
- Thay 28 bit thấp của PC bằng 26 bít thấp của lệnh được nạp và 2 bít 0



Đường dữ liệu

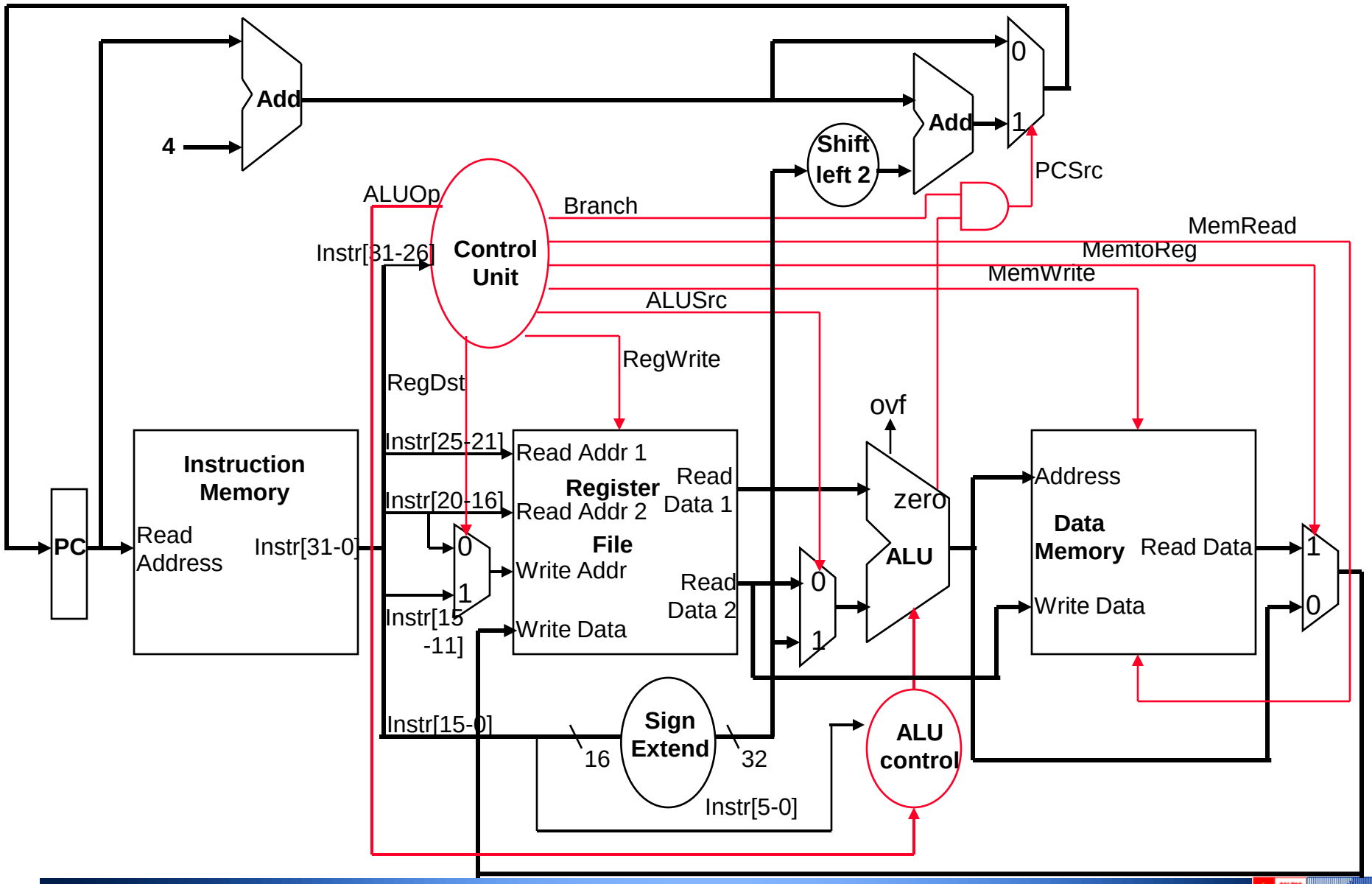
- Ghép các phần của đường dữ liệu thêm các đường tín hiệu điều khiển và bộ ghép (multiplexors)
- Thiết kế đơn xung nhịp – các pha thực hiện: nạp, giải mã and thực hiện, ghi của mỗi lệnh trong một chu kỳ đồng hồ
  - Các tài nguyên phần cứng của đường dữ liệu không thể tái sử dụng cho cùng 1 lệnh, một số tài nguyên phải nhân đôi (VD., bộ nhớ lệnh và dữ liệu riêng biệt, một vài bộ cộng)
  - bộ ghép được dùng ở đầu vào của các tài nguyên dùng chung và được điều khiển bằng tín hiệu điều khiển
- Chu kỳ đồng hồ: xác định bằng độ dài đường dữ liệu dài nhất

Đường dữ liệu: Phần nạp, thực hiện lệnh R, lw,sw

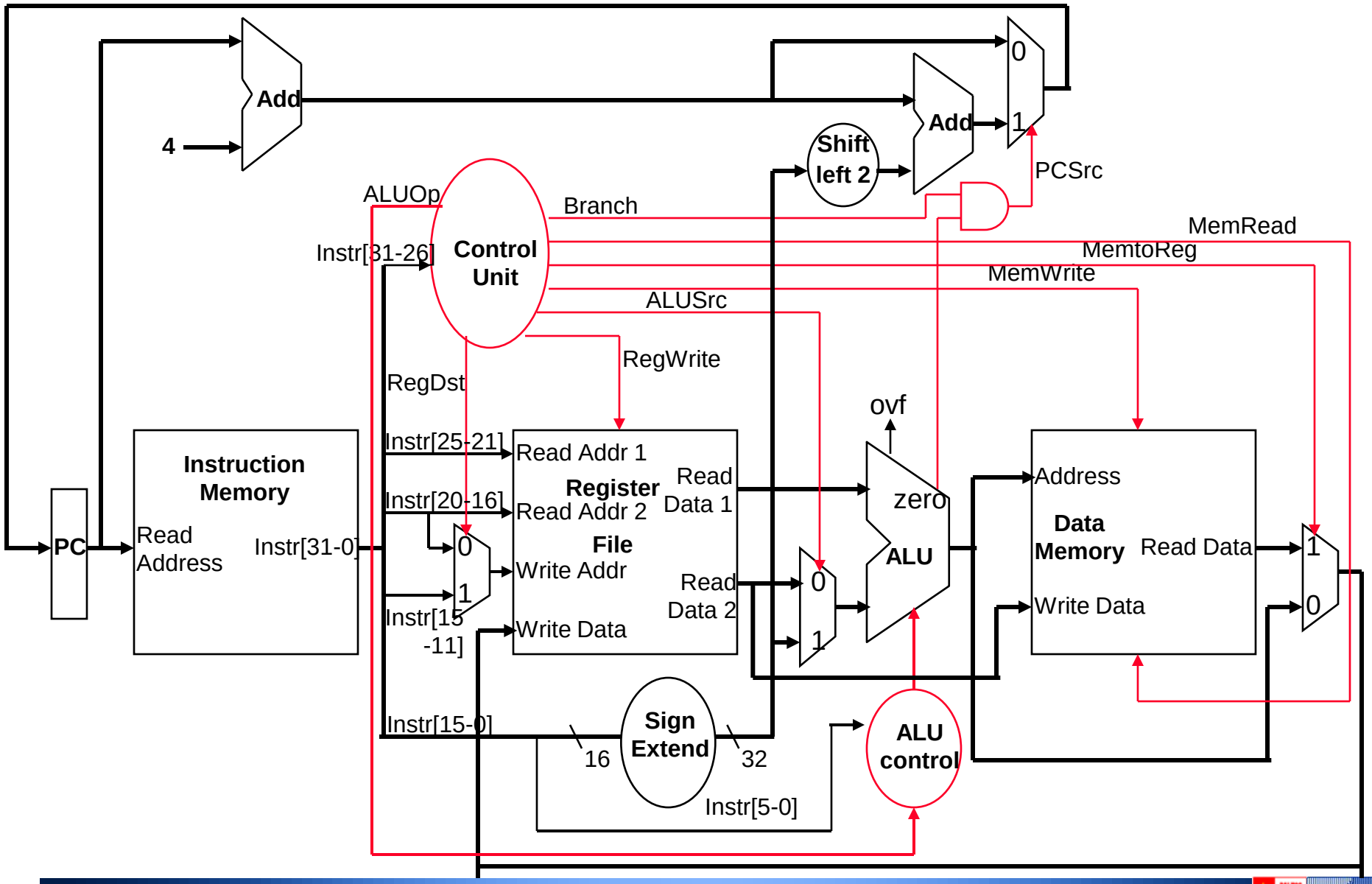


Các tín hiệu điều khiển bộ ghép: lựa chọn đầu vào cho các khối chức năng ➔ được tính bằng khối điều khiển từ trường mã lệnh (opcode) và trường chức năng lệnh (funct)

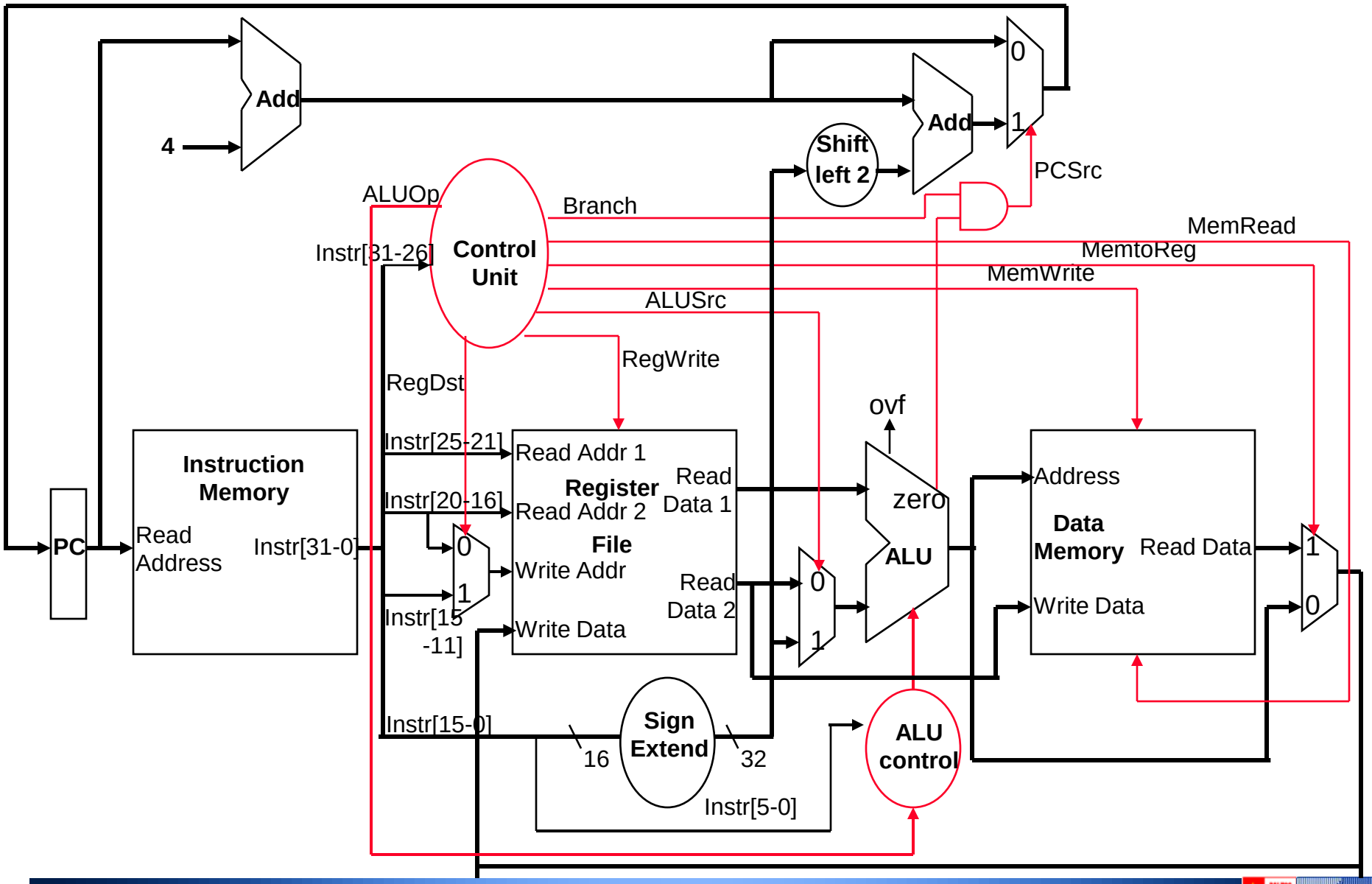
Bộ xử lý đơn xung nhịp (1) – Lệnh R



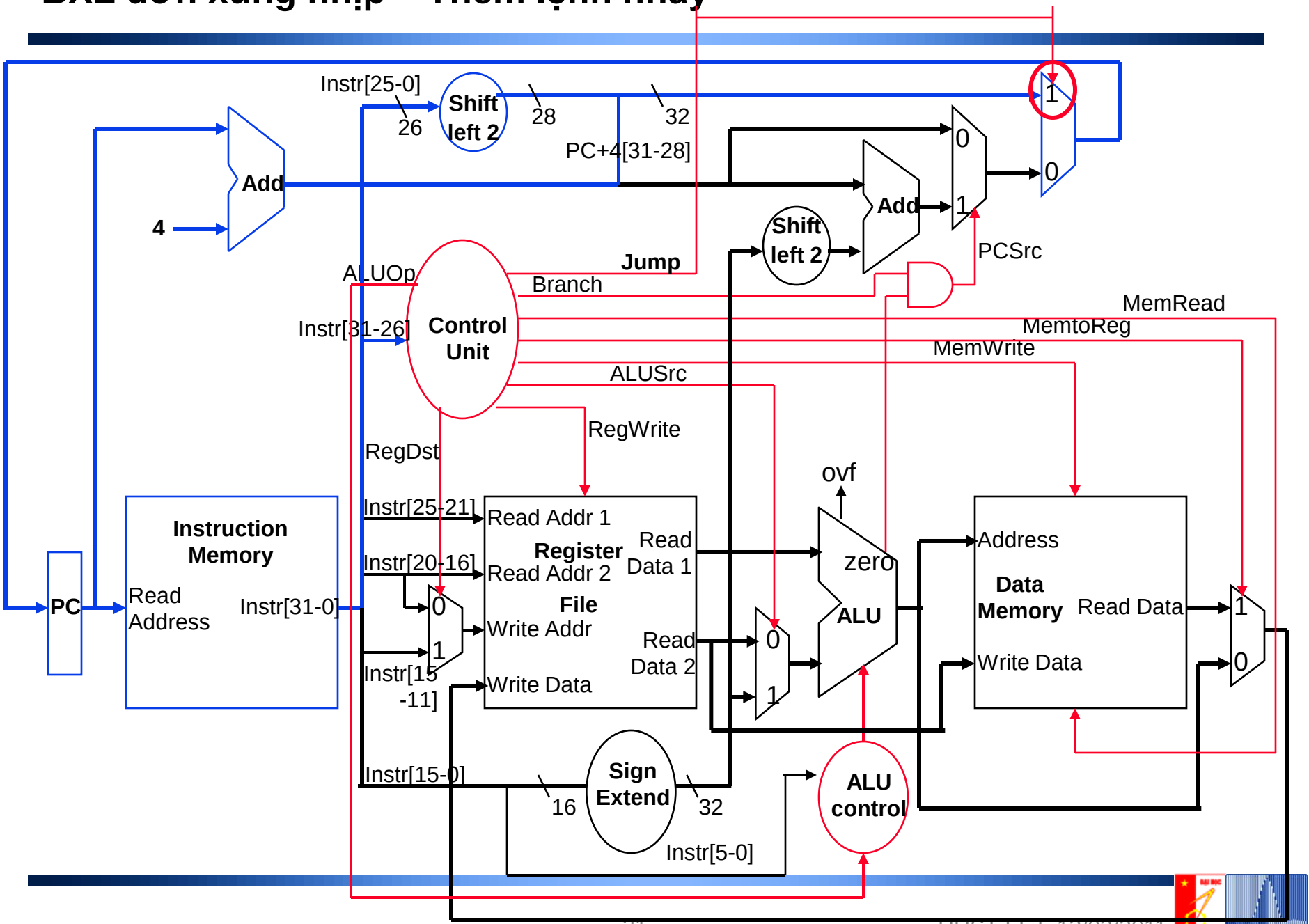
BXL đơn xung nhịp (3) – Lệnh lw, sw



BXL đơn xung nhịp (4) – Lệnh rẽ nhánh



BXL đơn xung nhịp – Thêm lệnh nhảy



Tính chu ky đồng hồ T_c – Đường dài nhất

- Tính chu kỳ đồng hồ trong trường hợp bỏ qua trễ ở bộ ghép, khối điều khiển, khối mở rộng dấu, khối đọc PC, khối dịch 2, dây dẫn, thời gian thiết lập và giữ. Cho biết độ trễ:
  - Truy cập bộ nhớ lệnh và bộ nhớ dữ liệu (200 ps)
  - Khối số học logic và bộ cộng (200 ps)
  - Truy cập tệp thanh ghi (đọc hoặc ghi) (100 ps)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type						
load						
store						
beq						
jump						

Ví dụ 3.1 – Hiệu năng thiết kế đơn xung nhịp

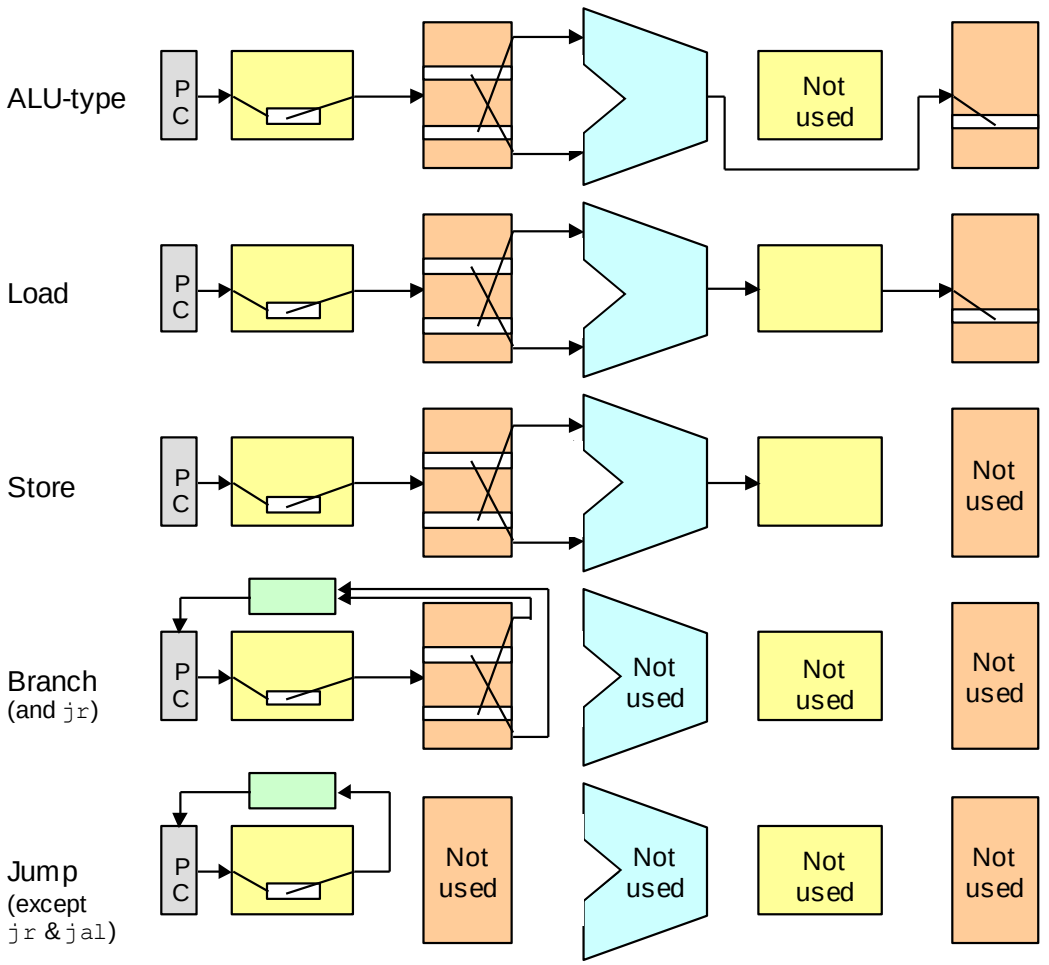
Độ trễ logic khi

Truy cập lệnh	2 ns
Đọc thanh ghi	1 ns
Hoạt động ALU	2 ns
Truy cập bộ nhớ DL	2 ns
Ghi thanh ghi	1 ns
Tổng	8 ns

Tốc độ đồng hồ =

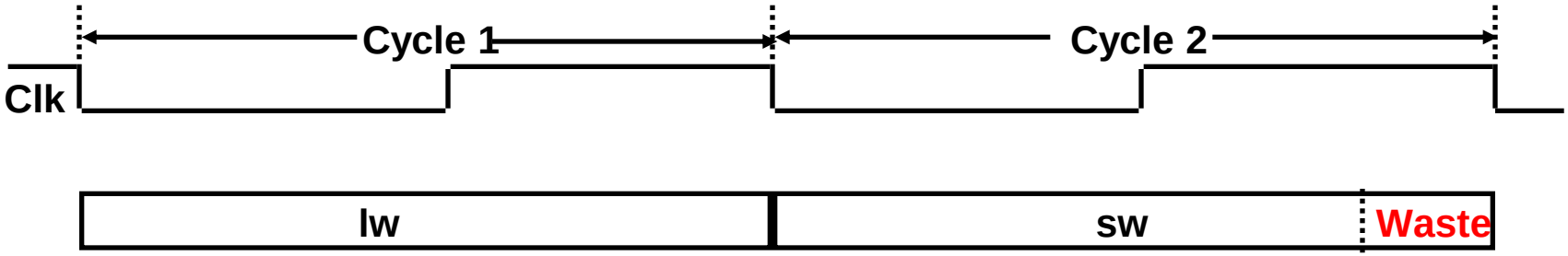
Các loại lệnh:

R-type	44%	6 ns
Load	24%	8 ns
Store	12%	7 ns
Branch	18%	5 ns
Jump	2%	3 ns
Trung bình		≅



Thiết kế đơn xung nhịp – Ưu nhược điểm

- Sử dụng chu kỳ đồng hồ không hiệu quả – chu kỳ đồng hồ được đặt theo lệnh chậm nhất
  - Vấn đề đặc biệt của các lệnh phức tạp như lệnh nhân dấu phẩy động



- Tốn diện tích thiết kế vì cần nhân đôi một số khối chức năng (VD. bộ cộng) vì chúng không thể được chia sẻ trong cùng 1 chu kỳ đồng hồ

Nhưng

- Đơn giản và dễ hiểu

So sánh đánh giá thiết kế đơn xung nhịp

Đồng hồ tốc độ 125 MHz là bình thường

Instruction access	2 ns
Register read	1 ns
ALU operation	2 ns
Data cache access	2 ns
Register write	1 ns
Total	8 ns

Single-cycle clock = 125 MHz

So sánh với các bộ xử lý trên thị trường:

Không tồi nếu so sánh độ trễ thực hiện 1 lệnh

Một bộ xử lý 2.5 GHz với 20 giai đoạn pipeline có độ trễ khoảng:

0.4 ns/cycle × 20 cycles = 8 ns

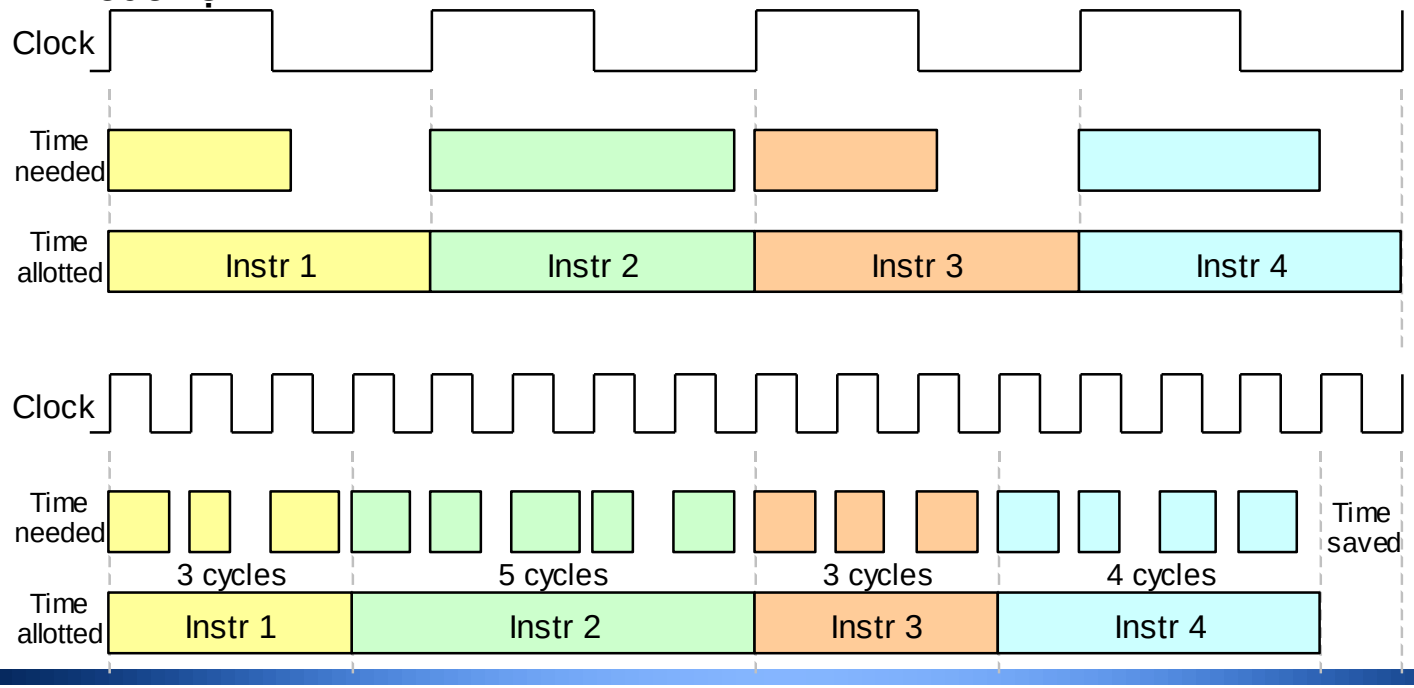
Lưu lượng của bộ xử lý có pipeline tốt hơn rất nhiều:

Tốt hơn tới 20 lần với các bộ xử lý phát hành đơn lệnh

Tốt hơn tới 100 lần với các bộ xử lý phát hành đa lệnh

Thiết kế đa xung nhịp

- Chia lệnh thành các pha thực hiện: IF, ID, EX, MEM, WB. Mỗi pha thực hiện trong 1 chu kỳ xung nhịp
- Các ưu điểm
  - Thời gian thực hiện (= số pha) của mỗi lệnh được điều chỉnh tùy thuộc độ phức tạp của lệnh
  - Các khối chức năng được chia sẻ giữa các pha khác nhau của lệnh do một khối chức năng cụ thể không cần trong toàn bộ các pha thực hiện của lệnh



Ví dụ 3.2 – Hiệu năng thiết kế đa xung nhịp

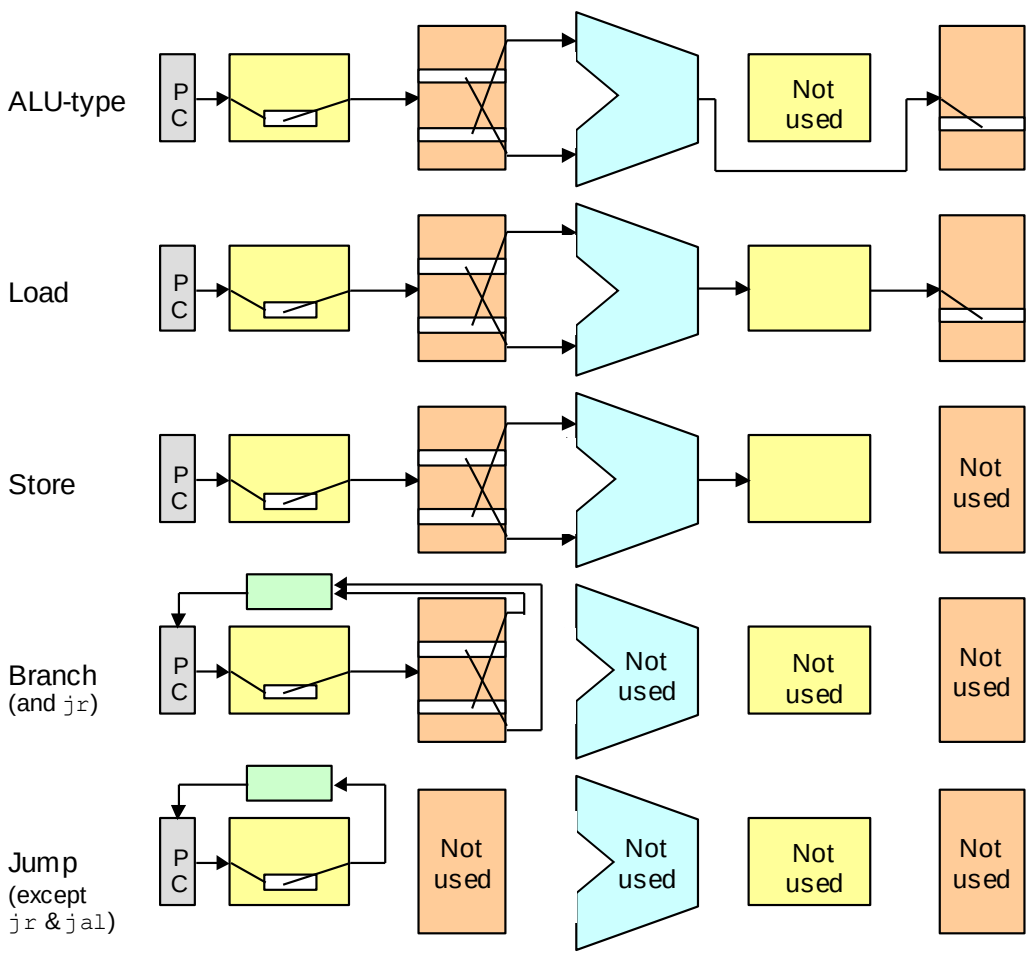
Các loại lệnh sử dụng số chu kỳ khác nhau

R-type	44%	4 cycles
Load	24%	5 cycles
Store	12%	4 cycles
Branch	18%	3 cycles
Jump	2%	3 cycles

Đóng góp vào số chu kỳ trung bình cần cho một lệnh:

R-type
Load
Store
Branch
Jump

CPI trung bình ≅



So sánh đánh giá thiết kế đa xung nhịp

Đồng hồ tốc độ 500MHz tốt hơn 125MHz của bộ xử lý một xung nhịp, nhưng vẫn là bình thường.

So sánh với các bộ xử lý trên thị trường:

Không tồi nếu so sánh độ trễ thực hiện 1 lệnh
Một bộ xử lý 2.5 GHz với 20 giai đoạn pipeline có độ trễ khoảng: 0.4 ns/cycle × 20 cycles = 8 ns

Lưu lượng của bộ xử lý có pipeline tốt hơn rất nhiều:

Tốt hơn tới 20 lần với các bộ xử lý phát hành đơn lệnh

Tốt hơn tới 100 lần với các bộ xử lý phát hành đa lệnh

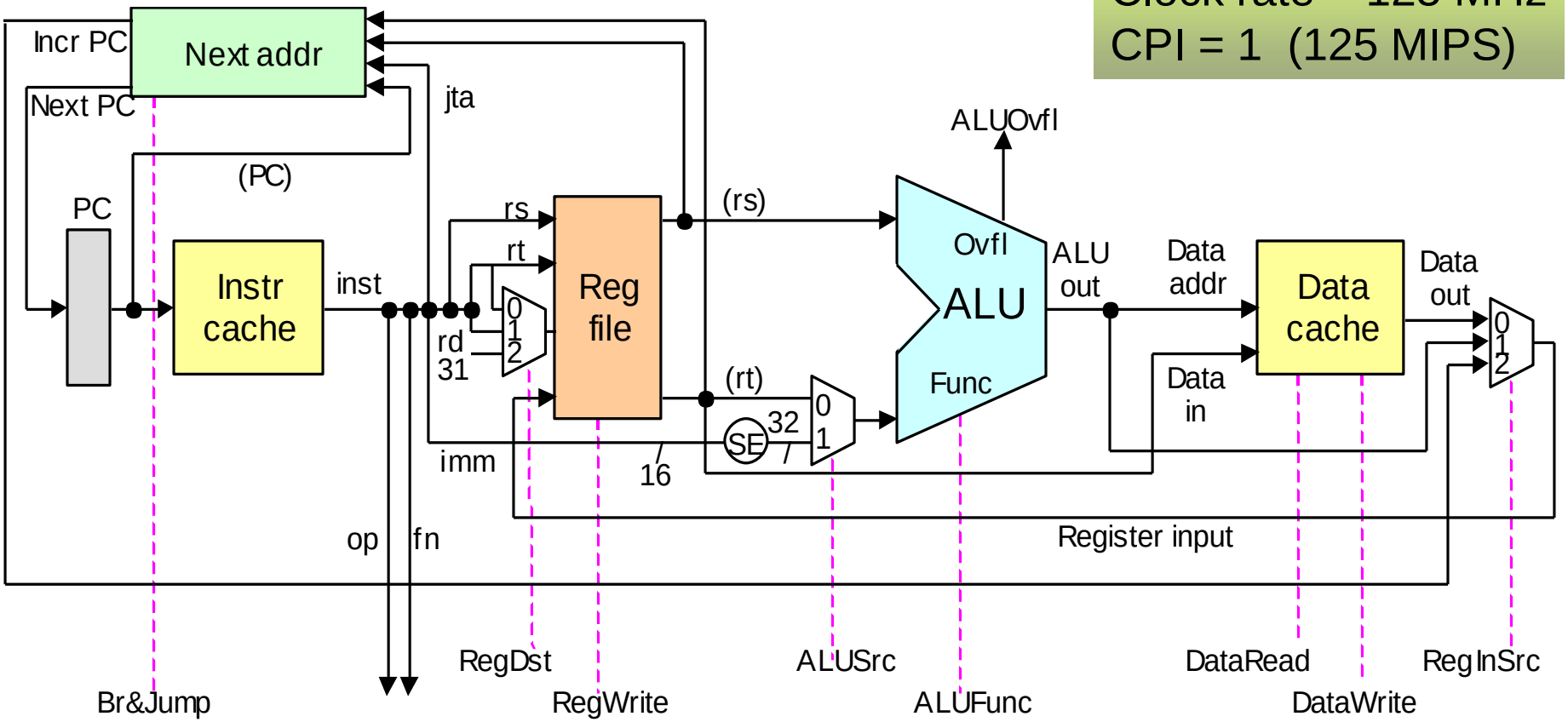
Cycle time = 2 ns
Clock rate = 500 MHz

R-type	44%	4 cycles
Load	24%	5 cycles
Store	12%	4 cycles
Branch	18%	3 cycles
Jump	2%	3 cycles

	Contribution to CPI
R-type	0.44×4 = 1.76
Load	0.24×5 = 1.20
Store	0.12×4 = 0.48
Branch	0.18×3 = 0.54
Jump	0.02×3 = 0.06
	Average CPI ≅ 4.04

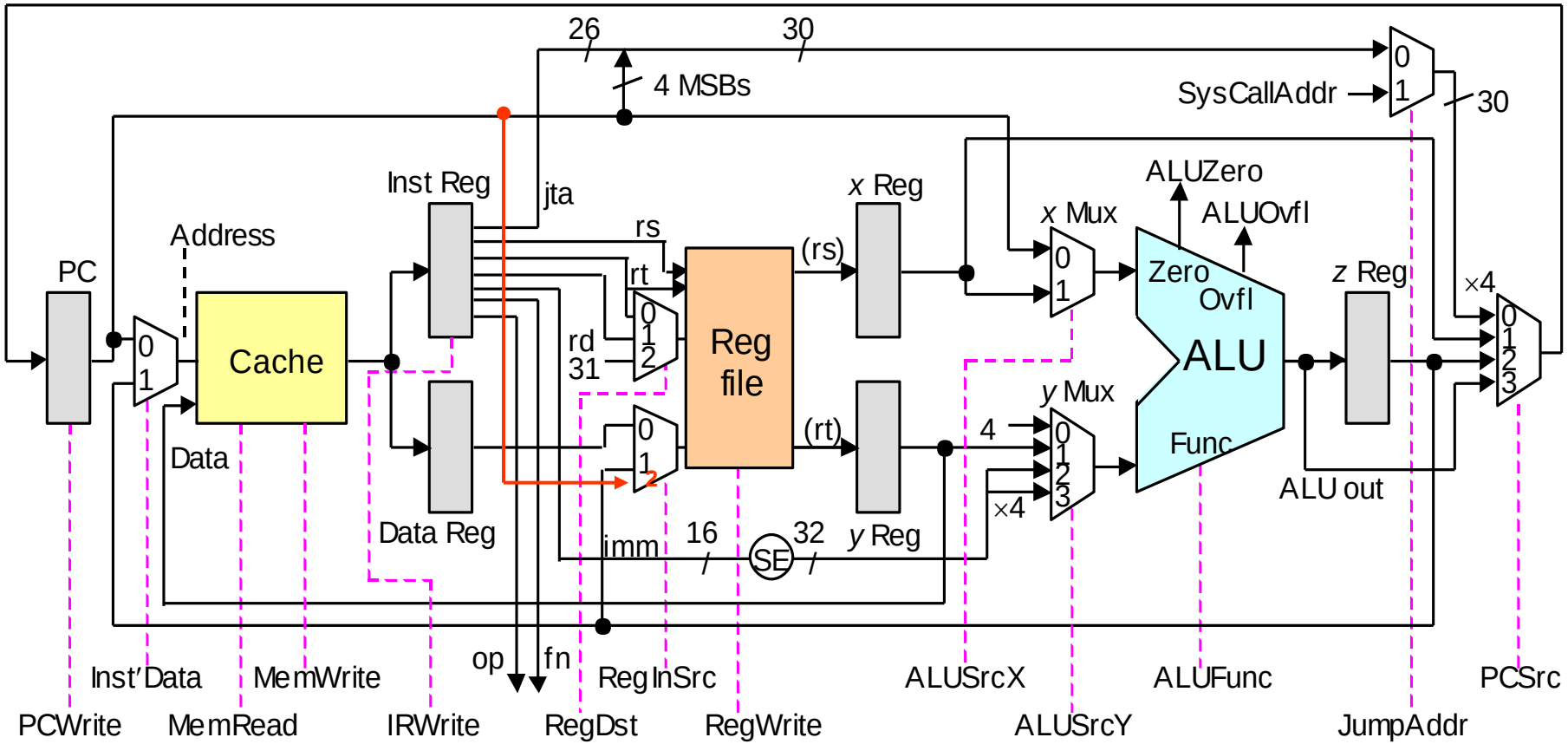
Thiết kế đơn xung nhịp

Clock rate = 125 MHz
CPI = 1 (125 MIPS)

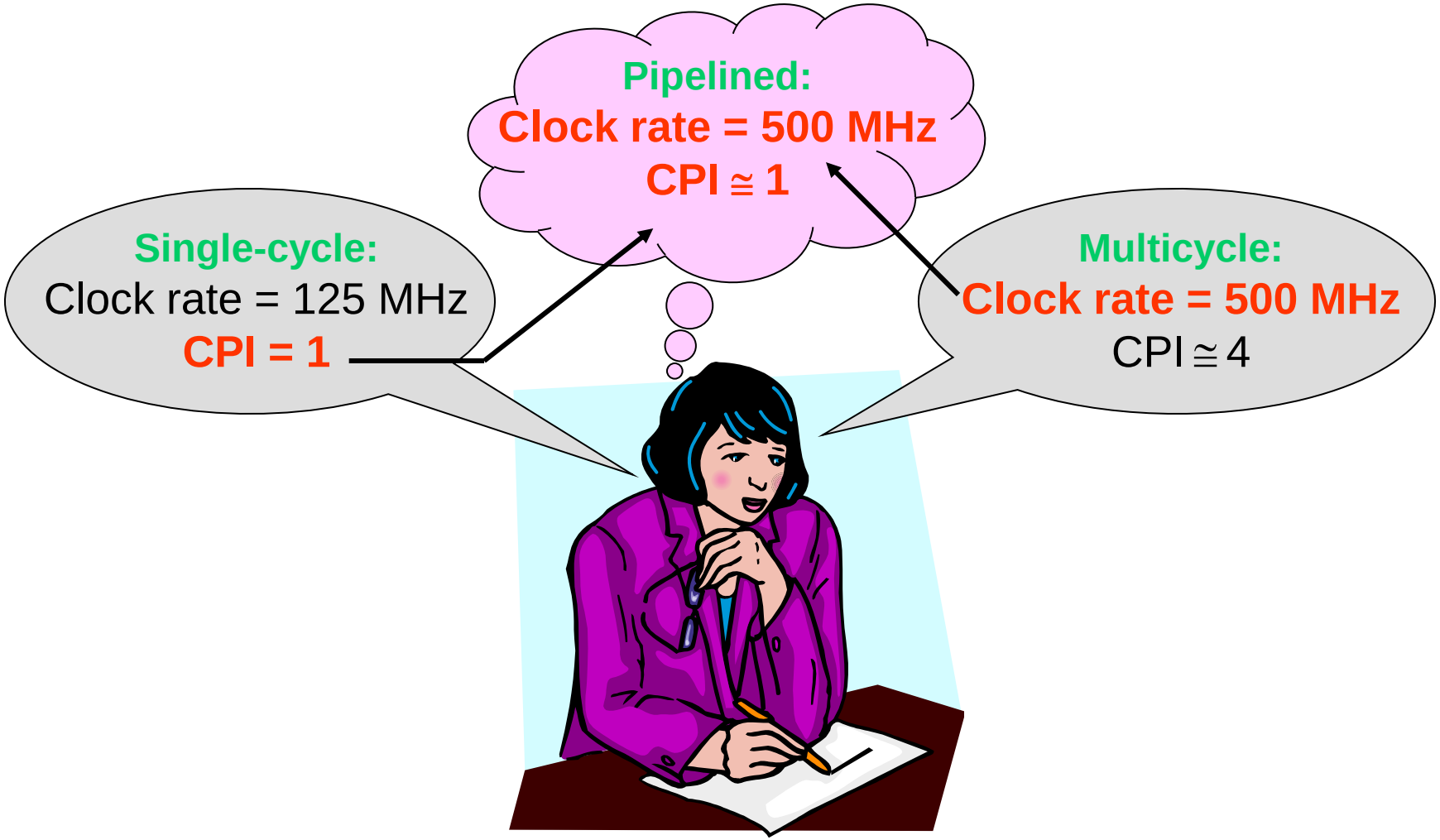


Thiết kế đa xung nhịp

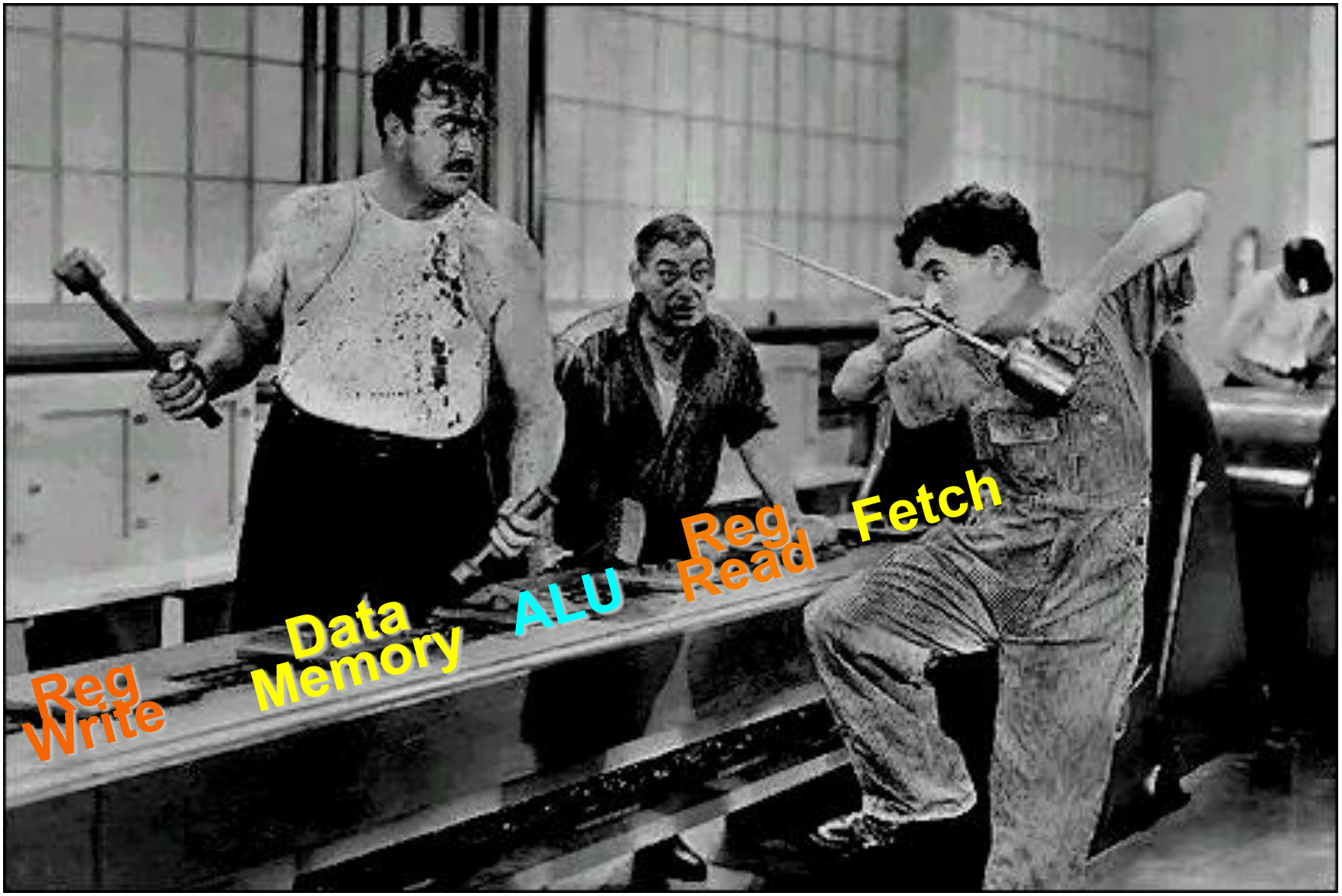
Clock rate = 500 MHz
CPI ≅ 4 (≅ 125 MIPS)



Đường ống (*Eng. pipeline*): Kết hợp ưu điểm



Dây chuyền sản xuất trong nhà máy



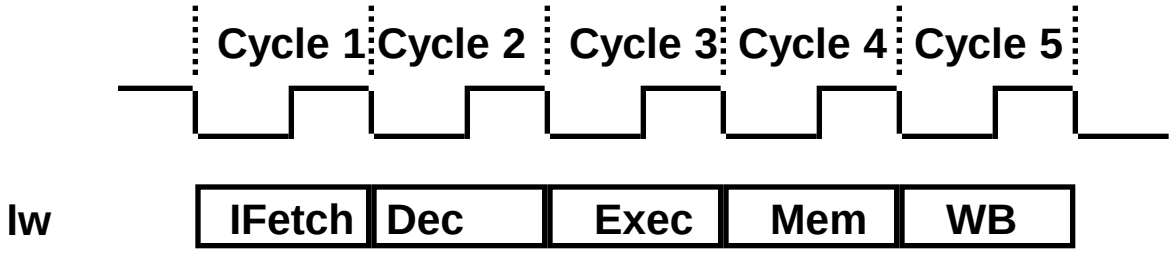
Tăng hiệu năng (tốc độ) bằng cách nào?

- Bắt đầu nạp và thực hiện lệnh tiếp theo trước khi lệnh hiện tại kết thúc:
  - Kỹ thuật đường ống – được áp dụng trong hầu hết các bộ xử lý hiện đại
  - Trong điều kiện lý tưởng với số lượng lệnh lớn, đường ống giúp tăng tốc độ bằng số giai đoạn đường ống. Đường ống 5 giai đoạn sẽ nhanh hơn gần 5 lần vì T_c tăng gấp 5.

$$T_{cpu} = I \times CPI \times T_c$$

- Nạp (và thực hiên) nhiều lệnh cùng một lúc
  - Xử lý superscalar

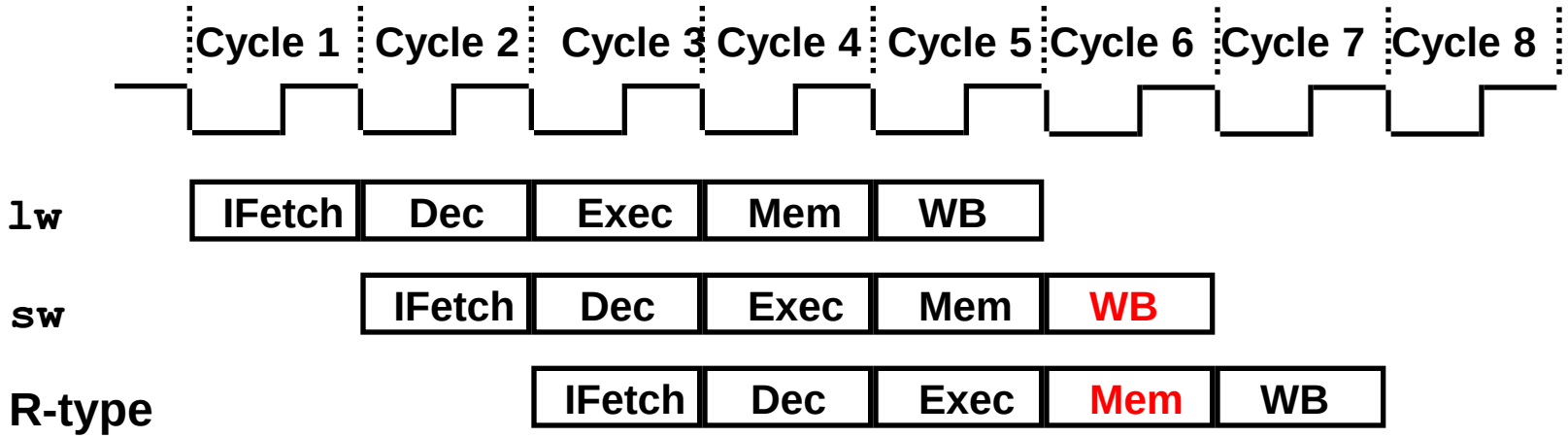
5 giai đoạn đường ống của lệnh lw



- IFetch: Nạp lệnh và cập nhập giá trị PC
- IDec: Đọc thanh ghi và giải mã lệnh
- EXec: Thực hiện lệnh R; tính địa chỉ bộ nhớ
- MEM: Đọc/ghi bộ nhớ dữ liệu
- WB: Ghi kết quả vào tệp thanh ghi

Đường ống trong MIPS

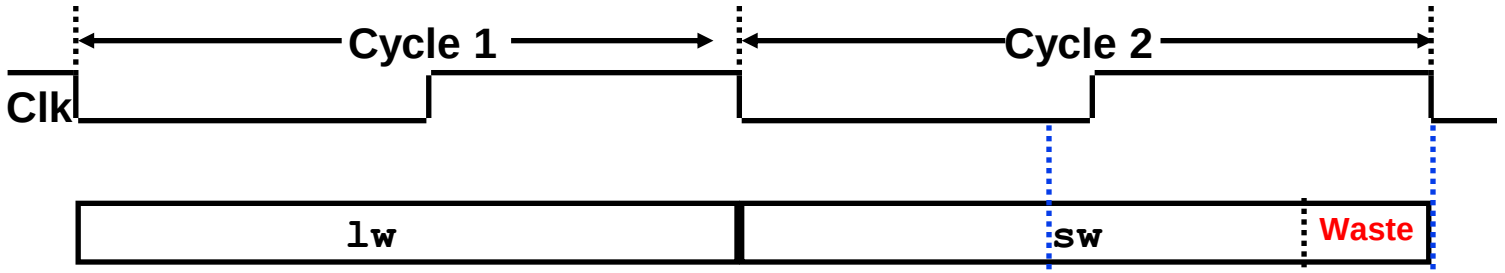
- Bắt đầu lệnh tiếp theo trước khi lệnh hiện tại kết thúc
  - cải thiện thông lượng – tổng số công việc hoàn thành trong 1 khoảng thời gian
  - độ trễ lệnh (thời gian thực hiện, thời gian đáp ứng – thời gian từ lúc bắt đầu đến lúc kết thúc lệnh) *không* được giảm



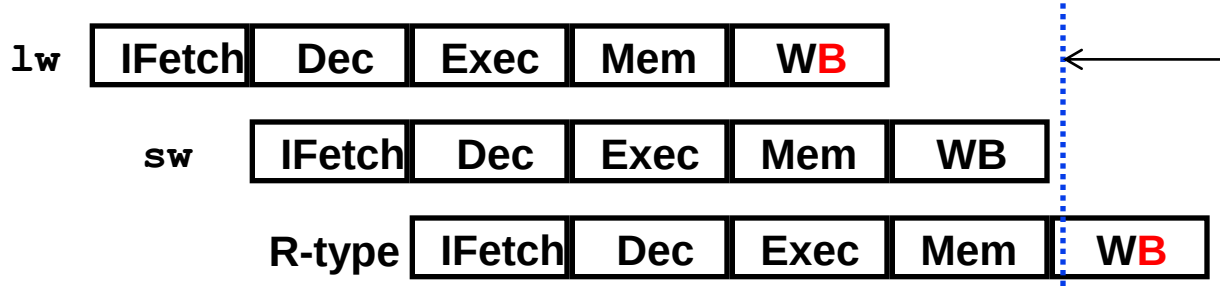
  - chu kỳ đồng hồ (thời gian 1 giai đoạn đường ống) quyết định bởi giai đoạn chậm nhất
  - một số giai đoạn không dùng toàn bộ chu kỳ đồng hồ (VD., WB)
  - một số lệnh, có các giai đoạn là chu kỳ lãng phí (nghĩa là, không thực hiện gì trong chu kỳ đó với lệnh đó)

Ví dụ 3.3 – Pipeline vs. Đơn xung nhịp

Triển khai đơn xung nhịp (Tc = 800 ps):



Triển khai pipeline (Tc = 200 ps):



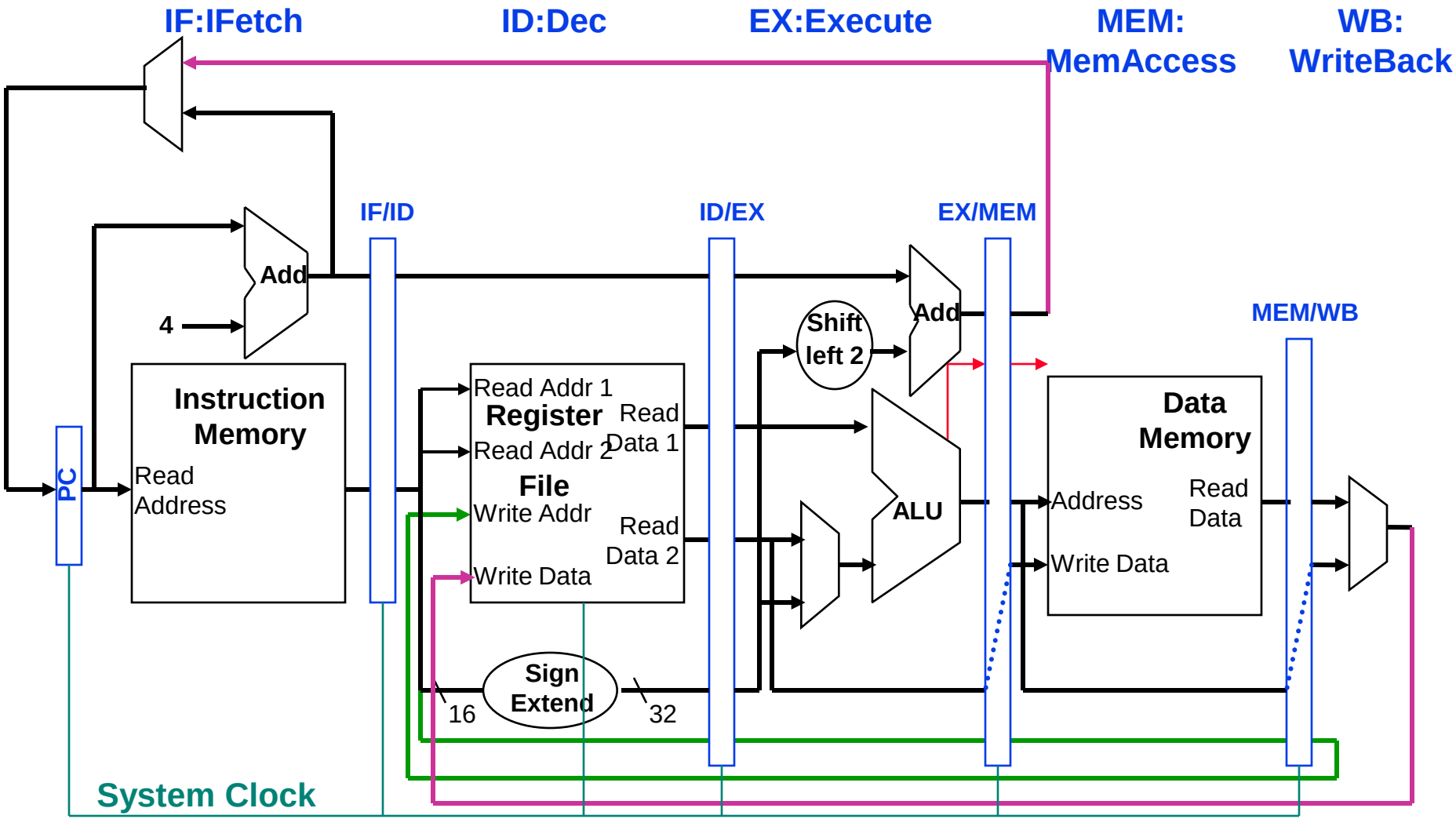
- Để hoàn thành 1 lệnh trong trường hợp pipeline cần 1000 ps (So với 800 ps trong trường hợp đơn xung nhịp). Tại sao?
- Để thực hiên 1.000.000 lệnh "*adds*" cần thời gian bao lâu?

Kỹ thuật đường ống cho MIPS(RISC) ISA

- Dễ triển khai:
  - Các lệnh có cùng độ dài ( )
    - ➔ Có thể nạp lệnh trong giai đoạn 1st và giải mã lệnh trong giai đoạn 2nd
  - Ít định dạng lệnh ( ). Các định dạng lệnh có tính đối xứng
    - Có thể đọc thanh ghi ở giai đoạn 2nd
  - Chỉ truy cập bộ nhớ bằng lệnh lw và sw
    - Có thể tính địa chỉ bộ nhớ ở giai đoạn EX ( )
  - Mỗi lệnh chỉ ghi lớn nhất 1 kết quả (làm thay đổi trạng thái máy) ở 2 giai đoạn cuối (MEM or WB)
  - Toán hạng được sắp xếp trong bộ nhớ sao cho 1 lệnh dịch chuyển dữ liệu chỉ cần 1 lần truy cập bộ nhớ.

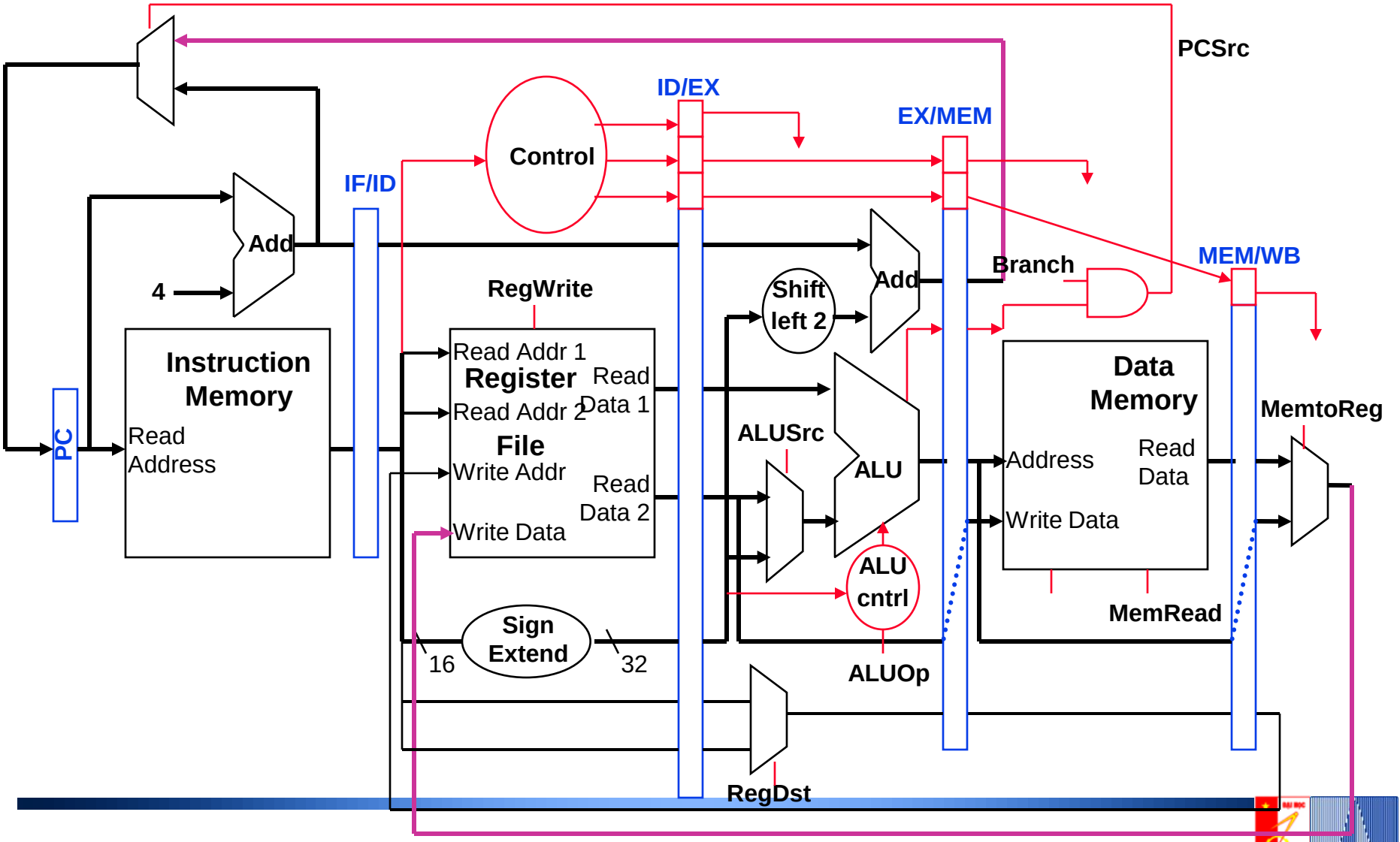
Đường dữ liệu MIPS pipeline

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

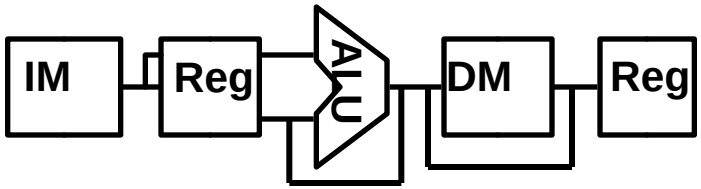


Điều khiển MIPS pipeline

- Các tín hiệu điều khiển được xác định trong giai đoạn giải mã và được lưu trong các thanh ghi trạng thái giữa các giai đoạn pipeline

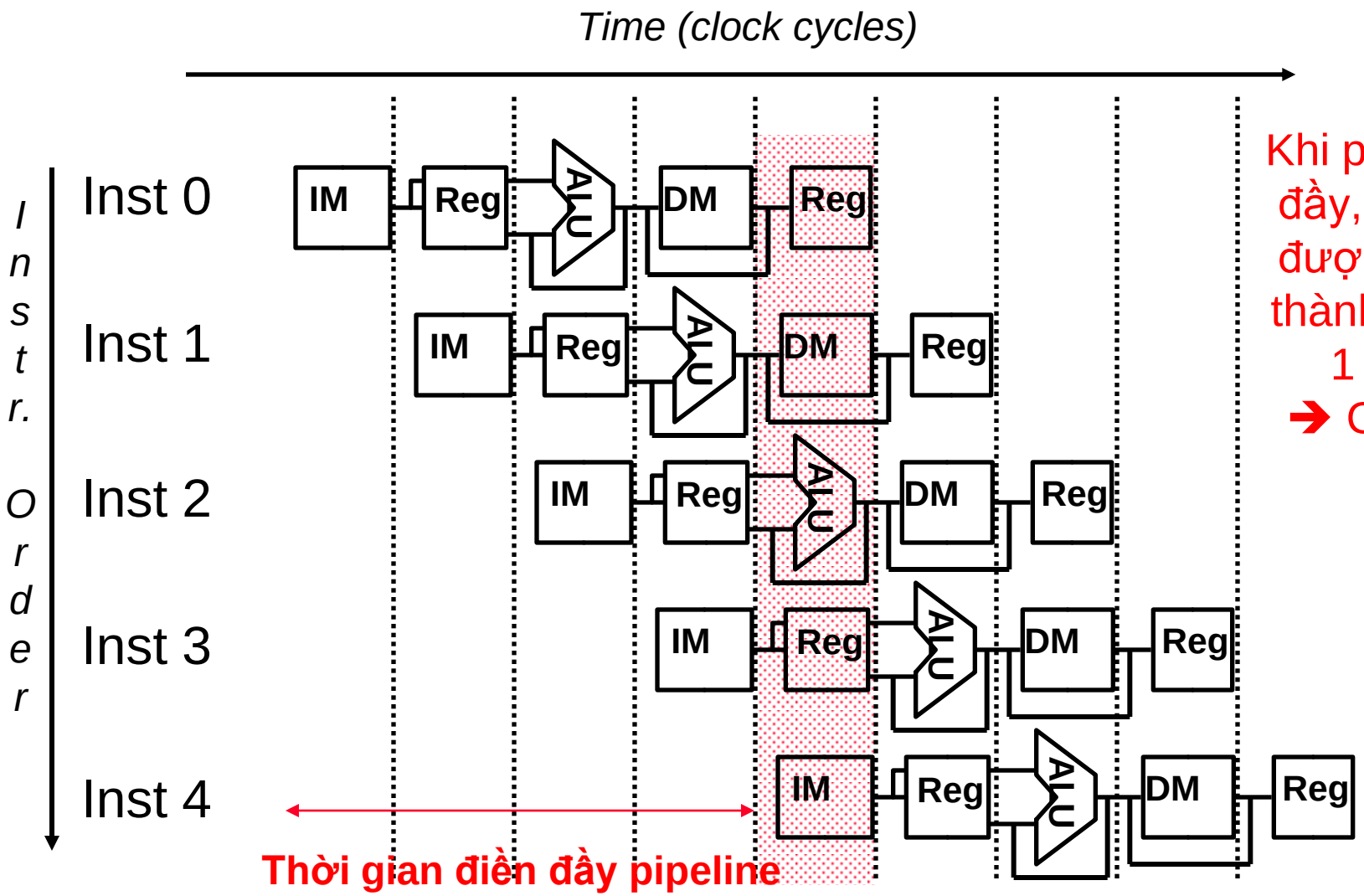


Mô tả hoạt động pipeline



- **Đo hiệu năng:** Cần bao nhiêu chu kỳ để thực hiện đoạn mã?
- **Phân tích hoạt động:** ALU làm gì ở chu kỳ 4?
- **Cải tiến:** Có xảy ra **hazard** không? Tại sao? Dùng cách nào để khắc phục?

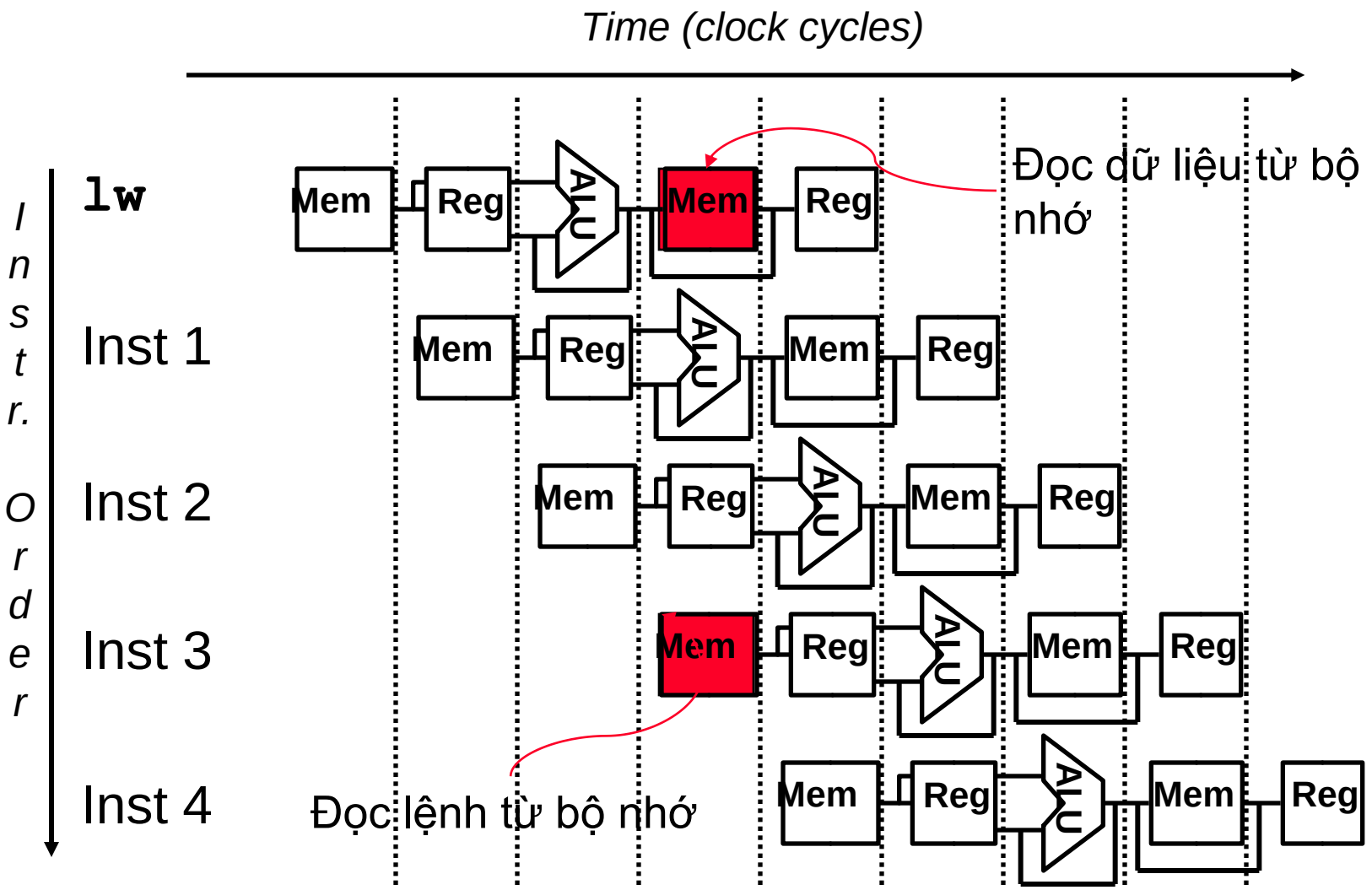
Hiệu năng pipeline



Xung đột Pipeline

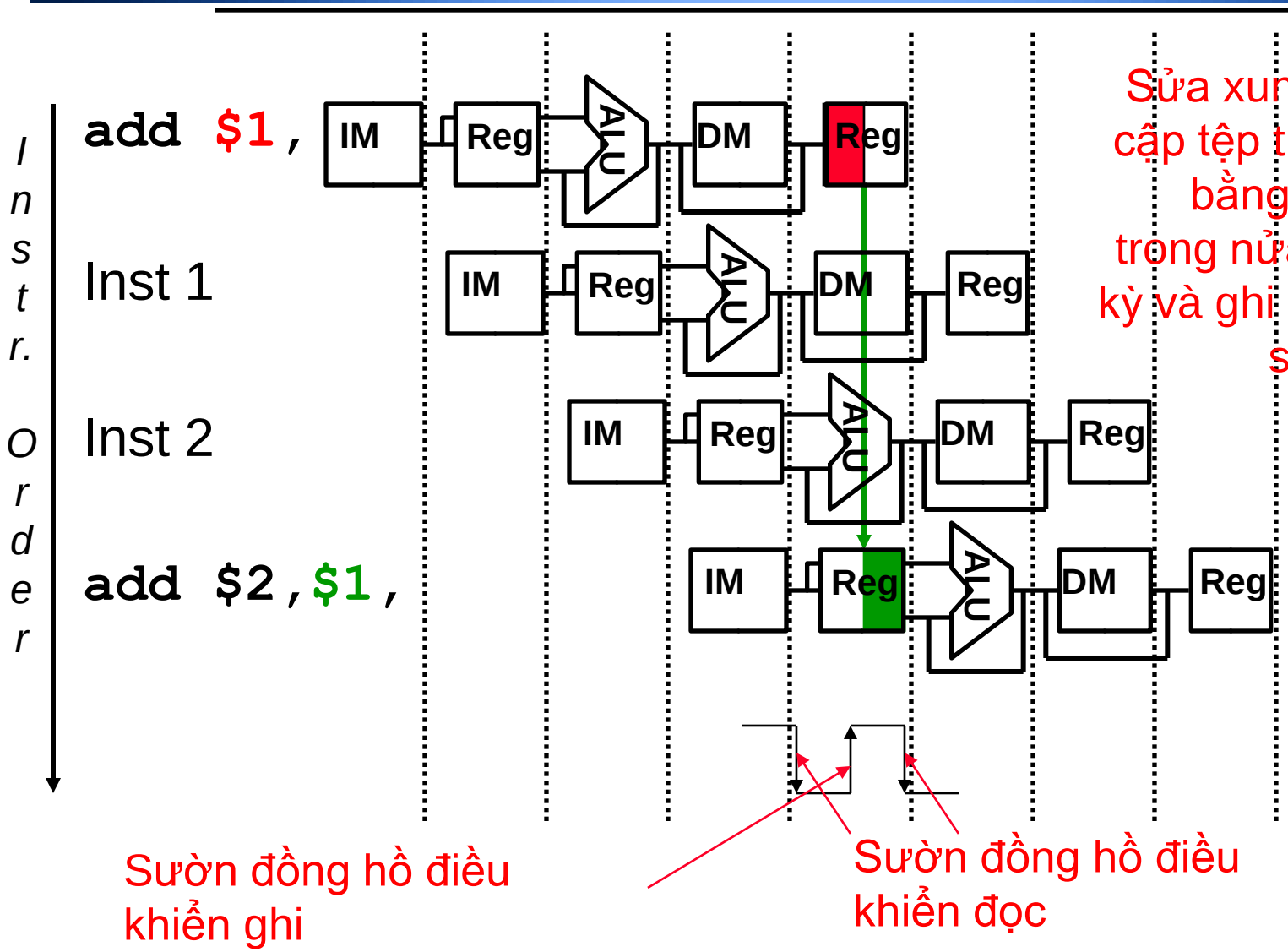
- Xung đột cấu trúc: yêu cầu sử dụng cùng một tài nguyên cho 2 lệnh khác nhau tại cùng 1 thời điểm
- Xung đột dữ liệu: yêu cầu sử dụng dữ liệu trước khi nó sẵn sàng
  - Các toán hạng nguồn của 1 lệnh được tạo ra bởi lệnh phía trước vẫn đang nằm trong pipeline
- Xung đột điều khiển: yêu cầu quyết định 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



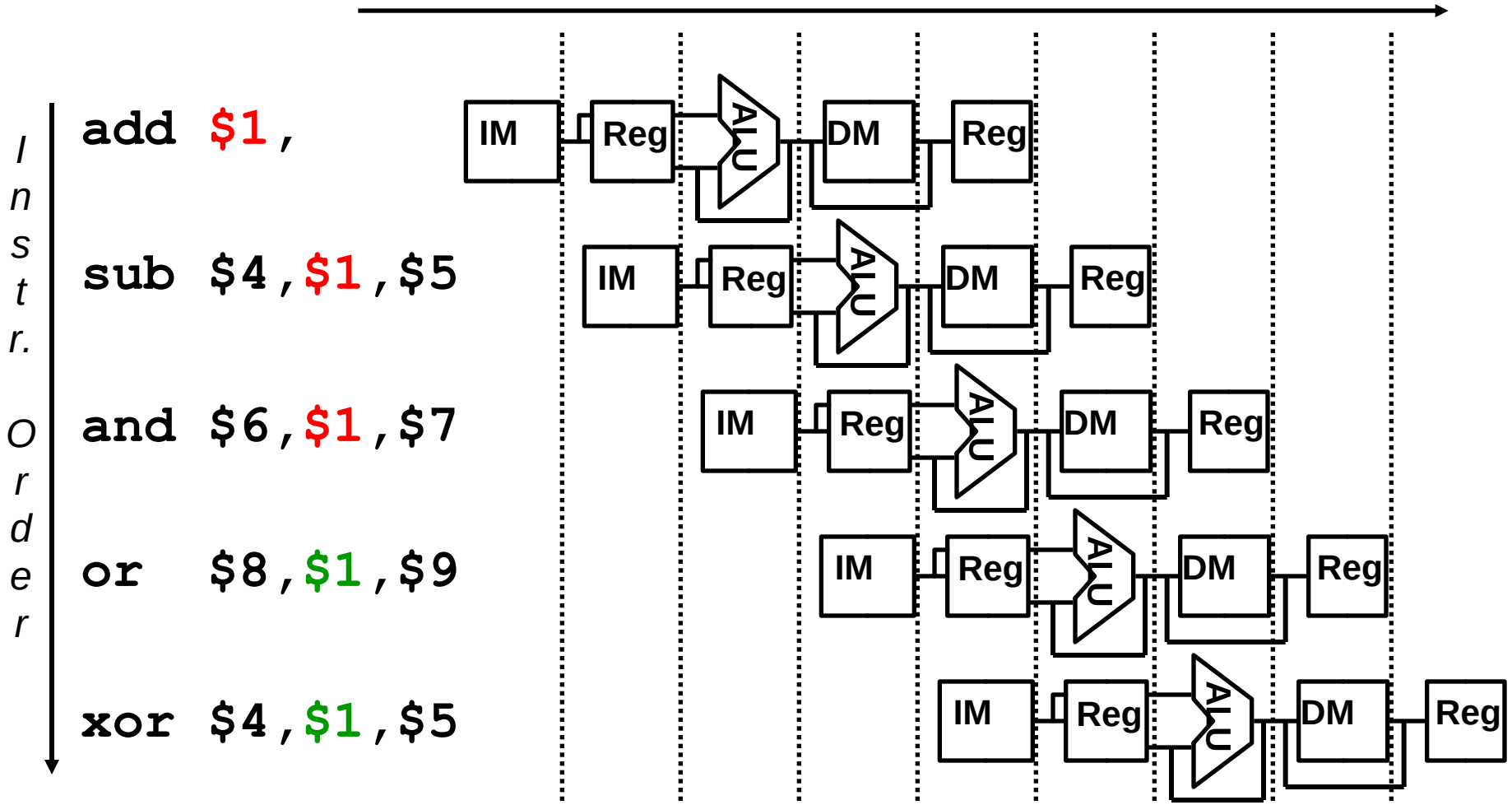
- Sửa: Bộ nhớ dữ liệu và lệnh riêng rẽ(I$ and D$)

Xung đột cấu trúc khi truy cập tệp thanh ghi



Sử dụng thanh ghi: Xung đột dữ liệu

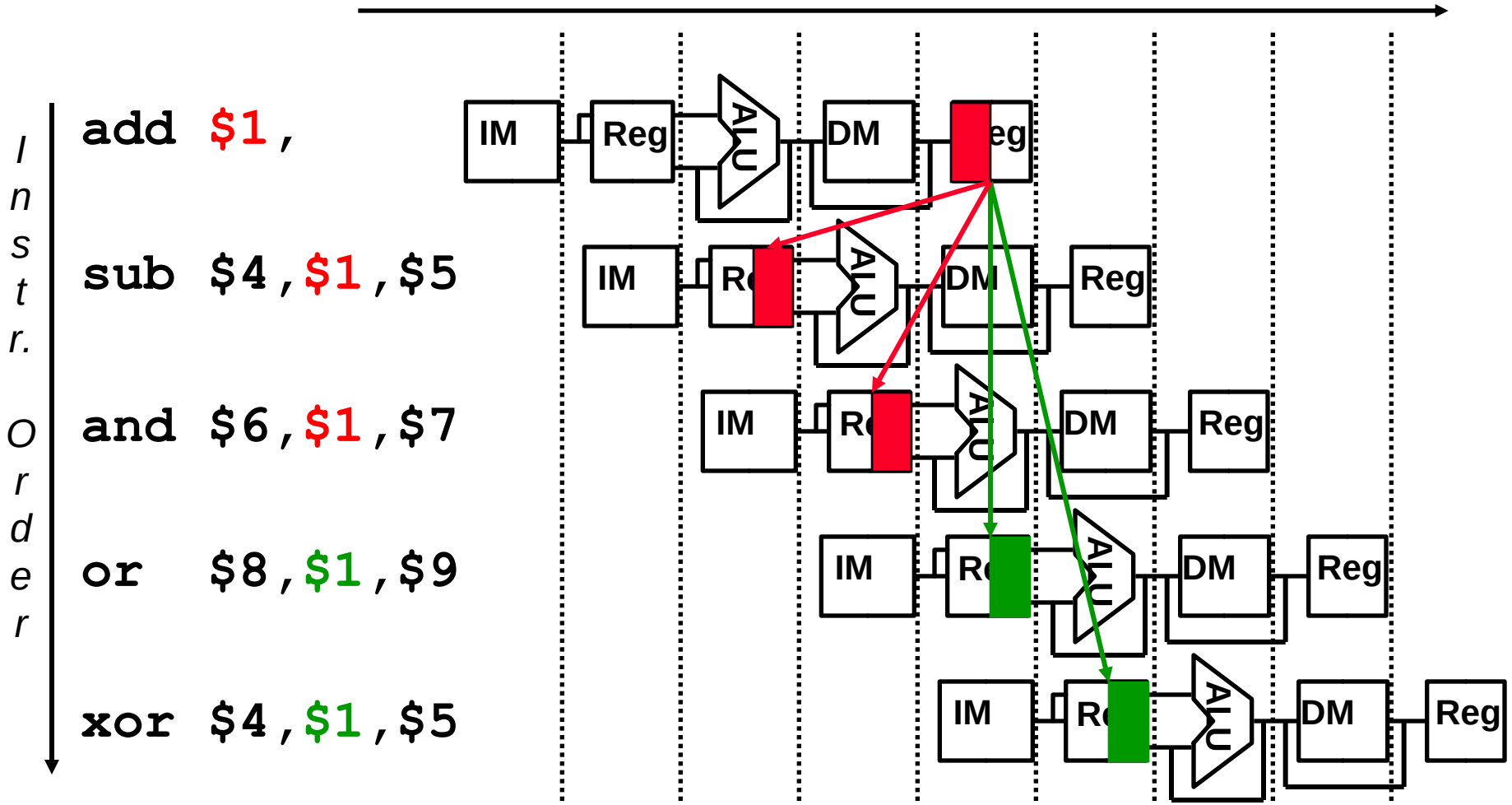
- Phụ thuộc dữ liệu ngược theo thời gian gây ra xung đột



- Xung đột đọc trước khi ghi (Read before write)

Sử dụng thanh ghi: Xung đột dữ liệu

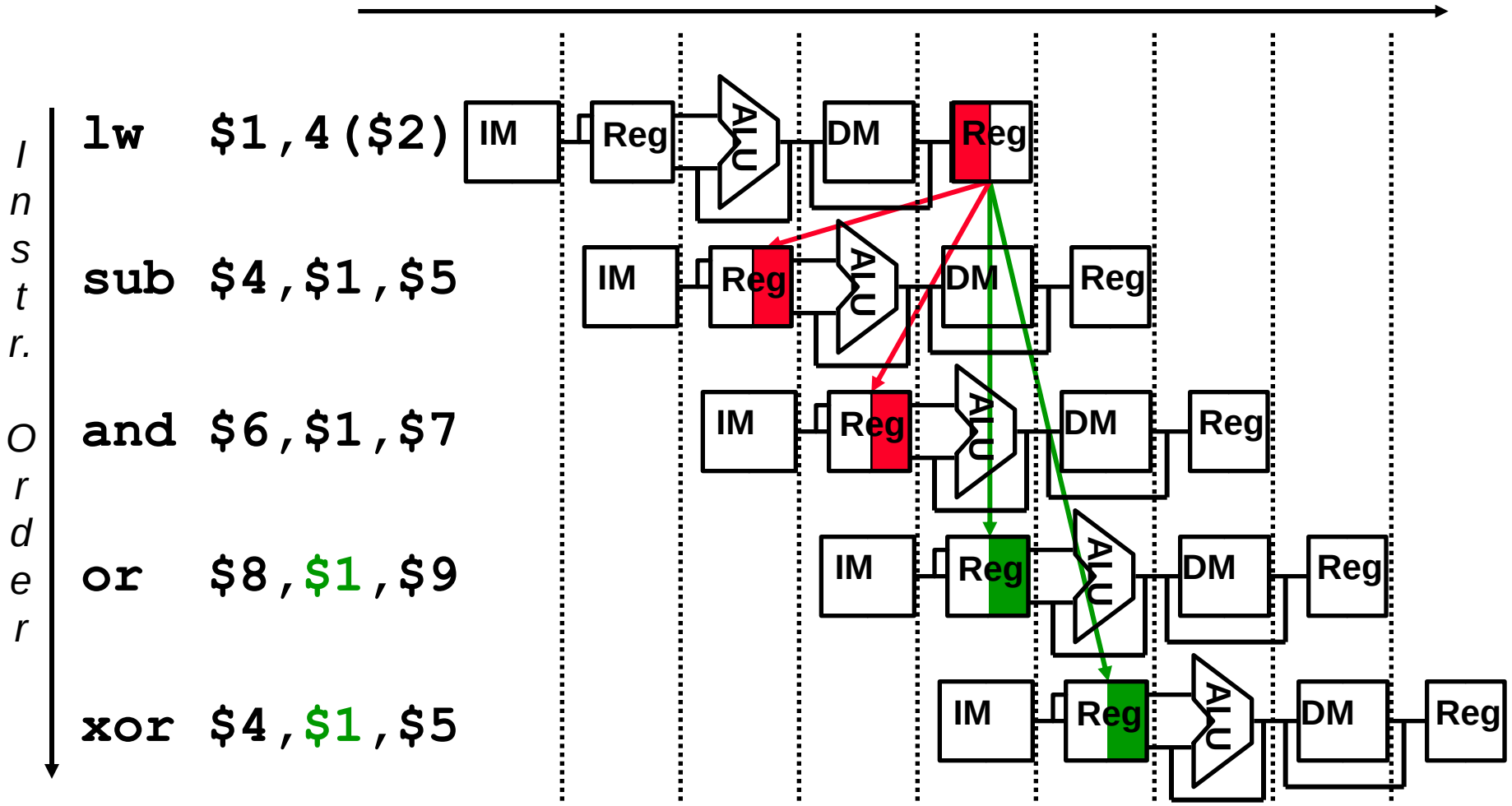
- Phụ thuộc dữ liệu ngược theo thời gian gây ra xung đột



- Xung đột đọc trước khi ghi (Read before write)

Đọc từ bộ nhớ: Gây xung đột dữ liệu

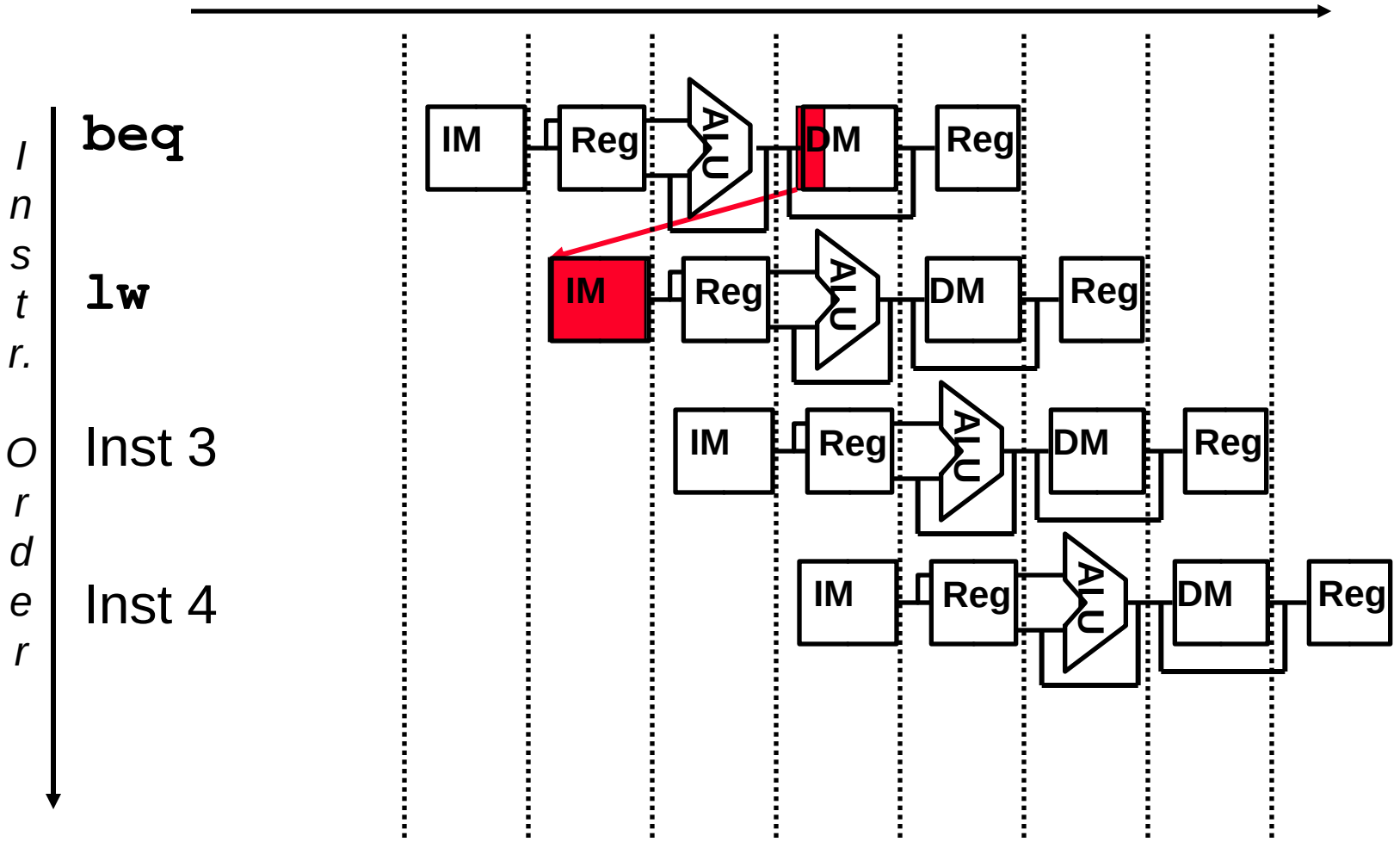
- Dependencies backward in time cause hazards



- Load-use data hazard

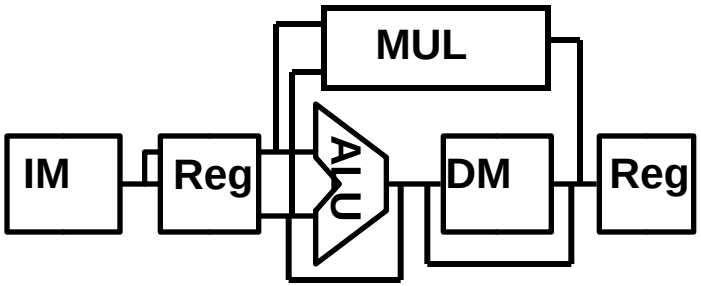
Xung đột điều khiển

- Dependencies backward in time cause hazards

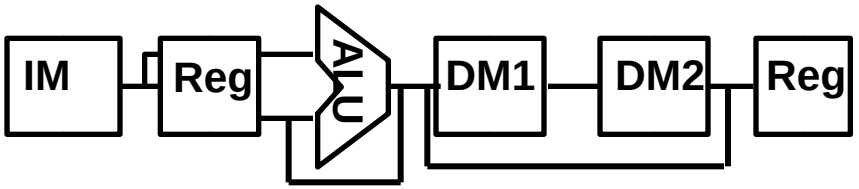


Các cấu trúc pipeline khác

- Phép toán nhân (chậm) gấp 2 lần?
  - Làm đồng hồ chậm đi 2 lần hoặc …
  - thực hiện trong 2 chu kỳ đồng hồ (vì không cần sử dụng giai đoạn DM)



- Truy cập bộ nhớ dữ liệu chậm hơn bộ nhớ lệnh 2 lần?
  - Làm đồng hồ chậm đi 2 lần hoặc …
  - thực hiện việc đọc trong 2 chu kỳ (và giữ nguyên chu kỳ đồng hồ)

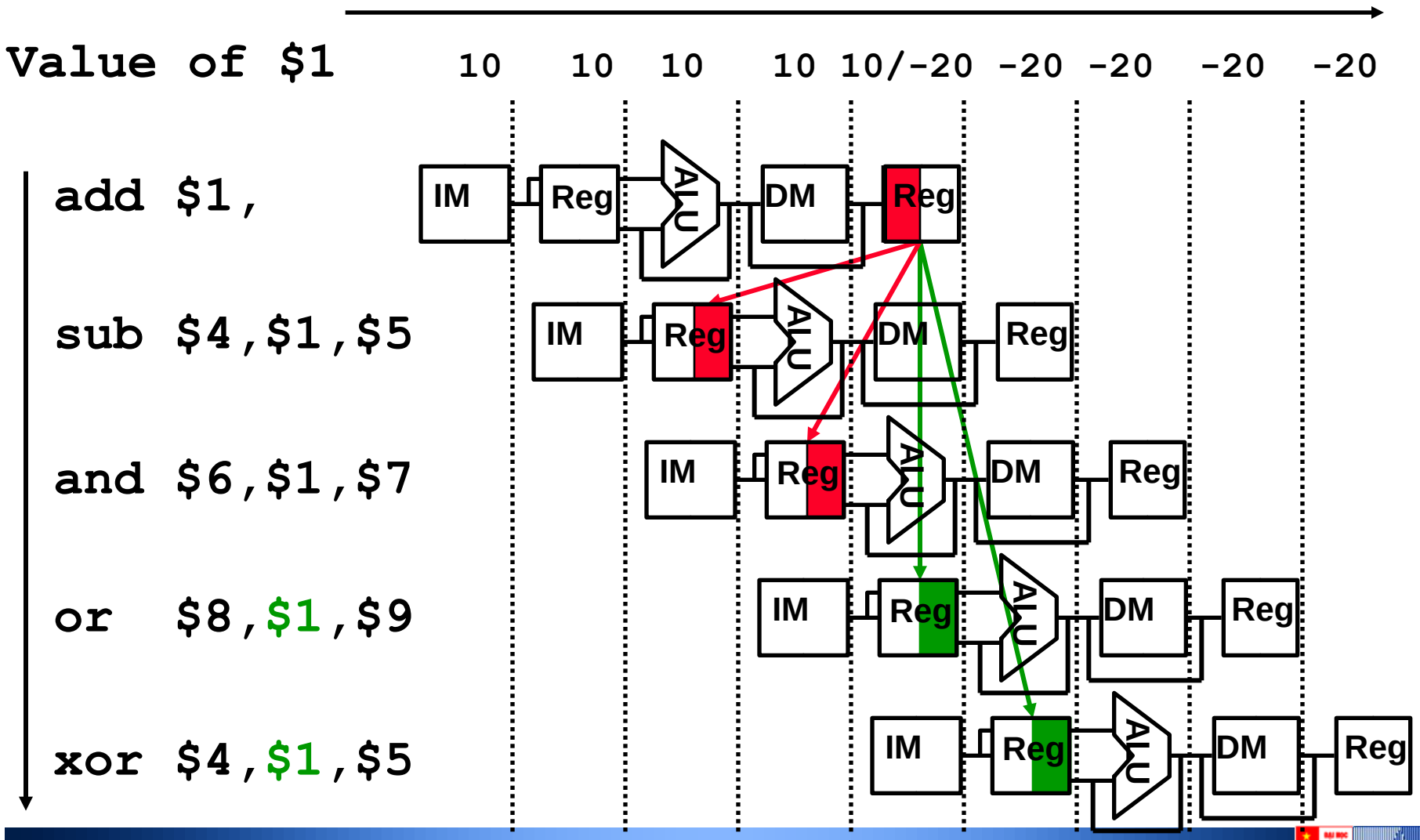


Tóm tắt 1

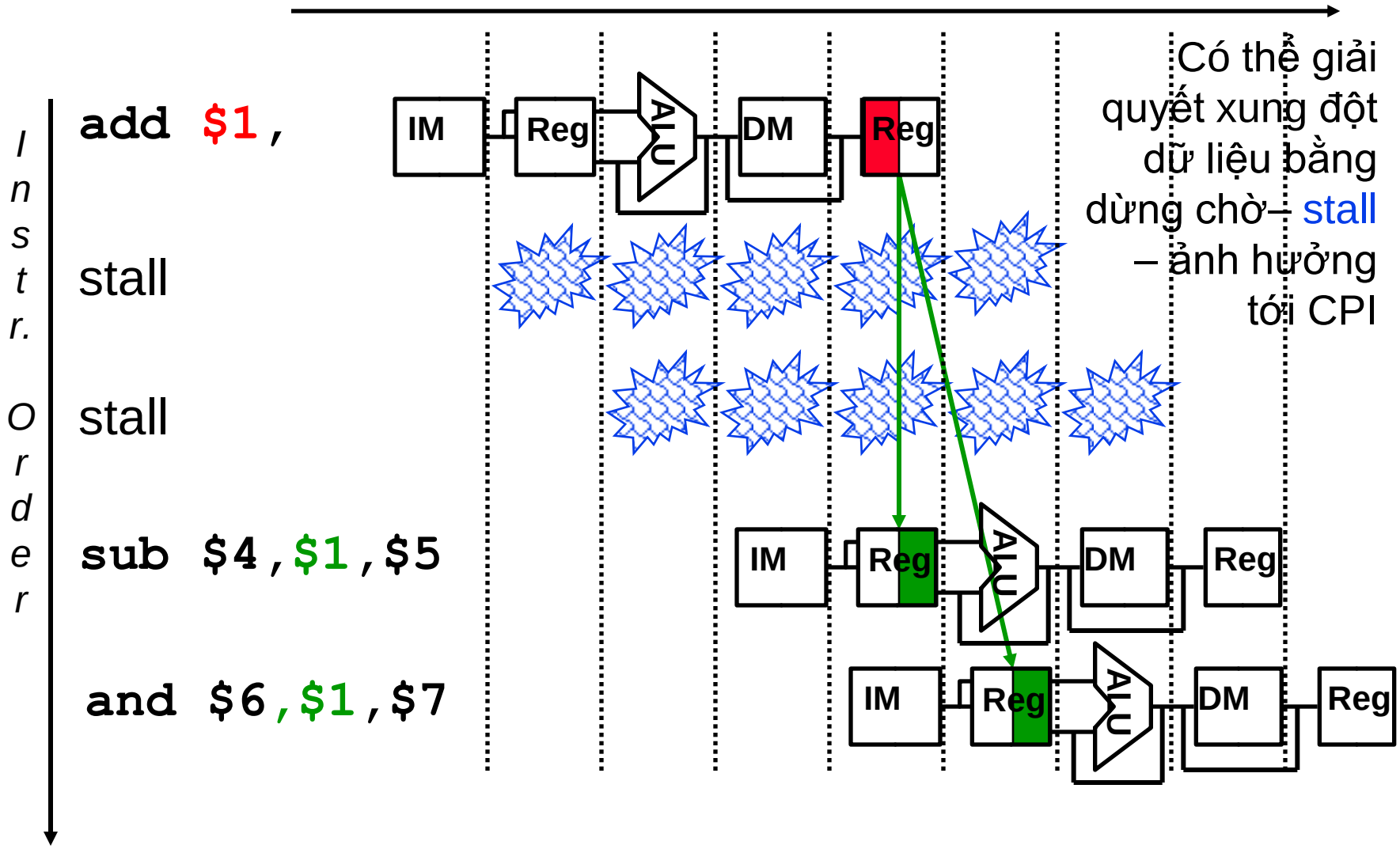
- Các bộ xử lý hiện đại đều dùng kỹ thuật pipeline
- Pipelining không làm giảm độ trễ của 1 nhiệm vụ đơn lẻ, nó giúp tăng thông lượng của toàn bộ
- Tăng tốc tiềm năng: CPI = 1 và đồng hồ nhanh, Tc nhỏ
- Tốc độ đồng hồ bị hạn chế bởi giai đoạn pipeline chậm nhất
  - Các giai đoạn pipeline không cân bằng 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)

Xung đột dữ liệu

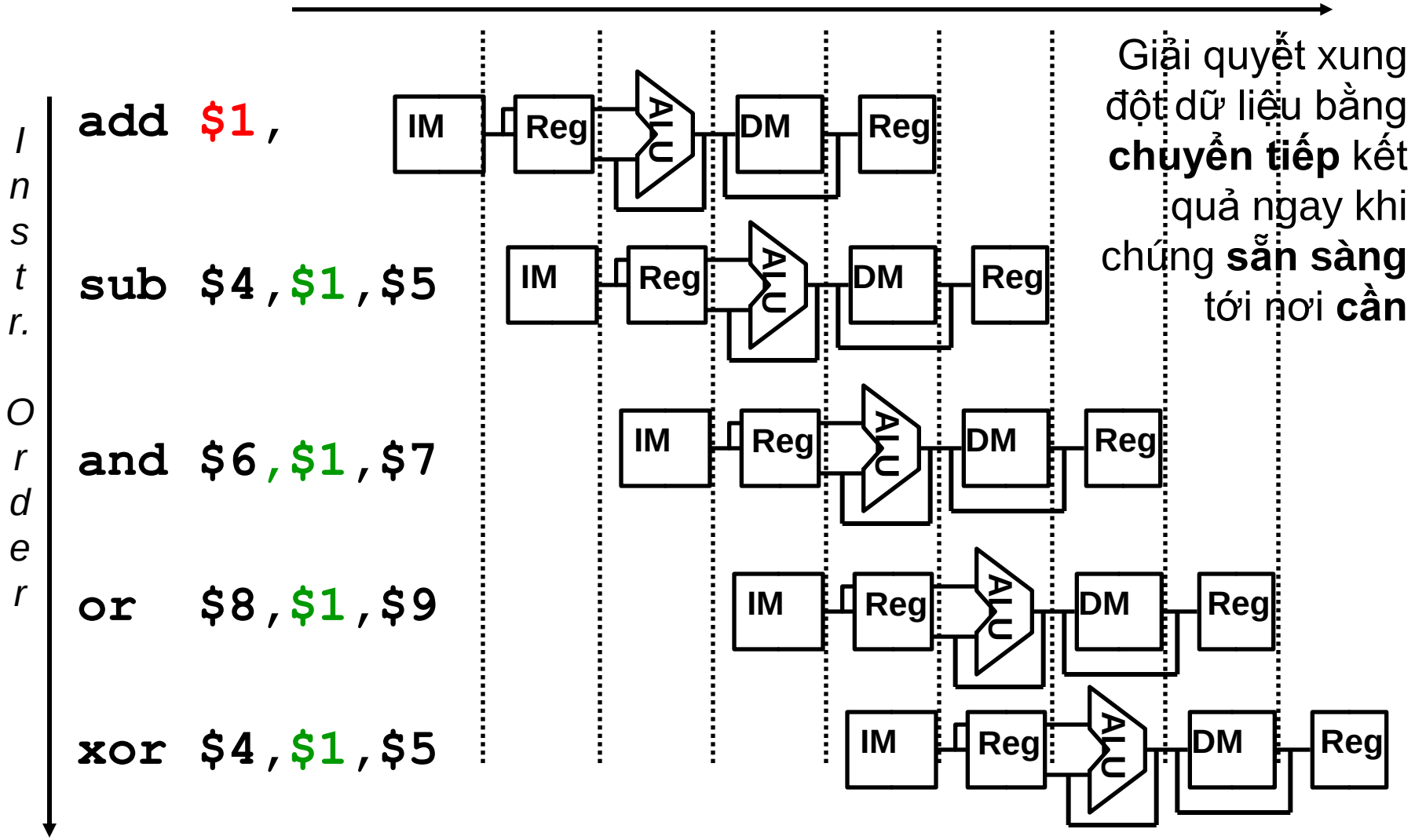
- Xung đột dữ liệu đọc trước ghi



Giải quyết xung đột: Tạm dừng



Giải quyết xung đột: Chuyển tiếp dữ liệu

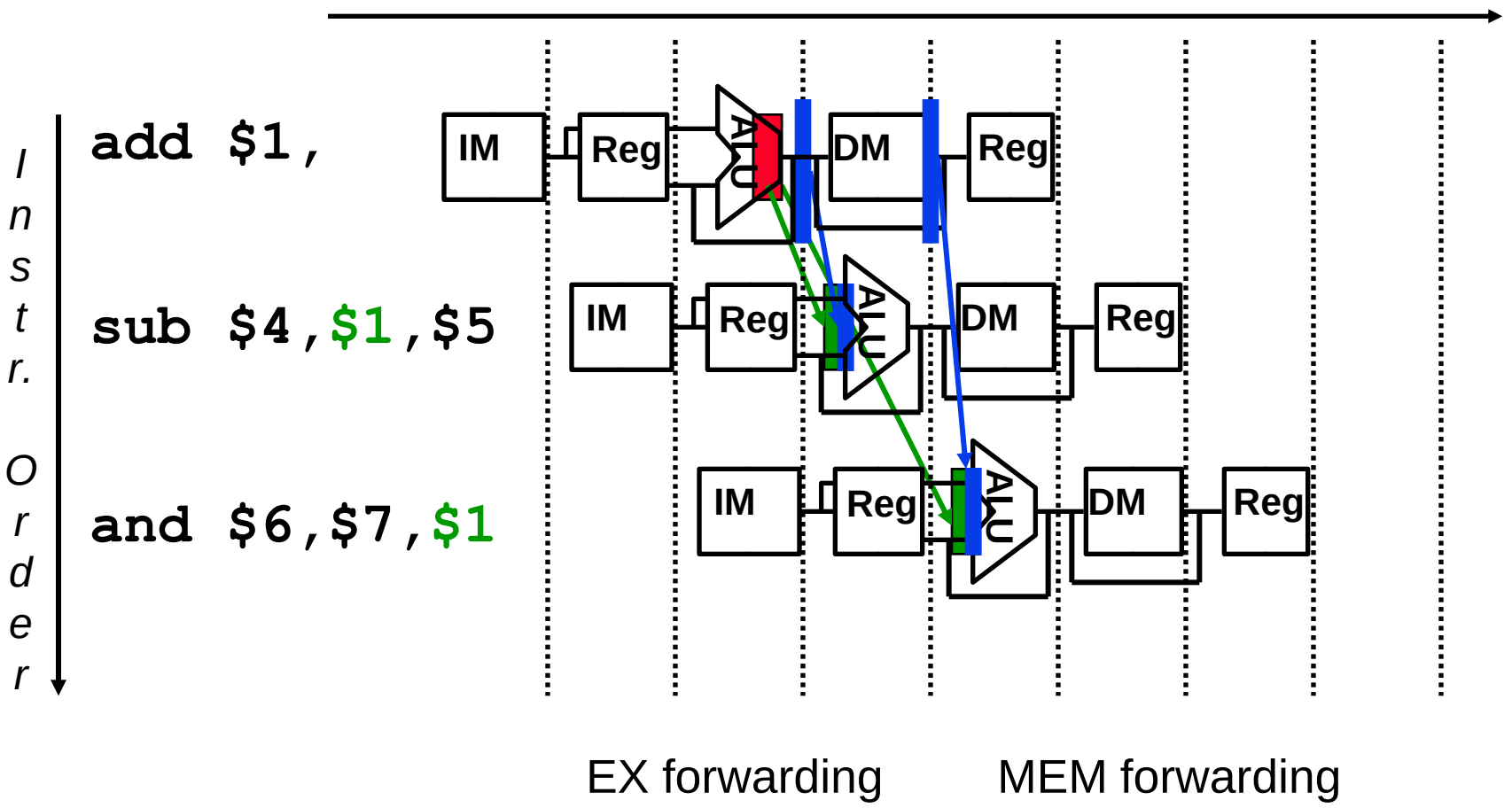


Giải quyết xung đột dữ liệu bằng **chuyển tiếp** kết quả ngay khi chúng **sẵn sàng** tới nơi **cần**

Chuyển tiếp dữ liệu

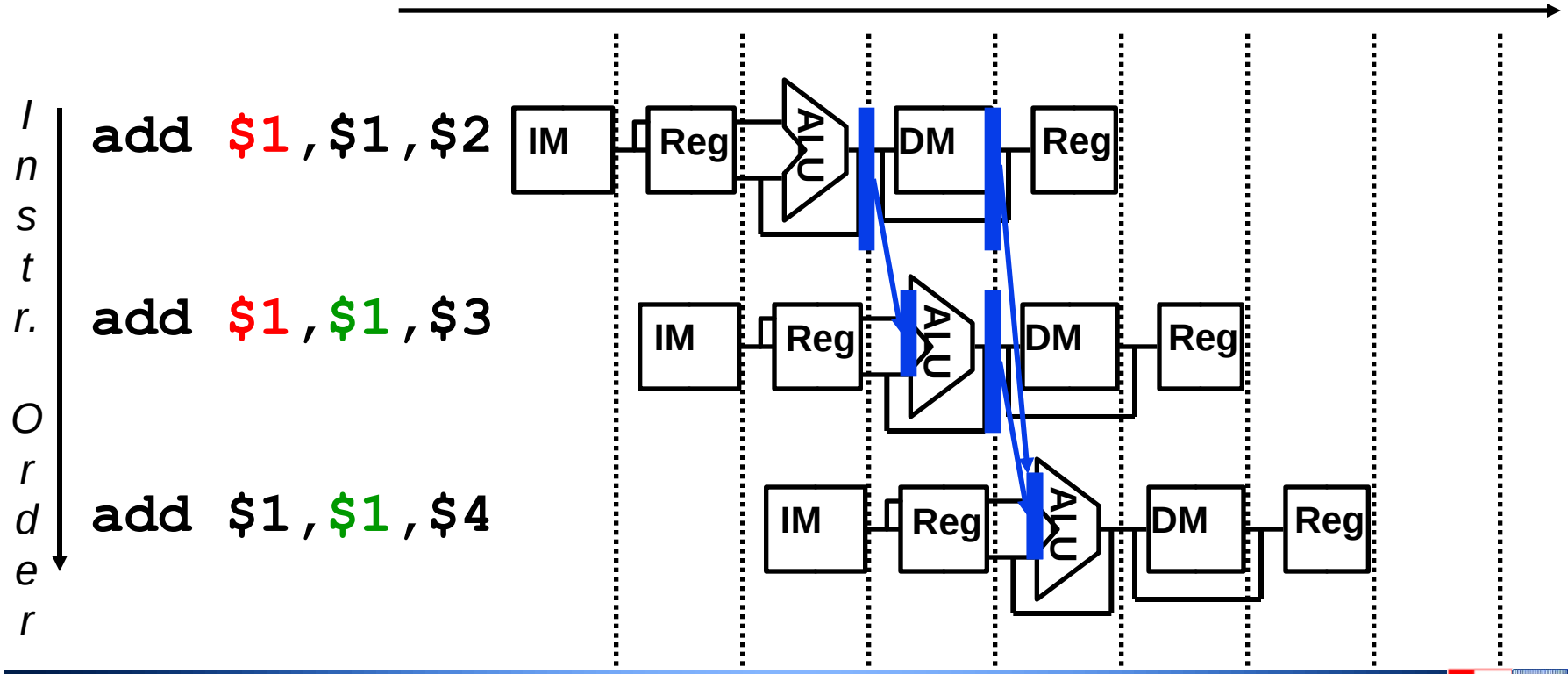
- Lấy kết quả ở thời điểm nó xuất hiện sớm nhất trong bất kỳ thanh ghi pipeline nào, và chuyển tiếp nó đến khối chức năng (VD. ALU) mà cần kết quả tại chu kỳ đồng hồ đó
- Với khối chức năng ALU: đầu vào có thể từ bất kỳ thanh ghi pipeline nào chứ không cần từ ID/EX bằng cách
  - thêm bộ chọn vào trước 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

Minh họa triển khai chuyển tiếp

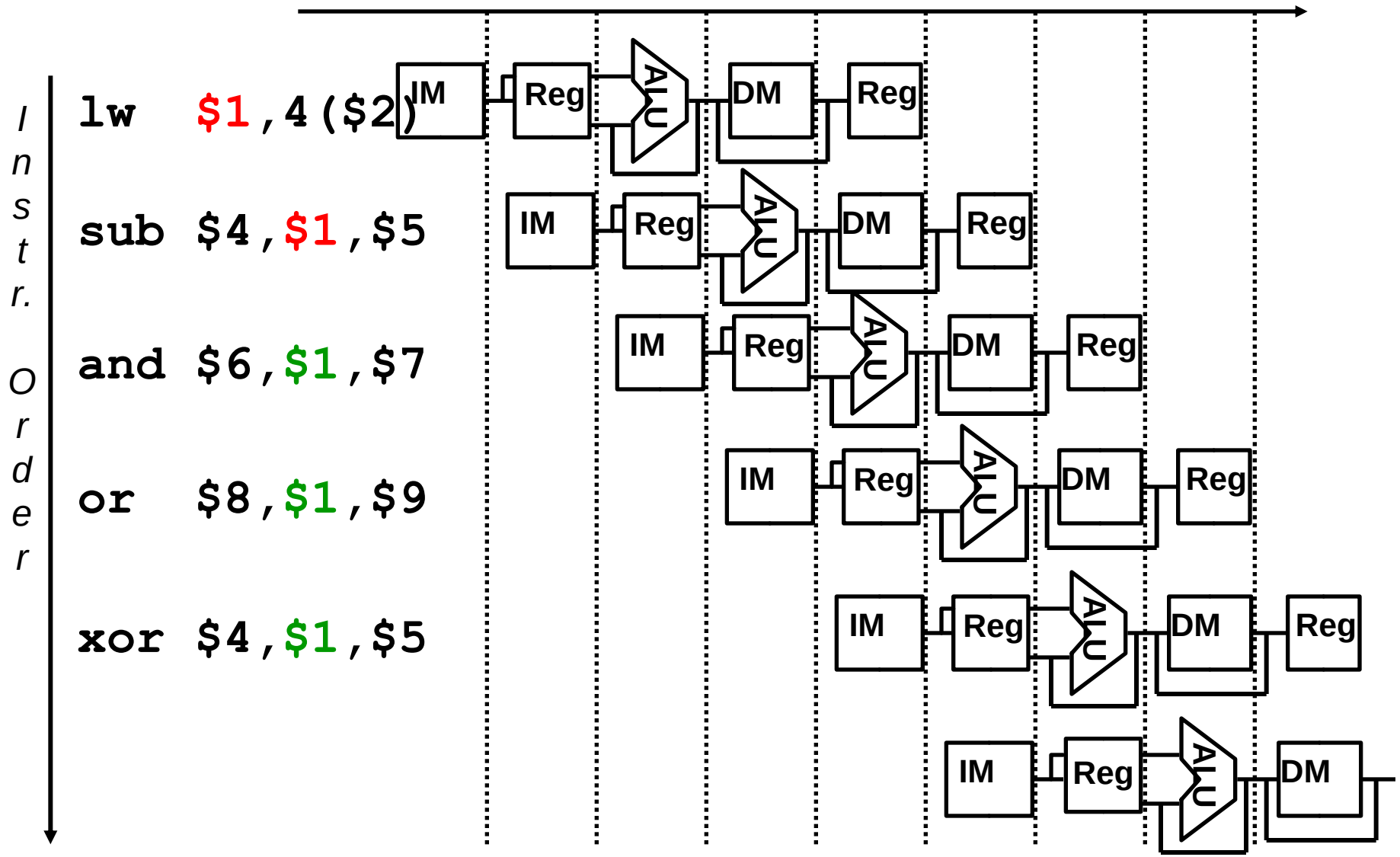


Xung đột dữ liệu khi chuyển tiếp

- 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

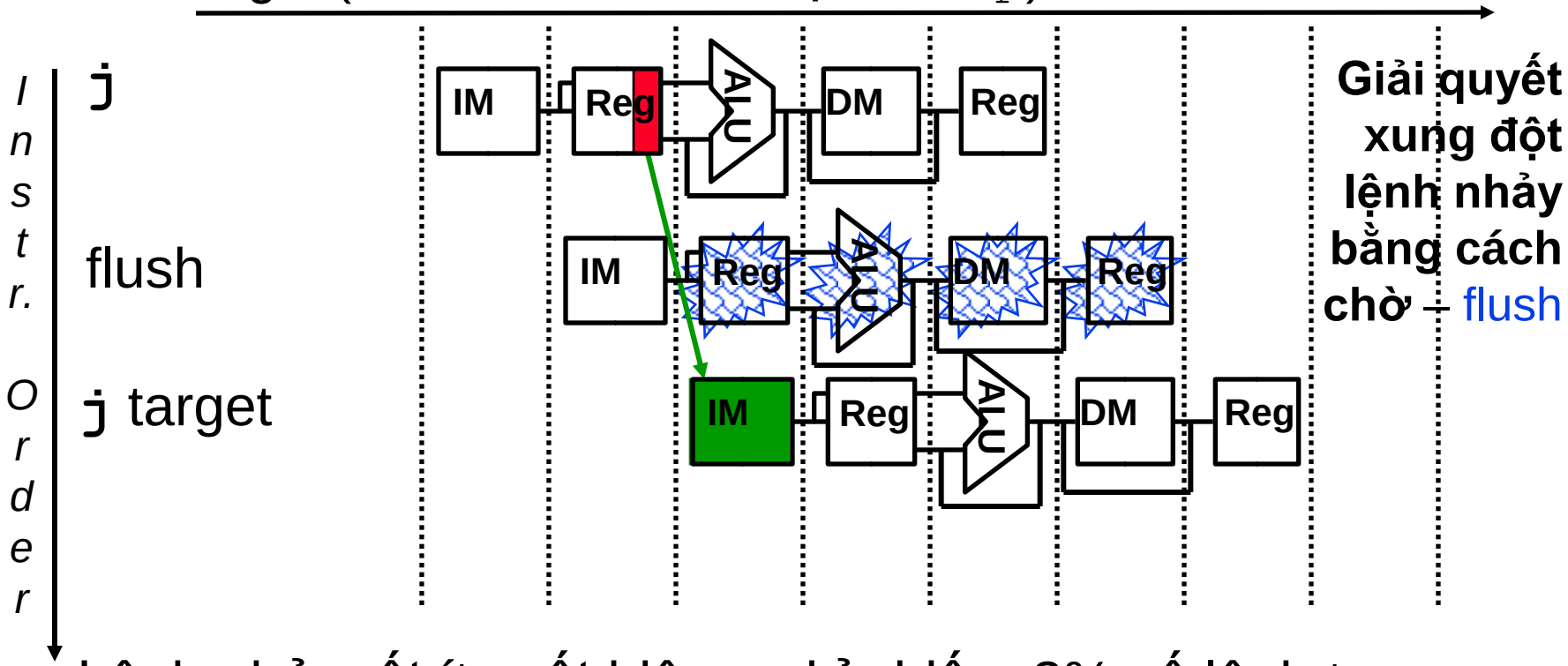


Xung đột điều khiển

- 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)
  - Tín 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

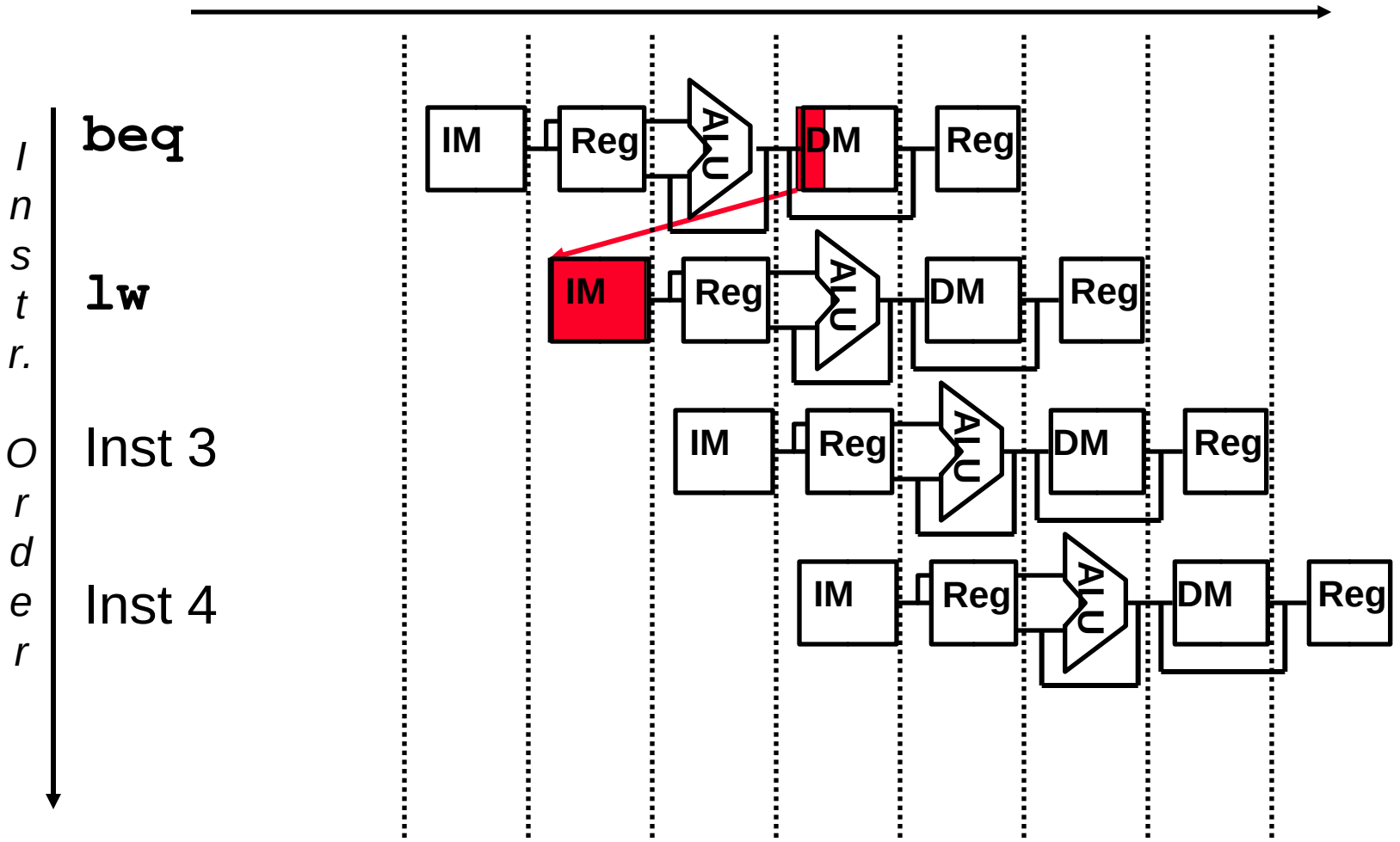
Lệnh nhảy: Cần một chu kỳ dừng

- 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`)

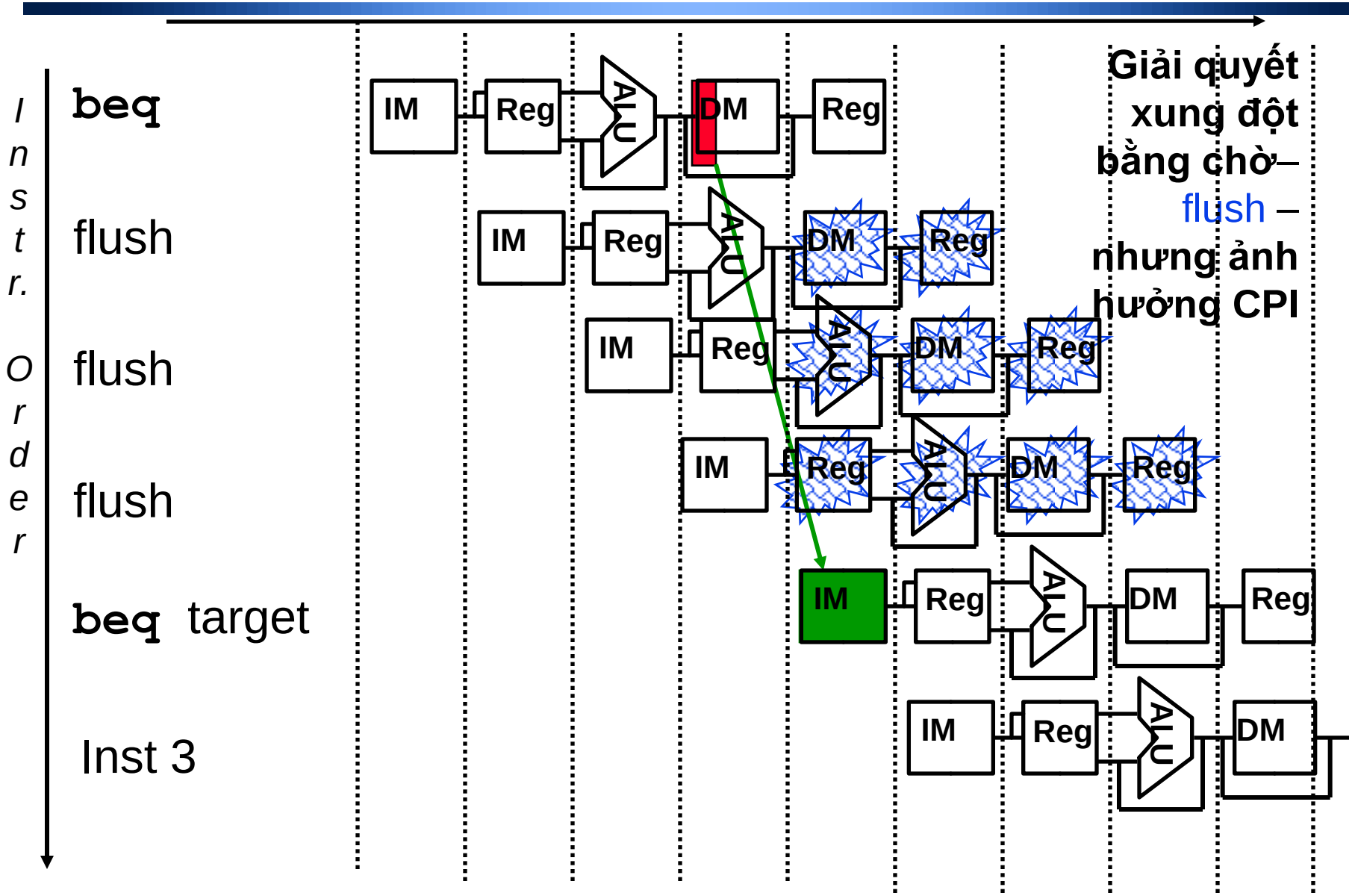


- Lệnh nhảy rất ít xuất hiện – chỉ chiếm 3% số lệnh trong SPECint

Xung đột điều khiển lệnh rẽ nhánh

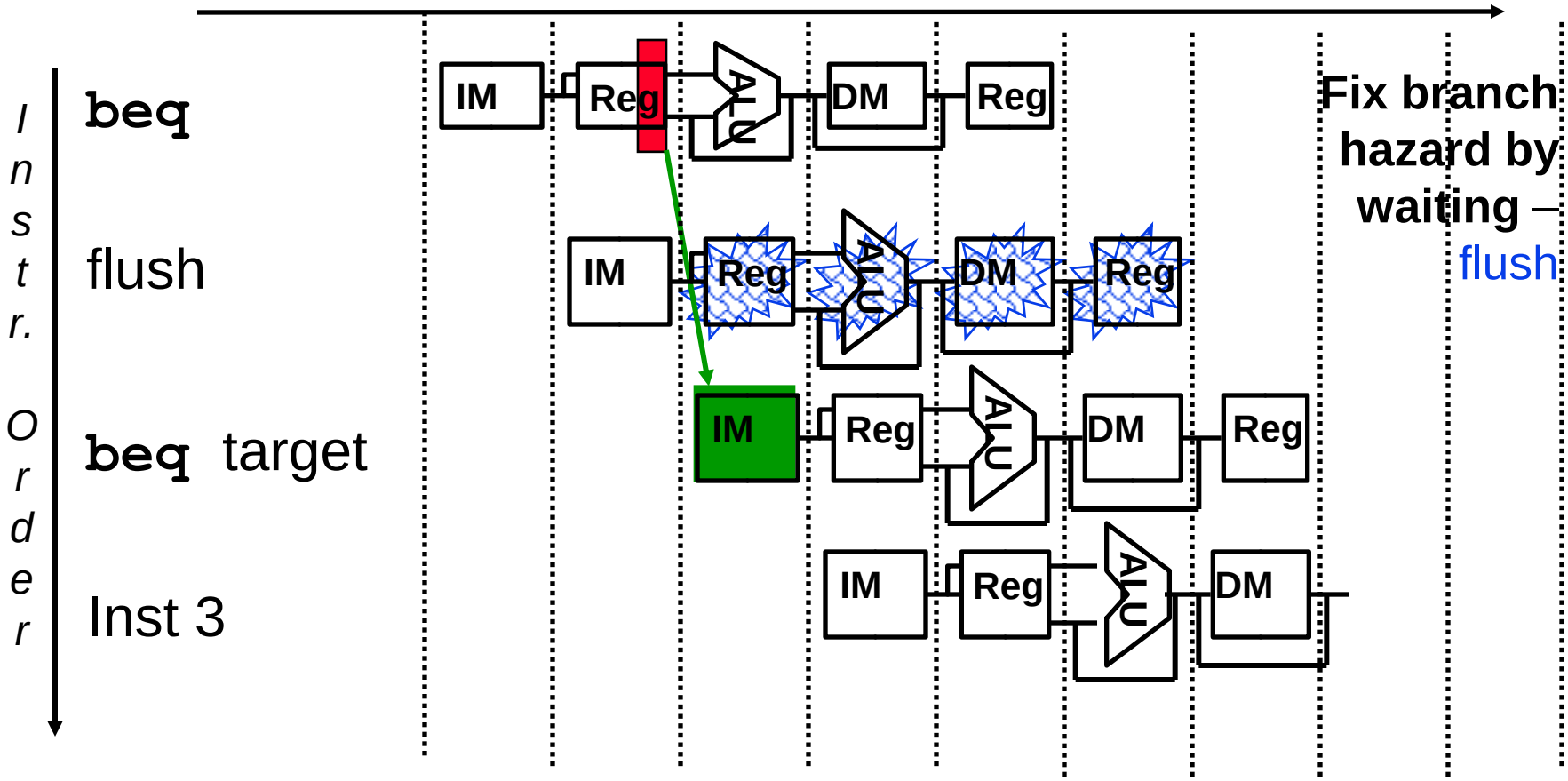


Giải quyết xung đột điều khiển lệnh rẽ nhánh



Giải quyết xung đột điều khiển lệnh rẽ nhánh

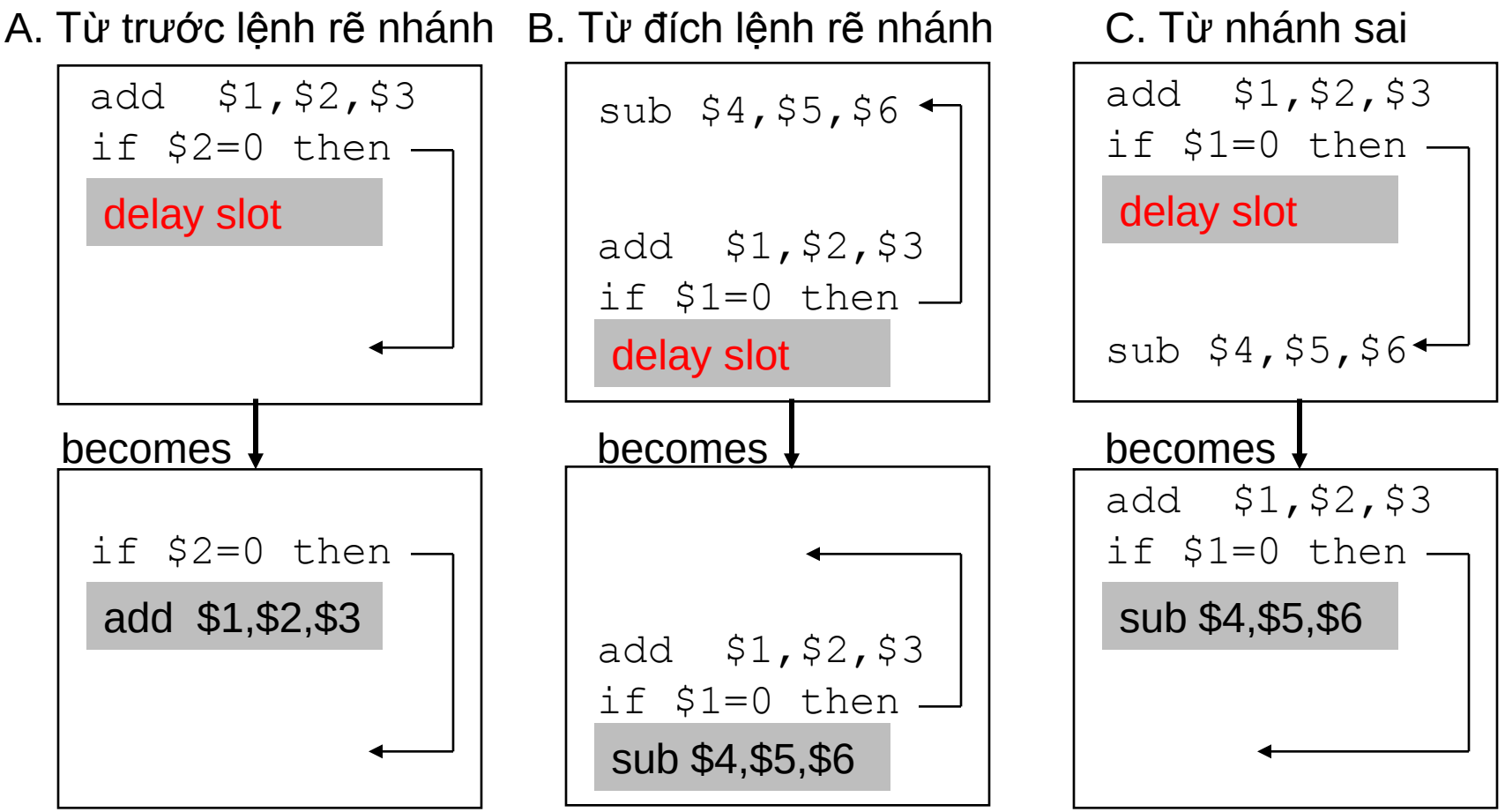
- Tính toán điều kiện rẽ nhánh càng sớm càng tốt, tức là trong giai đoạn giải mã ➔ chỉ cần 1 chu kỳ chờ



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 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 dấu 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



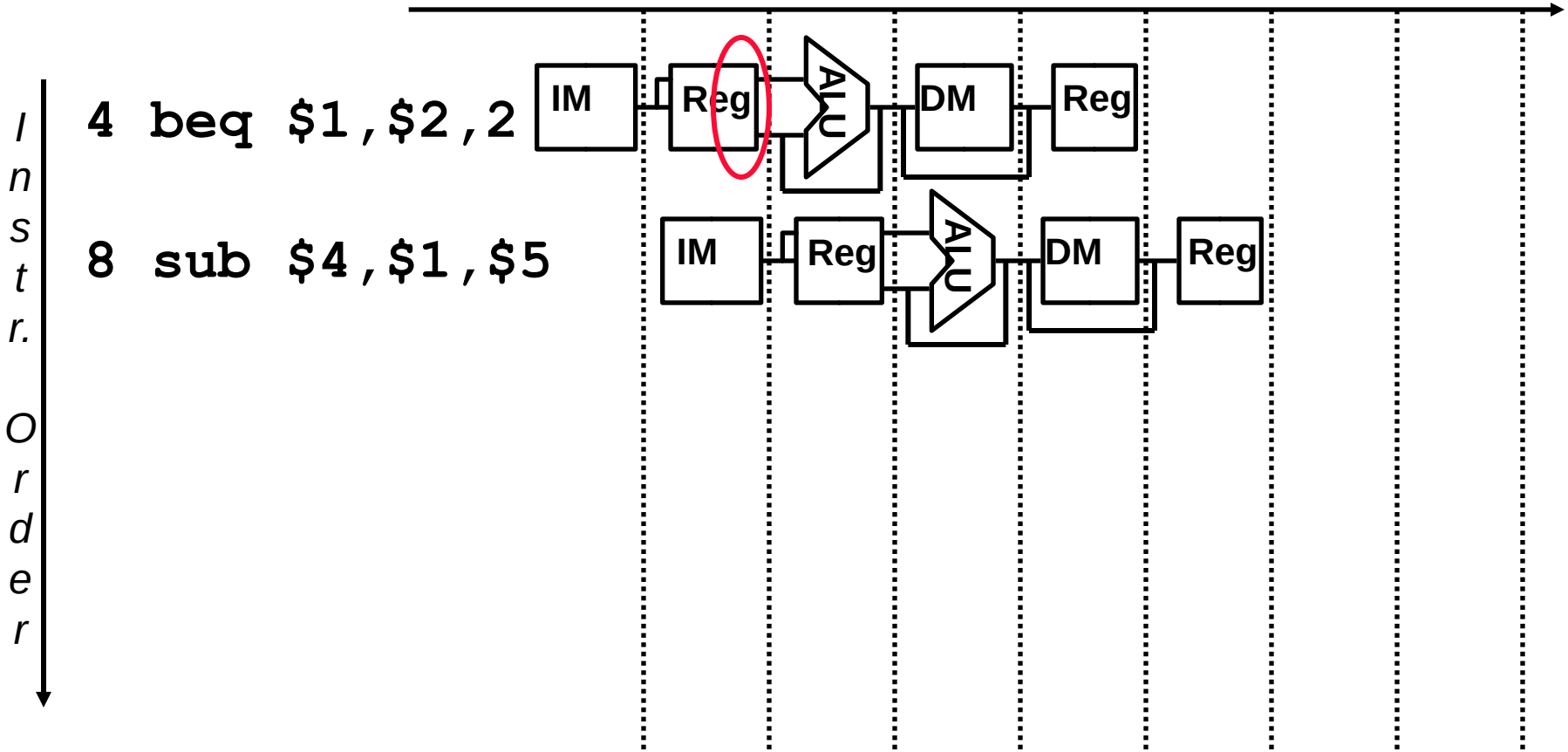
- TH A là lựa chọn tốt nhất, điền được khe trễ và giảm I
- TH B và C, lệnh `sub` cần sao lại, tăng I
- TH B và C, phải đảm bảo thực hiện lệnh `sub` không ảnh hưởng khi không rẽ nhánh

Dự đoán rẽ nhánh tĩnh

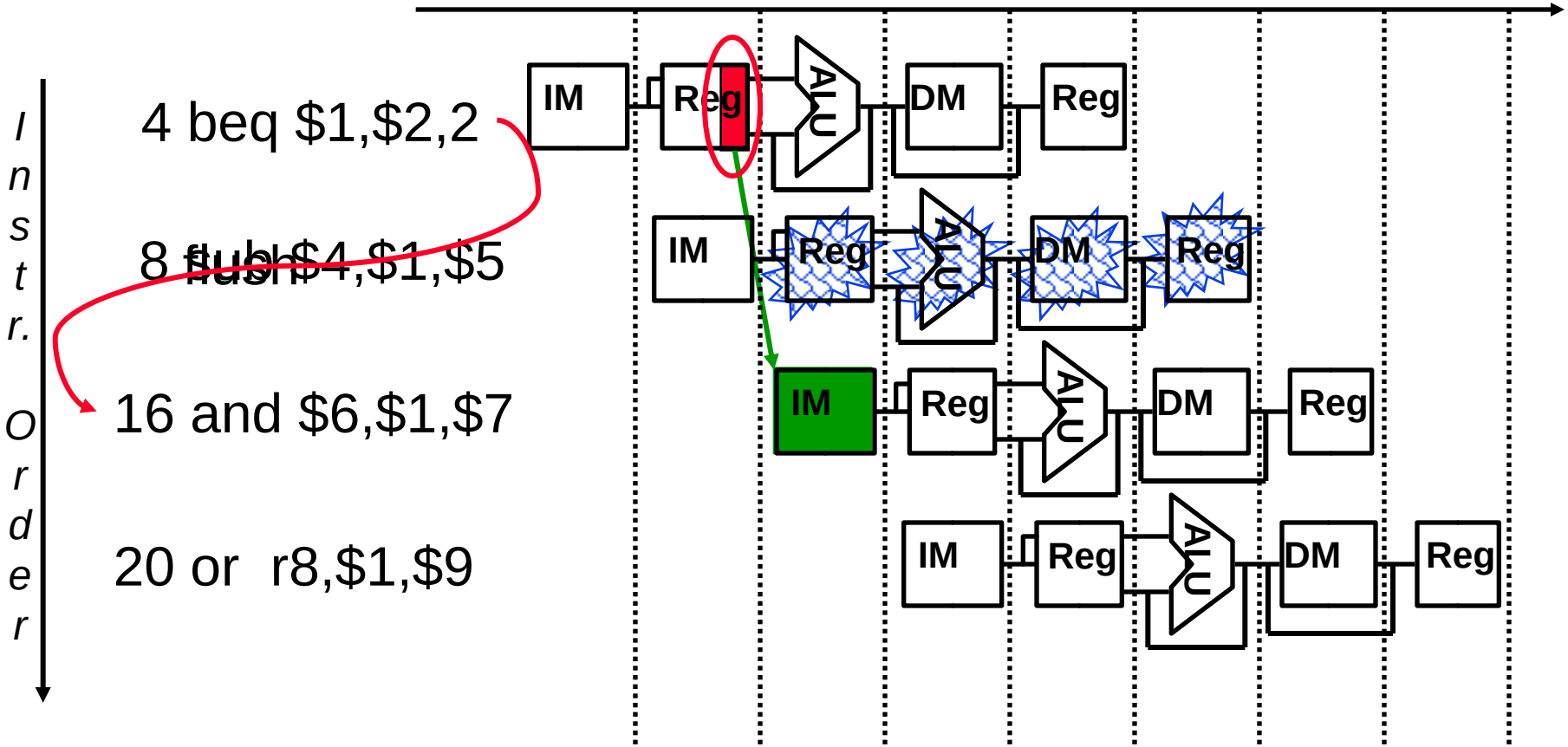
- Giải quyết xung đột rẽ nhánh bằng cách giả sử 1 hướng rẽ nhánh và tiếp tục không cần chờ tính toán kết quả rẽ nhánh thực sự.

1. Đoán không rẽ nhánh – luôn giả sử lệnh không rẽ nhánh, tiếp tục nạp các lệnh kế tiếp, chỉ khi có rẽ nhánh thì cần dừng pipeline
   - Nếu rẽ nhánh, xóa các lệnh sau rẽ nhánh (sớm ở trong pipeline)
     - trong giai đoạn IF, ID, và EX nếu bộ tính rẽ nhánh ở MEM – ba dừng
     - trong giai đoạn IF và ID nếu bộ tính rẽ nhánh ở EX – hai dừng
     - trong giai đoạn IF nếu bộ tính rẽ nhánh ở ID – một dừng
   - Đảm bảo rằng các lệnh bị xóa không ảnh hưởng tới trạng thái máy.
   - Khởi tạo lại pipeline ở đích lệnh rẽ nhánh

Xóa khi dự đoán sai



Xóa khi dự đoán sai (Đoán không rẽ nhánh)



- Để xóa, đặt trường mã lệnh của thanh ghi pipeline IF/ID bằng 0 (làm nó trở thành 1 lệnh `noop`)

Dự đoán rẽ nhánh

- Giải quyết xung đột bằng cách giả thiết kết quả rẽ nhánh và tiếp tục

2. Đoán có rẽ nhánh – dữ đoán luôn luôn có rẽ nhánh
   - Đoán có rẽ nhánh luôn cần 1 chu kỳ dừng (nếu phần cứng tính rẽ nhánh ở giai đoạn ID)
   - Cần phương pháp đọc trước (vào bộ đệm) lệnh ở địa chỉ đích??

- Vì thiệt hại do rẽ nhánh đang tăng lên (với các pipeline sâu), mô hình dự đoán rẽ nhánh tĩnh sẽ ảnh hưởng tới hiệu năng. Với nhiều phần cứng hơn, có thể thử dự đoán hoạt động rẽ nhánh động lúc chương trình được thực hiện

3. Dự đoán rẽ nhánh động – đoán rẽ nhánh lúc chạy dựa trên các thông tin chạy (*run-time information*)

Dự đoán rẽ nhánh động bằng 1 bit

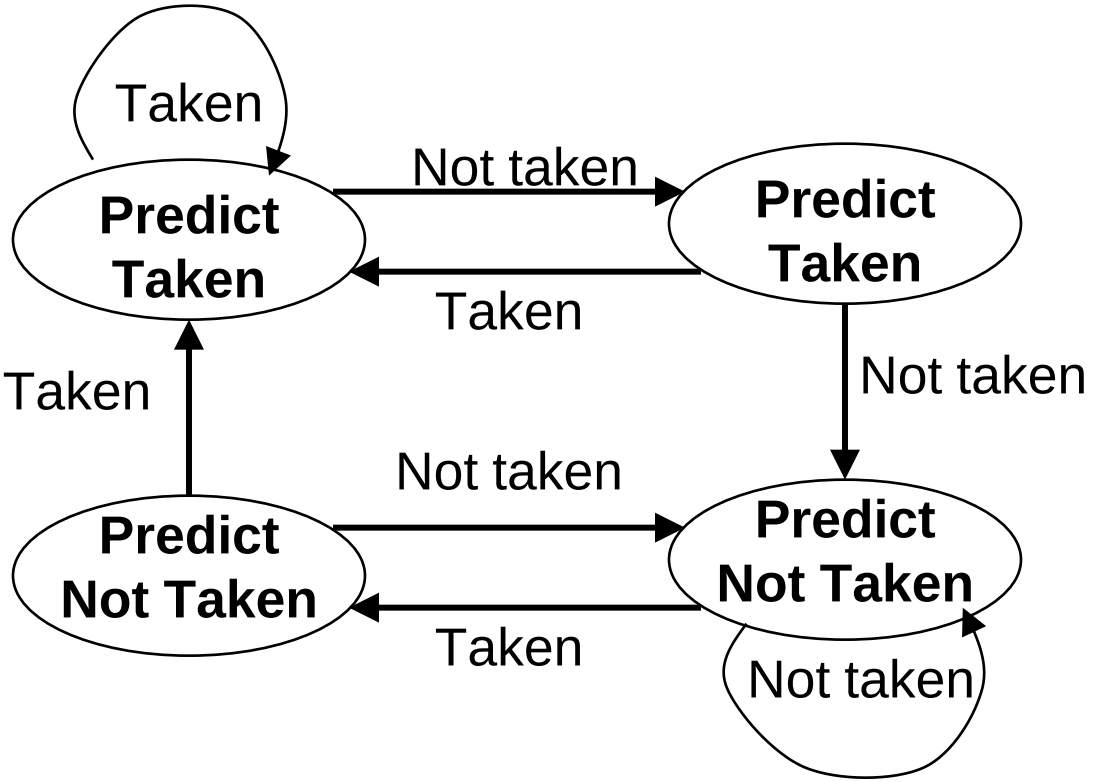
- Bô dự đoán 1 bít sẽ sai 2 lần nếu đoán không rẽ nhánh:
  - Giá sử predict_bit = 0 lúc bắt đầu (chỉ ra không rẽ nhánh) lệnh rẽ nhánh điều khiển ở cuối vòng lặp
  1. Lần thực hiện vòng lặp 1, bộ dự đoán sai cho lệnh rẽ nhánh vì nó dẫn quay lại đầu vòng lặp; cần đảo bít rẽ nhánh (predict_bit = 1)
  2. Khi nào vẫn còn rẽ nhánh (vẫn lặp), dự đoán đúng
  3. Khi thoát khỏi vòng lặp, bộ dự đoán sẽ sai 1 lần nữa vì lần này sẽ không rẽ nhánh mà ra ngoài vòng lặp; đảo bit rẽ nhánh (predict_bit = 0)

```
Loop: 1st loop instr
      2nd loop instr
            .
            .
            .
      last loop instr
      bne $1,$2,Loop
      fall out instr
```

- Nếu lặp 10 lần ta sẽ có tỉ lệ rẽ nhánh đúng là 80%. Nếu dùng lệnh rẽ nhánh beq thì đúng 90%

Dự đoán rẽ nhánh động bằng 2 bit

- Cơ chế dùng 2 bit cho độ chính xác 90% vì chỉ khi dự đoán sai 2 lần thì bit dự đoán mới thay đổi




```
Loop: 1st loop instr
      2nd loop instr
            .
            .
            .
      last loop instr
      bne $1,$2,Loop
      fall out instr
```

Exceptions

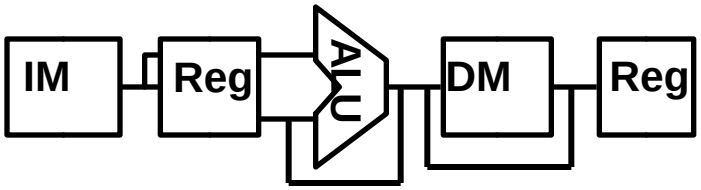
- Exceptions (ngắt - *interrupts*) có thể coi là 1 dạng xung đột dữ liệu. Exception xuất hiện từ:
  - Tràn khi thực hiện lệnh số học
  - Lệnh không được định nghĩa
  - Yêu cầu từ thiết bị vào ra
  - Yêu cầu dịch vụ hệ điều hành (VD. lỗi trang, lỗi TLB)
  - Lỗi chức năng phần cứng
- Pipeline cần phải:
  - dừng thực hiện lệnh lỗi,
  - để tất cả các lệnh trước đó hoàn thành,
  - xóa các lệnh sau đó,
  - đặt thanh ghi chỉ ra nguyên nhân exception,
  - lưu lại địa chỉ lệnh lỗi,
  - nhảy đến địa chỉ định trước (địa chỉ của hàm xử lý exception)
- Phần mềm (OS) sẽ xử lý tiếp exception.

Hai loại exceptions

- Ngắt – không đồng bộ với sự thực hiện chương trình
  - gây ra bởi sự kiện bên ngoài
  - có thể được xử lý giữa các lệnh, nên để các lệnh đang có trong pipeline *hoàn thành* trước khi chuyển điều khiển cho hàm xử lý ngắt của OS.
  - đơn giản là dừng và tiếp tục chương trình người dùng

- Bẫy (Exception) – đồng bộ với sự thực hiện chương trình
  - gây ra bởi sự kiện bên trong
  - hàm xử lý bẫy cần sửa chữa điều kiện cho đúng lệnh bị bẫy, nên phải dừng lệnh lỗi trong trong pipeline và chuyển điều khiển cho hàm xử lý bẫy của OS
  - lệnh lỗi có thể tiếp tục chương trình có thể bị kết thúc hoặc được tiếp tục

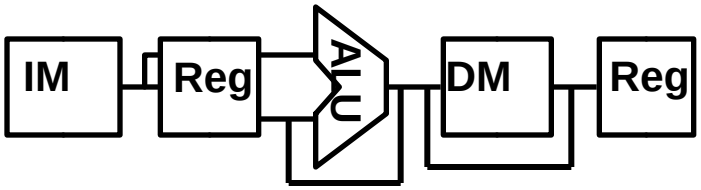
Exception có thể xuất hiện ở đâu trong pipeline



Stage(s)? Synchronous?

- Tràn số học
- Lệnh không định nghĩa
- Lỗi TLB hoặc trang
- Yêu cầu dịch vụ I/O
- Lỗi phần cứng

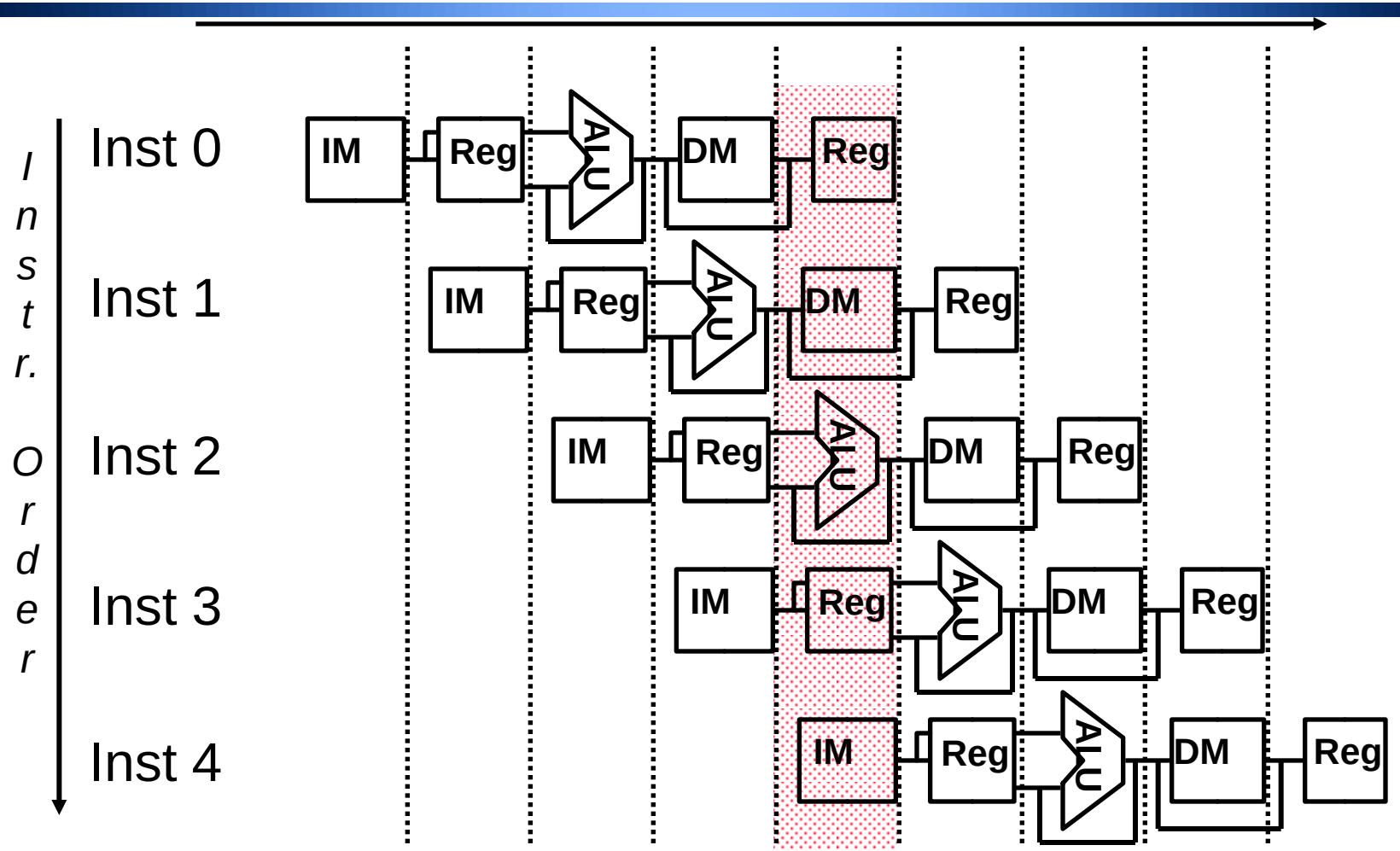
Exception có thể xuất hiện ở đâu trong pipeline



	Stage(s)?	Synchronous?
❑ Tràn số học	EX	yes
❑ Lệnh không định nghĩa	ID	yes
❑ Lỗi TLB hoặc trang	IF, MEM	yes
❑ Yêu cầu dịch vụ I/O	any	no
❑ Lỗi phần cứng	any	no

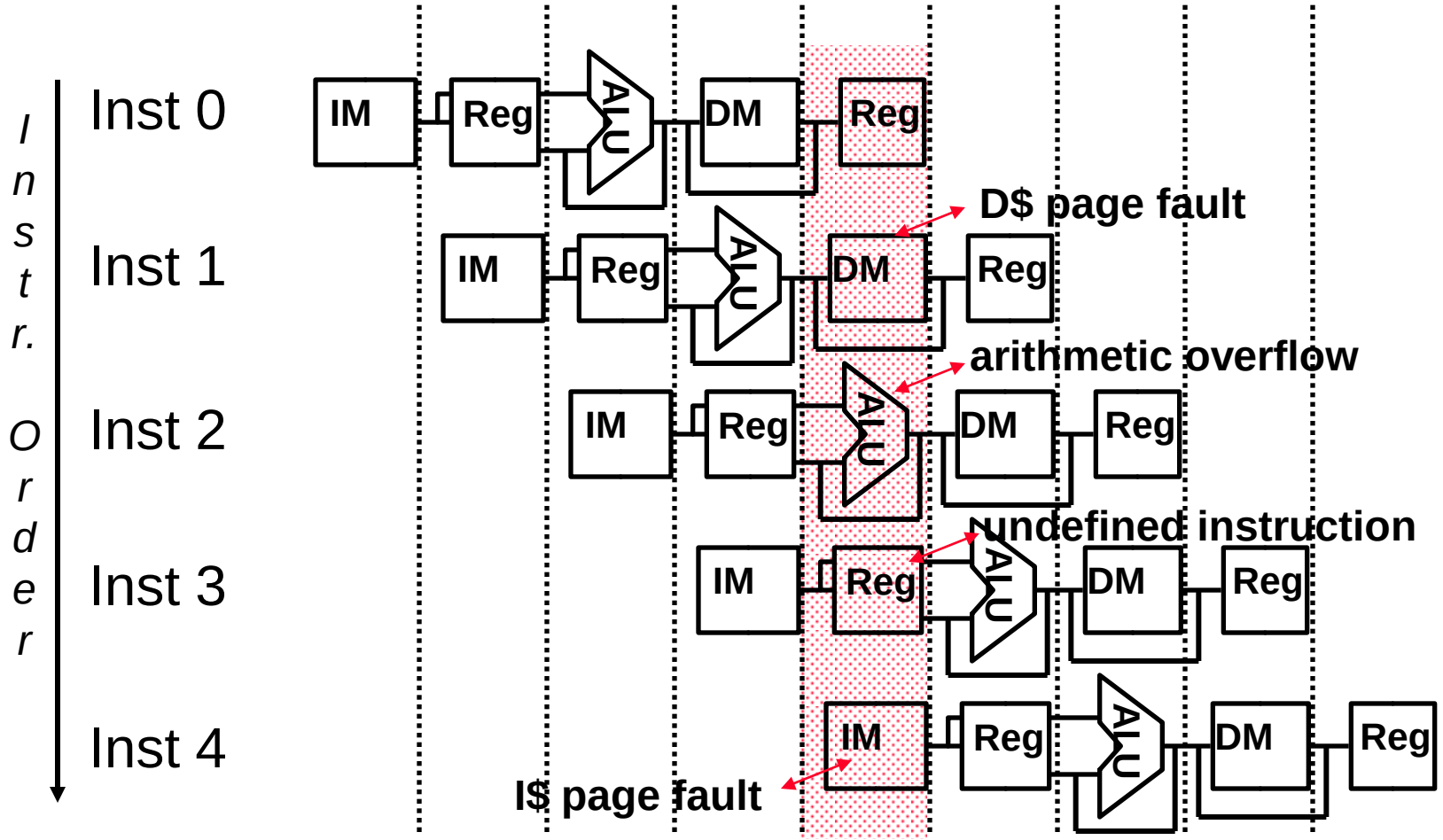
❑ Chú ý rằng nhiều exception có thể xuất hiện đồng thời trong *một* chu kỳ đồng hồ

Nhiều exception đồng thời



- Lệnh sớm nhất sẽ bị ngắt đầu tiên

Nhiều exception đồng thời



- Lệnh sớm nhất sẽ bị ngắt đầu tiên

Tổng kết

- Tất cả các bộ xử lý hiện đại đều dùng pipeline để tăng hiệu suất (CPI=1 và đồng hồ nhanh - fc lớn)
- Tốc độ đồng hồ pipeline bị giới hạn bởi giai đoạn pipeline chậm nhất – thiết kế pipeline cân bằng là rất quan trọng
- Cần phát hiện và giải quyết xung đột trong pipeline
  - Xung cấu trúc – giải quyết: thiết kế pipeline đúng
  - Xung đột dữ liệu
    - Dừng (ảnh hưởng CPI)
    - Chuyển tiếp (cần phần cứng hỗ trợ)
  - Xung đột điều khiển – đặt phần cứng quyết định rẽ nhánh lên các trạng thái đầu trong pipeline
    - Dừng (ảnh hưởng CPI)
    - Rẽ nhánh chậm (cần hỗ trợ của trình dịch)
    - Dự đoán rẽ nhánh tĩnh và động (cần phần cứng hỗ trợ)
- Xử lý ngắt trong pipeline phức tạp