Bộ xử lý đường ống

Processor Pipelining

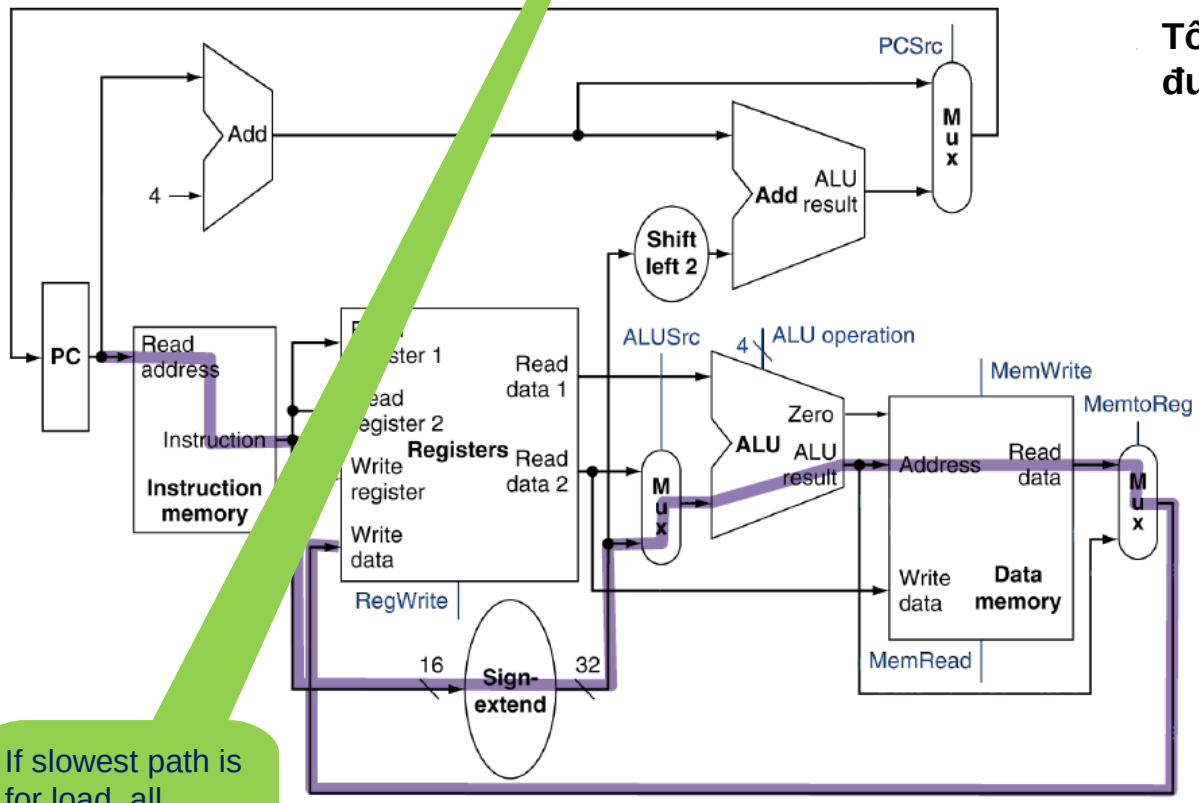
Nội dung

- **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?
- X**ây 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?)

Đường dữ liệu bộ xử lý đơn xung nhịp



Tốc độ xác định bởi lệnh có đường dữ liệu dài nhất.

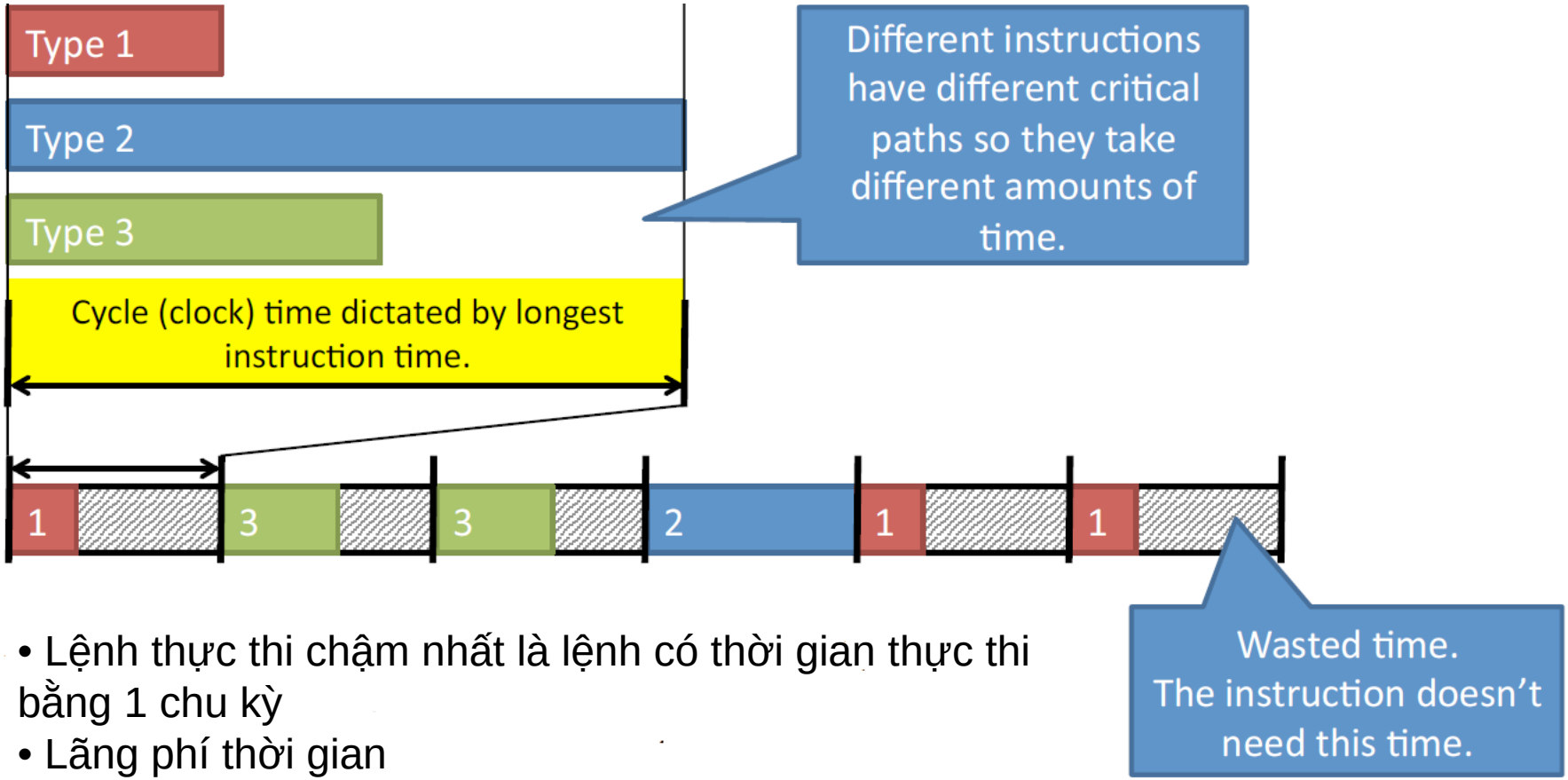
Q: Nếu truy cập vào bộ nhớ dữ liệu dài hơn gấp 2 lần các lệnh khác và 30% lệnh của chương trình là loads/stores, bao nhiêu phần trăm thời gian bộ xử lý nhàn rỗi?

1. 20% of the time
2. 35% of the time
3. 40% of the time

A: 70%•1/2 = 35% of the time.
70% các lệnh cần một nửa chu kỳ để xử lý. Như vậy 35% thời gian để lãng phí.

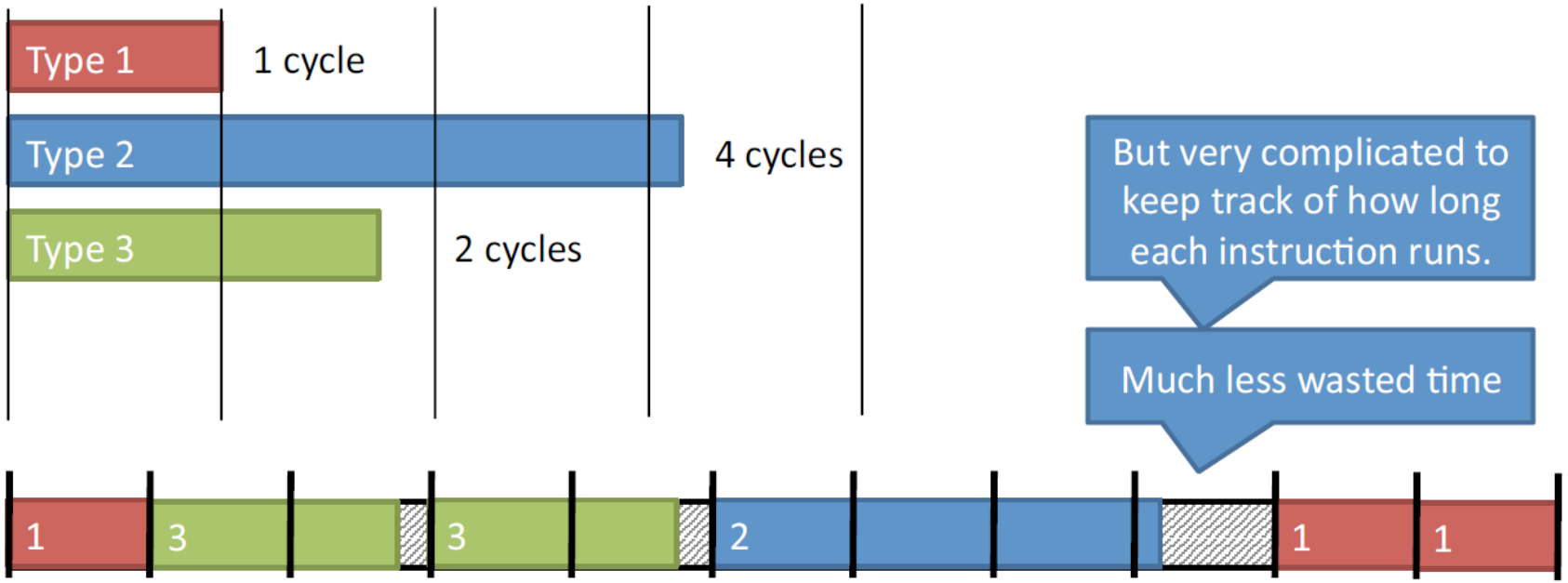
If slowest path is for load, all instructions go this slowly.

Thời gian thực thi đơn xung nhịp



- Lệnh thực thi chậm nhất là lệnh có thời gian thực thi bằng 1 chu kỳ
- Lãng phí thời gian

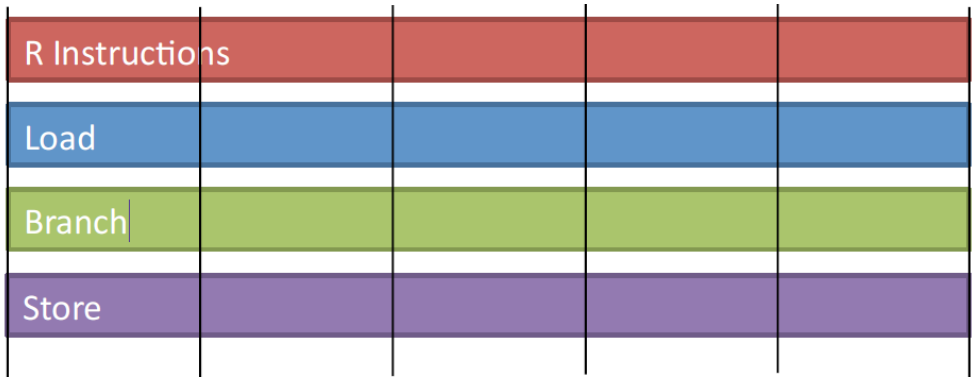
Giải pháp: Bộ xử lý đa xung nhịp



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

Cách nào tốt hơn?

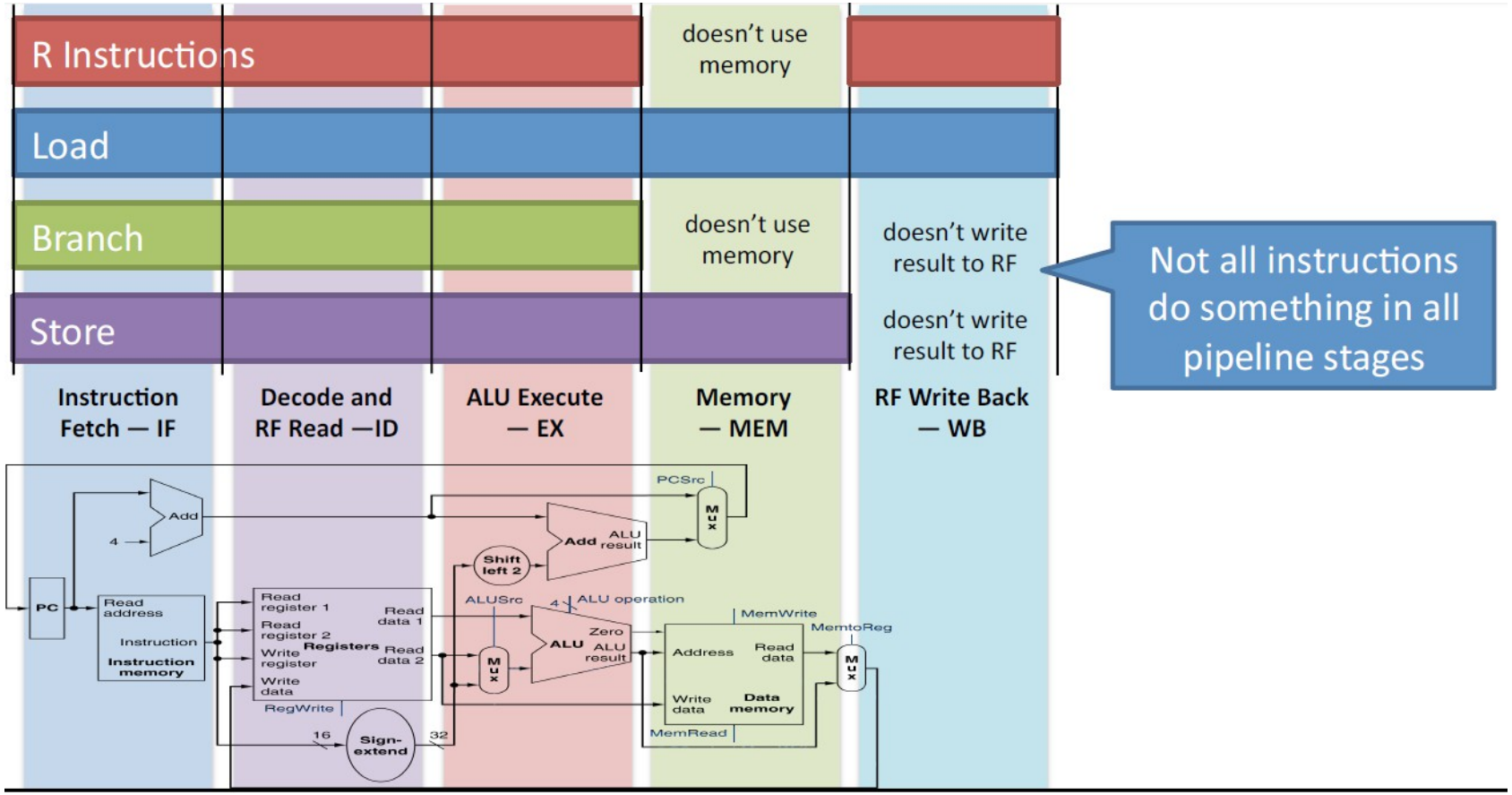
- Chia lệnh thành các giai đoạn khác nhau
- Giai đoạn dài nhất sẽ xác định tốc độ xử lý



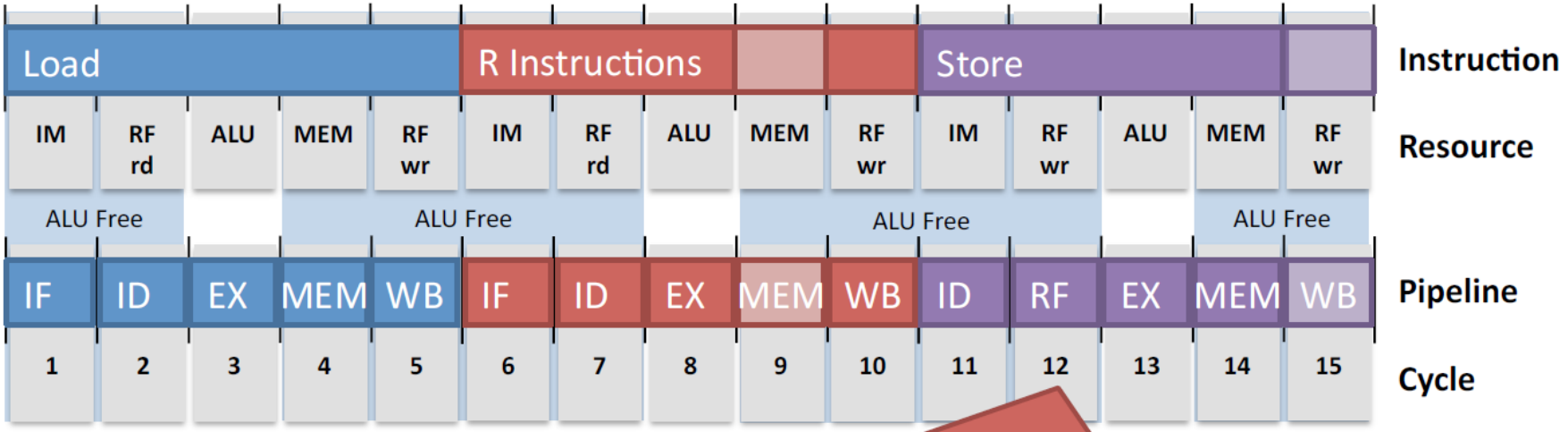
…Cần nhiều chu kỳ cho một lệnh!

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)

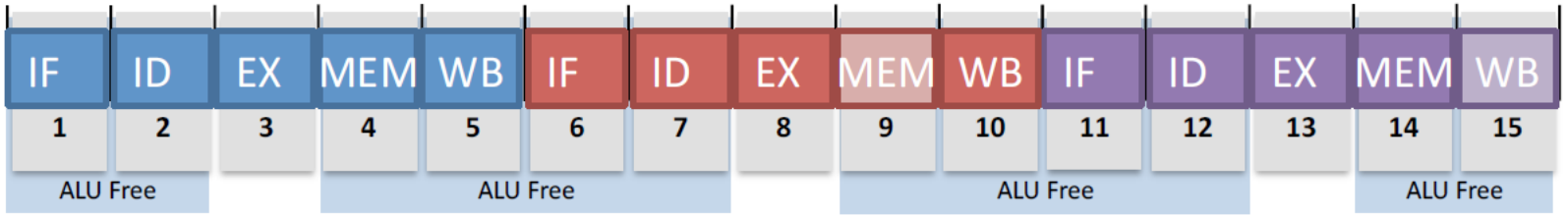
Q: What is the ALU doing in cycle 12?

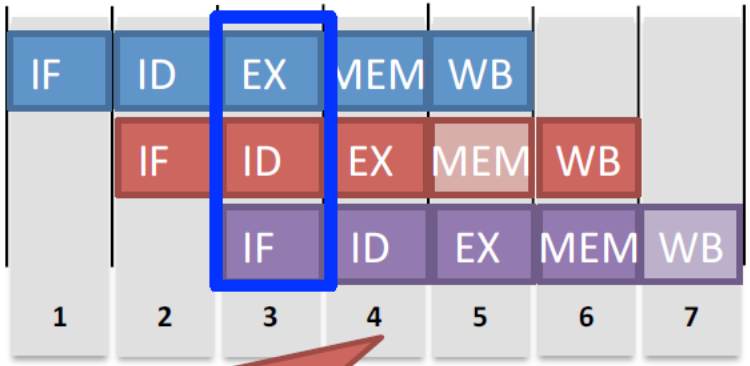
- Nothing
- Accessing the register file
- Calculating the store address

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)

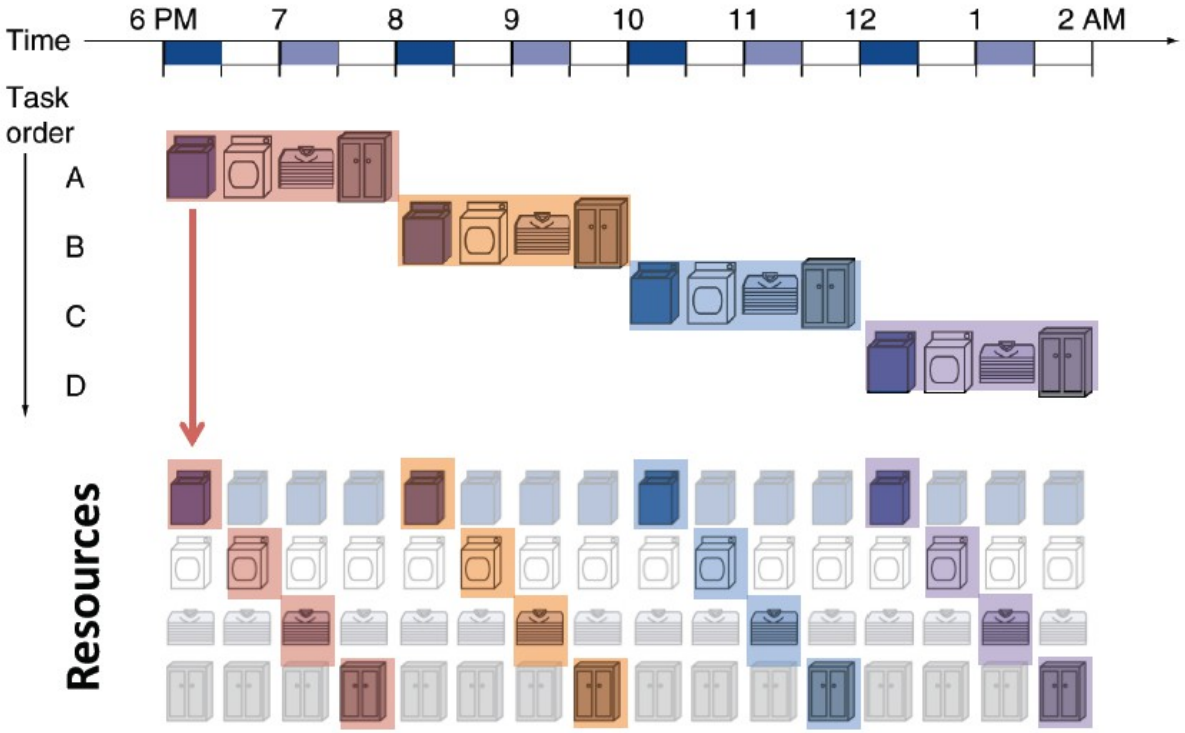
- 4 hoạt động cho một tải:
  - Wash (1h), Dry (1h), Fold (1h), Put away (1h)
- 4 tải mất bao lâu?
  - Wash + Dray + Fold + Put away = 4h
  - **4 loads * 4h/load = 16h**

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

1. 100%
2. 50%
3. 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?

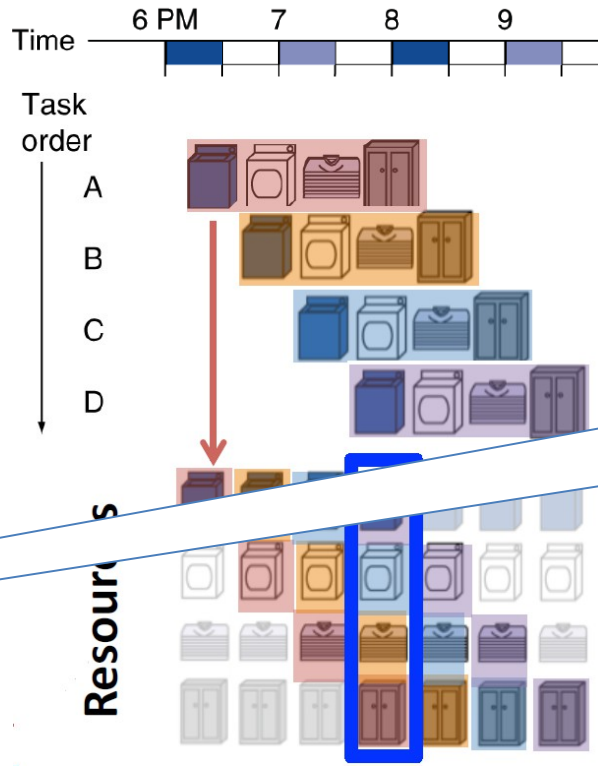
Ví dụ: Quy trình giặt là (pipelined)

- 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. 1
2. 2
3. 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.

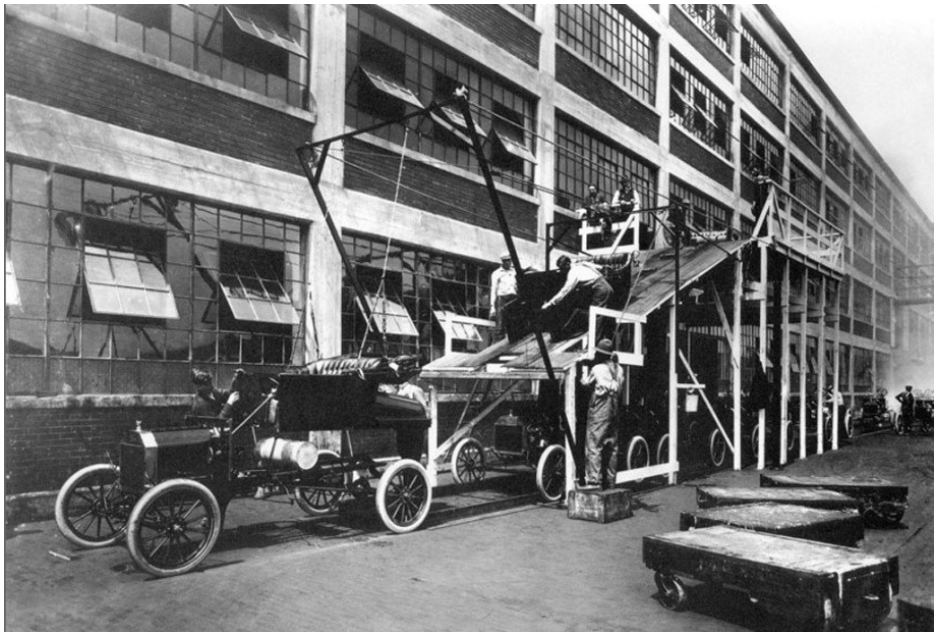


Đạt được hiệu xuất cao hơn. Khi cần 4 tải tại một thời điểm sử dụng toàn bộ tài nguyên.

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

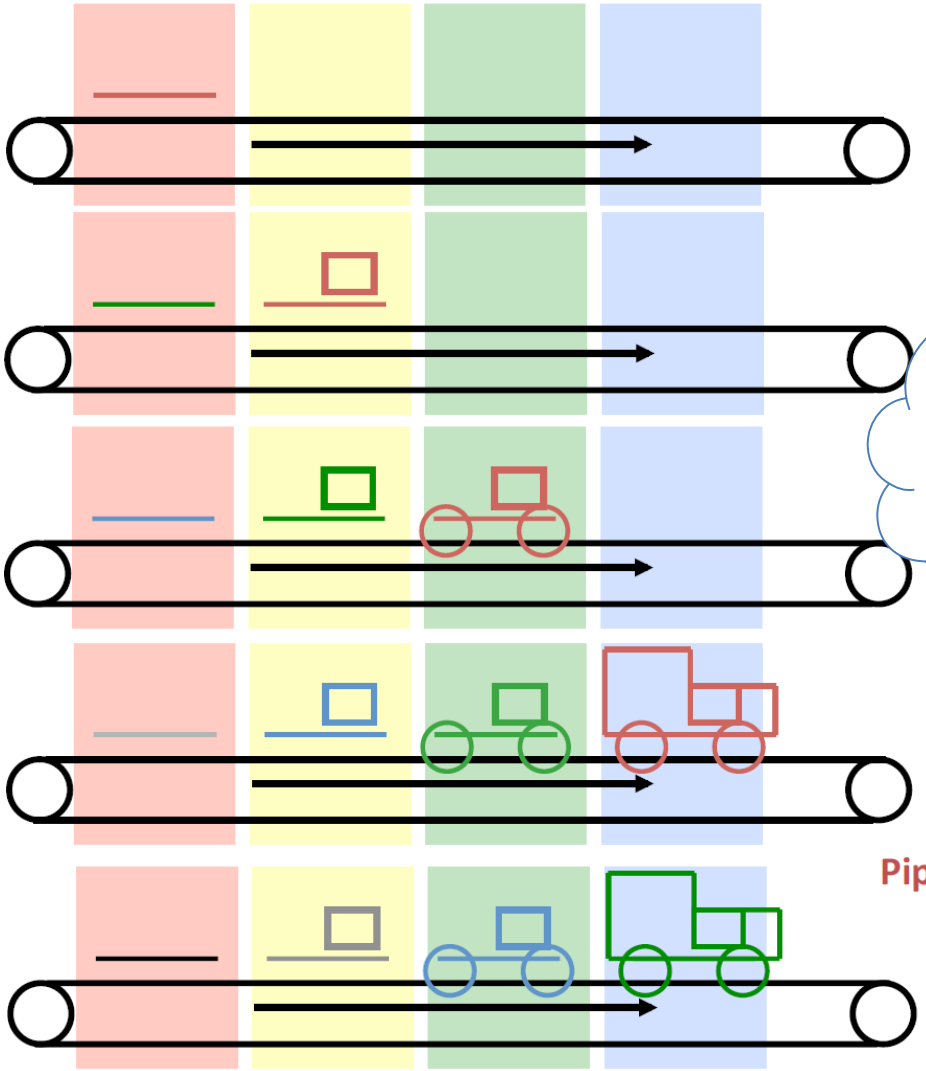
Pipelining example 2: car assembly (serial)

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

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

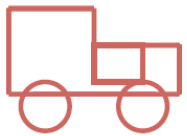
A: Goes down

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, gập, 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).

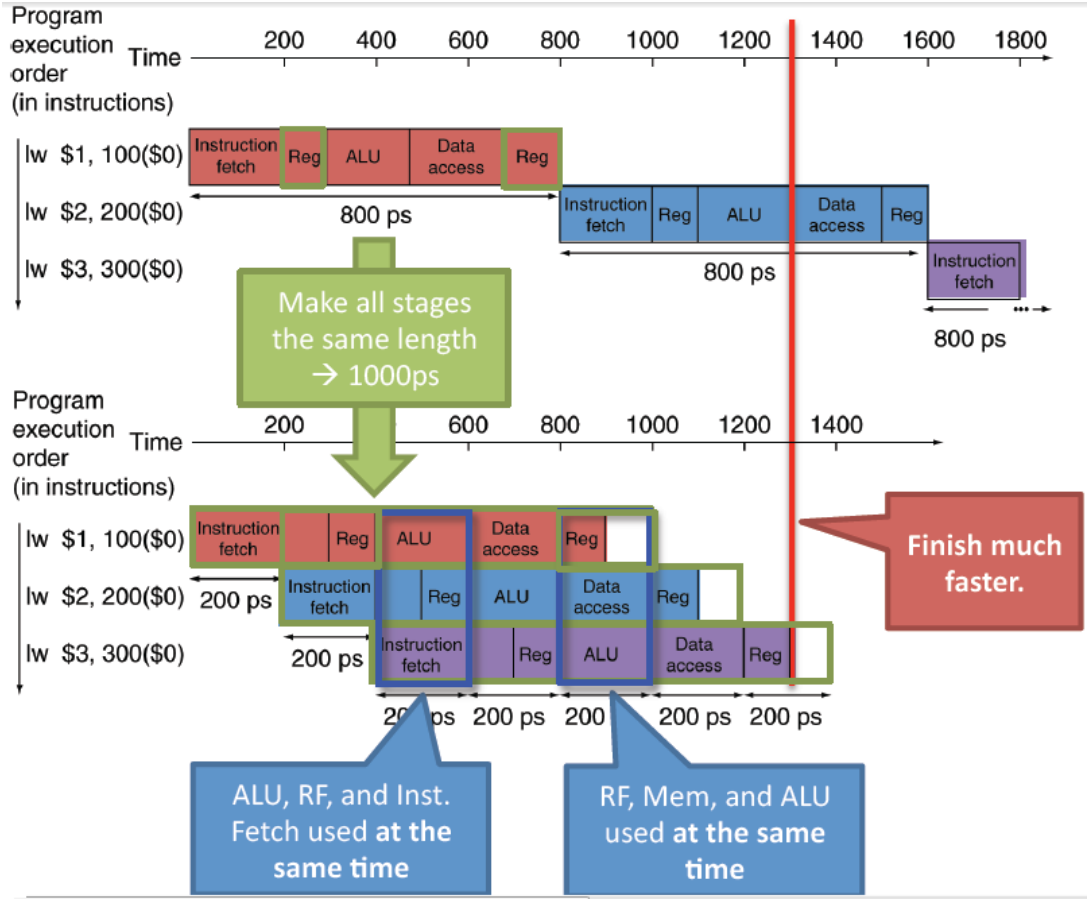
Hiệu năng đường ống hóa trong bộ xử lý

• 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. 1.7lần
2. 4lần
3. 5lần

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 lệnh 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).



Nhanh hơn bao nhiêu?

- Tăng tốc **Pipeline**
  - Nếu tất cả các pha có cùng chiều dài.

$$\text{Time per finished unit pipelined} = \frac{\text{Time per finished unit non-pipeline}}{\text{Number of pipeline stages}}$$

- **Ví dụ : Pipelined**
  - Thời gian cho một tải giặt là = 4h/4 giai đoạn = **1 load /1h** (throughput)
  - Thời gian cho một ô tô = 4h/4 giai đoạn = **1 car /1h** (throughput)
- **Nhưng**
  - Thời gian cho tải giặt là vẫn là **4h** (latency)
  - Thời gian tạo một xe ô tô vẫn là **4h** (latency)
- Đường ống hóa chỉ tăng thông lượng khi đường ống đầy
  - Tốc độ tăng lên 2.3x.

Tại sao không chia nhiều giai đoạn hơn?

- Ý tưởng là sẽ tăng tốc được N***x*** đối với một pipeline N pha?

- Why not use a zillion stages to get a zillion ***x*** speedup?

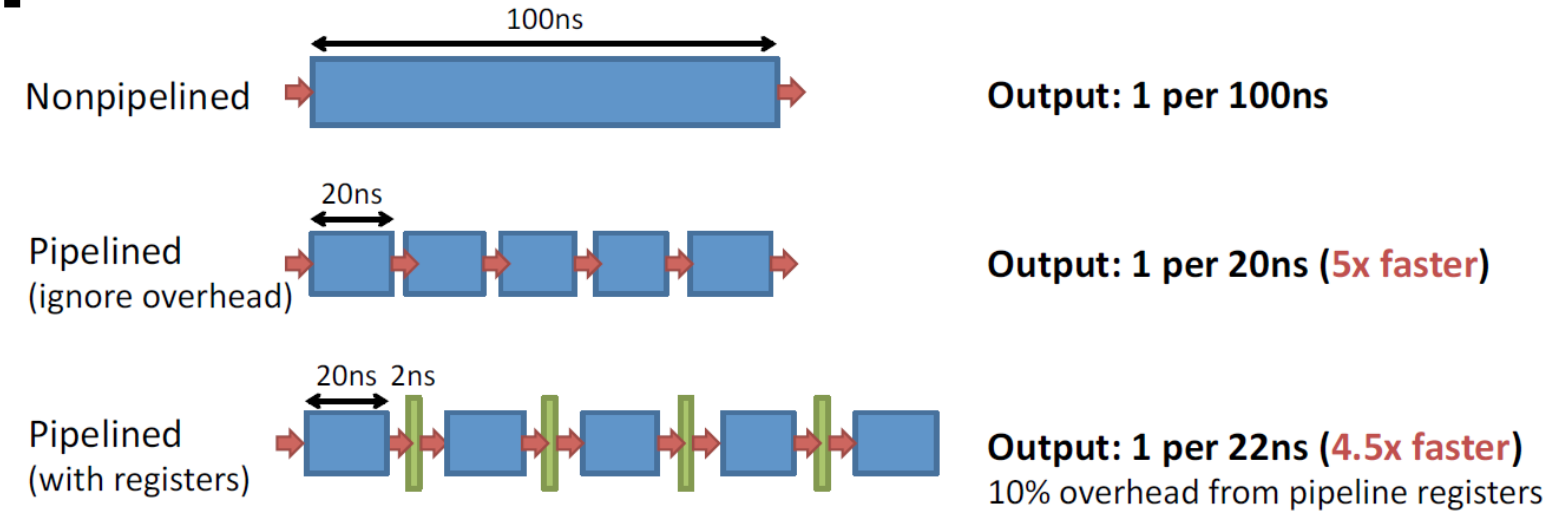
- Two problems:
  - Most things **can't be broken down into infinitely small chunks**

- Think about the processor we built:

- How much can we chop up the ALU? or the RF?

- Practical limit to logic design
  - There is an **overhead for every stage**
- We need to store the state (which instruction) for each stage
- This requires a register, and it takes some time

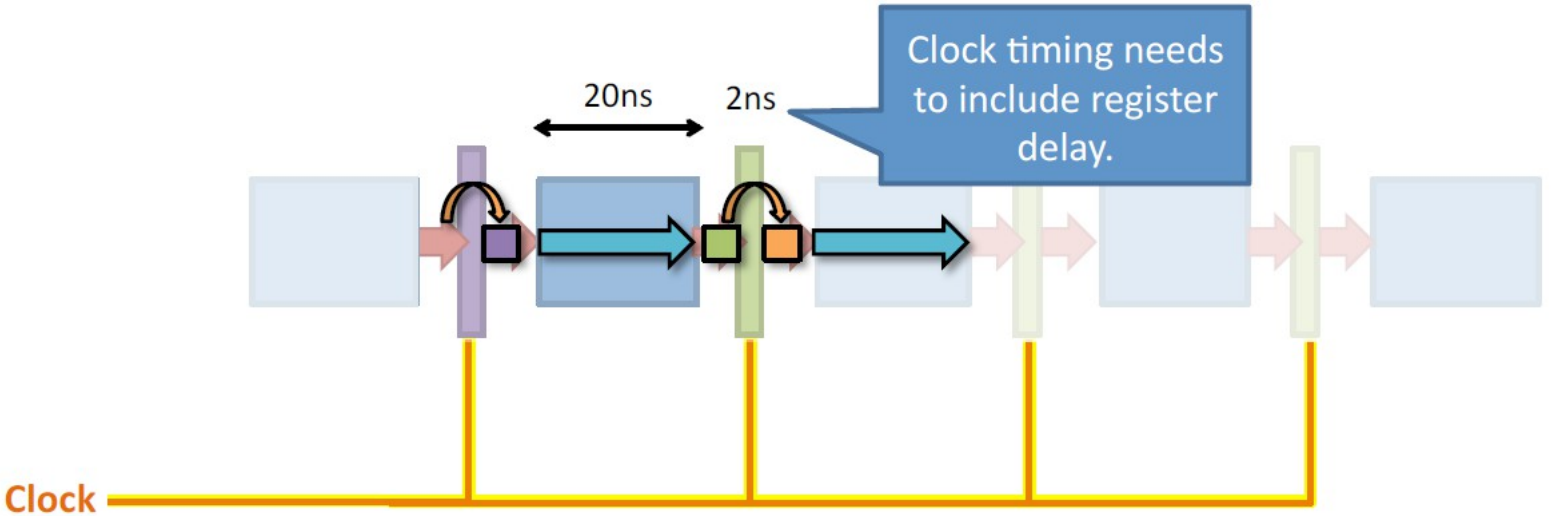
Các thanh ghi Pipeline và mào đầu (phí tổn điều khiển)



- 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?

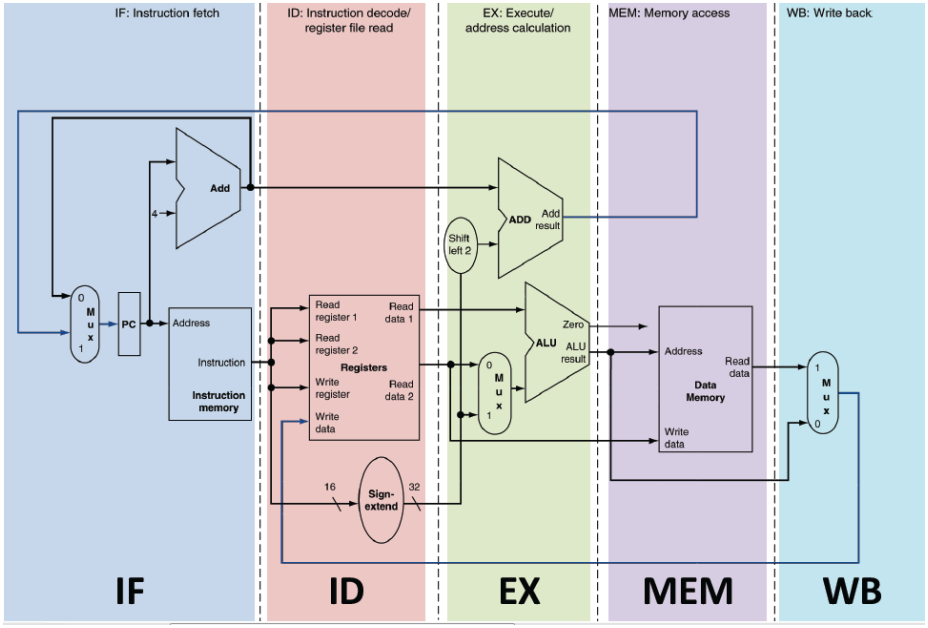
(You've already seen it…)

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

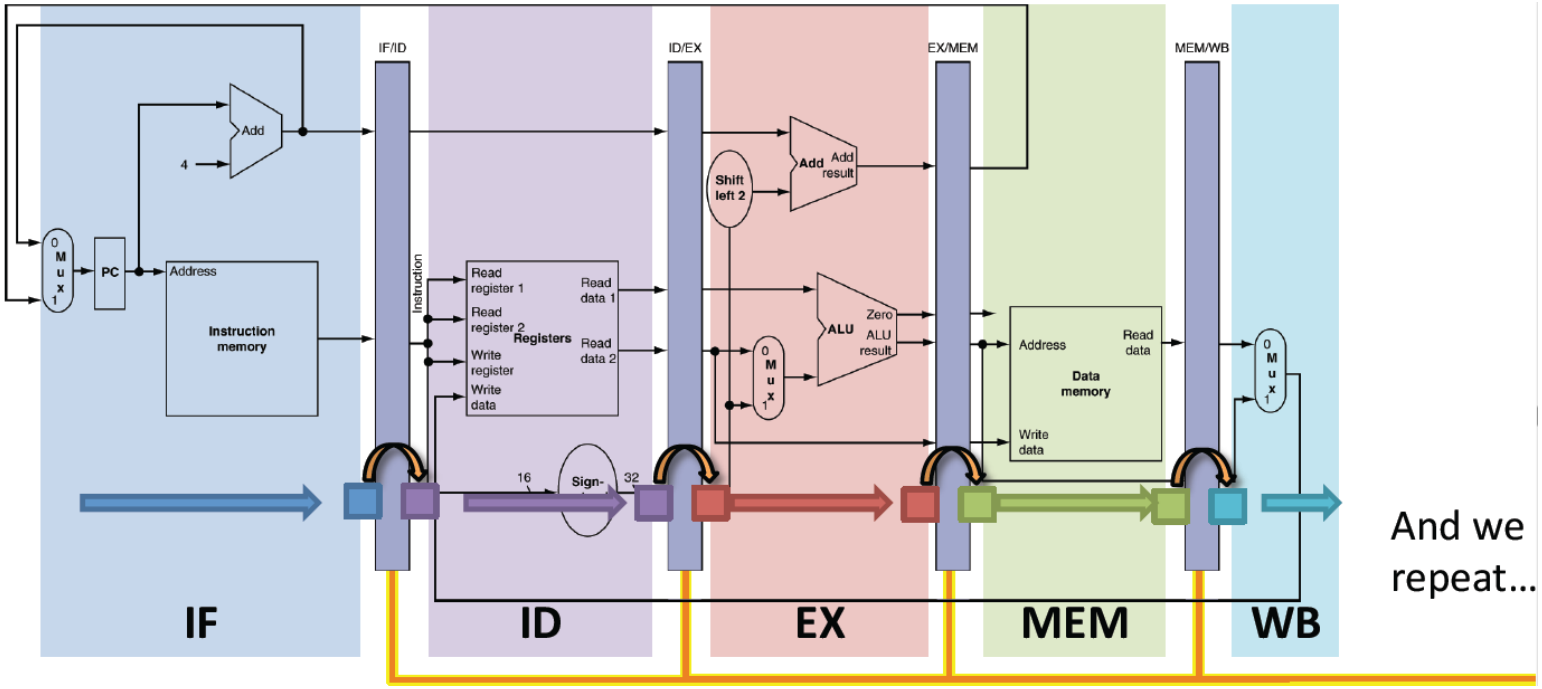
Q: Thiếu cái gì trong hình vẽ?

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.

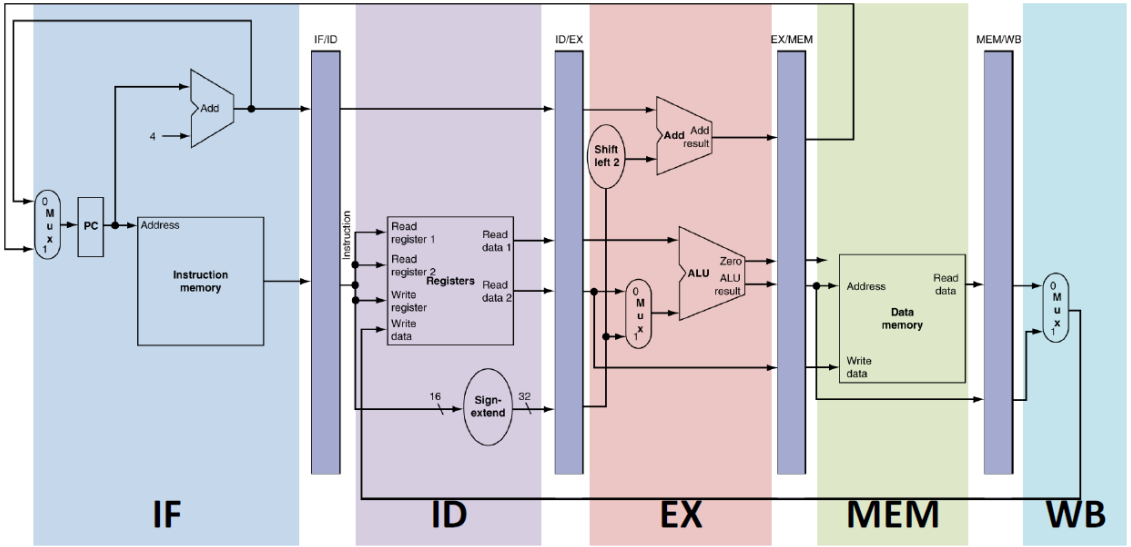


Các thanh ghi pipeline.



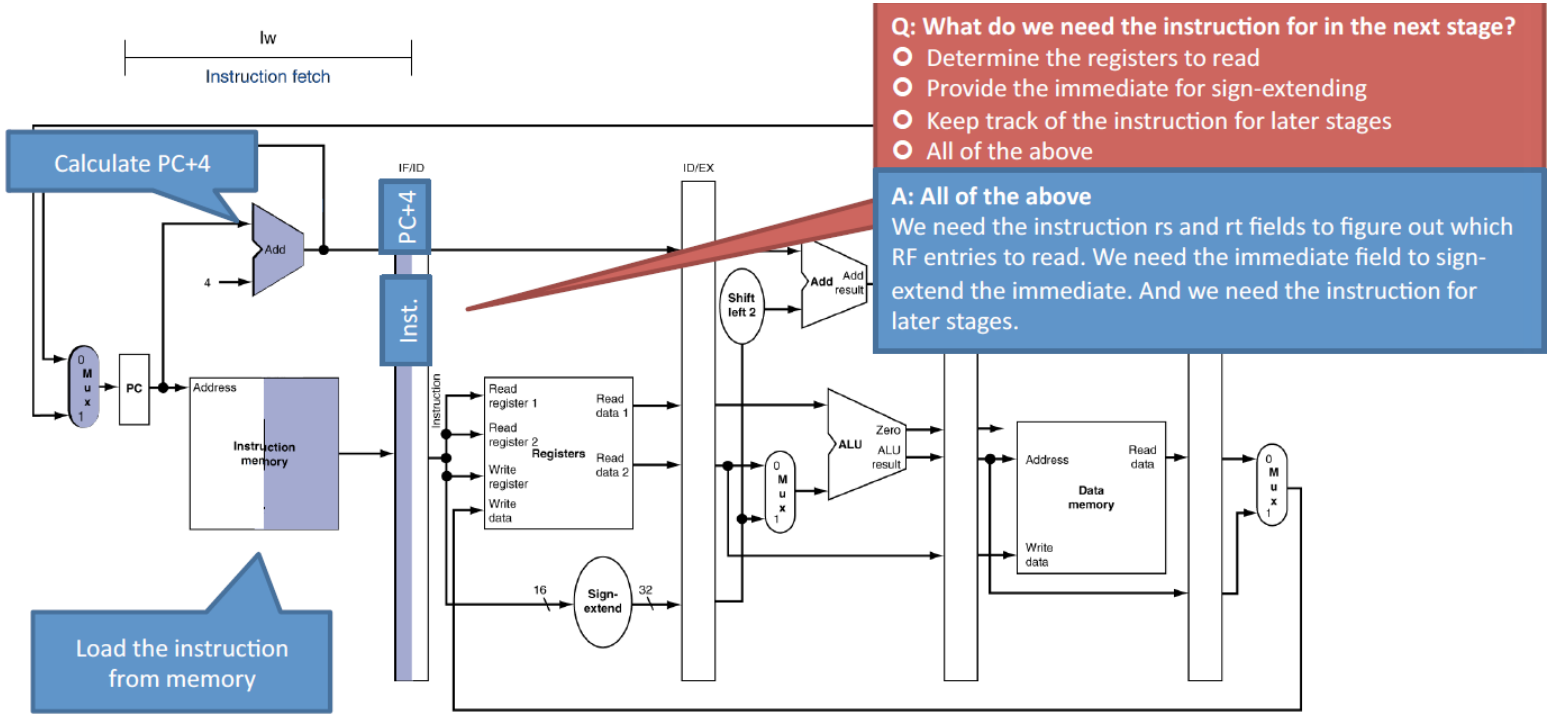
- Các thanh ghi lưu giữ thông tin thủ tục giữa các pha.
- Dịch chuyển dữ liệu đến các pha tiếp kế tiếp theo xung đồng hồ

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



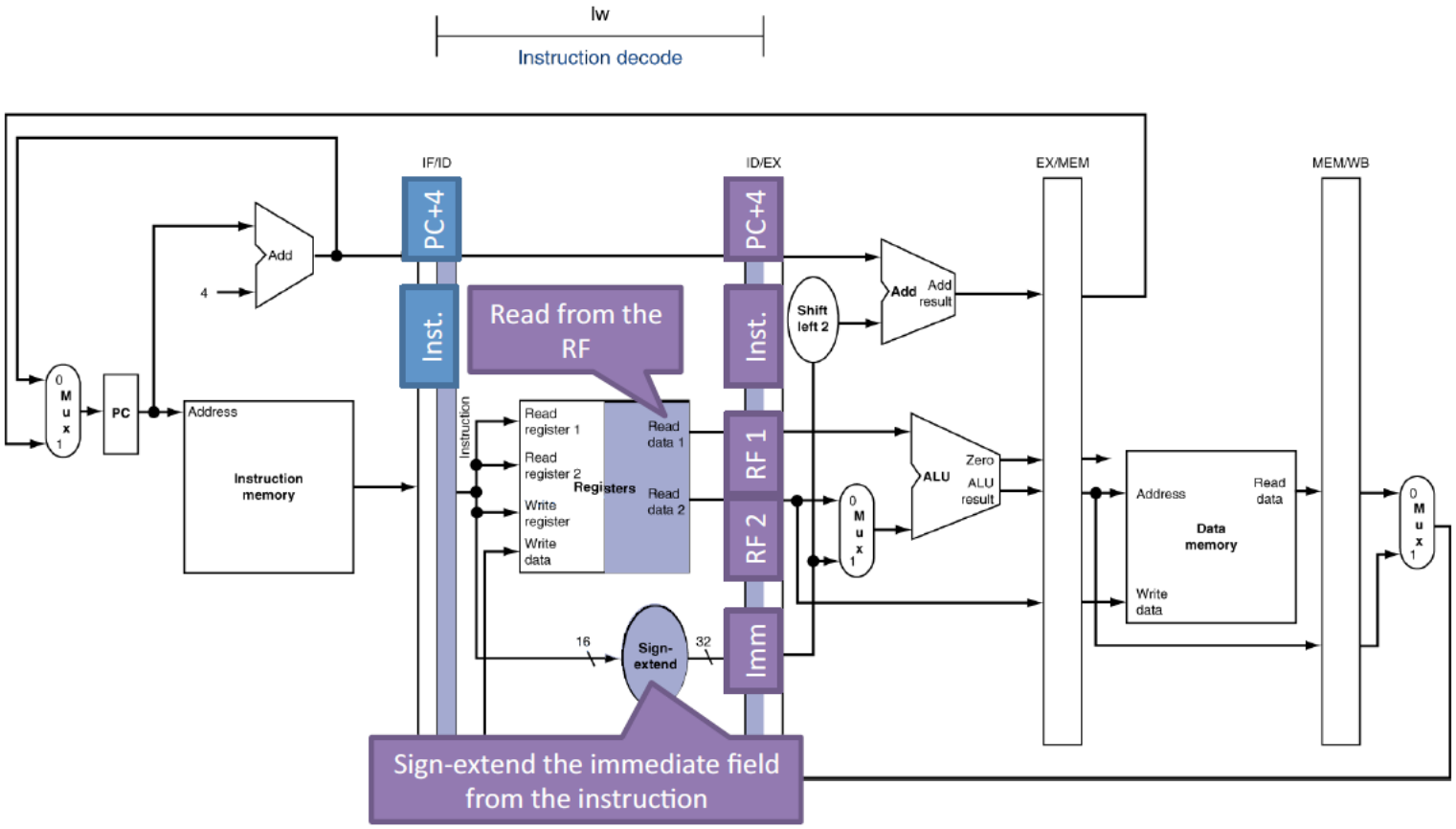
Q: What do we need the instruction for in the next stage?

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

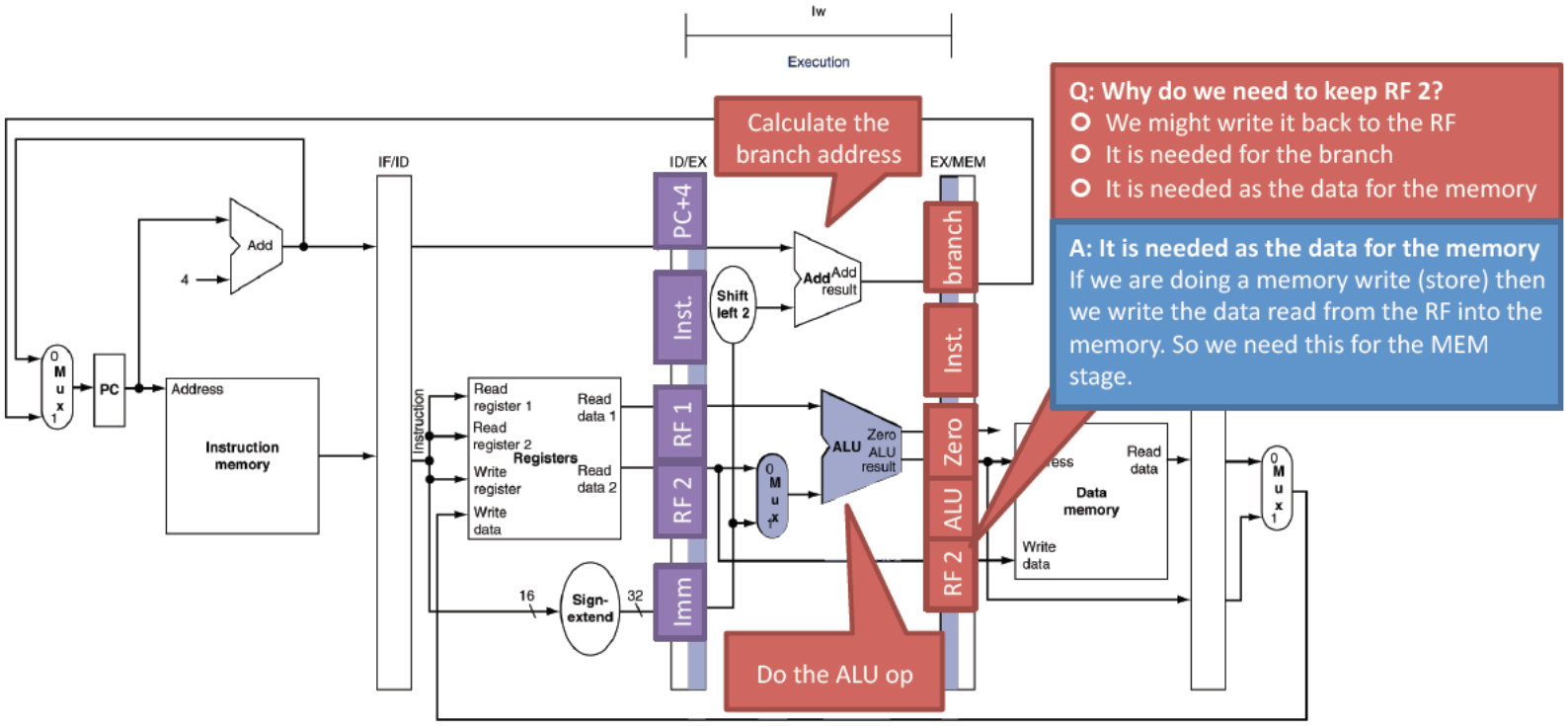
A: All of the above

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


lw
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
PC+4
Inst.
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Read from the RF
RF 1
RF 2
Imm
16
Sign-extend
32
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
Sign-extend the immediate field from the instruction

EX for load



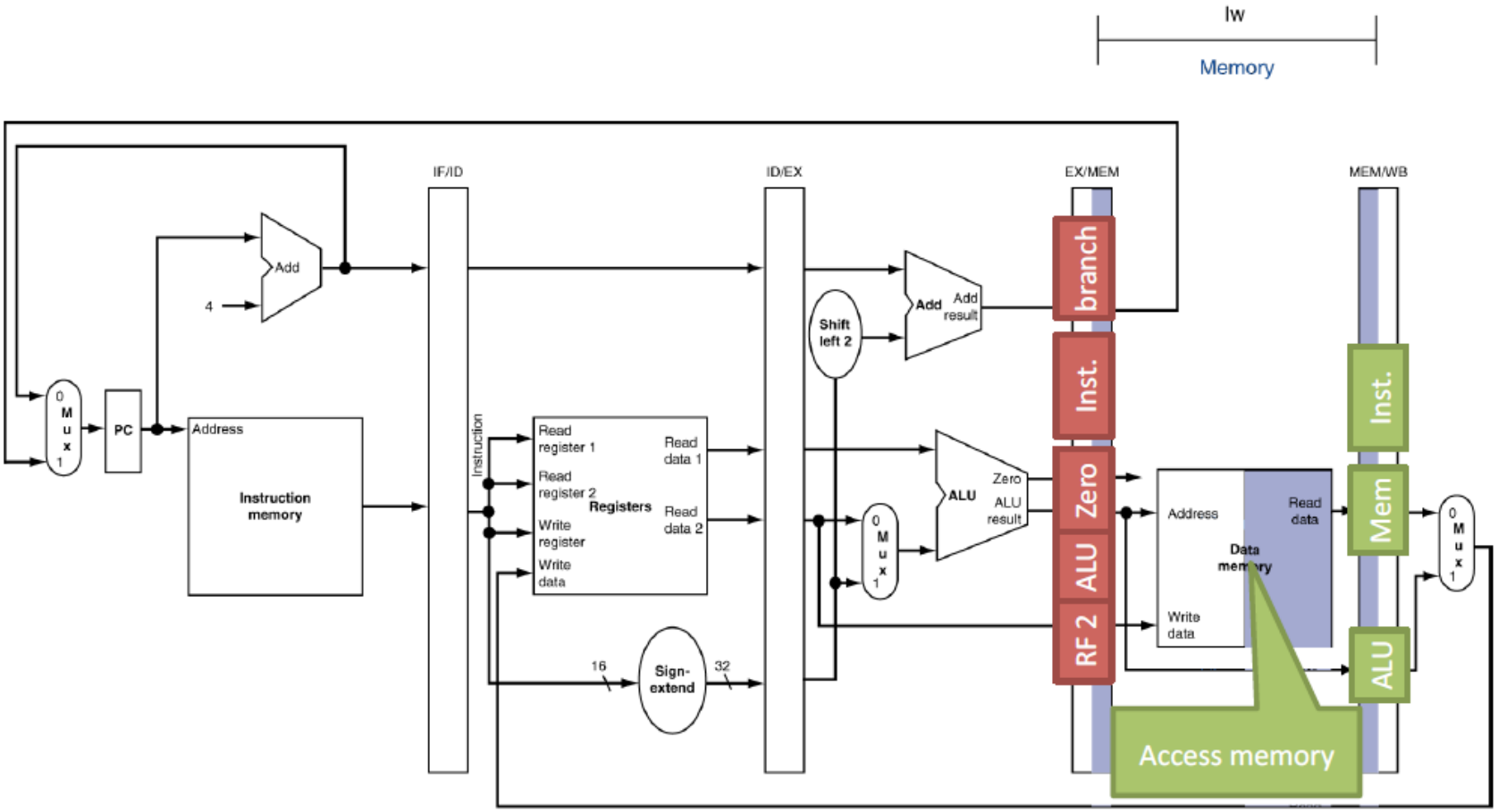
Q: Why do we need to keep RF 2?

- We might write it back to the RF
- It is needed for the branch
- It is needed as the data for the memory

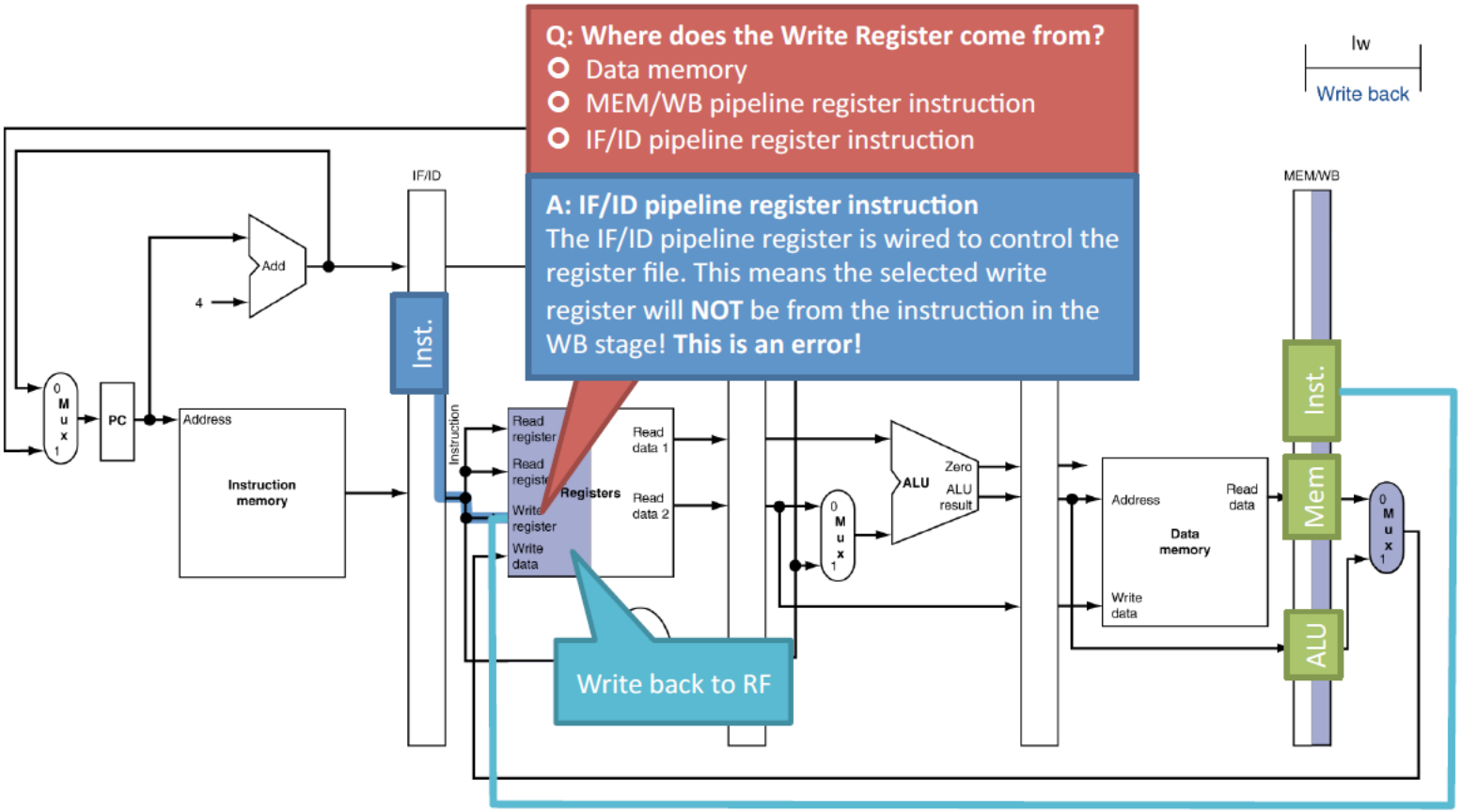
A: It is needed as the data for the memory

If we are doing a memory write (store) then we write the data read from the RF into the memory. So we need this for the MEM stage.

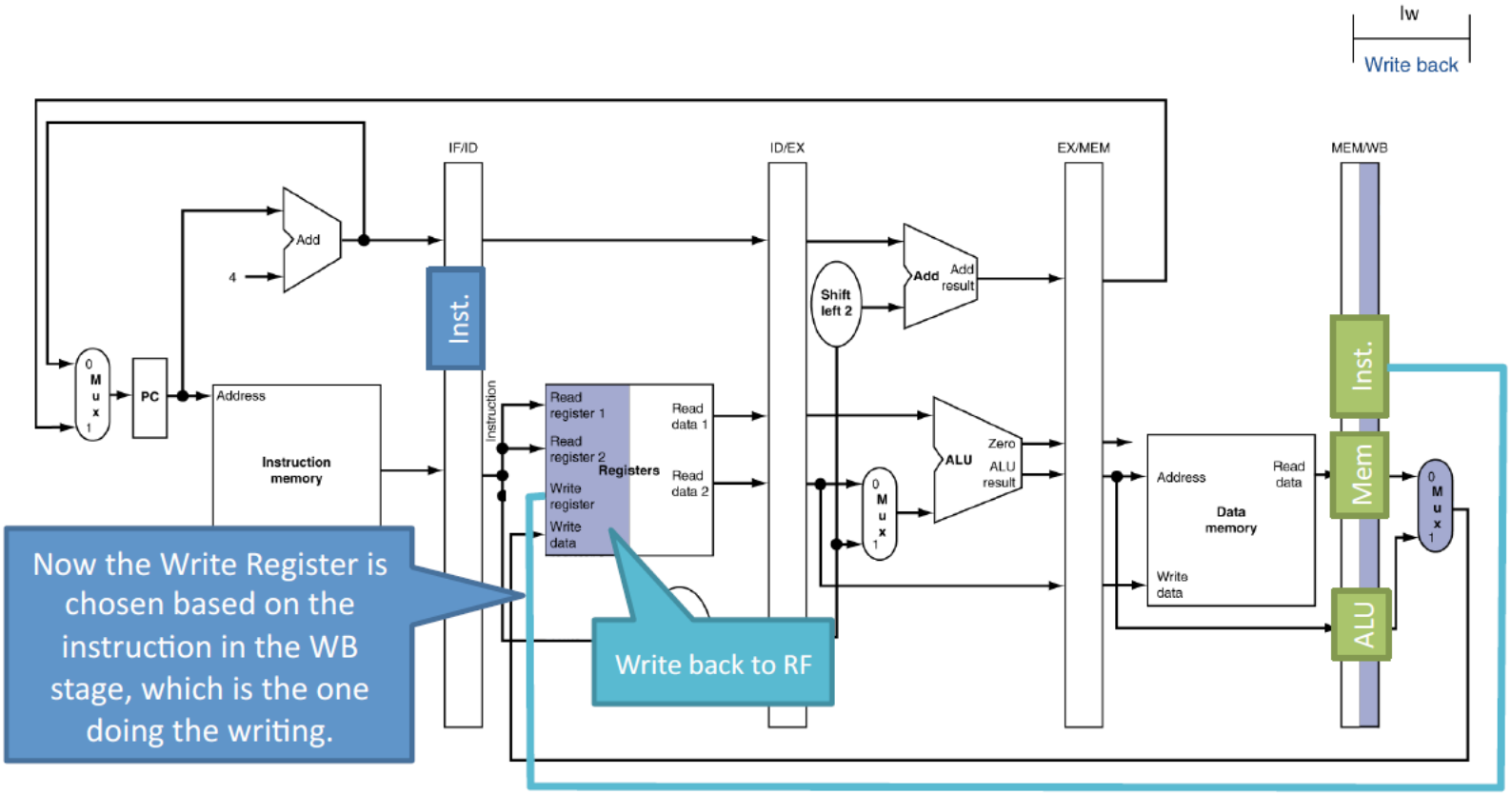
MEM for load


lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
branch
Inst.
Zero
ALU
RF 2
Address
Read data
Data memory
Write data
Mem
Inst.
ALU
16
Sign-extend
32
Access memory

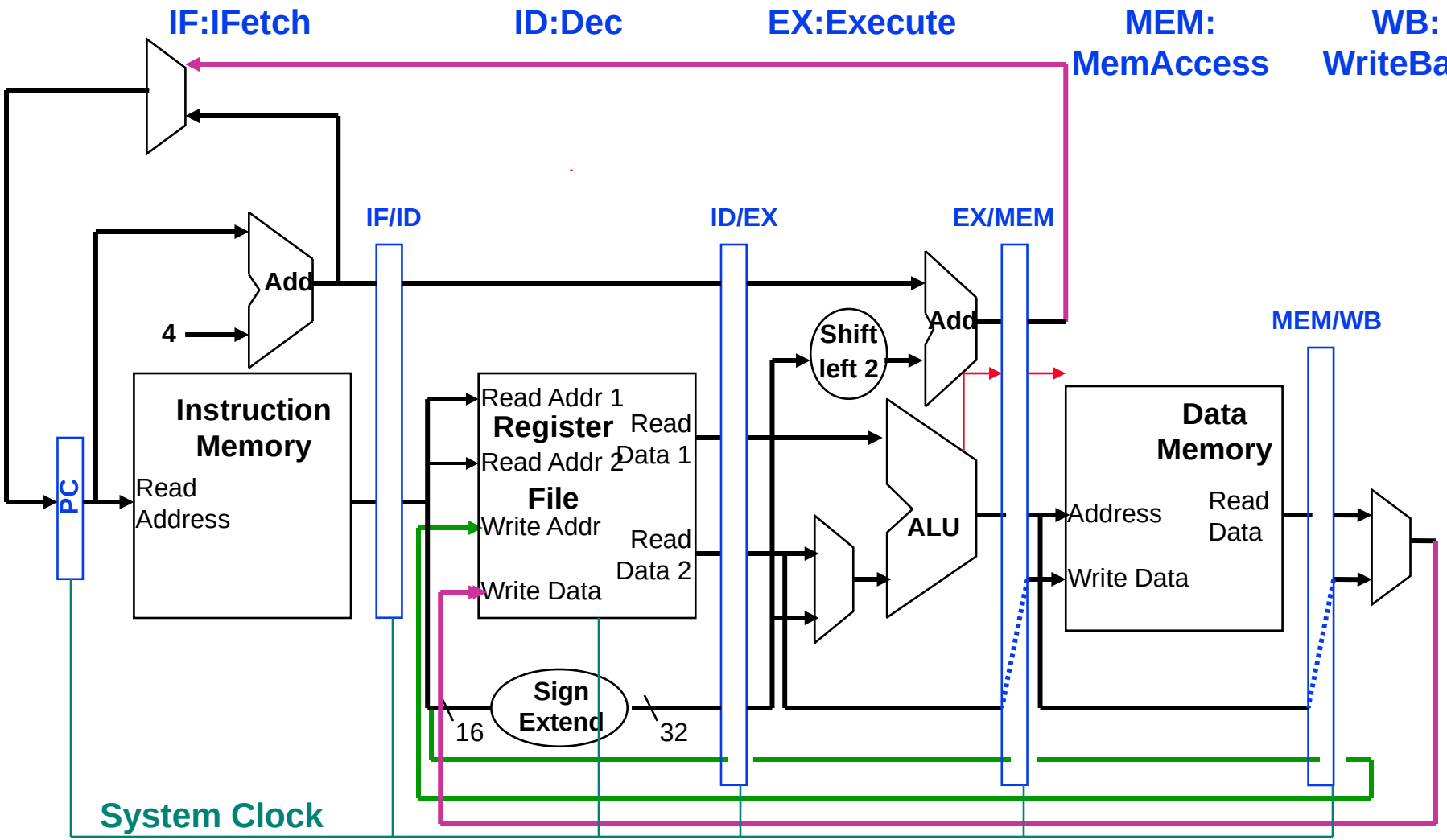
WB for load


Q: Where does the Write Register come from?
Data memory
MEM/WB pipeline register instruction
IF/ID pipeline register instruction
A: IF/ID pipeline register instruction
The IF/ID pipeline register is wired to control the register file. This means the selected write register will NOT be from the instruction in the WB stage! This is an error!
lw
Write back
IF/ID
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Inst.
Instruction
Read register
Read register
Write register
Write data
Registers
Read data 1
Read data 2
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
Inst.
Mem
ALU
Write back to RF

Fixing the WB stage

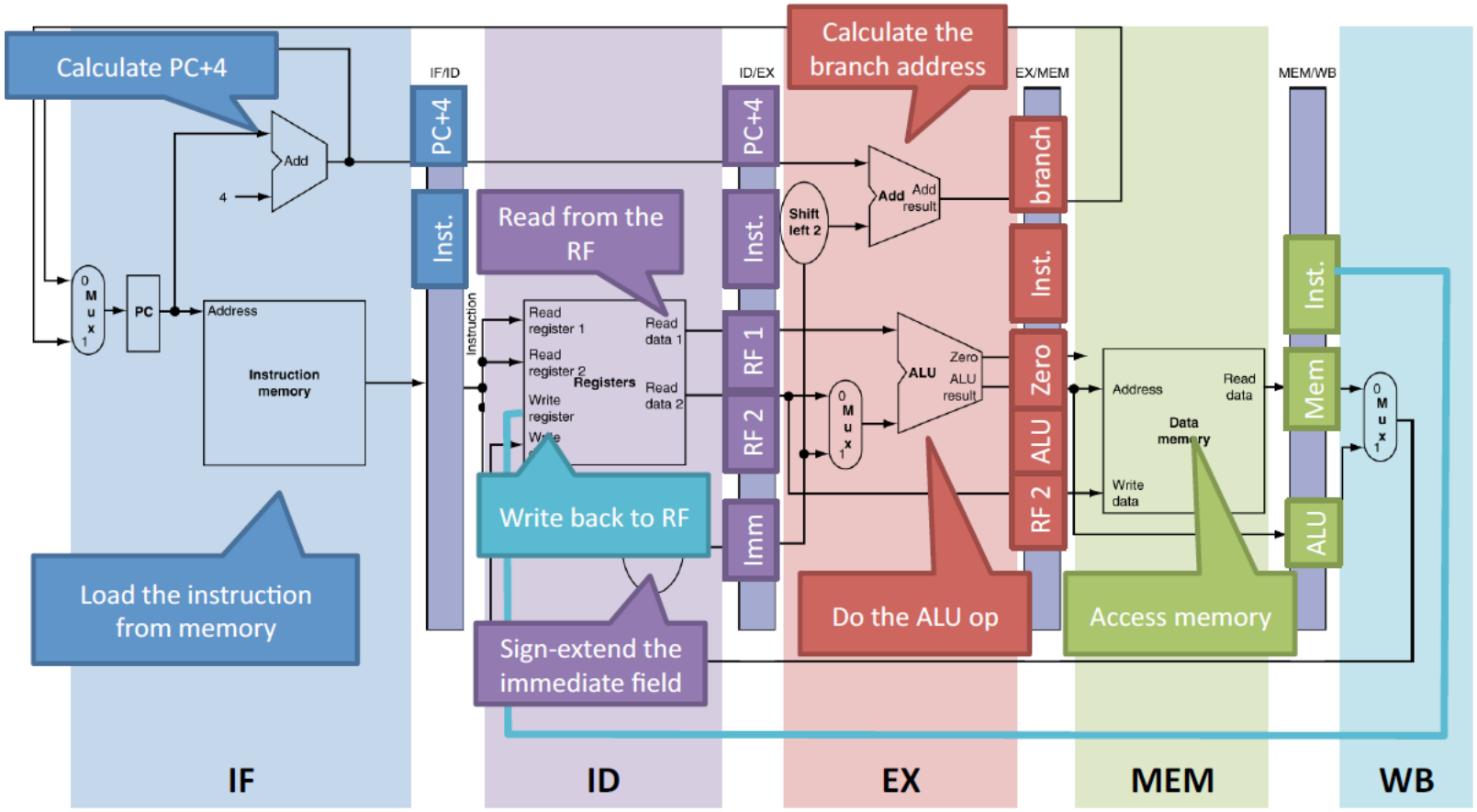


Đường dữ liệu MIPS pipeline

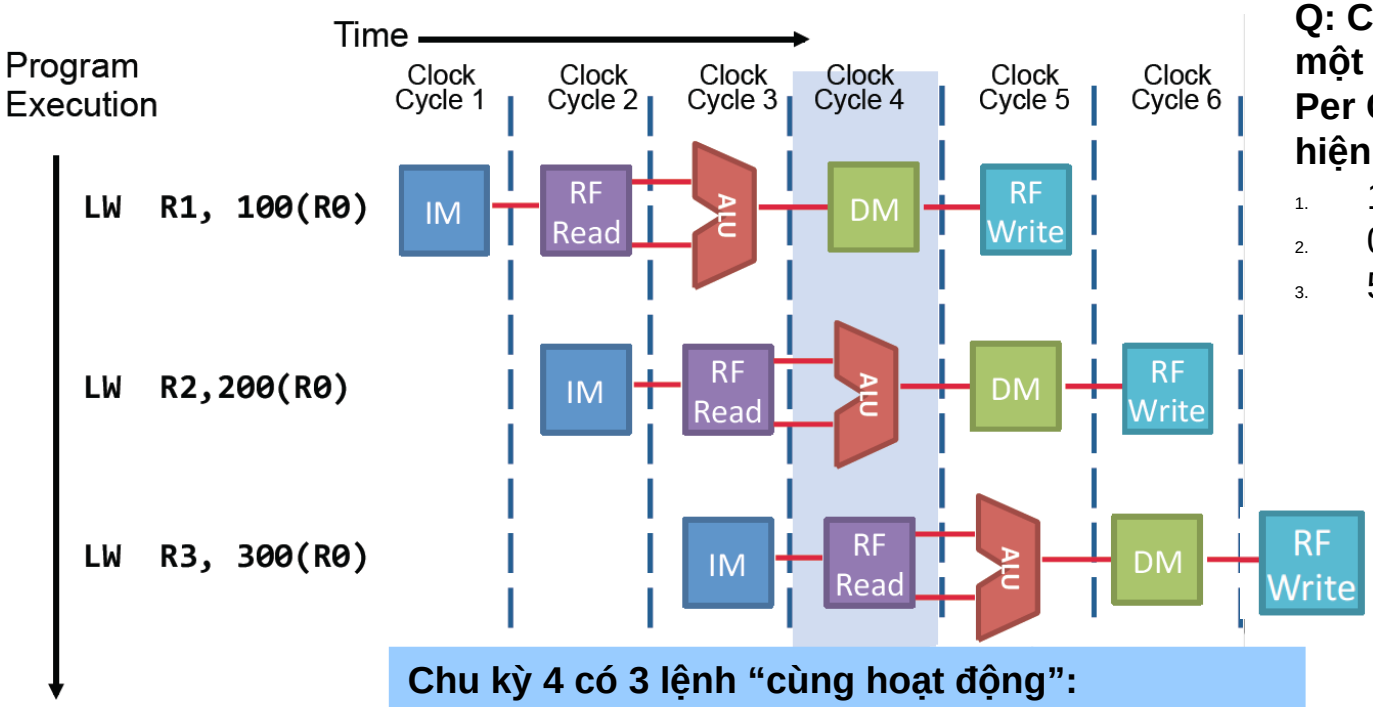


- Thanh ghi trạng thái giữa các giai đoạn thực hiện lệnh để phân cách

The MIPS pipeline



Luồng lệnh trong đường ống.



Chu kỳ 4 có 3 lệnh "cùng hoạt động":
Inst 1 is accessing the data memory (MEM)
Inst 2 is using the ALU (EX)
Inst 3 is access the register file (ID)

Q: Có bao nhiêu lệnh trên một chu kỳ - Instructions Per Cycle (IPC) nếu thực hiện lệnh tải?

1. 1.0
2. 0.2 (one every 5 cycles)
3. 5.0

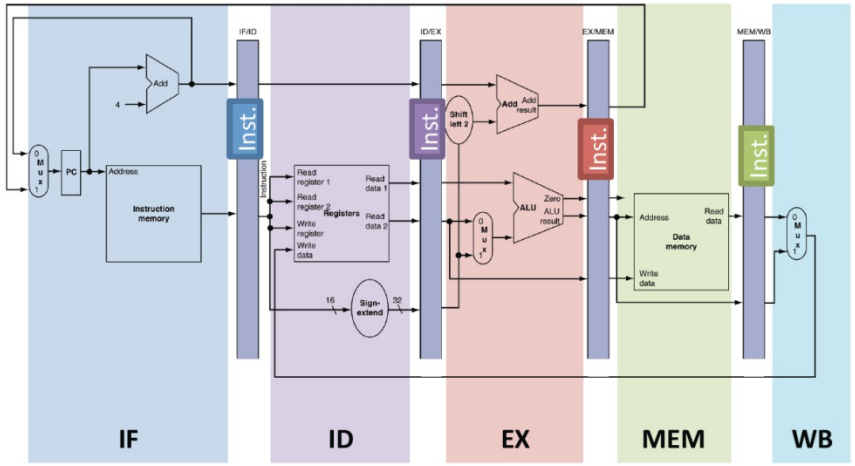
A: 1.0
Khi đường ống đầy, chỉ nhận được một lệnh mỗi chu kỳ
IPC = 1.0.

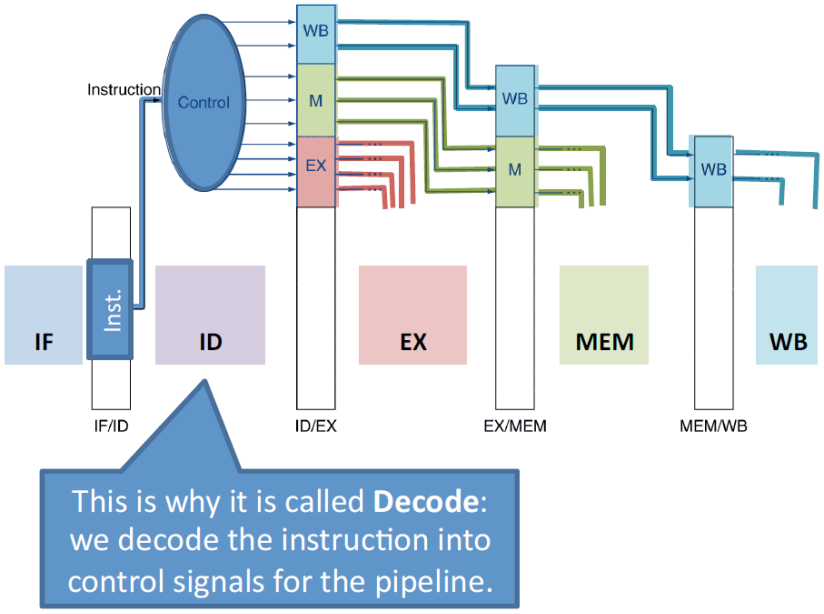
Điều khiển logic trong Pipeline

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

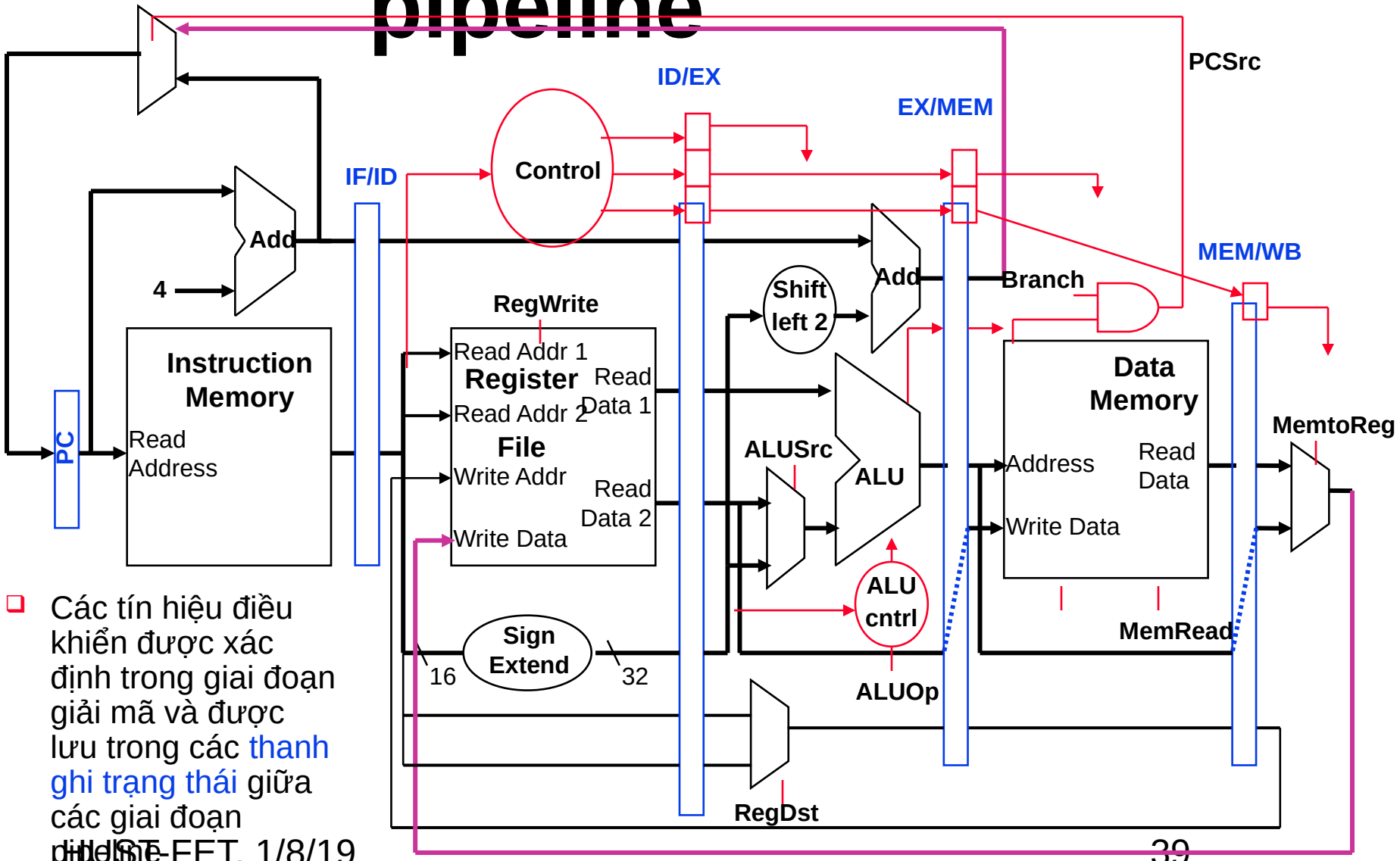
Điều khiển Pipeline

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





Điều khiển MIPS pipeline

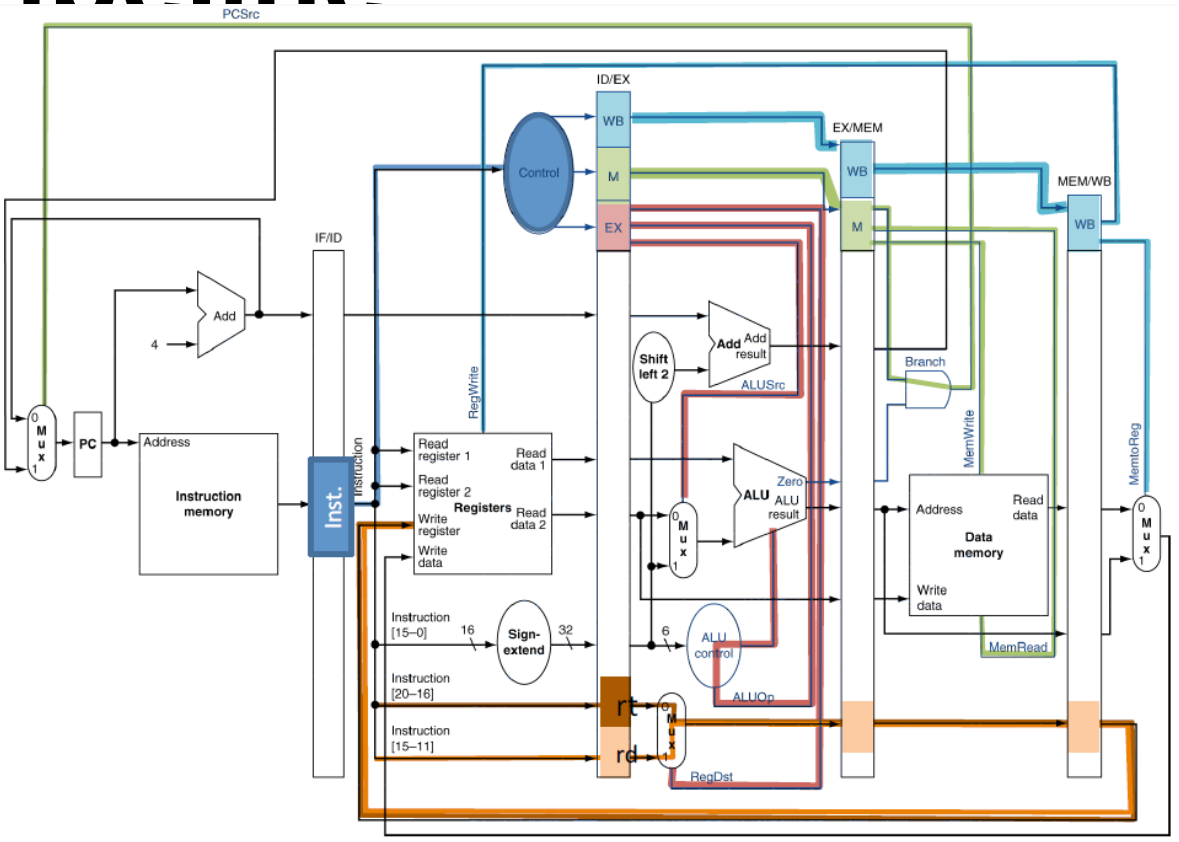


- Các tín hiệu điều khiển được xác định trong giai đoạn giải mã và được lưu trong các thanh ghi trạng thái giữa các giai đoạn pipeline

Chi tiết về điều khiển Pipeline

Q: Nơi nào tín hiệu Write Register đi đến?

1. The MEM/WB control bits (top)
2. instruction in the IF/ID register
3. Data in the MEM/WB register



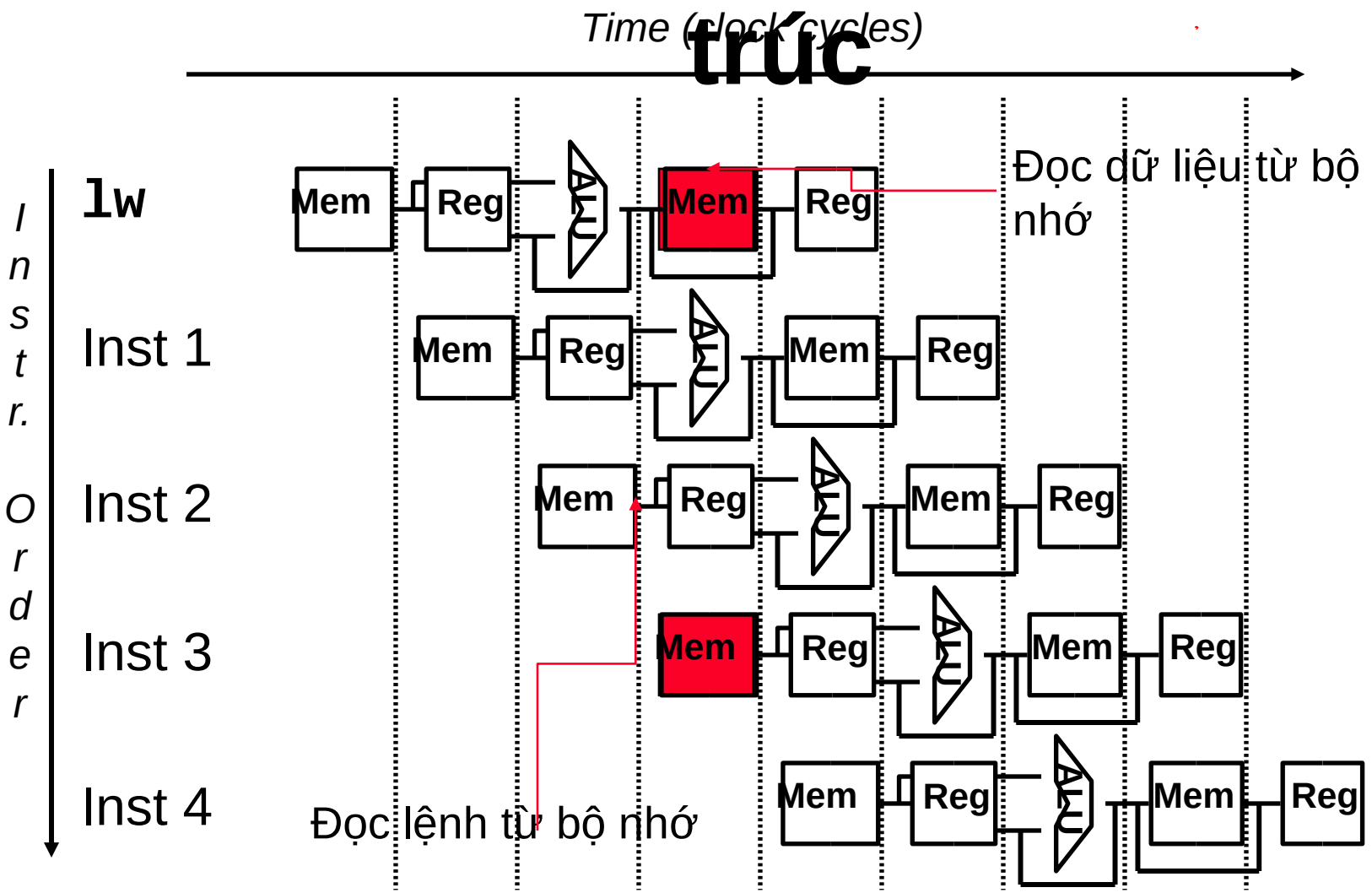
A: Data in the MEM/WB register

Các bit thứ tự 20-16 hoặc 11-15 được gửi tới thanh ghi MEM/WB để xác định thanh ghi cần ghi dữ liệu

Xung đột Pipeline

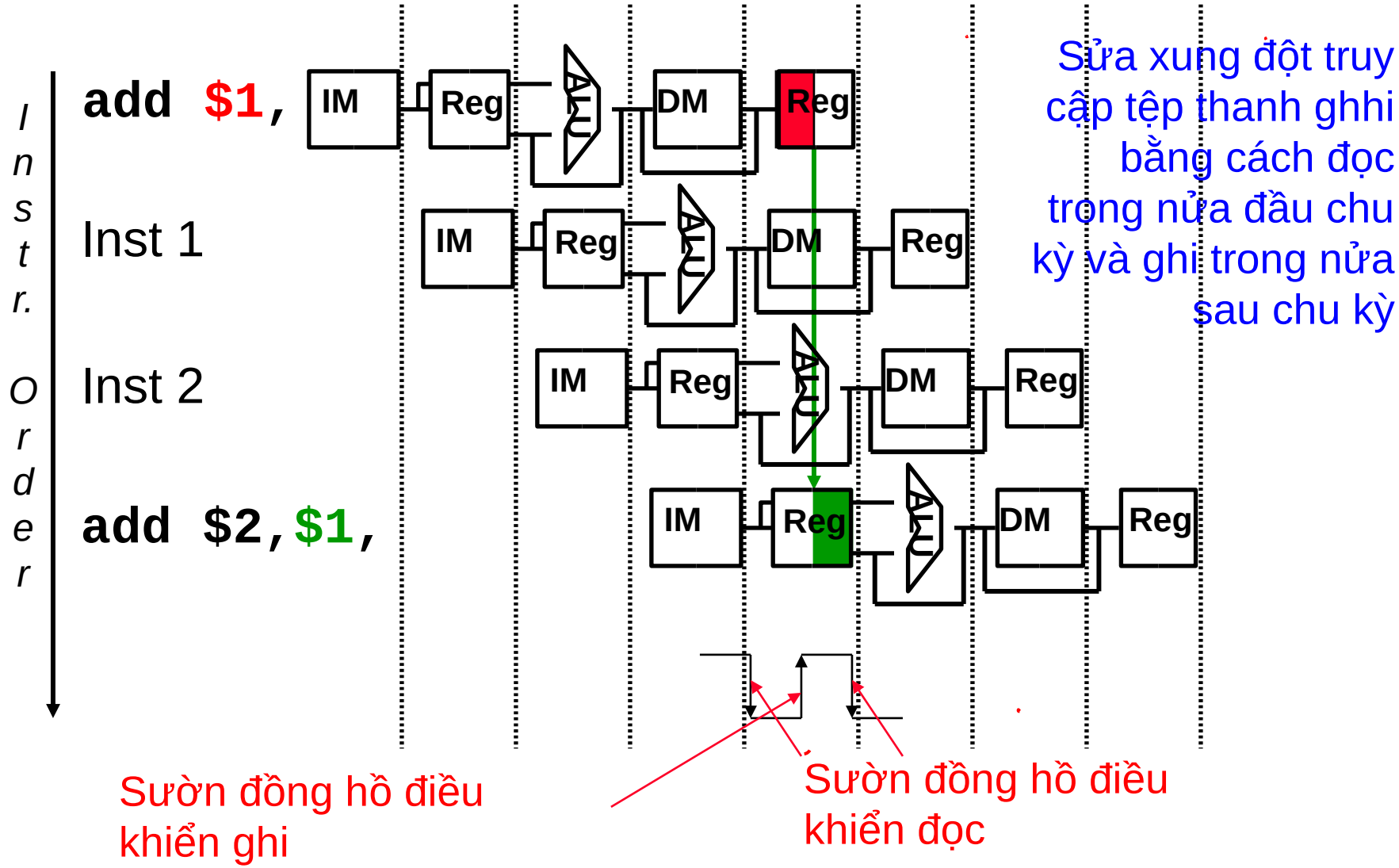
- **Xung đột cấu trúc:** yêu cầu sử dụng cùng một tài nguyên cho 2 lệnh khác nhau tại cùng 1 thời điểm
- **Xung đột dữ liệu:** yêu cầu sử dụng dữ liệu trước khi nó sẵn sàng
  - Các toán hạng nguồn của 1 lệnh được tạo ra bởi lệnh phía trước vẫn đang nằm trong pipeline
- **Xung đột điều khiển:** yêu cầu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

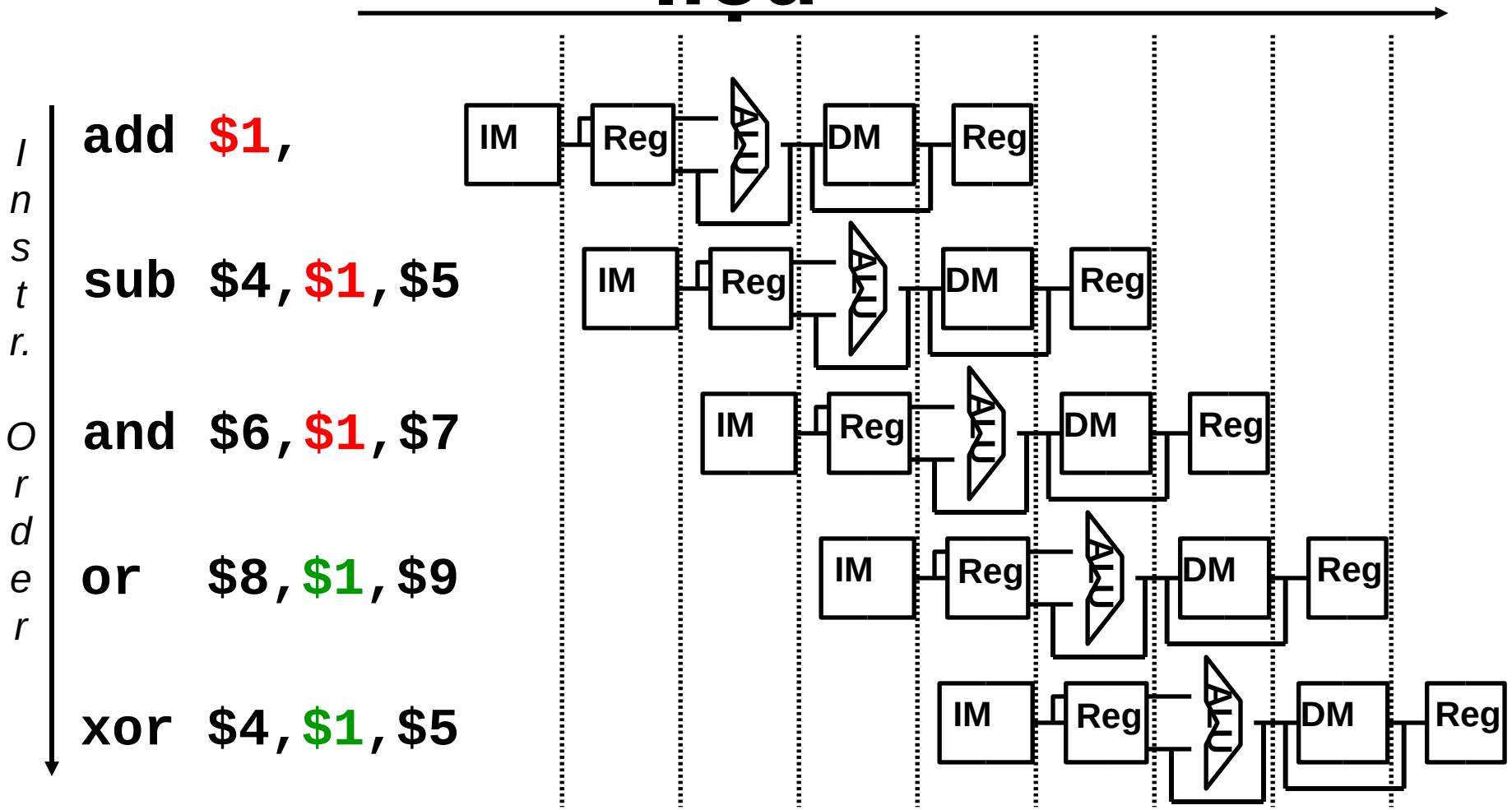


- Sửa: Bộ nhớ dữ liệu và lệnh riêng rẽ(Instr. and Data)

Truy cập tệp thanh ghi

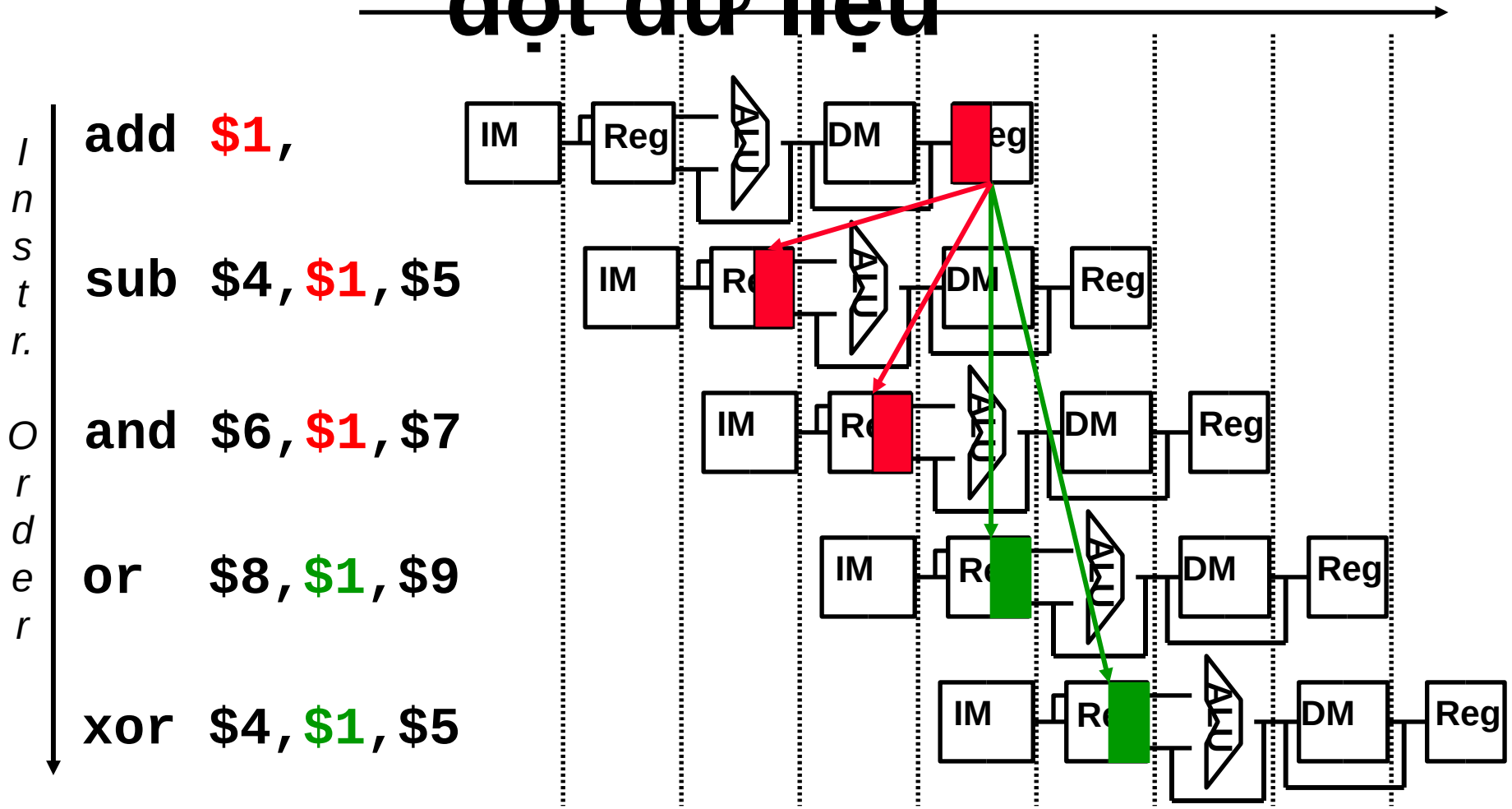


Sử dụng thanh ghi: Xung đột dữ liệu



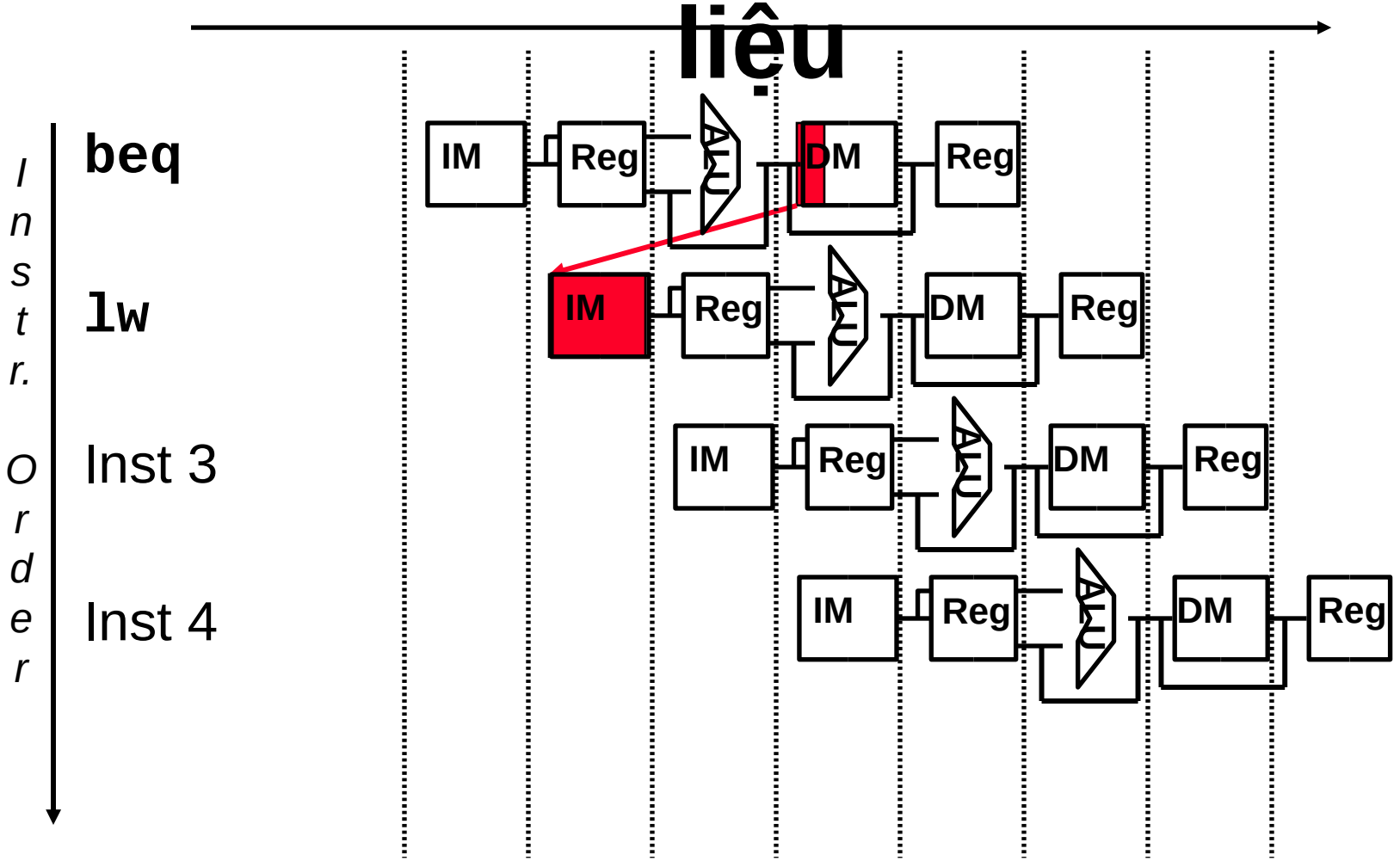
- Xung đột đọc trước khi ghi (Read before write)
- Phụ thuộc dữ liệu ngược theo thời gian gây ra xung đột

Sử dụng thanh ghi: Xung đột dữ liệu



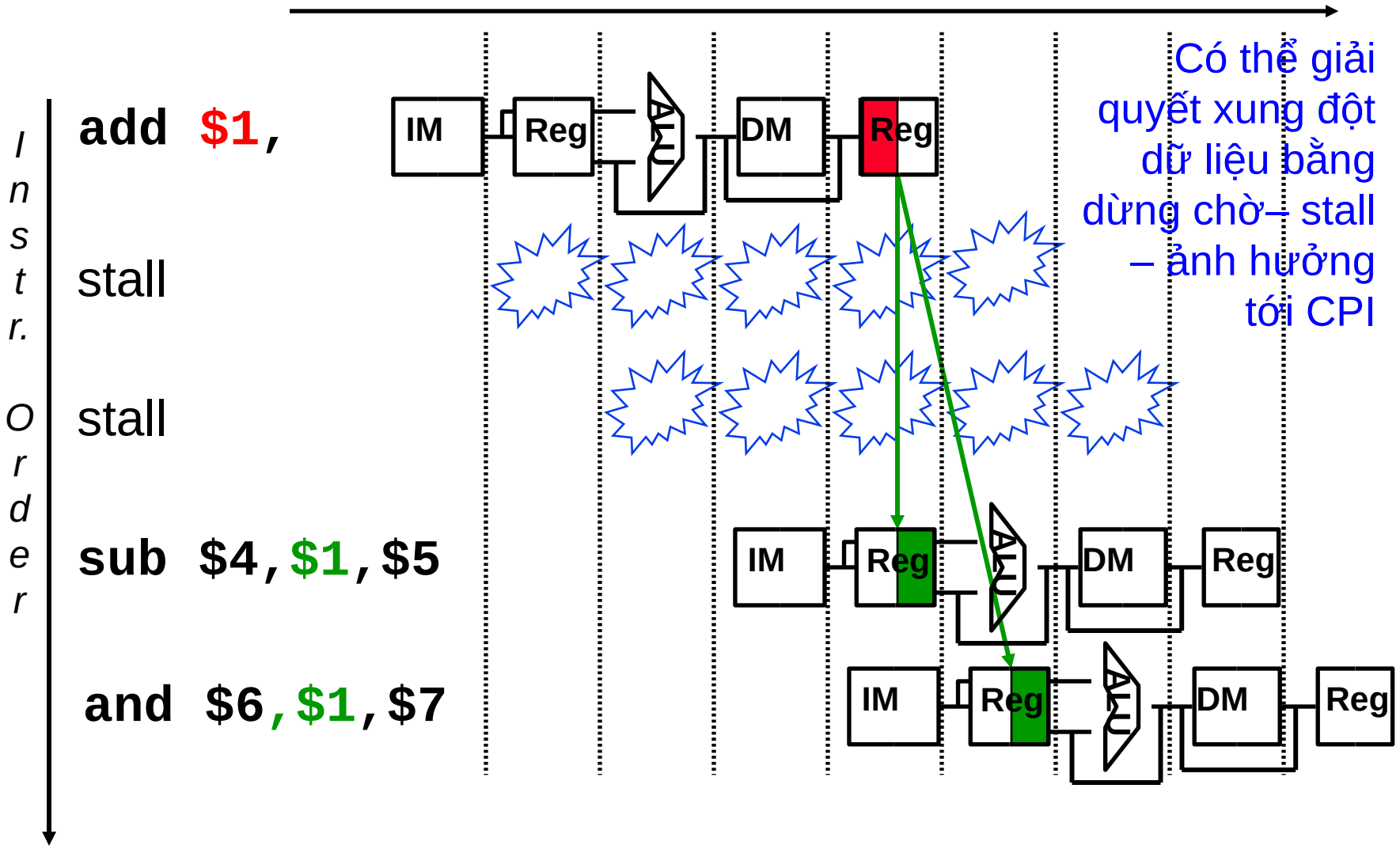
- Xung đột đọc trước khi ghi (Read before write)
- Phụ thuộc dữ liệu ngược theo thời gian gây ra xung đột

Đọc từ bộ nhớ gây xung đột dữ liệu



- Dependencies backward in time cause hazards

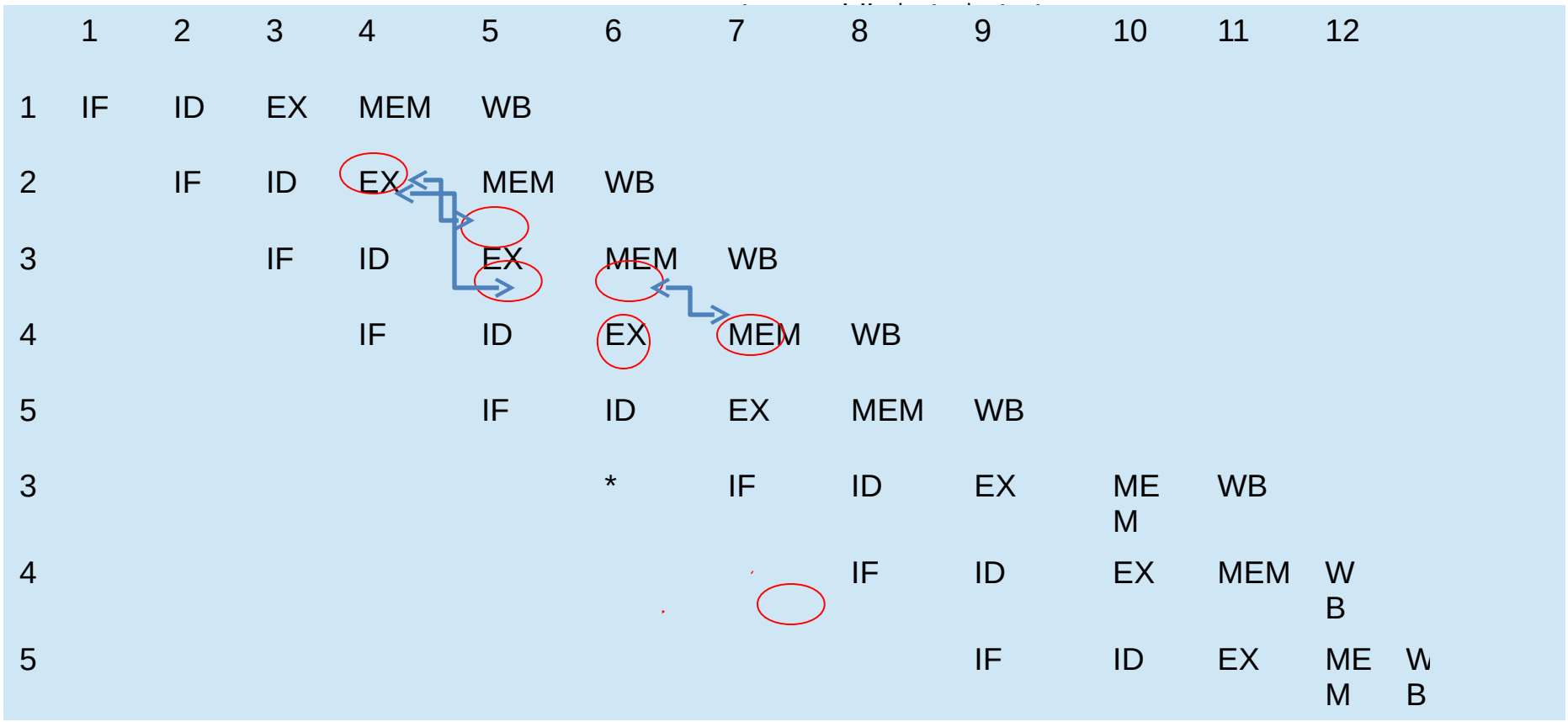
Giải quyết xung đột: Tạm dừng



Ví dụ:

Tính CPI cho chương trình

```
1:    addi  $s0, $zero, 10
2:    addi  $s1, $zero, 0
L1:
3:    add   $t0, $t0, $s1
```

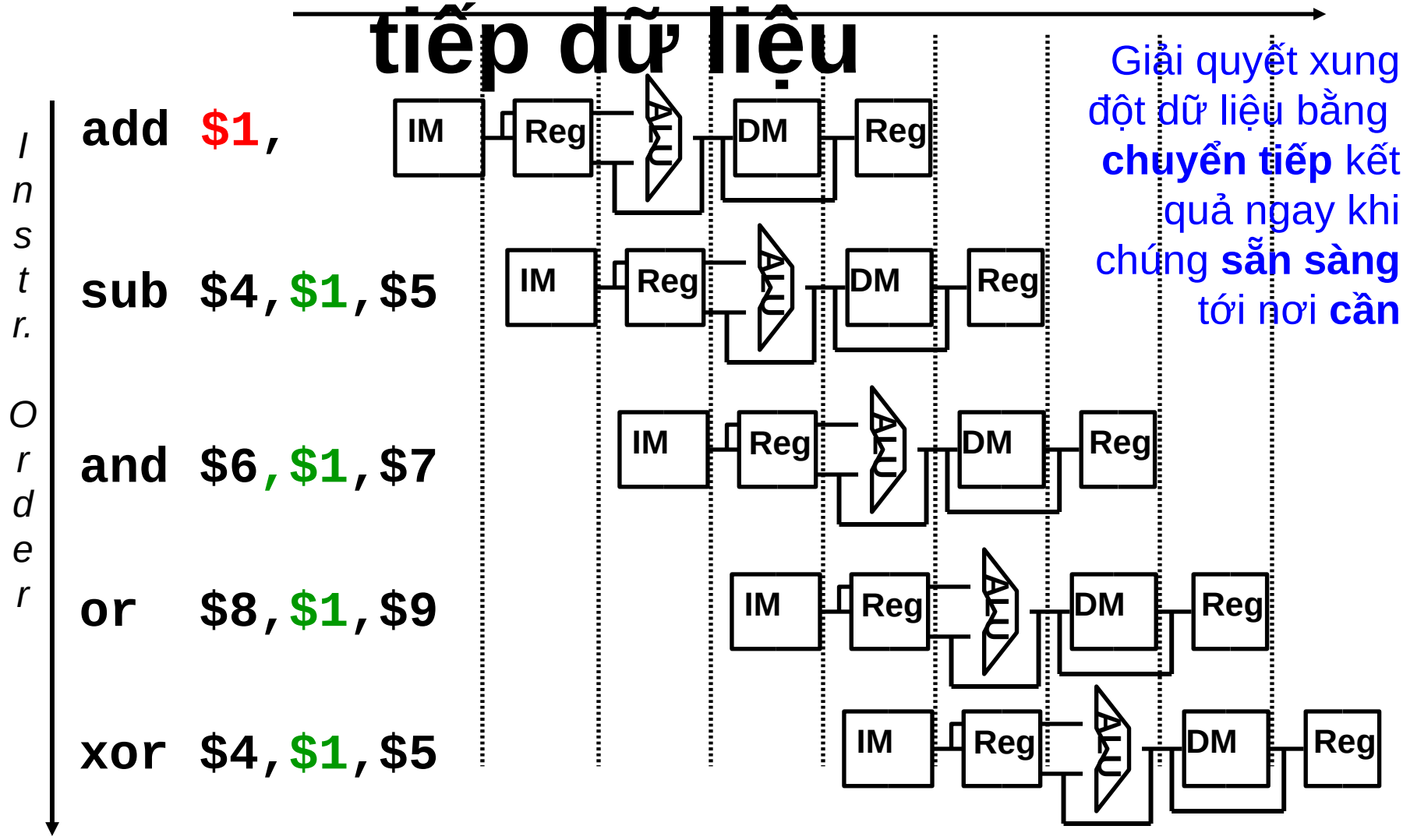


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	IF	ID	EX	MEM	WB										
2		IF	ID	EX	MEM	WB									
3			*	*	IF	ID	EX	MEM	WB						
4						IF	ID	EX	MEM	WB					
5							*	*	IF	ID	EX	MEM	WB		
3										*	*	*	*	IF	ID
4															IF
5															

Chuyển tiếp dữ liệu

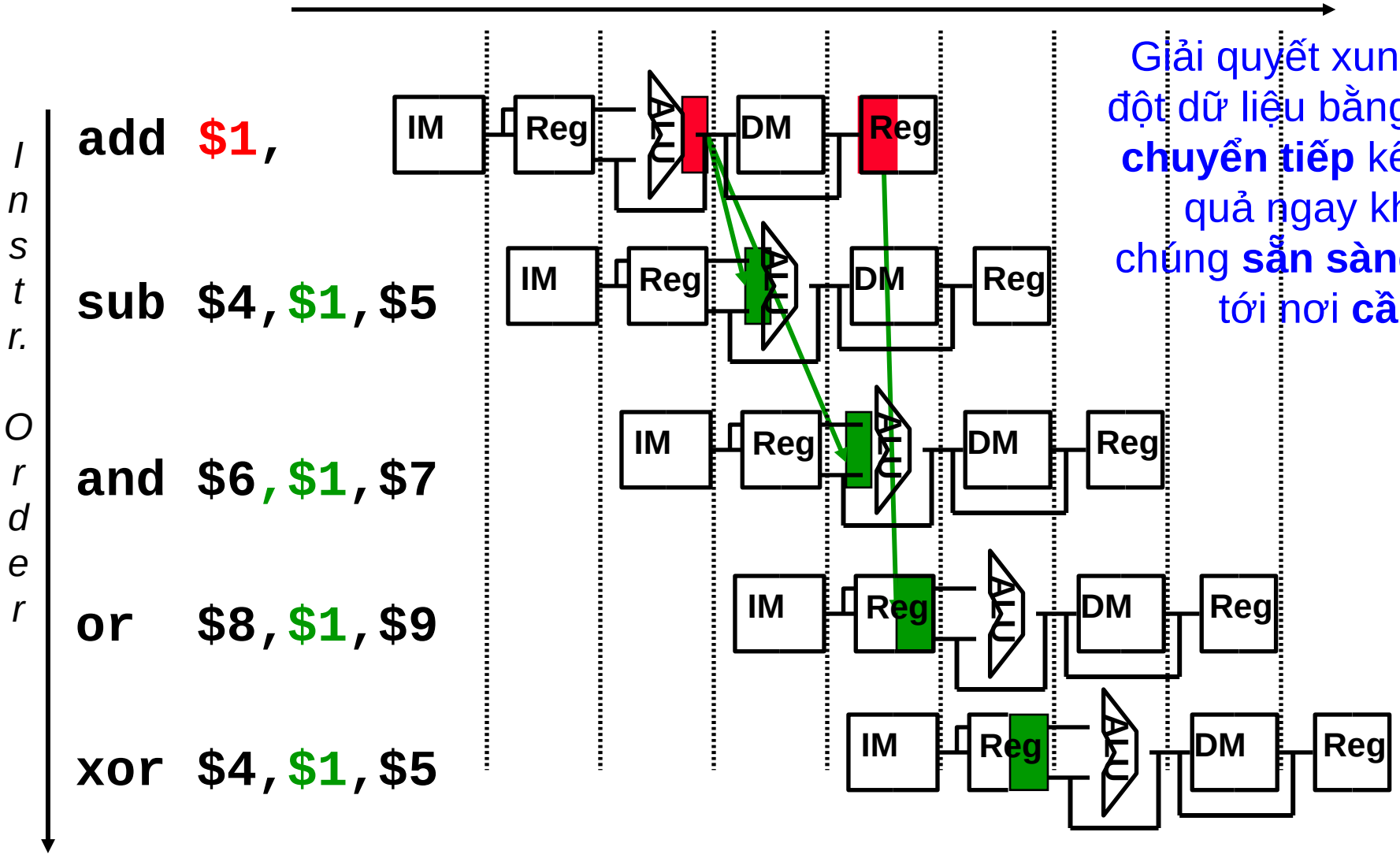
- Lấy kết quả ở thời điểm nó xuất hiện sớm nhất trong bất kỳ thanh ghi pipeline nào, và chuyển tiếp nó đến khối chức năng (VD. ALU) mà cần kết quả tại chu kỳ đồng hồ đó
- Với khối chức năng ALU: đầu vào có thể từ bất kỳ thanh ghi pipeline nào chứ không cần từ ID/EX bằng cách
  - Thêm bộ chọn vào trước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



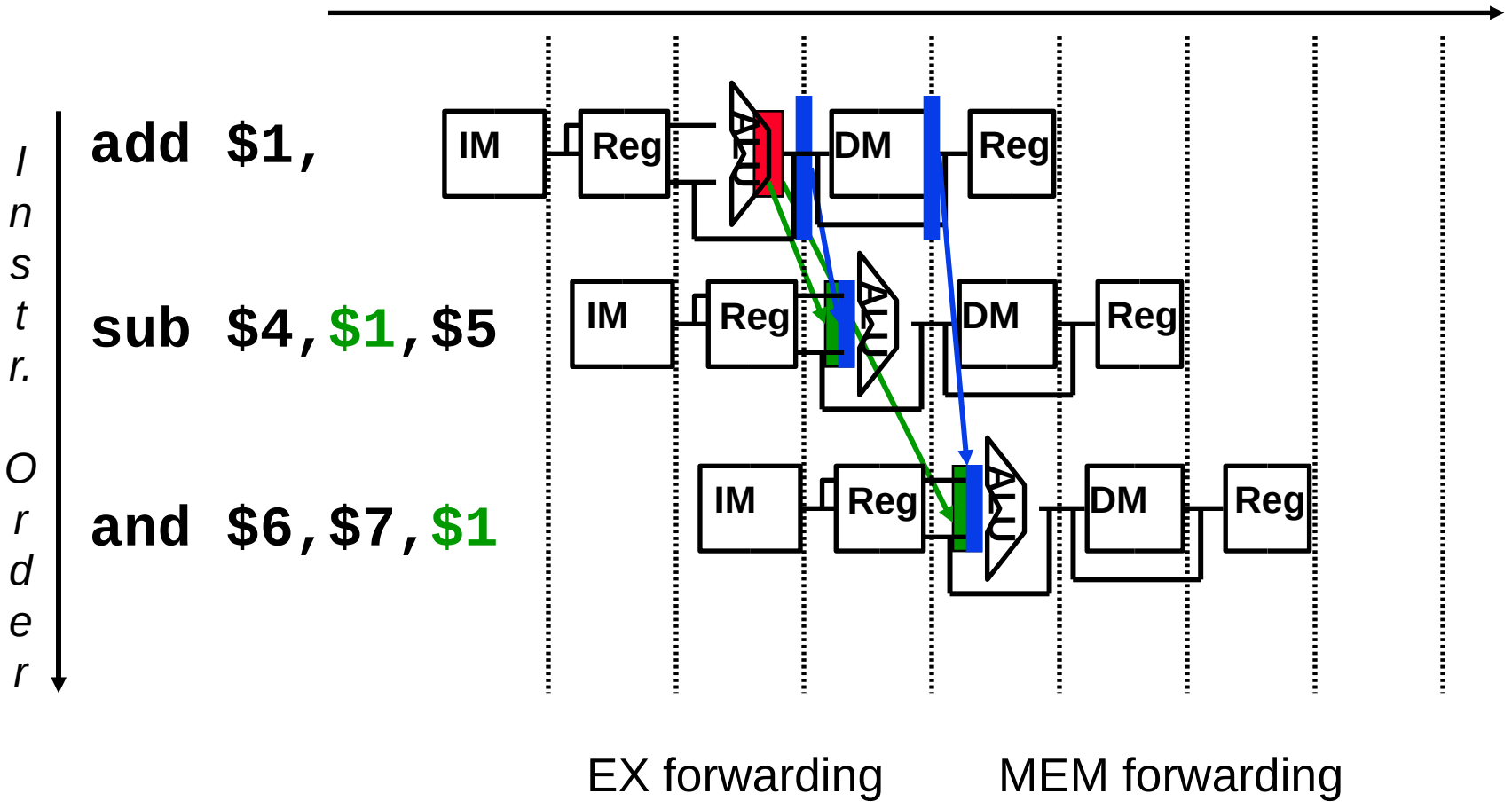
Giải quyết xung đột dữ liệu bằng **chuyển tiếp** kết quả ngay khi chúng **sẵn sàng** tới nơi **cần**

Giải quyết xung đột: Chuyển tiếp dữ liệu



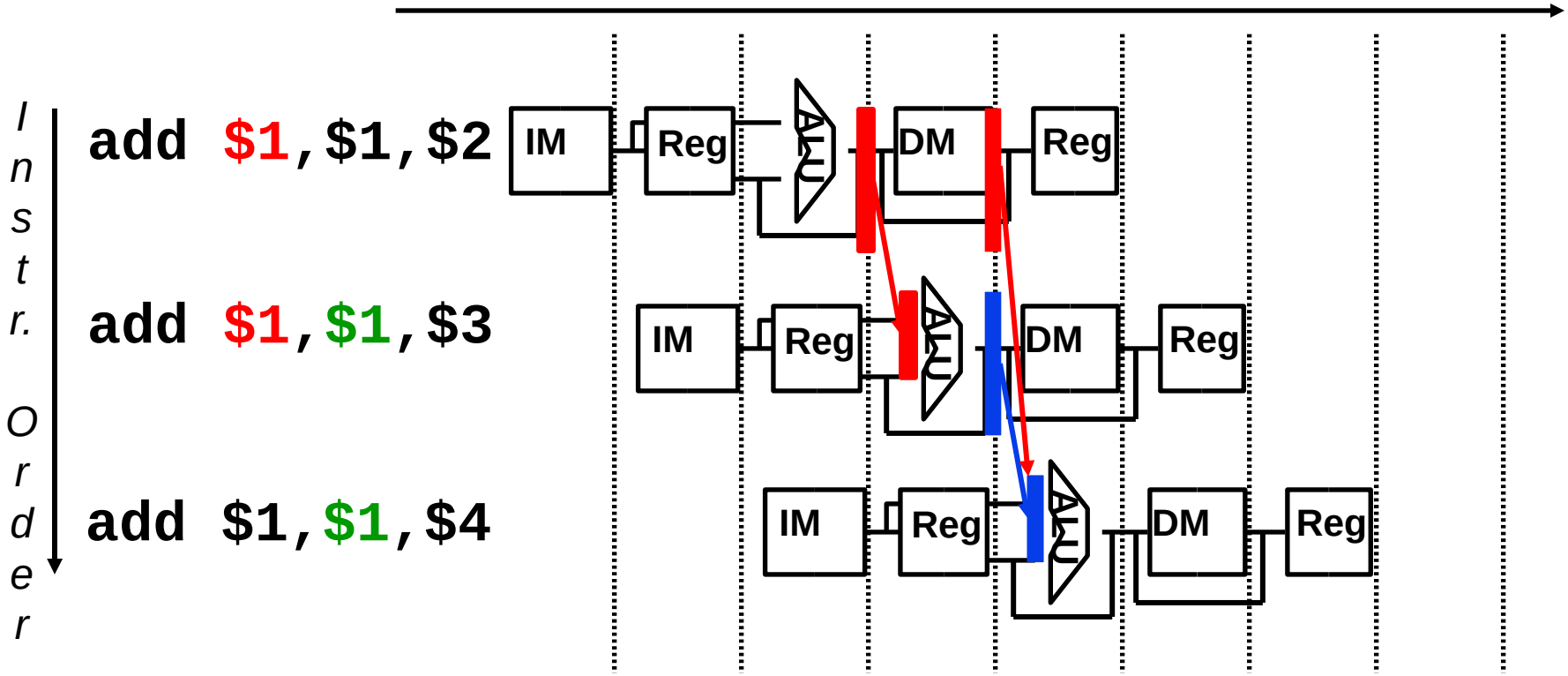
Giải quyết xung đột dữ liệu bằng **chuyển tiếp** kết quả ngay khi chúng **sẵn sàng** tới nơi **cần**

Minh họa triển khai chuyển tiếp

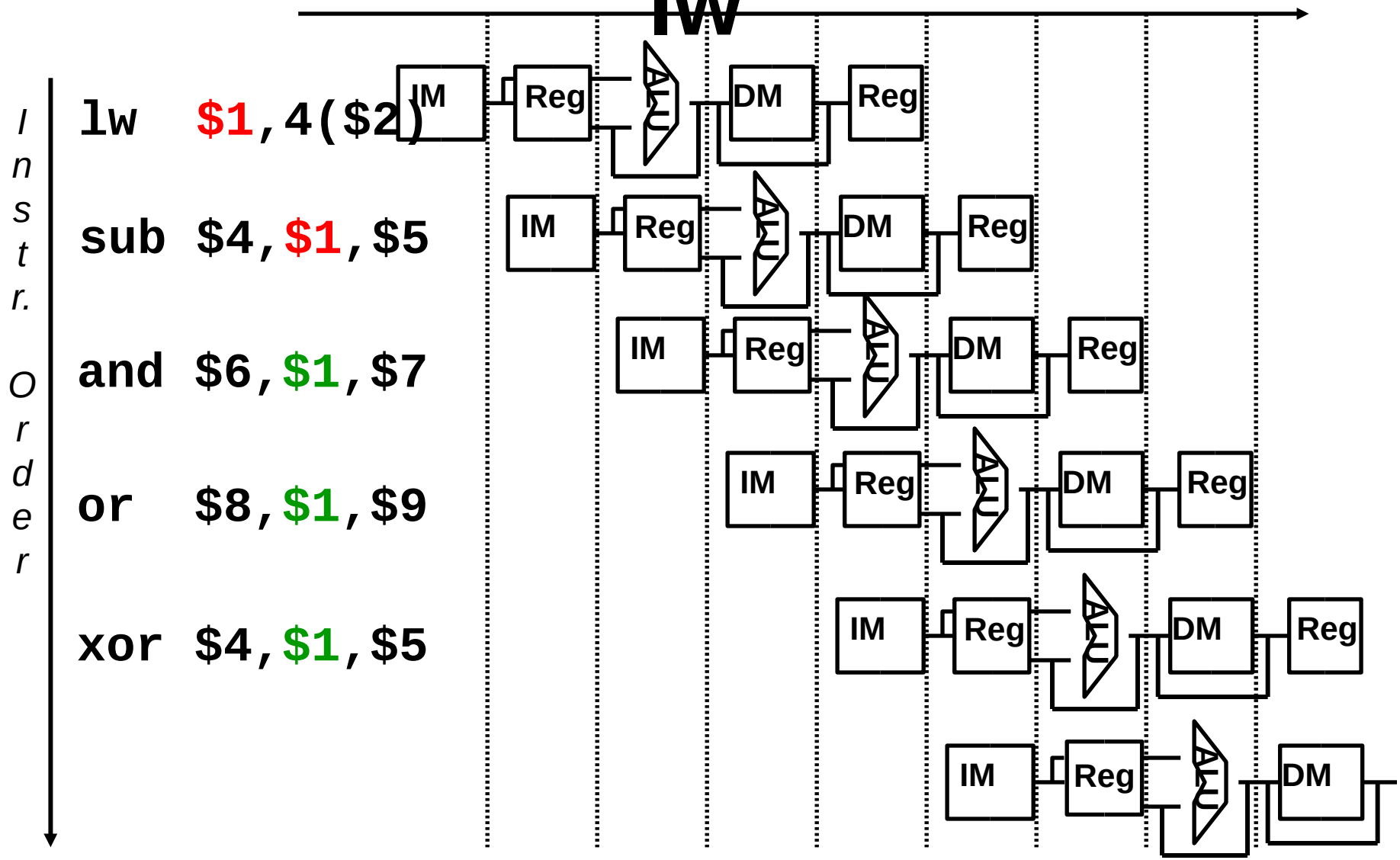


Xung đột dữ liệu khi chuyển tiếp

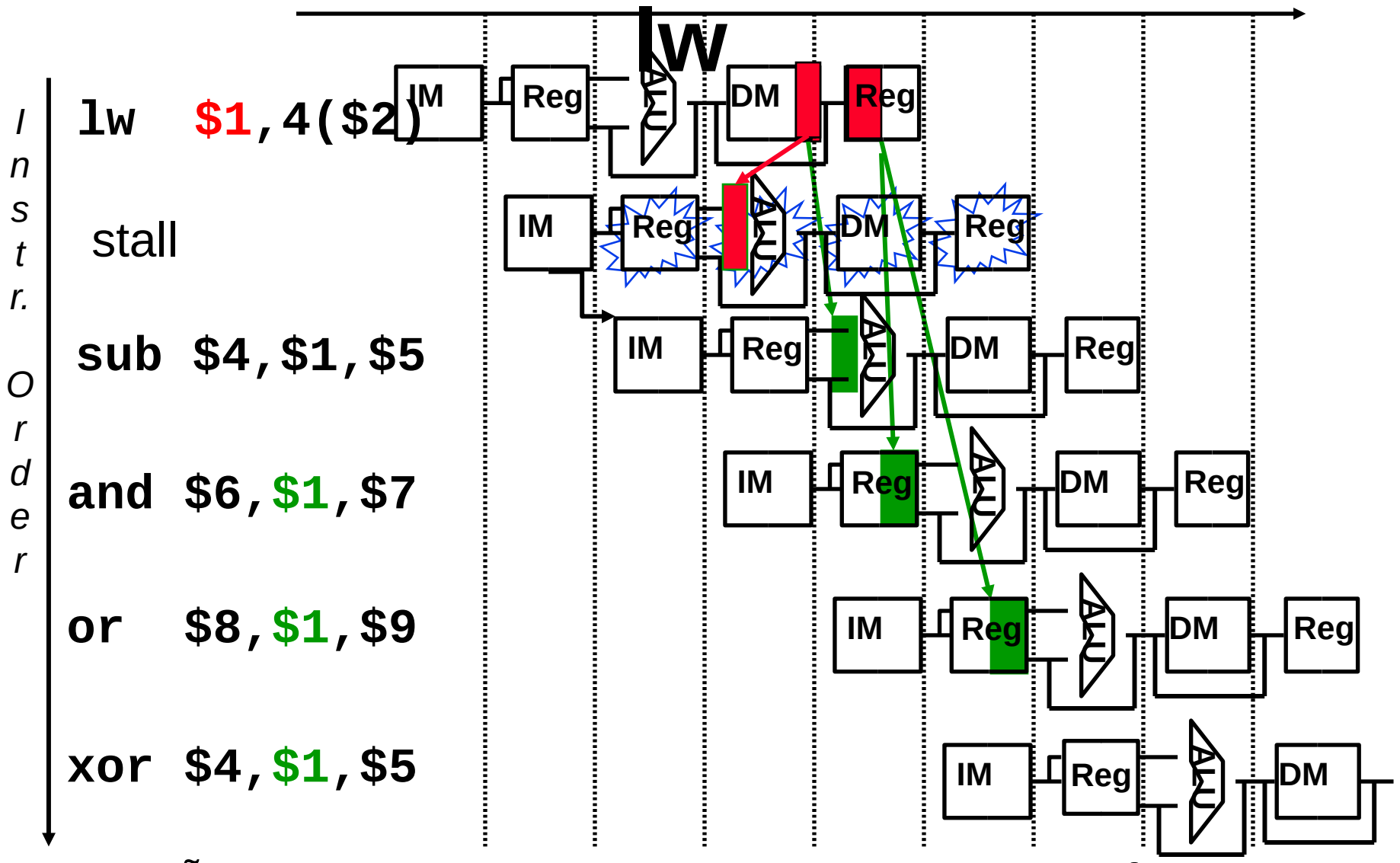
- Một loại xung đột dữ liệu xuất hiện khi chuyển tiếp: Xung đột giữa kết quả của lệnh đang ở giai đoạn WB và lệnh đang ở giai đoạn MEM – kết quả nào cần được chuyển tiếp?



Xung đột dữ liệu khi có lệnh lw



Xung đột dữ liệu khi có lệnh

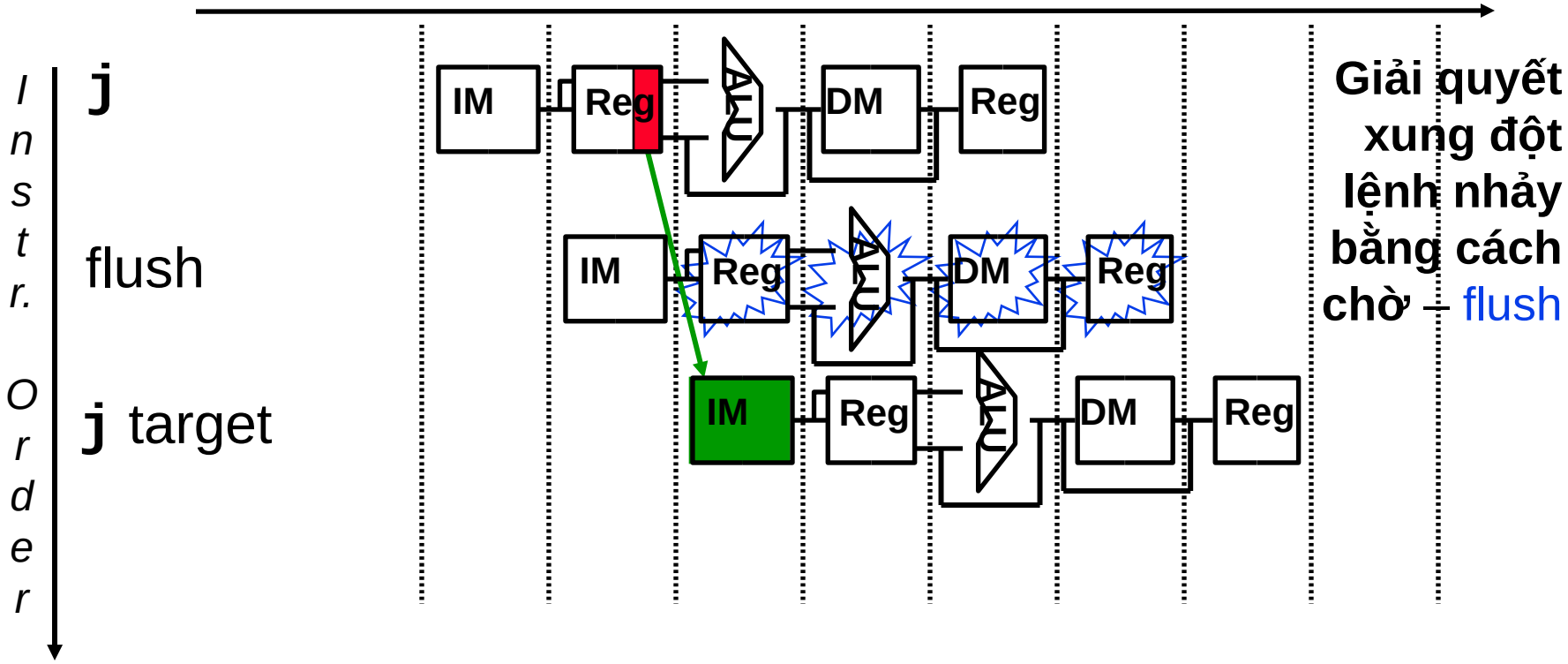


- Sẽ vẫn cần một chu kỳ chờ ngay cả khi có chuyển tiếp

Xung đột điều khiển

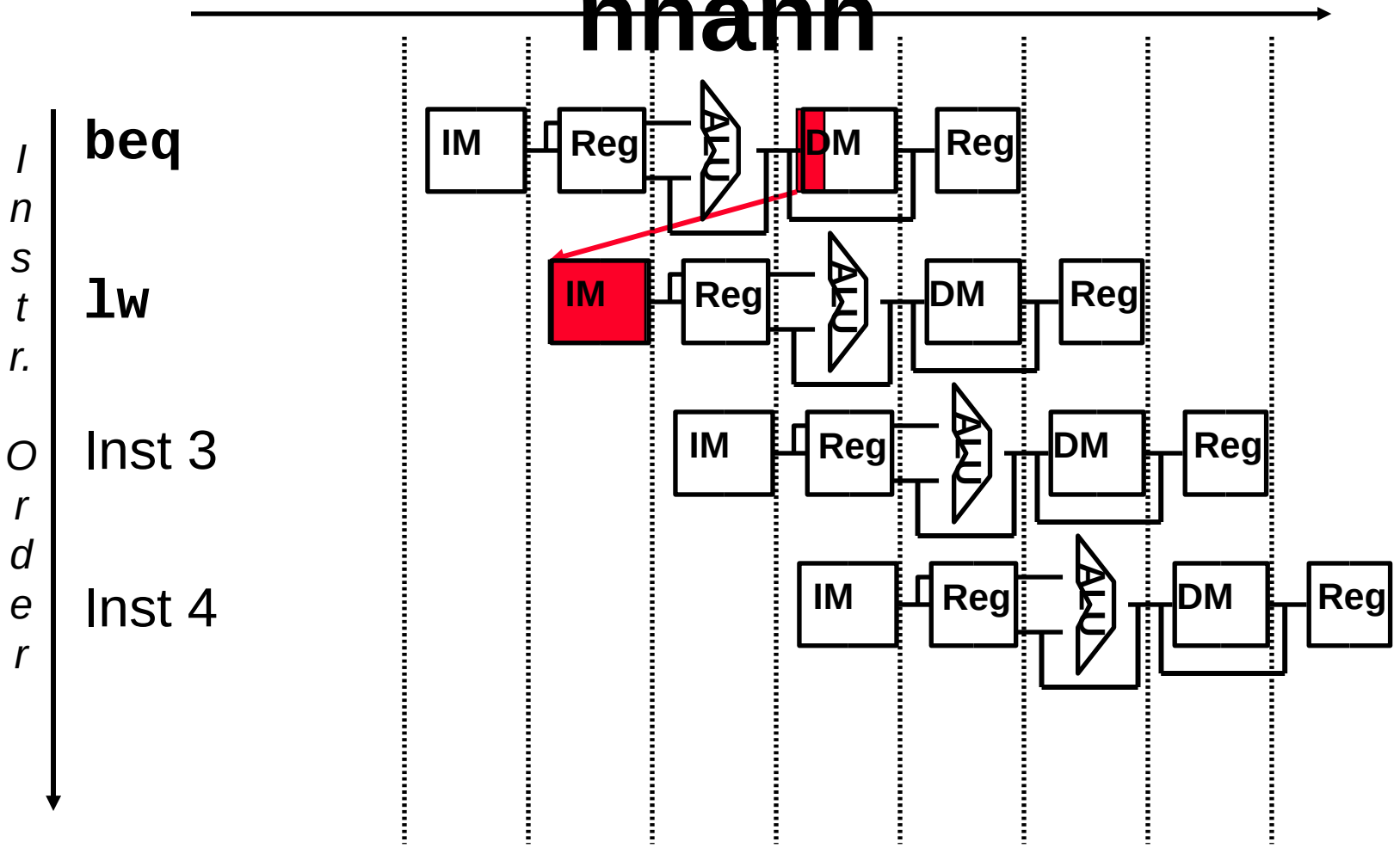
- Khi địa chỉ các lệnh không tuần tự (i.e., PC = PC + 4); xuất hiện khi có các lệnh thay đổi dòng chương trình
  - Lệnh rẽ nhánh không điều kiện (j, jal, jr)
  - Lệnh rẽ nhánh có điều kiện (beq, bne)
  - Ngắt, Exceptions
- Giải pháp
  - Tạm dừng (ảnh hưởng CPI)
  - Tín toán điều kiện rẽ nhánh càng sớm 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

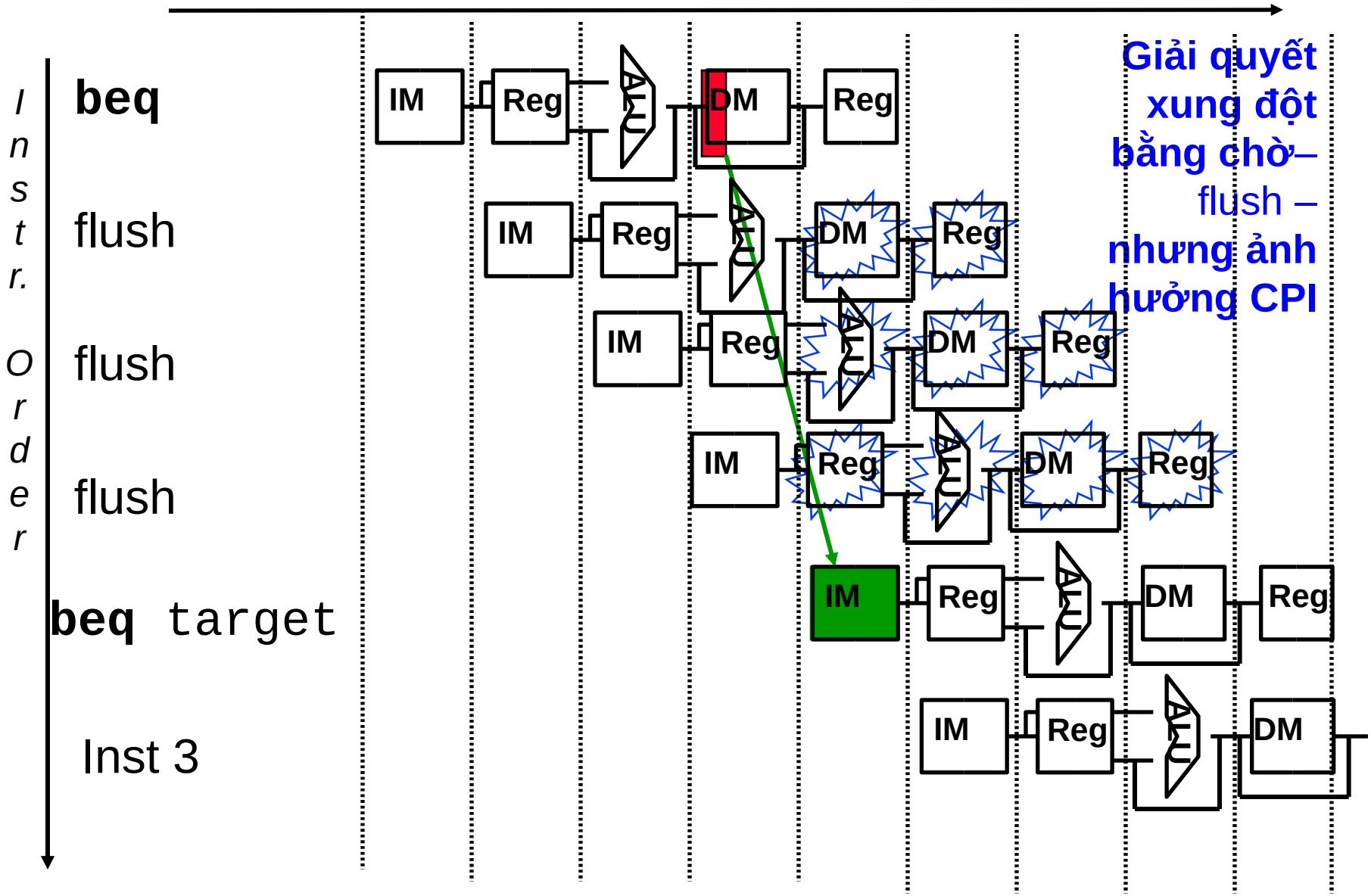


- Lệnh nhảy không được giải mã cho đến giai đoạn ID, cần một lệnh xóa (flush)
  - Để xóa, đặt trường mã lệnh của thanh ghi pipeline IF/ID bằng 0 (làm nó trở thành 1 lệnh noop)
- Lệnh nhảy xuất hiện – chỉ chiếm 3% số lệnh trong SPECint

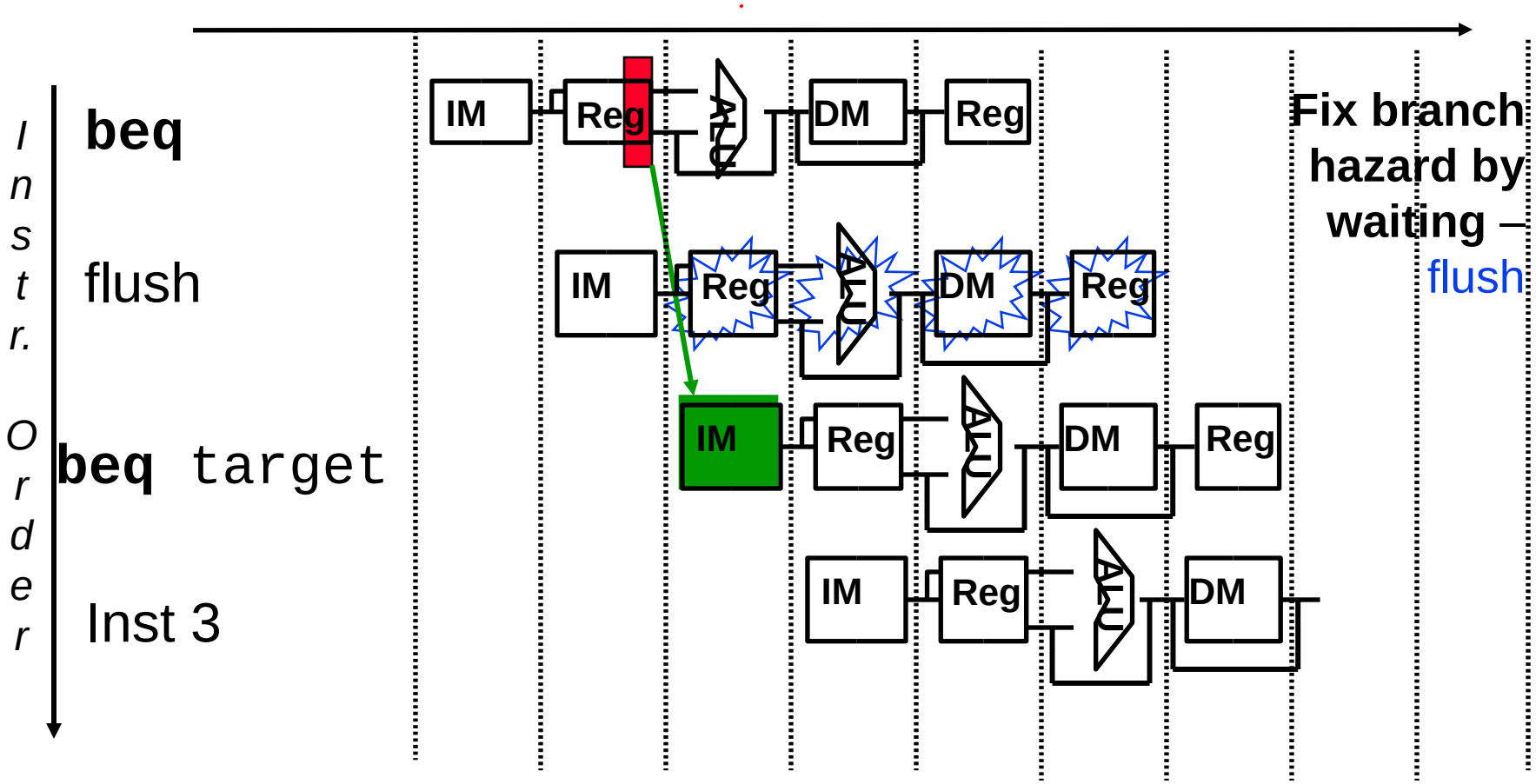
Xung đột điều khiển lệnh rẽ nhánh



Giải quyết xung đột điều khiển lệnh rẽ nhánh



Giải quyết xung đột điều khiển lệnh rẽ nhánh

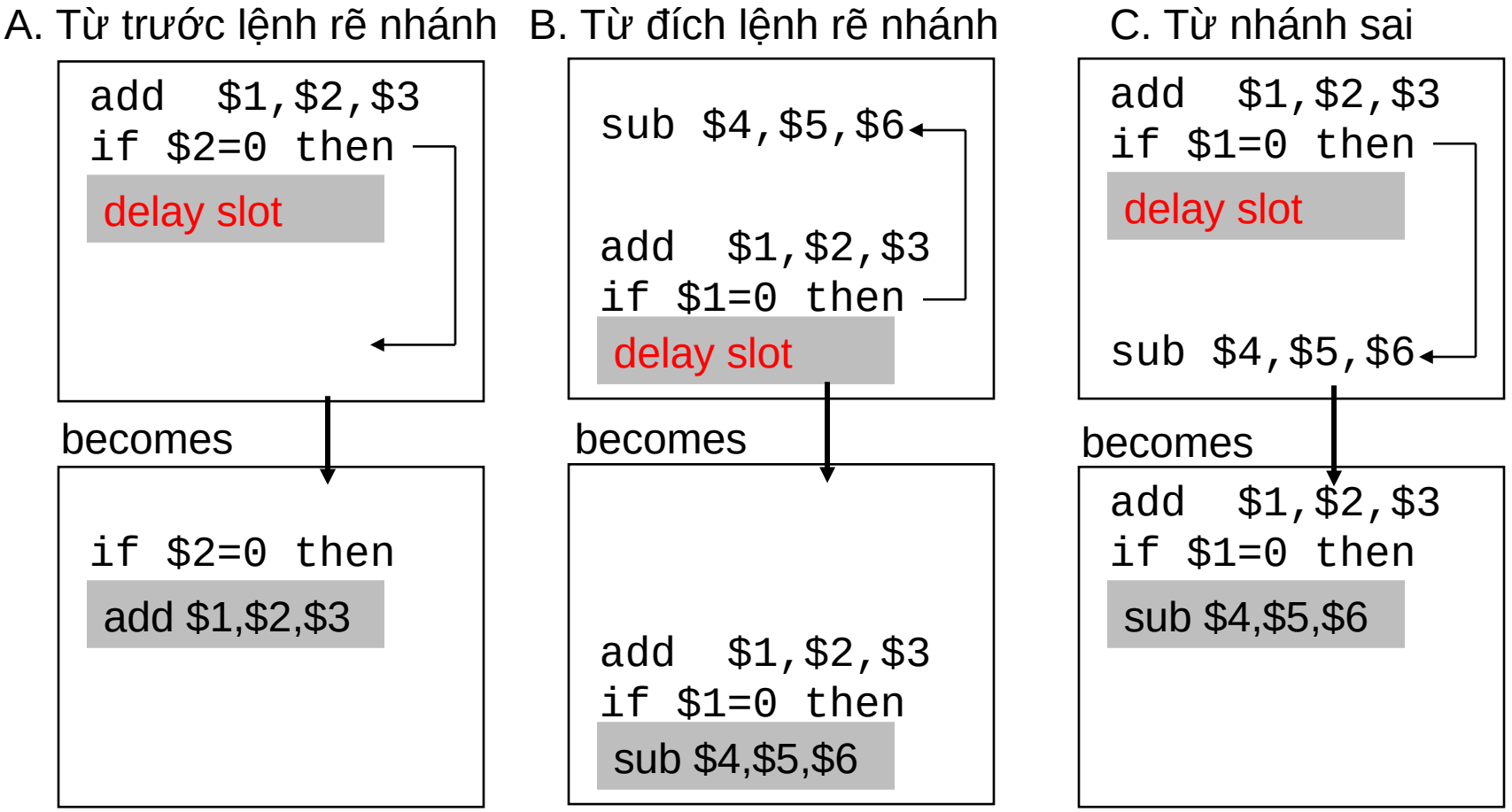


- Tính toán điều kiện rẽ nhánh càng sớm càng tốt, tức là trong giai đoạn giải mã ➔ chỉ cần 1 chu kỳ chờ

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

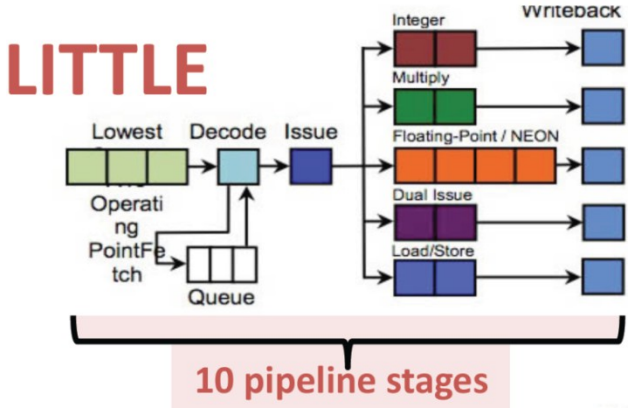
Sắp xếp lệnh trong rẽ nhánh chậm



- TH A là lựa chọn tốt nhất, điền được khe trễ và giảm I
- TH B , 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

Real world pipelines

We saw this earlier



Q: Which one is going to run at a faster clock frequency?

- ☐ Little
- ☐ Big
- ☐ Same

A: Big

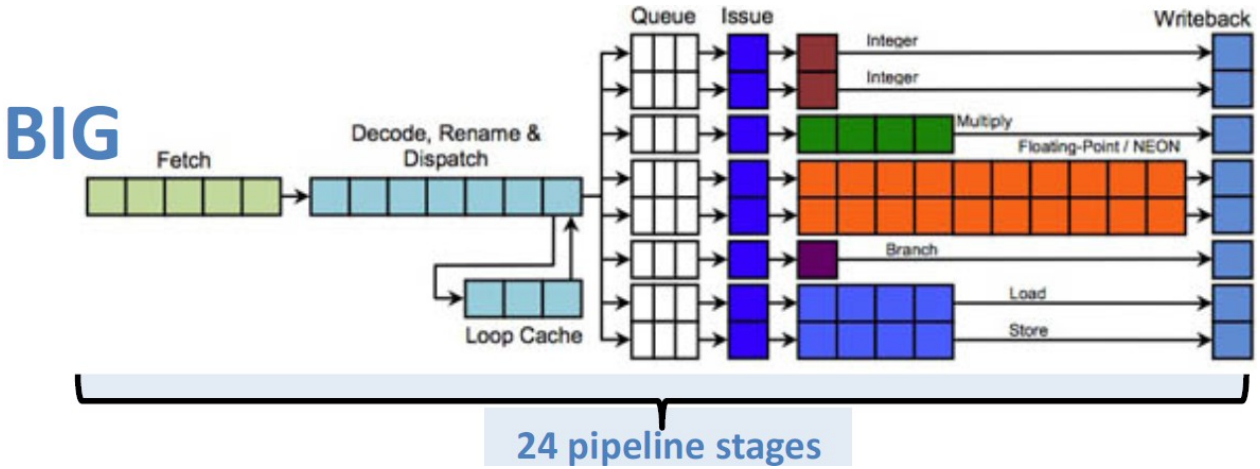
The big processor has a longer pipeline, which means each stage will be shorter, so a higher clock frequency.

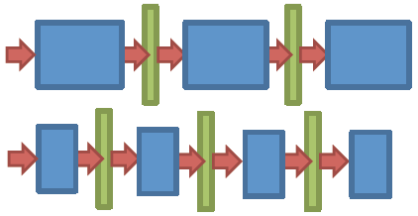
Q: Which pipeline will waste more time on pipeline registers?

- ☐ Little
- ☐ Big
- ☐ Same

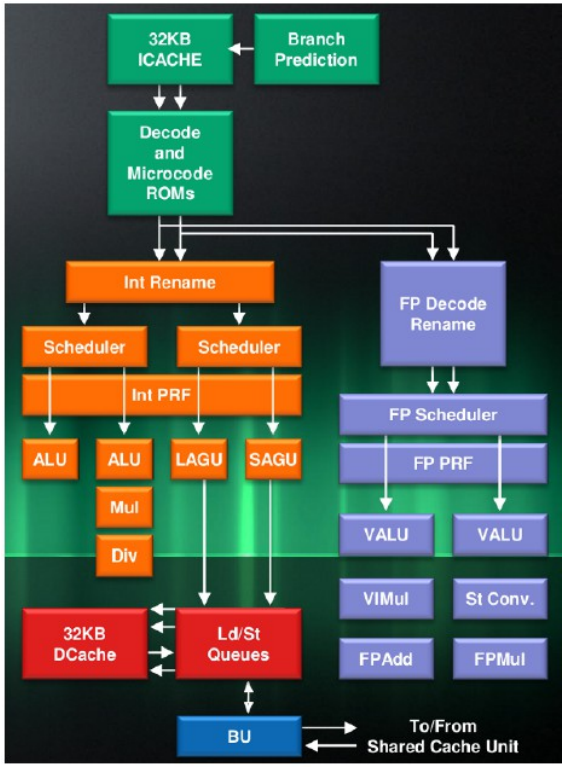
A: Big

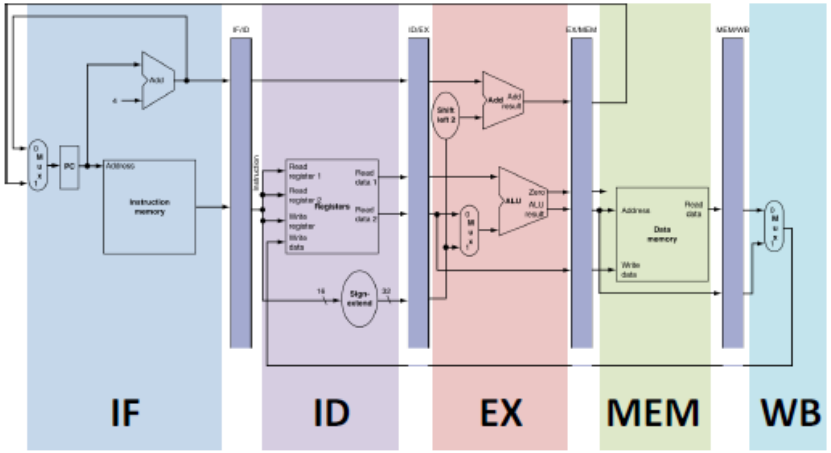
Running at a higher frequency means that a larger percentage of the time will be spent in pipeline registers. Equally important, because there are so many more stages, there will be more registers, which use more power and area.

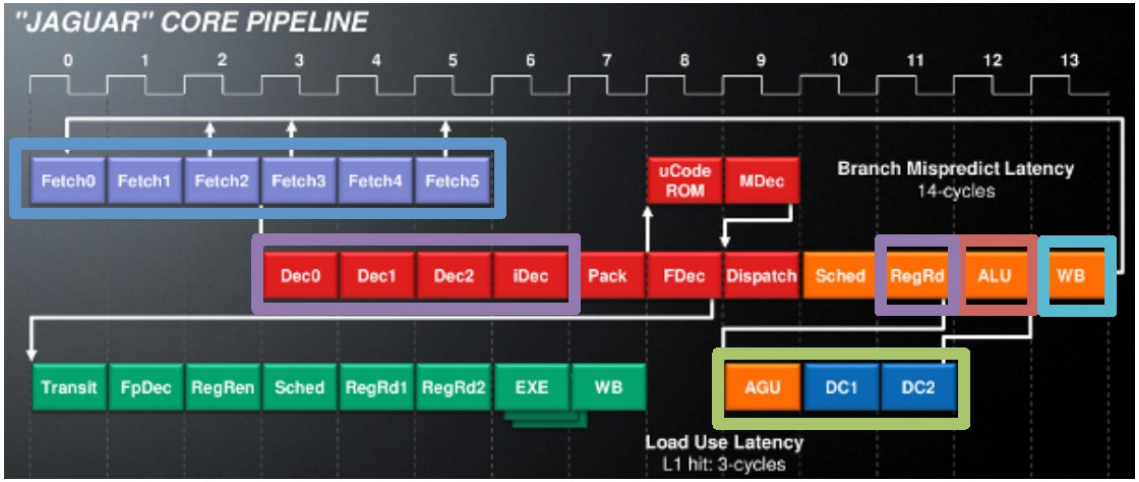




What is AMD doing?







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, 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)