COMPUTER ARCHITECTURE

Nội dung

- Số floating point
- Bộ xử lý (The Processor)
  - Đường dữ liệu
  - Đường điều khiển
- Pipelined Datapath and Control
- Hazards
  - Structural Hazards
  - Data Hazards
  - Control Hazards

IEEE Floating-Point Format

single: 8 bits
double: 11 bits

single: 23 bits
double: 52 bits

$$x = (-1)^{S} \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

- S: bit dấu (0 $\Rightarrow$ số khôn âm, 1 $\Rightarrow$ số âm)
- Chuẩn hóa phần lẻ (F): $1.0 \leq |\text{significand}| < 2.0$
- Phần mũ: phần mũ thực + Bias
  - Chắc chắn số mũ là không dấu (ex: $2^{-1} = 2^{126-127}$, F = 126 thay vì F= -1)
  - Chính xác đơn: Bias = 127; Chính xác kép: Bias = 1023

Single-Precision Range

- Giá trị nhỏ nhất
  - Phần mũ: 00000001
    $\Rightarrow$ phần mũ thực = 1 – 127 = –126
  - Phần lẻ: 000…00 $\Rightarrow$ significand = 1.0
  - $\pm 1.0 \times 2^{-126} \approx \pm 1.2 \times 10^{-38}$
- Giá trị lớn nhất
  - Phần mũ: 11111110
    $\Rightarrow$ phần mũ thực= 254 – 127 = +127
  - Phần lẻ: 111…11 $\Rightarrow$ significand $\approx$ 2.0
  - $\pm 2.0 \times 2^{+127} \approx \pm 3.4 \times 10^{+38}$

Floating-Point Example

- Biểu diễn số thực –0.75
  - $-0.75 = (-1)^1 \times 1.1_2 \times 2^{-1}$
  - S = 1
  - Fraction = $1000\ldots00_2$
  - Exponent = –1 + Bias
    - Single: –1 + 127 = 126 = 01111110_2
    - Double: –1 + 1023 = 1022 = 01111111110_2
- Single: 10111111000…00
- Double: 10111111111101000…00

Floating-Point Example

- Giá trị số thực của biểu diễn số sau là bao nhiêu?

  1100000101000…00

  - S = 1
  - Fraction = $01000…00_2$
  - Fxponent = 10000001_2 = 129

- $x = (-1)^1 \times (1 + 01_2) \times 2^{(129 - 127)}$

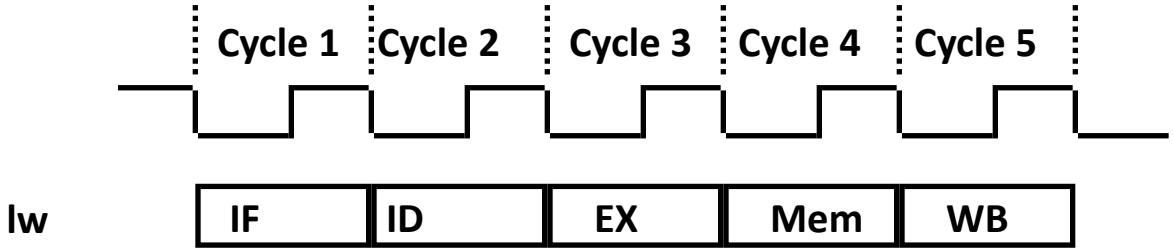
  $= (-1) \times 1.25 \times 2^2$

  $= -5.0$

MIPS Pipeline

- Chia làm 5 tầng, mỗi bước 1 tầng
  1. IF: Instruction fetch from memory (Nạp lệnh)
  2. ID: Instruction decode & register read (Giải mã lệnh & đọc thanh ghi)
  3. EX: Execute operation or calculate address (Thực thi phép toán hoặc tính địa chỉ)
  4. MEM: Access memory operand (truy xuất toán hạng bộ nhớ)
  5. WB: Write result back to register (ghi kết quả vào thanh ghi)

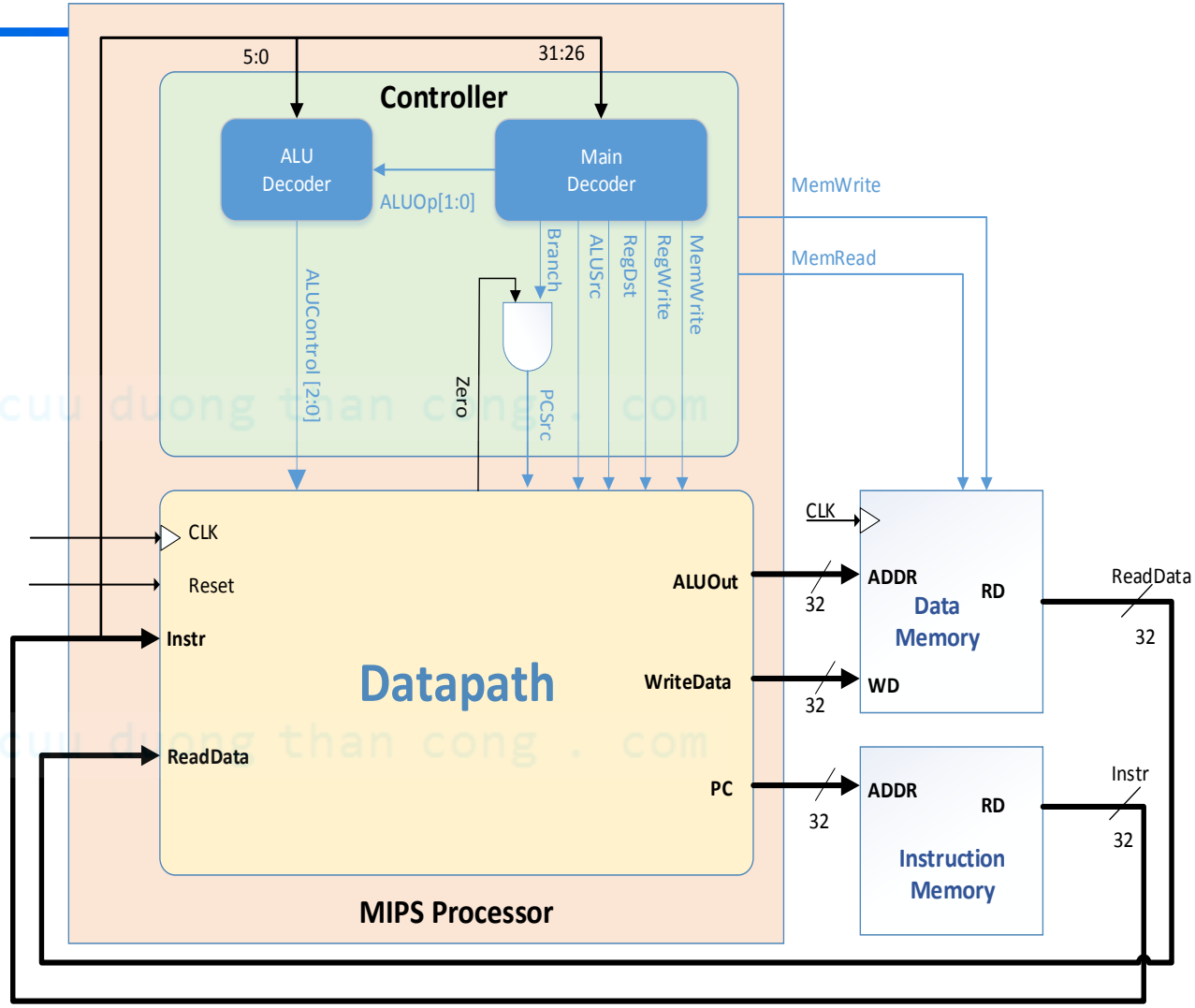
The Five Stages of Load Instructio



- IF: Nạp lệnh và cập nhật PC
- Dec: Nạp các thanh ghi và giải mã lệnh
- Exec: Tính toán địa chỉ bộ nhớ
- Mem: đọc dữ liệu từ bộ nhớ
- WB: ghi dữ liệu đọc được vào thanh ghi

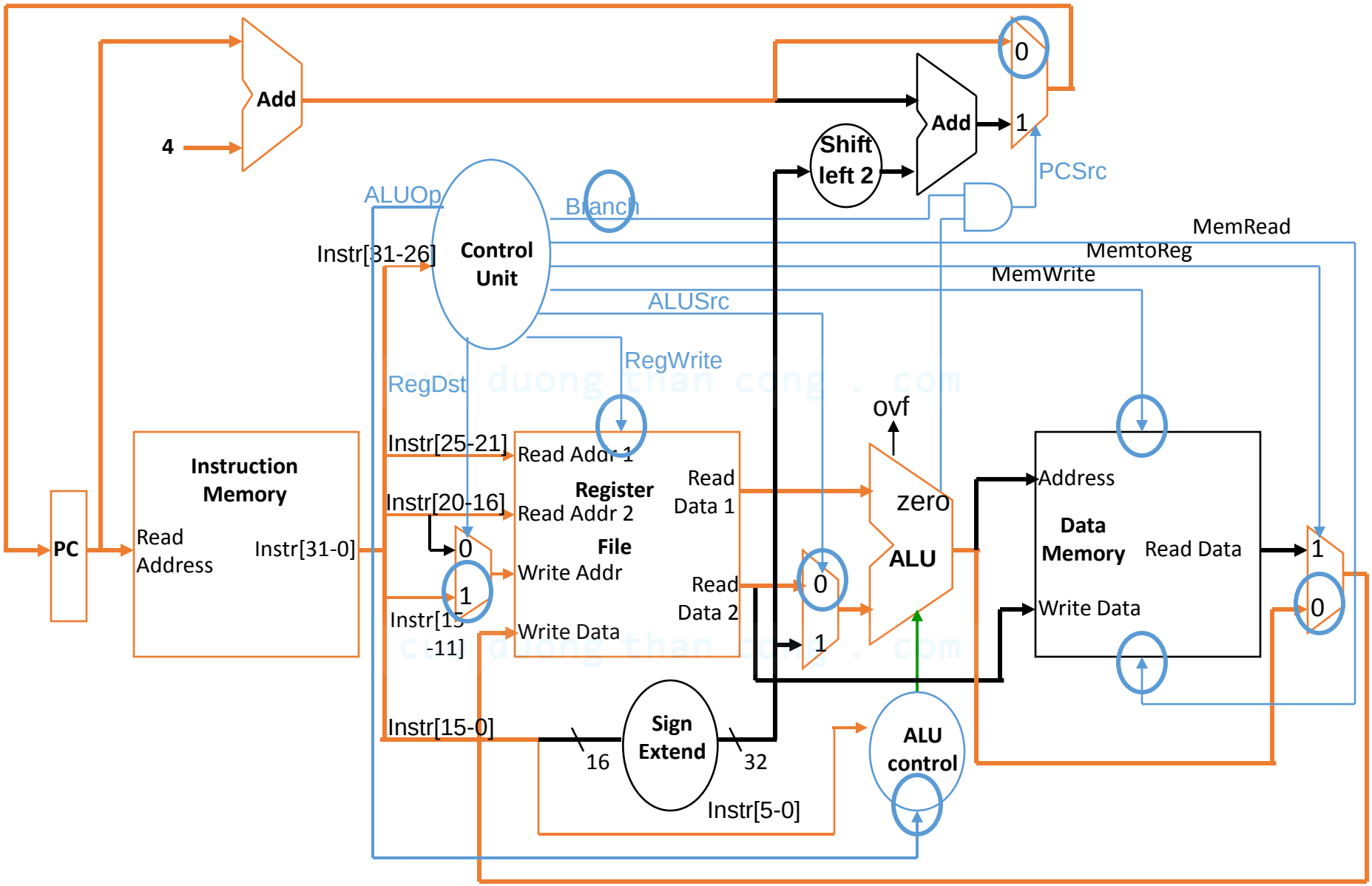
Single Cycle Processor

- Datapath
- Control

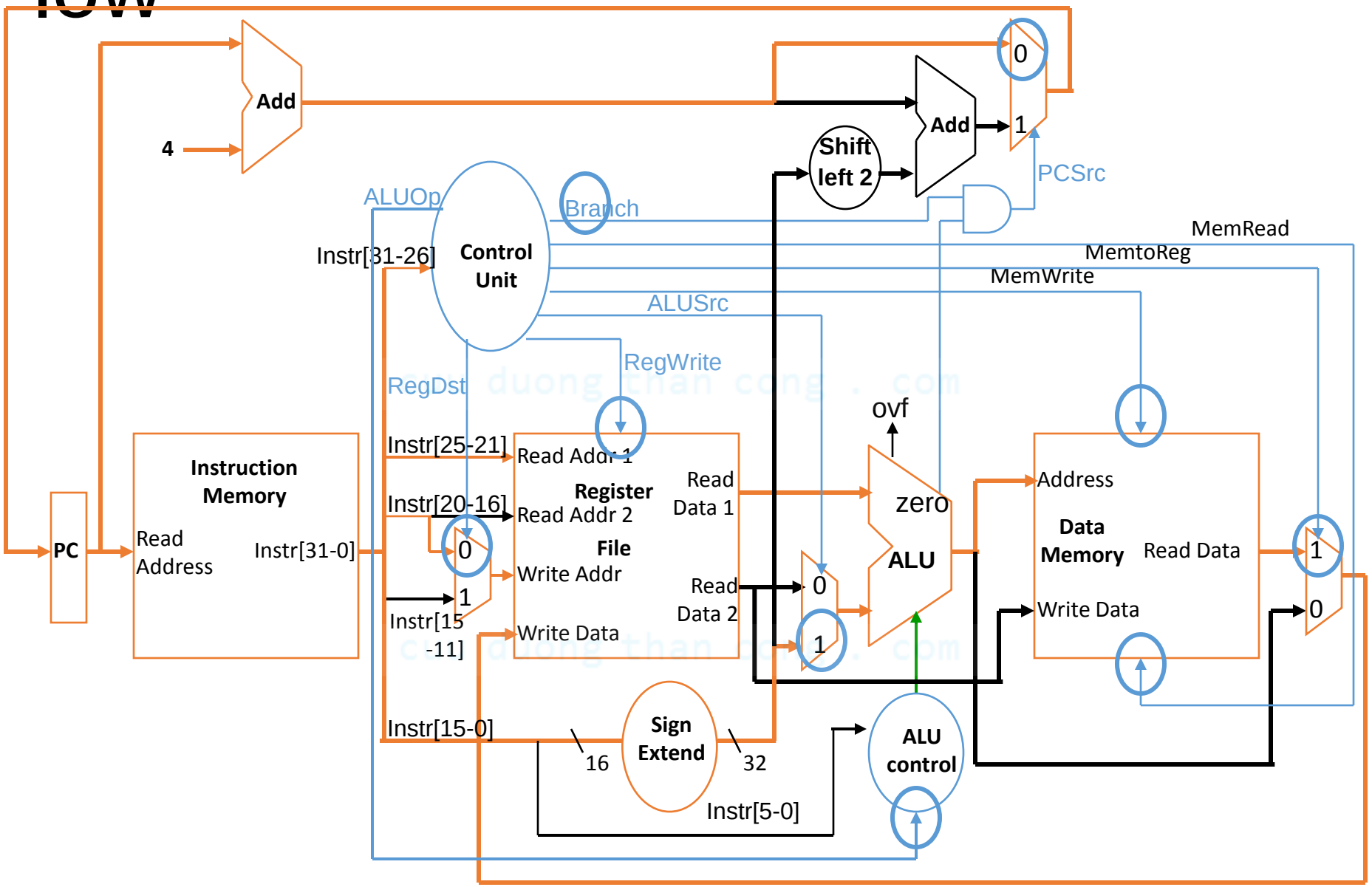


Lecture review

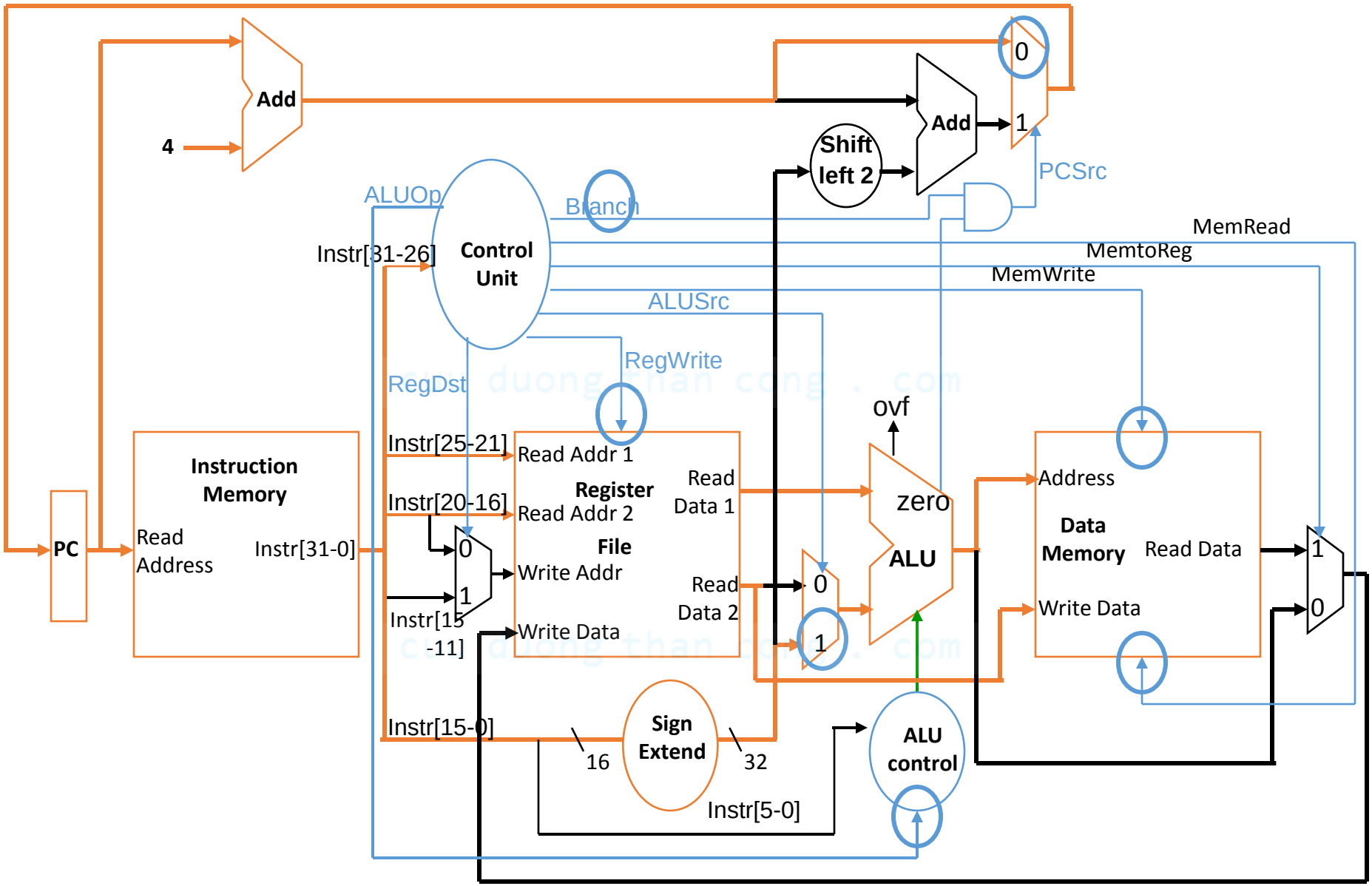
R-type Instruction Data/Control Flow



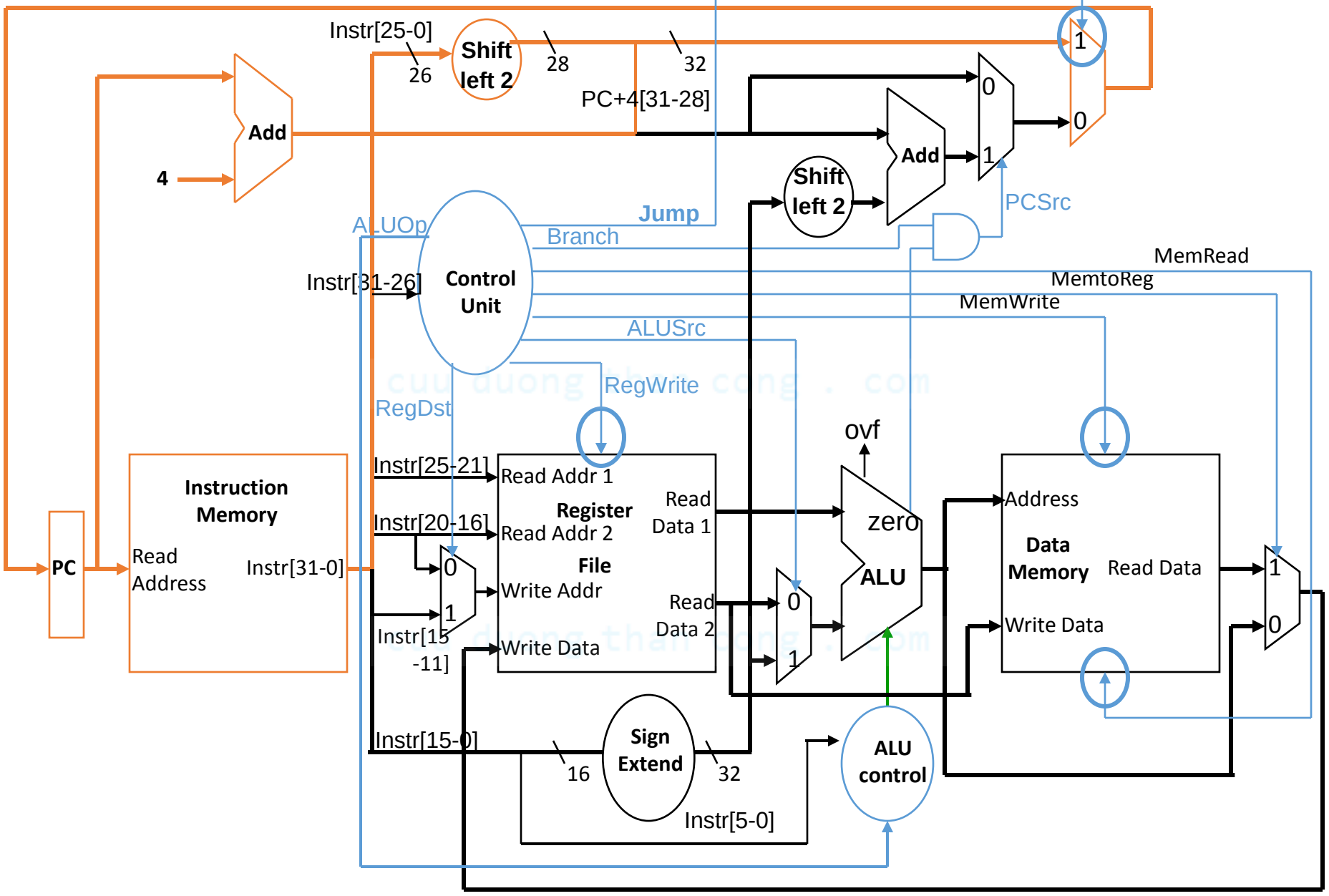
Load Word Instruction Data/Control Flow



Store Word Instruction Data/Control



Adding the Jump Operation



Pipeline Control

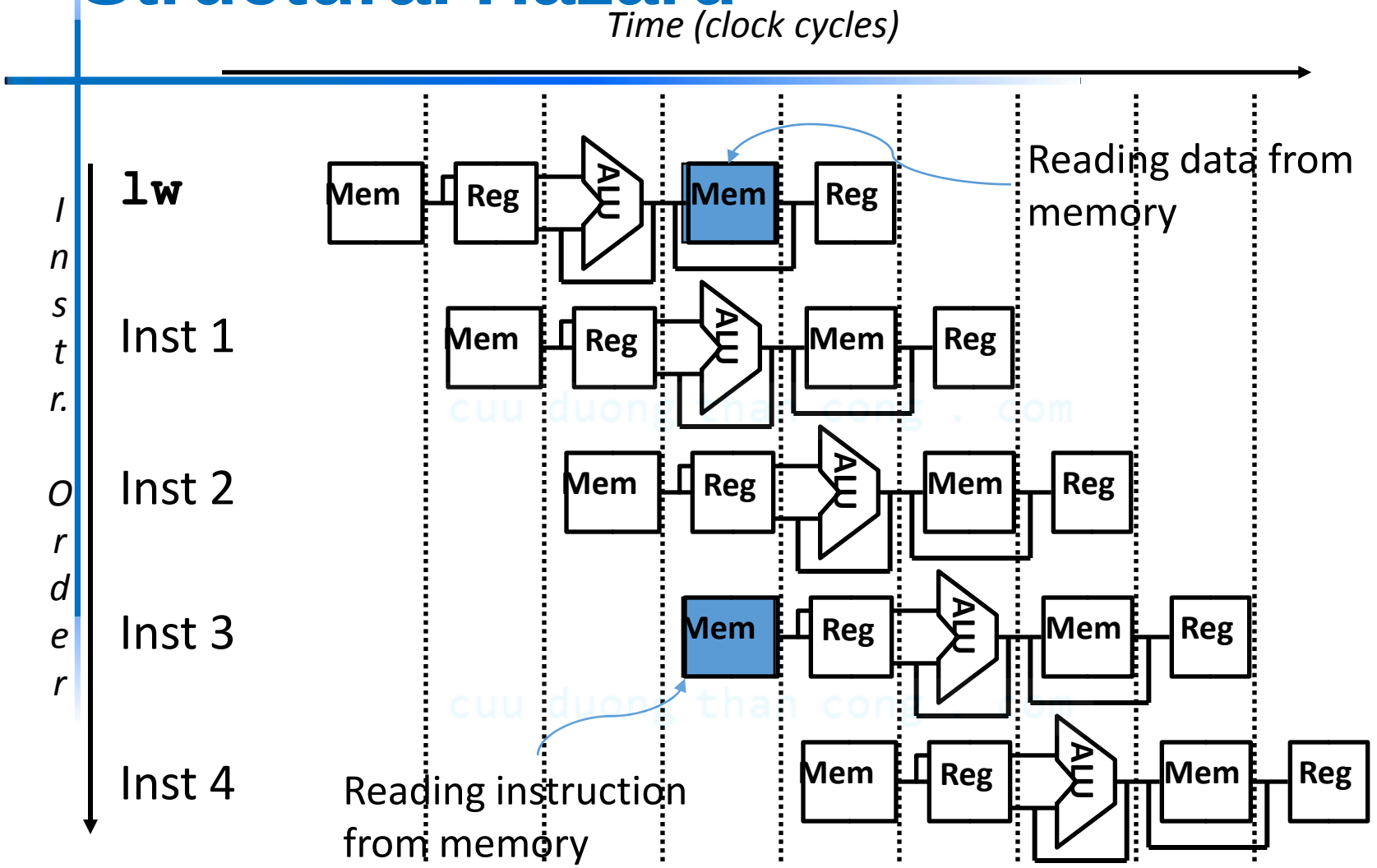
- IF Stage: read Instr Memory (always asserted) and write PC (on System Clock)
- ID Stage: no optional control signals to set

	EX Stage				MEM Stage			WB Stage	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Brch	Mem Read	Mem Write	Reg Write	Mem toReg
R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

Structure Hazards

- Xung đột do việc sự dụng nguồn tài nguyên
- MIPS pipeline với một single memory
  - Load/store yêu cầu truy xuất dữ liệu
  - Nạp lệnh có thể phải *stall* cho chu kỳ đó
    - Would cause a pipeline “bubble”
- Vì vậy, pipelined datapaths bộ nhớ độc lập instruction/data
  - Hoặc instruction/data caches

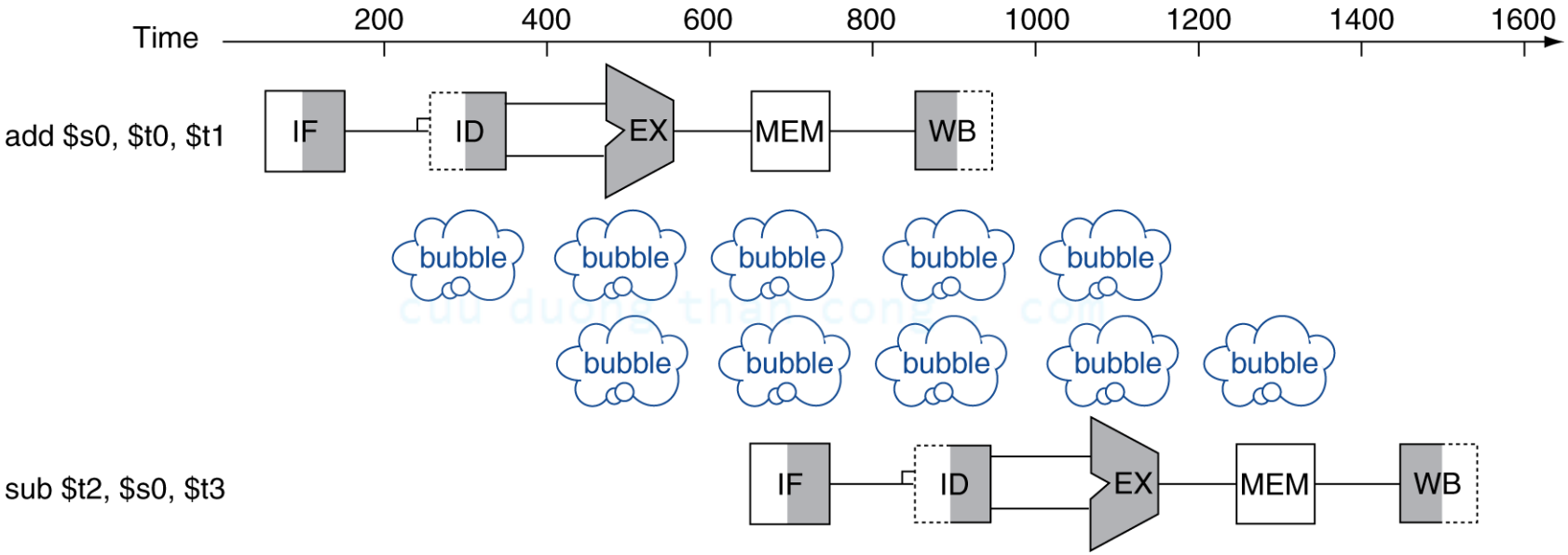
A Single Memory Would Be a Structural Hazard



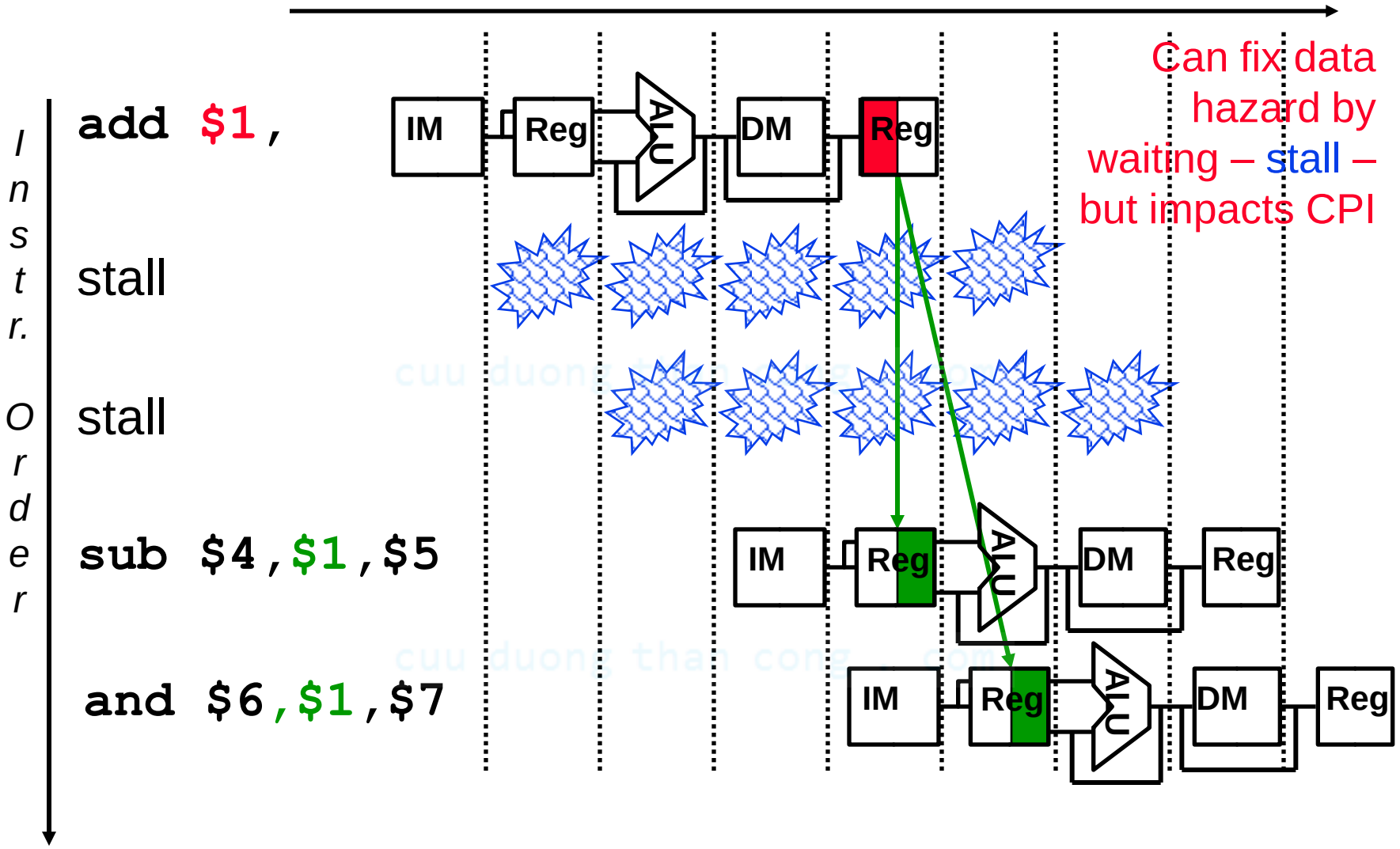
- Fix with separate instr and data memories (I$ and D$)

Data Hazards

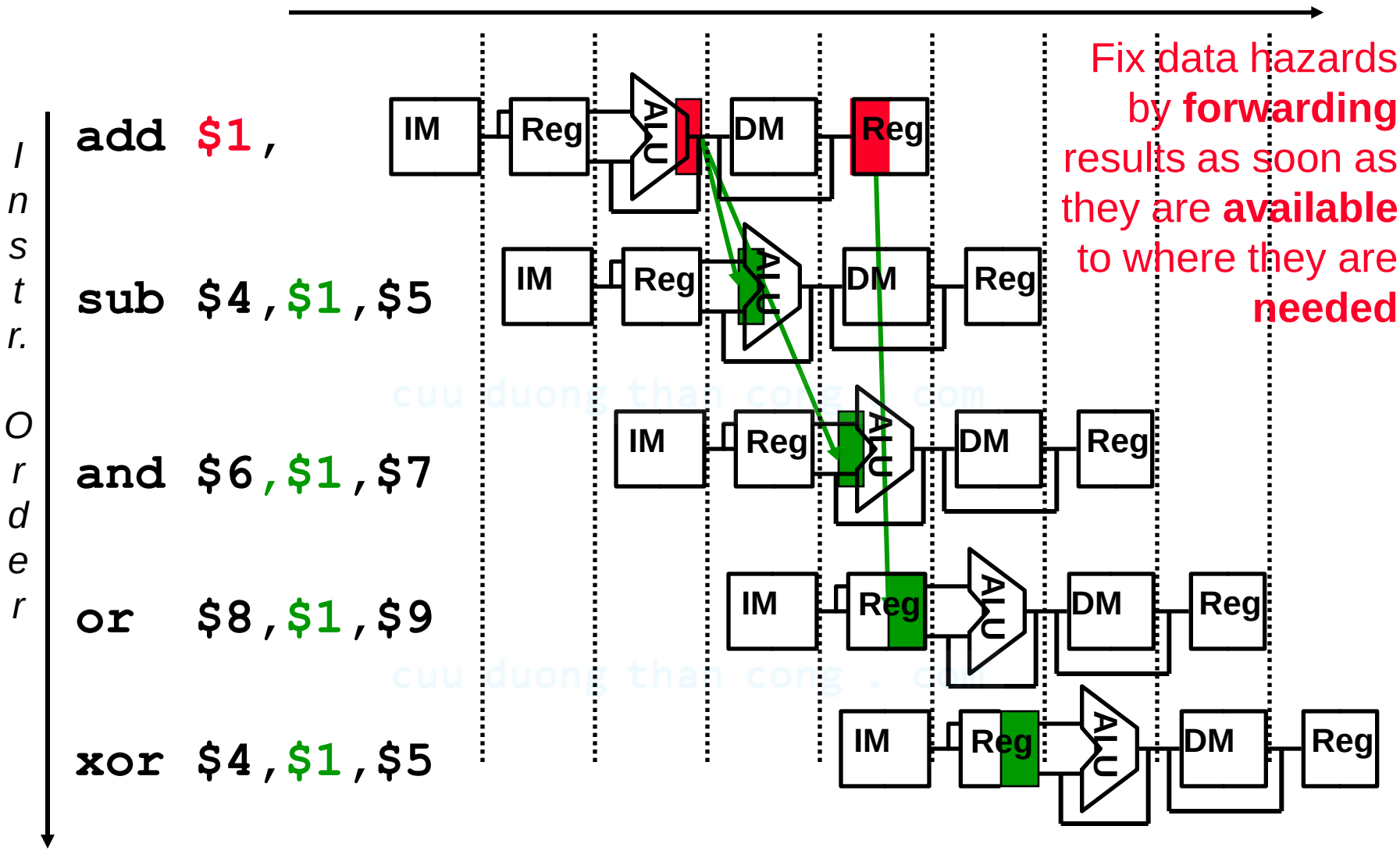
- Một lệnh phụ thuộc vào việc hoàn thành truy xuất dữ liệu bởi một lệnh trước đó.
  - `add  $s0, $t0, $t1`
    `sub  $t2, $s0, $t3`



One Way to "Fix" a Data Hazard



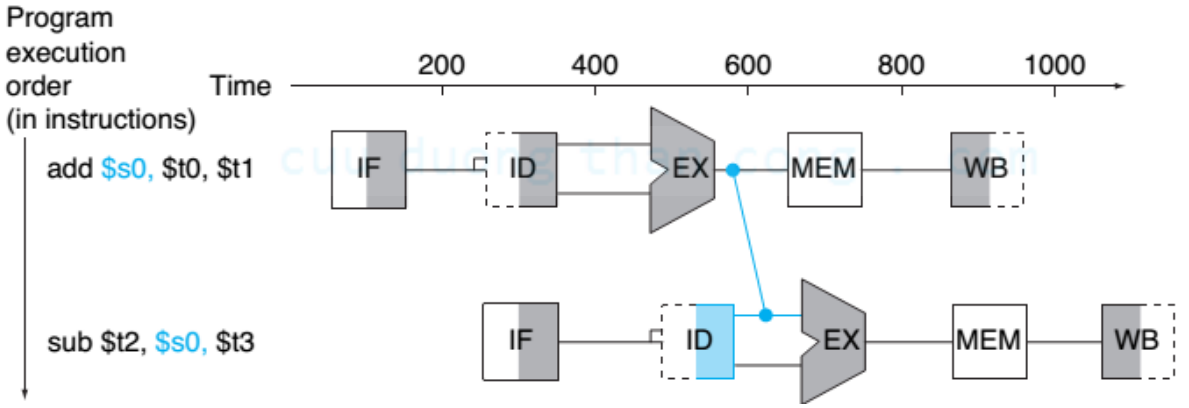
Another Way to "Fix" a Data Hazard



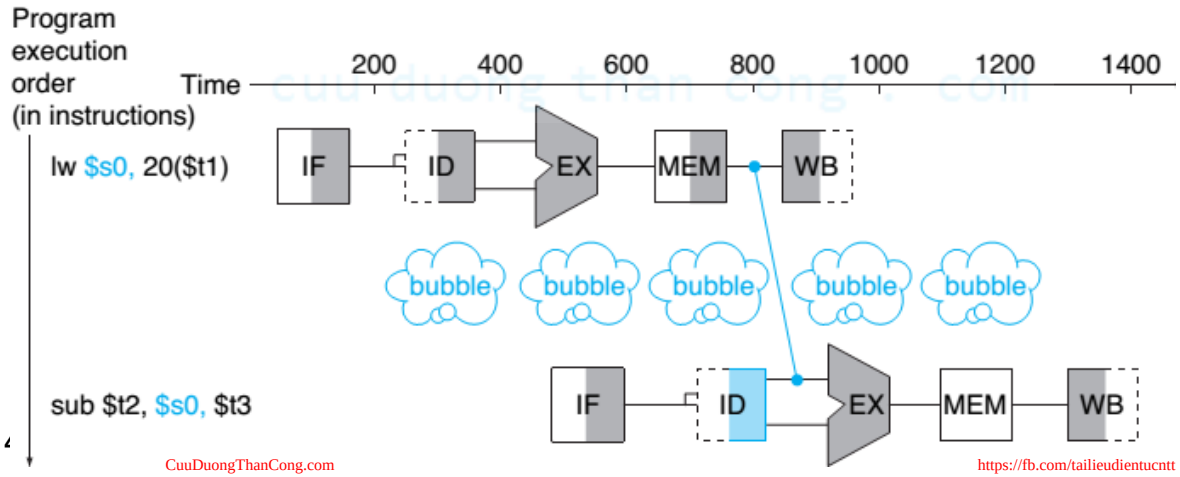
Xung đột dữ liệu

Tóm lại, với kỹ thuật forwarding có:

- ✓ **ALU-ALU forwarding** hay **EX-EX forwarding** (hình 1)
- ✓ **MEM-ALU forwarding** hay **MEM-EX forwarding** (hình 2)



Hình 1.



Hình 2.

Cho chuỗi lệnh như sau :

lw $1, 40($6)

add $6, $2, $2

sw $6, 50($1)

1. Trong trường hợp pipeline 5 tầng và không dùng kỹ thuật nhìn trước (no forwarding), sử dụng lệnh ‘nop’ để giải quyết xung đột xảy ra (nếu có) trong chuỗi lệnh trên.
2. Trong trường hợp pipeline 5 tầng và có kỹ thuật nhìn trước (forwarding), sử dụng lệnh ‘nop’ để giải quyết xung đột xảy ra (nếu có) trong chuỗi lệnh trên.

Control Hazards

Xung đột điều khiển

- Một số lệnh nhảy có điều kiện và không điều kiện trong MIPS (branches, jump) tạo ra xung đột điều kiển này

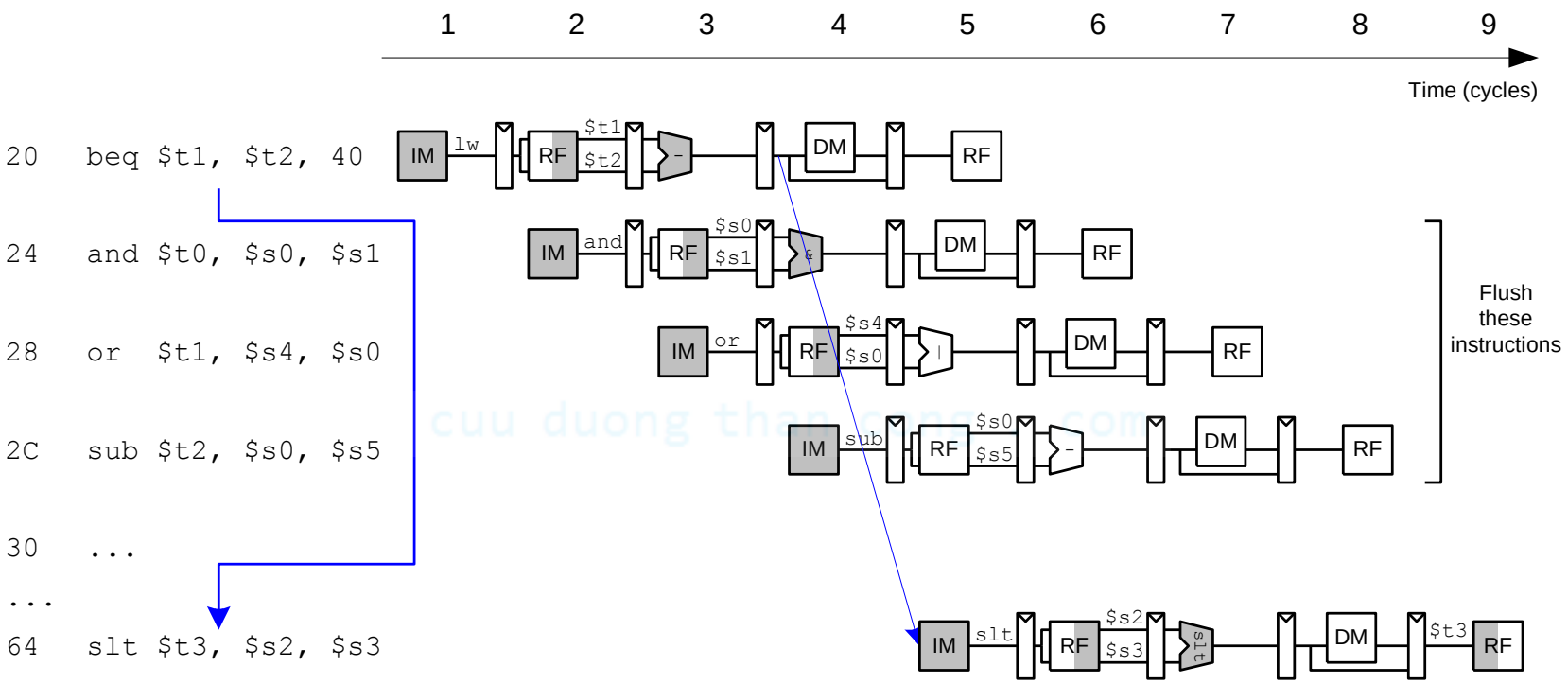
  Ví dụ xét đoạn chương trình sau:

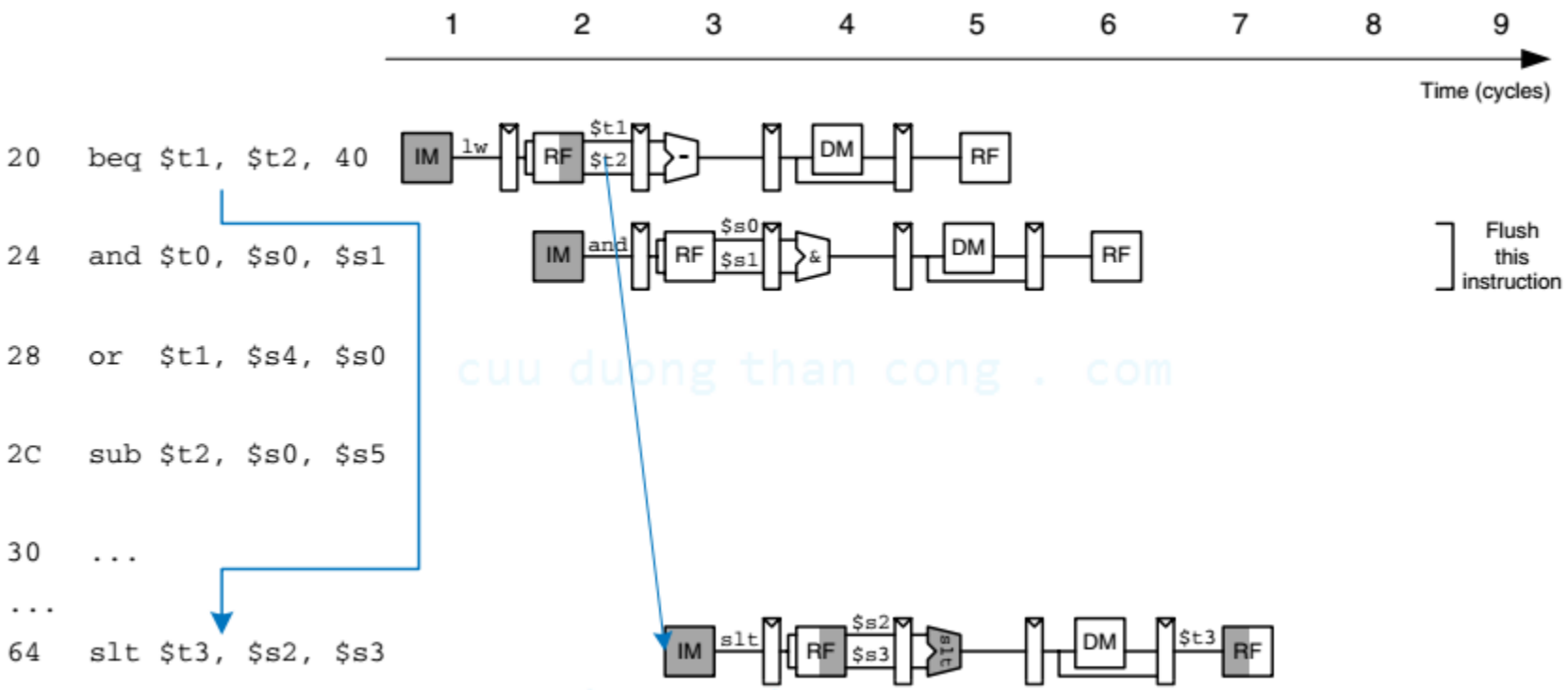
  add $1, $5, $6

  beq $1, $2, label

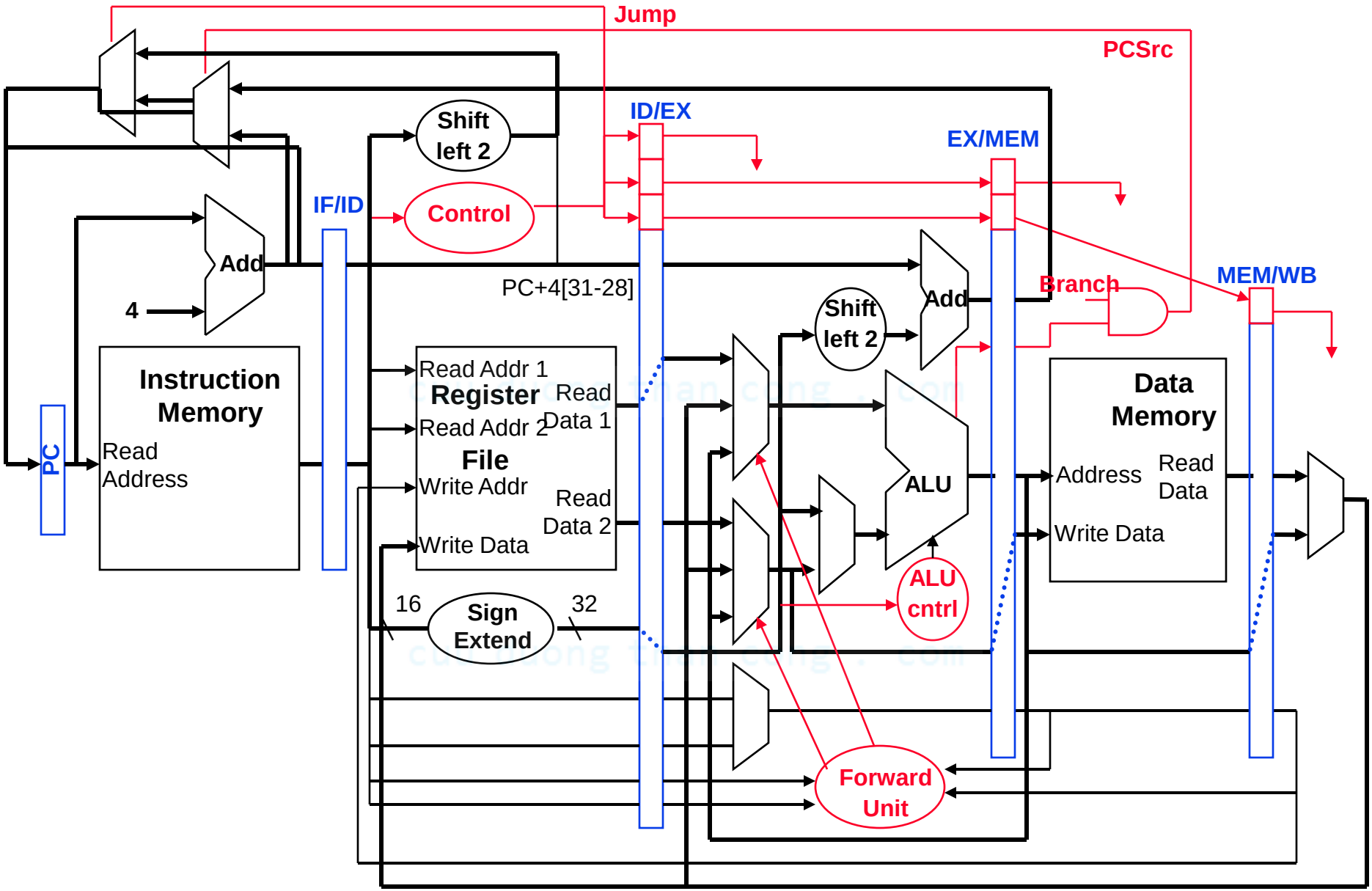
  lw $3, 300($s0)

  Nếu áp dụng pipeline thông thường, tại chu kỳ thứ ba của pipeline, khi *beq* đang thực thi công đoạn ID thì lệnh *lw* sẽ được nạp vào. Nhưng nếu điều kiện bằng của lệnh *beq* xảy ra thì lệnh thực hiệp tiếp sau đó không phải là *lw* mà là lệnh được gán nhãn '*label*', lúc này xảy ra xung đột điều khiên.





Datapath Branch and Jump Hardware



Summary

- All modern day processors use pipelining
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Potential speedup: a CPI of 1 and fast a CC
- Pipeline rate limited by slowest pipeline stage
  - Unbalanced pipe stages makes for inefficiencies
  - The time to "fill" pipeline and time to "drain" it can impact speedup for deep pipelines and short code runs
- Must detect and resolve hazards
  - Stalling negatively affects CPI (makes CPI less than the ideal of 1)

Assignments

- Trong bài tập này, chúng ta khảo sát pipeline ảnh hưởng như thế nào tới chu kỳ xung clock (clock cycle time) của processor. Giả sử rằng mỗi công đoạn (stage) trong pipeline có thời gian thực hiện

	IF	ID	EX	MEM	WB
a.	300ps	400ps	350ps	500ps	100ps
b.	200ps	150ps	120ps	190ps	140ps

1. Chu kỳ xung clock cần cho processor là bao nhiêu nếu processor thiết kế có pipeline và không pipeline.
2. Thời gian cần thiết để thực hiện lệnh lw, sw, add, beq cho trường hợp processor có pipeline và không pipeline có thể là bao nhiêu

Assignments

- Giả sử rằng các lệnh được thực thi trong processor được phân rã như sau (áp dụng cho câu 3 và 4)

	ALU	beq	lw	sw
a.	50%	25%	15%	10%
b.	30%	25%	30%	15%

3. Giả sử rằng không có khoảng thời gian rỗi (stalls) hoặc xung đột (hazards), phần truy xuất bộ nhớ (MEM) và phần truy xuất ghi trên tập thanh ghi (WB) sử dụng bao nhiêu % chu kỳ của toàn chương trình

4.

	IF	ID	EX	MEM	WB
lw	x	x	x	x	x
sw	x	x	x	x	
ALU (add, sub, AND, OR, slt)	x	x	x		x
Branch (beq)	x	x	x		

Assignments

- Giả sử rằng các lệnh được thực thi trong processor được phân rã như sau (áp dụng cho câu 3 và 4)

	ALU	beq	lw	sw
a.	50%	25%	15%	10%
b.	30%	25%	30%	15%

4. Giả sử có thiết kế mới như sau: mỗi lệnh chỉ sử dụng đúng các giai đoạn cần có của nó, có thể lấy nhiều chu kỳ để hoàn thành, nhưng một lệnh phải hoàn thành xong thì những lệnh khác mới được nạp vào. Thiết kế này tạm gọi là thiết kế đa chu kỳ.
Theo kiểu này, mỗi lệnh chỉ đi qua những công đoạn mà nó thực sự cần (ví dụ, sw chỉ sử dụng 4 công đoạn, không có công đoạn WB). Tính chu kỳ xung clock, so sánh thời gian thực thi của thiết kế đa chu kỳ này với thiết kế đơn chu kỳ (single cycle design) và pipeline.
(Chú ý: lw: sử dụng 5 stage; sw: 4 stage (không WB); ALU: 4 stage (không MEM), beq 4 stage (không WB))

Assignments

- The pipelined MIPS processor is running the following program. Which registers are being written, and which are being read on the fifth cycle?

```
add $s0, $t0, $t1
sub $s1, $t2, $t3
and $s2, $s0, $s1
or $s3, $t4, $t5
slt $s4, $s2, $s3
```

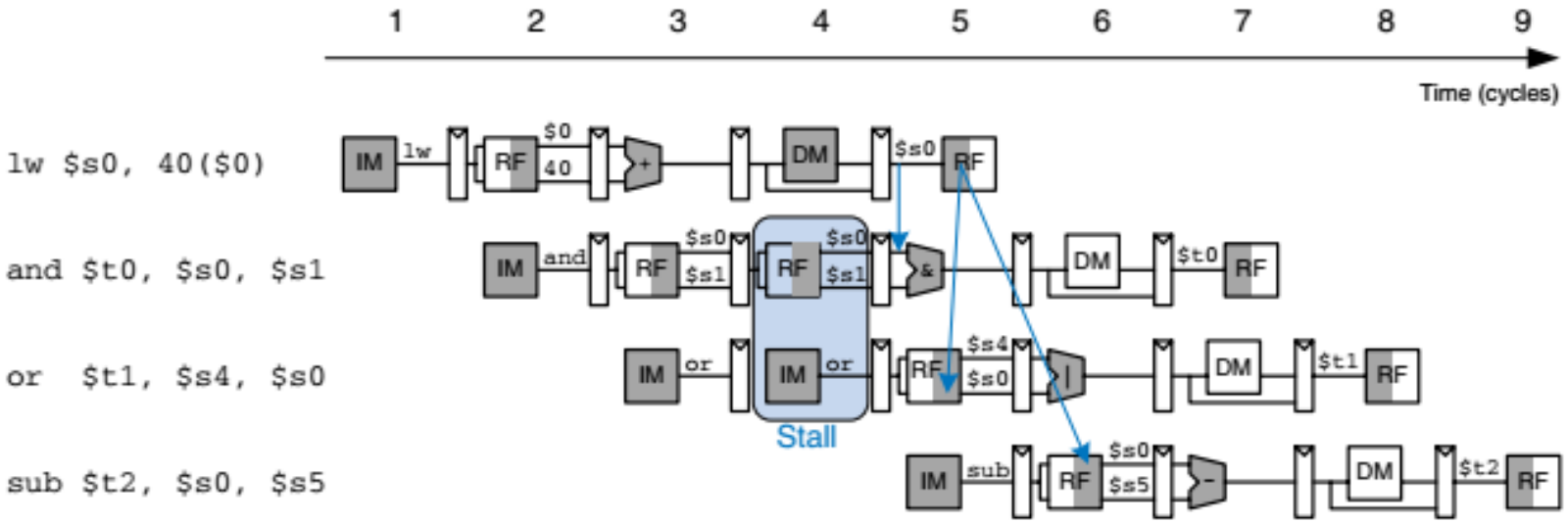


Figure 7.52 **Abstract pipeline diagram illustrating stall to solve hazards**

Using a diagram similar to above Figure show the forwarding and stalls needed to execute the following instructions on the pipelined MIPS processor.

add $t0, $s0, $s1
sub $t0, $t0, $s2
lw $t1, 60($t0)
and $t2, $t1, $t0

```
add $t0, $s0, $s1
lw  $t1, 60($s2)
sub $t2, $t0, $s3
and $t3, $t1, $t0
```