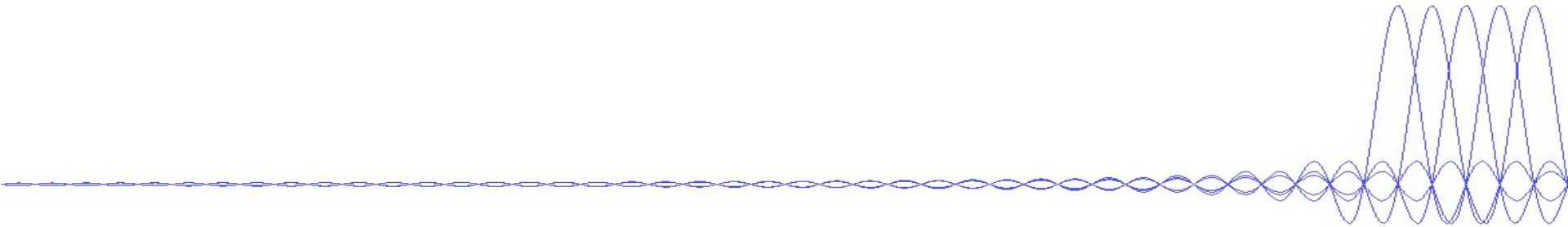
COMPUTER ENGINEERING

KIẾN TRÚC MÁY TÍNH

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

Tuần 13

Kỹ thuật ống dẫn (Pipeline)



COMPUTER ENGINEERING

Kỹ thuật ống dẫn (pipeline)

Mục đích:

- ✔ Tổng quan về kỹ thuật Pipeline
- ✔ Những vấn đề phát sinh và hướng giải quyết trong pipeline

Slide được dịch và các hình được lấy từ sách tham khảo:

Computer Organization and Design: The Hardware/Software Interface, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.

Kỹ thuật ống dẫn (pipeline)

- **Pipeline** là một kỹ thuật mà trong đó các lệnh được thực thi theo kiểu chồng lấn lên nhau (overlap).

- Ví dụ minh họa hoạt động như thế nào là **không pipeline** hay **pipeline**:

Giả sử một phòng có nhiều người, mỗi người đều cần giặt quần áo bẩn của mình. Quá trình giặt quần áo bao gồm 4 công đoạn:

1. Đặt quần áo bẩn vào máy giặt để giặt
2. Khi máy giặt hoàn thành, đưa quần áo ướt vào máy sấy
3. Khi máy sấy hoàn thành, đặt quần áo khô lên bàn và ủi
4. Khi ủi hoàn tất, xếp quần áo vào tủ

COMPUTER ENGINEERING

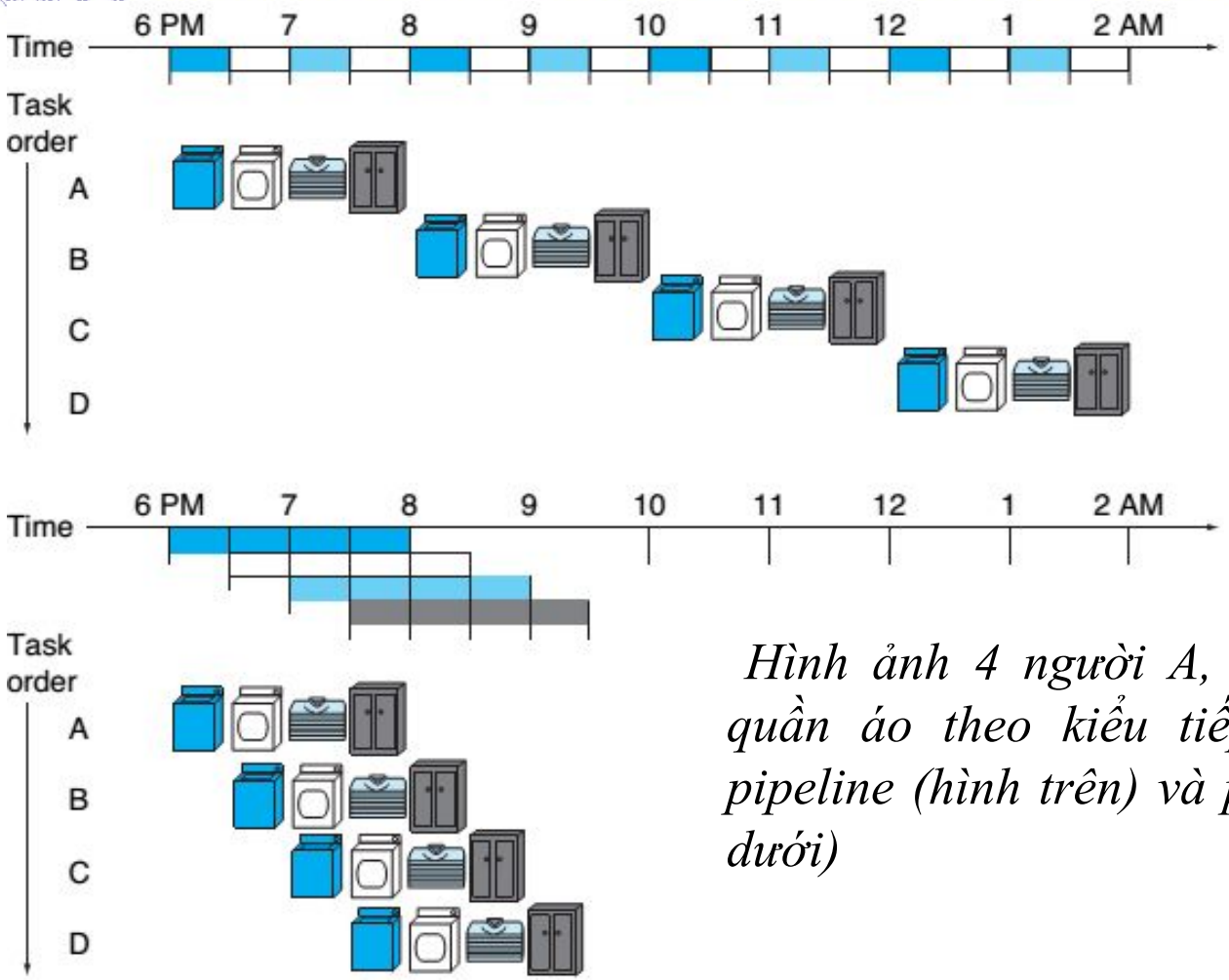
Kỹ thuật ống dẫn (pipeline)

- Nếu một người hoàn tất tất cả các công đoạn giặt quần áo (xong công đoạn ủi, xếp quần áo vào tủ) thì người khác mới bắt đầu (bắt đầu đặt quần áo bẩn vào máy giặt), quá trình thực hiện này gọi là **không pipeline**.

- Tuy nhiên, rõ ràng rằng khi người trước hoàn thành công đoạn 1, sang công đoạn 2 thì máy giặt đã trống, lúc này người tiếp theo có thể đưa quần áo bẩn vào giặt. Như vậy, người tiếp theo không cần phải chờ người trước xong công đoạn thứ 4 mới có thể bắt đầu, mà ngay khi người trước đến công đoạn thứ 2 thì người tiếp theo đã có thể bắt đầu công đoạn thứ nhất và cứ tiếp tục như vậy. Quá trình thực hiện chồng lấn này gọi là **pipeline.**

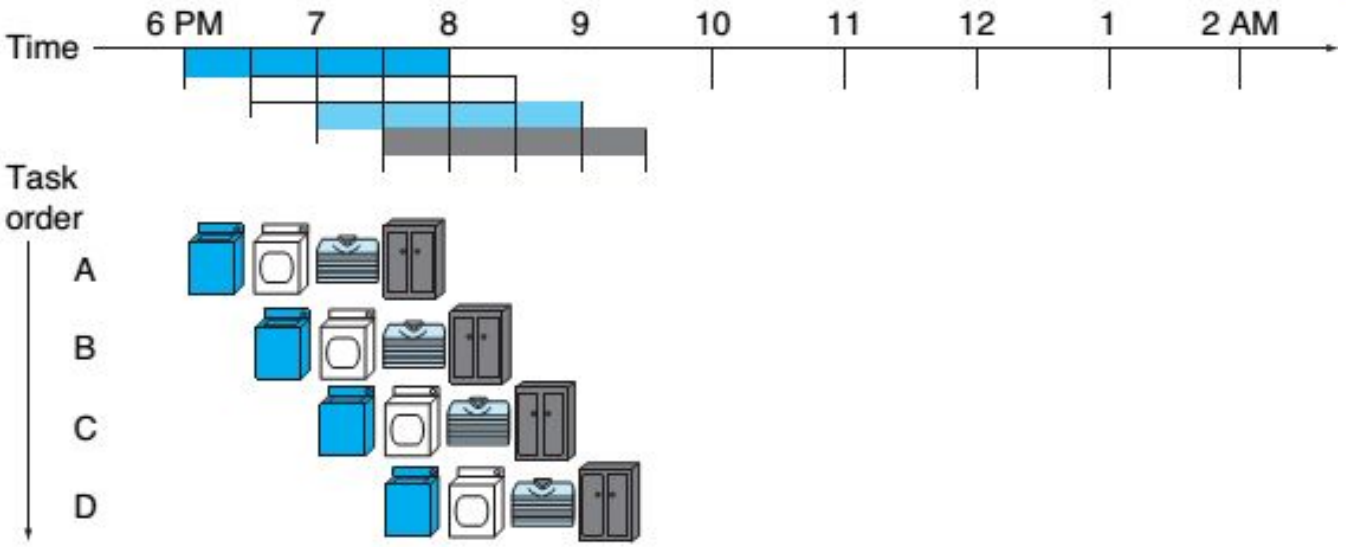
COMPUTER ENGINEERING

Kỹ thuật ống dẫn (pipeline)



Hình ảnh 4 người A, B, C, D giặt quần áo theo kiểu tiếp cận không pipeline (hình trên) và pipeline (hình dưới)

Kỹ thuật ống dẫn (pipeline)



- Cách tiếp cận dùng kỹ thuật pipeline tiêu tốn ít thời gian hơn cho tất cả các công việc hoàn tất bởi vì các công việc được thực hiện song song, vì vậy số công việc hoàn thành trong một giờ sẽ nhiều hơn so với không pipeline.
- Chú ý, pipeline không làm giảm thời gian hoàn thành một công việc mà làm giảm thời gian hoàn thành tổng số công việc (như trong ví dụ trên, thời gian cho người A hoàn thành việc giặt khi áp dụng pipeline hay không pipeline đều là 2 giờ, nhưng tổng số giờ cho 4 người A, B, C và D hoàn thành dùng pipeline giảm rất nhiều so với không pipeline)

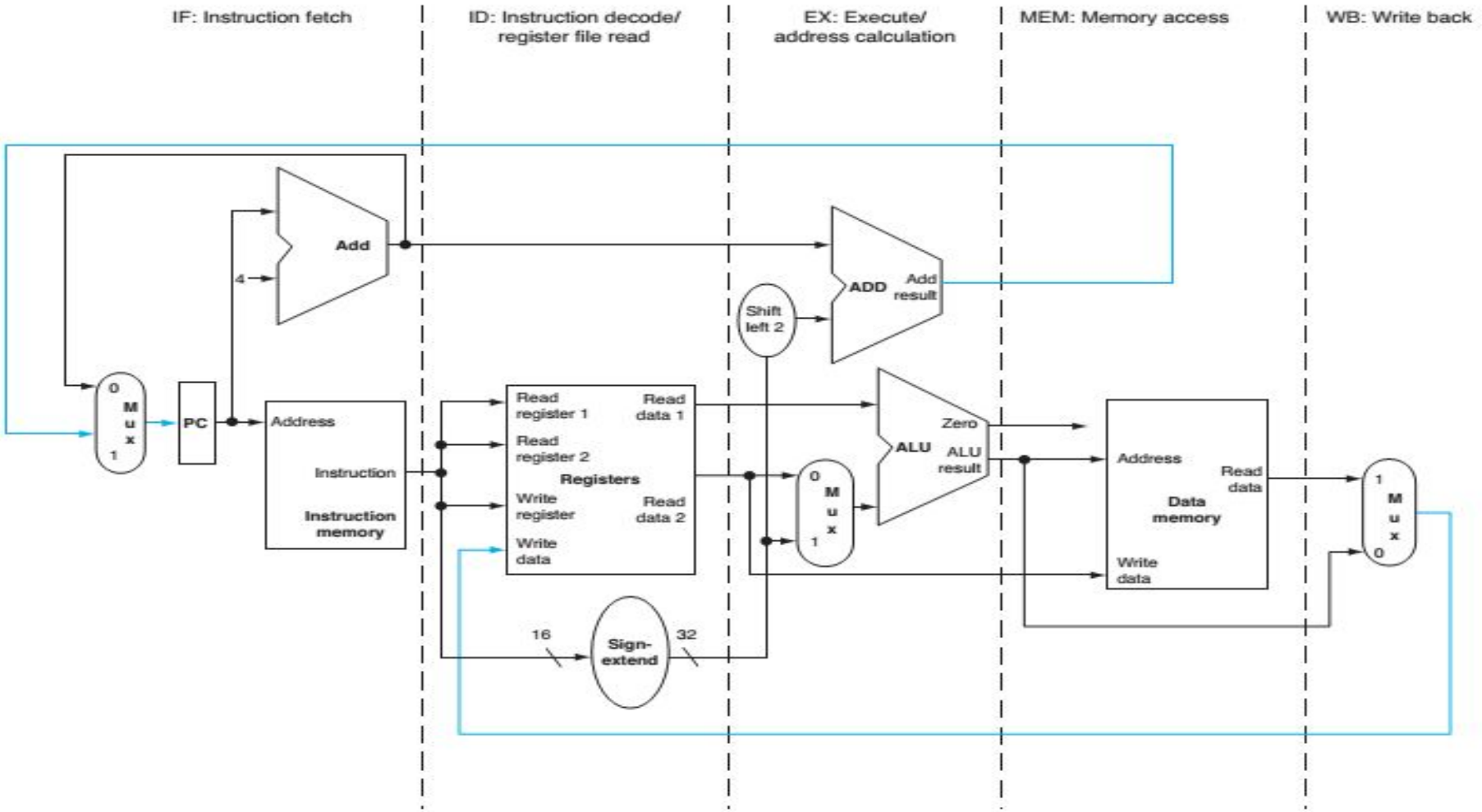
Kỹ thuật ống dẫn (pipeline)

Tương tự việc giặt quần áo, thay vì một lệnh phải chờ lệnh trước đó hoàn thành mới được thực thi thì các lệnh trong một chương trình của bộ xử lý có thể thực thi theo kiểu pipeline.

Khi thực thi, các lệnh MIPS được chia làm 5 công đoạn:

1. **Nạp lệnh từ bộ nhớ**
2. **Giải mã lệnh và đọc các thanh ghi cần thiết (MIPS cho phép đọc và giải mã đồng thời)**
3. **Thực thi các phép tính hoặc tính toán địa chỉ**
4. **Truy xuất các toán hạng trong bộ nhớ**
5. **Ghi kết quả cuối vào thanh ghi**

Vì vậy, MIPS pipeline trong chương này xem như có 5 công đoạn (còn gọi là pipeline 5 tầng)



1. Nạp lệnh từ bộ nhớ – **IF**
2. Giải mã lệnh và đọc các thanh ghi – **ID**
3. Thực thi – **EX**
4. Truy xuất bộ nhớ – **MEM**
5. Ghi kết quả vào thanh ghi – **WB**

COMPUTER ENGINEERING

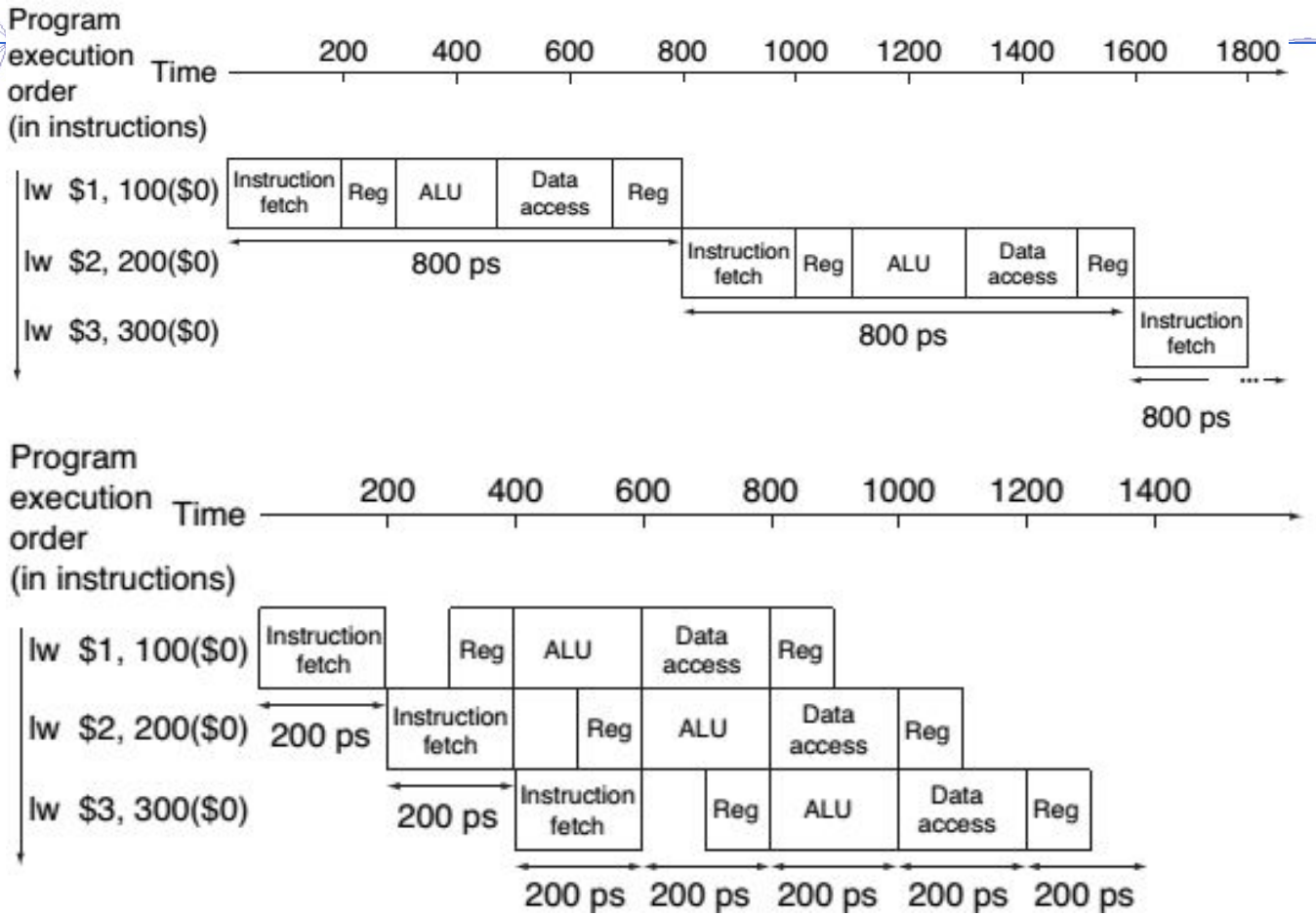
Kỹ thuật ống dẫn (pipeline)

- Xét một bộ xử lý với 8 lệnh cơ bản: load word (***lw***), store word (***sw***), add (***add***), subtract (***sub***), AND (***and***), OR (***or***), set less than (***slt***), và nhảy với điều kiện bằng (***beq***).
- Giả sử thời gian hoạt động các công đoạn như sau: 200 ps cho truy xuất bộ nhớ, 200 ps cho tính toán của ALU, 100 ps cho thao tác đọc/ghi thanh ghi
- So sánh thời gian trung bình giữa các lệnh của hiện thực đơn chu kỳ và pipeline.

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

COMPUTER ENGINEERING

Kỹ thuật ống dẫn (pipeline)



Ví dụ hình ảnh 3 lệnh lw thực hiện theo kiểu không pipeline, đơn chu kỳ (hình trên) và có pipeline (hình dưới)

- Thời gian giữa lệnh thứ nhất và thứ tư trong không pipeline là 3 x 800 = 2400 ps, nhưng trong pipeline là 3 x 200 = 600 ps

Kỹ thuật ống dẫn (pipeline)

Sự tăng tốc của pipeline

- **Trong trường hợp lý tưởng:** khi mà các công đoạn pipeline hoàn toàn bằng nhau thì thời gian giữa hai lệnh liên tiếp được thực thi trong pipeline bằng:

$$\text{Thời gian giữa hai lệnh liên tiếp pipeline} = \frac{\text{thời gian giữa hai lệnh liên tiếp không pipeline}}{\text{số tầng pipeline}}$$

Như vậy, trong ví dụ trên, thời gian giữa hai lệnh liên tiếp có pipeline bằng 160 ps (800:5 = 160)

 Trong trường hợp lý tưởng, pipeline sẽ tăng tốc so với không pipeline với số lần đúng bằng số tầng của pipeline.

- **Trong thực tế:** Các công đoạn thực tế không bằng nhau, việc áp dụng pipeline phải chọn công đoạn dài nhất để làm một chu kỳ pipeline.

Vì vậy, trong ví dụ trên, thời gian liên tiếp giữa hai lệnh pipeline là 200 ps. Và áp dụng pipeline tăng tốc gấp 4 lần so với không pipeline.

*Speed-up ≈ Thời gian giữa hai lệnh liên tiếp không pipeline **:** Thời gian giữa hai lệnh liên tiếp pipeline ≈ 800 **:** 200 = **4** < **5** (number pipeline stages)*

 Trong thực tế, pipeline sẽ tăng tốc so với không pipeline với số lần nhỏ hơn số tầng của pipeline.

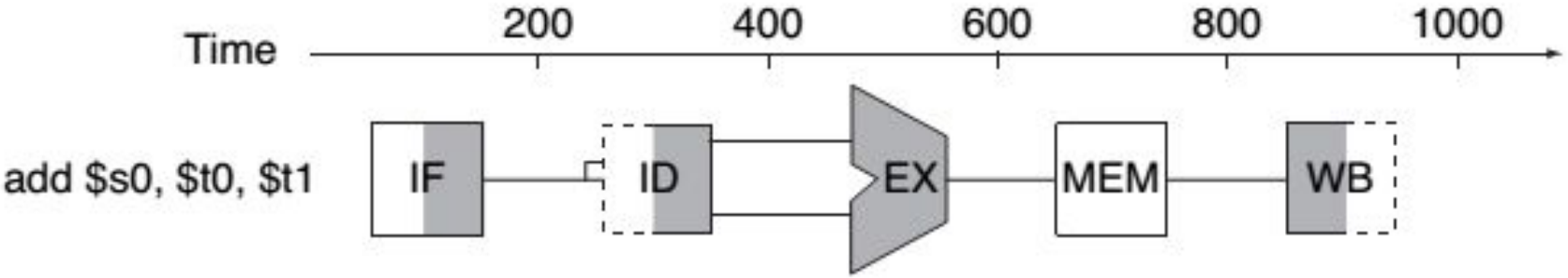
Kỹ thuật ống dẫn (pipeline)

Lưu ý, pipeline tăng tốc so với không pipeline:

- Kỹ thuật pipeline **không** giúp giảm thời gian thực thi của từng lệnh riêng lẻ mà giúp giảm tổng thời gian thực thi của đoạn lệnh/chương trình chứa nhiều lệnh (từ đó giúp thời gian trung bình của mỗi lệnh giảm)
- Việc giúp giảm thời gian thực thi cho nhiều lệnh vô cùng quan trọng, vì các chương trình chạy trong thực tế thông thường lên đến hàng tỉ lệnh.

Kỹ thuật ống dẫn (pipeline)

Quy ước trình bày 5 công đoạn thực thi một lệnh của pipeline:



Lưu ý cách vẽ hình các công đoạn pipeline như sau:

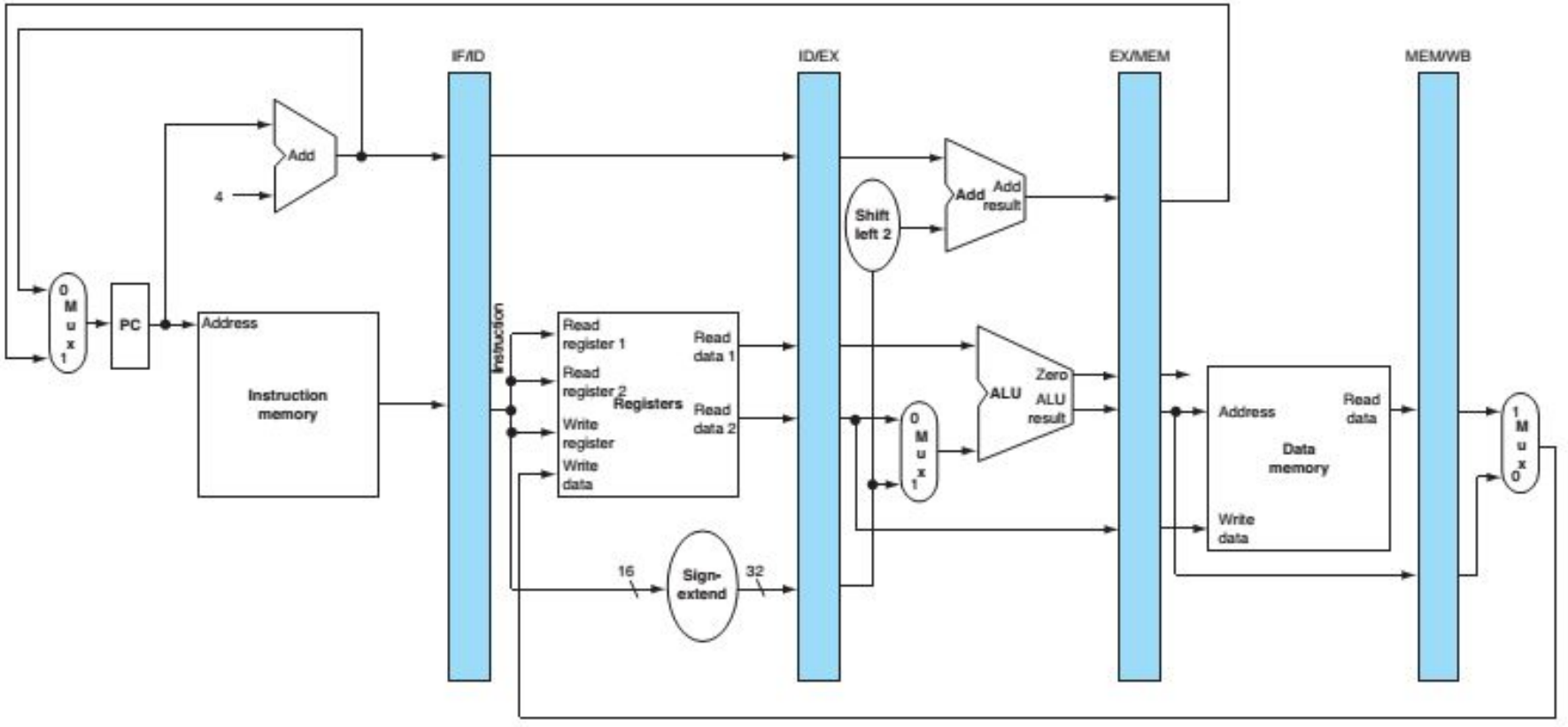
✔ Khối tô đen hoàn toàn hoặc để trắng hoàn toàn: Trong mỗi công đoạn pipeline, nếu lệnh thực thi không làm gì trong công đoạn này sẽ được tô trắng, ngược lại sẽ được tô đen.

Ví dụ lệnh "add" có EX đen và MEM trắng tức lệnh này có tính toán trong công đoạn EX và không truy xuất bộ nhớ dữ liệu trong công đoạn MEM.

✔ Các công đoạn liên qua đến bộ nhớ và thanh ghi có thể tô nữa trái hoặc nữa phải đen: Nếu nữa phải tô đen, tức công đoạn đó đang thực hiện thao tác đọc; ngược lại nếu nữa trái tô đen, công đoạn đó đang thực hiện thao tác ghi.

Kỹ thuật ống dẫn (pipeline)

Hình ảnh datapath có hỗ trợ pipeline



Chi tiết về datapath và control cho pipeline có thể xem thêm tại phần 4.6 sách tham khảo chính.

Kỹ thuật ống dẫn (pipeline)

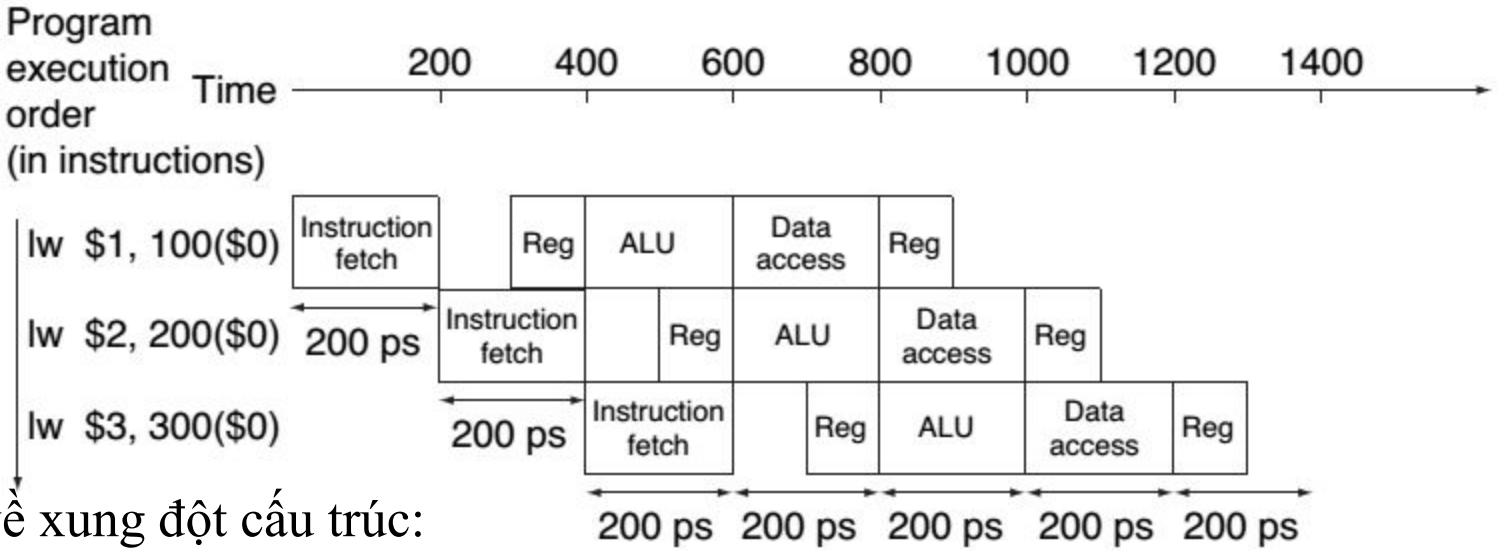
Các xung đột có thể xảy ra khi áp dụng kỹ thuật pipeline (Pipeline Hazards):

Xung đột là trạng thái mà lệnh tiếp theo không thể thực thi trong chu kỳ pipeline ngay sau đó (hoặc thực thi nhưng sẽ cho ra kết quả sai), thường do một trong ba nguyên nhân sau:

- **Xung đột cấu trúc (Structural hazard):** là khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do phần cứng cần không thể hỗ trợ. Nói cách khác, xung đột cấu trúc xảy ra khi có hai lệnh cùng truy xuất vào một tài nguyên phần cứng nào đó cùng một lúc.
- **Xung đột dữ liệu (Data hazard):** là khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do dữ liệu mà lệnh này cần vẫn chưa sẵn sàng.
- **Xung đột điều khiển (Control/Branch hazard):** là khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do lệnh nạp vào không phải là lệnh được cần. Xung đột này xảy ra trong trường hợp luồng thực thi chứa các lệnh nhảy.

Kỹ thuật ống dẫn (pipeline)

Xung đột cấu trúc



Ví dụ về xung đột cấu trúc:

Giả sử rằng chúng ta có một bộ nhớ đơn duy nhất thay vì hai bộ nhớ lệnh và dữ liệu rời rạc nhau. Nếu pipeline trong ví dụ ở hình trên có thêm lệnh thứ tư thì trong chu kỳ pipeline từ 600 tới 800 khi lệnh thứ nhất thực hiện truy xuất bộ nhớ lấy dữ liệu thì lệnh thứ tư sẽ thực hiện truy xuất bộ nhớ lấy lệnh. Do không có bộ nhớ lệnh và dữ liệu riêng lẻ, trong trường hợp này sẽ có xung đột cấu trúc xảy ra.

Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

Ví dụ cho đoạn lệnh sau: *add $s0, $t0, $t1*

sub $t2, $s0, $t3

Trong ví dụ trên, nếu áp dụng pipeline bình thường thì công đoạn ID của lệnh *sub* sẽ thực hiện cùng lúc với cộng đoạn EX của lệnh *add*. Trong công đoạn ID, lệnh *sub* sẽ cần đọc giá trị của thanh ghi *$s0*, trong khi đó giá trị mới của thanh ghi $s0 phải tới công đoạn WB của lệnh *add* mới sẵn sàng. Vì vậy, nếu thực hiện pipeline thông thường, trường hợp này sẽ xảy ra xung đột dữ liệu

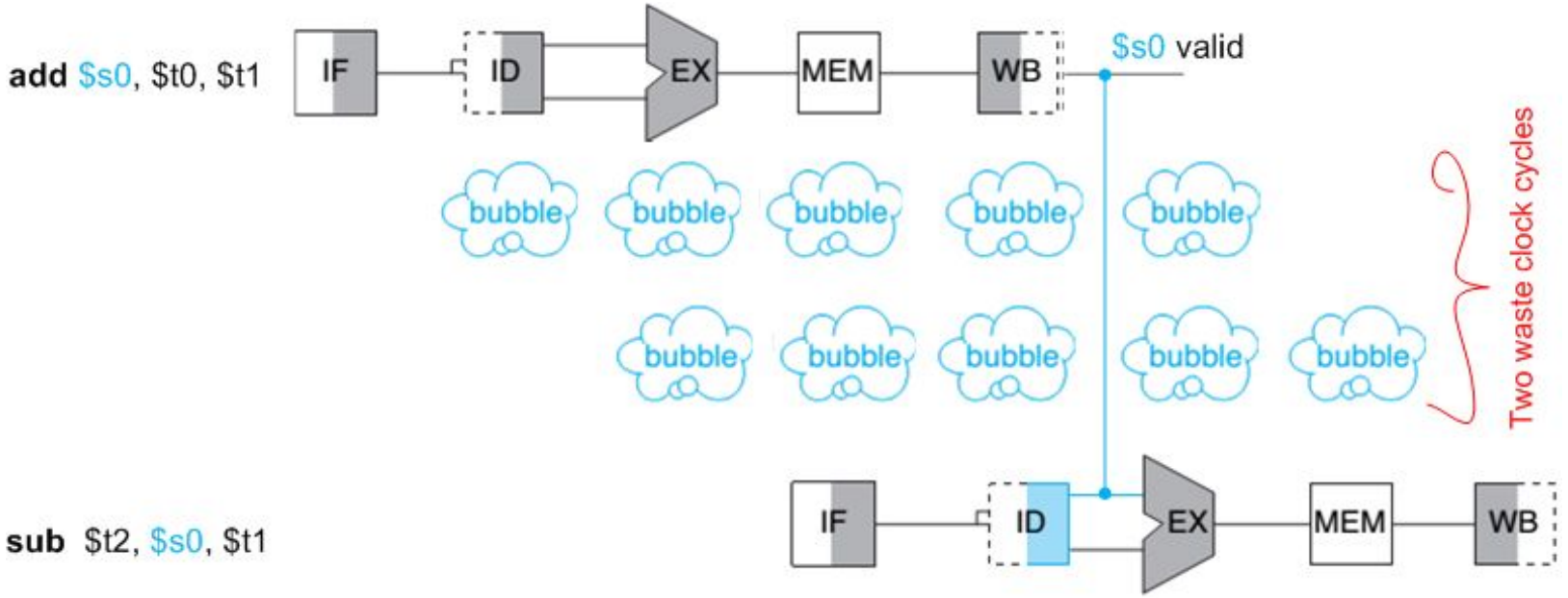
Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

Ví dụ cho đoạn lệnh sau: *add $s0, $t0, $t1*

sub $t2, $s0, $t3

Một cách giải quyết có thể trong trường hợp này là chờ **thêm hai chu kỳ xung xung clock** thì lệnh *sub* mới được nạp vào

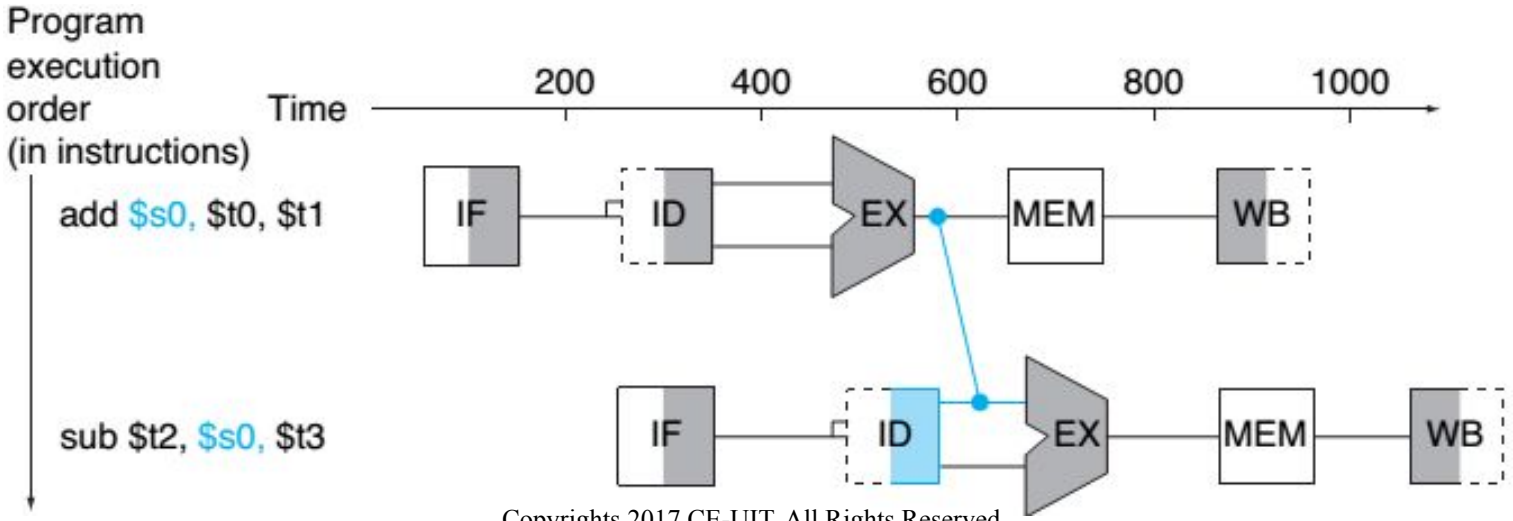


Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

- Thay vì chờ một số chu kỳ đến khi dữ liệu cần sẵn sàng, một kỹ thuật có thể được áp dụng để rút ngắn số chu kỳ rỗi, gọi là **kỹ thuật nhìn trước (forwarding** hay **bypassing).**

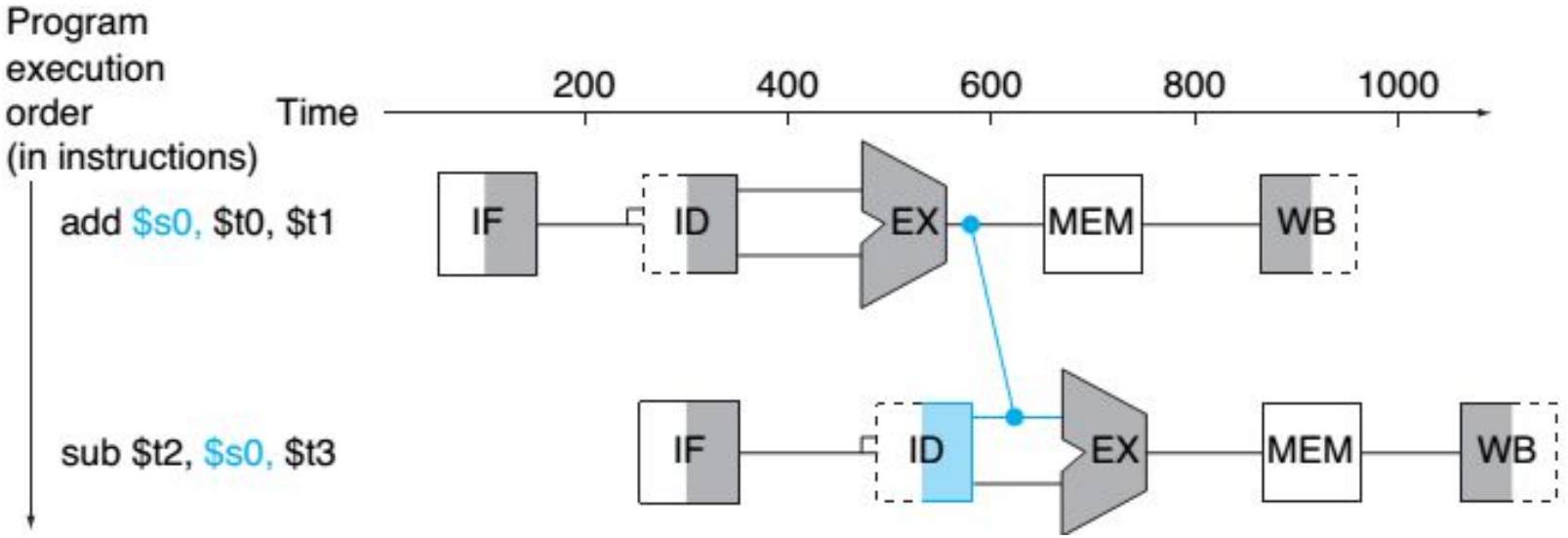
Như trong ví dụ trước, thay vì chờ sau hai chu kỳ rỗi mới nạp lệnh *sub* vào, ngay khi ALU hoàn thành tính toán tổng cho lệnh *add* thì tổng này cũng được cung cấp ngay cho công đoạn EX của lệnh *sub* (thông qua một bộ đệm dữ liệu gắn thêm bên trong) để ALU tính toán kết quả cho *sub* nhanh.



Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

- Kỹ thuật nhìn trước: một phương pháp giải quyết xung đột dữ liệu bằng đưa thêm vào các bộ đệm phụ bên trong, các dữ liệu cần có thể được truy xuất từ bộ đệm này hơn là chờ đợi đến khi nó sẵn sàng trong bộ nhớ hay trong thanh ghi.



Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

Lưu ý, với lệnh ***lw*** và các lệnh có chức năng tương tự, thông thường kết quả cuối của nó không phải khi hoàn tất công đoạn EX mà là khi hoàn tất công đoạn MEM.

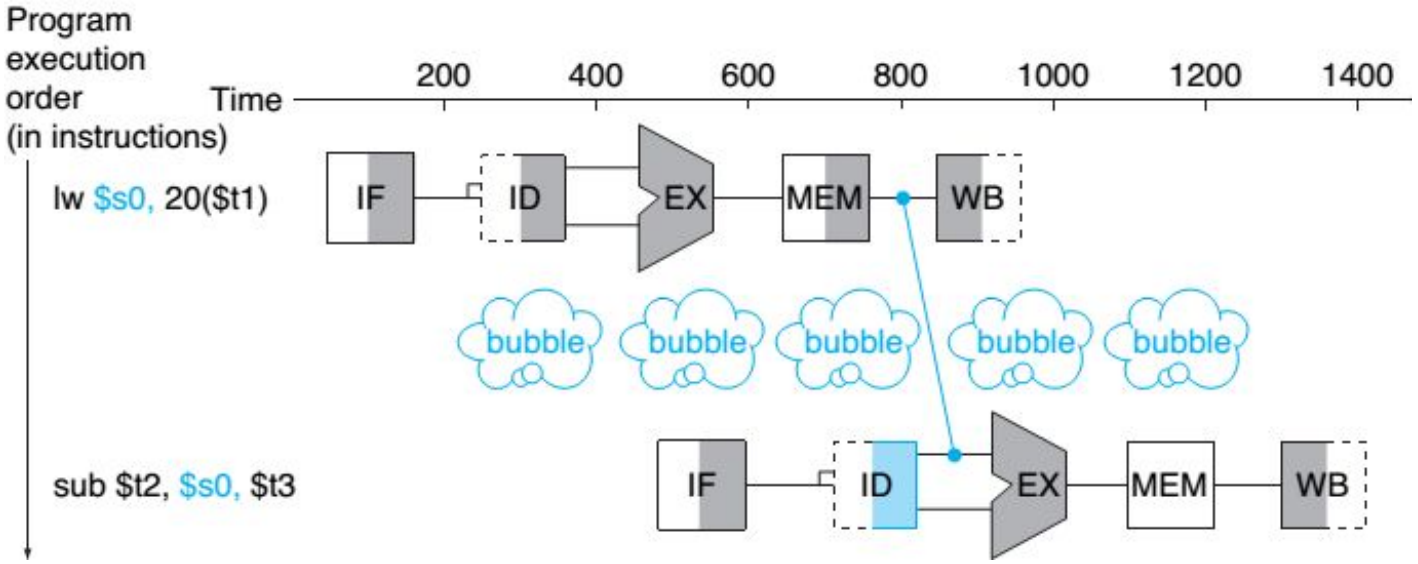
Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

Xét ví dụ sau: *lw $s0, 20($t1)*

sub $t2, $s0, $t3

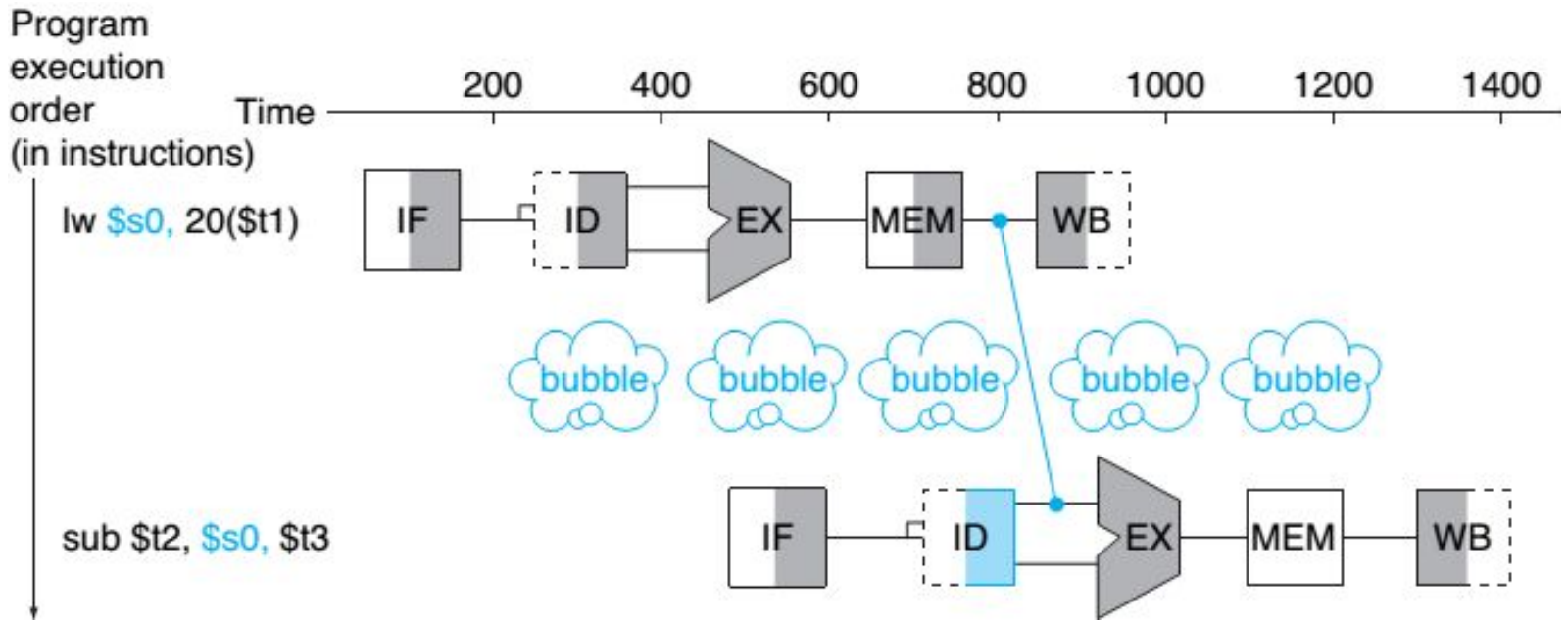
Với lệnh *lw,* dữ liệu mong muốn sẽ chỉ sẵn sàng sau 4 chu kỳ pipeline (tức sau khi công đoạn MEM hoàn tất). Vì vậy, giả sử dữ liệu đầu ra của công đoạn MEM của lệnh *lw* được truyền tới đầu vào công đoạn EX của lệnh sub theo sau, thì lệnh *sub* vẫn phải chờ sau một chu kỳ rồi mới được nạp vào.



Kỹ thuật ống dẫn (pipeline)

Xung đột dữ liệu

Kỹ thuật forwarding có thể hỗ trợ giải quyết xung đột dữ liệu hiệu quả, tuy nhiên nó không thể ngăn chặn tất cả các trường hợp chu kỳ rỗi

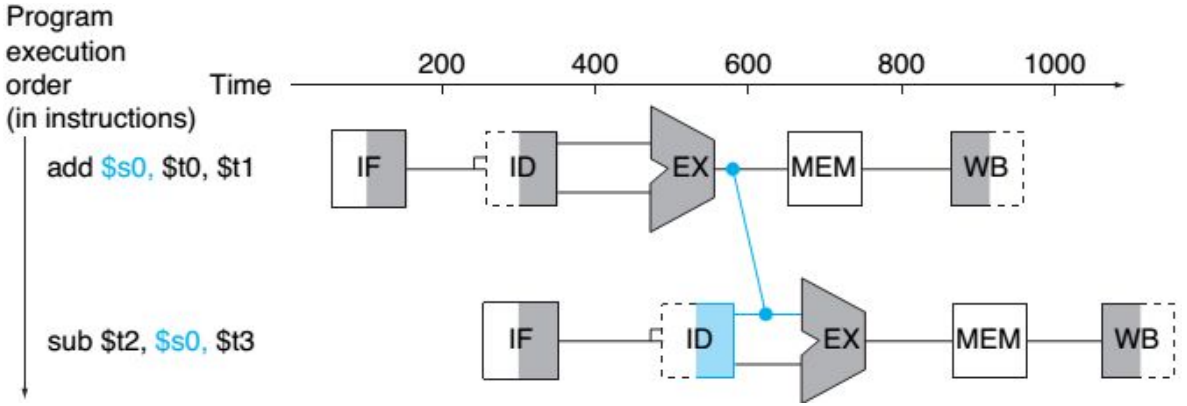


Kỹ thuật ống dẫn (pipeline)

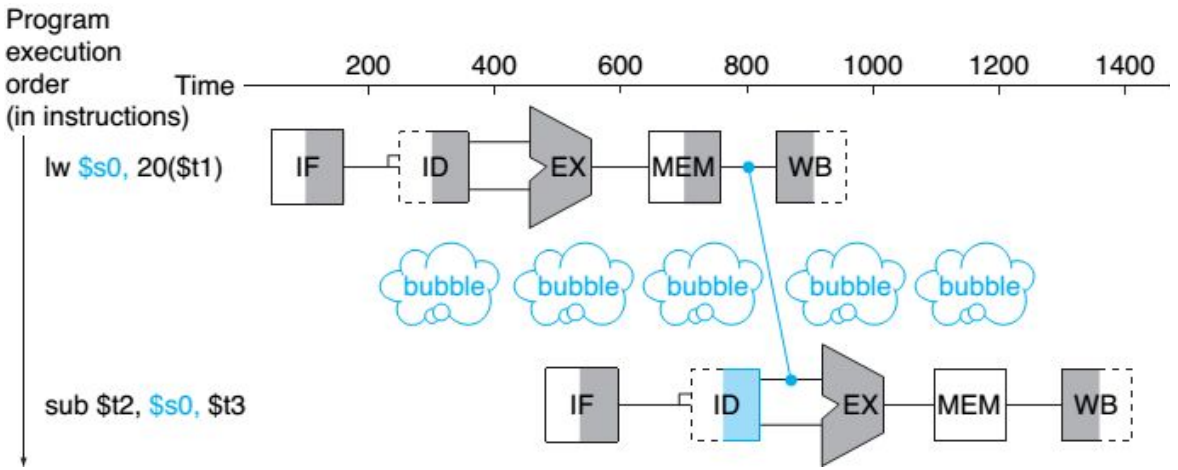
Xung đột dữ liệu

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.

Kỹ thuật ống dẫn (pipeline)

Xung đột điều khiển

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

Ví dụ xét đoạn chương trình sau:

```
add $4, $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 khiển.

Kỹ thuật ống dẫn (pipeline)

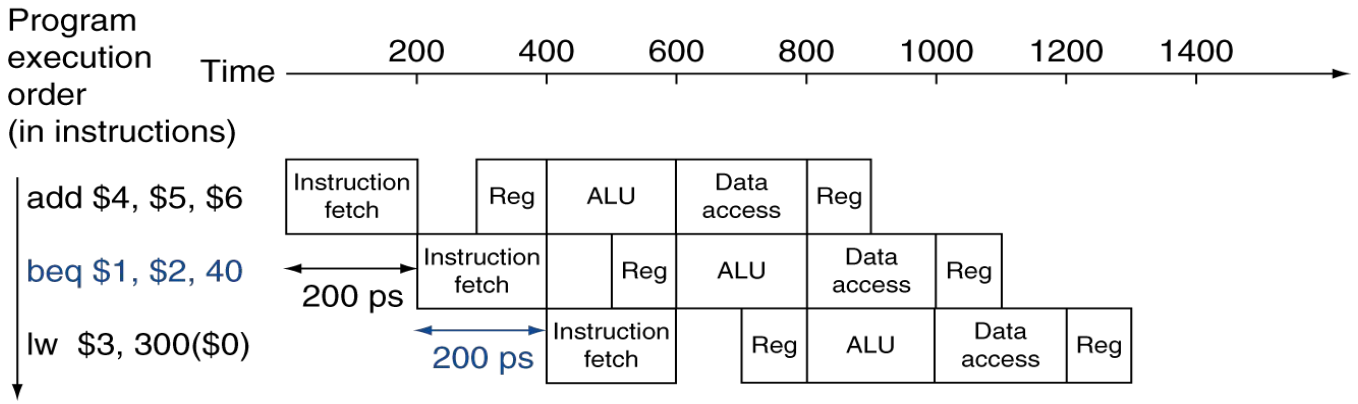
Xung đột điều khiển

- Các giải pháp giải quyết xung đột điều khiển cho lệnh nhảy có điều kiện, ví dụ với **beq**:
  - Cơ bản nhất là chờ cho tới khi điều kiện bằng được tính toán thì lệnh tiếp theo mới được nạp vào. Luôn lãng phí một chu kỳ xung clock để chờ điều kiện bằng xảy ra.
  - Cải tiến hơn, có thể dùng phương pháp dự đoán. Có hai cách dự đoán: dự đoán điều kiện bằng sẽ xảy ra (tức nhánh nhảy tới sẽ được lấy); hoặc dự đoán điều kiện bằng sẽ không xảy ra (tức nhánh nhảy tới sẽ không được lấy). Nếu dự đoán đúng, chương trình sẽ không lãng phí chu kỳ xung clock nào; còn nếu dự đoán sai, lệnh đúng sẽ được nạp lại và lãng phí một chu kỳ xung clock.
- Các giải pháp giải quyết xung đột điều khiển (tham khảo thêm mục 4.8, sách tham khảo chính)

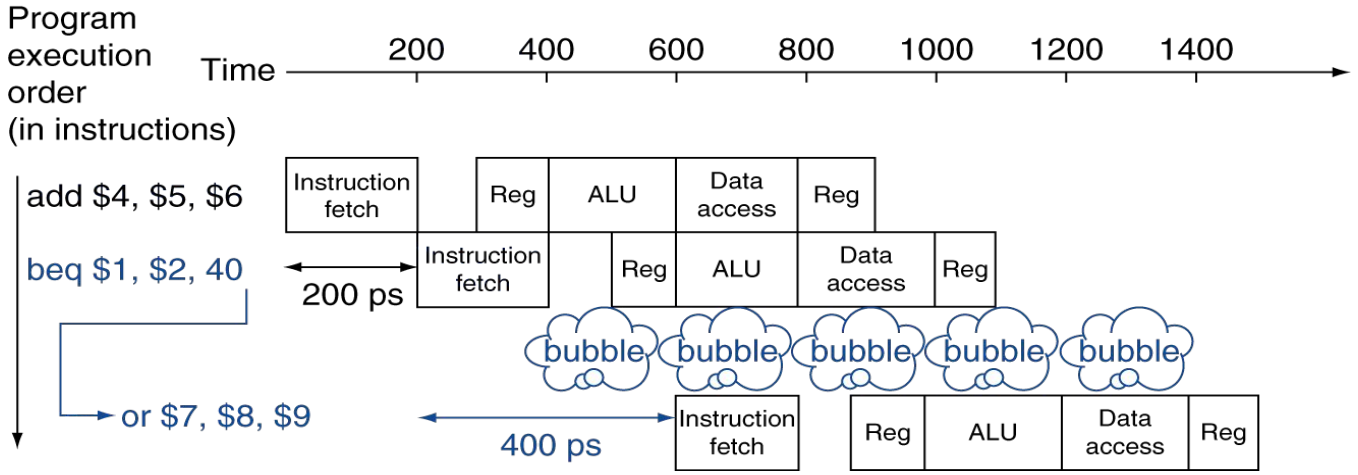
Kỹ thuật ống dẫn (pipeline)

Ví dụ giải quyết xung đột điều khiển theo kiểu dự đoán điều kiện bằng sẽ không xảy ra, tức nhánh nhảy tới sẽ không được lấy (**Predict Not Taken):**

Dự đoán đúng



Dự đoán sai



Kỹ thuật ống dẫn (pipeline)

Tổng kết:

- Hiểu kỹ thuật ống dẫn là gì
- Ba xung đột mà kỹ thuật ống dẫn có thể gây ra:
  - ✔ Xung đột cấu trúc
  - ✔ Xung đột dữ liệu
  - ✔ Xung đột điều khiển
- Cách giải quyết khi xảy ra các xung đột trên

Kỹ thuật ống dẫn (pipeline)

- **Lý thuyết: Đọc sách tham khảo**
  - Mục: 4.5
  - Sách: *Computer Organization and Design: The Hardware/Software Interface*, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.

- **Bài tập:** file đính kèm