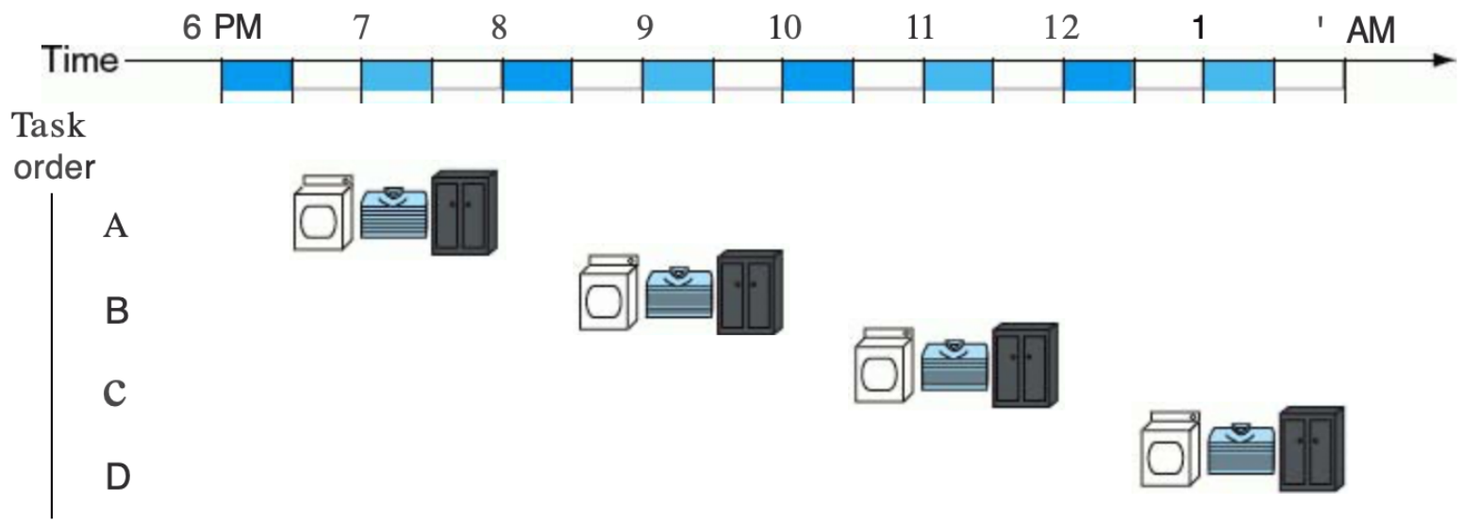
Thực hành ICT4

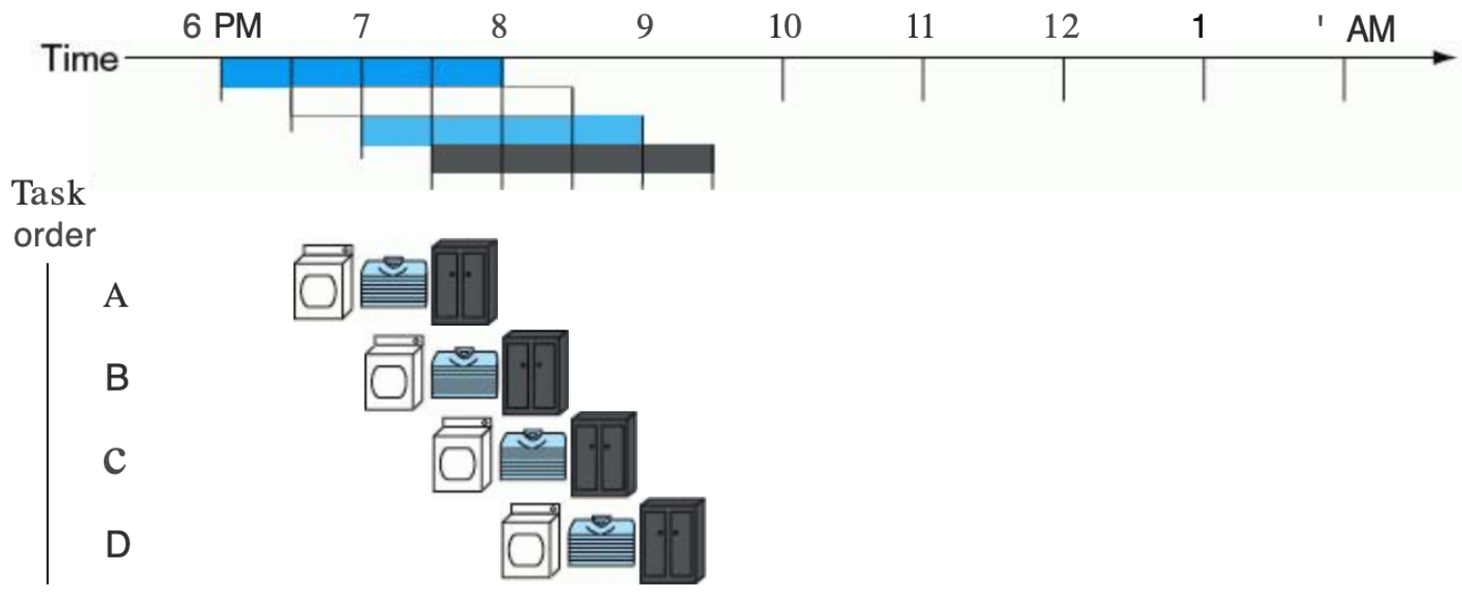
TS. Nguyễn Thị Thanh Nga
Bộ môn KTMT
Viện CNTT&TT

Tuần 6

Kỹ thuật pipeline

- Là kỹ thuật trong đó các lệnh được thực thi theo kiểu chồng lấn lên nhau (overlap)





Kỹ thuật pipeline

- Là kỹ thuật trong đó các lệnh được thực thi theo kiểu chồng lấn lên nhau (overlap)
- Tiêu tốn ít thời gian hơn cho tất cả các công việc hoàn tất.
- Chú ý: pipeline không làm giảm thời gian hoàn thành 1 công việc mà làm giảm thời gian hoàn thành tổng số công việc.

Kỹ thuật pipeline

- Khi thực thi, các lệnh MIPS được chia thành 5 giai đoạn:
  - Nạp lệnh từ bộ nhớ
  - Giải mã lệnh và đọc các thanh ghi cần thiết (MIPS cho phép đọc và giải mã đồng thời)
  - Thực thi các phép tính hoặc tính toán địa chỉ
  - Truy xuất các toán hạng trong bộ nhớ
  - Ghi kết quả cuối vào thanh ghi

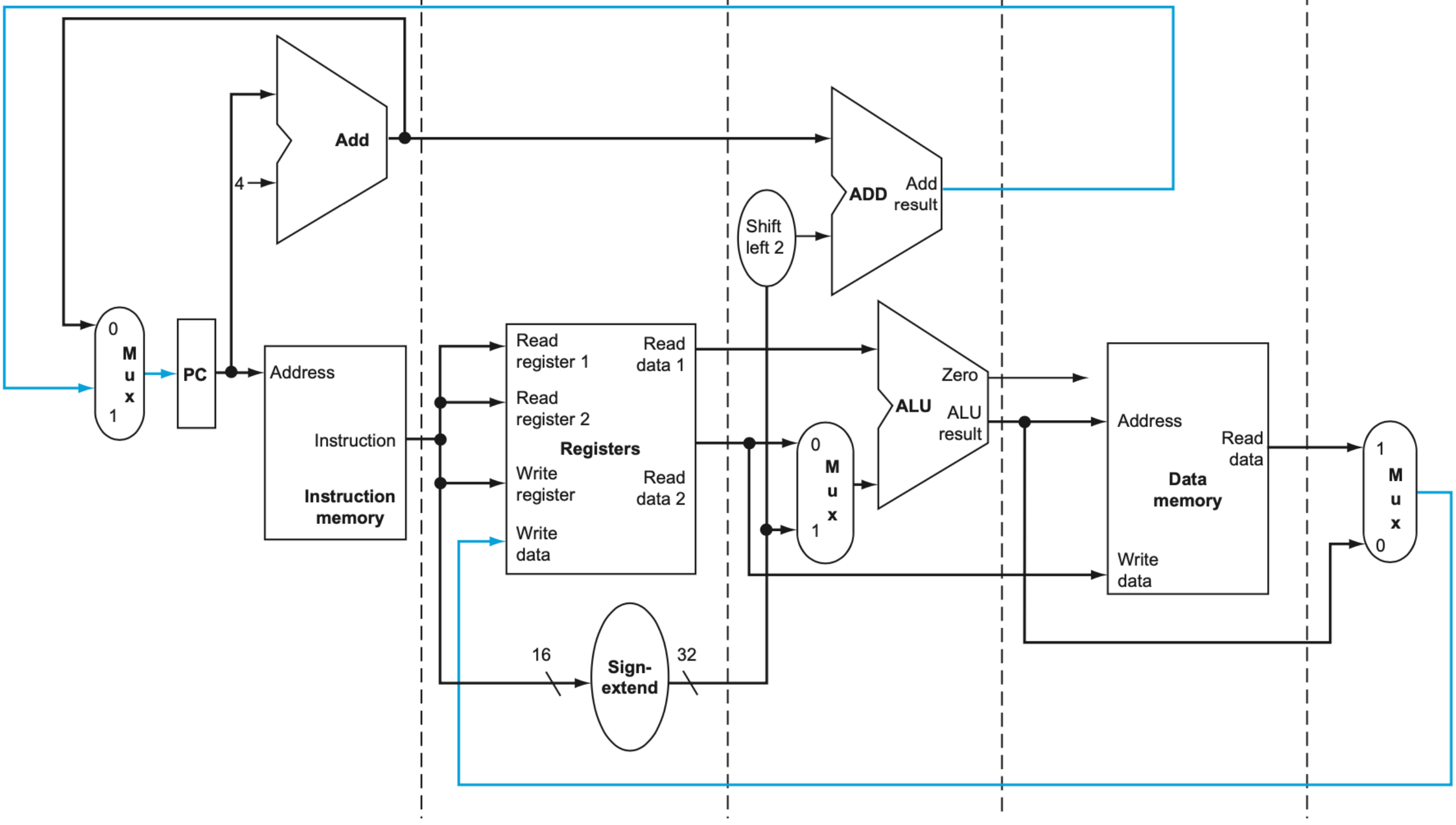
1. Giải mã lệnh – Intruction Fetch (IF)

2. Giải mã lệnh và đọc các thanh ghi – Instruction Decode (ID)

3. Thực thi – Execute (EX)

4. Truy xuất bộ nhớ – Memory (MEM)

5. Ghi kq vào thanh ghi – Write Back (WB)

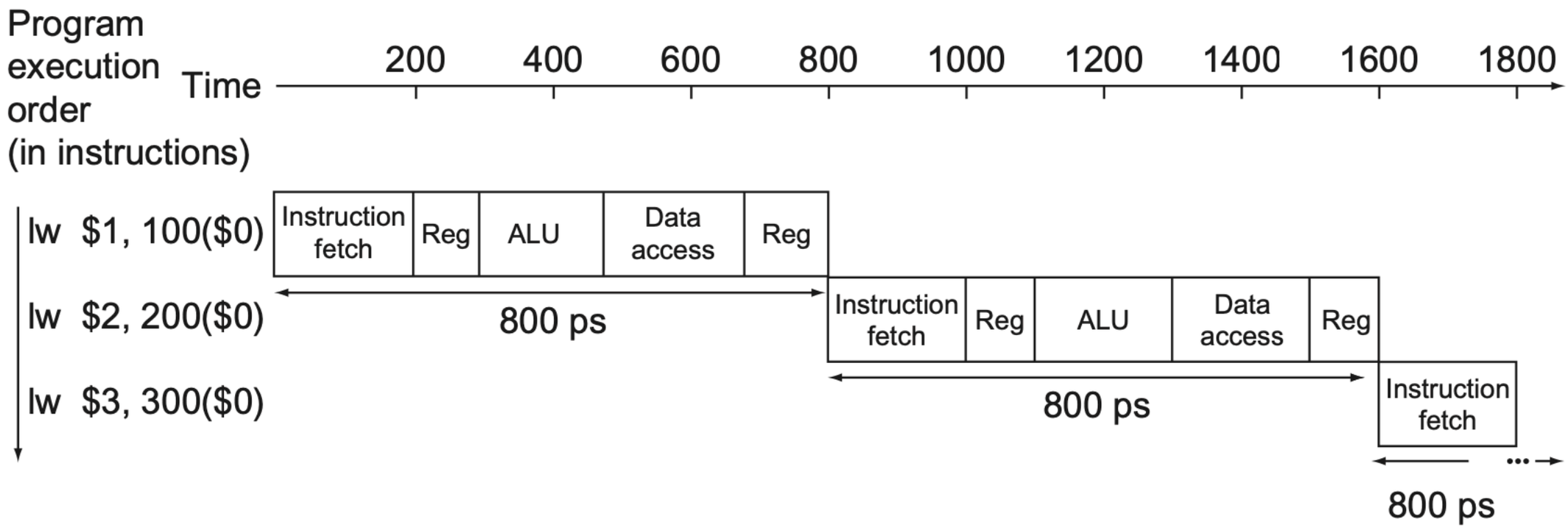


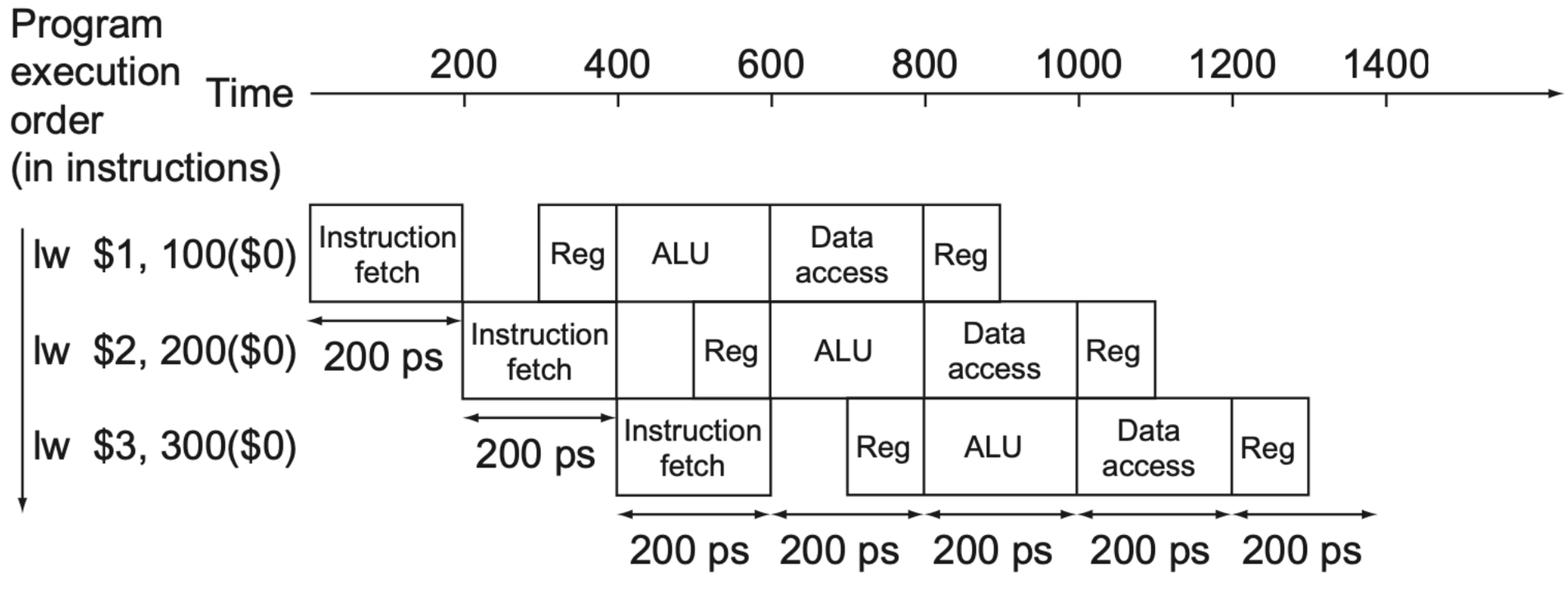
Kỹ thuật pipeline

- Xét bộ xử lý với 8 lệnh cơ bản, tương ứng với thời gian hoạt động cho từng giai đoạn như sau:

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

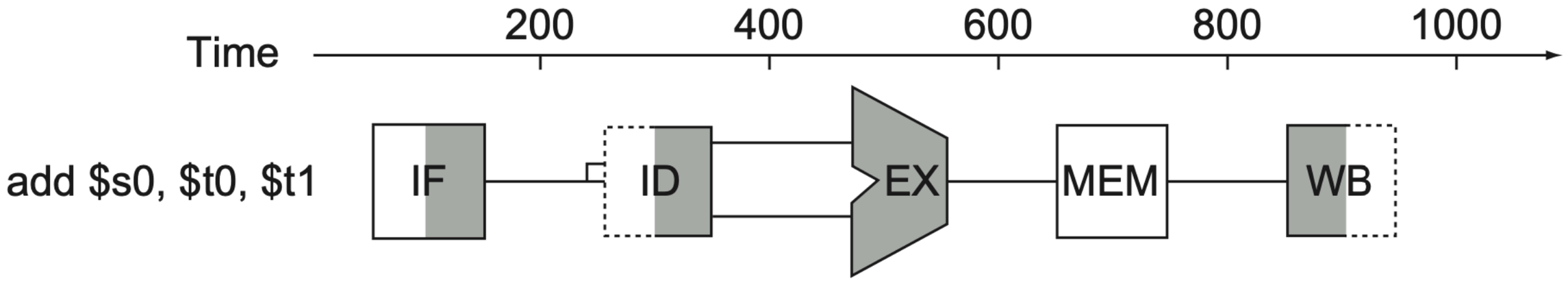
Kỹ thuật pipeline





Kỹ thuật pipeline

- Quy ước trình bày 5 giai đoạn thực thi một lệnh:



- Lệnh thực thi không làm gì được tô trắng và ngược lại được tô đen.

- Nửa phải tô đen là thao tác đọc, nửa trái tô đên là thao tác ghi.

Kỹ thuật pipeline

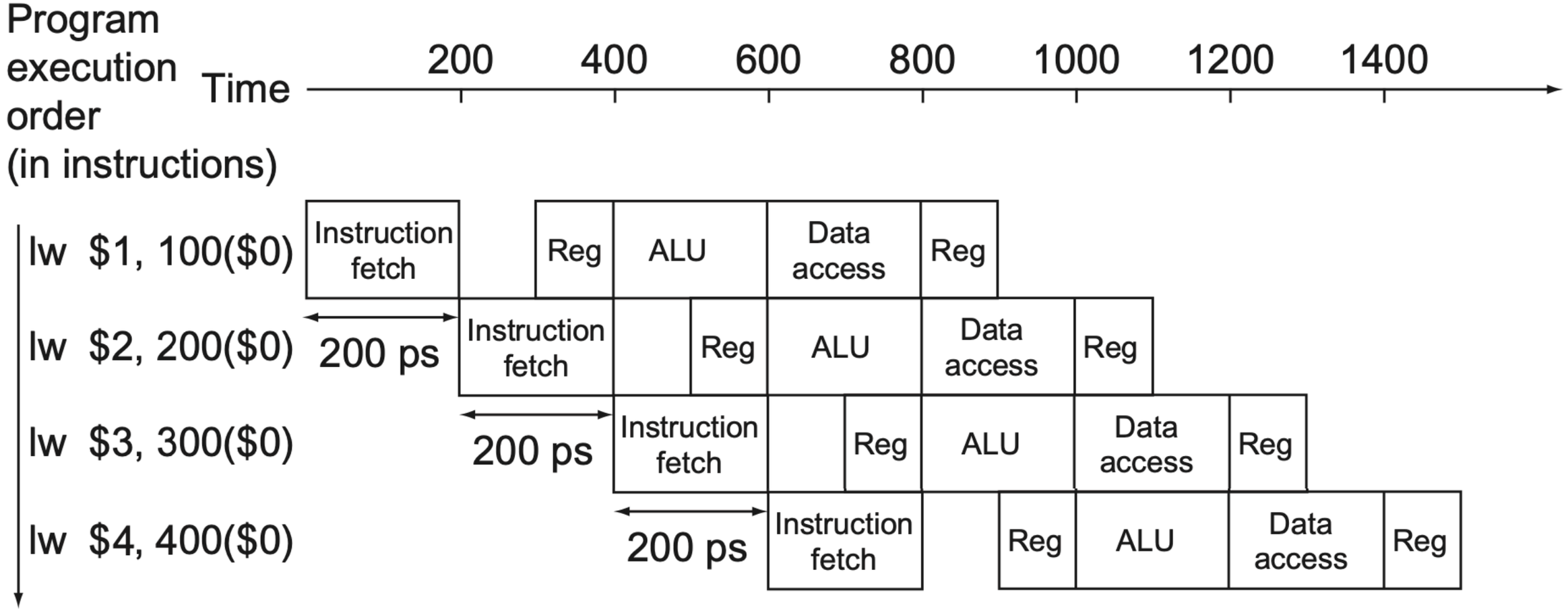
- Các xung đột có thể xảy ra khi áp dụng kỹ thuật pipeline.

- 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 kết quả sai), thường do một trong ba nguyên nhân sau:

  - Xung đột cấu trúc
  - Xung đột dữ liệu
  - Xung đột điều khiển

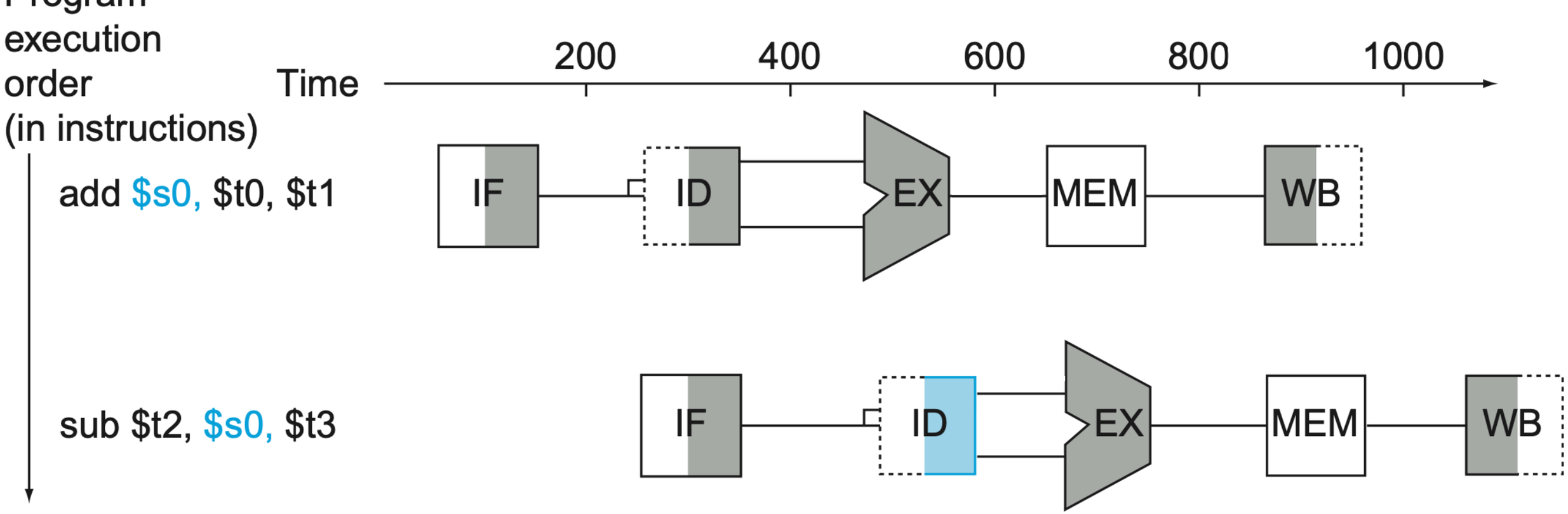
Xung đột cấu trúc

- Khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do phần cứng không thể hỗ trợ (đồng thời cùng truy xuất vào một tài nguyên phần cứng)



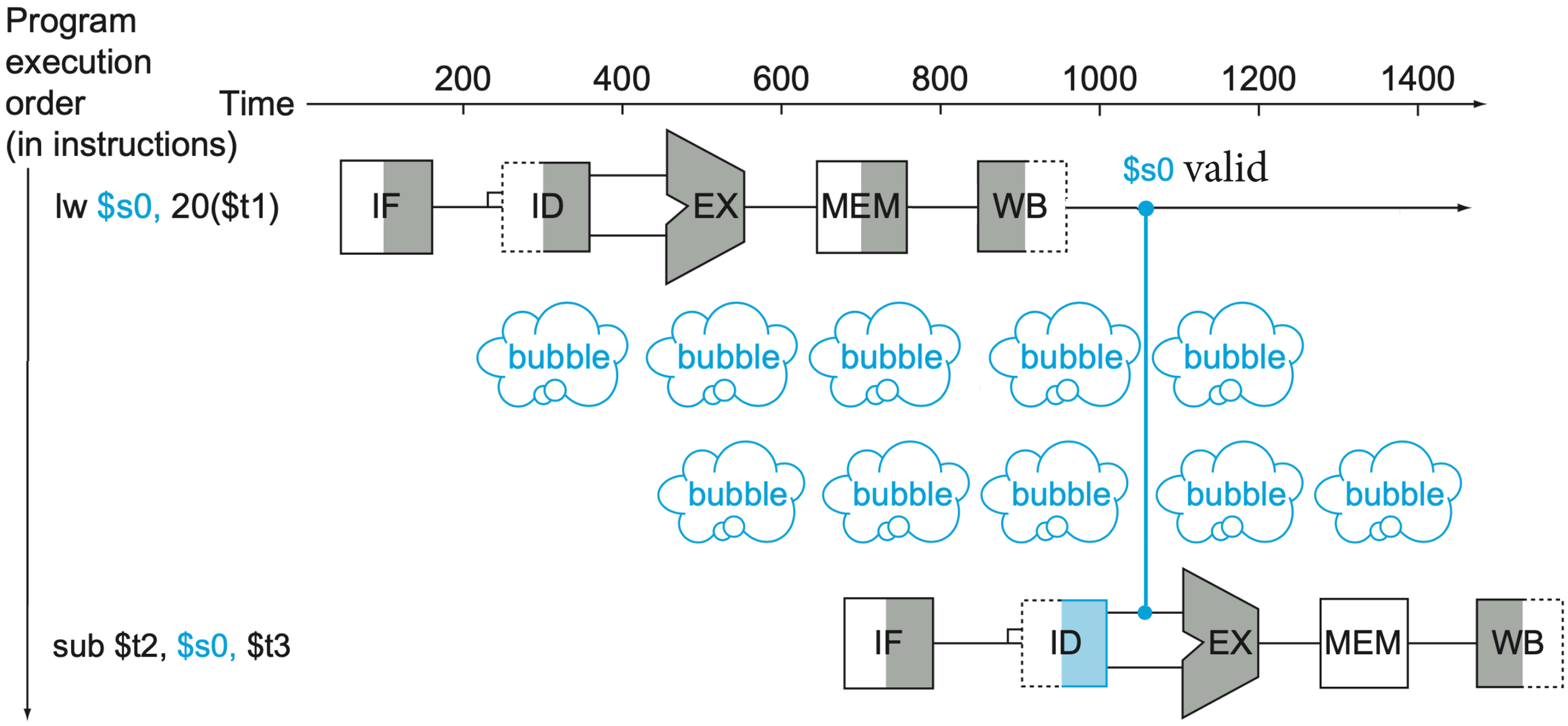
Xung đột dữ liệu

- 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 dữ liệu

- Chờ thêm 2 chu kỳ xung clock :



Xung đột điều khiển

- 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.

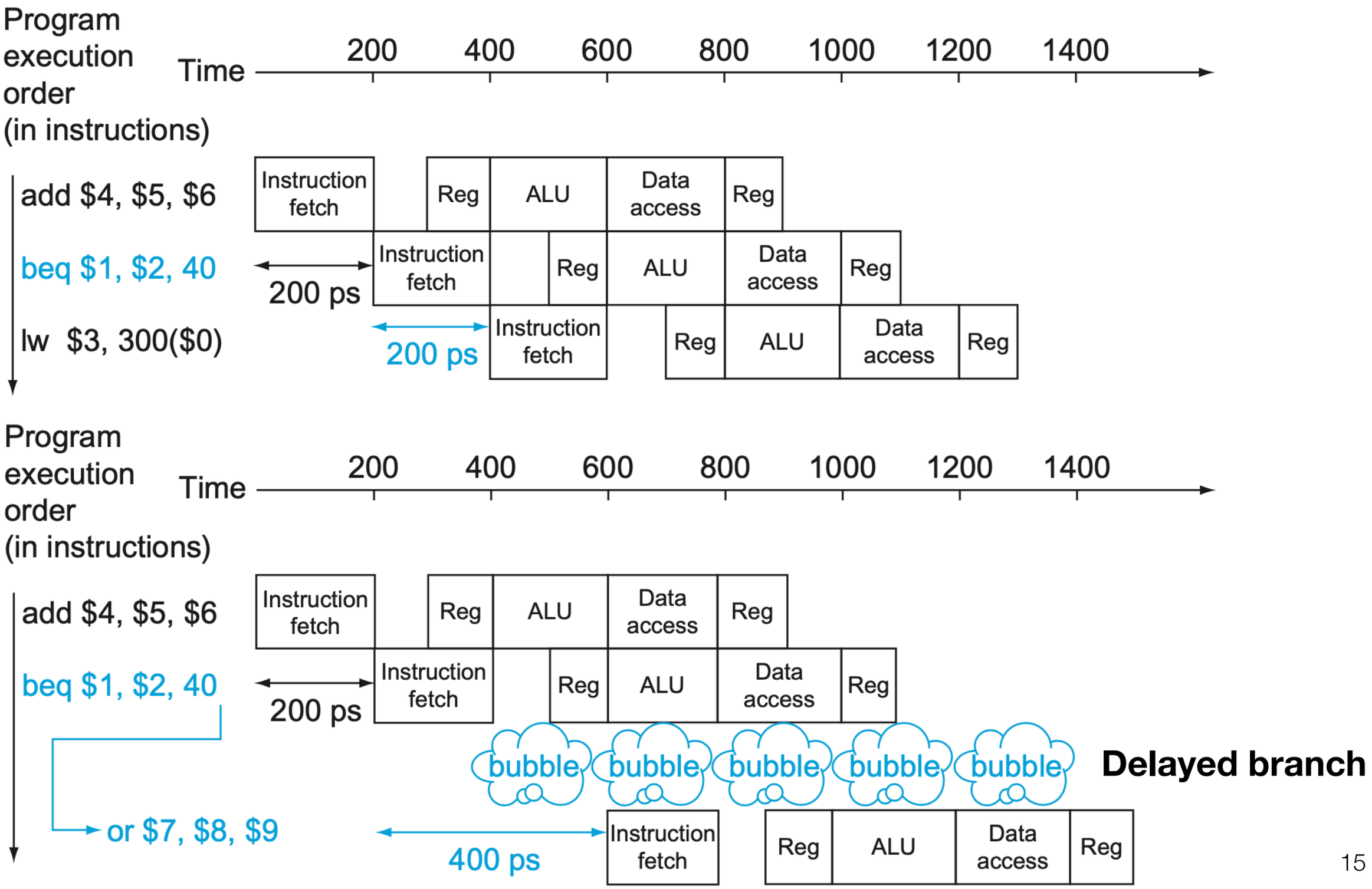
- Xét đoạn chương trình sau:

```
add $4, $5, $6

beq $1, $2, label

lw $3, 300($0)
```

Xung đột điều khiển



Mảng

- Mảng là một biến đa giá trị lưu trong một vùng nhớ liên tục chứa các phần tử có cùng kích thước.
- Cơ chế truy cập các phần tử trong mảng:

22	15	7	41	36

0x10010044

```
elemAddress = basePtr + index * size
```

Mảng

```
elemAddress = basePtr + index * size
```

- elemAddress: địa chỉ của (hay con trỏ của) phần tử truy cập
- basePtr: địa chỉ của biến mảng
- index: chỉ số của phần tử (sử dụng mảng bắt đầu từ 0)
- size: kích thước của từng phần tử

Mảng

- Để nạp phần tử ở chỉ số 0:

elemAddress=(0x10010044+(0*4))=0x10010044

→ basePtr của mảng

- Để nạp phần tử ở chỉ số 1:

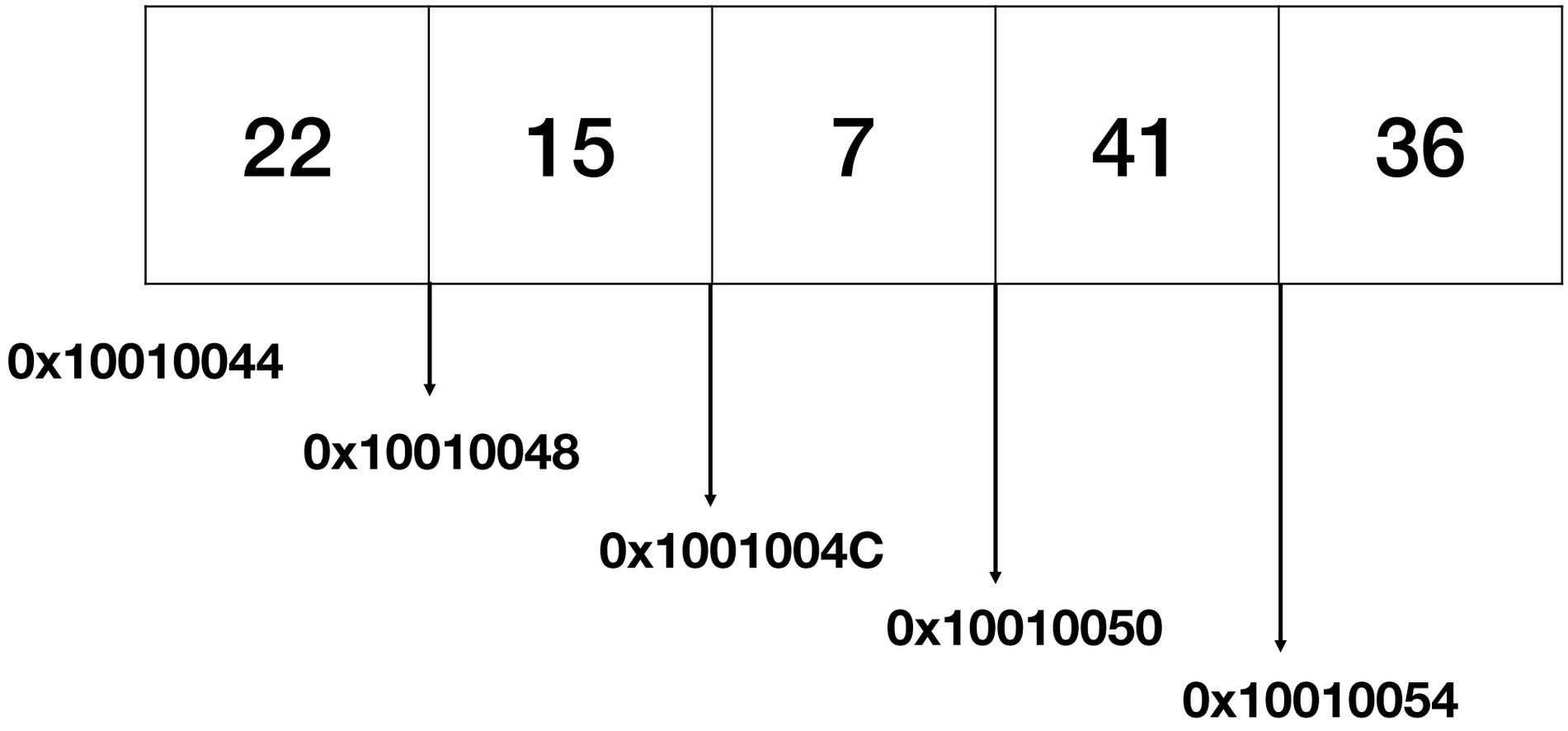
elemAddress=(0x10010044+(1*4))=0x10010048

- Để nạp phần tử ở chỉ số 2:

elemAddress=(0x10010044+(2*4))=0x1001004C

Mảng

- Cơ chế truy cập các phần tử trong mảng:




```
elemAddress = basePtr + index * size
```

Ví dụ 1

- Xét đoạn mã sau:

```
.data
.align 2
        grades: .space 40
        id: .space 10
```

- Mảng 10 words (4 bytes) số nguyên, tổng cộng 40 bytes
- Mảng 10 bytes

Ví dụ 1

- Truy cập các phần tử trong mảng

```
addi $t0, 2          # set element number 2
sll $t0, $t0, 2      # multiply $t0 by 4 (size) to get the offset
la $t1, basePtr      # $t1 is the base of the array
add $t0, $t0, $t1    # basePtr + (index * size)
lw $t2, 0($t0)       # load element 2 into $t2
```

  - grade 0: basePtr
  - grade 1: basePtr+4
  - grade 2: basePtr+8