Bài 3: Xử lý Hazard (Lệnh Load/Store)

Đoạn code ban đầu:

```
1 addi $t1, $zero, 100  # 1
2 addi $t2, $zero, 100  # 2
3 add  $t3, $t1, $t2    # 3 (Phu thuoc t1, t2)
4 lw   $t4, 0($a0)      # 4
5 lw   $t5, 4($a0)      # 5
6 and  $t6, $t4, $t5    # 6 (Phu thuoc t4, t5)
7 sw   $t6, 8($a0)      # 7 (Phu thuoc t6)
```

(a) Xác định sự phụ thuộc dữ liệu

Các cặp lệnh có quan hệ phụ thuộc (Producer → Consumer):

- $t1$: Lệnh 1 → Lệnh 3.
- $t2$: Lệnh 2 → Lệnh 3.
- $t4$: Lệnh 4 → Lệnh 6.
- $t5$: Lệnh 5 → Lệnh 6.
- $t6$: Lệnh 6 → Lệnh 7.

(b) Giải quyết bằng phần mềm (Chèn Stall - Không Forwarding)

Khi không có Forwarding, lệnh sau phải đợi lệnh trước hoàn tất giai đoạn WB (Write Back) mới có thể đọc dữ liệu ở ID.

- Giữa Lệnh 2 và 3: Cần **2 stall** (để $t2$ kịp ghi).
- Giữa Lệnh 5 và 6: Cần **2 stall** (để $t5$ kịp ghi từ MEM về WB).
- Giữa Lệnh 6 và 7: Cần **2 stall** (để $t6$ kịp ghi).

⇒ **Tổng số stall cần chèn = 6 stalls.**

(c) Giải quyết bằng phần cứng (Có Forwarding)

Kỹ thuật Forwarding cho phép chuyển dữ liệu từ EX/MEM hoặc MEM/WB về đầu EX.

- Lệnh 2 → 3 (ALU-ALU): Giải quyết triệt để (0 stall).
- Lệnh 5 → 6 (`lw` → `and`): Đây là **Load-Use Hazard**. Dữ liệu chưa có ở đầu EX của lệnh sau dù có forwarding. Bắt buộc chèn **1 stall**.
- Lệnh 6 → 7 (ALU-Store): Forwarding giải quyết được (0 stall).

⇒ **Tổng số stall = 1 stall.**

(d) Control Hazard

Đoạn chương trình không chứa các lệnh rẽ nhánh (`beq`, `bne`, `j`), do đó **không có stall do Control Hazard**.

(e) Sắp xếp lại code (Code Scheduling)

Để loại bỏ 1 stall do Load-Use Hazard (giữa Lệnh 5 và 6), ta chèn một lệnh độc lập vào giữa chúng. Lệnh 3 (`add`) là ứng viên phù hợp vì nó độc lập với các lệnh load.

Code sau khi sắp xếp (0 Stall):

```
addi $t1, $zero, 100
addi $t2, $zero, 100
lw   $t4, 0($a0)
lw   $t5, 4($a0)       # Load t5
add  $t3, $t1, $t2     # Chen vao day de lap khoang trong
and  $t6, $t4, $t5     # And su dung t5 (da san sang nho
   Forwarding)
sw   $t6, 8($a0)
```