CHƯƠNG 3. XỬ LÝ XEN KẼ DÒNG MÃ LỆNH VÀ CACHE

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1 Khuôn dạng lệnh và quá trình xử lý lệnh (Kiến trúc tập lệnh)

3.2 Cơ chế xử lý xen kẽ dòng mã lệnh (Kỹ thuật đường ống-Pipeline)

- Các vấn đề pipeline: xung đột dữ liệu (data hazards), xung đột rẽ nhánh (branch hazards), và xung đột tài nguyên (resource hazards).
- Xử lý siêu xen kẽ dòng mã lệnh (hyper-pipeline)

3.3 Bộ nhớ Cache

- Nguyên lý và tổ chức
- Chính sách thay thế dòng và hiệu năng bộ nhớ Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

- SAN, NAS, Object Storage

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1 Khuôn dạng lệnh

3.1.1 Giới thiệu về tập lệnh máy tính

- Lệnh máy tính (computer instruction):
  - Là một từ nhị phân (binary word);
  - Mỗi lệnh được gán một nhiệm vụ cụ thể;
  - Lệnh được lưu trữ trong bộ nhớ
  - Lệnh được đọc (fetch) từ bộ nhớ vào CPU để giải mã và thực hiện.
- Tập lệnh gồm nhiều lệnh có thể được chia thành một số nhóm theo chức năng:
  - Chuyển dữ liệu (data movement)
  - Tính toán (computational)
  - Điều kiện & rẽ nhánh (conditonal and branching)
  - Các lệnh khác…

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

- Việc thực hiện lệnh có thể được chia thành các pha (phase) hay giai đoạn (stage). Mỗi lệnh có thể được thực hiện theo 4 giai đoạn:
  - **Đọc lệnh** (Instruction fetch - **IF**): lệnh được đọc từ bộ nhớ về CPU;
  - **Giải mã** (Instruction decode - **ID**): CPU giải mã lệnh;
  - **Thực hiện** (Instruction execution – **EX**): CPU thực hiện lệnh;
  - **Lưu kết quả** (Write back - **WB**): kết quả thực hiện lệnh (nếu có) được lưu vào bộ nhớ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

- Chu kỳ thực hiện lệnh (Instruction execution cycle): là khoảng thời gian mà CPU thực hiện xong một lệnh:
  - Một chu kỳ thực hiện lệnh có thể gồm một số giai đoạn thực hiện lệnh;
  - Một giai đoạn thực hiện lệnh có thể gồm một số chu kỳ máy;
  - Một chu kỳ máy có thể gồm một số chu kỳ đồng hồ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

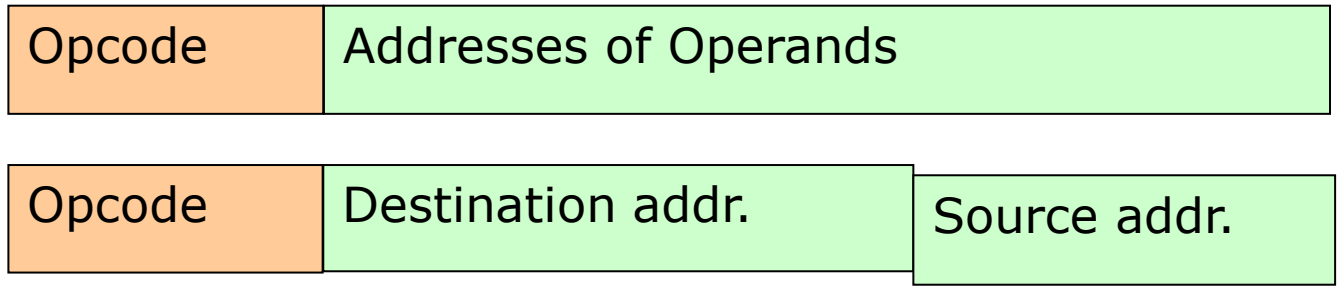
- Chu kỳ thực hiện lệnh có thể gồm các thành phần sau:
  - Chu kỳ đọc lệnh
  - Chu kỳ đọc bộ nhớ (dữ liệu)
  - Chu kỳ ghi bộ nhớ (dữ liệu)
  - Chu kỳ đọc thiết bị ngoại vi
  - Chu kỳ ghi thiết bị ngoại vi
  - Chu kỳ bus rỗi.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

➢ Dạng tổng quát của lệnh gồm 2 thành phần chính:

- Mã lệnh (**Opcode** - operation code): mỗi lệnh có mã lệnh riêng
- Địa chỉ của các toán hạng (**Addresses of Operands**): mỗi lệnh có thể gồm một hoặc nhiều toán hạng. Có thể có các dạng địa chỉ toán hạng sau:
  - 3 địa chỉ
  - 2 địa chỉ
  - 1 địa chỉ
  - 1,5 địa chỉ
  - 0 địa chỉ



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

- Toán hạng **3 địa chỉ**:
  - Dạng:
    - opcode addr1, addr2, addr3
    - Mỗi địa chỉ addr1, addr2, addr3 tham chiếu đến một ô nhớ hoặc một thanh ghi.
  - Ví dụ:

ADD R_1, R_2, R_3; $R_1 \leftarrow R_2 + R_3$

R_2 cộng với R_3, kết quả gán vào R_1.

R_i là thanh ghi của CPU.

ADD A, B, C; M[A] ← M[B] + M[C]

A, B, C là địa chỉ các ô nhớ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

- Toán hạng **2 địa chỉ**:
  - Dạng:
    - opcode addr1, addr2
    - Mỗi địa chỉ addr1, addr2 tham chiếu đến một ô nhớ hoặc một thanh ghi.
  - Ví dụ:

ADD R_1, R_2;

$R_1 \leftarrow R_1 + R_2$

R_1 cộng với R_2, kết quả gán vào R_1.

R_i là thanh ghi của CPU.

ADD A, B;

M[A] ← M[A] + M[B]

A, B là địa chỉ các ô nhớ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

- Toán hạng **1 địa chỉ**:
  - Dạng:
    - opcode addr1
    - Địa chỉ addr1 tham chiếu đến một ô nhớ hoặc một thanh ghi.
    - Ở dạng 1 địa chỉ, thanh ghi R_{acc} (Accumulator) được sử dụng như địa chỉ addr2 trong dạng 2 địa chỉ.
  - Ví dụ:

ADD R_1;	$R_{acc} \leftarrow R_{acc} + R_1$
	R_1 cộng với R_{acc}, kết quả gán vào R_{acc}.
	R_1 là thanh ghi của CPU.
ADD A;	$R_{acc} \leftarrow R_{acc} + M[A]$
	A là địa chỉ một ô nhớ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.1 Giới thiệu về tập lệnh máy tính

- Toán hạng **1,5 địa chỉ**:
  - Dạng:
    - opcode addr1, addr2
    - Một địa chỉ tham chiếu đến một ô nhớ và địa chỉ còn lại tham chiếu đến một thanh ghi.
    - Dạng 1,5 địa chỉ là dạng hỗn hợp giữa ô nhớ và thanh ghi.
      - Ví dụ:

ADD A, R_1; M[A] ← M[A] + R_1

Nội dung ô nhớ A cộng với R_1, kết quả lưu vào ô nhớ A.

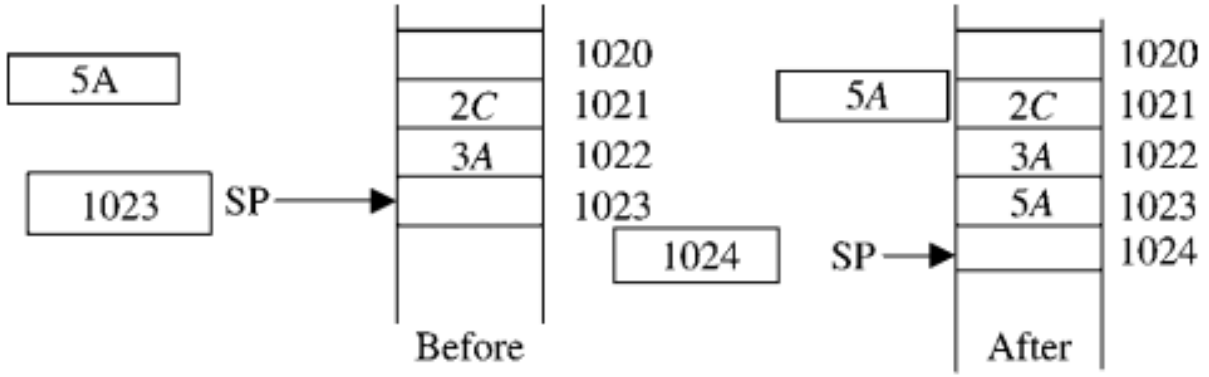
R_1 là thanh ghi của CPU và A là địa chỉ một ô nhớ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

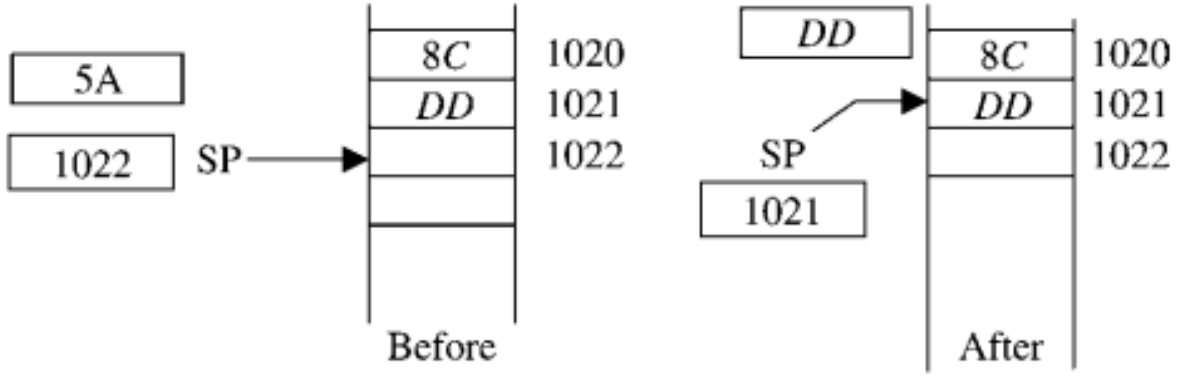
3.1.1 Giới thiệu về tập lệnh máy tính

- Toán hạng **0 địa chỉ**

Được sử dụng trong các lệnh thao tác với ngăn xếp: PUSH và POP



Stack *push* operation



Stack *pop* operation

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.2 Các chế độ định địa chỉ

- Chế độ địa chỉ (**Addressing modes**) là phương thức CPU tổ chức các toán hạng của lệnh.
  - Chế độ địa chỉ cho phép CPU kiểm tra dạng và tìm các toán hạng của lệnh.
- Các chế độ địa chỉ:
  - **Tức thì (Immediate)**
  - **Trực tiếp (Direct)**
  - **Gián tiếp qua thanh ghi (Register indirect)**
  - **Gián tiếp qua ô nhớ (Memory indirect)**
  - **Chỉ số (Indexed)**
  - **Tương đối (Relative)**

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

❖ Chế độ định địa chỉ tức thì (Immediate)

Giá trị hằng của toán hạng nguồn (source operand) nằm ngay sau mã lệnh;

Toán hạng đích có thể là 1 thanh ghi hoặc 1 địa chỉ ô nhớ;

Ví dụ:

LOAD R_1, #1000;	$R_1 \leftarrow 1000$
	Nạp giá trị 1000 vào thanh ghi R_1.
LOAD B, #500;	M[B] ← 500
	Nạp giá trị 500 vào ô nhớ B.

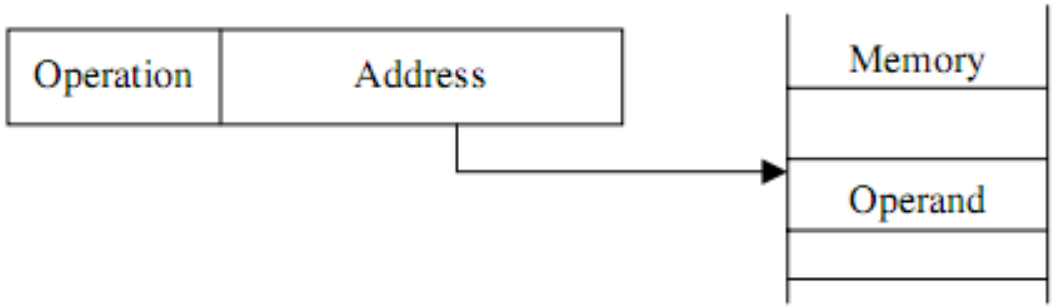
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

❖ Chế độ định địa chỉ tức thì Trực tiếp/Tuyệt đối (Direct)

Sử dụng một hằng để biểu diễn địa chỉ một ô nhớ làm một toán hạng;

Toán hạng còn lại có thể là 1 thanh ghi hoặc 1 địa chỉ ô nhớ;

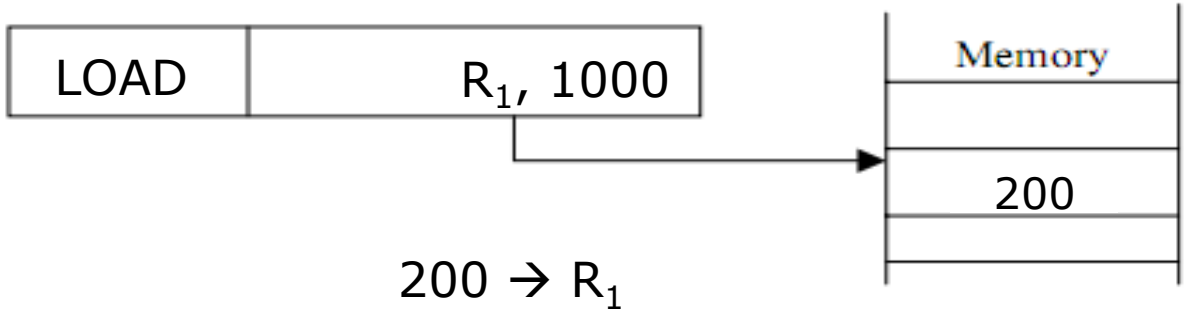
Ví dụ:



LOAD R_1, 1000; $R_1 \leftarrow M[1000]$

Nạp nội dung ô nhớ có địa chỉ 1000

vào thanh ghi R_1.



200 → R_1

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

❖ Chế độ định địa chỉ gián tiếp

Trong chế độ địa chỉ gián tiếp, một thanh ghi hoặc một ô nhớ được sử dụng để lưu địa chỉ toán hạng.

- **Gián tiếp qua thanh ghi (Register indirect)**

LOAD R_j, (R_i); $R_j \leftarrow M[R_i]$

Nạp nội dung ô nhớ có địa chỉ lưu trong thanh ghi R_i vào thanh ghi R_j.

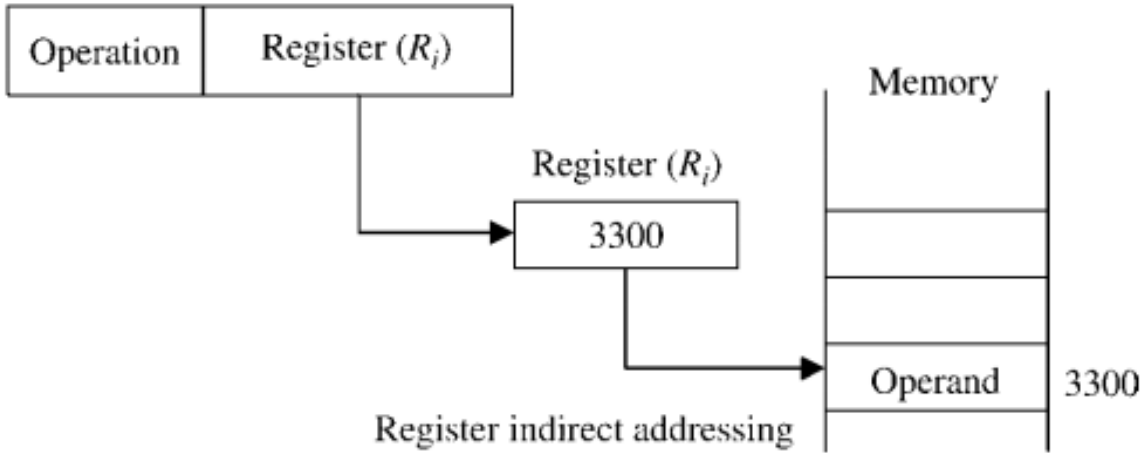
- **Gián tiếp qua ô nhớ (Memory indirect)**

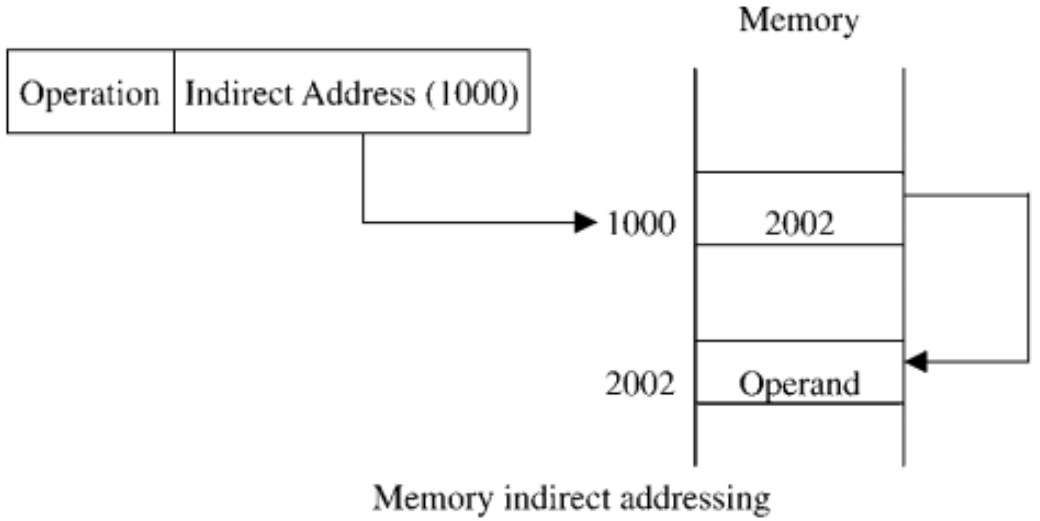
LOAD R_i, (1000); $R_i \leftarrow M[M[1000]]$

Nạp nội dung ô nhớ có địa chỉ lưu trong ô nhớ 1000 vào thanh ghi R_i.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

❖ Chế độ định địa chỉ gián tiếp





Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

❖ Chế độ định địa chỉ Chỉ số (Indexed)

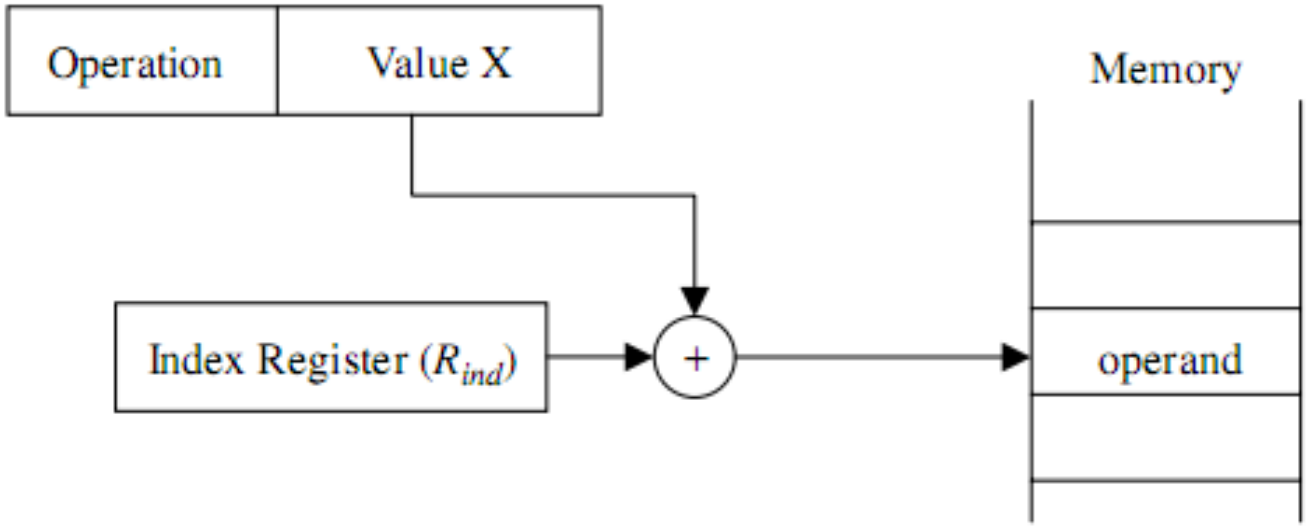
Địa chỉ của 1 toán hạng được tạo thành bởi phép cộng giữa 1 hằng và thanh ghi chỉ số (index register);

Ví dụ:

LOAD R_i, X(R_{ind});

$R_i \leftarrow M[X+R_{ind}]$

X là hằng và R_{ind} là thanh ghi chỉ số.



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

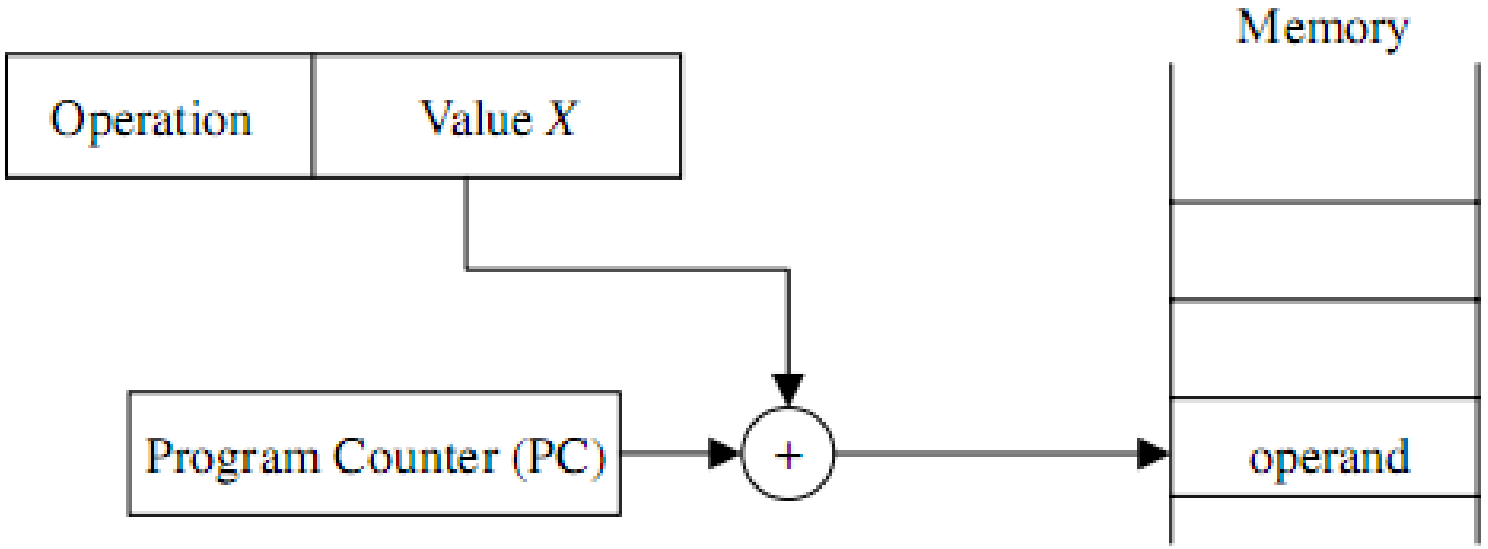
❖ Chế độ định địa chỉ Tương đối (Ralative)

Địa chỉ của 1 toán hạng được tạo thành bởi phép cộng giữa 1 hằng và bộ đếm chương trình PC (Program Counter);

Ví dụ:

LOAD R_i, X(PC); $R_i \leftarrow M[X+PC]$

X là hằng và PC là bộ đếm chương trình.



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các dạng lệnh thông dụng

- Tập lệnh máy tính có thể bao gồm một số nhóm lệnh sau:
  - Các lệnh vận chuyển dữ liệu (Data Movement Instructions)
  - Các lệnh toán học và logic (Arithmetic and Logical Instructions)
  - Các lệnh điều khiển chương trình (Control/Sequencing Instructions)
  - Các lệnh vào ra (Input/Output Instructions)

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các dạng lệnh thông dụng

- Lệnh vận chuyển dữ liệu giữa các bộ phận trong máy tính:
  - Giữa các thanh ghi của CPU:

    MOVE R_i, R_j; $R_i \leftarrow R_j$

  - Giữa 1 thanh ghi của CPU và một ô nhớ:

    MOVE 1000, R_j; $M[1000] \leftarrow R_j$

  - Giữa các ô nhớ:

    MOVE 1000, (R_j); $M[1000] \leftarrow M[R_j]$

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các dạng lệnh thông dụng

➢ Một số lệnh chuyển dữ liệu thông dụng:

- **MOVE**: chuyển dữ liệu giữa thanh ghi – thanh ghi, ô nhớ - thanh ghi và ô nhớ - ô nhớ.
- **LOAD**: nạp nội dung 1 ô nhớ vào 1 thanh ghi
- **STORE**: lưu nội dung 1 thanh ghi ra 1 ô nhớ
- **PUSH**: đẩy dữ liệu vào ngăn xếp
- **POP**: lấy dữ liệu ra khỏi ngăn xếp

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các dạng lệnh thông dụng

- Lệnh tính toán số học và logic

Các lệnh tính toán số học & logic được sử dụng để thực hiện các thao tác tính toán trên nội dung các thanh ghi và / hoặc nội dung các ô nhớ.

Ví dụ:

ADD R1, R2, R3;	$R_1 \leftarrow R_2 + R_3$
SUBSTRACT R1, R2, R3;	$R_1 \leftarrow R_2 - R_3$
ADD A, B, C;	$M[A] \leftarrow M[B] + M[C]$

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các dạng lệnh thông dụng

- Các lệnh tính toán số học thông dụng:
  - **ADD**: cộng 2 toán hạng
  - **SUBSTRACT**: trừ 2 toán hạng
  - **MULTIPLY**: nhân 2 toán hạng
  - **DIVIDE**: chia số học
  - **INCREMENT**: tăng 1
  - **DECREMENT**: giảm 1

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các dạng lệnh thông dụng

- Các lệnh logic thông dụng:
  - **NOT**: phủ định
  - **AND**: và
  - **OR**: hoặc
  - **XOR**: hoặc loại trừ
  - **COMPARE**: so sánh
  - **SHIFT**: dịch
  - **ROTATE**: quay

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các lệnh điều khiển chương trình

➢ Các lệnh điều khiển chương trình được sử dụng để thay đổi trật tự thực hiện các lệnh khác trong chương trình:

- CONDITIONAL BRANCHING (CONDITIONAL JUMP): các lệnh nhảy/ rẽ nhánh có điều kiện
- UNCONDITIONAL BRANCHING (JUMP): các lệnh nhảy/ rẽ nhánh không điều kiện
- **CALL** and **RETURN**: lệnh gọi thực hiện và trở về từ chương trình con.

  Một trong các đặc tính của các lệnh này là chúng làm thay đổi nội dung của bộ đếm chương trình PC.

  Các lệnh điều khiển chương trình sử dụng các cờ của ALU để xác định điều kiện rẽ nhánh hoặc nhảy.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các lệnh điều khiển chương trình

➢ Các lệnh điều khiển chương trình thông dụng:

- BRANCH-IF-CONDITION: chuyển đến thực hiện lệnh ở địa chỉ mới nếu điều kiện là đúng
- **JUMP**: chuyển đến thực hiện lệnh ở địa chỉ mới
- **CALL**: chuyển đến thực hiện chương trình con
- **RETURN**: trở về (từ chương trình con) thực hiện tiếp chương trình gọi.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các lệnh điều khiển chương trình

Ví dụ:

LOAD R_1, #100

LOAD R_2, #1000

LOAD R_0, #0

Loop:

ADD R0, (R_2)

INCREMENT R_2

DECREMENT R_1

BRANCH-IF-GREATER-THAN *Loop*

→ Lặp đến khi $R_1 = 0$

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các lệnh điều khiển chương trình

- **Các lệnh vào ra (I/O instructions)** được sử dụng để vận chuyển dữ liệu giữa máy tính và các thiết bị ngoại vi;
- Các thiết bị ngoại vi giao tiếp với máy tính thông qua các cổng chuyên dụng (dedicated ports). Mỗi cổng được gán một địa chỉ;
- Hai lệnh vào ra cơ bản:
  - **INPUT**: sử dụng để chuyển dữ liệu từ thiết bị vào (input devices) đến CPU;
  - **OUTPUT**: sử dụng để chuyển dữ liệu từ CPU đến thiết bị ra (output devices).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.1.3 Các lệnh điều khiển chương trình

Ví dụ lập trình: Đoạn chương trình cộng nội dung của 100 ô nhớ kề nhau bắt đầu từ địa chỉ 1000. Kết quả lưu vào ô nhớ có địa chỉ 2000.

```
        LOAD R1, #100;          R1 ← 100
        LOAD R2, #1000;         R2 ← 1000
        LOAD R0, #0;            R0 ← 0
Loop:   ADD R0, (R2);           R0 ← R0 + M[R2]
        INCREMENT R2;           R2 ← R2 + 1
        DECREMENT R1;           R1 ← R1 – 1
        BRANCH-IF-GREATER-THAN  Loop;
        ; Quay lại thực hiện lệnh sau nhãn Loop
        ; nếu R1 còn lớn hơn 0.
        STORE 2000, R0;         M[2000] ← R0
```

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.2 Cơ chế xử lý xen kẽ dòng mã lệnh (Pipeline)

Pipeline: cơ chế xử lý xen kẽ liên tục dòng mã lệnh → cơ chế đường ống

- Cơ chế ống lệnh (pipeline) hay còn gọi là cơ chế thực hiện xen kẽ các lệnh của chương trình là một phương pháp thực hiện lệnh tiên tiến, cho phép đồng thời thực hiện nhiều lệnh, do đó giảm thời gian trung bình thực hiện mỗi lệnh và tăng được hiệu năng xử lý lệnh của CPU.
- Việc thực hiện lệnh được chia thành một số giai đoạn và mỗi giai đoạn được thực thi bởi một đơn vị chức năng khác nhau của CPU. Nhờ vậy CPU có thể tận dụng tối đa năng lực xử lý của các đơn vị chức năng của mình, giảm thời gian chờ cho từng đơn vị chức năng.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.2 Cơ chế xử lý xen kẽ dòng mã lệnh (Pipeline)

Pipeline: cơ chế xử lý xen kẽ liên tục dòng mã lệnh → cơ chế đường ống

Chia chu trình lệnh thành các công đoạn và cho phép chúng thực hiện gối lên nhau (giống như dây chuyền lắp ráp).

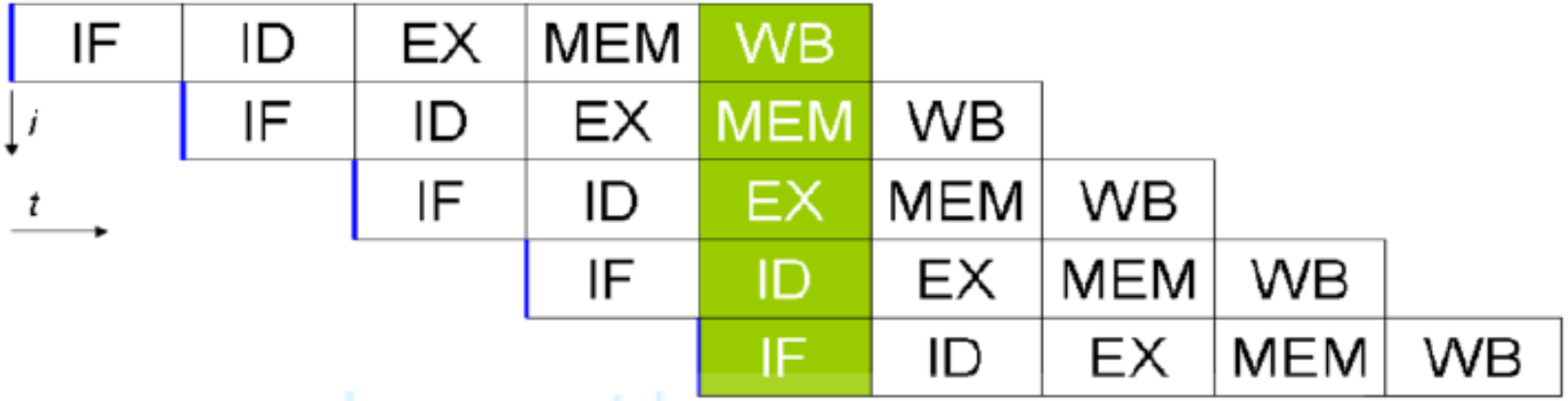
Giả sử có 6 công đoạn

- Nhận lệnh (Fetch Instruction)
- Giải mã lệnh (Decode Isntruction)
- Tính địa chỉ toán hạng (Calculate Operand Address)
- Nhận toán hạng (Fetch Operand)
- Thực hiện lệnh (Excute Instruction)
- Ghi toán hạng (Write Operand)

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.2 Cơ chế xử lý xen kẽ dòng mã lệnh (Pipeline)

(a) Không pipeline



(b) Có pipeline

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.2 Cơ chế xử lý xen kẽ dòng mã lệnh (Pipelining)

Kỹ thuật Pipelining làm tăng tốc độ thực hiện các lệnh. Tuy nhiên có một số ràng buộc:

- Cần phải có một mạch điện để thi hành mỗi giai đoạn của lệnh vì tất cả các giai đoạn của lệnh được thi hành cùng lúc.
- Phải có nhiều thanh ghi khác nhau dùng cho các tác vụ đọc và viết
- Kết quả của một tác vụ trước đó, là toán hạng nguồn của một tác vụ khác → xung đột
- Cần phải giải mã các lệnh một cách đơn giản để có thể giải mã và đọc các toán hạng trong một chu kỳ duy nhất của xung nhịp.
- Cần phải có các bộ ALU hữu hiệu để có thể thi hành lệnh số học dài nhất.
- Cần phải có nhiều thanh ghi lệnh để lưu giữ lệnh mà chúng ta phải xem xét cho mỗi giai đoạn thi hành lệnh.
- Phải có nhiều thanh ghi bộ đếm chương trình PC để có thể tiếp tục các lệnh trong trường hợp có ngắt.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.2 Cơ chế xử lý xen kẽ dòng mã lệnh (Pipelining)

Các xung đột (Hazards) khi sử dụng Pipelining

- Xung đột tài nguyên (Resource Hazads): do nhiều công đoạn dùng chung một tài nguyên
- Xung đột dữ liệu (Data Hazads): lệnh sau sử dụng dữ liệu kết quả của lệnh trước
- Xung đột rẽ nhánh (Branch Hazards): do rẽ nhánh gây ra

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột tài nguyên (Resource Hazads):**

- Vấn đề xung đột tài nguyên xảy ra khi hệ thống không cung cấp đủ tài nguyên phần cứng phục vụ CPU thực hiện đồng thời nhiều lệnh trong cơ chế ống lệnh.
- Hai xung đột tài nguyên thường gặp nhất là xung đột truy cập bộ nhớ và xung đột truy cập các thanh ghi.
- Giả sử bộ nhớ chỉ hỗ trợ một truy cập tại mỗi thời điểm và nếu tại cùng một thời điểm, có hai yêu cầu truy cập bộ nhớ đồng thời từ 2 lệnh được thực hiện trong ống lệnh (đọc lệnh – tại giai đoạn IF và đọc dữ liệu – tại giai đoạn ID) sẽ nảy sinh xung đột.
    - Điều tương tự cũng có thể xảy ra với các thanh ghi khi có 2 hay nhiều lệnh đang thực hiện đồng yêu cầu đọc/ghi cùng một thanh ghi.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột tài nguyên (Resource Hazads):**

- Giải pháp tối ưu cho vấn đề xung đột tài nguyên là nâng cao năng lực phục vụ của các tài nguyên phần cứng.
- Với xung đột truy cập bộ nhớ có thể sử dụng hệ thống nhớ hỗ trợ nhiều lệnh đọc ghi đồng thời, hoặc sử dụng các bộ nhớ tiên tiến như bộ nhớ Cache.
- Với xung đột truy cập các thanh ghi, giải pháp là tăng số lqợng thanh ghi vật lý và có cơ chế cấp phát thanh ghi linh hoạt khi thực hiện các lệnh.

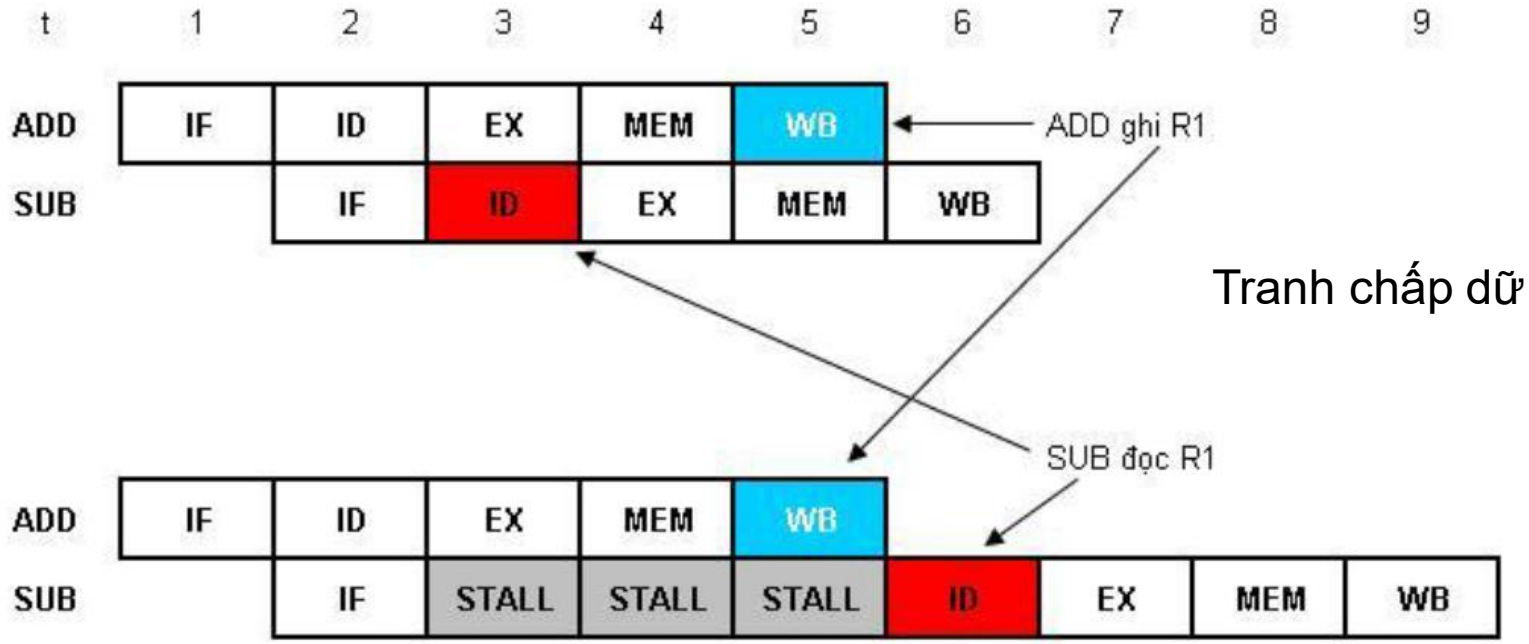
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột dữ liệu (Data Hazads):**

- Tranh chấp dữ liệu cũng là một trong các vấn đề lớn của cơ chế ống lệnh và tranh chấp dữ liệu kiểu *đọc sau khi ghi* (RAW – Read After Write) là dạng xung đột dữ liệu hay gặp nhất !

Ví dụ: Lệnh 1: **ADD** R_1, R_2, R_3 $R_1 \leftarrow R_2+R_3$

Lệnh 2: **SUB** R_4, R_1, R_2 $R_4 \leftarrow R_1-R_2$



Tranh chấp dữ liệu kiểu RAW

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột dữ liệu (Data Hazads):** Giải pháp !

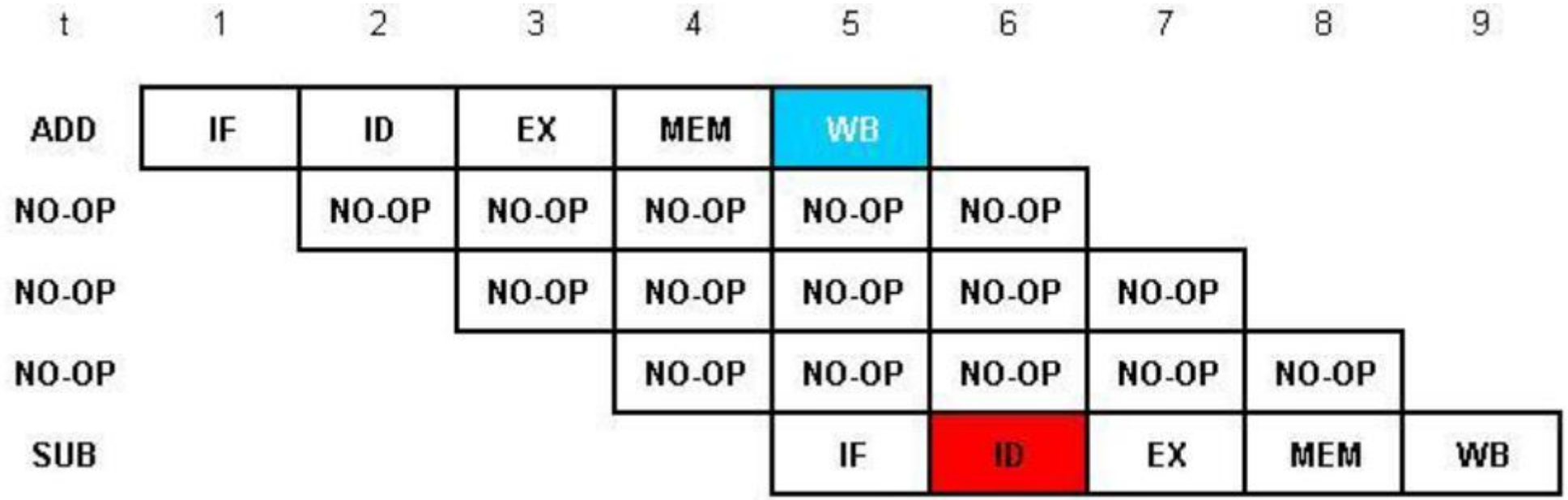
Có một số giải pháp cho vấn đề tranh chấp dữ liệu kiểu RAW

- Nhận dạng tranh chấp RAW khi nó diễn ra
- Sử dụng phần cứng để nhận dạng tranh chấp RAW và dự đoán trqớc giá trị dữ liệu phụ thuộc.
- Khi tranh chấp RAW xảy ra, tạm dừng (stall) ống lệnh cho đến khi lệnh phía trước hoàn tất giai đoạn WB.
- Có thể sử dụng trình biên dịch (compiler) để nhận dạng tranh chấp RAW và thực hiện việc:
  - ✓ Chèn thêm các lệnh **NO-OP** vào giữa các lệnh có thể gây ra tranh chấp RAW. Lệnh **NO-OP** là lệnh rỗng, không thực hiện tác vụ hữu ích mà chỉ tiêu tốn thời gian CPU.
  - ✓ Thay đổi trật tự các lệnh trong chương trình và chèn các lệnh độc lập vào giữa các lệnh có thể gây ra tranh chấp RAW.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột dữ liệu (Data Hazads):** Giải pháp !

Mục đích của cả hai phương pháp kể trên là lùi việc thực hiện lệnh gây tranh chấp dữ liệu cho đến khi lệnh trước nó hoàn tất việc lưu kết quả.

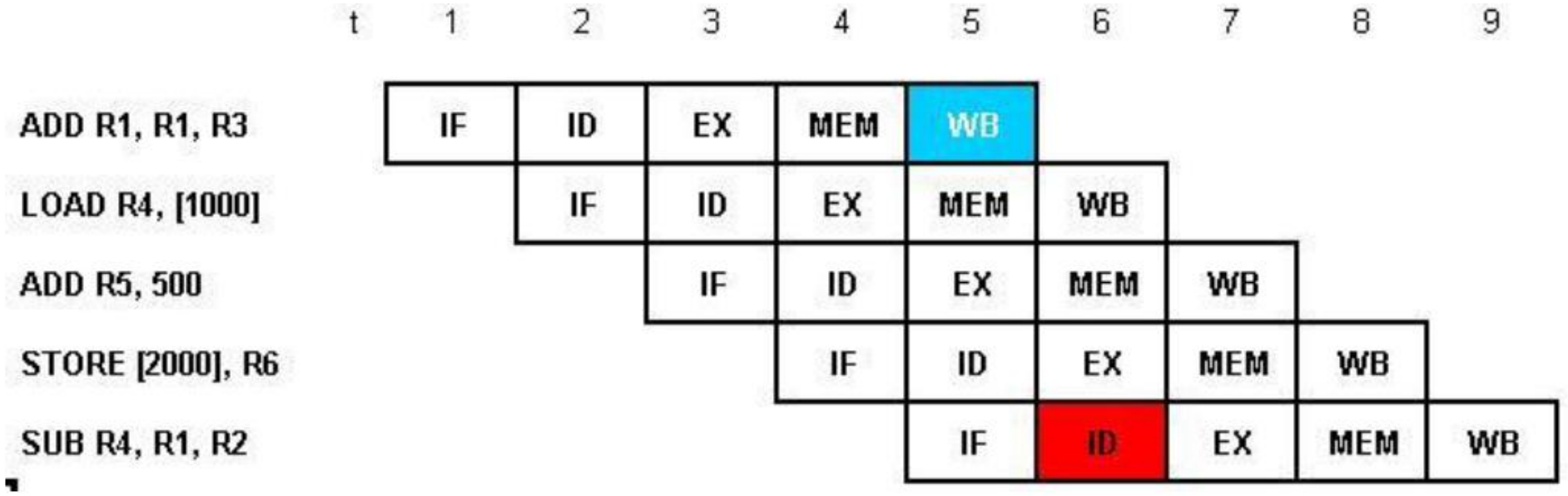


Khắc phục RAW bằng cách chèn thêm lệnh NO-OP

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột dữ liệu (Data Hazads):** Giải pháp !

Mục đích của cả hai phương pháp kể trên là lùi việc thực hiện lệnh gây tranh chấp dữ liệu cho đến khi lệnh trước nó hoàn tất việc lưu kết quả.



Khắc phục RAW bằng cách chèn thêm các lệnh độc lập

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

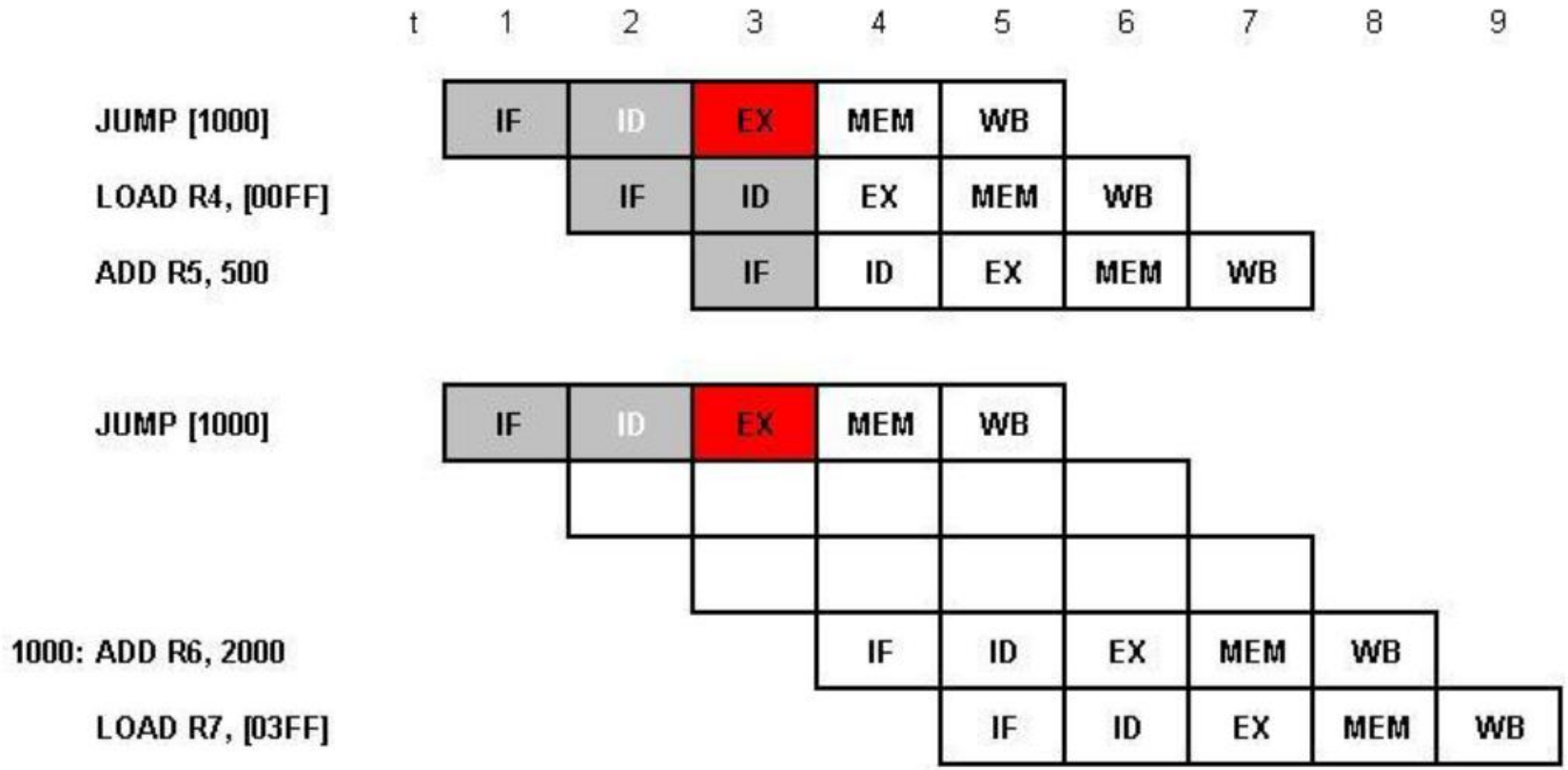
➢ **Xung đột rẽ nhánh (Branch Hazads):**

- Tỷ lệ các lệnh rẽ nhánh trong chương trình khoảng 10-30%.
- Do lệnh rẽ nhánh thay đổi nội dung của bộ đếm chương trình, chúng có thể phá vỡ tiến trình thực hiện tuần tự các lệnh trong ống lệnh vì lệnh được thực hiện sau lệnh rẽ nhánh có thể không phải là lệnh liền sau nó mà là một lệnh ở vị trí khác.
- Do kiểu thực hiện gối đầu, các lệnh liền sau lệnh rẽ nhánh đã được nạp và thực hiện dở dang trong trong ống lệnh sẽ bị đẩy ra làm cho ống lệnh bị trống rỗng và hệ thống phải bắt đầu nạp mới các lệnh từ địa chỉ đích rẽ nhánh.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Xung đột rẽ nhánh (Branch Hazads):**

- Các lệnh sau lệnh rẽ nhánh **LOAD** và **ADD** bị đẩy ra và hệ thống nạp mới các lệnh từ địa chỉ đích rẽ nhánh 1000



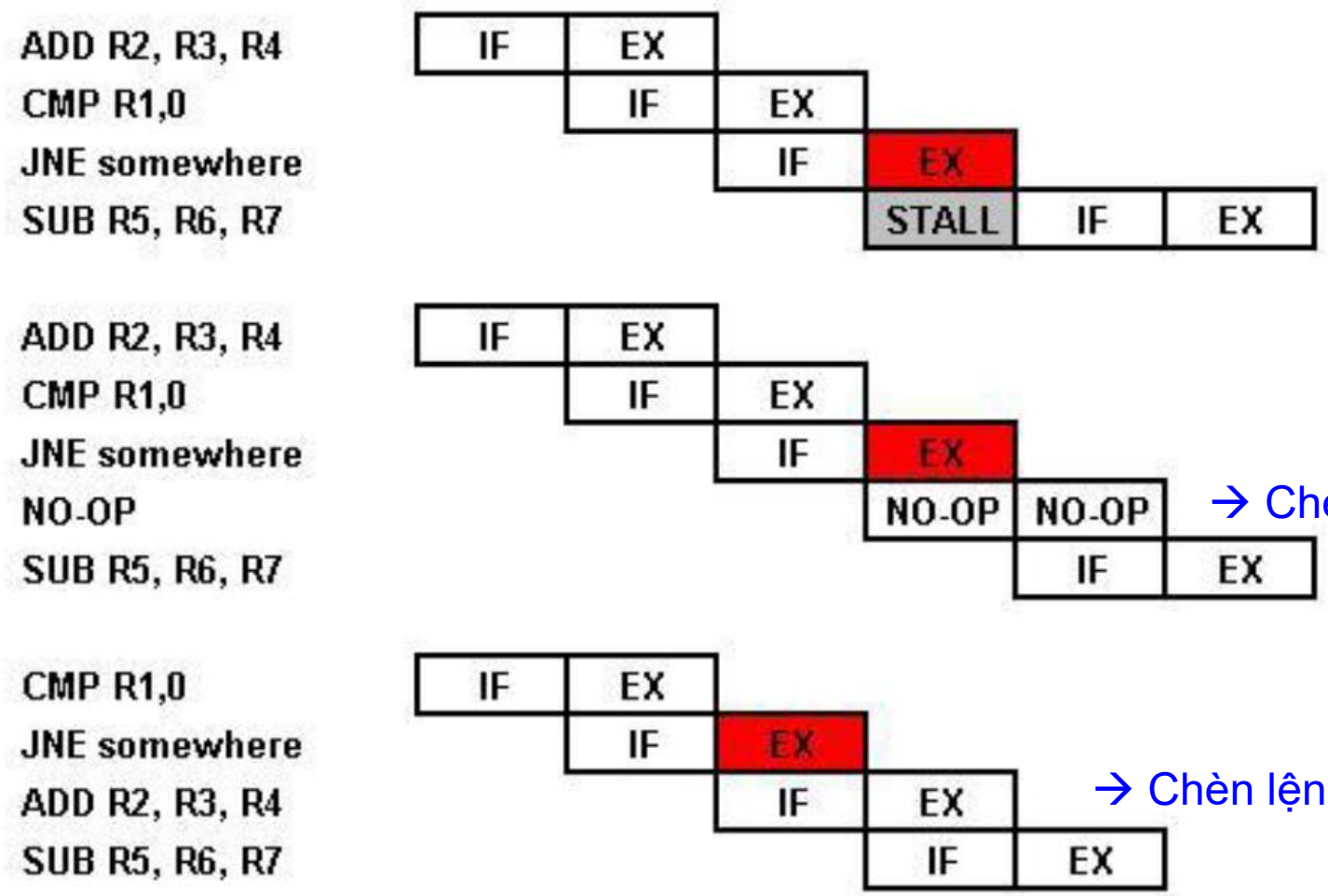
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- **Xung đột rẽ nhánh (Branch Hazads):** Giải pháp khắc phục

- Có nhiều giải pháp khắc phục các vấn đề nảy sinh do các lệnh rẽ nhánh !
- Sử dụng đích rẽ nhánh (branch targets)
- Làm chậm rẽ nhánh (delayed branching)
- Dự đoán rẽ nhánh (branch prediction)

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- **Xung đột rẽ nhánh (Branch Hazads):** Giải pháp khắc phục



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Siêu ống dẫn (Hyper Pipielining)

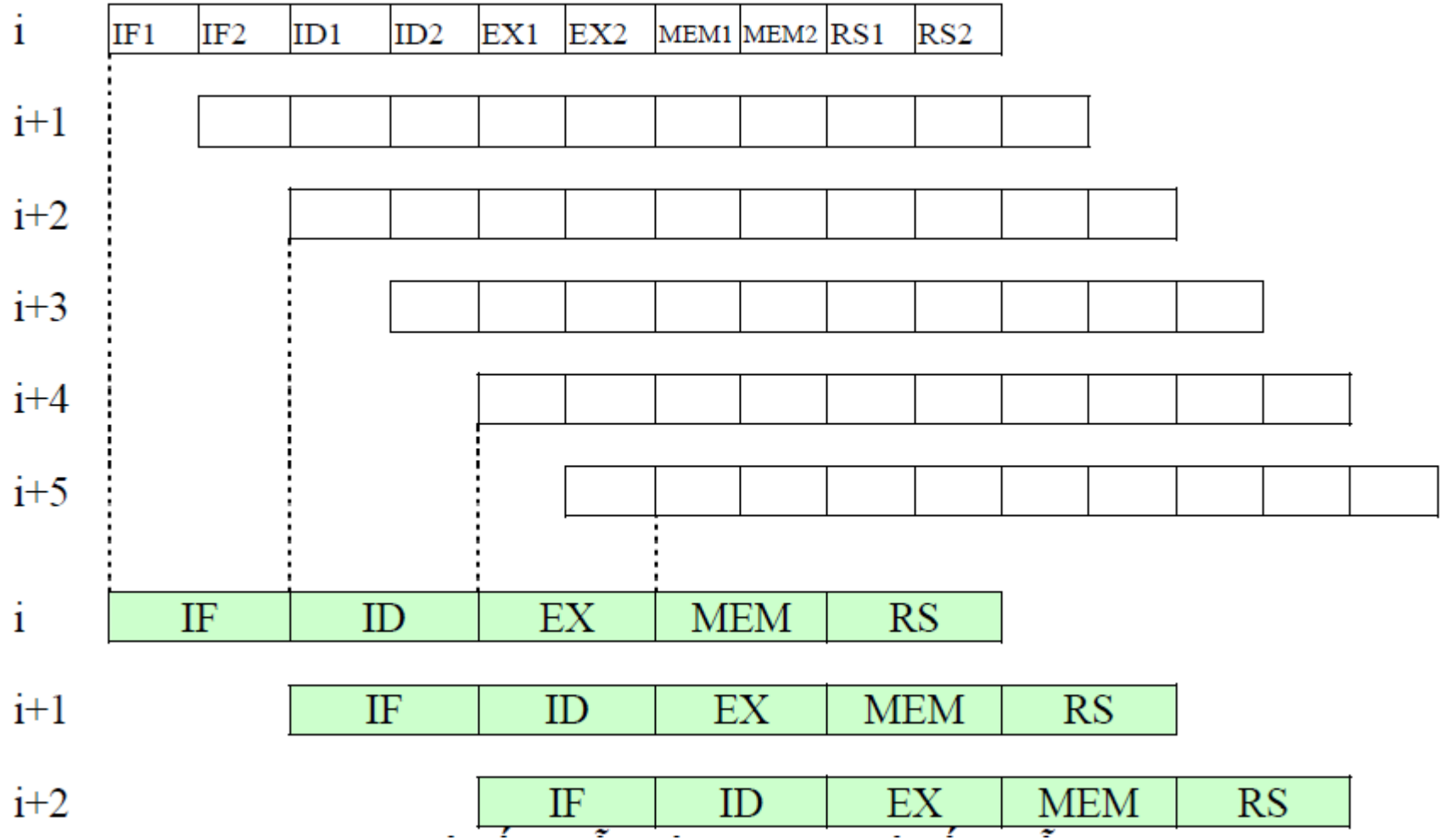
- Máy tính có kỹ thuật siêu ống dẫn bậc n bằng cách chia các giai đoạn của kỹ thuật ống dẫn đơn giản, mỗi giai đoạn được thực hiện trong khoản thời gian Tc, thành n giai đoạn con thực hiện trong khoản thời gian Tc/n. Độ hữu hiệu của kỹ thuật này tương đương với việc thi hành n lệnh trong mỗi chu kỳ Tc.
- Trong một chu kỳ Tc, máy dùng kỹ thuật siêu ống dẫn làm n lệnh thay vì làm1 lệnh trong máy dùng kỹ thuật ống dẫn bình thường. Trong máy tính siêu ống dẫn, tốc độ thực hiện lệnh tương đương với việc thực hiện một lệnh trong khoảng thời gian $\frac{T_C}{n}$

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Siêu ống dẫn (Hyper Pipielining)

Siêu ống dẫn bậc 2 so với siêu ống dẫn đơn giản.

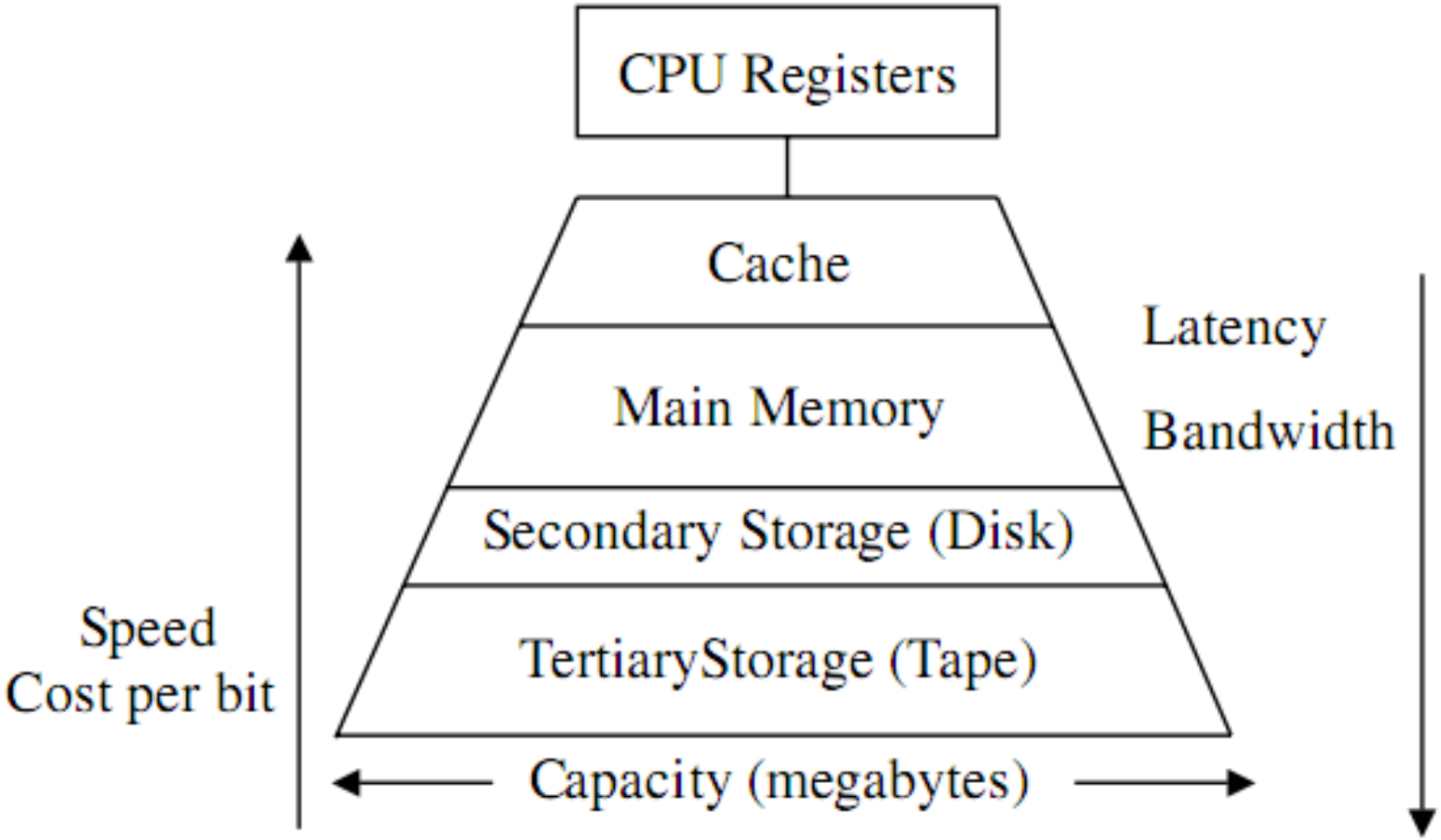
Trong khoảng thời gian Tc, máy có siêu ống dẫn làm 2 lệnh thay vì 1 lệnh như trong máy có kỹ thuật ống dẫn đơn giản.



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Bộ nhớ máy tính

3.4.1 Mô hình phân cấp của bộ nhớ máy tính



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- *Các tham số cơ bản của hệ thống nhớ*

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Các thành phần cơ bản của hệ thống nhớ*

CPU registers (các thanh ghi của CPU):

- Dung lượng rất nhỏ, khoảng từ vài chục bytes đến vài KB
- Tốc độ truy nhập rất cao (các thanh ghi hoạt động với tốc độ của CPU); thời gian truy nhập khoảng 0,25ns
- Giá thành đắt
- Sử dụng để lưu toán hạng đầu vào và kết quả của các lệnh.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Các thành phần cơ bản của hệ thống nhớ*

Cache (bộ nhớ cache): là bộ nhớ đệm chứa dữ liệu, các dữ liệu được nằm chờ yêu cầu từ ứng dụng hoặc phần cứng. Dữ liệu được chứa trong cache có thể là các thuật toán đã được thực hiện khi được yêu cầu, hoặc các dữ liệu trùng được lưu trữ ở một nơi khác.

- Dung lượng tương đối nhỏ (khoảng 64KB đến 32MB)
- Tốc độ truy nhập cao; thời gian truy nhập khoảng 1-5ns
- Giá thành đắt
- Còn được gọi là “bộ nhớ thông minh” (smart memory)
- Sử dụng để lưu lệnh và dữ liệu cho CPU xử lý.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Các thành phần cơ bản của hệ thống nhớ*

Main memory (bộ nhớ chính):

- Gồm ROM và RAM, có kích thước khá lớn; với hệ thống 32 bít, dung lượng khoảng 256MB-4GB
- Tốc độ truy nhập chậm; thời gian truy nhập khoảng 50-70ns
- Giá thành tương đối rẻ
- Sử dụng để lưu lệnh và dữ liệu của hệ thống và của người dùng.

Secondary memory (bộ nhớ thứ cấp – bộ nhớ ngoài):

- Có dung lượng rất lớn, khoảng từ 20GB-1000GB
- Tốc độ truy nhập rất chậm; thời gian truy nhập khoảng 5ms
- Giá thành rẻ
- Sử dụng để lưu dữ liệu lâu dài dưới dạng các tệp (files).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Vai trò của mô hình phân cấp bộ nhớ*

Tăng hiệu năng hệ thống

- Dung hoà được CPU có tốc độ cao và phần bộ nhớ chính và bộ nhớ ngoài có tốc độ thấp;
- Thời gian trung bình CPU truy nhập dữ liệu từ hệ thống nhớ tiệm cận thời gian truy nhập Cache.

Giảm giá thành sản xuất

- Các thành phần đắt tiền (thanh ghi và Cache) được sử dụng với dung lượng nhỏ;
- Các thành phần rẻ tiền hơn (bộ nhớ chính và bộ nhớ ngoài) được sử dụng với dung lượng lớn;

➔ Tổng giá thành của hệ thống nhớ theo mô hình phân cấp sẽ rẻ hơn so với hệ thống nhớ không phân cấp có cùng tốc độ

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.2 *Phân loại bộ nhớ*

Dựa trên kiểu truy nhập:

- Random Access Memory (RAM): bộ nhớ truy nhập ngẫu nhiên
- Serial Access Memory (SAM) : bộ nhớ truy nhập tuần tự
- Read Only Memory (ROM): bộ nhớ chỉ đọc

Dựa trên khả năng duy trì dữ liệu:

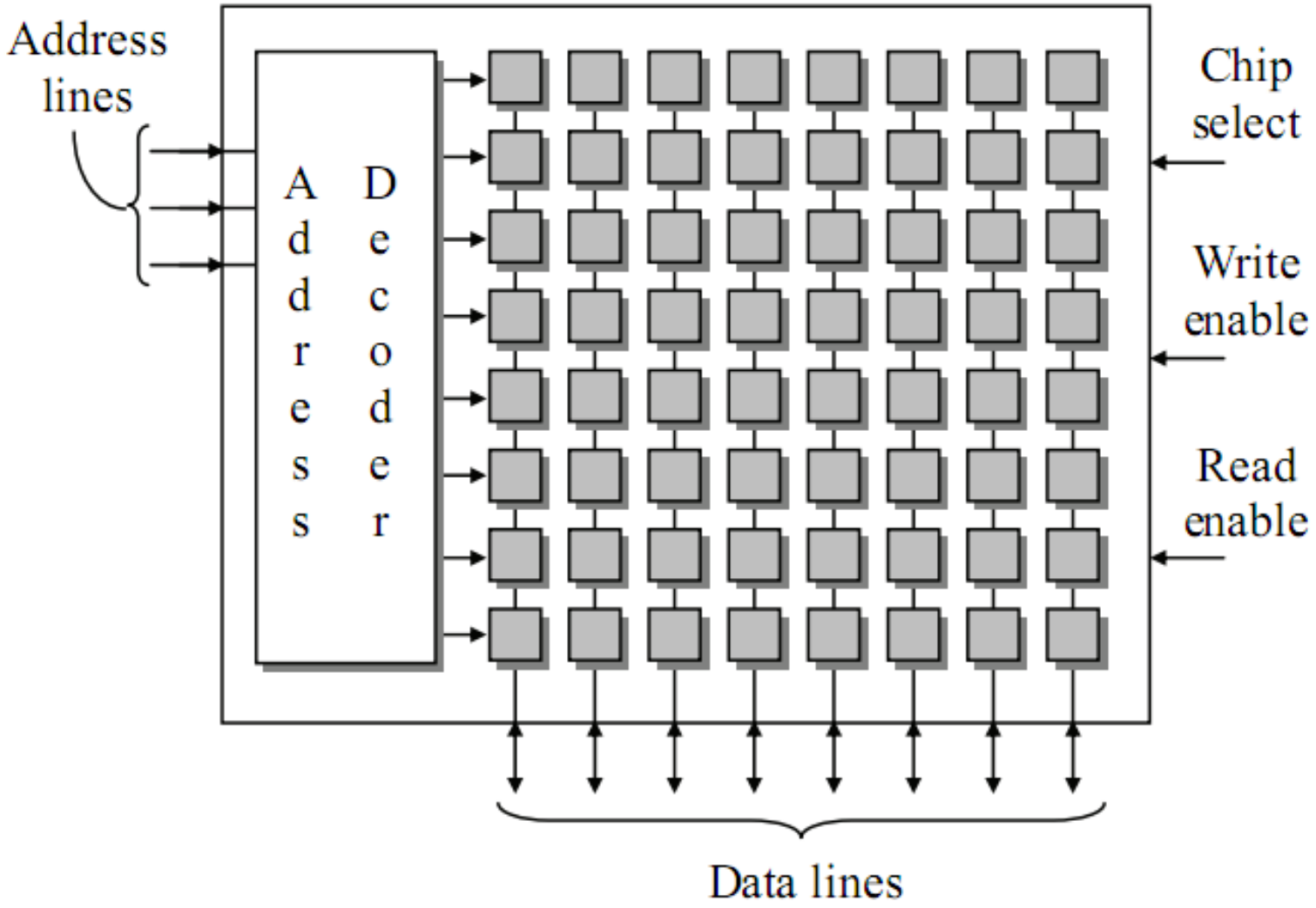
- Volatile memory: bộ nhớ không ổn định; thông tin mất khi mất nguồn nuôi: RAM.
- Non-volatile memory: bộ nhớ ổn định; thông tin vẫn được duy trì khi mất nguồn nuôi: ROM,HDD,...

Dựa trên công nghệ chế tạo:

- Bộ nhớ bán dẫn (Semiconductor memory): ROM, RAM
- Bộ nhớ từ tính (Magnetic memory): HDD, FDD, băng từ
- Bộ nhớ quang học (Optical memory): CD, DVD

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.3 *Tổ chức mạch nhớ*



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.3 *Tổ chức mạch nhớ*

Address lines:

- Các đường địa chỉ kết nối với bus A;
- Chuyển tín hiệu địa chỉ từ CPU đến mạch nhớ

Address decoder:

- Bộ giải mã địa chỉ
- Sử dụng tín hiệu địa chỉ để chọn ra và kích hoạt ô nhớ/dòng nhớ cần truy nhập.

Data lines:

- Các đường dữ liệu kết nối với bus D;
- Truyễn dữ liệu từ bộ nhớ về CPU và ngược lại.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.3 *Tổ chức mạch nhớ*

Chip select (CS):

- Chân tín hiệu chọn chip;
- Chip nhớ được kích hoạt khi CS = 0; Thông thường, CPU chỉ có thể làm việc với một chip nhớ tại một thời điểm.

Write enable (WE):

- Chân tín hiệu cho phép ghi;
- Cho phép ghi vào dòng nhớ khi WE = 0.

Read enable (RE):

- Chân tín hiệu cho phép đọc;
- Cho phép đọc dữ liệu từ dòng nhớ khi RE = 0.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.4 *Bộ nhớ ROM*

ROM là bộ nhớ chỉ đọc (Read Only Memory)

- Việc ghi thông tin vào ROM chỉ có thể được thực hiện bằng các thiết bị hoặc phương pháp đặc biệt;

ROM là bộ nhớ ổn định

- Thông tin trong ROM vẫn được duy trì khi mất nguồn nuôi

ROM là bộ nhớ bán dẫn: mỗi ô nhớ của ROM là một cổng bán dẫn

ROM thường được sử dụng để lưu các thông tin của hệ thống:

- Các thông tin về phần cứng hệ thống và BIOS (Basic Input Output System – hệ thống vào ra cơ sở).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.4 *Bộ nhớ ROM*

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Các loại bộ nhớ ROM*

ROM nguyên thuỷ (Ordinary ROM):

- ROM các thế hệ đầu tiên;
- Sử dụng tia cực tím để ghi thông tin vào ROM

PROM (Programmable ROM):

- ROM có thể lập trình được;
- Thông tin có thể được ghi vào PROM nhờ một thiết bị đặc biệt gọi là bộ lập trình PROM.

EPROM (Erasable programmable read-only memory):

- Là ROM có thể lập trình và xoá được;
- Thông tin trong EPROM có thể xoá được sử dụng tia cực tím có cường độ cao.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Các loại bộ nhớ ROM*

EEPROM: (Electrically Erasable PROM):

- Là PROM có thể xoá được thông tin bằng điện
- Có thể ghi được thông tin sử dụng phần mềm chuyên dụng.

Flash memory:

- Là một dạng EEPROM nhưng có tốc độ ghi và đọc thông tin nhanh hơn.
- Bộ nhớ flash chỉ có thể đọc/ghi theo khối.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.5 *Bộ nhớ RAM*

RAM (Random Access Memory) là bộ nhớ truy nhập ngẫu nhiên

- Mỗi ô nhớ của RAM có thể được truy nhập một cách ngẫu nhiên không theo trật tự nào;
- Tốc độ truy nhập các ô nhớ là tương đương.

RAM là bộ nhớ không ổn định:

- Tất cả thông tin trong RAM sẽ bị mất khi mất nguồn nuôi

RAM là bộ nhớ bán dẫn: mỗi ô nhớ của RAM là một cổng bán dẫn

RAM được sử dụng để lưu các thông tin của hệ thống và của người dùng:

- Thông tin của hệ thống: thông tin phần cứng và hệ điều hành
- Thông tin của người dùng: các chương trình ứng dụng và dữ liệu.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

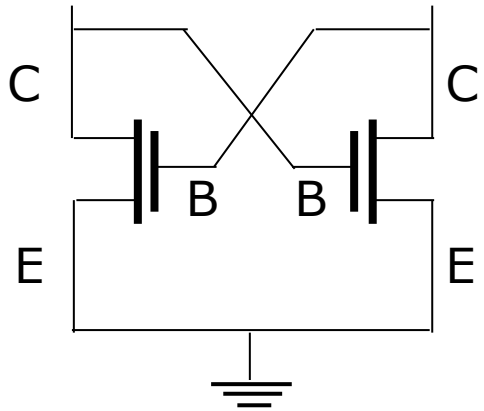
➢ *Các loại bộ nhớ RAM*

Hai loại RAM cơ bản:

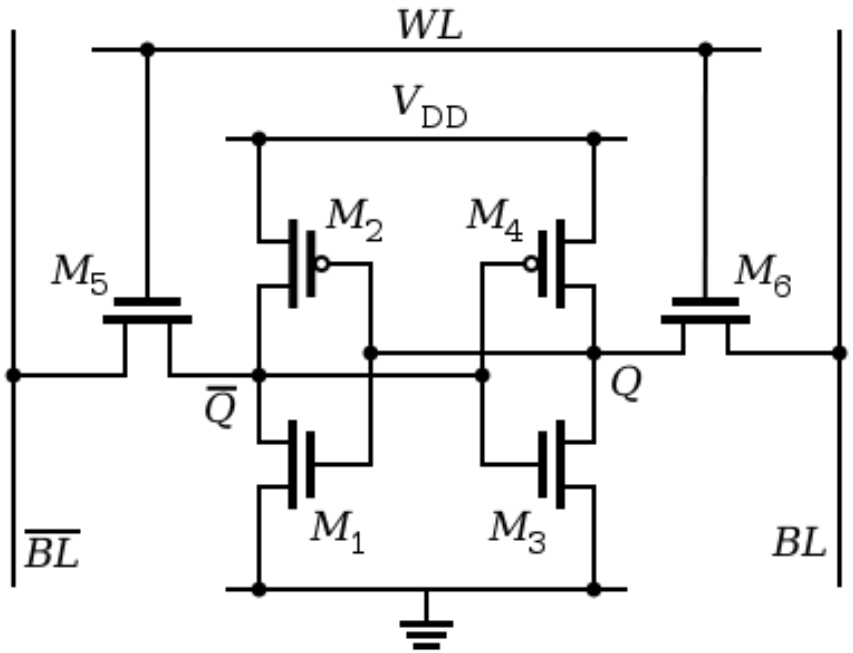
- RAM tĩnh (Static RAM – SRAM):
  - Mỗi bít SRAM dựa trên một mạch lật (flip-flop)
  - Thông tin lưu trong các bit SRAM luôn ổn định và không phải "làm tươi" định kỳ
  - SRAM nhanh hơn nhưng đắt hơn DRAM.
- RAM động (Dynamic RAM – DRAM):
  - Mỗi bít DRAM dựa trên một tụ điện
  - Thông tin lưu trong các bit DRAM không ổn định và phải được "làm tươi" định kỳ
  - DRAM chậm hơn nhưng rẻ hơn SRAM.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Cấu tạo SRAM***



Một mạch lật (flip-flop) đơn giản



Một ô nhớ SRAM loại 6T

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Đặc điểm của SRAM***

SRAM sử dụng một mạch lật trigger lưỡng ổn (*bistable latching circuit*) để lưu một bit thông tin;

Mỗi mạch lật lưu 1 bit thường sử dụng 6, 8 hoặc 10 transitors (gọi là mạch 6T, 8T hoặc 10T);

SRAM thường có tốc độ truy nhập nhanh do:

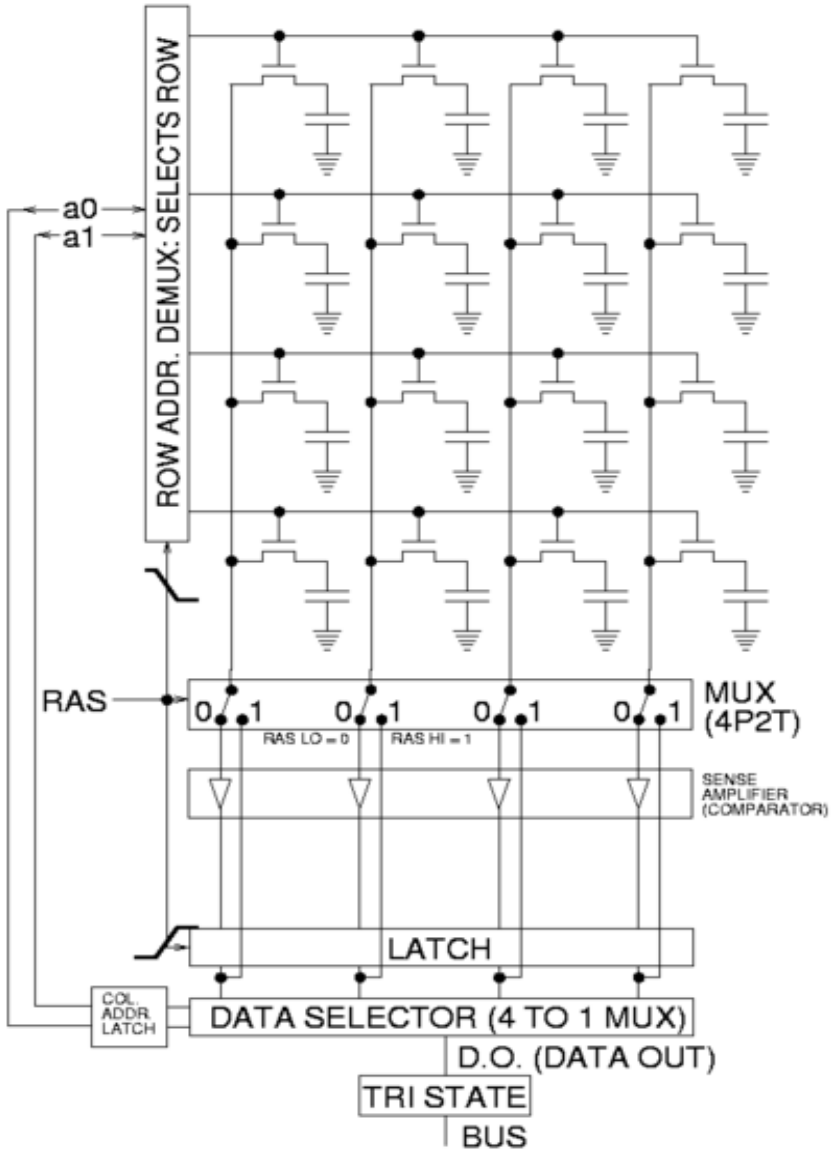
- Các bit của SRAM có cấu trúc đối xứng
- Các mạch nhớ SRAM chấp nhận tất cả các chân địa chỉ tại một thời điểm (không dồn kênh).

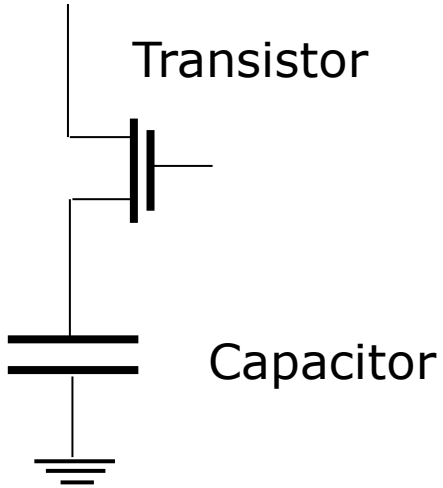
SRAM thường đắt hơn so với DRAM do:

- Mỗi bít SRAM dùng nhiều transitor hơn so với 1 bit DRAM
- Do cấu trúc bên trong của SRAM bit phức tạp hơn nên mật độ cấy linh kiện trong SRAM thường thấp.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- ***Cấu tạo DRAM***





Một bit DRAM

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ *Đặc điểm của DRAM*

Mỗi bit DRAM dựa trên một tụ điện và một transitor:

- Hai mức tích điện của tụ biểu diễn 2 mức logic 0 và 1:
  - Không tích điện: mức 0
  - Tích đầy điện: mức 1

Do tụ thường tự phóng điện, điện tích trong tụ có xu hướng bị tổn hao theo thời gian.

- Cần nạp lại thông tin trong tụ thường xuyên để tránh mất thông tin.
- Việc nạp lại thông tin cho tụ là quá trình làm tươi (refresh), phải theo định kỳ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Đặc điểm của DRAM***

Các bit nhớ của DRAM thường được sắp xếp thành ma trận:

- Một tụ + một transitor → một bit
- Các bít được tập hợp thành các dòng và cột

DRAM thường chậm hơn SRAM do:

- Cần quá trình làm tươi
- Việc nạp điện cho tụ cũng gây trễ
- Các mạch DRAM thường dùng kỹ thuật dồn kênh (địa chỉ cột/hàng) để tiết kiệm đường địa chỉ.

DRAM thường rẻ hơn SRAM do:

- Cấu trúc đơn giản, dùng ít transitor
- Mật độ cấy linh kiện cao hơn.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các loại DRAM***

SDRAM(Synchronous DRAM): DRAM đồng bộ (với nhịp đồng hồ của bus)

SRD SDRAM (Single Data Rate SDRAM): chấp nhận một thao tác đọc/ghi và chuyển 1 từ dữ liệu trong 1 chu kỳ đồng hồ; tốc độ 100MHz, 133MHz.

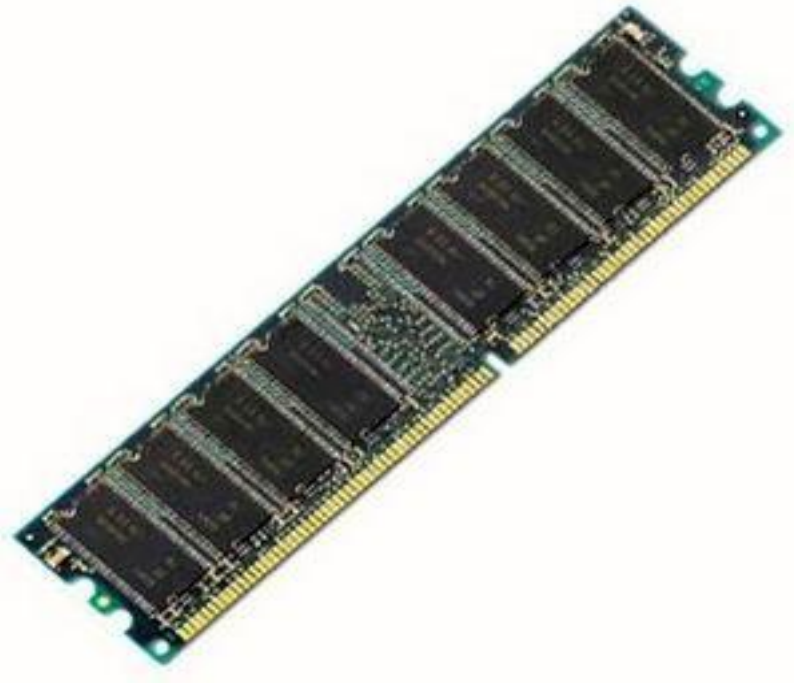
DDR SDRAM (Double Data Rate SDRAM)

- DDR1 SDRAM: DDR 266, 333, 400: có khả năng chuyển 2 từ dữ liệu trong 1 chu kỳ đồng hồ;
- DDR2 SDRAM: DDR2 400, 533, 800 : có khả năng chuyển 4 từ dữ liệu trong 1 chu kỳ đồng hồ;
- DDR3 SDRAM: DDR3 800, 1066, 1333, 1600 : có khả năng chuyển 8 từ dữ liệu trong 1 chu kỳ đồng hồ;

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- ***Các loại DRAM***

SDRAM PC133

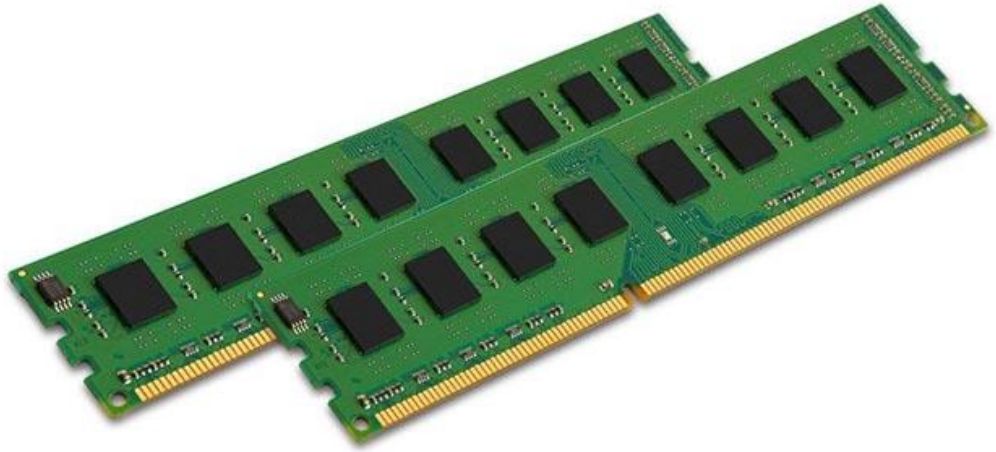


DDR3 1066 SDRAM

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các loại DRAM***

DDR2 SDRAM (Double Data Rate Two SDRAM - SDRAM tốc độ dữ liệu kép 2)



DDR3 SDRAM (Double Data Rate Three SDRAM - SDRAM tốc độ dữ liệu kép 3)

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các loại DRAM***

DDR SDRAM Standard	Internal rate (MHz)	Bus clock (MHz)	Prefetch	Data rate (MT/s)	Transfer rate (GB/s)	Voltage (V)
SDRAM	100-166	100-166	1n	100-166	0.8-1.3	3.3
DDR	133-200	133-200	2n	266-400	2.1-3.2	2.5/2.6
DDR2	133-200	266-400	4n	533-800	4.2-6.4	1.8
DDR3	133-200	533-800	8n	1066-1600	8.5-14.9	1.35/1.5
DDR4	133-200	1066-1600	8n	2133-3200	17-21.3	1.2

Bảng so sánh thông số kỹ thuật cơ bản của SDRAM, DDR, DDR2, DDR3, DDR4

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các loại DRAM***

•**DDR 1:** (tên đầy đủ của nó là DDR SDRAM, DDR là viết tắt của cụm từ Double Date Rate). Loại RAM DDR 1 này bây giờ rất hiếm, tuổi đời hơn 10 năm. Không còn phù hợp với cấu hình phần cứng hiện tại, quá yếu và không thể đáp ứng được nhu cầu sử dụng của người dùng.

•**DDR 2:** Thế hệ tiếp theo của RAM DDR 1 sử dụng cho các bảng mạch sử dụng Chipset Intel dòng 945 → G31. Loại chip này sử dụng công nghệ chân đế tiếp xúc Socket 775. Và cho tới thời điểm năm 2017 thì vẫn còn khá nhiều máy tính dùng loại này. Loại RAM này thường được sử dụng cho các CPU Intel Core Duo, Core 2 Duo…

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

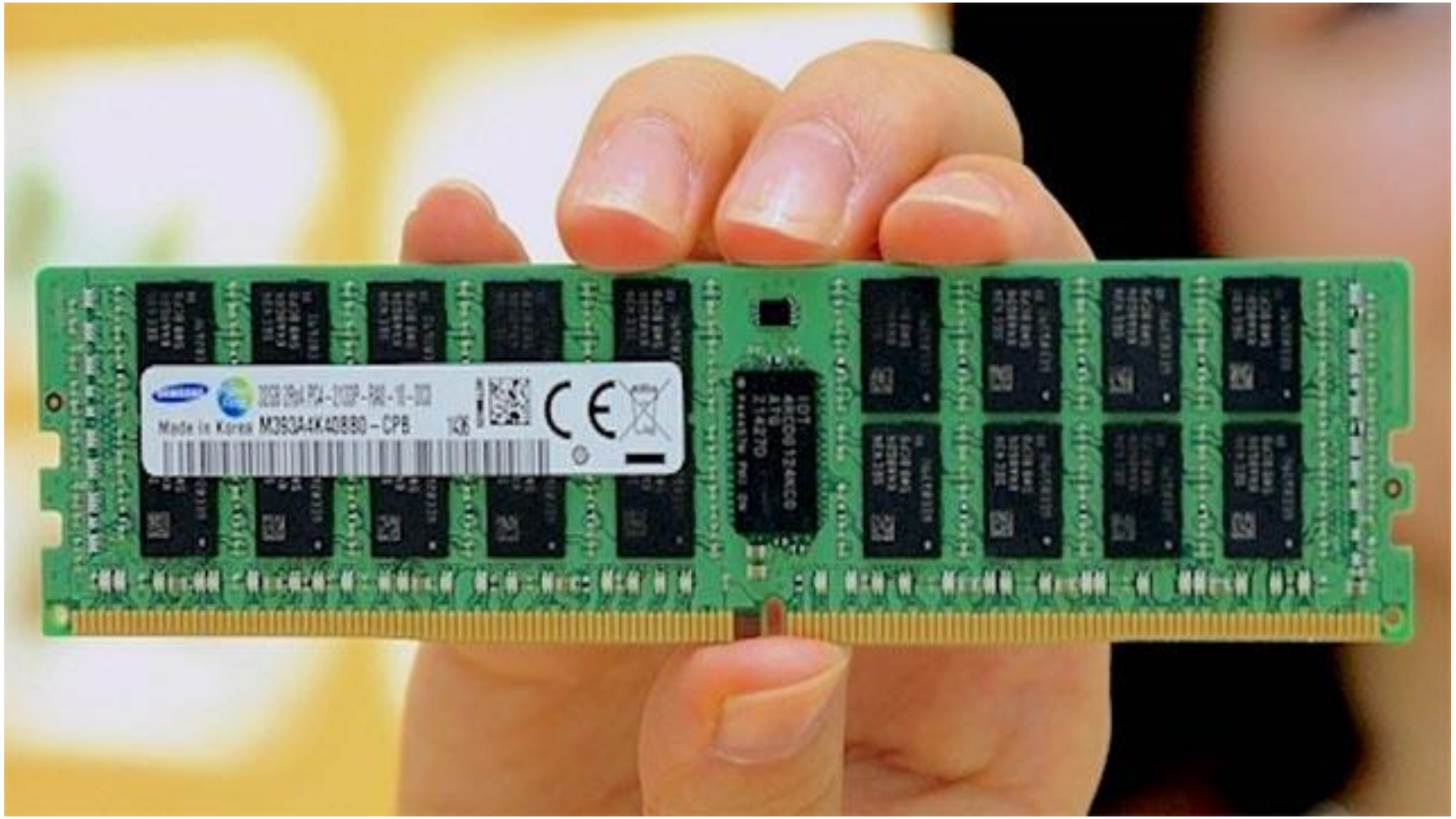
➢ ***Các loại DRAM***

•**DDR 3:** Có lẽ đây là loại RAM phổ biến nhất thị trường hiện nay, nó được sử dụng rộng rãi cho các thế hệ máy tính đời mới. Loại RAM này thường được sử dụng cùng với CPU Intel Core 2 Duo, Core I3/ I5 hoặc I7….

•**DDR 4:** Là loại RAM mạnh mẽ nhất hiện nay, nó chỉ tương thích với một số phần cứng đời mới hiện nay.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- ***Các loại DRAM***



DDR5

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

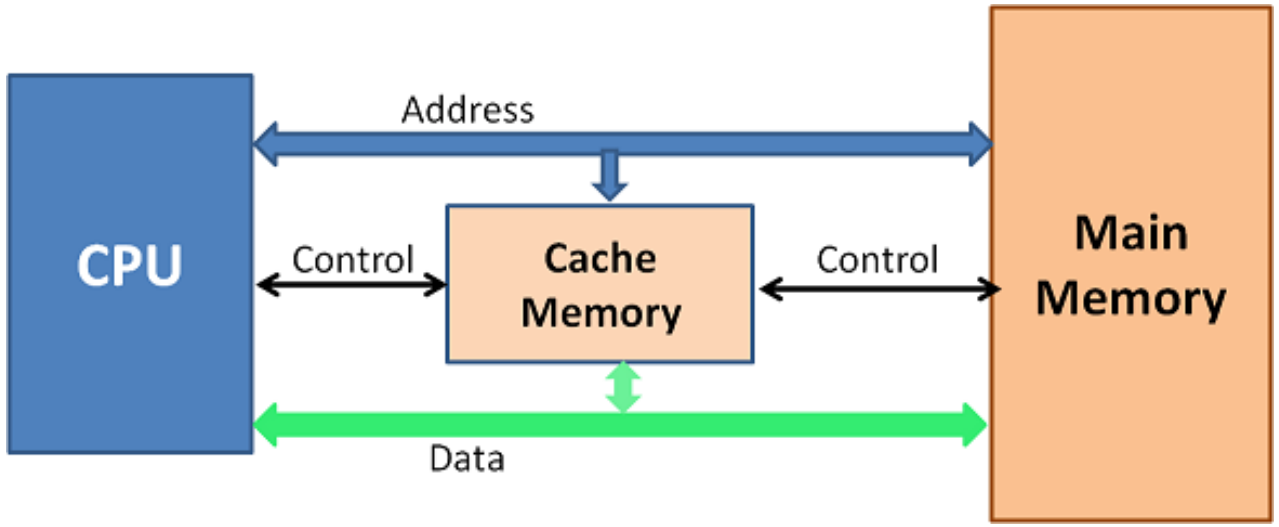
3.4.6 *Cache*

Cache là một thành phần trong hệ thống nhớ phân cấp của máy tính:

- Cache (bộ nhớ đệm tạm thời) đóng vai trong trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại;

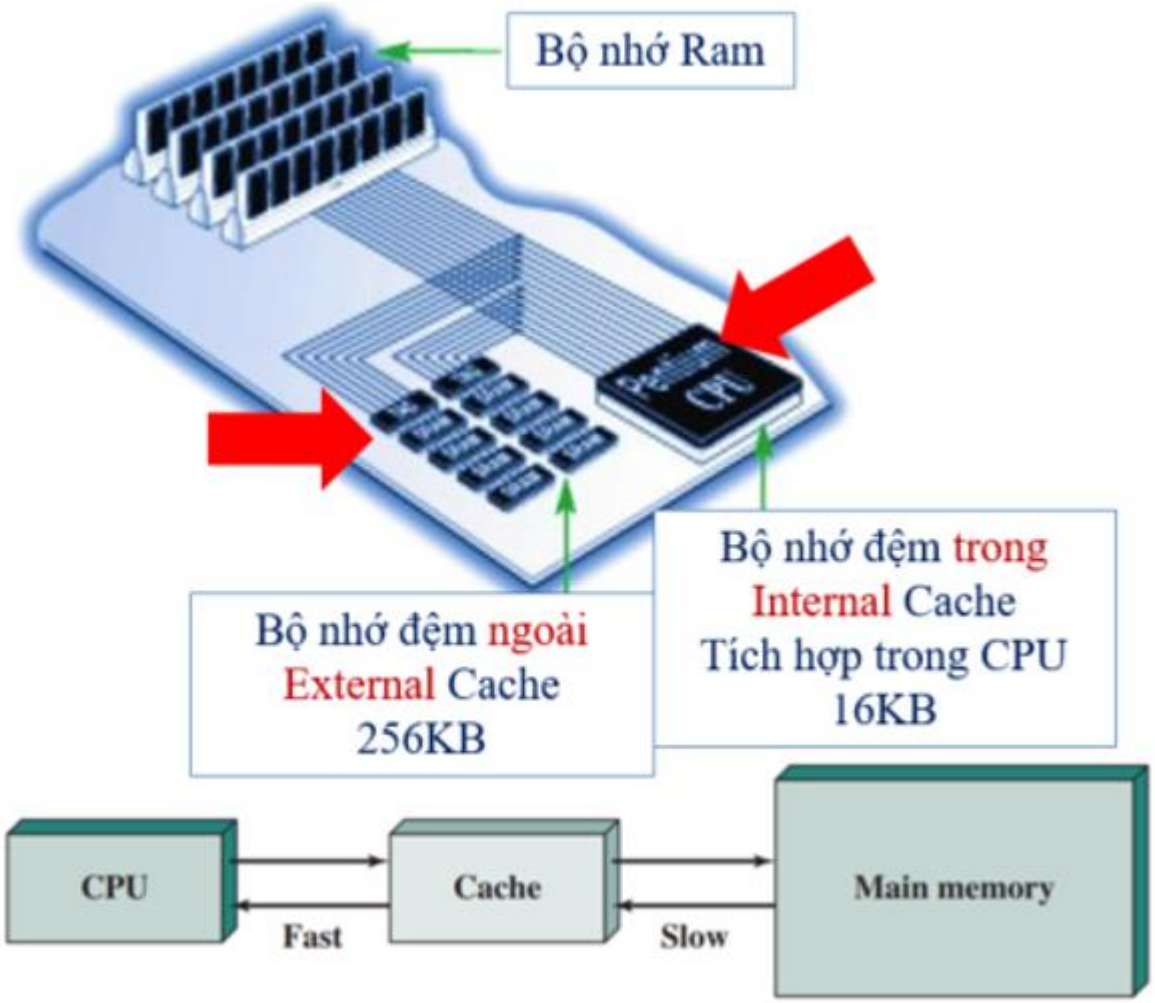
Vị trí của cache:

- Với các hệ thống CPU cũ, Cache thường nằm ngoài CPU
- Với các CPU mới, Cache thường được tích hợp vào trong CPU.



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.6 *Cache*



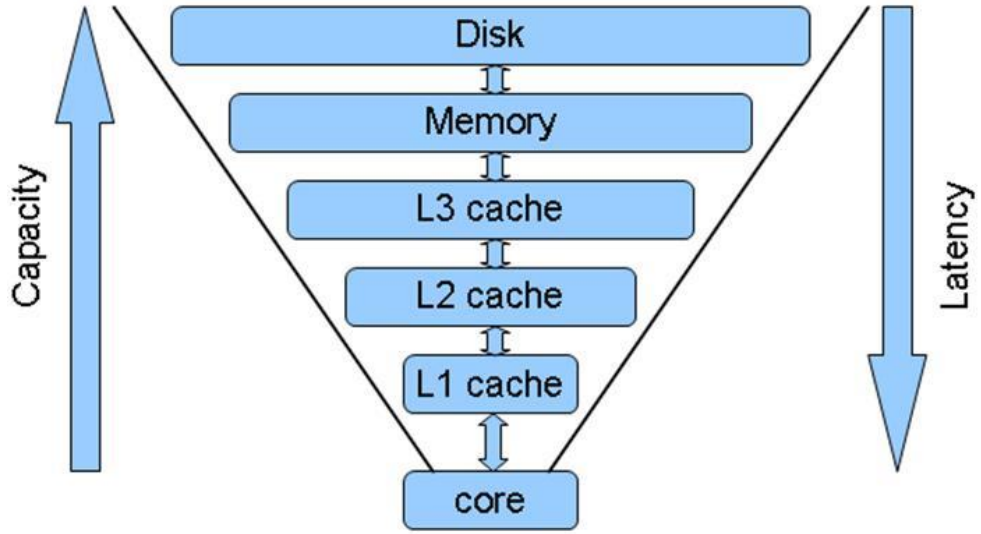
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

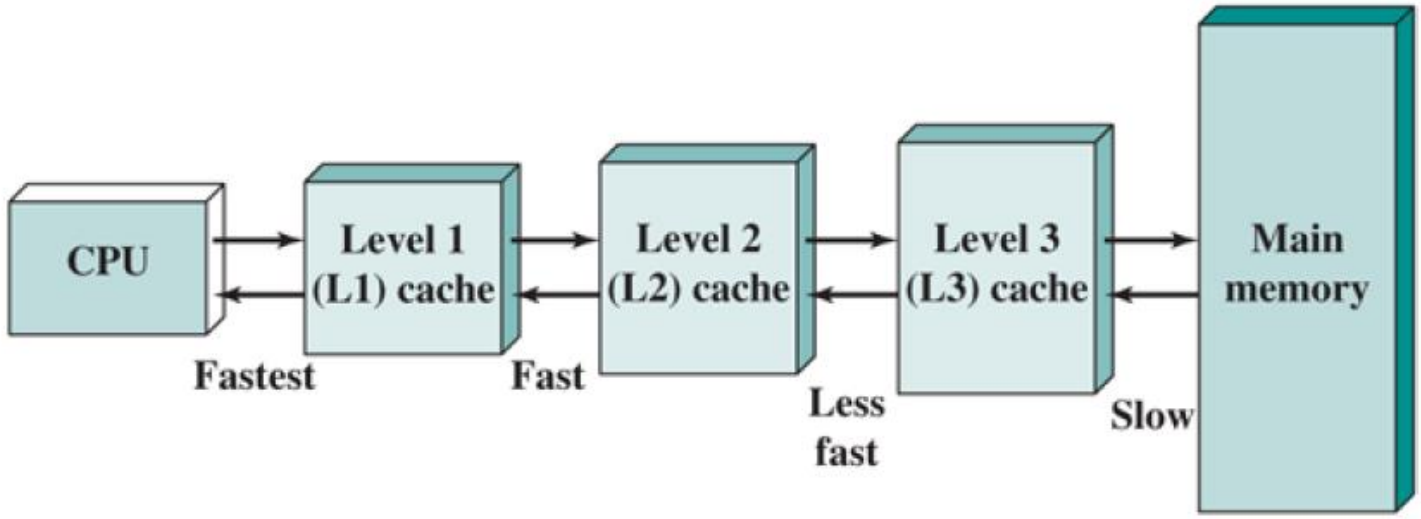
3.4.6 *Cache*

- Dung lượng của cache thường nhỏ:
  - Với các hệ thống cũ: 16K, 32K,..., 128K
  - Với các hệ thống mới: 256K, 512K, 1MB, 2MB, hoặc lớn hơn
- Cache có tốc độ truy nhập nhanh hơn nhiều so với bộ nhớ chính;
- Giá thành cache (tính theo bit) thường đắt hơn nhiều so với bộ nhớ chính.
- Với các hệ thống CPU mới, cache thường được chia thành nhiều mức (levels):
  - Mức 1: 16-32KB có tốc độ rất cao
  - Mức 2: 1-16MB có tốc độ khá cao
  - Mức 3: Tốc độ chậm hơn nhưng dung lượng lớn hơn Cache L1 L2

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.6 *Cache*





Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.6 *Cache*

➢ *Vai trò của Cache*

Tăng hiệu năng hệ thống

- Dung hoà được CPU có tốc độ cao và bộ nhớ chính có tốc độ thấp;
- Thời gian trung bình CPU truy nhập dữ liệu từ hệ thống nhớ tiệm cận thời gian truy nhập cache.

Giảm giá thành sản xuất

- Nếu hai hệ thống nhớ có cùng giá thành, hệ thống nhớ có Cache có tốc độ truy nhập nhanh hơn;
- Nếu hai hệ thống nhớ có cùng tốc độ, hệ thống nhớ có Cache có giá thành rẻ hơn.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4.6 *Cache*

➢ *Các nguyên lý hoạt động của Cache*

Cache được coi là bộ nhớ thông minh:

- Cache có khả năng đoán trước yêu cầu về dữ liệu và lệnh của CPU;
- Dữ liệu và lệnh cần thiết được chuyển trước từ bộ nhớ chính về cache → CPU chỉ truy nhập cache → giảm thời gian truy nhập hệ thống nhớ.

Cache hoạt động dựa trên 2 nguyên lý cơ bản:

- Nguyên lý lân cận về **không gian** (**Spatial locality**)
- Nguyên lý lân cận về **thời gian** (**Temporal locality**)

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các nguyên lý hoạt động của Cache***

Nguyên lý lân cận về không gian:

- Nếu một ô nhớ đang được truy nhập thì xác xuất các ô nhớ liền kề với nó được truy nhập trong tương lai gần là rất cao;

Áp dụng:

- Lân cận về không gian được áp dụng cho nhóm lệnh/dữ liệu có tính tuần tự cao trong không gian chương trình;

  Giải thích:

  - Do các lệnh trong một chương trình thường tuần tự → cache đọc cả khối lệnh từ bộ nhớ chính → phủ được các ô nhớ lân cận của ô nhớ đang được truy nhập.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các nguyên lý hoạt động của Cache***

Nguyên lý lân cận về không gian:

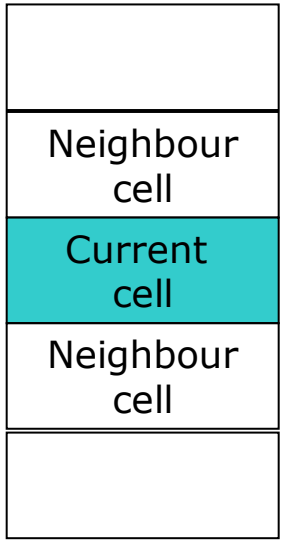
- Nếu một ô nhớ đang được truy nhập thì xác xuất các ô nhớ liền kề với nó được truy nhập trong tương lai gần là rất cao;

  Áp dụng:

- Lân cận về không gian được áp dụng cho nhóm lệnh/dữ liệu có tính tuần tự cao trong không gian chương trình;

  Giải thích:

  - Do các lệnh trong một chương trình thường tuần tự → cache đọc cả khối lệnh từ bộ nhớ chính → phủ được các ô nhớ lân cận của ô nhớ đang được truy nhập.



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các nguyên lý hoạt động của Cache***

Nguyên lý lân cận về thời gian:

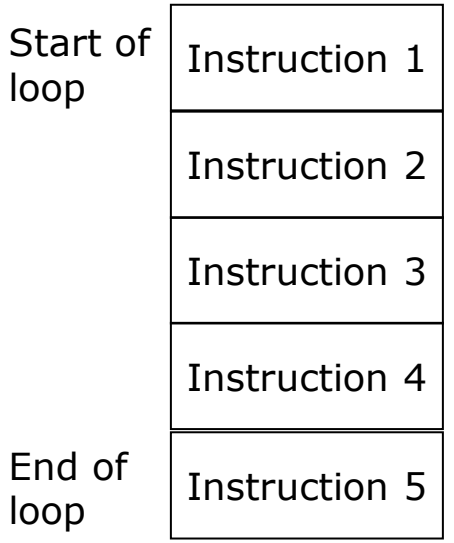
- Nếu một ô nhớ đang được truy nhập thì xác xuất nó được truy nhập lại trong tương lai gần là rất cao;

Áp dụng:

- Lân cận về thời gian được áp dụng cho dữ liệu và nhóm lệnh trong vòng lặp;

Giải thích:

- Các phần tử dữ liệu thường được cập nhật, sửa đổi thường xuyên;
- Cache đọc cả khối lệnh từ bộ nhớ chính → phủ được cả khối lệnh của vòng lặp.



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

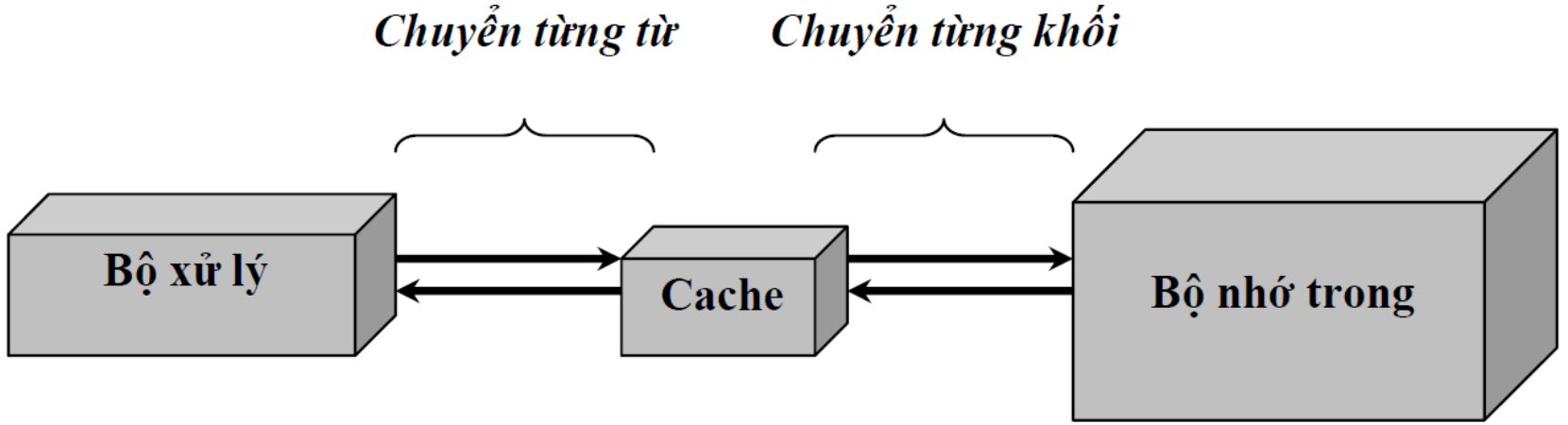
➢ ***Các nguyên lý hoạt động của Cache***

CPU đọc/ghi các phần tử dữ liệu đơn lẻ (word) với cache

- Tại sao?

Cache đọc/ghi các khối dữ liệu lớn (block) với bộ nhớ chính

- Tại sao?



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các nguyên lý hoạt động của Cache***

Hệ số Cache hit và Cache miss

Cache Hit (đoán trúng) là một sự kiện mà CPU truy nhập một mục tin có ở trong Cache:

- Xác suất để có một hit gọi là hệ số hit, hoặc ***H***.
- **0 <= H <= 1**
- Hệ số hit càng cao thì hiệu quả của Cache càng cao.

Cache Miss (đoán trượt) là một sự kiện mà CPU truy nhập một mục tin không có ở trong Cache:

- Xác suất của một miss gọi là hệ số miss, hoặc ***(1-H)***
- **0 <= (1 – H) <= 1**
- Hệ số miss thấp thì hiệu quả của Cache càng cao.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- ***Các nguyên lý hoạt động của Cache***

Cache penalty: Thời gian cần thiết để xử lý một Cache Miss.

- Thời gian bao gồm thời gian thâm nhập bộ nhớ trong cộng với thời gian chuyển khối dữ liệu chứa từ cần đọc từ bộ nhớ trong đến Cache.
- Thời gian này tuỳ thuộc vào kích thước của khối.

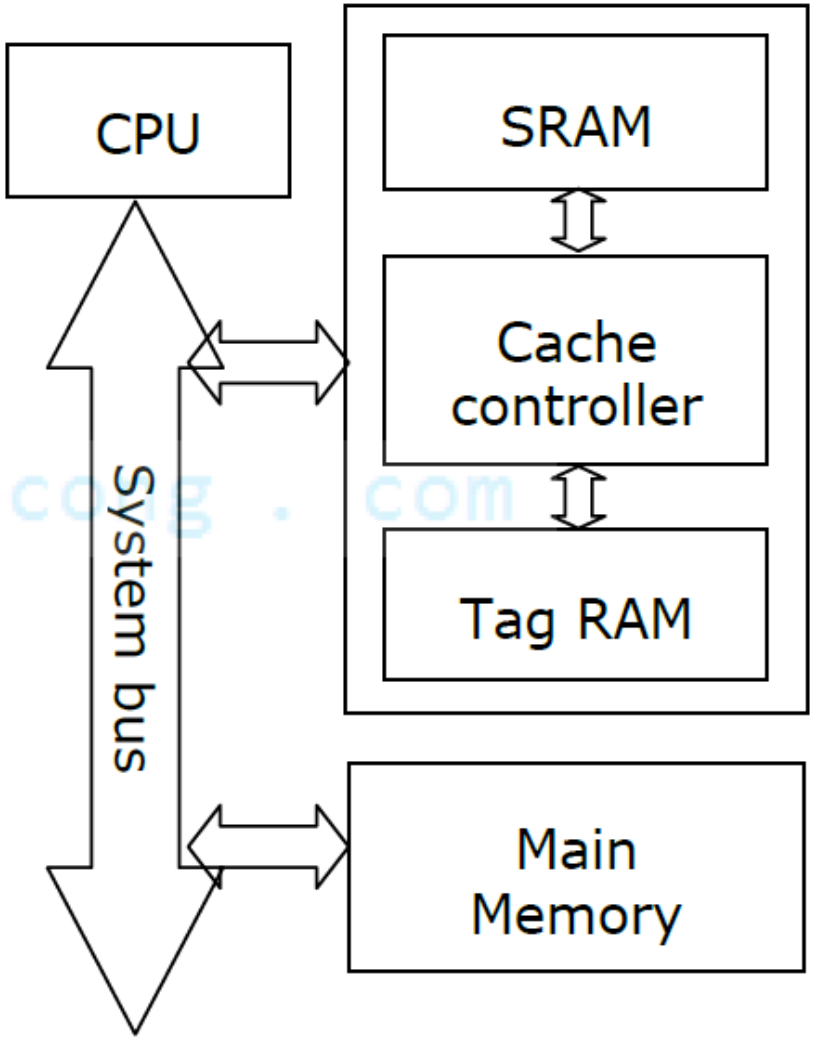
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Các dạng kiến trúc của Cache***

- Kiến trúc Cache đề cập đến việc Cache được bố trí vào vị trí nào trong quan hệ với CPU và bộ nhớ chính.
- Có hai loại kiến trúc Cache chính: kiến trúc Look Aside (cache được đặt ngang hàng với bộ nhớ chính) và kiến trúc Look Through (cache được đặt giữa CPU và bộ nhớ chính).
- Mỗi kiến trúc Cache kể trên có ưu điểm và nhược điểm riêng.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- ***Kiến trúc Look Aside***



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Kiến trúc Look Aside***

- Trong kiến trúc Look Aside, Cache và bộ nhớ chính cùng được kết nối vào bus hệ thống. Như vậy, cả Cache và bộ nhớ chính đều "thấy" chu kỳ bus của CPU tại cùng một thời điểm.
- Kiến trúc Look Aside có thiết kế đơn giản, dễ thực hiện
- Hệ số hit của kiến trúc này thường chậm do cache kết nối với CPU sử dụng bus hệ thống – thường có tần số làm việc không cao và băng thông hẹp.
- Các sự kiện miss của kiến trúc Look Aside thường nhanh hơn do khi CPU không tìm thấy mục tin trong Cache, nó đồng thời tìm mục tin trong bộ nhớ chính tại cùng một chu kỳ xung nhịp.

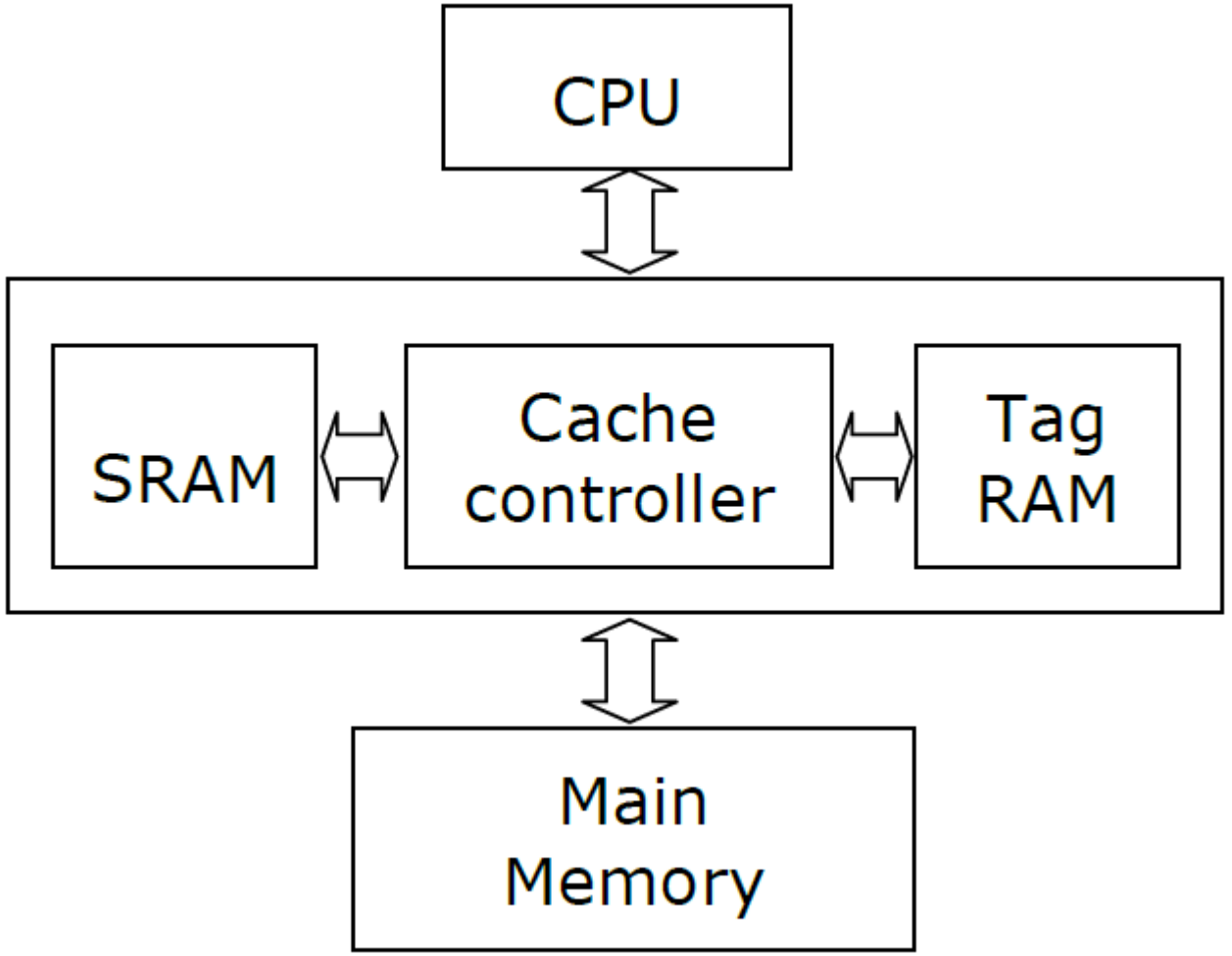
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ ***Kiến trúc Look Through***

- Trong kiến trúc kiểu Look Through, cache được đặt nằm giữa CPU và bộ nhớ chính. Cache có thể "thấy" chu kỳ bus của CPU trước, rồi nó mới "truyền" lại cho bộ nhớ chính.
- Cache kết nối với CPU bằng hệ thống bus riêng tốc độ cao và băng thông lớn, thường là bus mặt sau (BSB – Back Side Bus).
- Cache kết nối với bộ nhớ chính thông qua bus hệ thống hay bus mặt trước (FSB – Front Side Bus). FSB thường có tần số làm việc và băng thông thấp hơn nhiều so với BSB.
- Kiến trúc Look Through phức tạp hơn kiến trúc Look Aside.
    - Ưu điểm chính của kiến trúc này là các sự kiện hit của kiến trúc này thường rất nhanh do CPU kết nối với Cache bằng kênh riêng có tốc độ cao.
        - Các sự kiện miss của kiến trúc Look Through thường chậm hơn do khi CPU không tìm thấy mục tin trong Cache, nó cần tìm mục tin đó trong bộ nhớ chính tại một chu kỳ xung nhịp tiếp theo.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- ***Kiến trúc Look Through***



SRAM: RAM lưu dữ liệu cache

Tag RAM: RAM lưu địa chỉ bộ nhớ

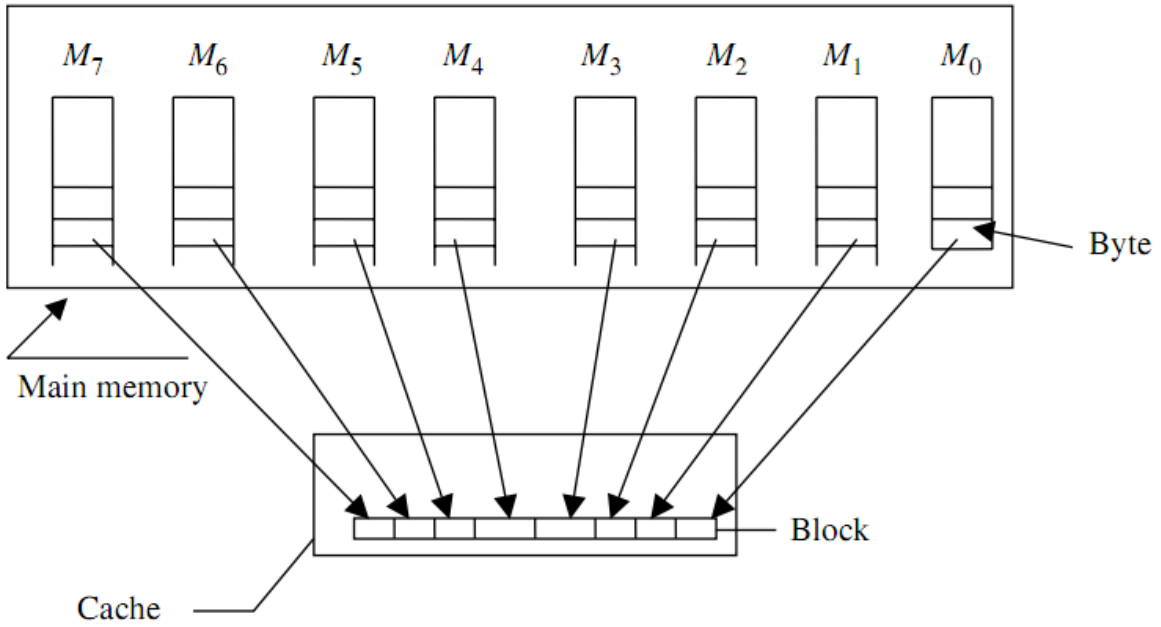
Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Tổ chức Cache

Tổ chức Cache giải quyết vấn đề Cache và bộ nhớ chính phối hợp làm việc như thế nào?

Kích thước của cache thường rất nhỏ so với kích thước bộ nhớ chính → tại mỗi thời điểm, chỉ có một phần nhỏ thông tin của bộ nhớ chính được chuyển vào Cache → phải xây dựng mô hình tổ chức / ánh xạ trao đổi dữ liệu giữa các phần tử nhớ bộ nhớ chính và các phần tử nhớ của Cache như thế nào để hệ thống nhớ đạt được tốc độ truy cập tối ưu ?

Quan hệ giữa các khối của bộ nhớ chính và dòng của cache



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Các phương pháp tổ chức Cache**

Ánh xạ trực tiếp (Direct Mapped)

- Đơn giản và nhanh
- Ánh xạ cứng dễ gây xung đột → hiệu quả Cache không cao

Ánh xạ kết hợp đầy đủ (Fully Associative)

- Phức tạp và chậm
- Ánh xạ mềm, ít xung đột

Ánh xạ tập kết hợp (Set Associative) → *kết hợp của 2 phương pháp trên*

- Phức tạp và nhanh
- Ánh xạ mềm, ít xung đột

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Các phương pháp tổ chức Cache

Ánh xạ trực tiếp (Direct mapping)

Cache:

- Chia thành *n* khối hoặc dòng, từ $Line_0$ đến $Line_{n-1}$

Bộ nhớ chính:

- Chia thành *m* trang, từ $page_0$ đến $page_{m-1}$.
- Một trang bộ nhớ có kích thước bằng kích thước cache
- Mỗi trang có *n* dòng, từ $Line_0$ đến $Line_{n-1}$

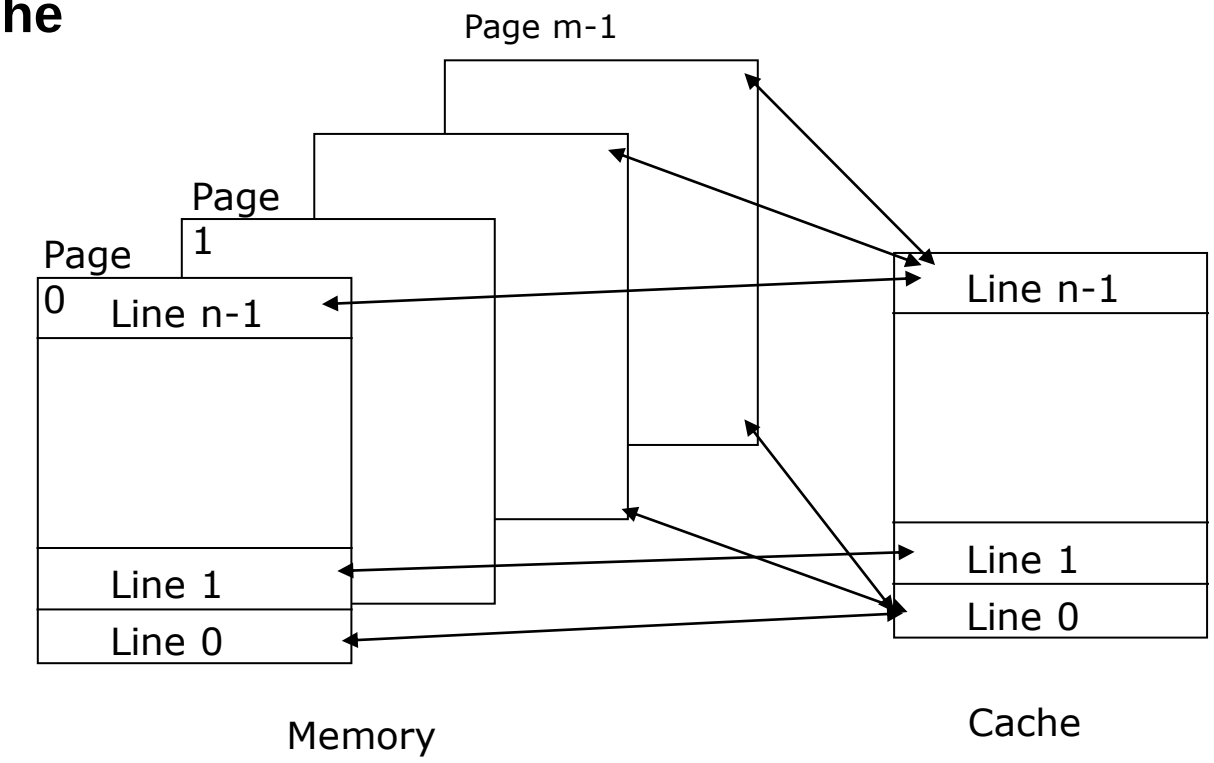
Ánh xạ:

- $Line_0$ của ($page_0$ đến $page_{m-1}$) ánh xạ đến $Line_0$ của cache;
- $Line_1$ của ($page_0$ đến $page_{m-1}$) ánh xạ đến $Line_1$ của cache;
- ....
- $Line_{n-1}$ của ($page_0$ đến $page_{m-1}$) ánh xạ đến $Line_{n-1}$ của cache;

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- **Các phương pháp tổ chức Cache**

Ánh xạ trực tiếp (Direct mapping)



- Tại mọi thời điểm luôn có ***m*** dòng bộ nhớ cùng cạnh tranh một dòng cache.

- Khi biết địa chỉ của dòng trong bộ nhớ → biết vị trí của nó trong cache

→ phương pháp ánh xạ trực tiếp còn được gọi là ánh xạ cứng hay ánh xạ cố định

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Các phương pháp tổ chức Cache

Ánh xạ trực tiếp (Direct mapping)

Cache sử dụng địa chỉ ánh xạ trực tiếp gồm 3 thành phần: *Tag*, *Line* và *Word* .

- *Tag* (tính bằng bit) là địa chỉ trang trong bộ nhớ chứa dòng được nạp vào cache.
- *Line* (bit) là địa chỉ dòng trong cache
- *Word* (bit) là địa chỉ của từ trong dòng.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- **Các phương pháp tổ chức Cache**

Ánh xạ trực tiếp (Direct mapping)

Ví dụ:

Vào:

Dung lượng bộ nhớ = 4GB (hệ thống 32 bit)

Dung lượng cache = 1MB

Kích thước dòng = 32 byte

Ra:

Kích thước dòng Line = 32 byte = 2^5 → Word = 5 bit

Dung lượng Cache = 1MB = 2^{20} → có 2^{20} / 2^5 = 2^{15} dòng → Line = 15 bit

Đ/c trang Tag: 4GB = 2^{32} ➔ cần 32 bít địa chỉ tổng cộng để địa chỉ hoá các ô nhớ:

Tag = 32bit địa chỉ – Line – Word = 32 – 15 – 5 = 12 bit.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Các phương pháp tổ chức Cache

Ánh xạ trực tiếp (Direct mapping)

Ưu điểm:

- Thiết kế *đơn giản*
- *Nhanh*: do các ánh xạ là cố định nên không tốn nhiều thời gian tìm địa chỉ ô nhớ trong cache.

Nhược điểm:

- Do ánh xạ cố định nên *dễ gây xung đột* vì có thể tạo ra nhiều dòng cache bị nút cổ chai trong quá trình hoạt động của cache. Có thể có nhiều dòng cache rảnh rỗi hay ít được sử dụng, nhưng cũng có nhiều dòng cache quá tải do bị nhiều dòng bộ nhớ cùng cạnh tranh.
- Cũng vì dễ gây xung đột nên *hiệu quả tận dụng không gian cache không cao* và *hệ số hit thấp*

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Các phương pháp tổ chức Cache

Ánh xạ kết hợp đầy đủ (Fully Associative)

Cache:

- Được chia thành *n* khối hoặc dòng, từ $Line_0$ đến $Line_{n-1}$

Bộ nhớ:

- Được chia thành *m* khối hoặc dòng, từ $Line_0$ đến $Line_{m-1}$.
- Kích thước mỗi dòng cache bằng kích thước một dòng bộ nhớ
- Số dòng trong bộ nhớ rất lớn so với số dòng của cache (m >> n).

Ánh xạ:

- Một dòng trong bộ nhớ có thể được ánh xạ vào một dòng bất kỳ trong cache;
- $Line_i$ trong bộ nhớ có thể được ánh xạ vào $Line_j$ của cache;

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Các phương pháp tổ chức Cache

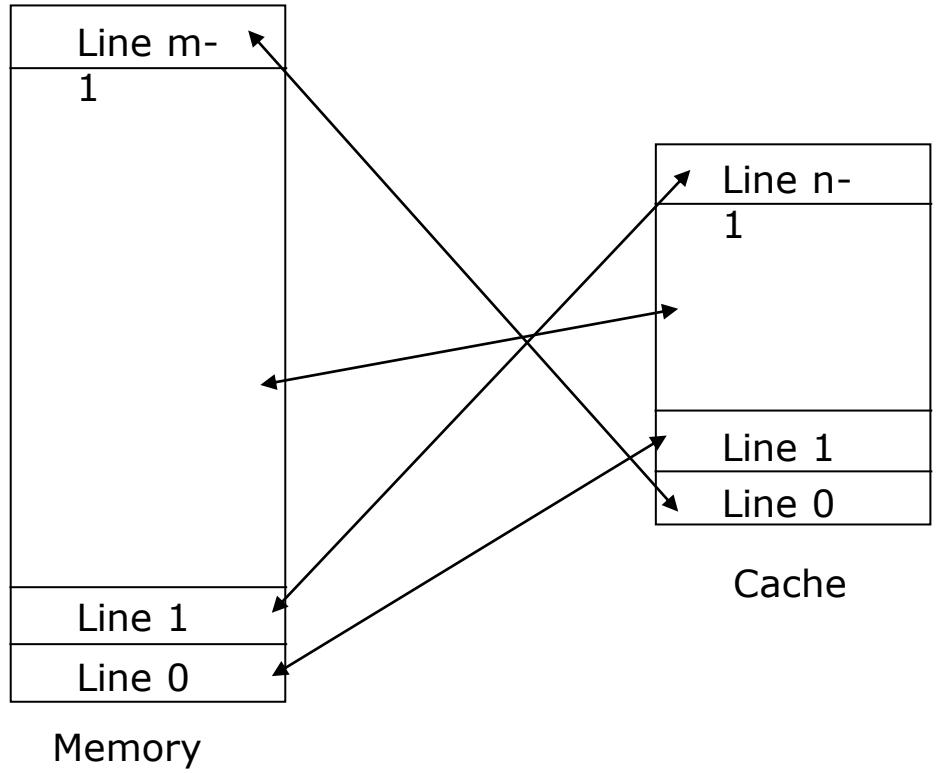
Ánh xạ kết hợp đầy đủ (Fully Associative)

Để có thể quản lý các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ kết hợp đầy đủ chỉ gồm 2 thành phần: *Tag* và *Word*

Tag (bit) là địa chỉ dòng trong bộ nhớ được nạp vào cache.

Word (bit) là địa chỉ của từ trong dòng.

Phần địa chỉ *Line* như trong địa chỉ ánh xạ trực tiếp bị bỏ do bộ nhớ chính chỉ còn là một trang duy nhất với m dòng



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ **Các phương pháp tổ chức Cache**

Ánh xạ kết hợp đầy đủ (Fully Associative)

Ví dụ:

- Vào:
  - Dung lượng bộ nhớ = 4GB (hệ thống 32 bit)
  - Dung lượng cache = 1MB
  - Kích thước dòng = 32 byte
    - Ra:
      - Kích thước dòng =32 byte = 2^5 → Word = 5 bit
      - Tag: 4GB = 2^{32} → cần 32 bít địa chỉ tổng cộng để địa chỉ hoá các ô nhớ:

Tag = 32bit địa chỉ – Word = 32 – 5 = 27 bit.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

➢ Các phương pháp tổ chức Cache

Ánh xạ kết hợp đầy đủ (Fully Associative)

Ưu điểm:

- Giảm được xung đột do ánh xạ là không cố định
- Hệ số Hit cao hơn ánh xạ trực tiếp.

Nhược điểm:

- Chậm do cần phải tìm địa chỉ ô nhớ trong cache
- Phức tạp do cần có n bộ so sánh địa chỉ bộ nhớ trong cache.

Thường được sử dụng với cache có dung lượng nhỏ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Ánh xạ tập kết hợp (Set Associative)

Cache:

- Được chia thành k đường (way) với kích thước bằng nhau;
- Mỗi đường được chia thành *n* dòng, từ $Line_0$ đến $Line_{n-1}$

Bộ nhớ:

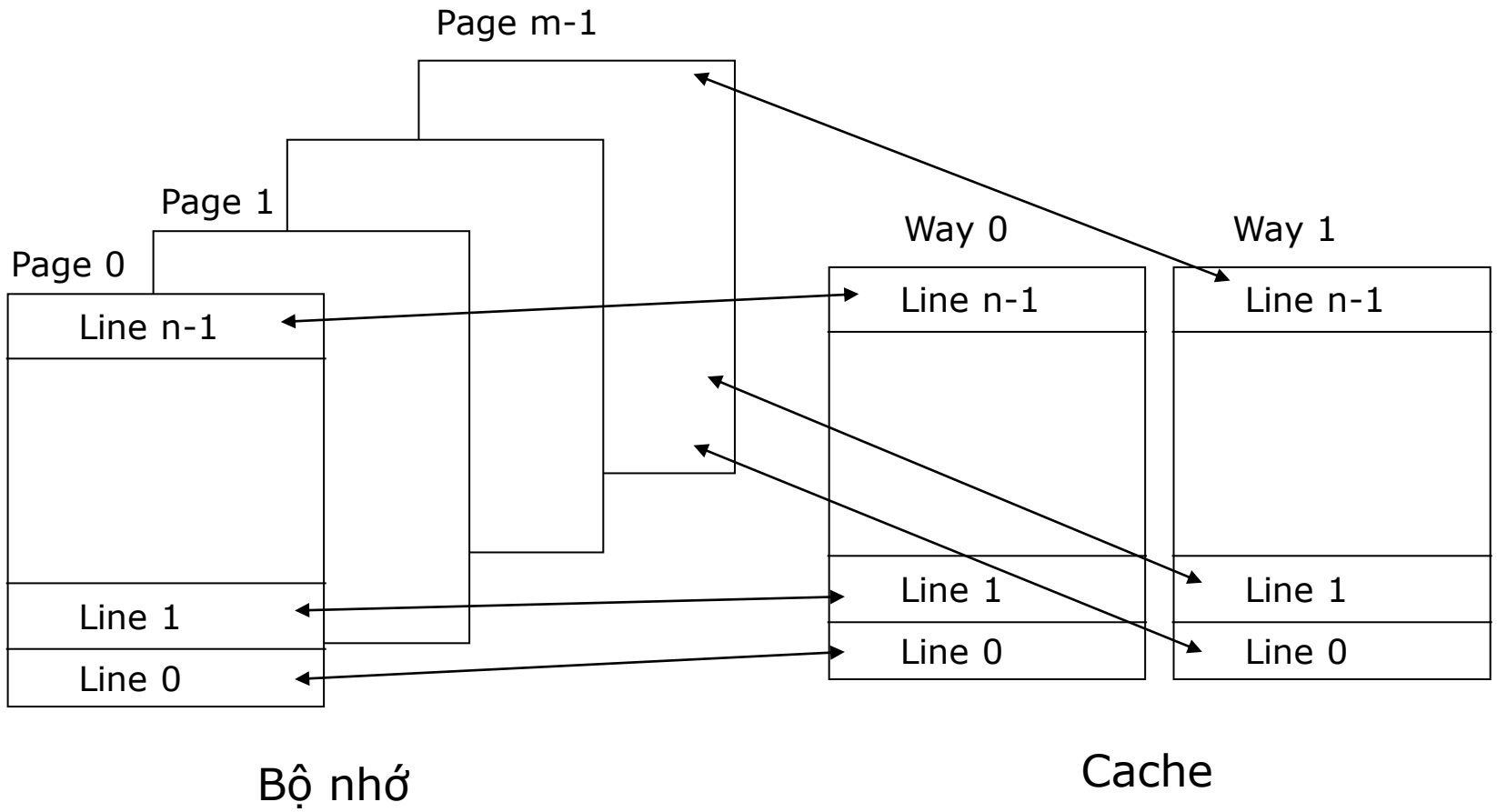
- Được chia thành *m* trang, từ $page_0$ đến $page_{m-1}$.
- Kích thước một trang bộ nhớ bằng kích thước một đường cache
- Mỗi trang có *n* dòng, từ $Line_0$ đến $Line_{n-1}$

Ánh xạ:

- ❖ Ánh xạ trang đến đường (ánh xạ mềm dẻo):
  - Một trang của bộ nhớ có thể ánh xạ đến một đường bất kỳ của cache.
- ❖ Ánh xạ dòng của trang đến dòng của đường (ánh xạ cố định):
  - $Line_0$ của $page_0$ ánh xạ đến $Line_0$ của way_j;
  - $Line_1$ của $page_i$ ánh xạ đến $Line_1$ của way_j;
  - ....
  - $Line_{n-1}$ của $page_i$ ánh xạ đến $Line_{j-1}$ của way_j

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Ánh xạ tập kết hợp (Set Associative)



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Ánh xạ tập kết hợp (Set Associative)

Địa chỉ cache:

- *Tag* (bit) là địa chỉ của trang trong bộ nhớ
- *Set* (bit) là địa chỉ của dòng trong đường cache
- *Word* (bit) là địa chỉ của từ trong dòng

Tag	Set	Word

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Ánh xạ tập kết hợp (Set Associative)

Ví dụ:

- Vào:
  - Dung lượng bộ nhớ = 4GB (hệ thống 32 bit)
  - Dung lượng cache = 1MB, 2 đường
  - Kích thước dòng = 32 byte
- Ra:
  - Kích thước dòng Line = 32 byte = 2^5 → Word = 5 bit
  - Dung lượng Cache = 1MB = 2^{20} → có 2^{20} / 2 đường / 2^5 = 2^{14} dòng/đường → Set = 14 bit
  - Đ/c trang Tag: 4GB = 232 ➔ cần 32 bít địa chỉ tổng cộng để địa chỉ hoá các ô nhớ:

    Tag = 32bit địa chỉ – Set – Word = 32 – 14 – 5 = 13 bit.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Ánh xạ tập kết hợp (Set Associative)

Ưu điểm:

- Nhanh do ánh xạ trực tiếp được sử dụng cho ánh xạ dòng (chiếm số lớn ánh xạ);
- Giảm xung đột do ánh xạ từ các trang bộ nhớ đến các đường cache là mềm dẻo.
- Hệ số Hit cao hơn.

Nhược điểm:

- Phức tạp trong thiết kế và điều khiển vì cache được chia thành một số đường.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Đọc ghi thông tin trong Cache

Đọc thông tin:

- Trường hợp hit (mẩu tin cần đọc có trong cache)
  - Mẩu tin được đọc từ cache vào CPU;
  - Bộ nhớ chính không tham gia.
- Trường hợp miss (mẩu tin cần đọc không có trong cache)
  - Mẩu tin trước hết được đọc từ bộ nhớ chính vào cache;
  - Sau đó nó được chuyển từ cache vào CPU.

    → đây là trường hợp miss penalty: thời gian truy nhập mẩu tin bằng tổng thời gian truy nhập cache và bộ nhớ chính.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Đọc ghi thông tin trong Cache

Ghi thông tin:

- Trường hợp hit (mẩu tin cần ghi có trong cache)
  - Ghi thẳng (write through): mẩu tin được ghi đồng thời ra cache và bộ nhớ chính;
  - Ghi trễ (write back): mẩu tin trước hết được ghi ra cache và dòng chứa mẩu tin được ghi ra bộ nhớ chính khi dòng đó bị thay thế.
- Trường hợp miss (mẩu tin cần ghi không có trong cache)
  - Ghi có đọc lại (write allocate / fetch on write): mẩu tin trước hết được ghi ra bộ nhớ chính và sau đó dòng chứa mẩu tin được đọc vào cache;
  - Ghi không đọc lại (write non-allocate): mẩu tin chỉ được ghi ra bộ nhớ chính (dòng chứa mẩu tin không được đọc vào cache).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Vì sao phải thay thế dòng cache ?

- Ánh xạ dòng (bộ nhớ) → dòng (cache) thường là ánh xạ nhiều → một;
- Nhiều dòng bộ nhớ chia sẻ một dòng cache → các dòng bộ nhớ được nạp vào cache sử dụng một thời gian và được thay thế bởi dòng khác theo yêu cầu thông tin phục vụ CPU.

Chính sách thay thế (replacement policies) xác định các dòng cache nào được chọn để thay thế bởi các dòng khác từ bộ nhớ.

Các chính sách thay thế:

- Ngẫu nhiên (Random)
- Vào trước ra trước (FIFO)
- Thay thế các dòng ít được sử dụng gần đây nhất (LRU).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Thay thế ngẫu nhiên (Radom Replacement):

- Các dòng cache được chọn ngẫu nhiên để thay thế
- Ưu:
  - Cài đặt đơn giản
- Nhược:
  - Hệ số miss cao:
    - Thay thế ngẫu nhiên không xem xét đến các dòng cache đang thực sự được sử dụng
    - Nếu một dòng cache đang được sử dụng và bị thay thế → xảy ra miss và nó lại cần được đọc từ bộ nhớ chính vào cache.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Thay thế kiểu vào trước ra trước (FIFO-First In First Out):

- Các dòng nhớ được đọc vào cache trước sẽ bị thay thế trước
- Ưu:
  - Có hệ số miss thấp hơn so với thay thế ngẫu nhiên
- Nhược:
  - Hệ số miss vẫn còn cao
    - Vẫn chưa thực sự xem xét đến các dòng cache đang được sử dụng. Một dòng cache "già" vẫn có thể đang được sử dụng.
    - Cài đặt phức tạp do cần có mạch điện tử để theo dõi trật tự nạp các dòng bộ nhớ vào cache.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Thay thế các dòng ít được sử dụng gần đây nhất (LRU-Least Recently Used)

- Các dòng cache ít được sử dụng gần đây nhất được lựa chọn để tha thế.
- Ưu:
  - Có hệ số miss thấp nhất so với thay thế ngẫu nhiên và thay thế FIFO
  - Do thay thế LRU có xem xét đến các dòng đang được sử dụng
- Nhược:
  - Cài đặt phức tạp do cần có mạch điện tử để theo dõi tần suất sử dụng các dòng cache.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Hiệu năng Cache

Thời gian truy nhập trung bình (t_{access}) của một hệ thống nhớ có cache:

t_{access} = (Hit cost) + (miss rate) * (miss penalty)

$t_{access} = t_{cache} + (1 - H) * (t_{memory})$

trong đó H là hệ số hit, t_{cache} là thời gian truy nhập cache,
t_{memory} là thời gian truy nhập bộ nhớ chính.

Nếu t_{cache} = 5ns, t_{memory} = 60ns và H=80%, ta có:

t_{access} = 5 + (1 – 0.8) * (60) = 5+12 = 17ns

Nếu t_{cache} = 5ns, t_{memory} = 60ns và H=95%, ta có:

t_{access} = 5 + (1 – 0.95) * (60) = 5+3 = 8ns

➔ Thời gian truy nhập trung bình tiệm cận thời gian truy nhập cache.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các yếu tố ảnh hưởng đến hiệu năng Cache

Các yếu tố ảnh hưởng đến hiệu năng cache:

- Kích thước cache:
  - Kích thước cache nên lớn hay nhỏ ?
- Tách cache:
  - Cache được tách thành 2 phần: cache lệnh (I-Cache) và D-Cache
- Tạo cache thành nhiều mức:
  - Cache được thiết kế thành nhiều mức: L1 – L2 – L3, ... với kích thước tăng dần.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các yếu tố ảnh hưởng đến hiệu năng Cache

Kích thước cache:

- Số liệu thống kê cho thấy:
  - Kích thước cache không ảnh hưởng nhiều đến hệ số miss
  - Hệ số miss của cache lệnh thấp hơn nhiều so với cache dữ liệu

8KB cache lệnh có hệ số miss < 1%

256KB cache lệnh có hệ số miss < 0.002% → tăng kích thước cache lệnh không giảm miss hiệu quả.

8KB cache dữ liệu có hệ số miss < 4%

256KB cache dữ liệu có hệ số miss < 3%

tăng kích thước cache dữ liệu lên 32 lần, hệ số miss giảm 25% (từ 4% xuống 3%).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các yếu tố ảnh hưởng đến hiệu năng Cache

Kích thước cache:

- Cache có kích thước lớn:
  - Có thể tăng được số dòng bộ nhớ lưu trong cache
  - Giảm tần suất tráo đổi các dòng cache của các chương trình khác nhau với bộ nhớ chính
  - Cache lớn thường chậm hơn cache nhỏ (tại sao?)
    - Không gian tìm kiếm địa chỉ ô nhớ lớn hơn
    - Xu hướng tương lai: cache càng lớn càng tốt (tại sao?)
      - Hỗ trợ đa nhiệm tốt hơn
      - Hỗ trợ xử lý song song tốt hơn
      - Hỗ trợ tốt hơn các hệ thống CPU nhiều nhân

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các yếu tố ảnh hưởng đến hiệu năng Cache

Tách cache

- Cache có thể được tách thành cache lệnh (I-Cache) và cache dữ liệu (D-Cache) để cải thiện hiệu năng, do:
  - Dữ liệu và lệnh có tính lân cận khác nhau;
  - Dữ liệu thường có tính lân cận về thời gian cao hơn lân cận về không gian; lệnh có tính lân cận về không gian cao hơn lân cận về thời gian;
  - Cache lệnh chỉ cần hỗ trợ thao tác đọc; cache dữ liệu cần hỗ trợ cả 2 thao tác đọc và ghi → tách cache giúp tối ưu hoá dễ dàng hơn;
    - Tách cache hỗ trợ nhiều lệnh truy nhập đồng thời hệ thống nhớ → giảm xung đột tài nguyên cho CPU pipeline.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

- Trên thực tế, hầu hết cache L1 được tách thành 2 phần:
  - I-Cache (Instruction Cache): cache lệnh
  - D-Cache (Data Cache): cache dữ liệu

  I-Cache và D-Cache thường hỗ trợ nhiều lệnh truy nhập
- Các cache ở mức cao hơn không được tách.
  - Cache L1 được tách vì nó ở gần CPU nhất; CPU trực tiếp đọc ghi lên cache L1. Cache L1 cần hỗ trợ nhiều lệnh truy nhập đồng thời và các biện pháp tối ưu hoá;
  - Các mức cao hơn của cache ít được tách do:
    - Điều khiển phức tạp
      - Hiệu quả mang lại không thực sự cao, do CPU không trực tiếp đọc/ghi các mức cache này. Hơn nữa, các mức cache cao hơn trao đổi dữ liệu với cache L1 theo khối, nên việc hỗ trợ nhiều lệnh truy nhập đồng thời không có nhiều ý nghĩa.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Tạo cache thành nhiều mức:

- Cải thiện được hiệu năng hệ thống do hệ thống cache nhiều mức có khả năng dung hoà tốt hơn tốc độ của CPU với tốc độ củabộ nhớ chính.

CPU	L1	L2	L3	Bộ nhớ chính
1ns	5ns	15ns	30ns	60ns
1ns	5ns			60ns

- Trên thực tế, đa số cache được tổ chức thành 2 mức: L1 và L2. Một số cache có 3 mức: L1, L2 và L3.
- Giảm giá thành hệ thống nhớ.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các biện pháp giảm Miss

Cache tốt

- Hệ số hit cao
- Hệ số miss thấp
- Nếu xảy ra miss thì không quá chậm

Các loại miss

- Miss bắt buộc (Compulsory misses): thường xảy ra tại thời điểm chương trình được kích hoạt, khi mã chương trình đang được tải vào bộ nhớ và chưa được nạp vao cache.
    - Miss do dung lượng (Capacity misses): thường xảy ra do kích thước của cache hạn chế, đặc biết trong môi trường đa nhiệm. Do kích thước cache nhỏ nên mã của các chương trình thường xuyên bị tráo đổi giữa bộ nhớ và cache.
    - Miss do xung đột (Conflict misses): xảy ra khi có nhiều dòng bộ nhớ cùng cạnh tranh một dòng cache.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các biện pháp giảm Miss

Tăng kích thước dòng cache:

- Giảm miss bắt buộc
  - Dòng kích thước lớn sẽ có khả năng bao phủ lân cận tốt hơn → giảm miss bắt buộc;
- Tăng miss do xung đột
  - Dòng kích thước lớn sẽ làm giảm số dòng cache → tăng mức độ cạnh tranh → tăng miss do xung đột
    - Dòng kích thước lớn có thể gây lãng phí dung lượng cache. Do dòng lớn nên có thể có nhiều phần của dòng cache không bao giờ được sử dụng.
    - Kích thước dòng thường dùng hiện nay là 64 bytes.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

Các biện pháp giảm Miss

Tăng mức độ liên kết của cache (tăng số đường cache):

- Giảm miss xung đột:
  - Tăng số đường cache → tăng tính mềm dẻo của ánh xạ trang bộ nhớ - đường cache (nhiều lựa chọn hơn) → giảm miss xung đột.
- Làm cache chậm hơn:
  - Tăng số đường cache → tăng không gian tìm kiếm địa chỉ ô nhớ → làm cache chậm hơn.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID (Redundant Array of Independent Disks) là một công nghệ thiết kế hệ thống lưu trữ dữ liệu tiên tiến trên cơ sở các ổ đĩa cứng, nhằm đạt được các mục đích:

- Tốc độ cao (High performance / speed)
- Tính tin cậy cao (High reliability)
- Dung lượng lớn (Large volume)

RAID:

- Một tập hợp các đĩa cứng được hệ điều hành xem như một ổ đĩa duy nhất.
- Các sơ đồ bố trí RAID có 7 mức từ Level 0 đến 6
- Các đĩa cứng theo chuẩn SATA và SCSI mới hỗ trợ tạo RAID.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

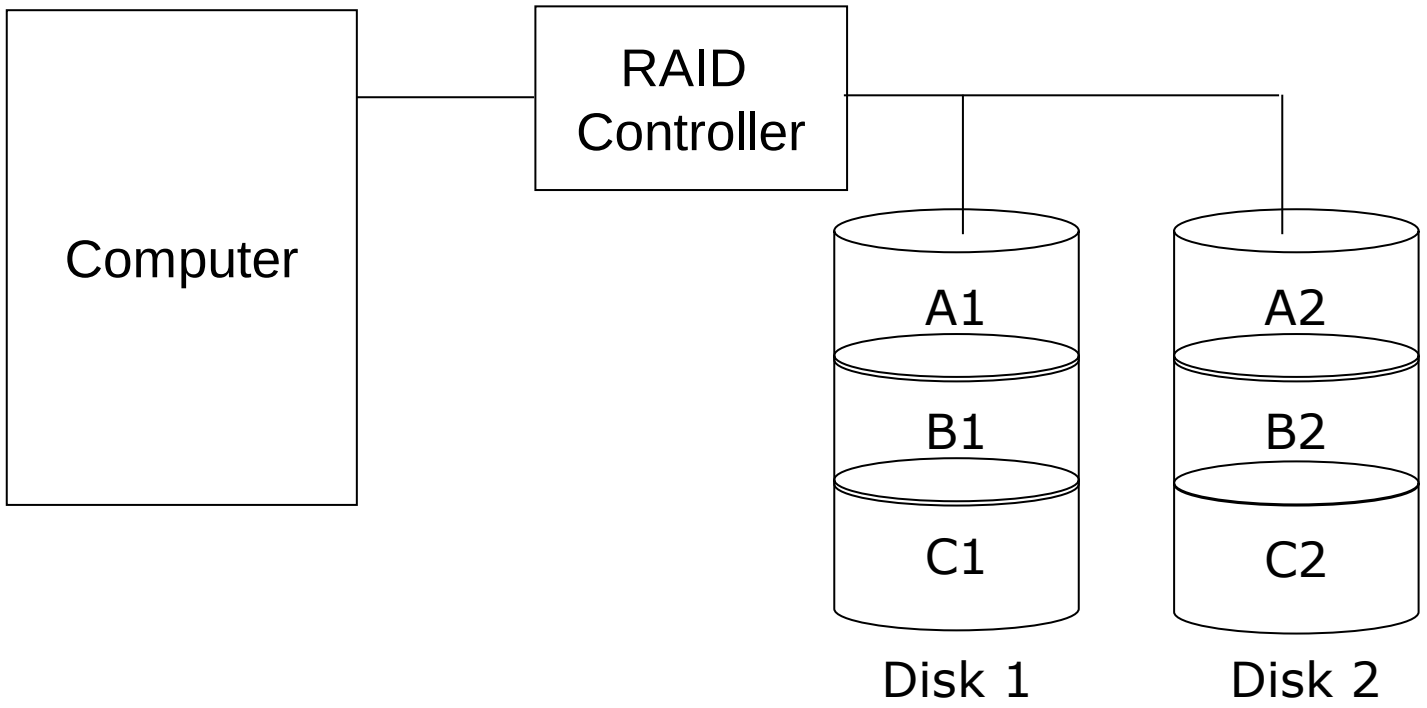
Hai kỹ thuật chính được sử dụng trong RAID:

- **Tạo lát đĩa** (Disk Stripping):
  - Ghi: Dữ liệu được chia thành các khối, mỗi khối được ghi đồng thời vào một đĩa độc lập;
  - Đọc: Các khối dữ liệu được đọc đồng thời ở các đĩa độc lập và được ghép lại tạo dữ liệu hoàn chỉnh → *tốc độ truy nhập được cải thiện*.
    - **Soi gương đĩa** (Disk Mirroring):
      - Ghi: Dữ liệu được chia thành các khối, mỗi khối được ghi đồng thời vào nhiều đĩa độc lập;
      - Tại mọi thời điểm ta luôn có nhiều hơn 1 bản sao vật lý của dữ liệu.

        → *Tính tin cậy được cải thiện.*

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

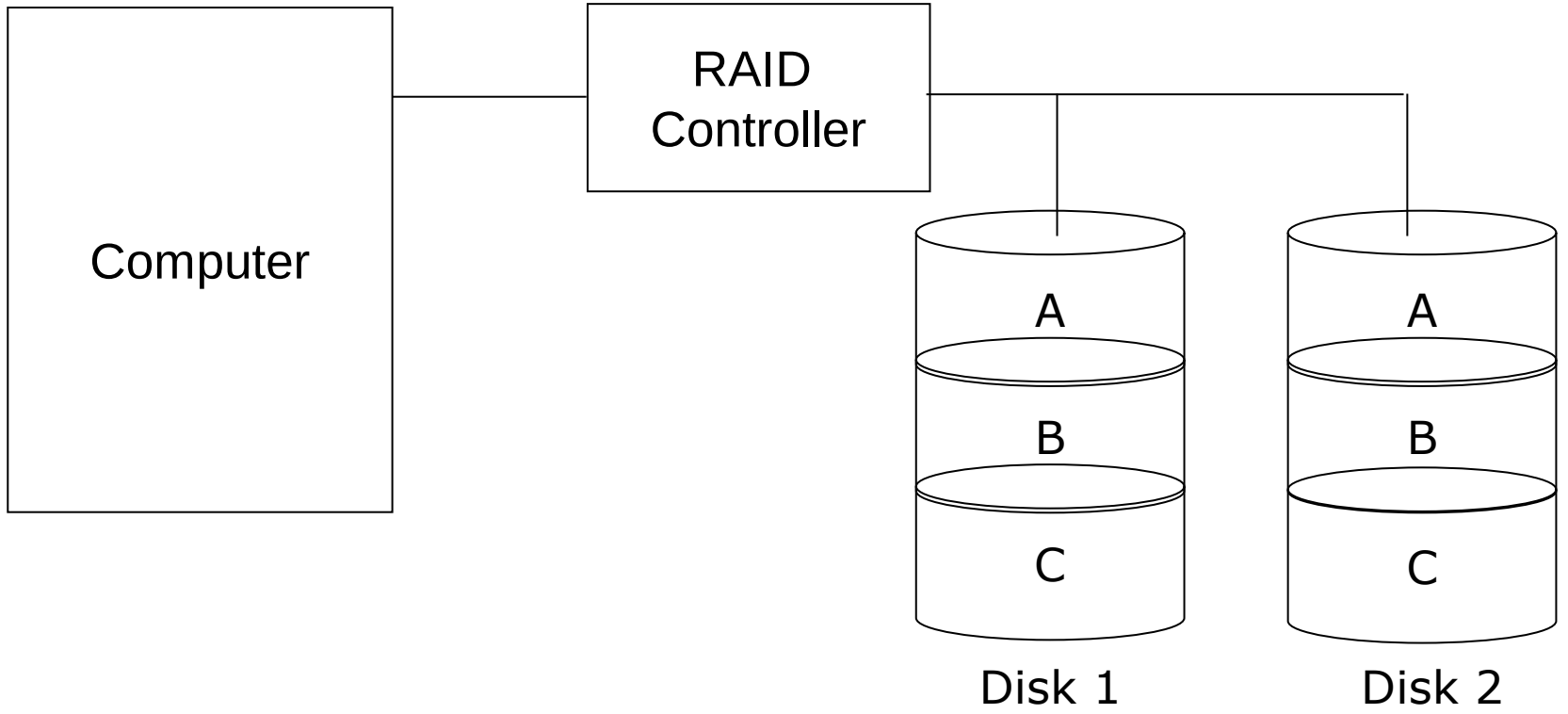
3.4 Công nghệ lưu trữ dữ liệu RAID



Kỹ thuật Disk Stripping

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID



Kỹ thuật Disk Mmirroring

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

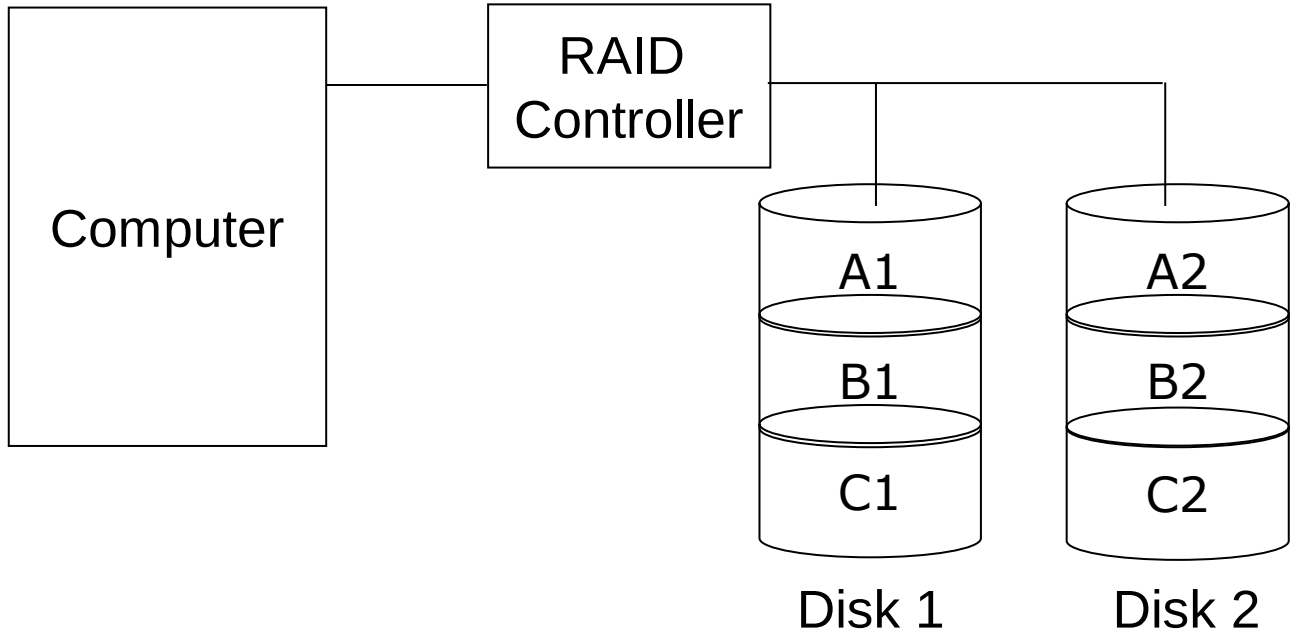
Có nhiều mức độ RAID.

- RAID 0
- RAID 1
- RAID 10
- RAID 01
- RAID 2
- RAID 3
- RAID 4
- RAID 5
- RAID 6

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID 0



RAID 0 - Disk stripping

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID 0

Đặc điểm:

- Yêu cầu tối thiểu **2** ổ đĩa cứng vật lý
- Sử dụng kỹ thuật tạo lát đĩa (disk stripping hoặc parallel read/write)

Ưu điểm:

- Nhanh: tốc độ truy nhập tỷ lệ thuận với số đĩa của RAID
- Tăng dung lượng: dung lượng RAID bằng tổng dung lượng các đĩa đơn.

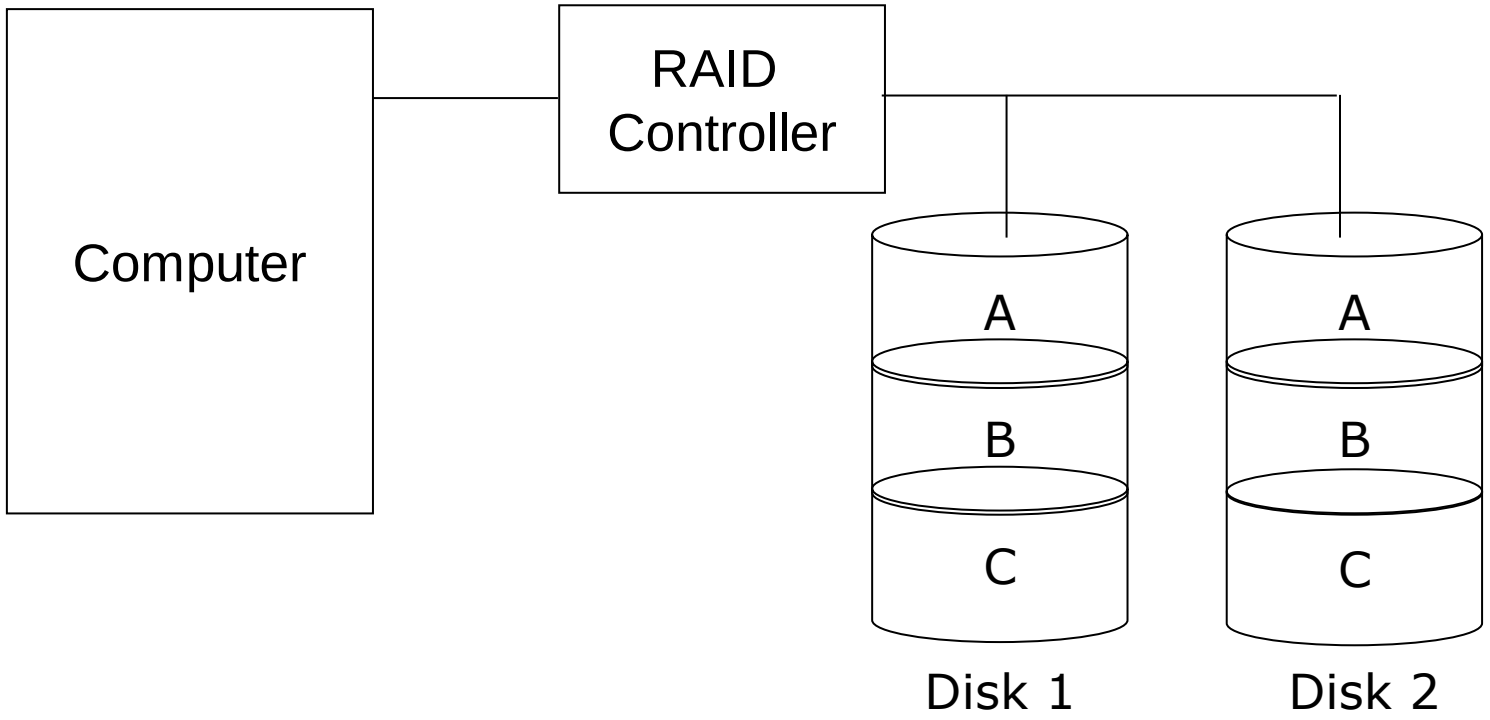
Nhược điểm:

- Tính an toàn chỉ tương đương một đĩa đơn

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID 1



RAID 1 - Disk mirroring

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID 1

Đặc điểm:

- Yêu cầu tối thiểu **2** ổ đĩa cứng vật lý
- Sử dụng kỹ thuật soi gương đĩa (disk mirroring)

  Ưu điểm:

- An toàn cao: do tại mỗi thời điểm RAID luôn chứa nhiều bản copy của dữ liệu ở các đĩa vật lý khác nhau.

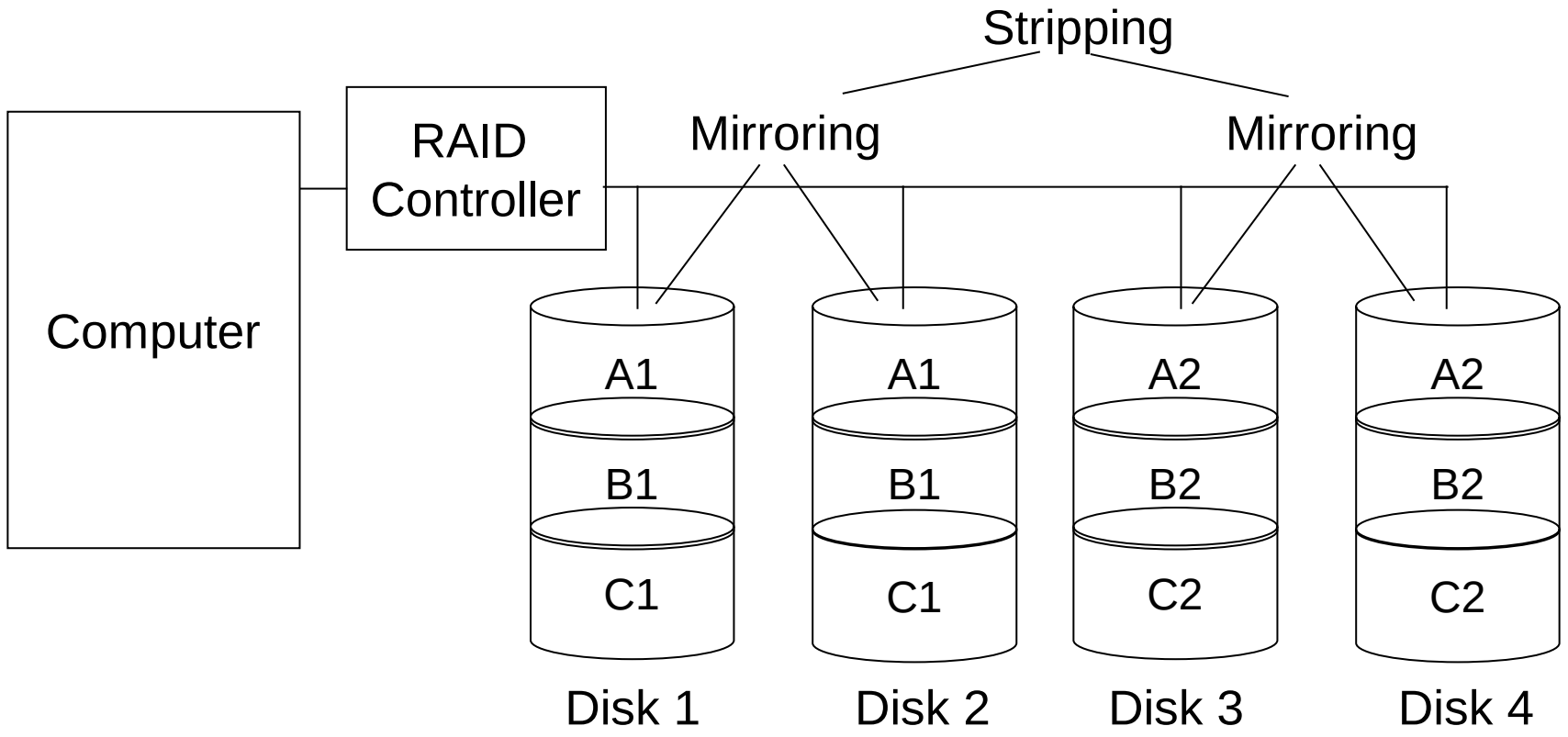
  Nhược điểm:

  - Tốc độ ghi đọc thấp chỉ tương đương một đĩa đơn
  - Dung lượng thấp chỉ tương đương một đĩa đơn

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID 10



RAID 10 là sự kết hợp của Disk Stripping và Disk Mirroring

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Công nghệ lưu trữ dữ liệu RAID

RAID 10

Đặc điểm:

- Yêu cầu tối thiểu 4 ổ đĩa cứng vật lý
- Sử dụng kỹ thuật tạo lát đĩa (disk stripping) và soi gương đĩa (disk mirroring)

Ưu điểm:

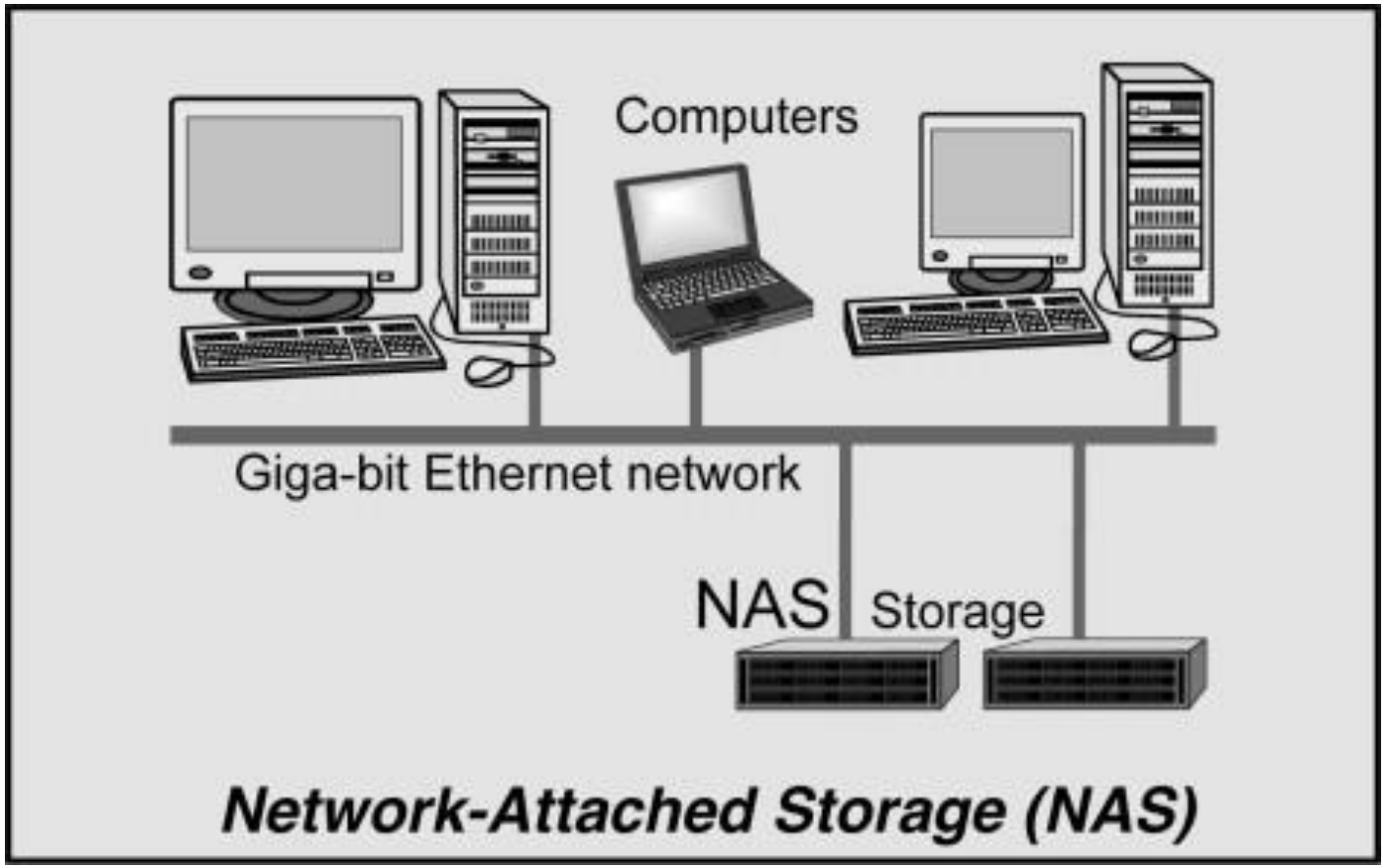
- *An toàn cao*: do tại mỗi thời điểm RAID luôn chứa nhiều bản copy của dữ liệu ở các đĩa vật lý khác nhau.
- *Nhanh*: tốc độ truy nhập tỷ lệ với số đĩa của RAID

Nhược điểm:

- *Dung lượng tăng* nhưng chỉ bằng ½ tổng dung lượng các đĩa đơn.
- *Giá thành cao*.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.5 Ổ đĩa mạng NAS (Network Attached Storage)



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.5 Ổ đĩa mạng NAS (Network Attached Storage)

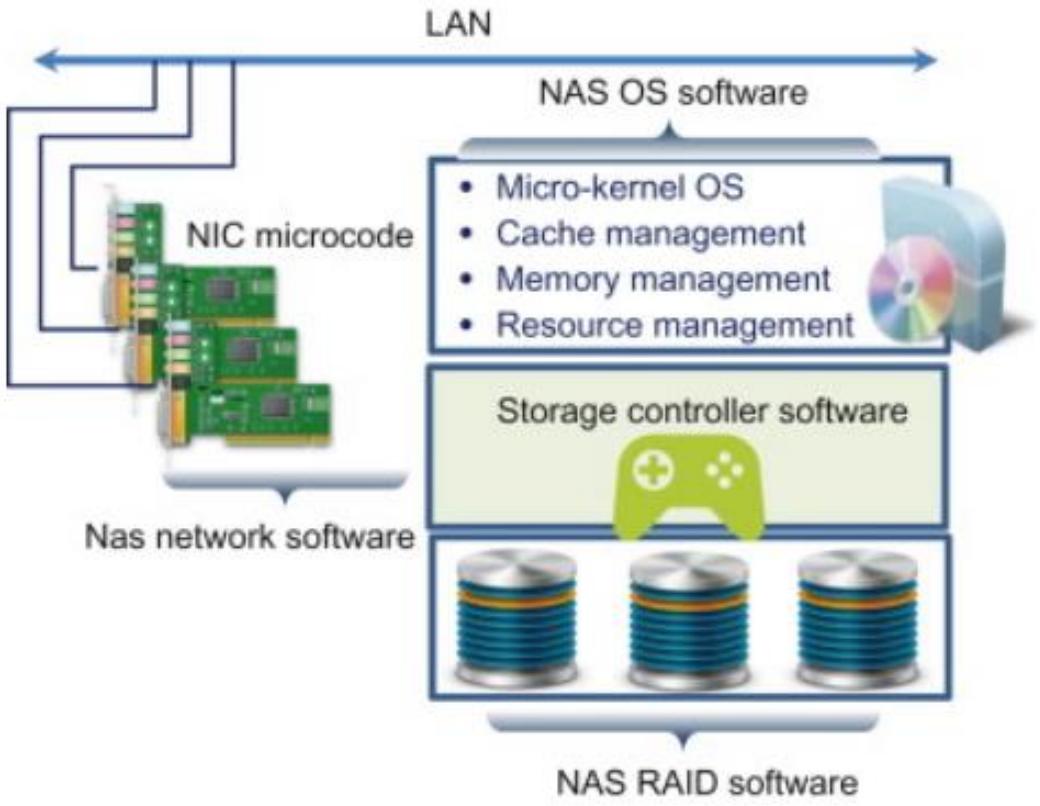
- NAS là một máy chủ chuyên dùng được sử dụng làm thiết bị lưu trữ;
- NAS được kết nối vào mạng (thường là LAN tốc độ cao) và cung cấp các dịch vụ lưu trữ thông qua mạng;
- NAS thường dựa trên nền tảng là một RAID có tốc độ cao, dung lượng lớn và độ tin cậy rất cao.
- NAS có thể cung cấp dịch vụ lưu trữ cho hầu hết các loại máy chủ có cấu hình phần cứng khác nhau và chạy các hệ điều hành khác nhau.

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Ổ đĩa mạng NAS (Network Attached Storage)

Các thành phần của NAS

- ❖ Phần mềm và hệ thống tệp tin
  - Phần mềm hệ điều hành NAS gồm 4 thành phần
  - Vi mã NIC
  - Phần mềm RAID

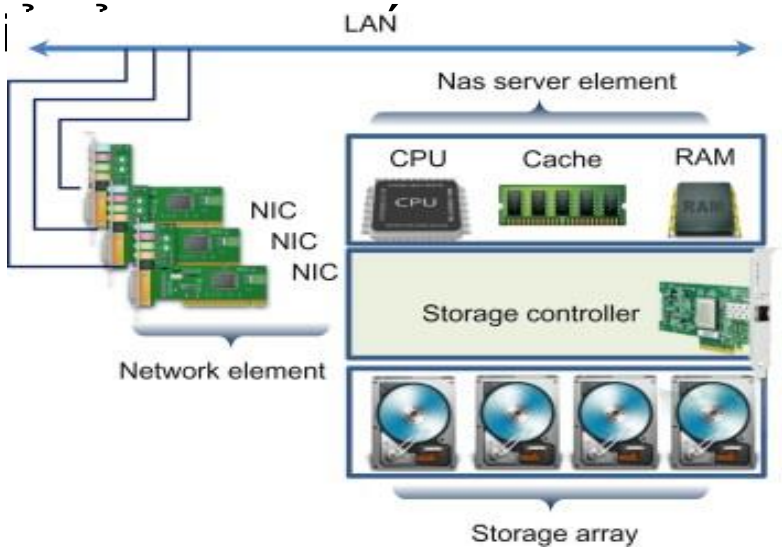


Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.4 Ổ đĩa mạng NAS (Network Attached Storage)

Các thành phần của NAS

- Các phần tử mạng và máy chủ đặc biệt
  - Đầu máy NAS: đây là một máy chủ đặc biệt, bao gồm CPU, RAM và các hệ thống bus.
  - Card mạng (NIC): Có thể bao gồm nhiều loại card mạng khác nhau như Gigabit Ethernet, Fast Ethernet hoặc FDDI (fiber distributed data interface)
  - Thiết bị lưu trữ: bộ điều khi[illegible] RAID và LUN



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.6 SAN (Storage Area Network)

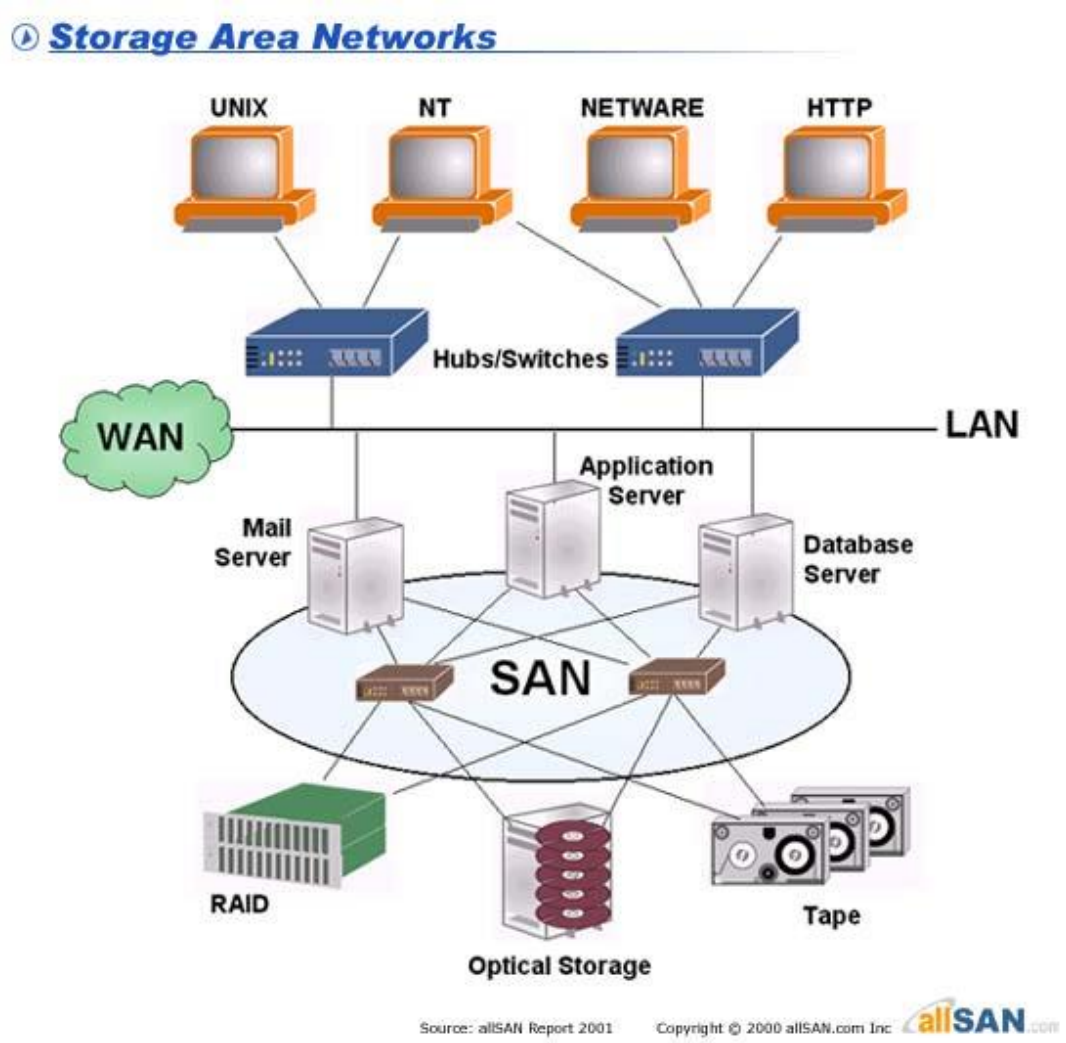
SAN là một mạng của các máy chủ chuyên dụng cung cấp dịch vụ lưu trữ;

SAN thường cung cấp dịch vụ lưu trữ với các đặc điểm:

- Tốc độ truy nhập rất cao;
- Dung lượng cực lớn;
- Độ an toàn rất cao
  - An toàn dữ liệu cục bộ
  - An toàn dữ liệu với các bản copy được đồng bộ ở khoảng cách xa về địa lý
    - SAN thường được tổ chức dưới dạng các hệ thống file phân tán (Distributed File System).

Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.6 SAN (Storage Area Network)



Chương 3. Xử lý xen kẽ dòng mã lệnh và Cache

3.6 Object Storage (Lưu trữ đối tượng)