**Một số lưu ý về cách thức mapping giữa bộ nhớ đệm (cache) và bộ nhớ chính (RAM)**

1. **Cách lưu trữ**

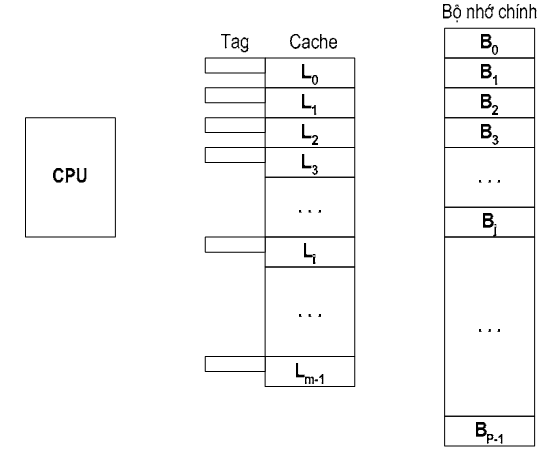
* Cả cache lẫn RAM đều được phân thành nhiều ô nhớ, mỗi ô nhớ có dung lượng mặc định là 1 byte
  + Cache = 256 KB = 218 byte  Cache có tổng cộng 218 ô nhớ
  + RAM = 4 GB = 232 byte  RAM có tổng cộng 232 ô nhớ
* Đối với cache, nó sẽ gom W ô nhớ liên tiếp nhau (tức W byte) để tạo thành 1 Line
* Đới với RAM, do quy định kích thước 1 Block = 1 Line nên W ô nhớ liên tiếp trong RAM cũng sẽ tạo thành 1 Block
* Nếu ta dùng w bit để đánh địa chỉ các ô nhớ trong 1 Line (hoặc 1 Block) thì ta có công thức W = 2w
  + Kích thước 1 Line = 1 Block = W = 32 byte (25)  Mỗi Line (Block) có 32 ô nhớ  Ta cần w = 5 bit để có thể đánh địa chỉ tất cả ô nhớ trong 1 Line (Block)
* Trong quá trình lưu chuyển để thực thi lệnh CPU cần truy xuất giá trị 1 ô nhớ nào đó, nó sẽ đi tìm trong Cache xem ô nhớ đó có nằm trong đó sẵn hay chưa, do đó có 2 khả năng:
  + Cache hit: Ô nhớ CPU đang cần đang nằm sẵn trong Cache, do vậy chỉ cần lấy ra là xong
  + Cache miss: Ô nhớ CPU đang cần không nằm sẵn trong Cache mà vẫn còn nằm trong RAM, do vậy cần phải chuyển Block chứa ô nhớ đó từ RAM sang Cache để CPU truy xuất
* Câu hỏi đặt ra là:
  + Giả sử CPU muốn tìm 1 ô nhớ có địa chỉ là X (bit), làm sao nó biết được X đang nằm trong Cache (cache hit) hay vẫn còn nằm trong RAM (Cache miss) ?
  + Nếu xảy ra cache miss thì những Block chứa ô nhớ chuyển từ RAM vào Cache sẽ nằm ở Line nào trong Cache ? Việc chuyển vào xảy ra lúc nào ?

 Trong Cache người ta cần phải có 1 trường Tag kết nối với từng Line để chứa thông tin trước đó Block nào RAM đã được đưa vào Line này

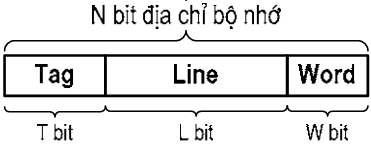
Do lượng ô nhớ trong Cache và RAM rất lớn, người ta buộc phải có 1 cách thức lưu trữ hợp lý và cụ thể, nếu không thì khi tìm kiếm phải vét cạn (duyệt hết tất cả các ô nhớ)

1. **Các thuật toán mapping (ánh xạ) giữa Cache và RAM**
   1. **Direct mapping**

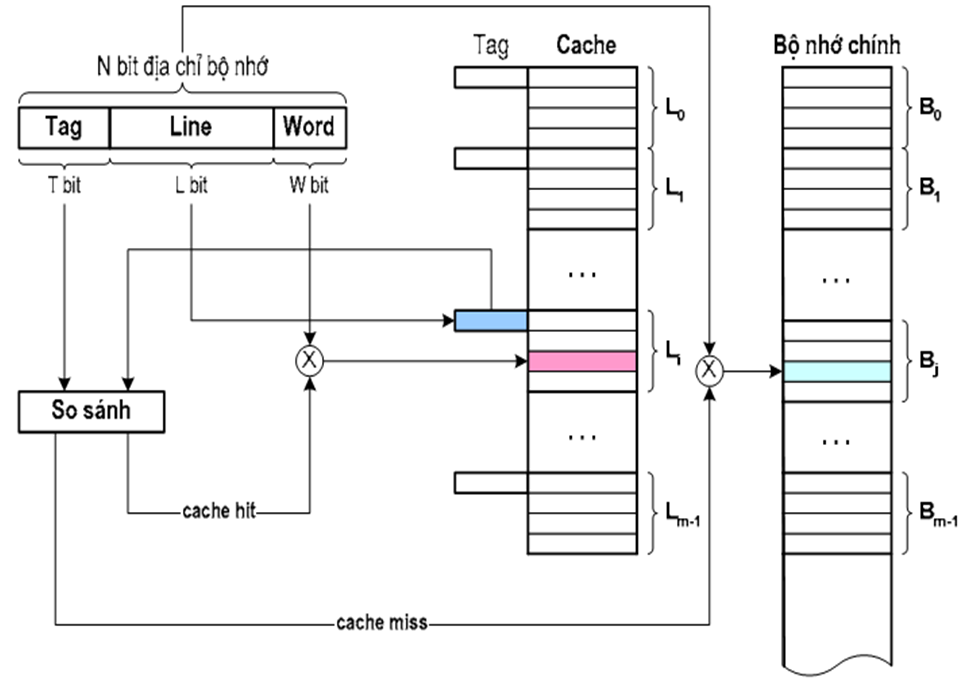
* Đây là phương pháp ánh xạ trực tiếp, có nghĩa là:
  + Block 0 đưa vào Cache chắc chắn sẽ được đưa vào Line thứ 0 của Cache
  + Block 1 đưa vào Cache chắc chắn sẽ được đưa vào Line thứ 1 của Cache
  + …
  + Block m – 1 đưa vào Cache chắc chắn sẽ được đưa vào Line thứ m - 1 của Cache
* Giả sử tới đây số lượng Line trong Cache chỉ là m Line trong khi số lượng Block lớn hơn rất nhiều, vậy Block thứ m sẽ được đưa vào Line nào? Câu trả lời là Line thứ 0 (xoay vòng)
  + Block m đưa vào Cache chắc chắn sẽ được đưa vào Line thứ 0 của Cache
  + Block m + 1 đưa vào Cache chắc chắn sẽ được đưa vào Line thứ 1 của Cache
  + …
* Suy ra tổng quát Block thứ j đưa vào Cache chắc chắn sẽ được đưa vào Line thứ j % m của Cache (với m là tổng số lượng Line trong Cache)



* Do vậy để biết được Block nào được đưa vào Line đang xét, ta sẽ sử dụng trường Tag của Line đó để xác định.
* Vậy bây giờ giả sử CPU cần truy xuất ô nhớ có địa chỉ là X gồm N bit, nếu bộ nhớ đang sử dụng phương pháp Direct mapping thì N bit đó phải được tổ chức như thế nào để trả lời được các câu hỏi đã đặt ra ở trên?
* Trả lời: N bit đó sẽ được chia thành 3 phần: Tag (bit) – Line (bit) – Word (bit) với ý nghĩa



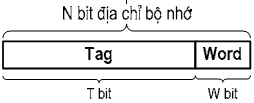
* + Word (có W bit chính là số bit để đánh địa chỉ các ô nhớ trong 1 Line hay 1 Block) Giá trị của trường Word cho biết ô nhớ CPU đang tìm nằm ở chính xác vị trí nào trong Line / Block (vì nó lưu giá trị địa chỉ ô nhớ trong Line / Block)
  + Line (có L bit chính là số bit để đánh địa chỉ các Line trong Cache): Giá trị của trường Line cho biết ô nhớ CPU đang tìm nằm ở chính xác Line nào trong Cache
  + Tag (có T bit = N – L – W): Ý nghĩa của Tag cho biết Block nào của RAM sẽ được lưu ở Line đang xét, do vậy giá trị của trường Tag (T bit) ở đây dùng để kiểm tra xem Line đang xét có thực sự lưu thông tin của ô nhớ (được chứa bởi Block trong RAM) đang tìm hay không
  + Ví dụ: Xem ví dụ trong slide Bộ nhớ (Slide 33, 34, 35)
* Vậy với cách thức lưu trữ mapping như vậy, khi xác định ô nhớ cần tìm nằm ở Cache (màu hồng) hay RAM (màu xanh) thì quy trình làm thế nào?



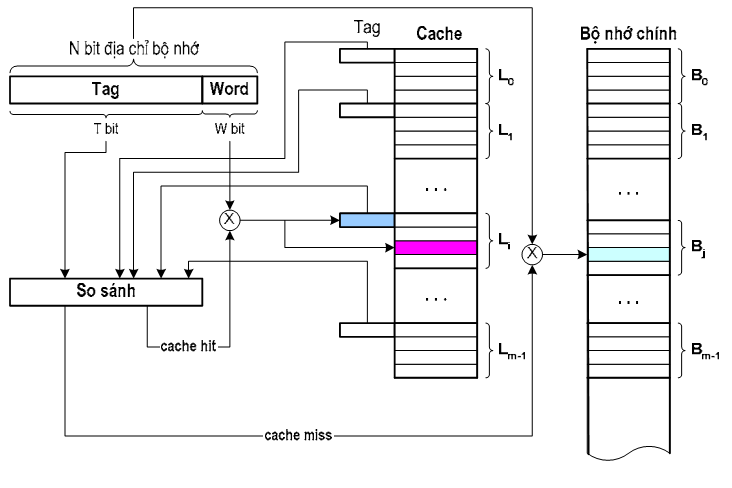
* Bước 1: Lấy giá trị Line trong N bit của X ra, đây là giá trị cho biết ô nhớ đang tìm nếu nằm trong Cache chỉ có thể nằm ở Line này
* Bước 2: Lấy giá trị Tag của Line đó ra, so khớp với giá trị Tag của X, có 2 khả năng:
  + Bằng nhau (Cache hit): Chắc chắn ô nhớ cần tìm đang nằm ở Line này, vấn đề còn lại chỉ là nằm ở ô nhớ nào trong các ô nhớ của Line đó thôi  Dễ dàng xác định nhờ vào giá trị Word (địa chỉ ô nhớ trong Line/Block)
  + Không bằng nhau (Cache miss): Chắc chắn ô nhớ cần tìm không nằm ở Line đó  nó nằm ở RAM. Để xác định nằm ở Block nào trong RAM, chỉ cần kết hợp 2 giá trị Tag và Line của X ta sẽ có địa chỉ của Block đang chứa ô nhớ đó trong RAM (Xem ví dụ trong slide 35). Vấn đề còn lại chỉ là nằm ở ô nhớ nào trong các ô nhớ của Block đó thôi  Dễ dàng xác định nhờ vào giá trị Word (địa chỉ ô nhớ trong Line/Block)

Nhận xét về Direct mapping: Xem slide 37

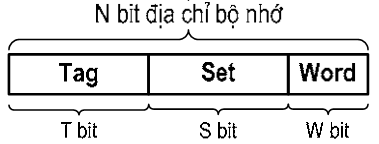
* 1. **Associative mapping**
* Để khắc phục điểm yếu của Direct mapping, phương pháp này cho phép 1 Block khi chuyển vào Cache sẽ có thể nằm ở bất kỳ Line nào trong Cache miễn sao Line đó đang còn trống.
* Nhưng cũng chính vì cách thức tổ chức như vậy nên sau đó khi CPU muốn tìm ô nhớ nào đó phải duyệt lần lượt từng Line một trong Cache cho đến khi gặp thì thôi 
* Đối với phương pháp này, trường địa chỉ ô nhớ X cần tìm gồm N bit sẽ không cần bận tâm đến Line nữa (vì trước đó nó lưu ở Line nào cũng được) mà chỉ còn Tag và Word thôi.
* Ví dụ: Xem slide 39, 40, 41



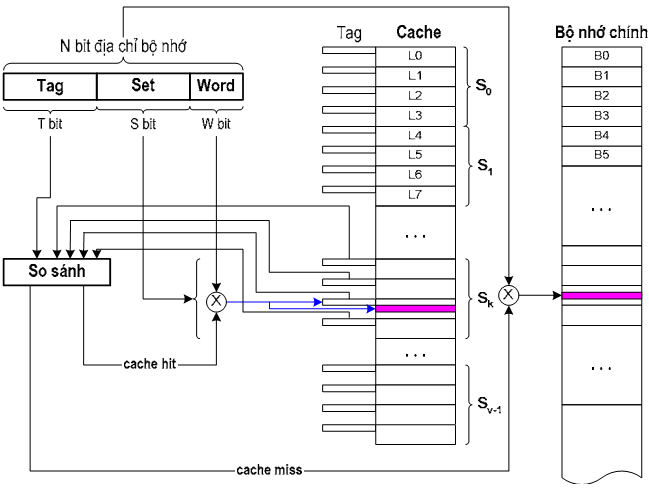
* Vậy với cách thức lưu trữ mapping như vậy, khi xác định ô nhớ cần tìm nằm ở Cache (màu hồng) hay RAM (màu xanh) thì quy trình làm thế nào?



* Bước 1: Duyệt lần lượt từng Line trong Cache. ứng với mỗi Line lấy giá trị Tag của nó ra
* Bước 2: So khớp giá trị lấy ra ở bước 1 với giá trị Tag của X, có 2 khả năng:
  + Bằng nhau (Cache hit): Chắc chắn ô nhớ cần tìm đang nằm ở Line này, vấn đề còn lại chỉ là nằm ở ô nhớ nào trong các ô nhớ của Line đó thôi  Dễ dàng xác định nhờ vào giá trị Word (địa chỉ ô nhớ trong Line/Block)
  + Không bằng nhau (Cache miss): Chắc chắn ô nhớ cần tìm không nằm ở Line đó  nó nằm ở RAM. Để xác định nằm ở Block nào trong RAM, chỉ cần lấy giá trị Tag của X ta sẽ có địa chỉ của Block đang chứa ô nhớ đó trong RAM (Xem ví dụ trong slide 41). Vấn đề còn lại chỉ là nằm ở ô nhớ nào trong các ô nhớ của Block đó thôi  Dễ dàng xác định nhờ vào giá trị Word (địa chỉ ô nhớ trong Line/Block)
* Nhận xét về Associative mapping: Xem slide 43
  1. **Set associative mapping**
* Phương pháp này là sự kết hợp của 2 phương pháp direct mapping & associative mapping bằng cách nó tận dụng các ưu điểm và cố gắng giảm bớt các khuyết điểm.
  + Thay vì khi tìm phải duyệt từ Line đầu tiên cho đến khi gặp Line chứa ô nhớ cần tìm thì dừng (trường hợp xấu nhất là phải duyệt hết tất cả các Line), đây là khuyết điểm của associative mapping, để khắc phục nó nghĩ ra 1 khái niệm mới là Set với 1 Set gồm nhiều Line (1 Set = 2/4/8/16… Line)
    - Ví dụ: 4 Line / Set à 4-way associative mapping
  + Thay vì 1 Block khi đưa vào Cache sẽ cho phép nằm ở Line nào cũng được trong Cache (associative mapping), nó sẽ chỉ cho phép nằm ở Line nào cũng được trong 1 Set thôi (do vậy khi tìm chỉ cần duyệt lần lượt từng Line trong Set chứ không phải từng Line trong toàn bộ Cache nữa  rút ngắn phạm vi tìm kiếm)
  + Tuy nhiên việc chọn Set nào để đưa Block vào thì phải tuân thủ theo nguyên tắc của Direct mapping, tức là:
    - Block 0 đưa vào Cache chắc chắn sẽ được đưa vào Set thứ 0 của Cache (Còn bên trong Set thì nằm ở Line nào cũng được)
    - Block 1 đưa vào Cache chắc chắn sẽ được đưa vào Set thứ 1 của Cache
    - …
    - Block m – 1 đưa vào Cache chắc chắn sẽ được đưa vào Set thứ m - 1 của Cache
* Đối với phương pháp này, trường địa chỉ ô nhớ X cần tìm gồm N bit sẽ không cần bận tâm đến Line nữa (vì trước đó nó lưu ở Line nào cũng được) mà chỉ còn Tag, Set và Word thôi với trường Set cho biết 1 Block từ RAM trước đó đưa vào sẽ nằm ở chính xác Set nào trong Cache.
* Ví dụ: Xem slide 45, 46



* Vậy với cách thức lưu trữ mapping như vậy, khi xác định ô nhớ cần tìm nằm ở Cache (màu hồng) hay RAM (màu xanh) thì quy trình làm thế nào?



* Bước 1: Lấy giá trị Set trong N bit của X ra, đây là giá trị cho biết ô nhớ đang tìm nếu nằm trong Cache chỉ có thể nằm ở Set này
* Bước 2: Duyệt lần lượt từng Line trong Set, ứng với mỗi Line lấy giá trị Tag của nó ra
* Bước 2: So khớp giá trị lấy ra ở bước 1 với giá trị Tag của X, có 2 khả năng:
  + Bằng nhau (Cache hit): Chắc chắn ô nhớ cần tìm đang nằm ở Line này, vấn đề còn lại chỉ là nằm ở ô nhớ nào trong các ô nhớ của Line đó thôi  Dễ dàng xác định nhờ vào giá trị Word (địa chỉ ô nhớ trong Line/Block)
  + Không bằng nhau (Cache miss): Chắc chắn ô nhớ cần tìm không nằm ở Line đó  nó nằm ở RAM. Để xác định nằm ở Block nào trong RAM, chỉ cần lấy giá trị Tag + Set của X ta sẽ có địa chỉ của Block đang chứa ô nhớ đó trong RAM (Xem ví dụ trong slide 46). Vấn đề còn lại chỉ là nằm ở ô nhớ nào trong các ô nhớ của Block đó thôi  Dễ dàng xác định nhờ vào giá trị Word (địa chỉ ô nhớ trong Line/Block)