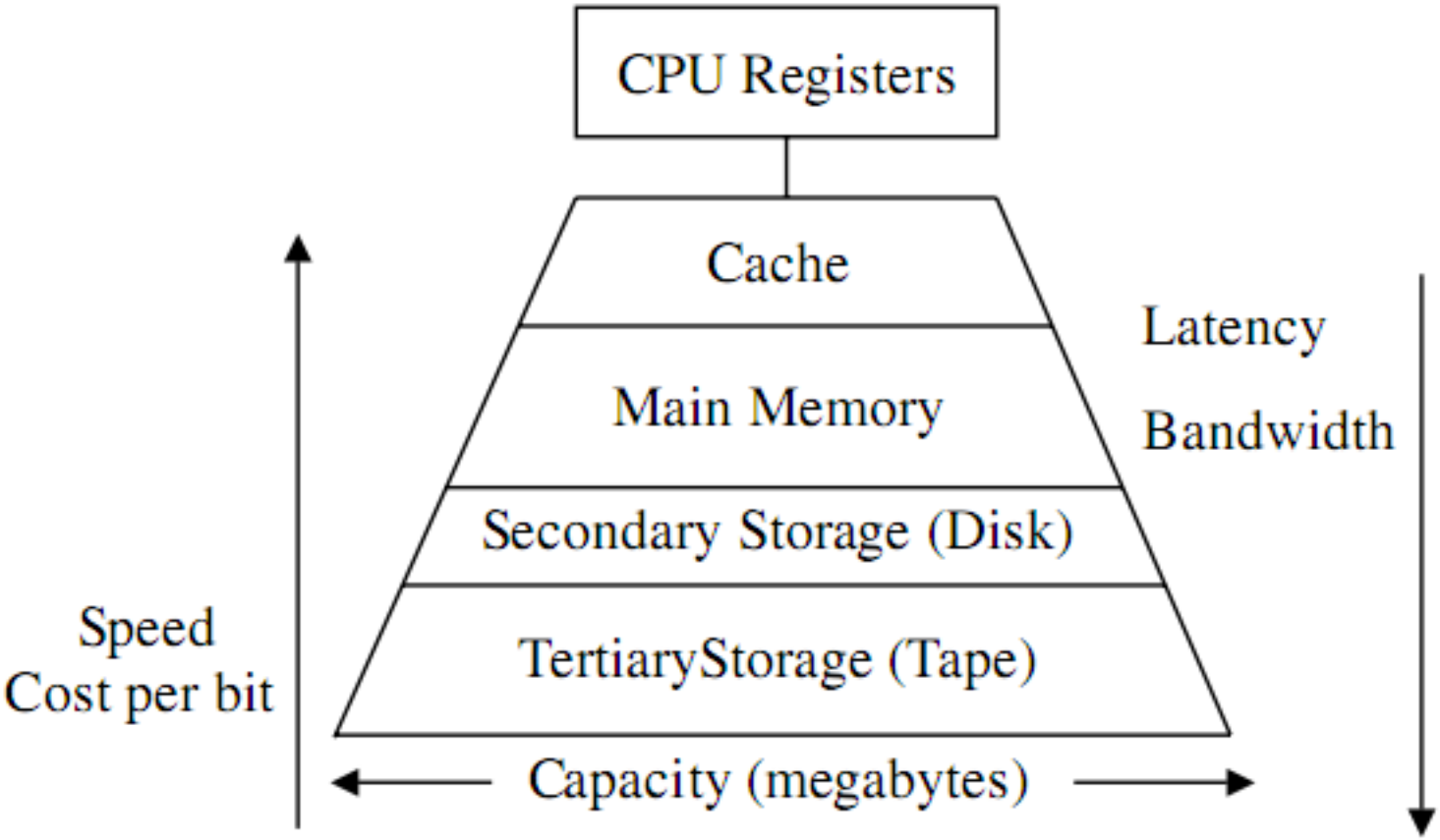
Chương 4: Hệ thống bộ nhớ

Chương 4: Nội dung chính

1. Giới thiệu về bộ nhớ trong và mô hình phân cấp bộ nhớ
2. Phân loại bộ nhớ và tổ chức mạch nhớ
3. ROM
4. RAM
5. Bộ nhớ cache

1. Mô hình phân cấp hệ thống bộ nhớ



Các tham số phân cấp bộ 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

Các thành phần phân cấp bộ nhớ

- Thanh ghi của CPU:
  - Kích thước rất nhỏ (vài chục byte tới vài KB)
  - Tốc độ rất nhanh, thời gian truy cập khoảng 0.25 ns
  - Giá thành đắt
  - Lưu trữ tạm thời dữ liệu đầu vào và ra cho các lệnh
- Cache:
  - Kích thước nhỏ (64KB tới 16MB)
  - Tốc độ nhanh, thời gian truy cập khoảng 1 – 5ns
  - Giá thành đắt
  - Lưu trữ lệnh và dữ liệu cho CPU
  - Còn được gọi là "bộ nhớ thông minh" (smart memory)

Các thành phần phân cấp bộ nhớ

- Bộ nhớ chính:
  - Kích thước lớn, dung lượng từ 256MB tới 4GB cho các hệ 32bits
  - Tốc độ chậm, thời gian truy cập từ 50 – 70ns
  - Lưu trữ lệnh và dữ liệu cho hệ thống và người dùng
  - Giá thành rẻ
- Bộ nhớ phụ:
  - Kích thước rất lớn, dung lượng từ 20GB tới 1000GB
  - Tốc độ rất chậm, thời gian truy cập khoảng 5ms
  - Lưu trữ lượng dữ liệu lớn dưới dạng file trong thời gian lâu dài
  - Giá thành rất rẻ

Vai trò của mô hình phân cấp

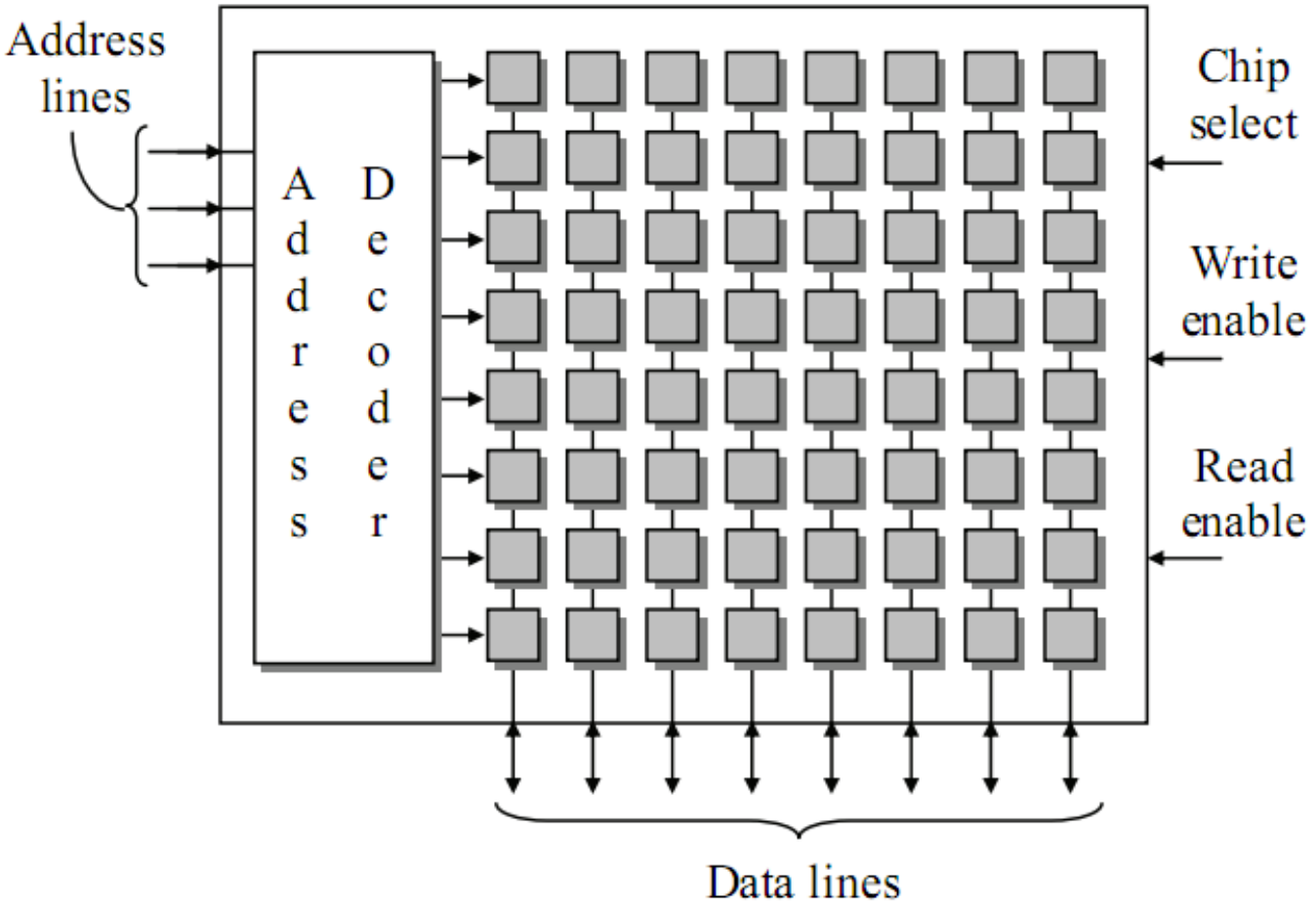
- Nâng cao hiệu năng hệ thống:
  - Dung hòa được CPU có tốc độ cao với bộ nhớ chính và bộ nhớ phụ có tốc độ thấp
  - Thời gian truy cập dữ liệu trung bình của CPU từ hệ thống bộ nhớ gần bằng thời gian truy cập cache
- Giảm giá thành sản xuất:
  - Các thành phần đắt tiền sẽ được sử dụng với dung lượng nhỏ hơn
  - Các thành phần rẻ hơn được sử dụng với dung lượng lớn hơn

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

2. Phân loại bộ nhớ

- Dựa vào kiểu truy cập:
  - Bộ nhớ truy cập ngẫu nhiên (RAM: Random Access Memory)
  - Bộ nhớ truy cập tuần tự (SAM: Serial Access Memory)
  - Bộ nhớ chỉ đọc (ROM: Read Only Memory)
- Dựa vào khả năng chịu đựng/ lưu giữ thông tin:
  - Bộ nhớ không ổn định (volatile memory): thông tin lưu trữ bị mất khi tắt nguồn
  - Bộ nhớ ổn định: thông tin lưu trữ được giữ lại khi tắt nguồn
- Dựa vào công nghệ chế tạo:
  - Bộ nhớ bán dẫn: ROM, RAM, SSD, USB, ...
  - Bộ nhớ từ: HDD, FDD, tape
  - Bộ nhớ quang: CD, DVD

Tổ chức mạch nhớ



Tổ chức của thiết bị nhớ

- Address lines:
  - Các đường địa chỉ nối tới bus A,
  - Truyền tín hiệu địa chỉ từ CPU tới mạch nhớ
- Address decoder:
  - Bộ giải mã địa chỉ
  - Sử dụng 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
  - Truyền dữ liệu từ bộ nhớ về CPU và ngược lại

Tổ chức của thiết bị nhớ

- Chip select CS:
  - Chân tín hiệu chọn chip
  - Chip nhớ được kích hoạt khi CS=0. Thông thường CPU chỉ làm việc với 1 chip nhớ tại 1 thời điểm
- Write enable WE:
  - Chân tín hiệu cho phép ghi
  - Cho phép ghi vào đường nhớ khi WE =0
- Read enable RE:
  - Chân tín hiệu cho phép đọc
  - Cho phép đọc dữ liệu từ đường nhớ khi RE =0

3. Bộ nhớ ROM

- ROM (Read Only Memory): là bộ nhớ chỉ đọc
  - Ghi thông tin vào ROM bằng cách sử dụng các thiết bị hoặc phương pháp đặc biệt
- ROM là bộ nhớ ổn định: tất cả thông tin vẫn được duy trì khi mất nguồn nuôi
- Là bộ nhớ bán dẫn: mỗi ô nhớ là một cổng bán dẫn
- Thường dùng để lưu trữ thông tin hệ thống: thông tin phần cứng và BIOS

Các loại ROM

- ROM nguyên thủy (Ordinary ROM):
  - ROM các thế hệ đầu tiên
  - Sử dụng tia cực tím để ghi thông tin
- PROM (Programmable ROM)
  - ROM có thể lập trình
  - Thông tin có thể được ghi vào PROM nhờ thiết bị đặc biệt gọi là bộ lập trình PROM

Các loại ROM

- EPROM (Erasable programmable read-only memory)
  - Là ROM có thể lập trình và xóa được
  - Thông tin trong EPROM có thể xóa bằng cách chiếu các tia cực tím có cường độ cao
- EEPROM (Electrically Erasable PROM :
  - Là PROM có thể xóa được thông tin bằng điện
  - Có thể ghi được thông tin sử dụng phần mềm chuyên dụng

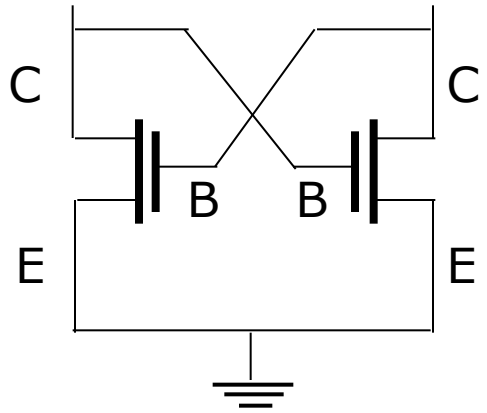
4. Bộ nhớ RAM

- RAM (Random Access Memory) là bộ nhớ truy cập ngẫu nhiên
  - Mỗi ô nhớ có thể được truy cập một cách ngẫu nhiên không theo trật tự nào
  - Tốc độ truy cập các ô nhớ là tương đương
- Là bộ nhớ không ổn định (dễ bay hơi): mọi thông tin lưu trữ sẽ bị mất khi tắt nguồn
- Là bộ nhớ bán dẫn. Mỗi ô nhớ là một cổng bán dẫn
- Sử dụng để lưu trữ thông tin hệ thống và của người dùng
  - Thông tin hệ thống: thông tin phần cứng & hệ điều hành
  - Thông tin người dùng: mã lệnh và dữ liệu các chương trình ứng dụng

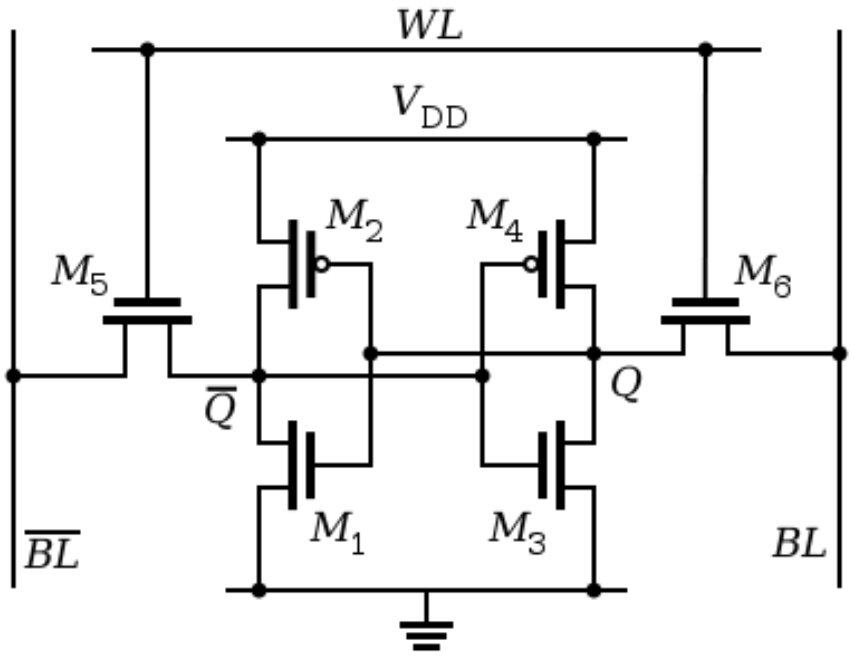
Các loại RAM

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

Cấu tạo SRAM



Một mạch lật đơn giản

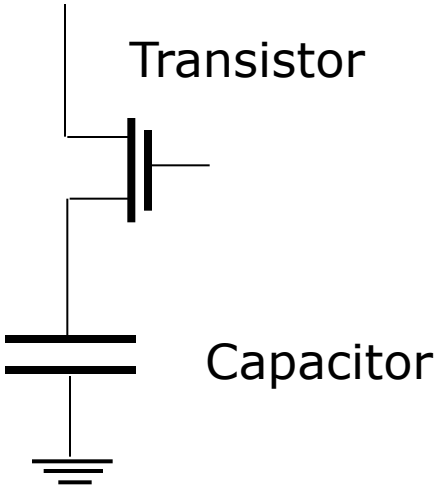


Một ô nhớ SRAM loại 6T

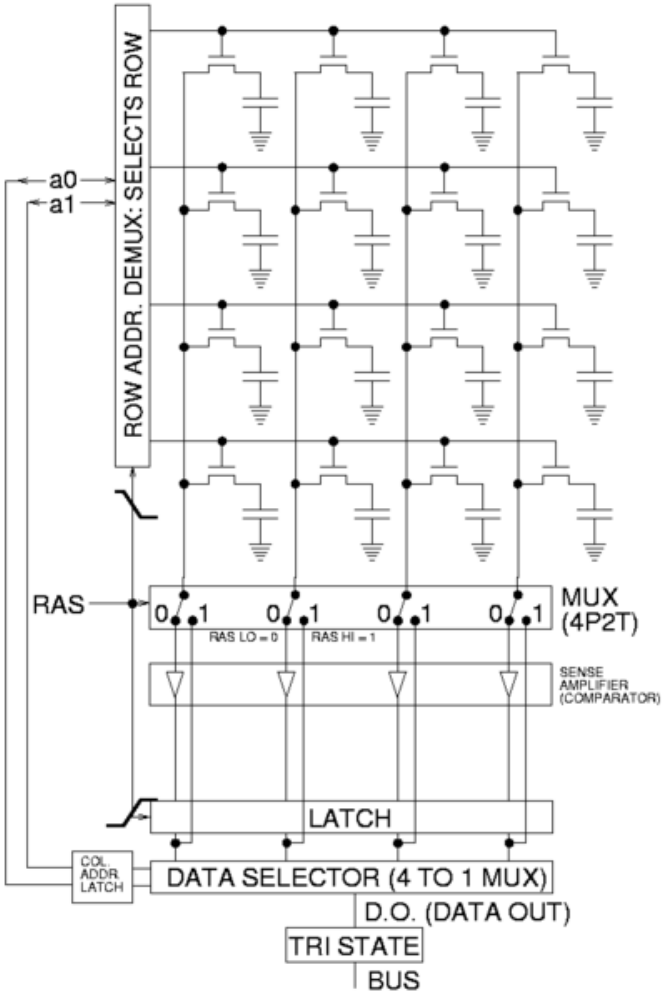
Các đặc điểm của SRAM

- SRAM sử dụng mạch lật trigơ lưỡng ổn (bistable latching circuit) để lưu 1 bit thông tin
- Mỗi mạch lật lưu 1 bit thường sử dụng 6, 8, 10 transistor (gọi là mạch 6T, 8T, 10T)
- SRAM thường có tốc độ truy cập nhanh vì:
  - Các bit 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 đắt vì:
  - Nó sử dụng nhiều transistor cho một bit hơn DRAM
  - Vì cấu trúc bên trong phức tạp hơn, mật độ của SRAM thấp hơn DRAM

Cấu tạo DRAM



Một bit DRAM



Các đặc điểm của DRAM

- Bit của DRAM dựa trên 1 tụ điện và 1 transistor.
  - 2 mức tích điện của tụ sẽ biểu diễn 2 mức logic:
    - Không tích điện: mức 0
    - Tích đầy điện: mức 1
- Do tụ thường tự phóng điện, điện tích trong tụ điện có xu hướng bị tổn hao
  - 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

Các đặc điểm của DRAM

- Các bit nhớ của DRAM thường được sắp xếp thành ma trận:
  - Một tụ + một transitor -> một bit
  - Các bit được tập hợp thành các cột và dòng
- DRAM chậm hơn SRAM vì:
  - Cần làm tươi định kỳ
  - Quá trình nạp điện tụ mất thời gian
  - Các mạch DRAM thường dùng kỹ thuật dồn kênh (địa chỉ cột/ hàng) để tiết kiệm đường địa chỉ
- DRAM rẻ hơn SRAM vì:
  - Cấu trúc đơn giản, sử dụng ít transistor
  - Mật độ cấy cao hơn

Các loại DRAM

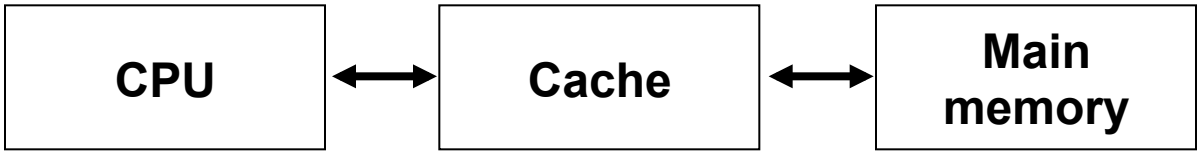
- SDRAM: Synchronous DRAM
- SRD SDRAM: (Single Data Rate SDRAM) chấp nhận 1 thao tác đọc/ ghi và chuyển 1 từ dữ liệu trong 1 chu kỳ đồng hồ; tốc độ 100MHz, 133MHz
- DDR SDRAM: Double Data Rate SDRAM
  - DDR1 SDRAM: DDR 266, 333, 400: có khả năng chuyển 2 từ dữ liệu trong 1 chu kỳ
  - DDR2 SDRAM: DDR2 400, 533, 800: có khả năng chuyển 4 từ dữ liệu trong 1 chu kỳ
  - DDR3 SDRAM: DDR3 800, 1066, 1333, 1600: có khả năng chuyển 8 từ dữ liệu trong 1 chu kỳ

5. Bộ nhớ cache

- Cache là gì?
- Vai trò của cache
- Các nguyên lý cơ bản hoạt động của cache
- Kiến trúc cache
- Tổ chức cache
- Đọc/ ghi trong cache
- Các chính sách thay thế
- Cách thức để nâng cao hiệu năng cache

Cache là gì?

- Cache là thành phần nhớ trong sơ đồ phân cấp bộ nhớ máy tính.
  - Nó hoạt động như thành phần trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại
- Vị trí của cache:
  - Với các hệ thống cũ, cache thường nằm ngoài CPU
  - Với các CPU mới, cache thường được tích hợp vào trong CPU



Cache là gì?

- Dung lượng cache thường nhỏ:
  - Với các hệ thống cũ: 16K, 32K,..., 128K
  - Với các hệ thống mới: 256K, 512K, 1MB, 2MB, …
- Tốc độ truy nhập của cache nhanh hơn so với tốc độ bộ nhớ chính
- Giá thành cache (tính trên bit) thường đắt hơn so với bộ nhớ chính
- Với các hệ thống CPU mới, cache thường được chia thành nhiều mức:
  - Mức 1: 16 – 32 KB có tốc độ rất cao
  - Mức 2: 1 -16MB có tốc độ khá cao

Vai trò của Cache

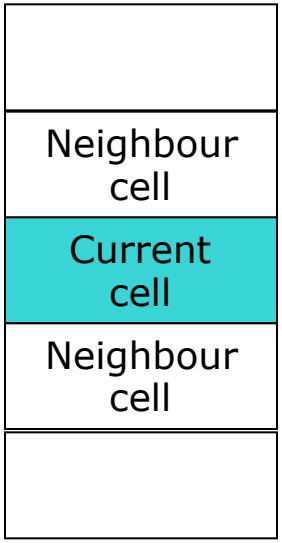
- Nâng cao hiệu năng hệ thống:
  - Dung hòa giữa CPU có tốc độ cao và bộ nhớ chính tốc độ thấp (giảm số lượng truy cập trực tiếp của CPU vào bộ nhớ chính)
  - Thời gian trung bình CPU truy cập hệ thống bộ nhớ gần bằng thời gian truy cập cache
- Giảm giá thành sản xuất:
  - Nếu 2 hệ thống có cùng hiệu năng thì hệ thống có cache sẽ rẻ hơn
  - Nếu 2 hệ thống cùng giá thành, hệ thống có cache sẽ nhanh hơn

Các nguyên lý hoạt động của Cache

- Cache được coi là bộ nhớ thông minh:
  - Cache có khả năng đoán trước yêu cầu về lệnh và dữ liệu 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 bộ nhớ
- Cache hoạt động dựa trên 2 nguyên lý cơ bản:
  - Nguyên lý cục bộ/ lân cận về không gian (spatial locality)
  - Nguyên lý cục bộ/ lân cận về thời gian (temporal locality)

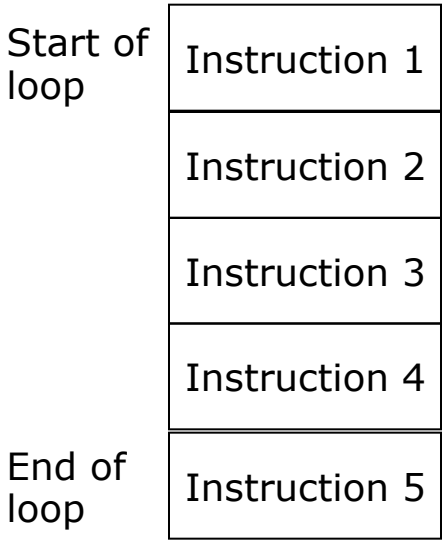
Các nguyên lý hoạt động của Cache

- Cục bộ (lân cận) về không gian:
  - Nếu một vị trí bộ nhớ được truy cập, thì khả năng/ xác suất các vị trí gần đó được truy cập trong thời gian gần tới là cao
  - Áp dụng với các mục dữ liệu và các lệnh có thứ tự tuần tự theo chương trình
  - Hầu hết các lệnh trong chương trình có thứ tự tuần tự, do đó cache đọc một khối lệnh trong bộ nhớ, mà bao gồm cả các phần tử xung quanh vị trí phần tử hiện tại được truy cập



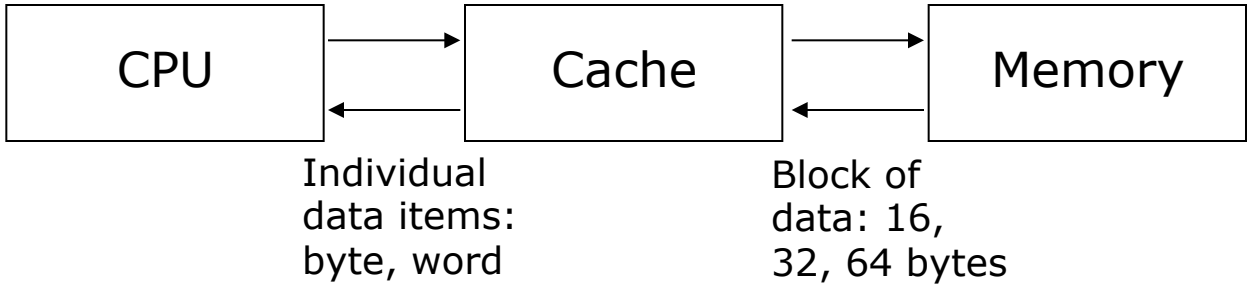
Các nguyên lý cơ bản của Cache

- Cục bộ (lân cận) về thời gian:
  - Nếu một vị trí bộ nhớ được truy cập, thì khả năng nó sẽ được truy cập **lại** trong thời gian gần tới là cao
  - Áp dụng với các mục dữ liệu và các lệnh trong vòng lặp
  - Cache đọc khối dữ liệu trong bộ nhớ bao gồm tất cả các thành phần trong vòng lặp



Trao đổi dữ liệu

- CPU đọc/ ghi từng mục dữ liệu riêng biệt từ/ vào cache
- Cache đọc/ ghi khối dữ liệu từ/ vào bộ nhớ



Hệ số Hit và Miss của Cache

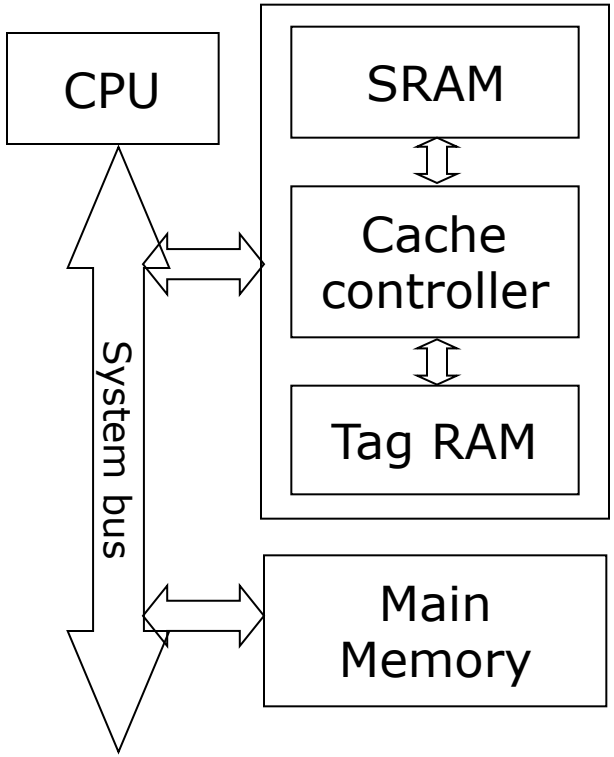
- Hit là sự kiện CPU truy cập tới mục dữ liệu/ mục tin mà tìm được trong cache
  - Xác suất xảy ra Hit được gọi là hệ số hit H
  - 0<= H <= 1
  - H càng cao thì cache càng tốt
- Miss là sự kiện CPU truy cập tới mục dữ liệu mà không tìm thấy nó trong cache
  - Khả năng xảy ra Miss gọi là hệ số miss hay 1-H
  - 0<= (1-H) <= 1
  - Hệ số miss thấp thì hiệu quả cache cao

Kiến trúc Cache – look aside

- Cache và bộ nhớ cùng được kết nối tới bus hệ thống
- Cache và bộ nhớ chính "thấy" chu kỳ bus CPU tại cùng một thời điểm
- Ưu:
  - Thiết kế đơn giản
  - Miss nhanh
- Nhược:
  - Hit chậm

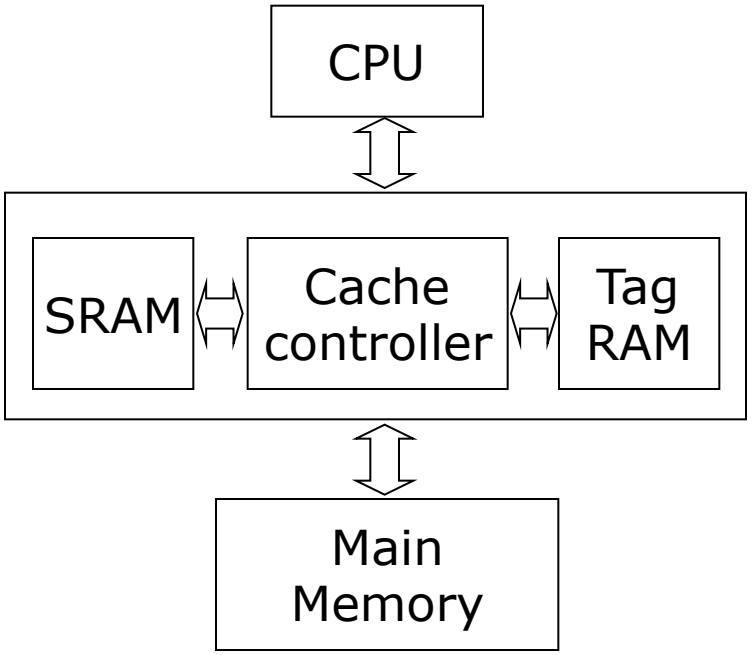
SRAM: RAM lưu dữ liệu cache

Tag RAM: RAM lưu địa chỉ bộ nhớ



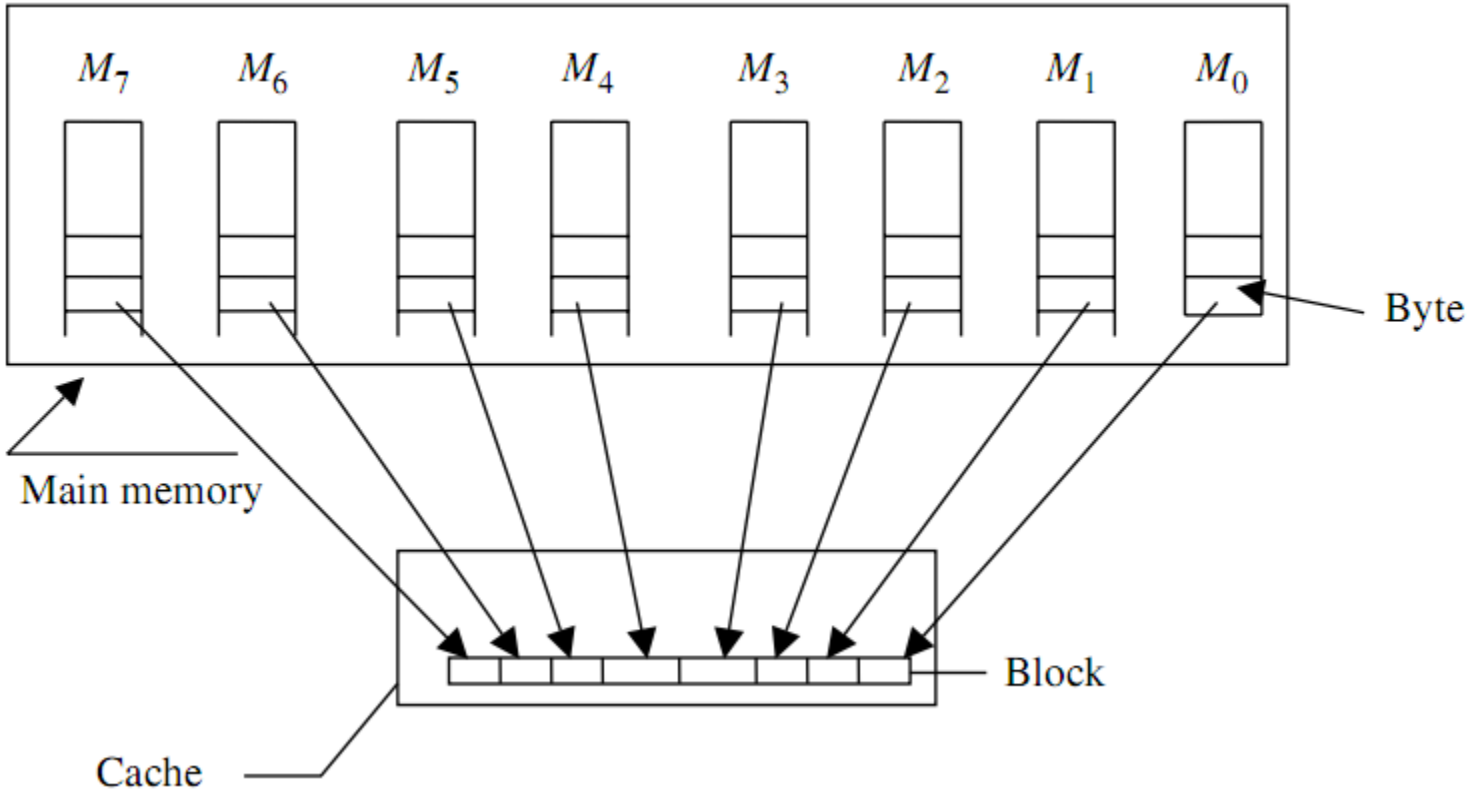
Kiến trúc Cache – look through

- Cache nằm giữa CPU và bộ nhớ chính
- Cache "thấy" chu kỳ bus CPU trước sau đó nó "truyền" lại cho bộ nhớ chính
- Ưu:
  - Hit nhanh
- Nhược:
  - Thiết kế phức tạp
  - Đắt
  - Miss chậm



Tổ chức Cache

- Tổ chức cache giải quyết vấn đề cache và bộ nhớ chính phối hợp làm việc với nhau như thế nào



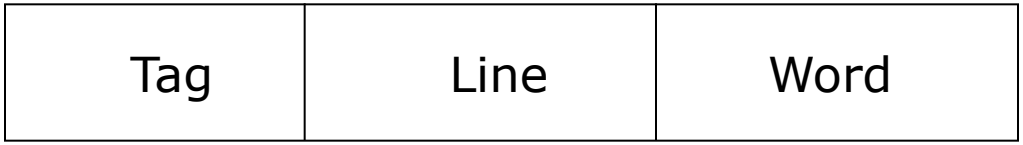
Các kỹ thuật tổ chức Cache

- Ánh xạ trực tiếp (direct mapping):
  - Đơn giản, nhanh
  - Ánh xạ cố định
- Ánh xạ kết hợp đầy đủ (fully associative mapping):
  - Phức tạp, chậm
  - Ánh xạ linh hoạt
- Ánh xạ tập kết hợp/ theo bộ (set): (set associative mapping)
  - Phức tạp
  - Nhanh, ánh xạ linh hoạt

Ánh xạ trực tiếp

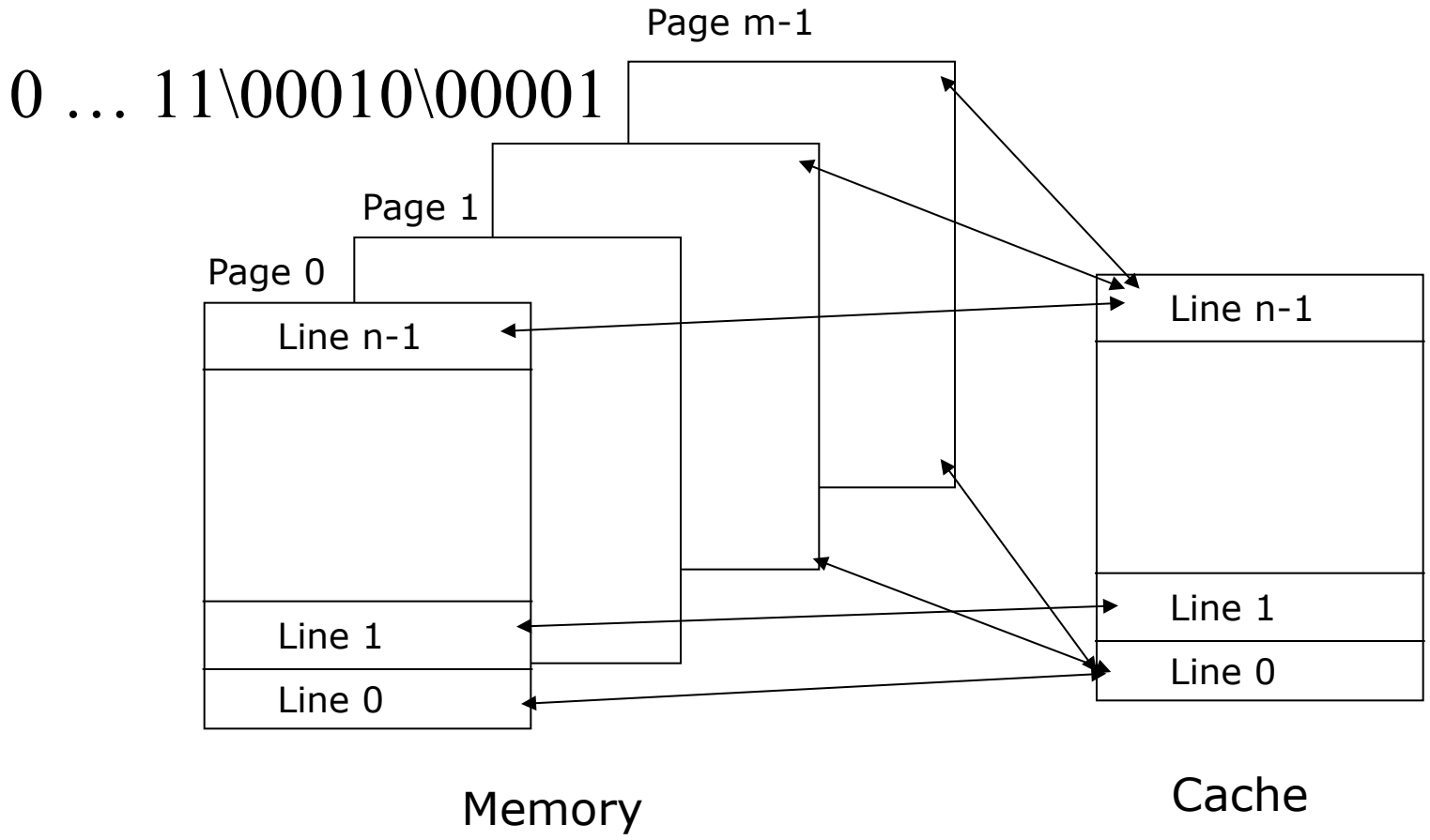
- Cache: Được chia thành n khối hoặc dòng (block or line), từ $Line_0$ tới $Line_{n-1}$
- Bộ nhớ:
  - Được chia thành m trang (page), từ $page_0$ tới $page_{m-1}$
  - Mỗi trang bộ nhớ có kích thước bằng cache
  - Mỗi trang có n lines, từ $Line_0$ tới $Line_{n-1}$
- Ánh xạ:
  - $Line_0$ của ($page_0$ tới $page_{m-1}$) được ánh xạ tới $Line_0$ của cache
  - $Line_1$ của ($page_0$ tới $page_{m-1}$) được ánh xạ tới $Line_1$ của cache
  - ….
  - $Line_{n-1}$ của ($page_0$ tới $page_{m-1}$) được ánh xạ tới $Line_{n-1}$ của cache

Địa chỉ ánh xạ trực tiếp



- *Tag* (bit): là địa chỉ của trang trong bộ nhớ
- *Line* (bit): là địa chỉ của line trong cache
- *Word* (bit): là địa chỉ của word trong line

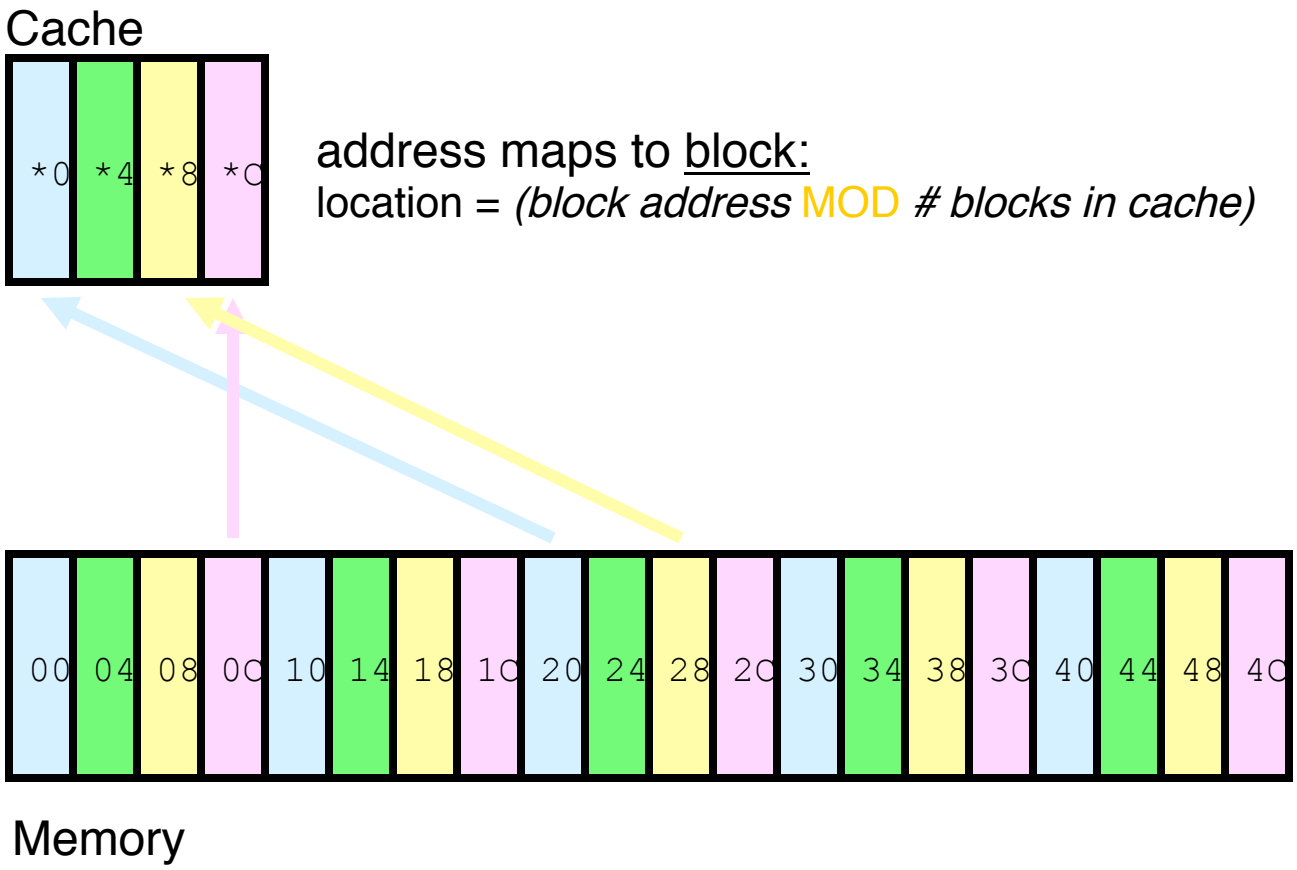
Ánh xạ trực tiếp



Địa chỉ ánh xạ trực tiếp

- Ví dụ: Tìm kích thước trường địa chỉ (32 bit) dùng direct mapping
  - Đầu vào:
    - Kích thước bộ nhớ: 4GB
    - Kích thước cache: 1KB
    - Kích thước line: 32 byte
  - Đầu ra:
    - Kích thước line: 32 byte = 2^5 -> Word = 5 bit
    - Kích thước cache: 1 KB = 2^{10} -> có $2^{10} / 2^5 = 2^5$ lines -> Line = 5 bit
    - Tag = địa chỉ 32 bit – Line – Word = 32 – 5 – 5 = 22 bit

Ví dụ: Direct Mapped block



Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 1: Mapping: 0 modulo 4 = 0

Mem Block	DM Hit/Miss
0	

Block 0

Block 1

Block 2

Block 3

Accessing A Direct-Mapped Cache

DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 1: Mapping: 0 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss

Block 0
Block 1
Block 2
Block 3

Mem[0]

Set 0 is empty: write Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 2: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	

Block 0

Block 1

Block 2

Block 3

Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 2: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss

Block 0
Block 1
Block 2
Block 3

Mem[8]

Set 0 contains Mem[0]. Overwrite with Mem[8]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 3: Mapping: 0 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	

Block 0

Block 1

Block 2

Block 3

Mem[8]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 3: Mapping: 0 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss

Block 0	Mem[0]
Block 1	
Block 2	
Block 3	

Set 0 contains Mem[8]. Overwrite with Mem[0]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 4: Mapping: 6 mod 4 = 2

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	

	Mem[0]
Block 0	
Block 1	
Block 2	
Block 3	

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 4: Mapping: 6 mod 4 = 2

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	miss

Block 0	Mem[0]
Block 1	
Block 2	Mem[6]
Block 3	

Set 2 empty. Write Mem[6]

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 5: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	miss
8	

Block 0	Mem[0]
Block 1	
Block 2	Mem[6]
Block 3	

Accessing A Direct-Mapped Cache

- DM cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

DM Memory Access 5: Mapping: 8 mod 4 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	miss
6	miss
8	miss

Block 0	Mem[8]
Block 1	
Block 2	Mem[6]
Block 3	

Set 0 contains Mem[0]. Overwrite with Mem[8]

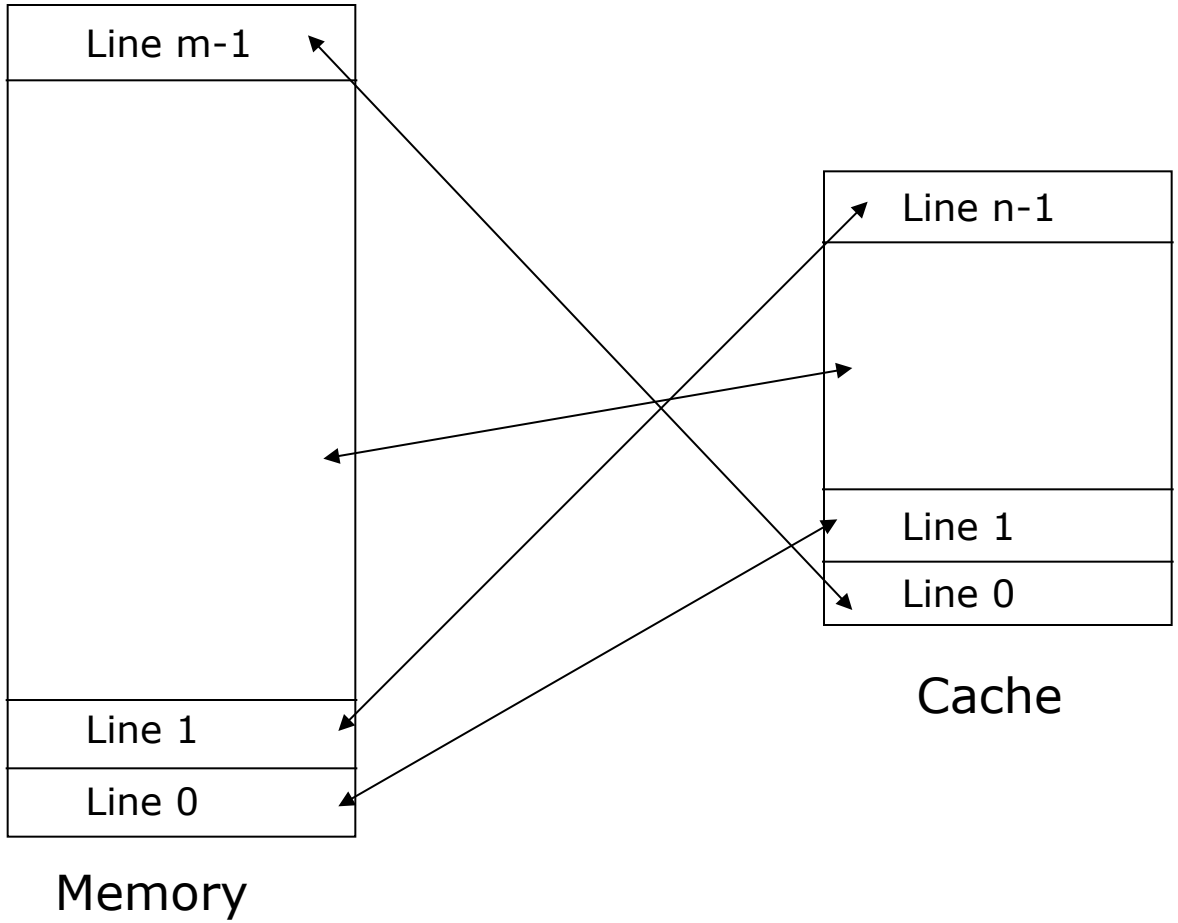
Ánh xạ trực tiếp

- Ưu điểm:
  - Thiết kế đơn giản
  - Nhanh vì ánh xạ cố định: khi biết địa chỉ bộ nhớ có thể tìm nó trong cache rất nhanh
- Nhược điểm:
  - Vì ánh xạ cố định nên khả năng xảy ra xung đột cao
  - Tỷ lệ hit thấp

Ánh xạ kết hợp (liên kết) đầy đủ

- Cache: Được chia thành n khối hoặc dòng (block or line), từ $Line_0$ tới $Line_{n-1}$
- Bộ nhớ:
  - Được chia thành m khối hay dòng, từ $Line_0$ tới $Line_{m-1}$
  - Kích thước mỗi dòng cache bằng kích thước một dòng bộ nhớ
  - Số lượng dòng trong bộ nhớ có thể lớn hơn nhiều số lương dòng của cache (m >>n)
- Ánh xạ:
  - Một dòng của bộ nhớ có thể ánh xạ tới dòng bất kì của cache
  - $Line_i$ của bộ nhớ có thể ánh xạ tới $line_j$ của cache

Ánh xạ kết hợp đầy đủ



- 000…..01\00011

Địa chỉ ánh xạ kết hợp đầy đủ

- *Tag* (bit) là địa chỉ của line trong bộ nhớ (page =1)
- *Word* (bit) là địa chỉ của từ trong line

Địa chỉ ánh xạ kết hợp đầy đủ

- Ví dụ: Tìm kích thước các trường địa chỉ (32 bit)
  - Input:
    - Kích thước cache = 1KB
    - Kích thước line = 32 byte
  - Output:
    - Kích thước Line = 32 byte = 2^5 -> Word = 5 bit
    - Tag = 32 bit – word = 32 – 5 = 27 bit

Ánh xạ kết hợp đầy đủ

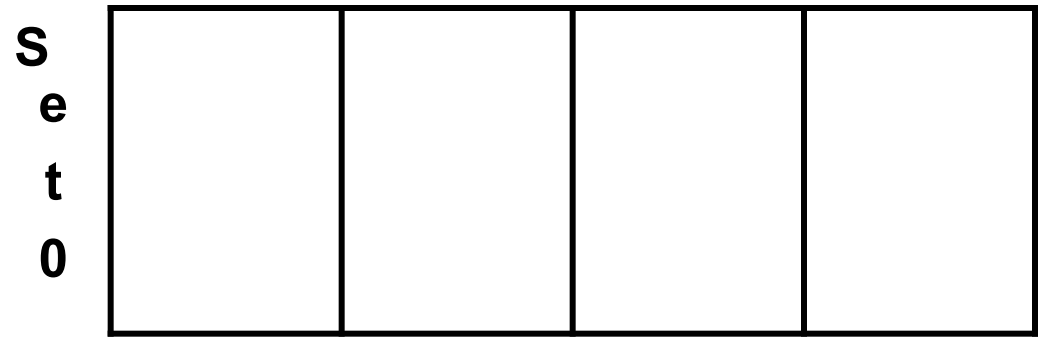
- Ưu:
  - Ít xung đột vì ánh xạ linh hoạt
  - Tỉ lệ hit cao hơn
- Nhược:
  - Chậm vì phải tìm kiếm địa chỉ bộ nhớ trong cache
  - Phức tạp vì có thêm n bộ so sánh địa chỉ trong cache
- Thường sử dụng cho cache có kích thước nhỏ

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 1:

Mem Block	DM Hit/Miss
0	



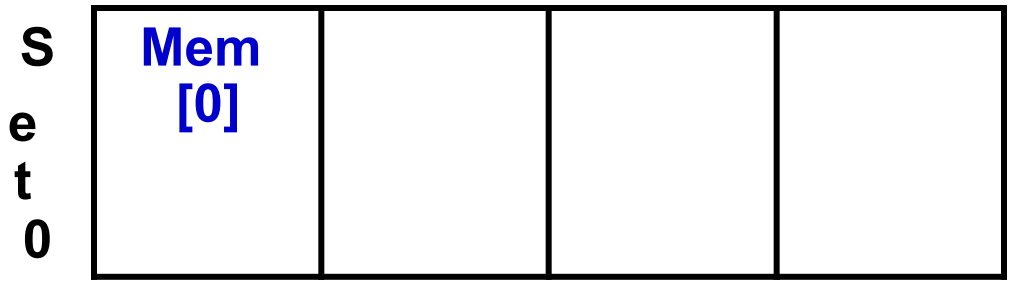
FA Block Replacement Rule: replace least recently used block in set

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 1:

Mem Block	DM Hit/Miss
0	miss



Set 0 is empty: write Mem[0] to Block 0

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 2:

Mem Block	DM Hit/Miss
0	miss
8	

Set 0	Mem [0]			

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 2:

Mem Block	DM Hit/Miss
0	miss
8	miss

Set 0

Mem [0]	Mem [8]		

Blocks 1-3 are LRU: write Mem[8] to Block 1

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 3:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	

Set 0	Mem [0]	Mem [8]		

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 3:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit

Set 0				
	Mem [0]	Mem [8]		

Block 0 contains Mem[0]

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 4:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	

Set 0				
	Mem [0]	Mem [8]		

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 4:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss

Set 0				
	Mem [0]	Mem [8]	Mem [6]	

Blocks 2-3 are LRU : write Mem[6] to Block 2

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 5:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	

Set 0	Mem [0]	Mem [8]	Mem [6]	

Example: Accessing A Fully-Associative Cache

- Fully-Associative cache contains 4 1-word blocks. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

FA Memory Access 5:

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	hit

Set 0				
	Mem [0]	Mem [8]	Mem [6]	

Block 1 contains Mem[8]

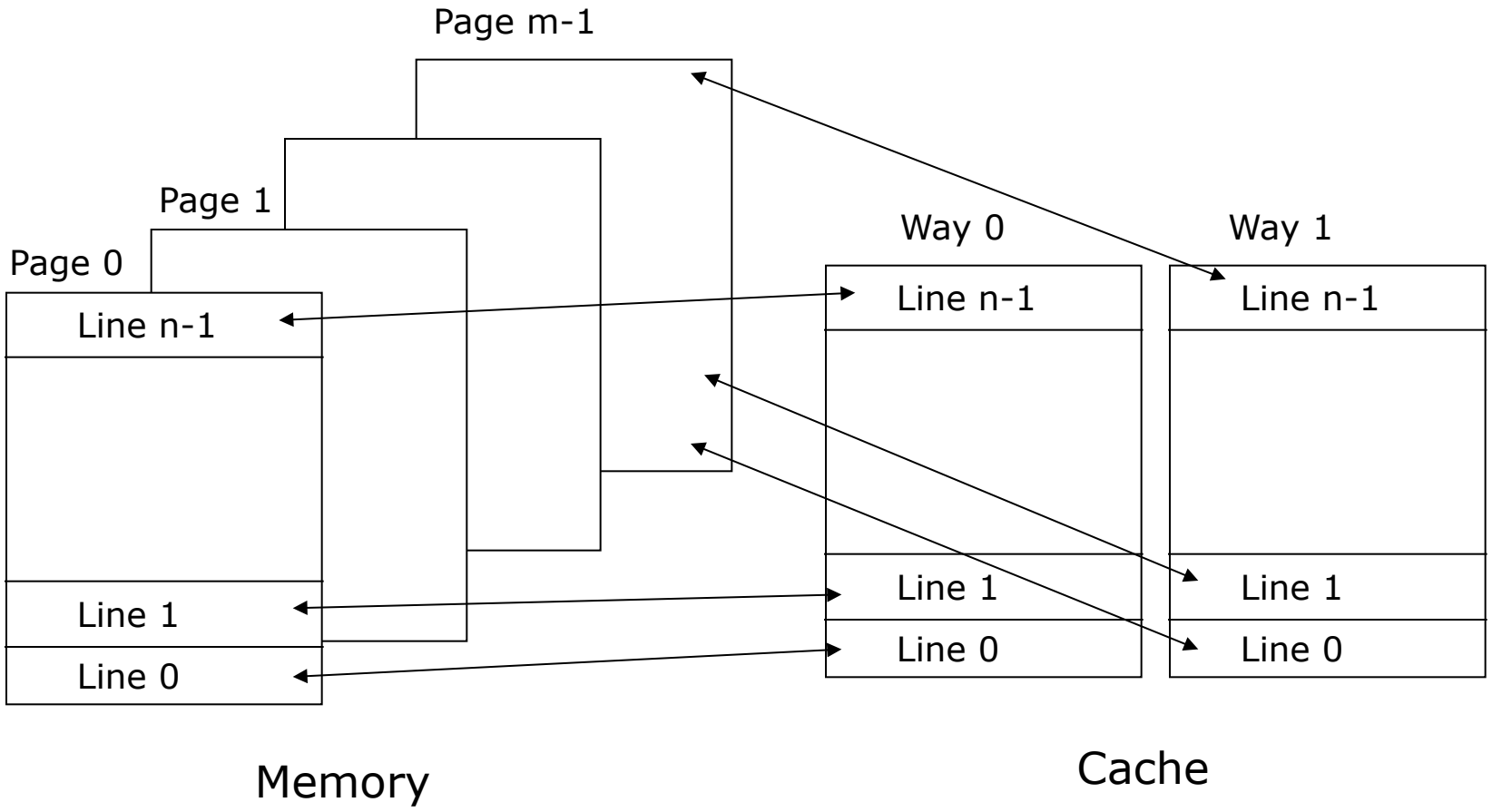
Ánh xạ tập kết hợp (liên kết theo nhóm)

- Cache:
  - Được chia thành k đường (ways) có kích thước bằng nhau
  - Mỗi đường được chia thành n dòng (block or line), từ $Line_0$ tới $Line_{n-1}$
- Bộ nhớ:
  - Được chia thành m trang, từ $page_0$ tới $page_{m-1}$
  - Kích thước trang page bằng kích thước way của cache
  - Mỗi trang có n line, từ $Line_0$ tới $Line_{n-1}$

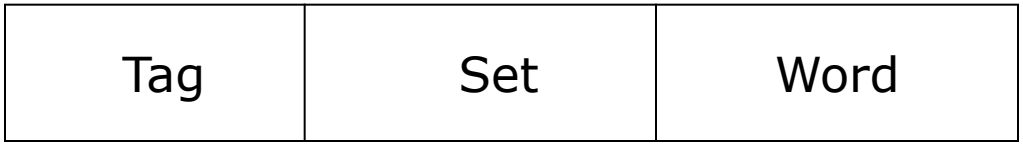
Ánh xạ tập kết hợp

- Ánh xạ:
  - Page được ánh xạ tới Way (ánh xạ linh hoạt):
    - Một page bộ nhớ có thể được ánh xạ tới way bất kì của cache
  - Line của page ánh xạ tới line của way (ánh xạ cố định):
    - $Line_0$ của $page_i$ được ánh xạ tới $Line_0$ của way_j;
    - $Line_1$ của $page_i$ được ánh xạ tới $Line_1$ của way_j;
    - …
    - $Line_{n-1}$ của $page_i$ được ánh xạ tới $Line_{n-1}$ của way_j;

Ánh xạ tập kết hợp



Địa chỉ ánh xạ tập kết hợp



- *Tag* (bit): là địa chỉ của trang trong bộ nhớ
- *Set* (bit): là địa chỉ của line trong way của cache
- *Word* (bit): là địa chỉ của word trong line

Địa chỉ ánh xạ tập kết hợp

- Ví dụ:
  - Input:
    - Kích thước bộ nhớ: 4GB
    - Kích thước cache: 1KB, 2 ways
    - Kích thước Line: 32 byte
  - Output:
    - Line size =32 byte = 2^5 → Word = 5 bit
    - Cache size = 1KB = 2^{10} →có 2^{10} / 2^5 / 2 = 2^4 lines trong 1 way → Set = 4 bit
    - Tag = 32bit địa chỉ – Set – Word = 32 – 4 – 5 = 23 bit.

Ánh xạ tập kết hợp

- Ưu:
  - Nhanh vì ánh xạ trực tiếp được sử dụng cho ánh xạ dòng
  - Ít xung đột vì ánh xạ từ trang nhớ tới đường của cache là linh hoạt
  - Tỷ lệ tìm thấy (hit) cao
- Nhược:
  - Thiết kế và điều khiển phức tạp vì cache được chia thành các way

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 1: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	

Set 0		
Set 1		

SA Block Replacement Rule: replace least recently used block in set

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 1: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss

Set 0	Mem[0]	
Set 1		

Set 0 is empty: write Mem[0] to Block 0

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 2: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	

Set 0	Mem[0]	
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 2: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss

	Mem[0]	Mem[8]
Set 0		
Set 1		

Set 0, Block 1 is LRU: write Mem[8]

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 3: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	

	Mem[0]	Mem[8]
Set 0		
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 3: Mapping: 0 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit

Set 0	Mem[0]	Mem[8]
Set 1		

Set 0, Block 0 contains Mem[0]

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 4: Mapping: 6 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	

	Mem[0]	Mem[8]
Set 0		
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 4: Mapping: 6 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss

	Mem[0]	Mem[6]
Set 0		
Set 1		

Set 0, Block 1 is LRU: overwrite with Mem[6]

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 5: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	

	Mem[0]	Mem[6]
Set 0		
Set 1		

Accessing A Set-Associative Cache

- 2-way Set-Associative cache contains 2 sets, 2 one-word blocks each. Find the # Misses for each cache given this sequence of memory block accesses: 0, 8, 0, 6, 8

SA Memory Access 5: Mapping: 8 mod 2 = 0

Mem Block	DM Hit/Miss
0	miss
8	miss
0	hit
6	miss
8	miss

	Mem[8]	Mem[6]
Set 0		
Set 1		

Set 0, Block 0 is LRU: overwrite with Mem[8]

Đọc/ ghi thông tin trong cache

- Thao tác đọc:
  - Trường hợp tìm thấy (hit case): mục dữ liệu yêu cầu tìm thấy trong cache
    - Mục dữ liệu được đọc từ cache vào CPU
    - Bộ nhớ chính không tham gia
  - Trường hợp không tìm thấy (miss case):
    - Mục dữ liệu được đọc từ bộ nhớ vào cache
    - Sau đó dữ liệu được chuyển từ cache vào CPU
    - ⇒ Miss penalty: thời gian truy cập mục dữ liệu bằng tổng thời gian truy cập cache và bộ nhớ chính

Ghi thông tin trong cache

- Trường hợp hit (bản tin thuộc 1 khối trong cache)
  - Write through (ghi thẳng): mục dữ liệu được ghi vào cache và ghi vào bộ nhớ đồng thời
  - Write back (ghi trễ): mục dữ liệu trước tiên được ghi vào cache và cả dòng (block) chứa nó ở trong cache sẽ được ghi lại vào bộ nhớ sau đó, khi mà dòng đó bị thay thế
- Trường hợp miss (bản tin ko có trong cache)
  - Write allocate (ghi có đọc lại): mục dữ liệu trước hết được ghi vào bộ nhớ sau đó cả dòng chứa nó sẽ được đọc vào cache
  - Write non-allocate (ghi không đọc lại): mục dữ liệu chỉ được ghi vào bộ nhớ

Các chính sách thay thế dòng cache

- Tại 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 cách thức lựa chọn các dòng trong cache để thay thế khi có dòng mới từ bộ nhớ cần chuyển vào
- 3 chiến lược chính:
  - Thay thế ngẫu nhiên (random replacement)
  - Vào trước ra trước FIFO (First In First Out)
  - Thay thế các dòng ít được sử dụng gần đây nhất LRU (Least Recently Used)

Các chính sách thay thế cache

- Thay thế ngẫu nhiên:
  - Các dòng trong cache được chọn ngẫu nhiên để thay
  - Đơn giản
  - Tỷ lệ miss cao vì phương pháp này không xét tới dòng cache nào đang thực sự được sử dụng
    - Nếu một dòng cache đang được sử dụng mà bị thay thế thì sự kiện miss xảy ra và cần phải đọc lại vào cache

Các chính sách thay thế cache

- Thay thế kiểu vào trước ra trước FIFO
  - Dựa trên nguyên lý FIFO
  - Các dòng cache được đọc vào cache trước sẽ được chọn để thay trước
  - Tỷ lệ miss thấp hơn so với phương pháp ngẫu nhiên
  - Tỷ lệ miss vẫn cao vì phương pháp này vẫn chưa thực sự xem xét tới block nào đang thực sự được sử dụng
    - Một dòng cache "cũ" có thể vẫn đang được sử dụng
  - Cài đặt phức tạp vì cần thêm mạch để giám sát thứ tự nạp các dòng bộ nhớ vào cache

Các chính sách thay thế cache

- Thay thế các dòng ít được sử dụng gần đây nhất LRU
  - Các dòng cache ít được sử dụng nhất trong thời gian gần đây sẽ được chọn để thay
  - Xét tới các dòng đang được sử dụng
  - Tỷ lệ miss thấp nhất so với phương pháp ngẫu nhiên và FIFO
  - Cài đặt phức tạp vì cần thêm mạch để giám sát tần suất sử dụng các dòng cache

Hiệu năng cache

- Thời gian truy cập trung bình của hệ thống nhớ có cache:

  Taccess = (hit cost) + (miss cost)

  = (Hit cost) + (miss rate) * (miss penalty)

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

  Trong đó H là hệ số hit

  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

Các yếu tố ảnh hưởng – kích thước cache

- Kích thước cache lớn:
  - Có thể lưu trữ nhiều khối dữ liệu trong bộ nhớ hơn
  - Giảm tần suất trao đổi các khối dữ liệu của chương trình khác nhau giữa bộ nhớ và cache
  - Cache lớn chậm hơn cache nhỏ:
    - Tìm kiếm vị trí bộ nhớ trong không gian lớn hơn
  - Xu hướng: cache càng ngày càng lớn
    - Hỗ trợ đa nhiệm (multi-tasking) tốt hơn
    - Hỗ trợ tốt hơn cho xử lý song song
    - Hỗ trợ tốt hơn cho các hệ thống CPU đa nhân

Các yếu tố ảnh hưởng - Cache nhiều mức

- Nâng cao hiệu năng hệ thống vì cache nhiều mức có thể dung hòa tốc độ của CPU và MEM tốt hơn cache một mức

CPU	L1	L2	L3	Main memory
1ns	5ns	15ns	30ns	60ns
1ns	5ns			60ns

- Thực tế, cache thường có 2 mức: L1 và L2. Một số cache có 3 mức: L1, L2 và L3
- Giảm giá thành hệ thống nhớ

Các yếu tố ảnh hưởng – phân chia cache

- Cache có thể được chia thành cache lệnh và cache dữ liệu để hiệu năng tốt hơn vì:
  - Dữ liệu và lệnh khác nhau về tính cục bộ
    - Dữ liệu có tính chất cục bộ về thời gian hơn
    - Lệnh có tính chất cục bộ về không gian hơn
  - Cache lệnh chỉ hỗ trợ thao tác đọc còn cache dữ liệu hỗ trợ cả 2 thao tác đọc ghi

  ⇒ Dễ tối ưu cache hơn
  - Hỗ trợ nhiều thao tác đọc/ ghi cùng lúc => giảm xung đột tài nguyên
  - Kết hợp các chức năng khác như giải mã trước lệnh vào trong cache lệnh => xử lý lệnh tốt hơn

Các yếu tố ảnh hưởng – phân chia cache

- Trong thực tế, hầu hết cache L1 được chia thành 2 phần:
  - I_cache (Instruction cache): cache lệnh
  - D_cache (data cache): cache dữ liệu.
- Các cache mức cao hơn không được chia
  - Chia cache L1 có lợi ích cao nhất vì nó gần CPU nhất. CPU đọc/ ghi trực tiếp lên cache L1. Cache L1 cần hỗ trợ nhiều lệnh truy nhập đồng thời và các biện pháp tối ưu hóa
  - Chia các cache mức cao hơn không đem lại hiệu quả cao và làm phức tạp trong hệ thống điều khiển cache

Các biện pháp giảm miss

- Cache tốt:
  - Hệ số hit cao
  - Hệ số miss thấp
  - Miss penalty không quá chậm
- Các kiểu miss:
  - Compulsory misses (không tìm thấy bắt buộc): 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 vào cache
  - Capacity misses (không tìm thấy do dung lượng): do kích thước của cache hạn chế, đặc biệt trong môi trường đa nhiệm. Khi cache nhỏ, mã chương trình thường xuyên bị chuyển đổi giữa cache và bộ nhớ
  - Conflict misses (không tìm thấy do xung đột): do có xung đột khi có nhiều dòng nhớ cùng cạnh tranh 1 dòng cache

Các biện pháp giảm miss

- Tăng kích thước dòng cache
  - Giảm compulsory misses: dòng có kích thước lớn sẽ có khả năng bao phủ lân cận tốt hơn
  - Tăng conflict miss:
    - Kích thước dòng lớn => giảm số lượng dòng trong cache => có nhiều dòng bộ nhớ cùng tham chiếu tới dòng cache hơn => tăng conflict miss
  - Kích thước dòng lớn có thể gây lãng phí dung lượng của cache (một số phần của dòng có thể không bao giờ được sử dụng)
  - Kích thước dòng thường dùng là 64 bytes

Các phương pháp giảm miss

- Tăng mức độ liên kết (tăng số lượng cache way):
  - Giảm conflict miss:
    - Tăng số lượng ways => ánh xạ bộ nhớ - cache linh hoạt hơn hay nhiều lựa chọn hơn => giảm conflict miss
  - Làm cache chậm hơn:
    - Tăng số lượng ways => không gian tìm kiếm lớn hơn => làm cache chậm đi

Calculating Bits in Cache

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address?

- How many total bits would be needed for a 4-way set associative cache to store the same amount of data

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8 word blocks, assuming a 32-bit address?

Calculating Bits in Cache

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and one word blocks, assuming a 32-bit address?
  - 64 Kbytes = 16 K words = 2^14 words = 2^14 blocks
  - block size = 4 bytes => offset size = 2 bits,
  - #sets = #blocks = 2^14 => index size = 14 bits
  - tag size = address size - index size - offset size = 32 - 14 - 2 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 32 + 16 + 1 = 49
  - bits in cache = #blocks x bits/block = 2^14 x 49 = 98 Kbytes

Calculating Bits in Cache

- How many total bits would be needed for a 4-way set associative cache to store the same amount of data
  - block size and #blocks does not change
  - #sets = #blocks/4 = (2^14)/4 = 2^12 => index size = 12 bits
  - tag size = address size - index size - offset = 32 - 12 - 2 = 18 bits
  - bits/block = data bits + tag bits + valid bit = 32 + 18 + 1 = 51
  - bits in cache = #blocks x bits/block = 2^14 x 51 = 102 Kbytes
- Increase associativity => increase bits in cache

Calculating Bits in Cache

- How many total bits are needed for a direct- mapped cache with 64 KBytes of data and 8 word blocks, assuming a 32-bit address?
  - 64 Kbytes = 2^14 words = (2^14)/8 = 2^11 blocks
  - block size = 32 bytes => offset size = 5 bits,
  - #sets = #blocks = 2^11 => index size = 11 bits
  - tag size = address size - index size - offset size = 32 - 11 - 5 = 16 bits
  - bits/block = data bits + tag bits + valid bit = 8x32 + 16 + 1 = 273 bits
  - bits in cache = #blocks x bits/block = 2^11 x 273 = 68.25 Kbytes
- Increase block size => decrease bits in cache

Q1: Block Placement

- Where can block be placed in cache?
  - In one predetermined place - direct-mapped
    - Use part of address to calculate block location in cache
    - Compare cache block with tag to check if block present
  - Anywhere in cache - fully associative
    - Compare tag to every block in cache
  - In a limited set of places - set-associative
    - Use portion of address to calculate set (like direct-mapped)
    - Place in any block in the set
    - Compare tag to every block in set
    - Hybrid of direct mapped and fully associative

Q2: Block Replacement

- On a miss, data must be read from memory.
- So, where do we put the new data?
  - Direct-mapped cache: must place in fixed location
  - Set-associative, fully-associative - can pick within set

Replacement Algorithms

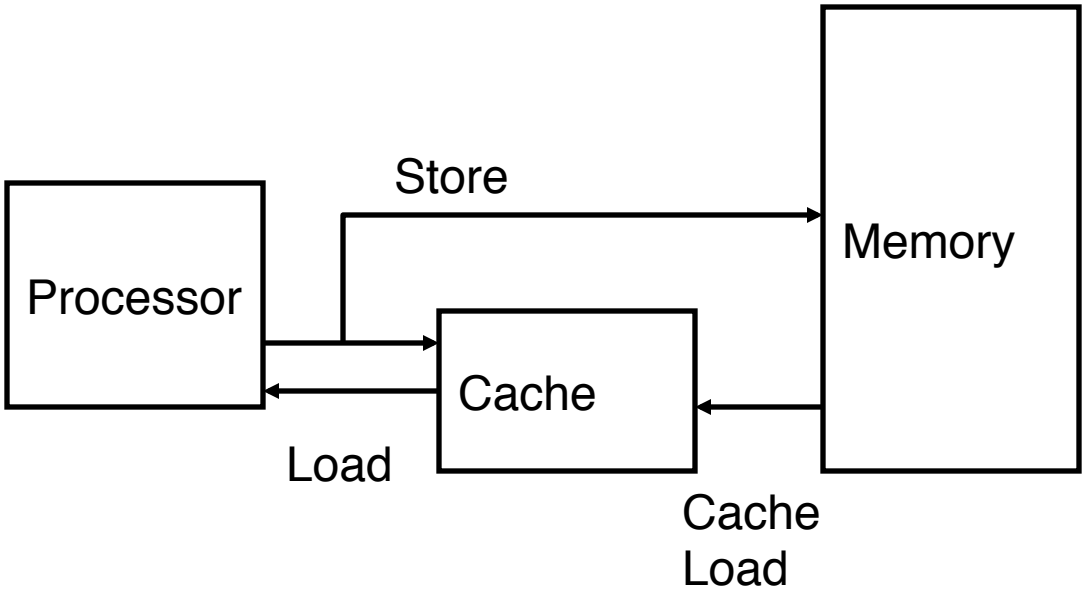
- *When a block is fetched, which block in the target set should be replaced?*
- Optimal algorithm:
  - replace the block that will not be used for the longest time (must know the future)
- Usage based algorithms:
  - Least recently used (LRU)
    - replace the block that has been referenced least recently
    - hard to implement
- Non-usage based algorithms:
  - First-in First-out (FIFO)
    - treat the set as a circular queue, replace head of queue.
    - easy to implement
  - Random (RAND)
    - replace a random block in the set
    - even easier to implement

Q4: Write Strategy

- What happens on a write?
  - Write through - write to memory, stall processor until done
  - Write buffer - place in buffer (allows pipeline to continue*)
  - Write back - delay write to memory until block is replaced in cache

Write Through

- Store by processor updates cache *and* memory
- Memory always consistent with cache
- ~2X more loads than stores
- WT always combined with write buffers so that don't wait for lower level memory



Write Back

- Store by processor only updates cache line
- Modified line written to memory only when it is evicted
  - Requires "dirty bit" for each line
    - Set when line in cache is modified
    - Indicates that line in memory is not updated
- Memory not always consistent with cache
- No writes of repeated writes