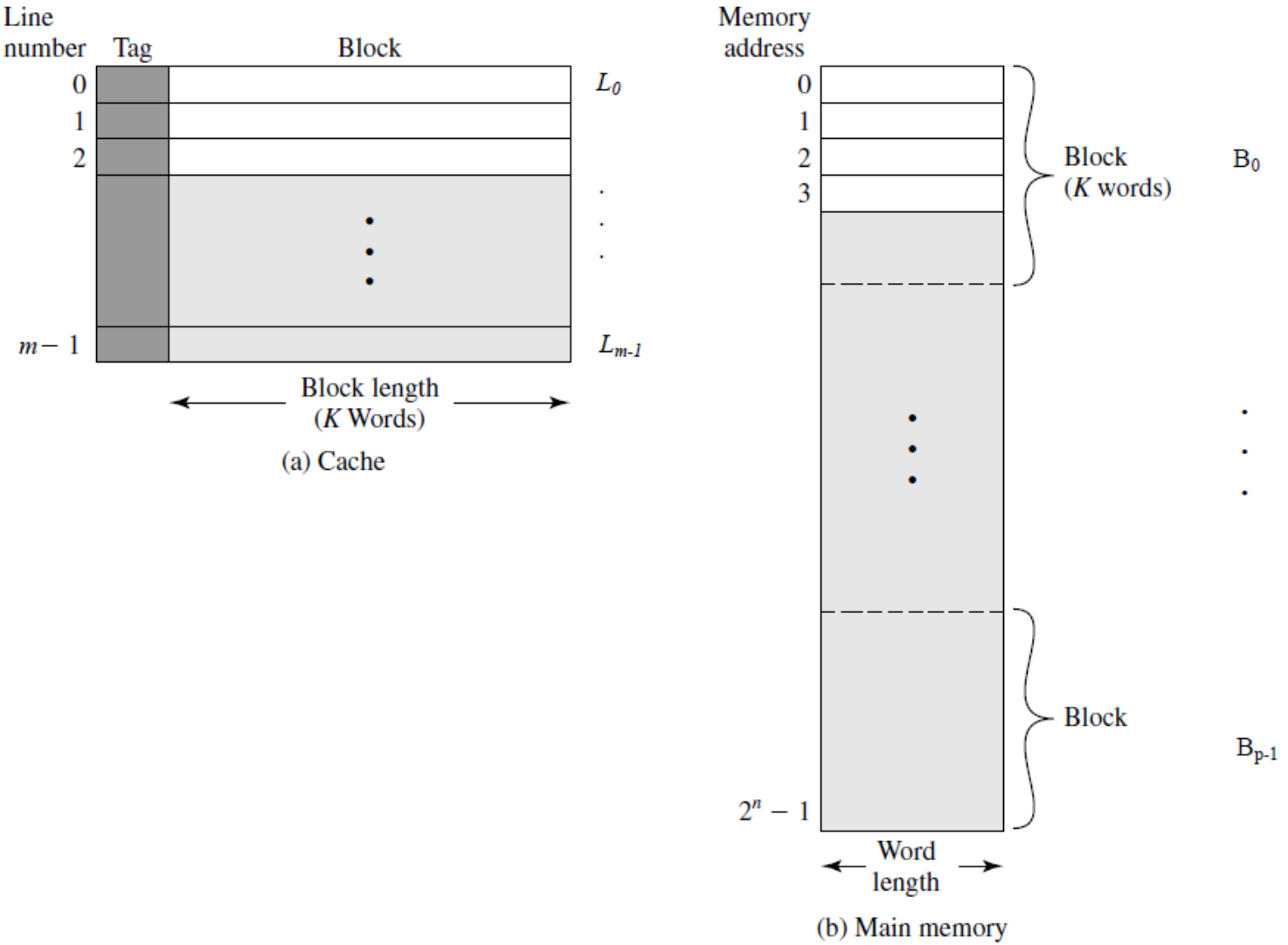
Chương 10. BỘ NHỚ CACHE

TS. Phạm Công Thắng
Bộ môn hệ thống nhúng
Khoa Công Nghệ Thông Tin
Trường Đại học Bách Khoa
Đại học Đà Nẵng

Thao tác của cache

- CPU yêu cầu nội dung của ngăn nhớ
- CPU kiểm tra trên cache với dữ liệu này
  - Nếu có, CPU nhận dữ liệu từ cache (nhanh)
  - Nếu không có, đọc Block nhớ chứa dữ liệu từ bộ nhớ chính vào cache
- Tiếp đó chuyển dữ liệu từ cache vào CPU

Cấu trúc chung của cache / Bộ nhớ chính



Cấu trúc chung của cache / Bộ nhớ chính

- Bộ nhớ chính có 2^N byte nhớ
- Bộ nhớ chính và cache đều được chia thanh các khối có kích thước bằng nhau
  - Bộ nhớ chính: $B_0, B_1, B_2, \ldots, B_{(p-1)}$ (p Blocks)
  - Bộ nhớ cache: $L_0, L_1, L_2, \ldots, L_{(m-1)}$ (m Lines)
  - Kích thước của Block = 8, 16, 32 ,64 hoặc 128 byte

Cấu trúc chung của cache / Bộ nhớ chính

- Một số Block của bộ nhớ chính được nạp vào các Line của cache.
- Nội dung Tag (thẻ nhớ) cho biết Block nào của bộ nhớ chính hiện đang được chứa ở Line đó.
- Khi CPU truy nhập (đọc/ghi) một từ nhớ, có hai khả năng xảy ra:
  - Từ nhớ đó có trong cache (cache hit)
  - Từ nhớ đó không có trong cache (cache miss)

Các phương pháp ánh xạ

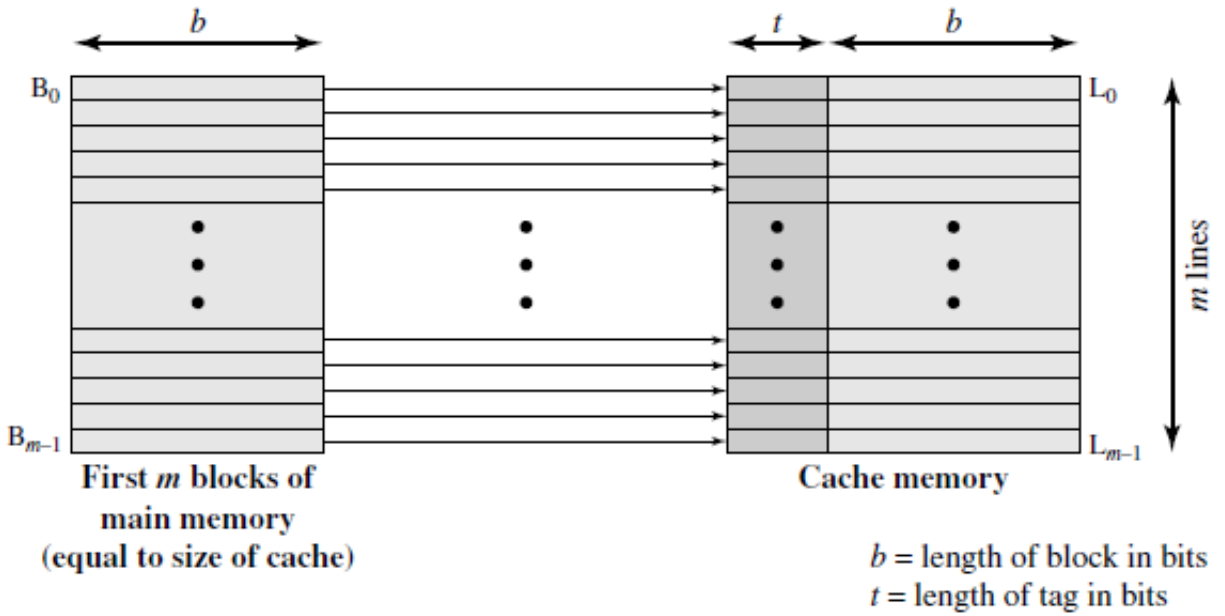
- Chính là các phương pháp tổ chức bộ nhớ cache
  - Ánh xạ trực tiếp (Direct mapping)
  - Ánh xạ liên kết toàn phần (Fully associative mapping)
  - Ánh xạ liên kết tập hợp (Set associative mapping)

Ánh xạ trực tiếp (Direct mapping)

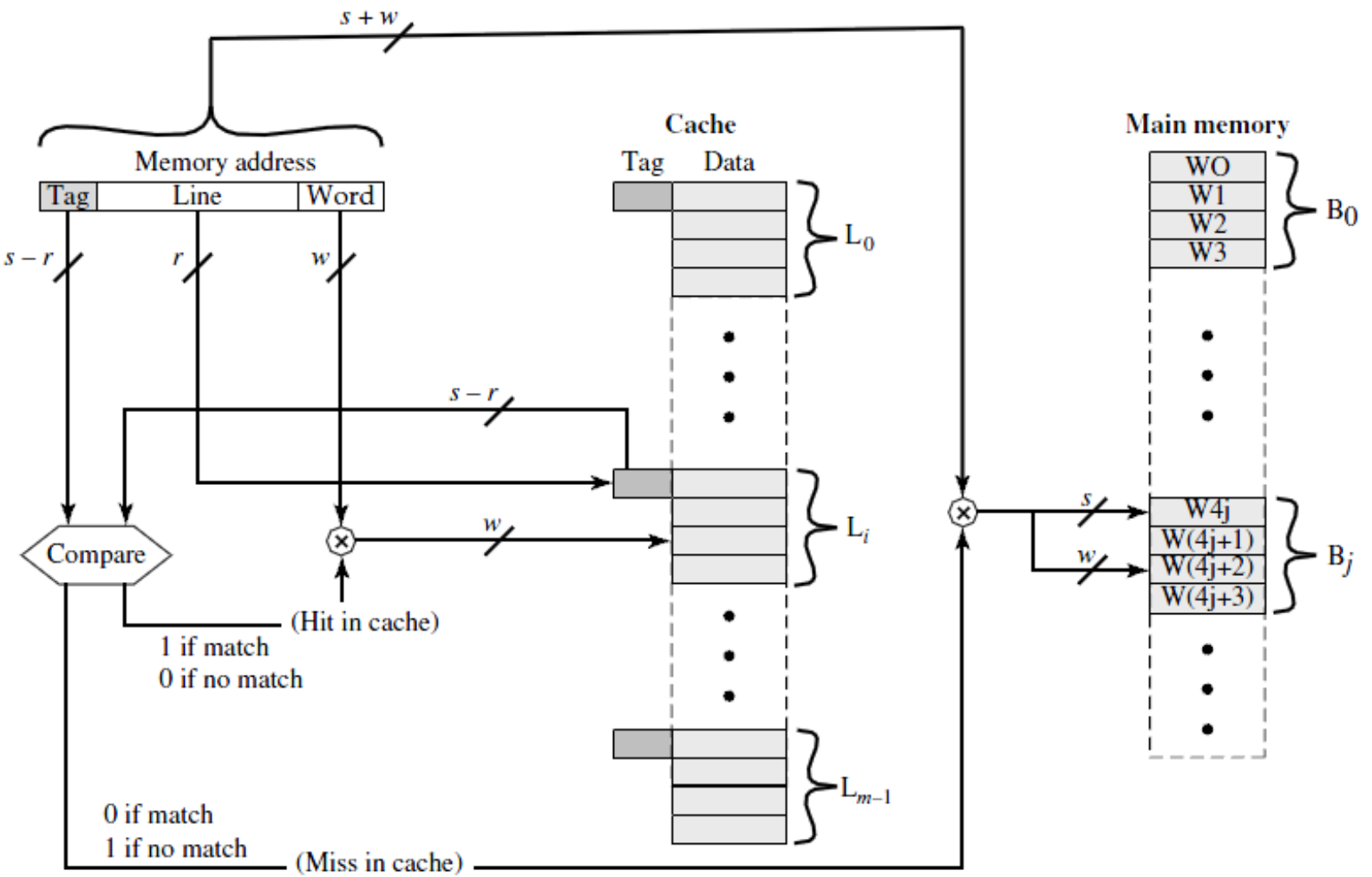
- Mỗi Block của bộ nhớ chính chỉ có thể được nạp vào một Line của cache:
  - $B_0 \rightarrow L_0$
  - …
  - $B_{(m-1)} \rightarrow L_{(m-1)}$
  - …
  - $B_m \rightarrow L_0$
  - …

Ánh xạ trực tiếp (Direct mapping)

- Tổng quát:
  - Mỗi khối (block) bộ nhớ chính B_j chỉ có thể được ánh xạ vào một phần tử cache $L_{(j \bmod n)}$
  - m là số dòng (Line) của cache



Ánh xạ trực tiếp (Direct mapping)



Ánh xạ trực tiếp (Direct mapping)

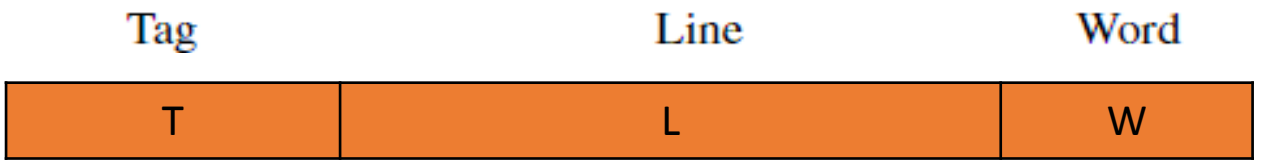
- Tín hiệu địa chỉ truy xuất bộ nhớ sẽ có cấu trúc như sau
  - Mỗi một địa chỉ N bits của bộ nhớ chính
    - Word gồm w bit xác định một từ nhớ trong Block hay Line:

$$2^w = \text{Block size} = \text{Line size}$$

    - Line gồm L bit xác định một trong số các Line trong cache:

$$2^L = \text{Number of lines in cache} = m$$

    - Trường Tag gồm T bit: T = N-(W+L)



Ánh xạ trực tiếp (Direct mapping)

- Ví dụ
  - Input: Kích thước bộ nhớ chính : 4GB
    - Kích thước Cache : 256 KB
    - Kích thước Line = 32 byte = 2^5 byte
  - Output:
    - Kích thước bộ nhớ chính : 4GB = 2^{32} byte => N =log2(2^{32})= 32 bit
    - Kích thước Cache : 256 KB = 2^{18} byte.
    - Kích thước Line = 32 byte = 2^5 byte => W = 5 bit

$\Rightarrow$Số Line trong cache = 2^{18}/ 2^5 = 2^{13} Line => L = 13 bit

$\Rightarrow$Số bit của trường Tag T = 32 - (13 + 5) = 14 bit

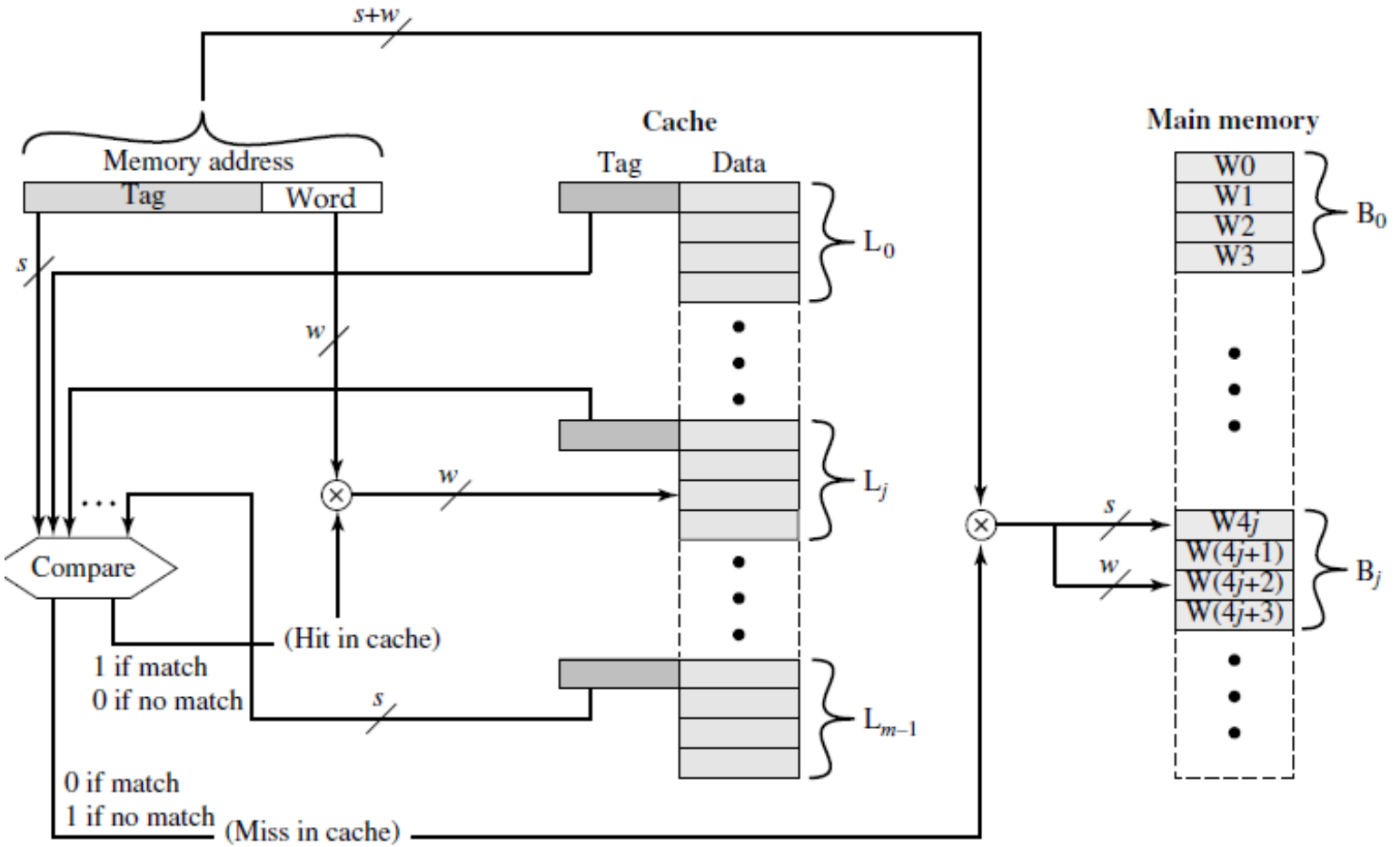
Ánh xạ trực tiếp (Direct mapping)

- Đánh giá Direct mapping
  - Đơn giản
  - Chi phí tổ chức thấp
  - Xác suất cache hit thấp:
  - Mỗi block phải cố định tại một cache line
  - Nếu 1 block (ví dụ block L) cần được nạp vào line (line 0) mà line này đang chứa một block khác (ví dụ block 2L) thì phải thay block 2L ra để giành chỗ cho block L, mặc dù có thể cache còn rất nhiều line trống

Ánh xạ liên kết toàn phần (Fully associative mapping)

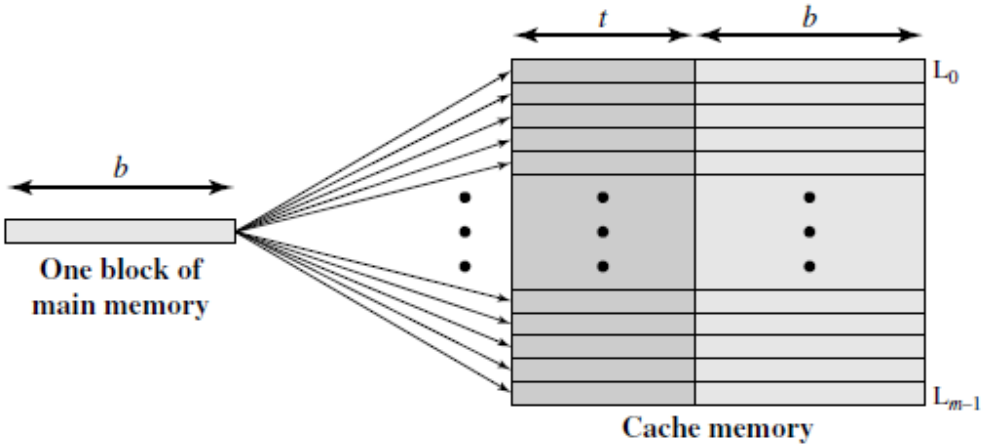
- Mỗi Block có thể nạp vào bất kỳ Line nào của cache.
- Địa chỉ của bộ nhớ chính bao gồm hai trường:
  - Trường Word giống như trường hợp ở trên (w bit)
  - Trường Tag dùng để xác định Block của bộ nhớ chinh (T bit).
- Tag xác định Block đang nằm ở Line đó
- Do 1 block có thể được nạp vào bất cứ line nào nên để xác định block đã nằm trong line hay chưa thì phải duyệt tất cả các line

Ánh xạ liên kết toàn phần (Fully associative mapping)



Ánh xạ liên kết toàn phần (Fully associative mapping)

- Để xác định chính xác liệu một khối nằm trong cache hay không, cần so sánh đồng thời với tất cả các Tag do đó mất nhiều thời gian (Số bit làm Tag phải lưu trong cache nhiều hơn)
- Xác suất cache hit cao hơn ánh xạ trực tiếp
- Bộ so sánh phức tạp.



Ánh xạ liên kết toàn phần (Fully associative mapping)

- Ví dụ
  - Input: Kích thước bộ nhớ chính : 4GB
    - Kích thước Cache : 1MB
    - Kích thước Line = 32 byte
  - Output:
    - Kích thước bộ nhớ chính : 4GB = 2^{32} byte => N = 32 bit
    - Kích thước Line = 32 byte = 2^5 byte => W = 5 bit

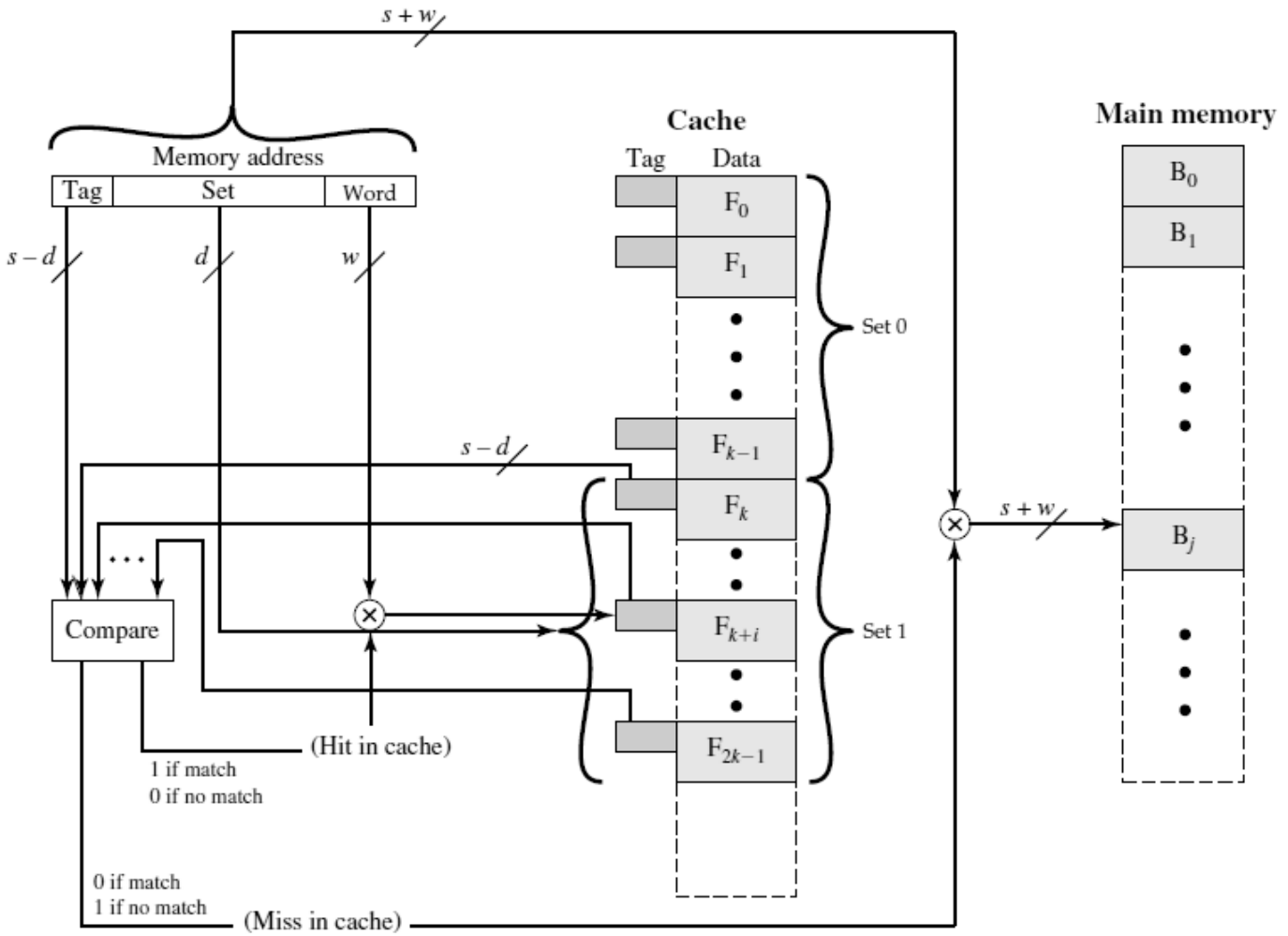
$\Rightarrow$Số bit của trường Tag: T = 32 - 5 = 27 bit

Tag 27 bit	W 5 bit

Ánh xạ liên kết tập hợp (Set associative mapping)

- Là phương pháp dung hòa của 2 phương pháp trên
- Cache đươc chia thành các Tập (Set)
  - Mỗi một Set chứa một số Line
    - Ví dụ: 4 Line/Set 4 -way associative mapping
    - Thông thường 2,4,8 hoặc 16 Lines/Set
  - Ánh xạ theo nguyên tắc sau:
  - $B_0 \rightarrow S_0$
  - $B_1 \rightarrow S_1$
  - …
  - $B_{(k-1)} \rightarrow S_{(k-1)}$
  - $B_k \rightarrow S_0$

Ánh xạ liên kết tập hợp (Set associative mapping)

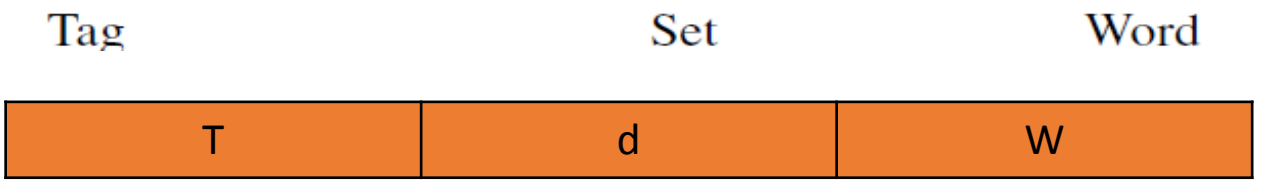


Ánh xạ liên kết tập hợp (Set associative mapping)

- Mỗi block B_j được ánh xạ (direct-mapping) vào duy nhất một tập line S_i theo nguyên tắc:

$$S_i = B_j \bmod S$$

- Các line trong một tập được quản lý theo kiểu fully associate
  - Kích thước Block = 2^w Word
  - Trường Set có d bit dùng để xác định số : 2^d Set
  - Trường Tag có T bit: T = N-(w+d)



Ánh xạ liên kết tập hợp (Set associative mapping)

- Ví dụ
  - Input: Kích thước bộ nhớ chính : 4GB
    - Kích thước Cache : 32 Kb
    - Kích thước Line = 64 byte
    - 4 Line/Set 4 -way associative mapping
  - Output:
    - Kích thước bộ nhớ chính : 4GB = 2^{32} byte => N = log2(2^{32}) = 32 bits
    - Kích thước Line = 64 byte = 2^6 byte => W = 6 bits
    - Số lượng lines: 32 Kb / 64 =512
    - Số lượng các tập hợp: 512 / 4 = 128= 2^7 => d=7 bits

$\Rightarrow$Số bit của trường Tag: T = 32 – (6+7) = 19 bits

Tag 19 bits	d 7 bits	W 6 bits

Ánh xạ liên kết tập hợp (Set associative mapping)

- Thiết kế và điều khiển phức tạp vì cache được chia thành các way
- Nhanh vì ánh xạ trực tiếp được sử dụng cho ánh xạ dòng
- It xung đột vì ánh xạ từ bộ nhớ tới đường cache là linh hoạt
- Tỷ lệ hit cao

Hiệu quả của cache

- Thông thường người ta dùng thời gian thâm nhập trung bình bộ nhớ trong để đánh giá hiệu quả của cache

$$t_{access} = t_{hit} + (miss_{rate}) * (t_{miss\_penalty})$$

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

Với H là tỉ lệ (hệ số) hit

Hiệu quả của cache

- Ví dụ:
  - t_{hit} =5 , t_{memory}=60 , =80%
    - t_{access} =5+(1−0.8)∗(60)=5+12=17
  - t_{hit} =5 , t_{memory} =60 , =95%
    - t_{access} =5+(1−0.95)∗(60)=5+3=8

Hiệu quả của cache

Hiệu quả của cache

- Giảm Miss Penalty với Multilevel Cache
  - Nếu miss xảy ra với cache L1 thì sẽ truy cập đến cahe L2
  - Nếu dữ liệu được tìm thấy ở cache L2 thì miss penalty là thời gian truy cập cache L2 (nhỏ hơn nhiều so với thời gian truy cập bộ nhớ chính DRAM)
  - Nếu miss xảy ra với cache L2 thì phải truy cập bộ nhớ chính và sẽ xuất hiện miss penalty lớn hơn.

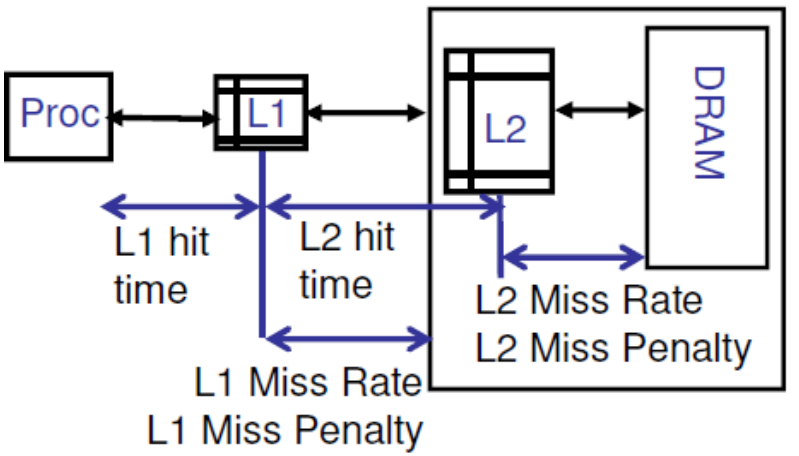
Giảm Miss Penalty với Multilevel Cache

$$t_{access} = t_{hit} + miss_{rate} * t_{miss\_penalty}$$

$$t_{access} = t_{hit\_L1} + miss_{rate\_L1} * t_{miss\_penalty\_L1}$$

$$t_{miss\_penalty\_L1} = t_{hit\_L2} + miss_{rate\_L2} * t_{miss\_penalty\_L2}$$

$$t_{access} = t_{hit\_L1} + miss_{rate\_L1} * (t_{hit\_L2} + miss_{rate\_L2} * t_{miss\_penalty\_L2})$$



Giảm Miss Penalty với Multilevel Cache

- Ta có:
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15%
  - L1, L2 Miss Penalty = 200 cycles
    - **Không dùng cache L2**
      - t_{access} = 1 + 0.05 x 200 = 11 cycles
    - **Dùng cache L2**
      - $t_{miss\_penalty\_L1}$ = 5 + 0.15 * 200 = 35
      - t_{access} = 1 + 0.05 x 35 = 2.75 cycles
    - ***Dùng L2 nhanh hơn gấp 4 lần không dùng L2***

Đồng bộ hóa cache và bộ nhớ chính

- CPU có thể thay đổi line/block . IO có thể truy xuất trực tiếp bộ nhớ chính?
- Nếu nội dung line/block bị thay đổi trong cache/bộ nhớ chính, khi nào sẽ thực hiện cập nhật thay đổi này lên block/line tương ứng trong bộ nhớ chính/cache ?

Chiến lược thay thế

- Vấn đề
  - Khi cần chuyển một block mới vào trong cache mà không tìm được line trống theo yêu cầu của kiểu tổ chức cache thì phải thay line nào ra ?

Chiến lược thay thế

- Direct Mapping
  - Không có lựa chọn, Mỗi Block chỉ ánh xạ vào một Line xác định do đó thay thế Block ở Line đó
- Fully Associate Mapping và Set Associate Mapping
  - Random: Thay thế ngẫu nhiên
  - FIFO (First In First Out): Thay thế Block nào nằm lâu nhất ở trong Set đó
  - LFU (Least Frequently Used):Thay thế Block nào trong Set có số lần truy nhập ít nhất trong cùng một khoảng thời gian
  - LRU (Least Recently Used):Thay thế Block ở trong Set tương ứng có thời gian lâu nhất không được tham chiếu tới.(tối ưu nhất)

Các phương pháp đọc ghi cache

- Quá trình trao đổi thông tin giữa CPU – cache và giữa cache – bộ nhớ chính là một trong các vấn đề có ảnh hưởng lớn đến hiệu năng cache.
- Cần có chính sách trao đổi hay đọc ghi thông tin giữa các thành phần này như thế nào để đạt được hệ số hit cao nhất và giảm thiểu miss.

Các phương pháp đọc ghi cache

•Khi đọc thông tin với trường hợp hit (mẩu tin cần đọc có trong cache):

- *Thông tin được đọc từ cache vào CPU và bộ nhớ chính không tham gia.*
- *Do vậy thời gian CPU truy nhập thông tin bằng thời gian CPU truy nhập cache.*

•Khi đọc thông tin với trường hợp miss (mẩu tin cần đọc không có trong cache):

- *Thông tin trước hết được chuyển từ bộ nhớ chính vào cache, sau đó nó được đọc từ cache vào CPU.*
- *Thời gian CPU truy cập thông tin bằng thời gian truy nhập cache cộng với thời gian cache truy nhập bộ nhớ chính – còn gọi là miss penalty.*

Các phương pháp đọc ghi cache

•Trường hợp ghi thông tin và nếu đó là trường hợp hit, có thể áp dụng một trong 2 chính sách ghi:

- ghi thẳng (write through)
- ghi trễ (write back).

•Trường hợp ghi thông tin và nếu đó là trường hợp miss, có thể áp dụng một trong 2 chính sách ghi:

- ghi có đọc lại (write allocate / fetch on write)
- ghi không đọc lại (write non-allocate)

Các phương pháp đọc ghi cache

- Write through (ghi thẳng)
  - Khi một line/block bị thay đổi trong cache/bộ nhớ chính bởi CPU/IO, block/line tương ứng trong bộ nhớ chính/cache sẽ lập tức được cập nhật
  - Cách ghi này làm chậm tốc độ chung của hệ thống.
  - I/O có thể truy cập bộ nhớ trực tiếp
- Write back (ghi trả sau hay trễ)
  - Khi một line/block bị thay đổi trong cache, sử dụng bit đánh dấu (dirty bit)
  - Khi một line/block bị thay thế, khối này sẽ được ghi lại vào bộ nhớ trong chỉ khi bit trạng thái đã được thiết lập
  - I/O phải truy xuất bộ nhớ chính thông qua cache
  - Tốc độ nhanh

Các phương pháp đọc ghi cache

- **Ghi có đọc lại (write allocate / fetch on write)**
  - Thông tin trước hết được ghi ra bộ nhớ chính, và sau đó dòng nhớ chứa thông tin vừa ghi được đọc vào cache.
  - Việc đọc lại thông tin vừa ghi từ bộ nhớ chính vào cache có thể giúp giảm miss đọc kế tiếp theo nguyên lý lân cận theo thời gian: thông tin vừa được truy nhập có thể được truy nhập lại trong thời gian gần
- **Ghi không đọc lại (write non-allocate)**
  - Thông tin chỉ được ghi ra bộ nhớ chính.
  - Không có thao tác đọc dòng nhớ chứa thông tin vừa ghi vào cache.

Các yếu tố ảnh hưởng đến hiệu năng cache

- Kích thước cache
- Các loại cache
- Tạo cache nhiều mức

Các yếu tố ảnh hưởng đến hiệu năng cache

- **Kích thước cache:**
  - Với kích thước lớn, có thể tăng được số dòng bộ nhớ lưu trong cache, do vậy giảm tần suất tráo đổi các dòng cache của các chương trình khác nhau với bộ nhớ chính
  - Cache lớn hỗ trợ đa nhiệm, xử lý song song và các hệ thống CPU nhiều nhân tốt hơn do không gian cache lớn có khả năng chứa đồng thời thông tin của nhiều chương trình
  - Nhược điểm của cache lớn là chậm, do có không gian tìm kiếm lớn hơn cache nhỏ

Các loại cache

- Thường sử dụng 2 loại cache
  - **Unified cache**: một cache cho cả lệnh và dữ liệu => có tranh chấp khi một lệnh muốn thâm nhập một số liệu trong cùng một chu kỳ của giai đoạn đọc một lệnh khác
  - **Split cache**: cache riêng cho lệnh (instruction cache) và dữ liệu (data cache)

    $\Rightarrow$ tránh các khó khăn do kiến trúc khi thi hành các lệnh, giúp tối ưu hoá mỗi loại cache về mặt kích thước tổng quát, kích thước các khối và độ phối hợp các khối.

    $\Rightarrow$ Cache lệnh chỉ cần hỗ trợ thao tác đọc; cache dữ liệu cần hỗ trợ cả 2 thao tác đọc và ghi

    $\Rightarrow$hỗ trợ nhiều lệnh truy nhập đồng thời hệ thống nhớ, nhờ vậy giảm xung đột tài nguyên cho CPU pipeline.

Các mức cache

- Khi cache được chia thành nhiều mức với kích thước tăng dần và tốc độ truy nhập giảm dần:
  - 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 của bộ nhớ chính.

Các mức cache

- Cache mức một (L1 cache): thường là cache trong (on-chip cache; nằm bên trong CPU)
  - Kích thước 10s KB
  - Hit time: 1 chu kỳ
  - Miss rate: 1-5%
- Cache mức hai (L2 cache) thường là cache ngoài (off-chip cache; cache này nằm bên ngoài CPU).
  - Kích thước 100s KB
  - Hit time: 1 chu kỳ
  - miss rates: 10-20%
- Ngoài ra, trong một số hệ thống còn có tổ chức cache mức ba (L3 cache), đây là mức cache trung gian giữa cache L2 và một thẻ bộ nhớ.

Kích thước cache của một số hệ thống

Bộ xử lý	Kiểu	Năm phát hành	L1 Cache [a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 KB	-	-
PDP-11/70	Mini Computer	1975	1 KB	-	-
VAX 11/780	Mini Computer	1978	16 KB	-	-
IBM 3033	Mainframe	1978	64 KB	-	-
IBM 3090	Mainframe	1985	128 to 256 KB	-	-
Intel 80486	PC	1989	8 KB	-	-
Pentium	PC	1993	8 KB / 8 KB	256 to 512 KB	-
PowerPC 601	PC	1993	32 KB	-	-
PowerPC 620	PC	1996	32 KB / 32 KB	-	-
PowerPC G4	PC/Server	1999	32 KB / 32 KB	256KB to 1MB	2 MB
IBM S390/G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S390/G6	Mainframe	1999	256 KB	8 MB	-
Pentium 4	PC/Server	2000	8 KB / 8 KB	256 KB	-
IBM SP	High-End server/ Super Computer	2000	64 KB / 32 KB	8 MB	-
CRAY MTA [b]	Super Computer	2000	8 KB	2 MB	-
Itanium	PC/Server	2001	16 KB / 16 KB	96 KB	2 MB
SGI Origin 2001	High-End server	2001	32 KB / 32 KB	4 MB	-

Hết chương 4