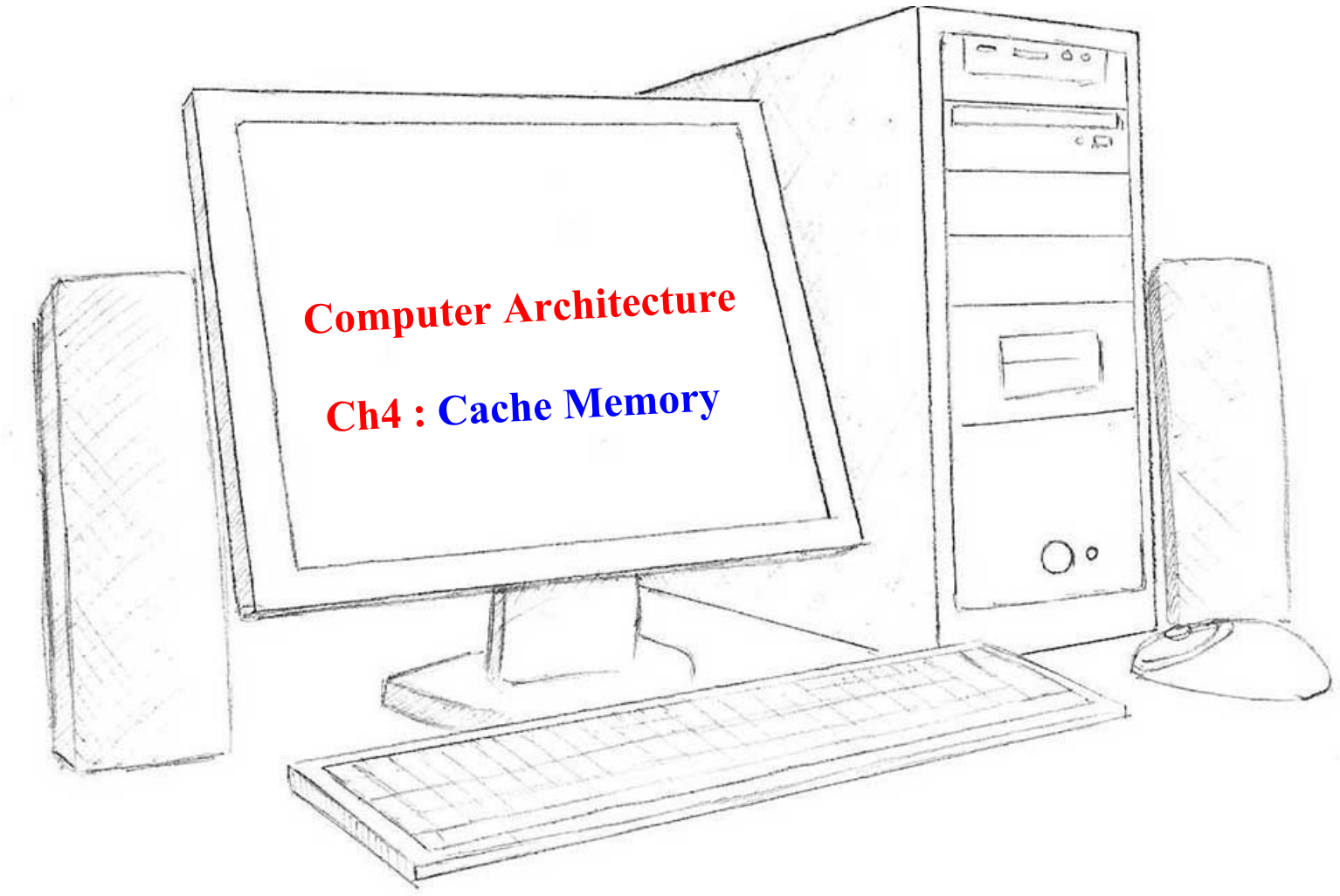


ดร. พิสิทธิ วิสุทธิเมธีกร : pisit.kmutnb@gmail.com

Part II : The Computer System

Ch 4: Cache Memory

4.1 Computer Memory System Overview

4.2 Cache Memory Principle

4.3 Element of Cache Design

4.4 Pentium 4 Cache Organization

4.5 ARM Cache Organization

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Present an overview of the main characteristics of computer memory systems and the use of a memory hierarchy.
- Describe the basic concepts and intent of cache memory.
- Discuss the key elements of cache design.
- Distinguish among direct mapping, associative mapping, and set-associative mapping.
- Explain the reasons for using multiple levels of cache.
- Understand the performance implications of multiple levels of memory.

Characteristic of Memory System

หน่วยความจำของระบบคอมพิวเตอร์อาจแบ่งตามคุณสมบัติที่สำคัญของหน่วยความซึ่งได้แก่

ตารางที่ 4.1 คุณสมบัติของหน่วยความจำ

Location	**Performance**
Internal (e.g., processor registers, cache, main memory)	Access time
External (e.g., optical disks, magnetic disks, tapes)	Cycle time
Capacity	Transfer rate
Number of words	**Physical Type**
Number of bytes	Semiconductor
Unit of Transfer	Magnetic
Word	Optical
Block	Magneto-optical
Access Method	**Physical Characteristics**
Sequential	Volatile/nonvolatile
Direct	Erasable/nonerasable
Random	**Organization**
Associative	Memory modules

Location : แบ่งตามตำแหน่งของหน่วยความจำว่าอยู่ภายในหรือภายนอกคอมพิวเตอร์

Internal Memory โดยปกติจะหมายถึงหน่วยความจำหลัก (Main Memory) แต่อย่างไรก็ตามจะมีหน่วยความจำภายในรูปแบบอื่นอีกเช่น

- ใน Processor ก็จะมีรีจิสเตอร์เป็นหน่วยความจำ
- ในหน่วยควบคุม (Control unit) ซึ่งเป็นส่วนหนึ่งและอยู่ภายใน CPU ก็จะมีหน่วยความจำของส่วนที่เป็น Control Unit โดยเฉพาะ

สำหรับในบทนี้จะสนใจเฉพาะ Internal Memory ส่วนที่เรียกว่า Cache

External Memory จะประกอบด้วยอุปกรณ์ภายนอกที่ใช้ในการเก็บข้อมูล เช่น Disk, Tape ซึ่งจะเชื่อมต่อกับ Processor โดยผ่านทาง I/O Controller

Capacity : สำหรับ Internal Memory แล้วหน่วยของหน่วยความจำจะเป็น bytes หรือ words โดยทั่วไปความยาวแต่ละ words จะมีค่าเท่ากับ 8, 16 และ 32 บิต

Unit of Transfer : สำหรับ Internal Memory จะมีค่าเท่ากับจำนวนสายสัญญาณที่เข้าออกจากหน่วยความจำ ปกติเท่ากับจำนวนบิตของ WORD หรือมากกว่า เช่น 64, 128, 256 บิต เพื่อชี้แจงประเด็นนี้จะกล่าวถึงหลักการสำคัญ 3 ประการที่เกี่ยวข้องกับหน่วยความจำภายใน ได้แก่ Word, Addressable Units และ Unit of Transfer

WORD : ขนาดของ word ปกติมักจะเท่ากับจำนวนบิตข้อมูลที่ใช้แทนข้อมูลแบบ integer และขนาดของคำสั่ง แต่ก็มีข้อยกเว้น เช่น CPU ตระกูล Intel x86 มีคำสั่งหลายคำสั่งที่มีความยาวแตกต่างกัน (เป็นจำนวนเท่าของ Byte) และ word มีขนาดเท่ากับ 32 บิต

Addressable units : ในบางระบบ address จะอ้างหน่วยความจำขนาด 1 word แต่หลายๆระบบยินยอมให้อ้าง address ในระดับไบต์ได้ หากค่า address แทนด้วย **A** bit จำนวนหน่วยความจำที่อ้างแอดเดรสได้จะเท่ากับ **N** แอดเดรสและคำนวณจาก $N = 2^A$

Unit of Transfer : กรณี main memory ค่านี้คือจำนวนบิตที่อ่านหรือเขียนไปยัง memory ในเวลาหนึ่งซึ่งไม่จำเป็นต้องเท่ากับ word หรือ Addressable Unit โดยการส่งผ่านข้อมูลใน Storage จะเป็นลักษณะของ Block มากกว่าจะเป็น word

Access Method :

Sequential access

หน่วยความจำถูกจัดในหน่วยของข้อมูลที่เรียกว่า record

มีการเข้าถึงข้อมูลจะเป็นการเข้าถึงแบบเรียงลำดับเวลาเข้าถึง (Access time) ไม่แน่นอน

Memory ที่ใช้วิธีนี้ : Tape unit

Direct access

แต่ละ Block หรือ record มีแอดเดรสเฉพาะเวลาเข้าถึง (Access time) ไม่แน่นอน

Memory ที่ใช้วิธีนี้ : Disk

Random access

หน่วยความจำมีแอดเดรสเฉพาะเวลาเข้าถึงมีค่าคงที่

Memory ที่ใช้วิธีนี้ : Main Memory และ Cache Memory

Associative

การเข้าถึงแบบ Random, การเข้าถึงข้อมูลจะขึ้นกับส่วนหนึ่งของข้อมูลมากกว่าแอดเดรส, เวลาในการเข้าถึงคงที่

Memory ที่ใช้วิธีนี้ : Cache Memory

Performance :

Access Time (Latency)

กรณีของ RAM คือเวลาในการทำการ Read หรือ Write นับตั้งแต่แอดเดรสถูกส่งไปยัง memory จนกระทั่งข้อมูลถูกเก็บในหน่วยความจำหรือข้อมูลในหน่วยความจำถูกอ่านออกมาไปใช้งาน ส่วนกรณีไม่ใช่ RAM จะเป็นเวลาในการเคลื่อนที่ไปยังตำแหน่งที่ต้องการอ่านหรือเขียนข้อมูล

Memory Cycle Time สำหรับ RAM คือ Access Time บวกด้วยเวลาช่วงหนึ่งที่ต้องการก่อนที่จะมีการเข้าถึงหน่วยความจำครั้งถัดไป ซึ่งเวลานี้จะเกี่ยวกับ System bus ไม่เกี่ยวกับ Processor

Transfer Rate เป็นอัตราการส่งผ่านข้อมูลเข้าหรือออกหน่วยความจำ

กรณีของ RAM มีค่าเท่ากับ 1/Cycle Time

กรณี Non random access memory $T_n = T_A + \frac{n}{R}$

T_n = Average time to read or write n bits

T_A = Average access time

n = Number of bits

R = Transfer rate, in bits per second (bps)

Physical Type Memory

หน่วยความจำที่ใช้งานในปัจจุบันได้แก่

- Semiconductor memory
- Magnetic surface memory (disk และ tape)
- Optical
- Magneto-Optical

Physical Characteristic

- Volatile Memory : ข้อมูลหายไปเมื่อหยุดจ่ายไฟเลี้ยงให้ memory
- Nonvolatile Memory : ข้อมูลยังคงอยู่แม้ไม่มีไฟเลี้ยง
- Nonerasable Memory : ไม่สามารถแก้ไขข้อมูลได้ ถือเป็น Nonvolatile ชนิดหนึ่ง

The Memory Hierarchy

การออกแบบหน่วยความจำระบบคอมพิวเตอร์ต้องคำนึงถึงปัจจัยต่อไปนี้

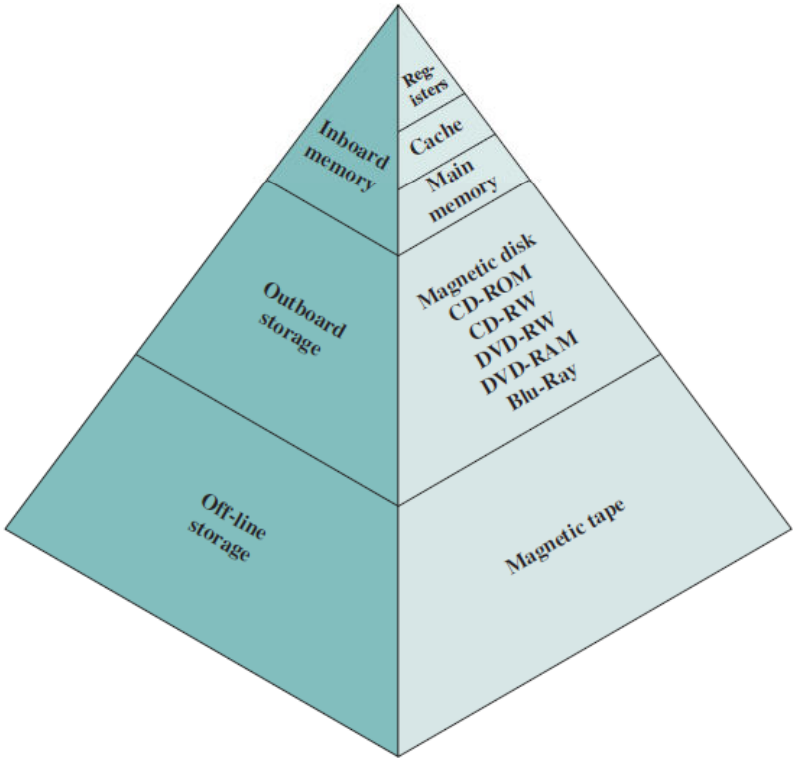
- Capacity
- Speed (Access Time)
- Cost

จากปัจจัยทั้งสามข้างต้นหากหน่วยความจำถูกออกแบบโดยคำนึงถึงปัจจัยใดมากเกินไป ก็อาจมีข้อดีข้อเสียตามมา ซึ่งจะมีความสัมพันธ์ของปัจจัยทั้งสาม ดังนี้

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time

อย่างเช่น ผู้ออกแบบมักจะเลือกใช้เทคโนโลยีหน่วยความจำที่มีความจุสูงด้วยเหตุผลว่าต้องการหน่วยความจำความจุสูงและราคาต่อบิตต่ำ แต่ถ้าหากการสมรรถนะการทำงานสูงผู้ออกแบบหน่วยความจำระบบคอมพิวเตอร์ก็ต้องเลือกหน่วยความจำราคาแพงซึ่งจะมีราคาแพงและมีเวลาในการเข้าถึงสั้น (Short access time)

จากตัวอย่างที่นักออกแบบต้องออกแบบหน่วยความจำตามเงื่อนไขที่ยกตัวอย่างมาซึ่งเงื่อนไขที่ต้องการอาจขัดแย้งกันเอง ดังนั้นการใช้หน่วยความจำเพียงชนิดเดียวในการออกแบบหน่วยความสำหรับคอมพิวเตอร์คงจะไม่ใช่ทางออกที่ดี ดังนั้นในทางปฏิบัติจะใช้การออกแบบหน่วยความจำแบบแบบลำดับชั้น (Memory Hierarchy) ซึ่งมีลักษณะดังรูปที่ 1



รูปที่ 1 The Memory Hierarchy

จากลำดับชั้นหน่วยความจำจากบนลงล่างจะมีคุณลักษณะของหน่วยความจำดังนี้

a. Decreasing cost per bit

b. Increasing capacity

c. Increasing access time

d. Decreasing frequency of access of the memory by the processor

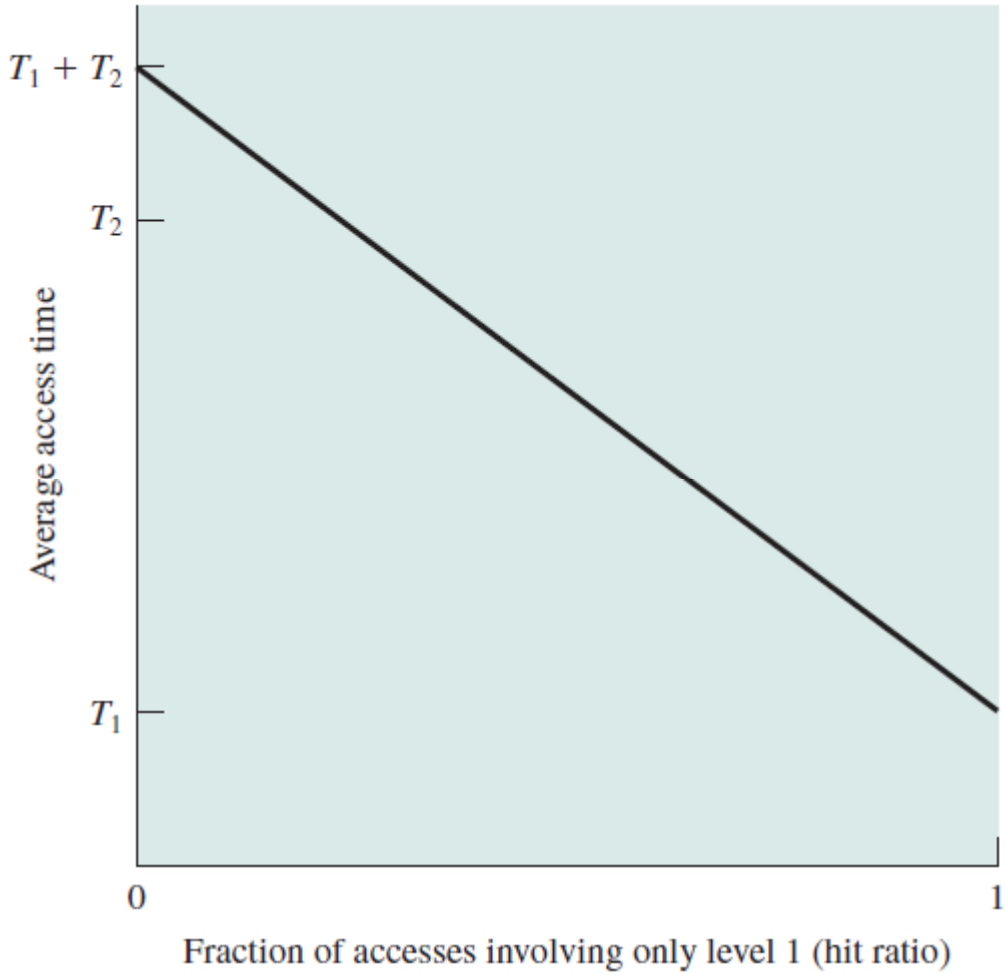
หน่วยความจำคอมพิวเตอร์ แบบที่มีความเร็วสูง จะมีความจุน้อย และราคาแพง ซึ่งในการใช้งานจะมีการนำหน่วยความจำทีมีความจุมาก ราคาถูก ซึ่งมีความเร็วต่ำกว่ามาทำงานเสริมกัน การใช้หน่วยความจำสองระดับจะช่วยลดค่าเฉลี่ยของ access time เมื่อใช้หลักการตามเงื่อนไข a) ถึง d) โดยในหลักการข้อ d) คือการลดความถี่ในการเข้าถึงหน่วยความจำโดย Processor ซึ่งพื้นฐานของแนวคิดนี้อยู่บนหลักการของ Locality of Reference ระหว่างที่ Processor ทำการประมวลผลจะมีการอ้างหน่วยความจำทั้งในส่วนของข้อมูลและคำสั่ง นอกจากนั้นโปรแกรมอาจมีคำสั่งที่ทำงานแบบวนรอบหรือมีการเรียกโปรแกรมย่อย โดยกรณีทำคำสั่งซ้ำๆนี้จะเกิดการอ้างตำแหน่งหน่วยความจำที่ซ้ำๆกัน

Example 4.1 Suppose that the processor has access to two levels of memory. Level 1 contains 1000 words and has an access time of 0.01 μs; level 2 contains 100,000 words and has an access time of 0.1 μs. Assume that if a word to be accessed is in level 1, then the processor accesses it directly. If it is in level 2, then the word is first transferred to level 1 and then accessed by the processor. For simplicity, we ignore the time required for the processor to determine whether the word is in level 1 or level 2. Figure 4.2 shows the general shape of the curve that covers this situation. The figure shows the average access time to a two-level memory as a function of the hit ratio H, where H is defined as the fraction of all memory accesses that are found in the faster memory (e.g., the cache), T_1 is the access time to level 1, and T_2 is the access time to level 2.[1] As can be seen, for high percentages of level 1 access, the average total access time is much closer to that of level 1 than that of level 2.

In our example, suppose 95% of the memory accesses are found in level 1. Then the average time to access a word can be expressed as

$$(0.95)(0.01\,\mu s) + (0.05)(0.01\,\mu s + 0.1\,\mu s) = 0.0095 + 0.0055 = 0.015\,\mu s$$

The average access time is much closer to 0.01 μs than to 0.1 μs, as desired.

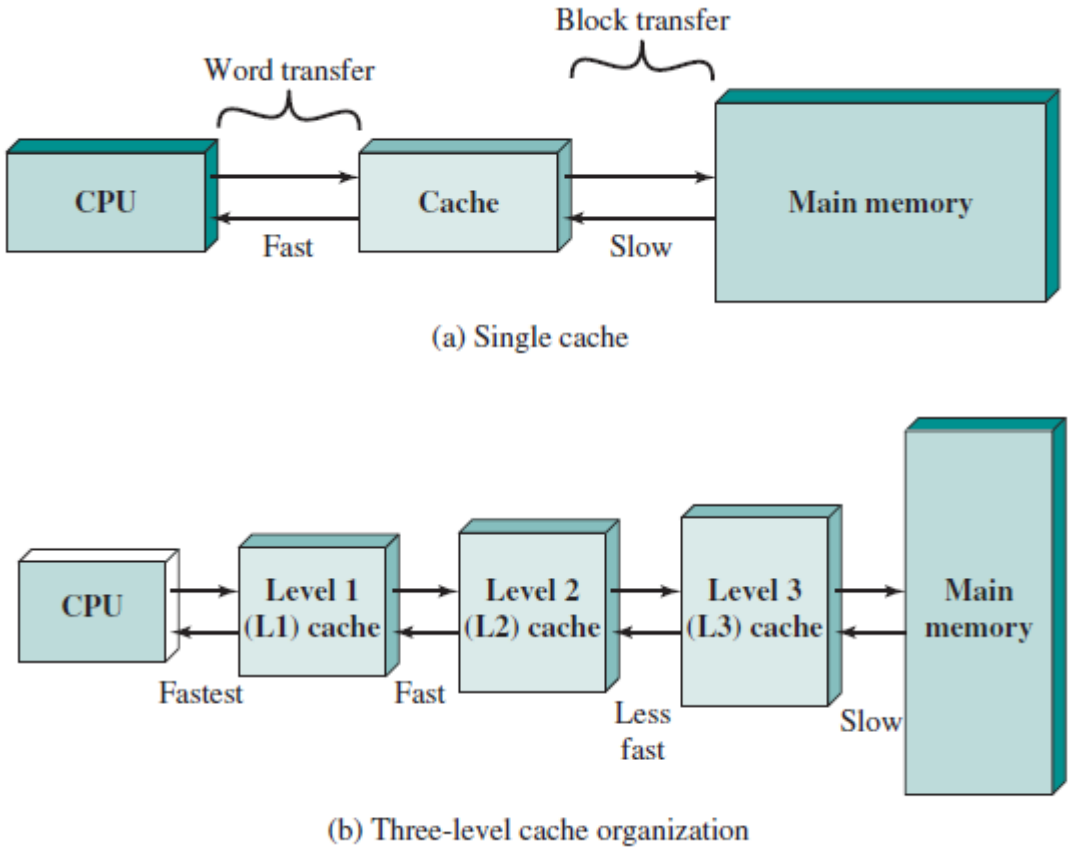


รูปที่ 4.2 Performance of Accesses Involving only Level 1 (hit ratio)

ด้วยวิธีการจัดการหน่วยความจำระหว่างระดับชั้น เช่น เปอร์เซ็นต์การเข้าถึงหน่วยความจำในระดับที่ต่ำกว่าจะมีค่าน้อยกว่าระดับที่อยู่สูงขึ้นไป จากที่ผ่านมาเป็นตัวอย่างของการแบ่งหน่วยความจำเป็นสองระดับ ถ้าหน่วยความจำ level 2 มีคำสั่งและข้อมูล cluster หรือส่วนหนึ่งของโปรแกรมจะถูกนำไปวางในหน่วยความจำ Level 1 และเวลาผ่านไปก็จะมีการสลับ cluster ในหน่วยความจำ Level 1 กลับไปยังหน่วยความจำ Level 2 เพื่อให้เกิดที่ว่างสำหรับ cluster ใหม่ที่จะถูกนำมายังหน่วยความจำ Level 1 โดยเฉลี่ยแล้วการอ้างอิงหน่วยความจำก็จะเป็นการอ้างคำสั่งและข้อมูลที่อยู่ในหน่วยความจำ Level 1 หลักการนี้สามารถประยุกต์ใช้กับหน่วยความจำที่มีมากกว่าสองระดับอย่างที่นำเสนอในรูปที่ 4.1 หน่วยความที่เร็วที่สุดขนาดเล็กที่สุดและแพงที่สุดก็จะเป็นรีจิสเตอร์ภายใน Processor ซึ่งปกติจะมีประมาณ 20 กว่ารีจิสเตอร์เป็น 100 ใน Processor บางตัว Main memory เป็นหน่วยความจำหลักภายในระบบคอมพิวเตอร์โดยแต่ละตำแหน่งของ main memory จะมีค่าแอดเดรสเฉพาะและโดยทั่วไป main memory นี้จะเป็น cache ขนาดเล็กและความเร็วสูง โดย cache ใช้เคลื่อนย้ายข้อมูลระหว่าง main memory และ processor

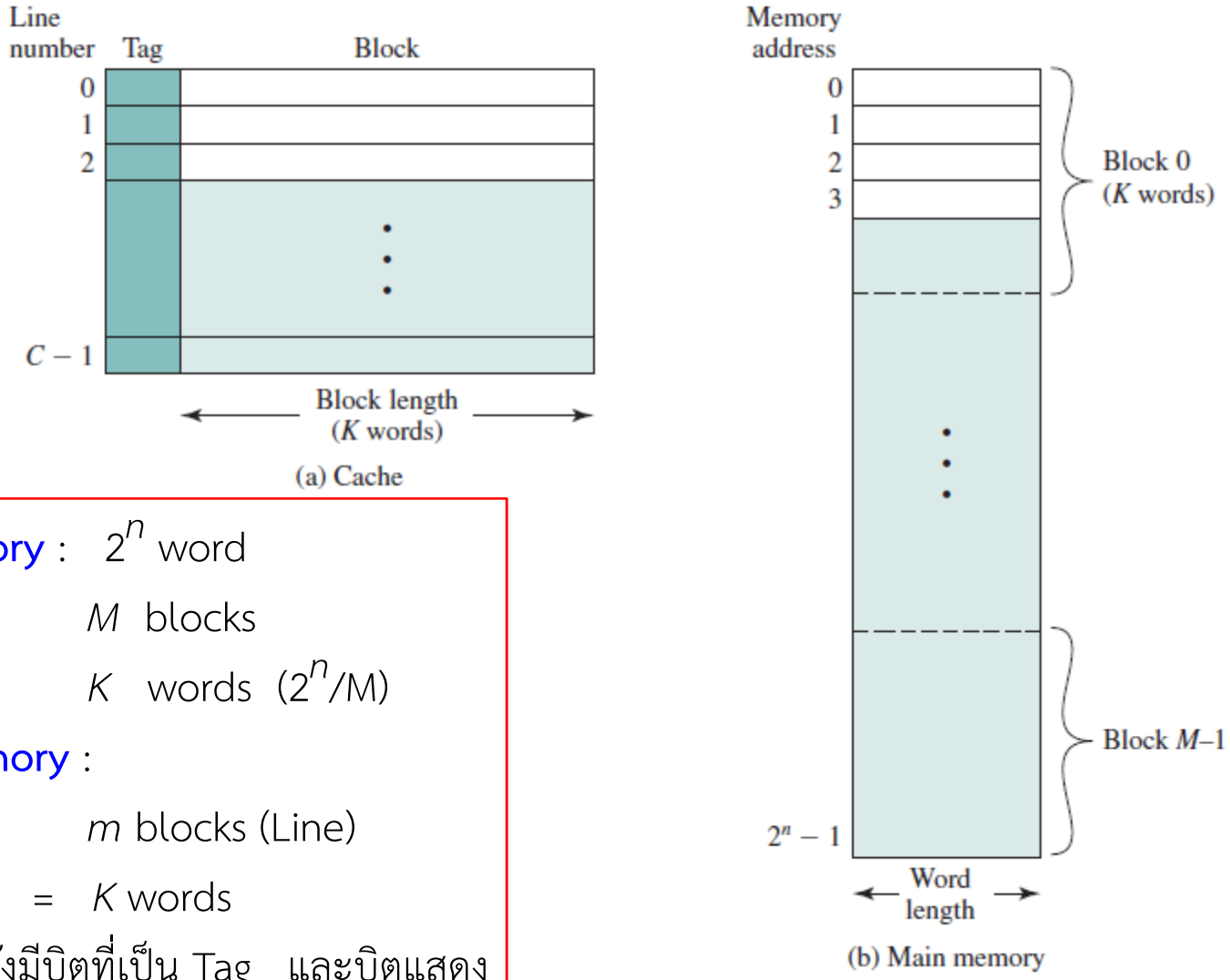
ที่ผ่านมาได้กล่าวถึงหน่วยความจำ 3 แบบได้แก่

Volatile memory เป็นหน่วยความจำที่เป็น semiconductor แต่เทคโนโลยี Semiconductor ก็มีหลากหลายแตกต่างกันในเรื่องของ ความเร็ว, ราคา โดยปกติ data มักจะถูกเก็บแบบถาวรในอุปกรณ์เก็บข้อมูลภายนอก (External Mass Storage Device) เช่น hard disk และ removable media (removable magnetic disk, tape, and optical storage) สำหรับ External, nonvolatile memory ปกติแล้วจะใช้งานเป็น secondary memory หรือ auxiliary memory ซึ่งมักจะถูกใช้ในการเก็บโปรแกรมและไฟล์ข้อมูลซึ่งมุมมองของโปรแกรมเมอร์จะมองเห็นข้อมูลในระดับ files และ records ส่วนกรณีของ Disk อาจถูกใช้เป็นส่วนขยายของ main memory ที่เรียกว่า virtual memory,



รูปที่ 4.3 Cache and Main Memory

Cache memory ถูกออกแบบมาเพื่อผสมผสานการใช้งานหน่วยความจำที่มีราคาแพงความเร็วสูง กับหน่วยความจำที่มีขนาดความจุมาก ราคาถูก ความเร็วต่ำ ซึ่งหลักการของ Cache แสดงดังในรูปที่ 4.3 a การทำงานของ Cache จะมีการ copy ข้อมูลบางส่วนจาก main memory เมื่อ Processor ต้องการอ่านข้อมูลในหน่วยความจำจะมีการตรวจสอบว่าข้อมูลที่ต้องการอ่านนั้นอยู่ใน Cache หรือไม่ ถ้าอยู่ใน Cache ก็จะเกิดกระบวนการส่งข้อมูลนั้นไปยัง Processor แต่ถ้าข้อมูลที่ต้องการไม่พบใน Cache Block ของหน่วยความจำอันใหม่จะถูกอ่านเข้ามายัง Cache และข้อมูลจะถูกส่งไปยัง Cache ในรูปที่ 4.3 b เป็นระบบหน่วยความจำที่ Cache แบบสามระดับ ซึ่งในกรณี cache สามระดับนี้ L2 cache จะมีความจุมากกว่าและทำงานช้ากว่า L1 cache ส่วน L3 ก็จะยิ่งมีความจุของหน่วยความจำที่มากกว่า และทำงานช้ากว่าทั้ง cache ในระดับ L1 และ L2



Main memory : 2^n word

แบ่งเป็น M blocks

block ละ K words (2^n/M)

Cache memory :

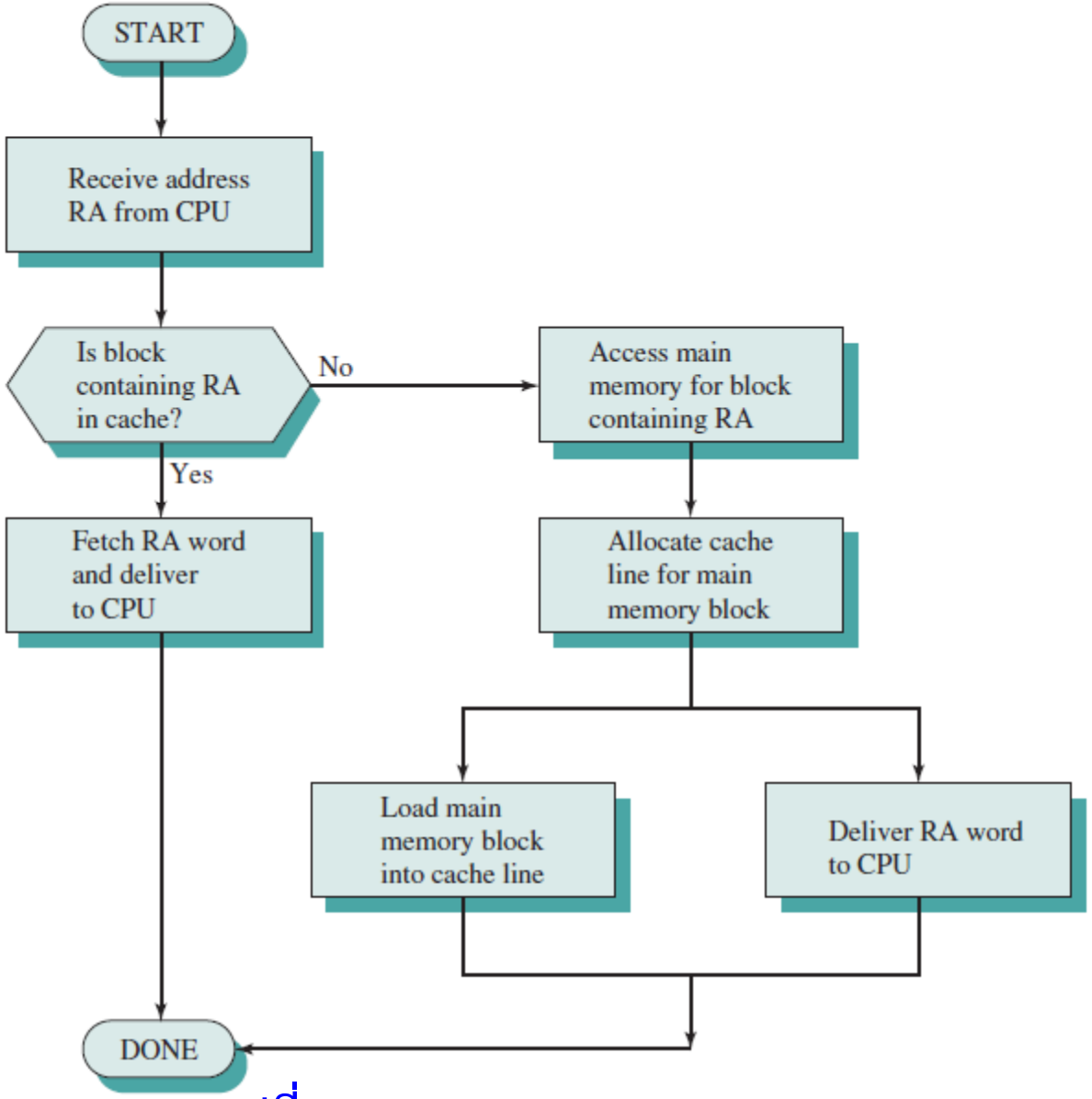
แบ่งเป็น m blocks (Line)

1 Line = K words

ใน cache ยังมีบิตที่เป็น Tag และบิตแสดงสถานะของแต่ละ Line อีกจำนวนหนึ่ง

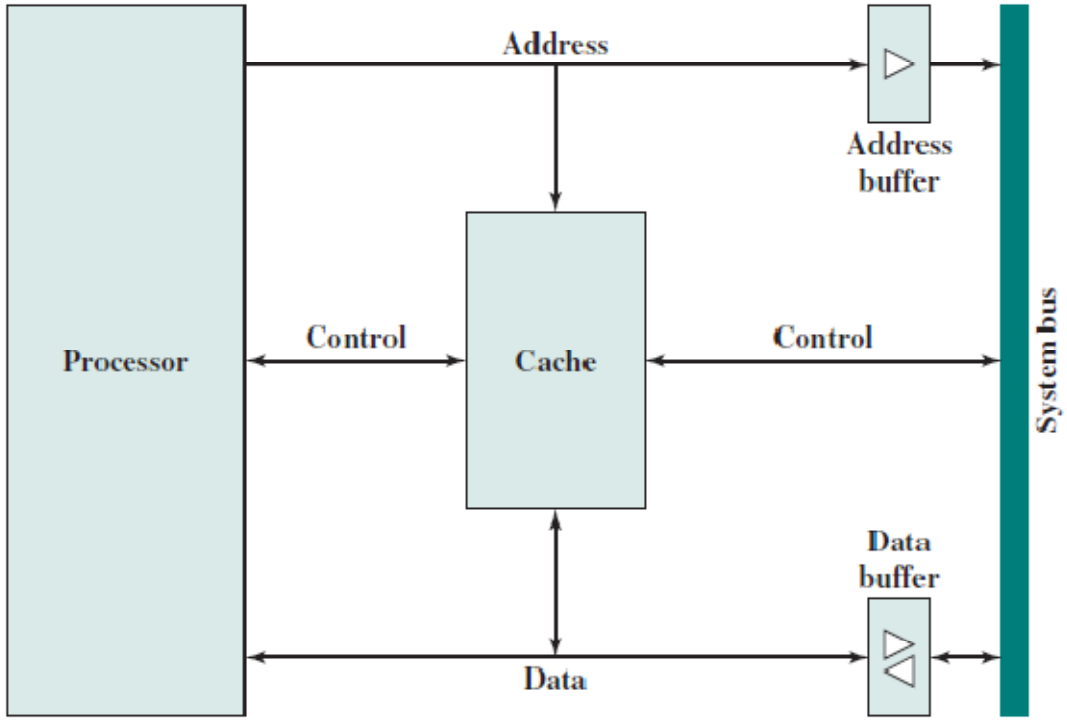
รูปที่ 4.4 Cache / Main Memory Structure

ในแต่ละ Line ของ cache จะประกอบด้วยพื้นที่เก็บข้อมูล *K* words และมีบิตของ Tag และบิตแสดงสถานะว่าข้อมูลใน Line ดังกล่าวมีการปรับปรุงไปตอนที่โหลดมายัง cache หรือไม่ โดยขนาดของ Cache แต่ละ Line เรียกว่า Line size นั้นจะไม่รวม Tag และ บิตสถานะเหล่านี้ ซึ่ง Line size อาจจะมีขนาดเล็กเพียง 32 บิต และแต่ละ word ก็มีขนาดเป็น ไบต์ นั่นคือกรณีตัวอย่างนี้ Line size มีค่าเป็น 4 ไบต์ ปกติแล้วจำนวน Line ของ cache memory (*m*) มักจะกำหนดให้มีค่าน้อยกว่าจำนวน block ของ main memory (*M*) หรือ $m << M$ ทำให้ไม่สามารถมี Line สำหรับ ทุก block ของ main memory ได้ จึงต้องมีการสลับสับเปลี่ยน block ของ main memory ที่จะไปเก็บใน Line ของ cache โดย **Tag** จะเป็นค่าที่ใช้บอกว่า Block ใดของ main memory ที่ถูกเก็บใน cache line นั้นๆ (Tag เป็นส่วนหนึ่งของค่าแอดเดรสที่ชี้ main memory)



รูปที่ 4.5 Cache Read Operation

ในรูปที่ 4.5 แสดงขั้นตอนการอ่านของ Processor โดยเริ่มที่ Processor ส่งค่าแอดเดรสที่ต้องการอ่าน (Read Address : RA) จากหน่วยความจำ ถ้าข้อมูลที่ต้องการอยู่ใน cache ข้อมูลนั้นก็จะถูกส่งมายัง Processor แต่ถ้าไม่มีใน cache ก็จะเกิดการโหลด block ของ main memory ที่มีข้อมูลที่ต้องการนั้นมายัง Cache memory และข้อมูลก็จะส่งไปยัง Processor ในเวลาเดียวกันด้วย ซึ่งถ้าการทำงานดังกล่าวนี้ Organization ของระบบจะแสดงดังรูปที่ 4.6 ซึ่งจาก Processor ไปยัง System Bus จะมีบัฟเฟอร์โดย System Bus ก็คือส่วนที่จะเชื่อมต่อไปยัง Main memory นั่นเอง ในตอนที่ข้อมูลที่Processor ต้องการอ่านมีอยู่ใน cache วงจรส่วน Data และ Address Buffer จะถูก disable และการส่งผ่านข้อมูลก็จะเกิดขึ้นเฉพาะ Processor และ cache เท่านั้น



รูปที่ 4.6 Cache Organization

ในหัวข้อนี้จะอธิบายเกี่ยวกับพารามิเตอร์ที่เกี่ยวข้องกับการออกแบบ Cache ในระบบคอมพิวเตอร์และนำเสนอข้อมูลที่เกี่ยวข้อง โดยการใช้งาน Cache ปกติแล้วจะใช้งานในการประมวลผลสมรรถนะสูง (HPC : High Performance Computing) ซึ่งมักจะเป็นระบบซุปเปอร์คอมพิวเตอร์ ที่ใช้งานเพื่อการประมวลผลข้อมูลขนาดใหญ่ ทั้งที่เป็นการคำนวณที่เกี่ยวข้องกับ Vector และ Matrix และยังใช้ในอัลกอริทึมแบบขนาน (Parallel Algorithm) ซึ่งมีผู้วิจัยหลายท่านได้นำเสนอและแสดงให้เห็นว่า Cache แบบลำดับชั้นนั้นมีประโยชน์ในการปรับปรุงสมรรถนะของระบบโดยรวม ถ้า Application Software บนคอมพิวเตอร์นั้นถูกปรับให้มีการใช้ประโยชน์จาก Cache สำหรับพารามิเตอร์ในการออกแบบ Cache ที่สำคัญแสดงในตารางที่ 4.2

ตารางที่ 4.2 Element of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Cache Address

ในระบบคอมพิวเตอร์โดยส่วนใหญ่แล้วจะรองรับหน่วยความเสมือน (Virtual Memory) โดยหน่วยความจำเสมือนนี้เป็นสิ่งอำนวยความสะดวกให้โปรแกรมในการอ้างถึงหน่วยความจำโดยไม่ได้ขึ้นกับว่าหน่วยความจำหลัก (Main memory) นั้นมีหน่วยความจำทางกายภาพที่แท้จริงเท่าใด โดยเมื่อมีการใช้งานหน่วยความจำเสมือนส่วนที่เป็น address ในชุดคำสั่งก็จะต้องเป็นค่าตำแหน่งเสมือนที่ชี้ไปยังหน่วยความจำเสมือนด้วย โดยในการอ่านหรือเขียนข้อมูลกับหน่วยความจำหลักนั้นจะมีส่วนที่เรียกว่า MMU (Memory Management Unit) เป็นฮาร์ดแวร์ทำหน้าที่แปลงแอดเดรสเสมือนให้กลายเป็น Physical address ที่ชี้ไปยังหน่วยความจำหลัก

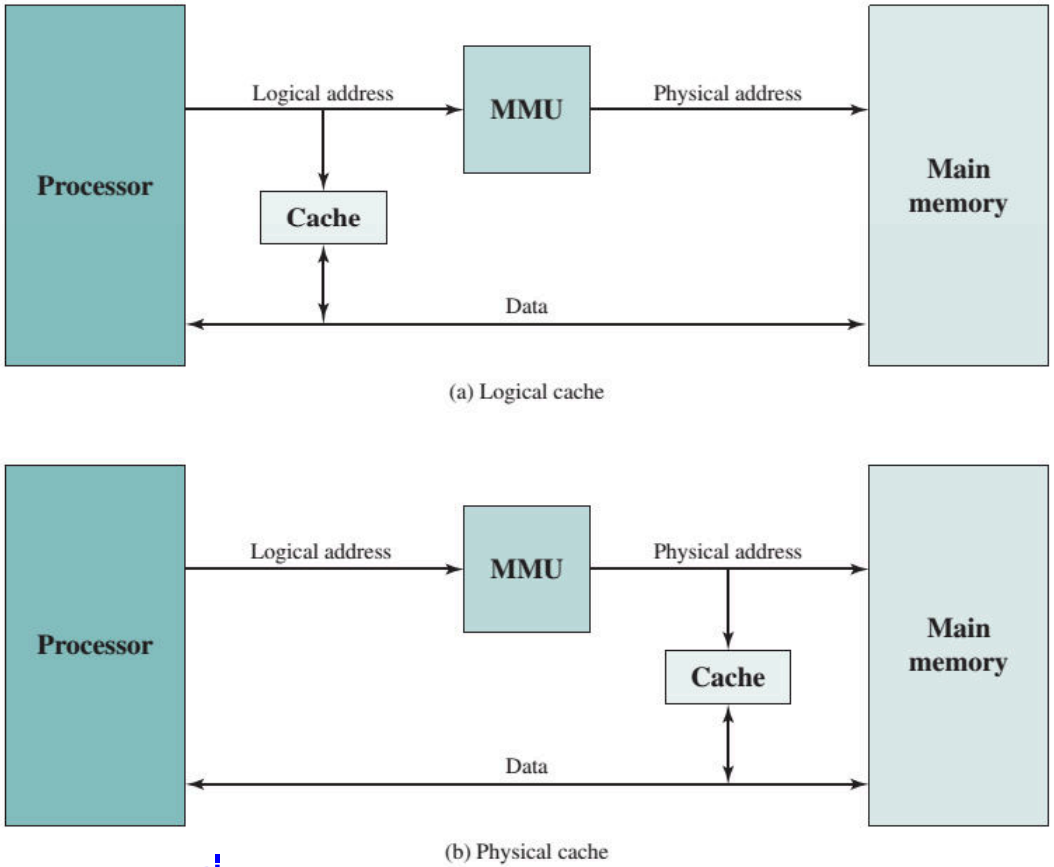
โดย cache address จะแบ่งออกเป็น 2 ลักษณะ ขึ้นกับว่าผู้ออกแบบระบบนั้นเลือกวางตำแหน่งของ Cache ไว้อย่างไรดังในรูปที่ 4.7 ซึ่งได้แก่

Logical Address : Cache ถูกวางระหว่าง CPU และ MMU

Physical Address : Cache ถูกวางระหว่าง MMU และ Main memory

Logical Cache หรือ **Virtual Cache** : CPU เข้าถึง cache โดยตรงโดยไม่ผ่าน MMU โดยจะใช้ Virtual address เพราะ cache ไม่ได้เป็นส่วนหนึ่งของ main memory

Physical Cache แบบนี้ cache จะเป็นส่วนหนึ่งของ main memory ดังนั้นคือแอดเดรสจะเป็นส่วนของ main memory ด้วย



รูปที่ 4.3 Cache and Main Memory

Logical Cache มีข้อดีคือ CPU เชื่อมต่อโดยตรงทำให้การเข้าถึงเร็วกว่ากรณี Physical Cache แต่ก็มีข้อเสียในเรื่องของการกำหนด virtual address ให้กับ Application เพราะในระบบ Virtual Memory จะต้องกำหนดให้แต่ละ application ด้วยแอดเดรสเดียวกัน (Virtual Address) แต่จะชี้ไปยัง Physical Address ที่แตกต่างกันเพราะ Application คนละตัวกันก็จะโหลดไปยังPhysical Main memory คนละตำแหน่งกัน ดังนั้นการนำ block ข้อมูลจาก main memory มายัง cache จะต้องมีการนำข้อมูลรายละเอียดของ Application มาด้วยหรืออาจมีการเพิ่มบิตพิเศษในแต่ละ Line ของ Cache เพื่อบอกว่าแต่ละ Virtual Address นั้นอ้างถึงแอดเดรสใดของ main memory

Cache Size

การกำหนดขนาด Cache จะกำหนดให้ความจุน้อยจนราคาเฉลี่ยของหน่วยความจำต่อบิตมีค่าใกล้เคียงกับราคาของ main memory และต้องมีความจุมากพอที่จะทำให้ค่าเฉลี่ยของ Access time มีค่าใกล้เคียงกับกรณีที่ระบบมีหน่วยความจำเป็น cache เพียงชนิดเดียว ตารางที่ 4.3 ตัวอย่าง cache ใน processor

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16–32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128–256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256–512 kB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 kB to 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/ server	2011	6 × 32 kB/ 32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

ตารางที่ 4.3 Cache sizes ของ Processor บางเบอร์

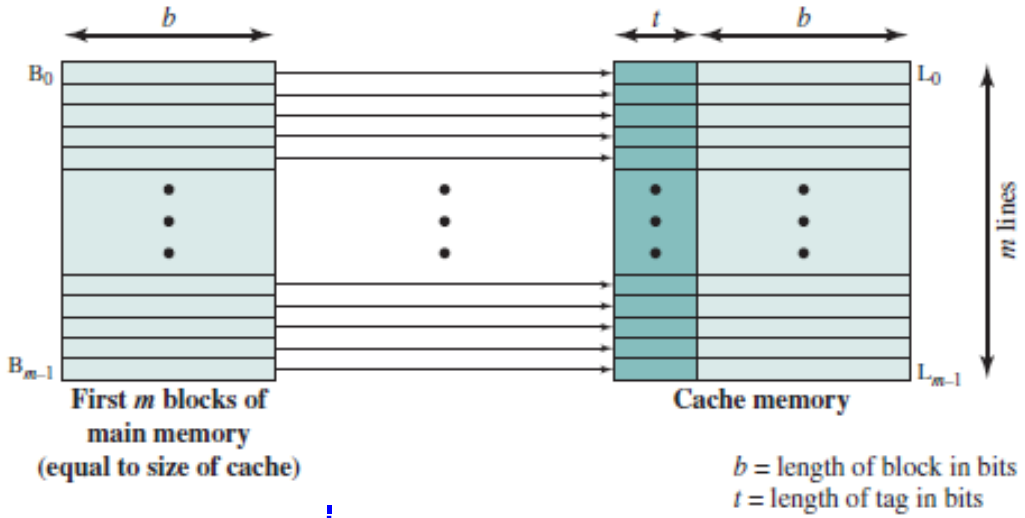
Mapping function

เนื่องจากมี Cache line น้อยกว่าจำนวน block ของ main memory ดังนั้นจะต้องมีอัลกอริทึมนำข้อมูลในหน่วยความจำหลักไปเก็บใน cache หรือก็คือวิธีการในการหาว่าจะนำ block ใดของ main memory ไปเก็บใน Cache นั่นเอง ซึ่งในที่นี้ได้แก่

- Direct Mapping
- Associative Mapping
- Set Associative Mapping

Example 4.2 For all three cases, the example includes the following elements:

- The cache can hold 64 Kbytes.
- Data are transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as $16K = 2^{14}$ lines of 4 bytes each.
- The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address ($2^{24} = 16M$). Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.



รูปที่ 4.4 Direct Mapping

Direct Mapping

ถือว่าเป็นวิธีการง่ายที่สุด โดยให้ 1 block ของ main memory เท่ากับขนาด 1 Line ของ Cache โดยการจะย้ายข้อมูลจาก main memory ใน block ที่ *j* ไปที่ Cacheหมายเลข Line ที่ *i* จะคำนวณหมายเลย cache Line ได้ดังนี้

$$i = j \text{ modulo } m$$

เมื่อ *i = cache line number* *j = main memory block number*

m = number of lines in the cache

วิธี Direct Mapping **จะโหลดข้อมูลในหน่วยความจำที่ละ block จาก main memory ไปยัง Cache โดย block ที่โหลดไปนั้นจะมีหมายเลขของ Line ที่จะโหลดไปแน่นอน** ตัวอย่างเช่น

กรณี 1. ย้าย block ที่ 0 ถึง block ที่ *m*-1 ไปยัง Cache

block ที่ 0 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ 0

block ที่ 1 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ 1

- - - - -

block ที่ *m*-1 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ *m*-1

กรณี 2. ย้าย block ที่ m ถึง 2m-1 ไปยัง Cache

block ที่ *m* ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ 0

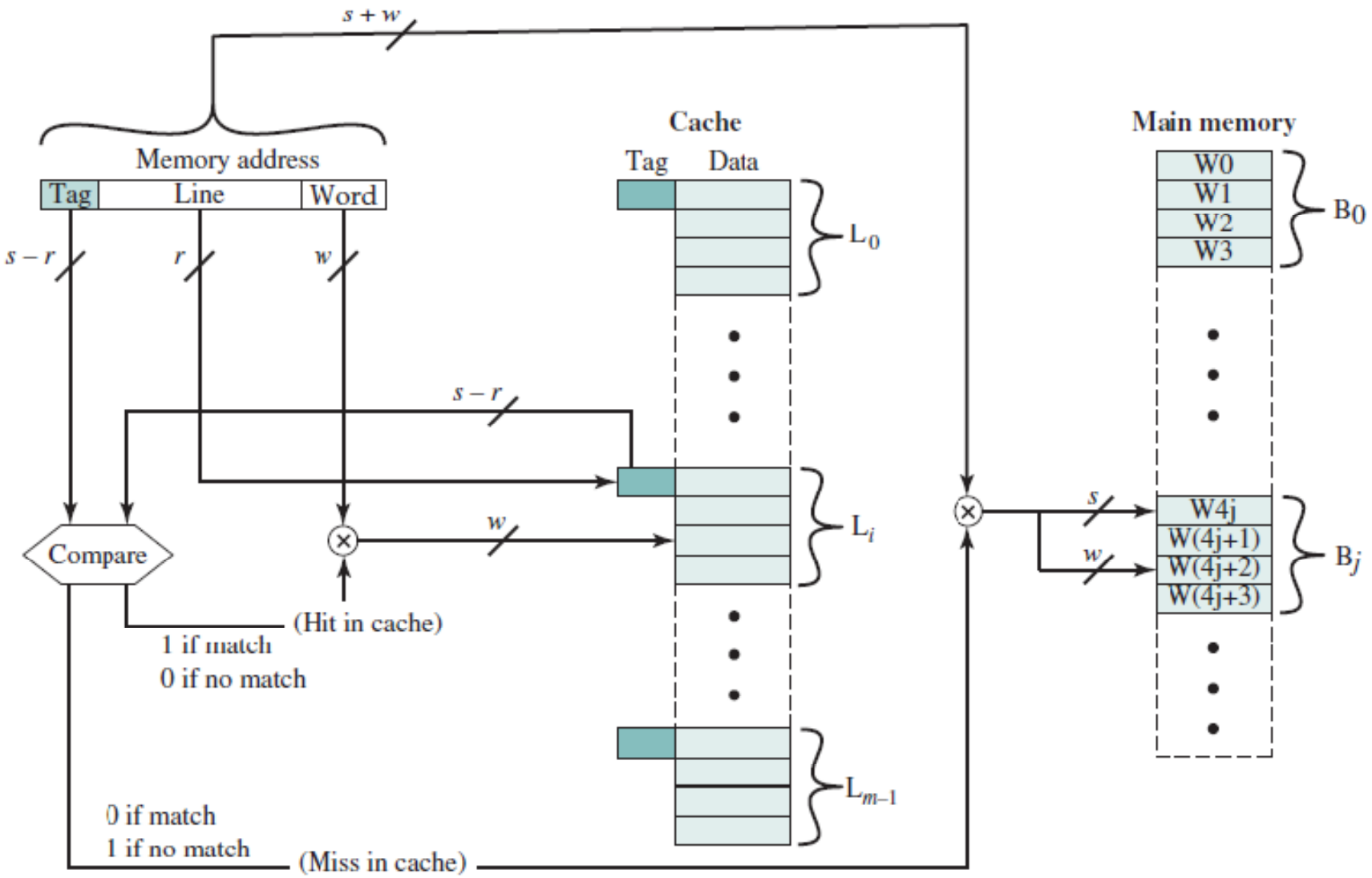
block ที่ *m*+1 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ 1

- - - - -

block ที่ 2*m*-1 ถูกโหลดไปยัง Cache ที่มีหมายเลข Line เท่ากับ *m*-1

ตารางที่ 4.4 ตัวอย่าง Direct Mapping Cache Line Table

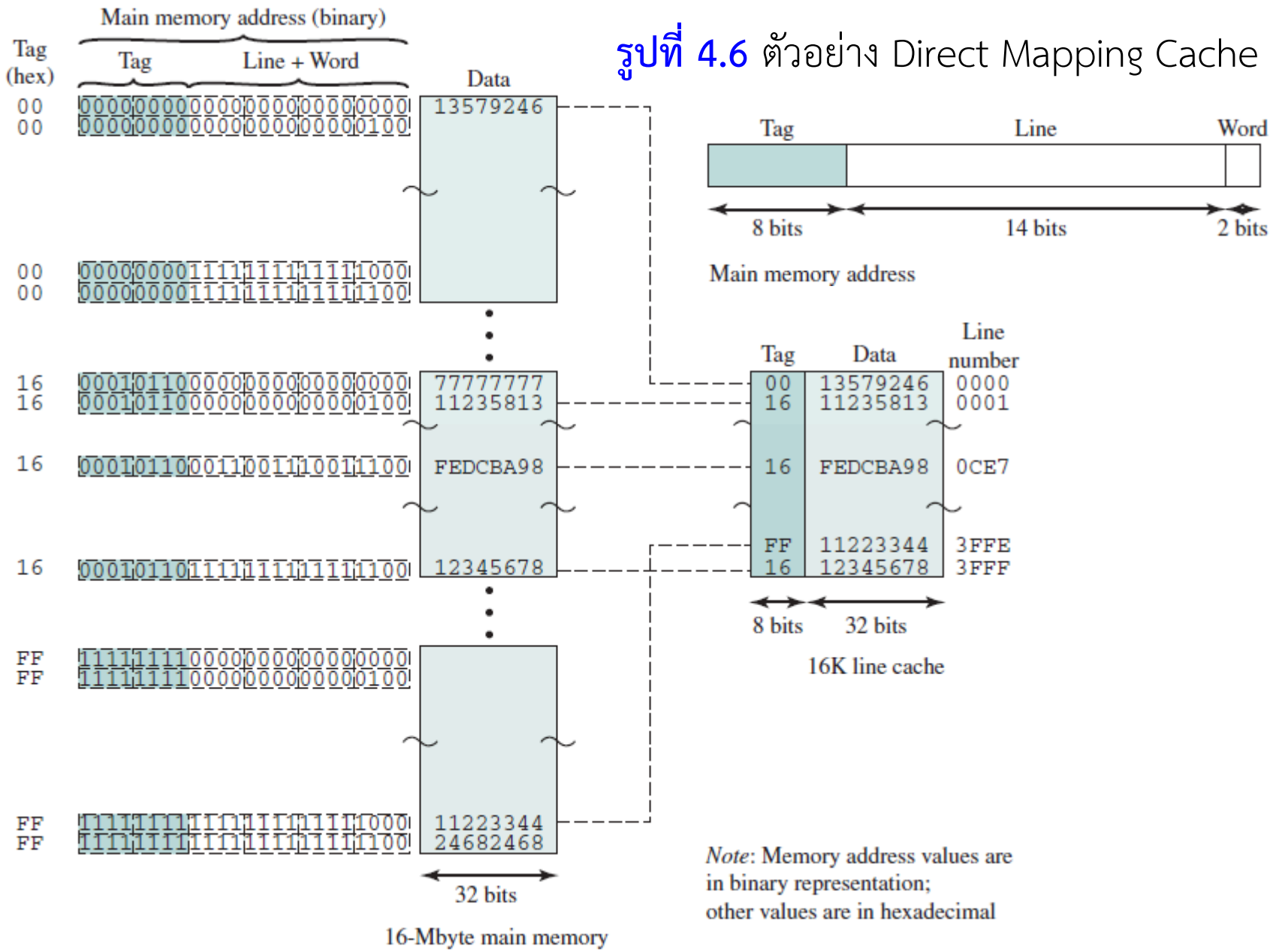
Cache line	Main Memory blocks held
0	0, m, 2m, 3m...2s-m
1	1,m+1, 2m+1...2s-m+1
...	
m-1	m-1, 2m-1,3m-1...2s-1



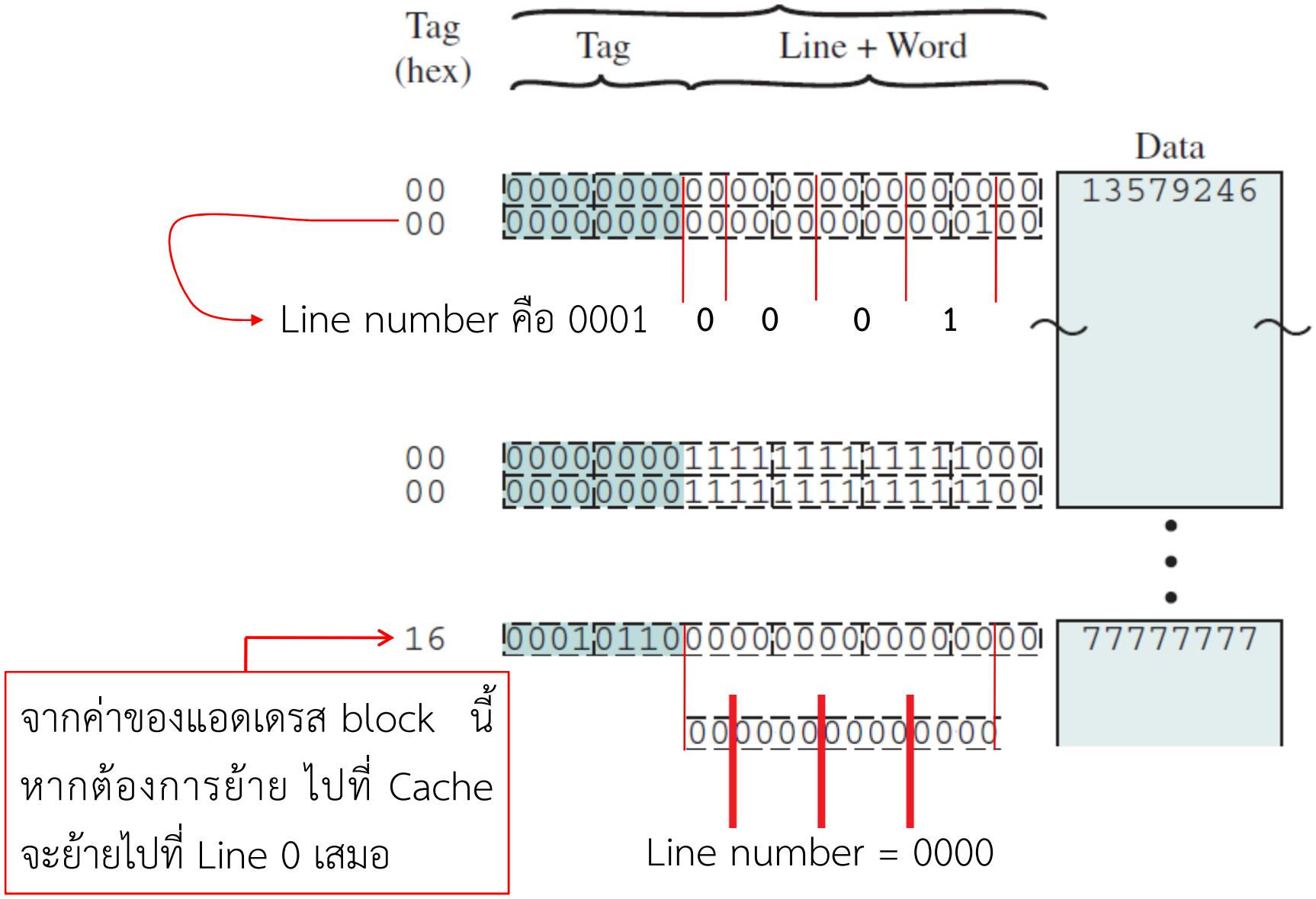
รูปที่ 4.5 Direct Mapping Cache Organization

Address Line of Main Memory & Cache for Direct Mapping

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits



รูปที่ 4.6 ตัวอย่าง Direct Mapping Cache



รูปที่ 4.7 ตัวอย่าง Line number ของ memory block กรณี Direct Mapping

Example 4.2a Figure 4.6 shows our example system using direct mapping. In the example, $m = 16K = 2^{14}$ and $i = j$ modulo 2^{14}. The mapping becomes

Cache Line	Starting Memory Address of Block
0	000000, 010000, …, FF0000
1	000004, 010004, …, FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, …, FFFFFC

จะไม่มี 2 block ใดๆ ยัง cache ที่ Line number เดียวกันโดยที่มีหมายเลข Tag เหมือนกัน ดังนั้นในตัวอย่างนี้ blocks ที่มีแอดเดรสเริ่มต้นด้วยแอดเดรส 000000, 010000, ..., FF0000 จะมีหมายเลข tag เท่ากับ 00,01, เรื่อยไปจน FF, ตามลำดับ

ข้อดี Direct mapping

- Simple
- inexpensive to implement.

ข้อเสีย Direct mapping

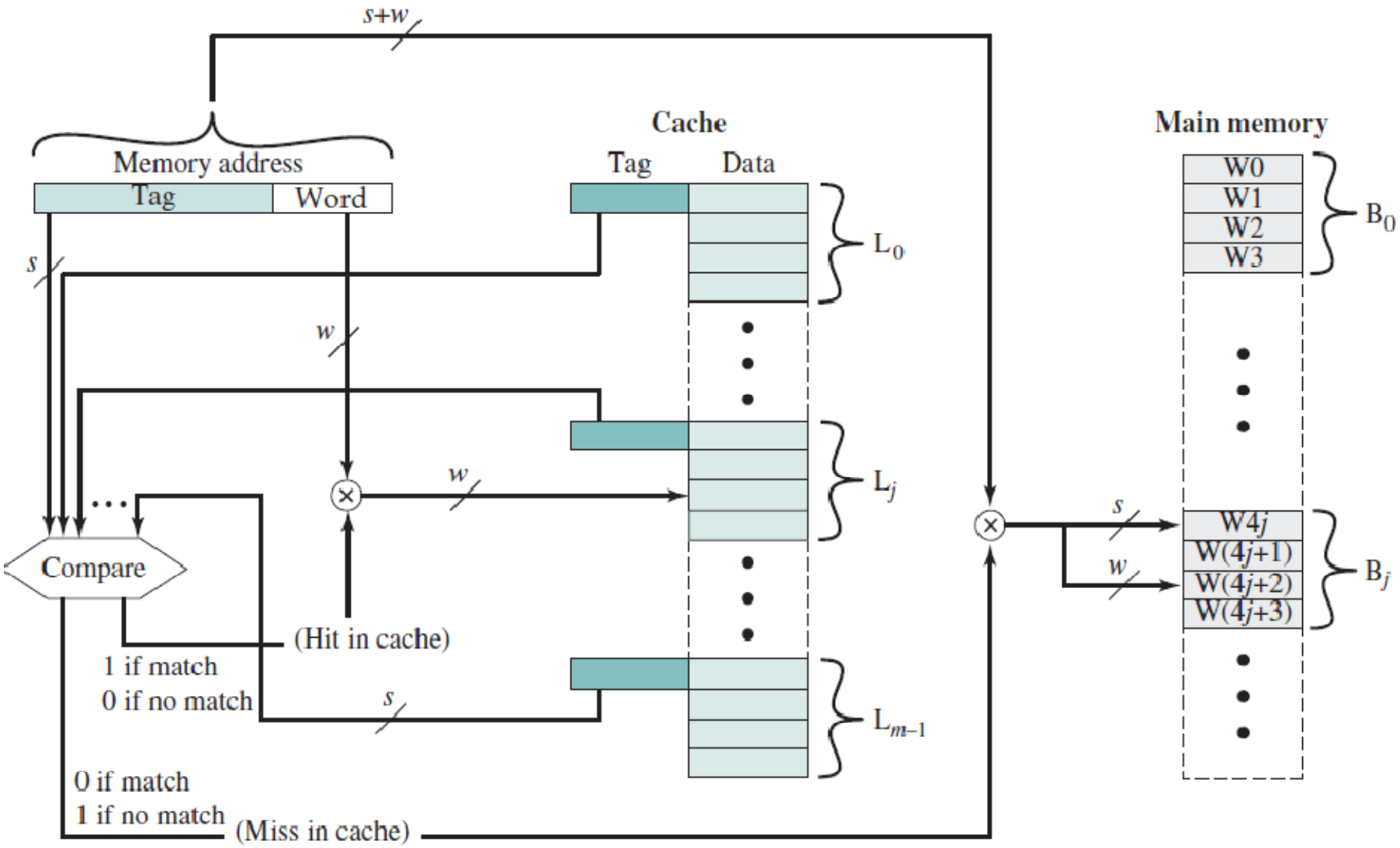
ข้อเสียของวิธีการนี้ก็คืออาจมี 2block ใดที่มีหมายเลข Line number เดียวกันเมื่อโหลดไปยัง Cache ก็จะต้องอยู่ที่ตำแหน่งเดียวกัน ดังนั้นจะมีเพียง 1 block เท่านั้นที่อยู่ใน Cache ณ เวลาใดๆ ดังนั้นหาก Application บนระบบอ่าน Memory block ทั้งสองนี้สลับกันตลอดเวลา ก็จะเกิดปัญหาคือจะมีการ สลับ memory block ทั้งสองนี้เข้าออกหน่วยความจำ Cache และ main memory ตลอดเวลาซึ่งแทนที่ Processor จะติดต่อกับ Cache ตลอดเพื่อให้เกิดการทำงานที่รวดเร็วกว่าการติดต่อกับ main memory หรือก็คือมี Hit ratio สูงๆ ซึ่งกรณีเกิดการสลับ 2 block ตลอดเวลาดังกล่าวนี้จะเรียกว่า *thrashing*

Victim Cache

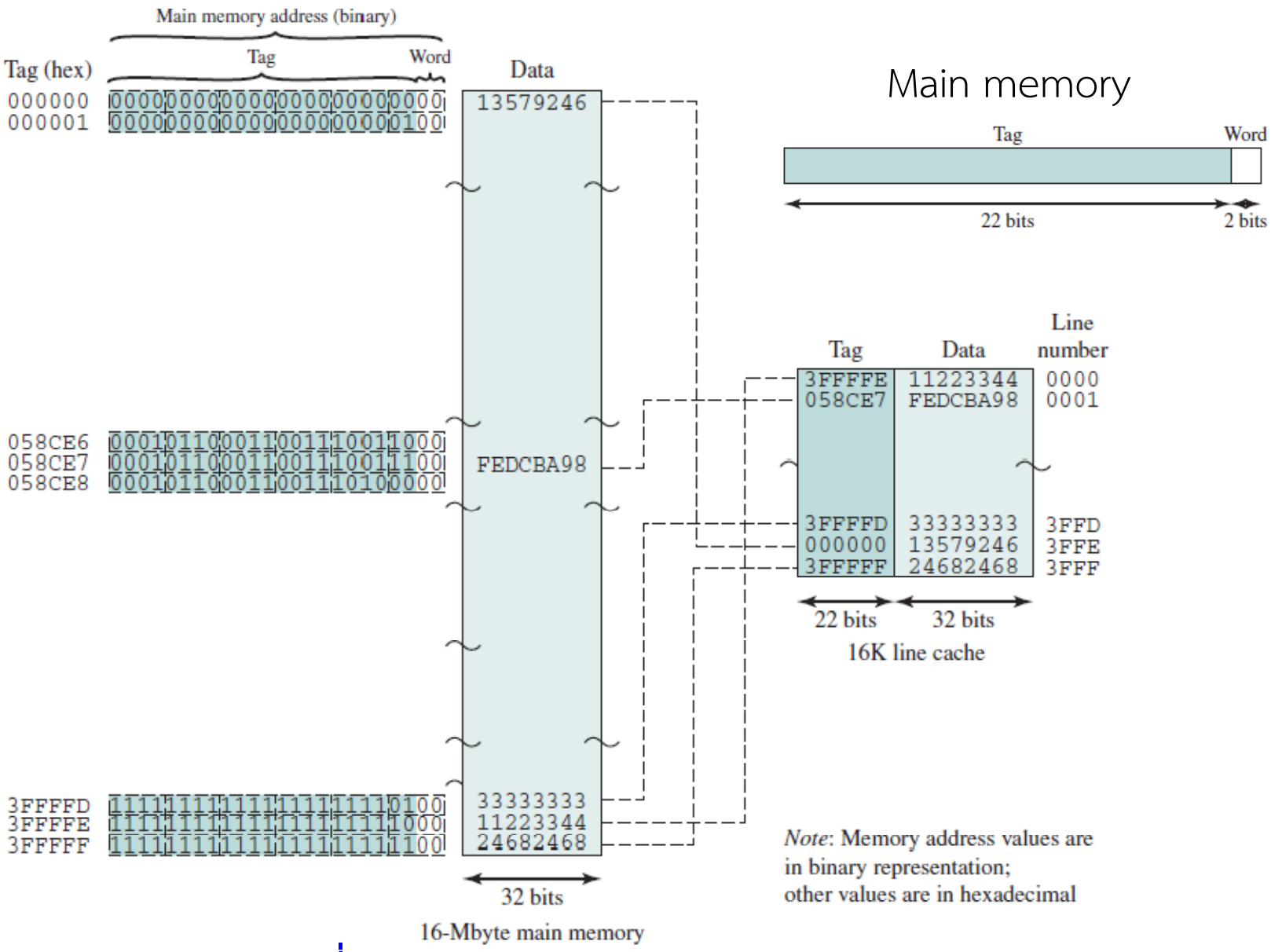
- ถูกนำเสนอมาเพื่อแก้ไขปัญหาของ direct mapped cache ในกรณีที่ line number ซ้ำกัน
- มีการทำงานแบบ Fully associative cache
- ปกติจะมีขนาดเพียงแค่ 4 – 16 lines
- จะวางในตำแหน่งระหว่าง direct mapped cache และ memory ชั้นถัดไป

Associate Mapping

วิธีการนี้นำมาแก้ปัญหาของ Direct Mapping โดย block ของ main memory จะถูกโหลดไปยัง Line ใดๆของ Cache ที่ว่าง โดยแอดเดรสของ main memory จะถูกแบ่งเป็น 2 ส่วนคือ Tag และ Word โดย main memory block หนึ่งๆจะมีค่า Tag 1 ค่าเท่านั้น



รูปที่ 4.8 Fully Associative Cache Organization



รูปที่ 4.9 ตัวอย่าง Associative Mapping

Address Line of Main Memory & Cache กรณี Associative Mapping

- ความยาว Address ของ cache = (s + w) bits
- จำนวนของแอดเดรสที่อ้างถึงได้ = 2^{s+w} words หรือ bytes
- Block size = line size = 2^{w} words หรือ bytes
- จำนวนของ block ในหน่วยความจำหลัก = 2^{s}
- จำนวนของ line ใน cache = undetermined
- ขนาดของ cache tag = s bits

ด้วยวิธีการนี้จะต้องมีการใช้ Replacement Algorithm เมื่อมีการโหลด block ใหม่จาก main memory จะต้องหาว่าจะไปแทนที่ Line ใดใน cache ซึ่งจะกล่าวถึงกันต่อไป

Set Associate Mapping

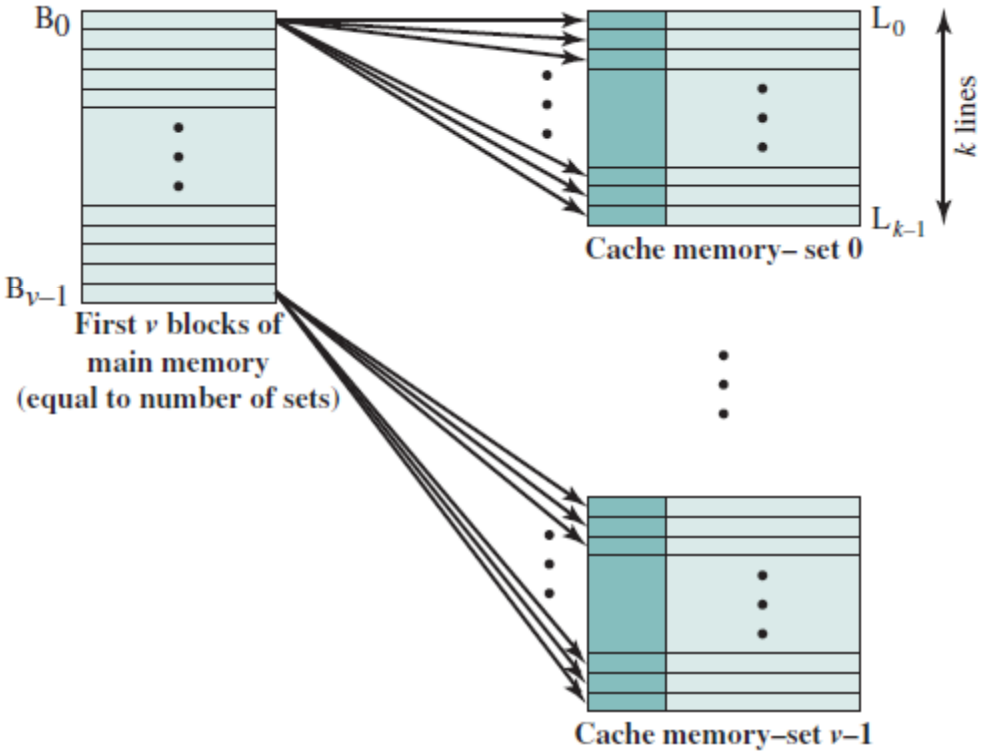
- เป็นการ mapping ที่พยายามนำข้อดีของเทคนิค Direct map และ Fully Associative map มาใช้และลดข้อเสียที่เกิดในแต่ละแบบด้วย
- มีการแบ่ง Cache จออกเป็นกลุ่มๆเรียกว่า Set
- ในแต่ละ Set ของ Cache จะแบ่งออกเป็น Lines
- Block ของหน่วยความจำจะถูก map ลงที่ line ไหนก็ได้ (fully associative) แต่อยู่ใน set ที่กำหนด (direct)
- กำหนดให้

$$i = j \bmod v$$

$$m = v * k$$

i = cache set number
j = main memory block number
m = number of lines in the cache
v = number of sets
k = number of lines in each set

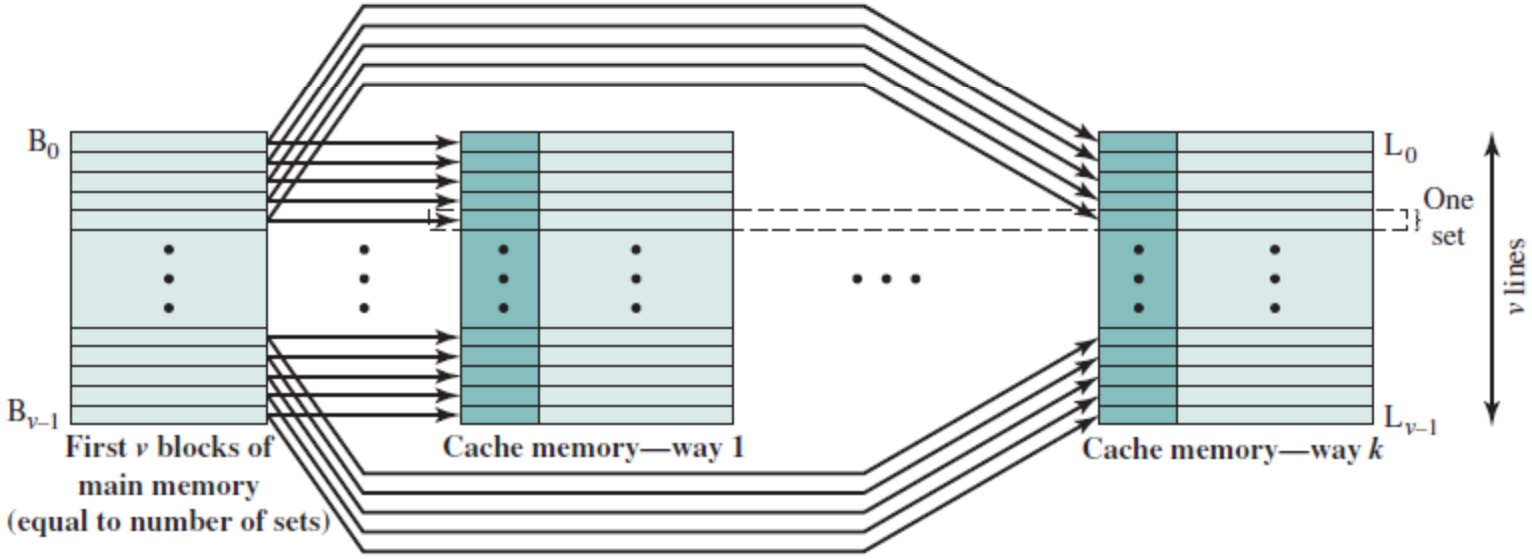
วิธีการนี้เรียกว่า K way set associative mapping โดย main memory block ที่ Bj จะโหลดไปยัง cache ใน set ที่ i (ตามสมการหาหมายเลข Set) ส่วนจะอยู่ใน Line ไหนใน Set นั้นขึ้นกับว่า Line ไหนว่างอยู่ ดังแสดงแนวคิดนี้ในรูปที่ 4.10



v associative-mapped cache

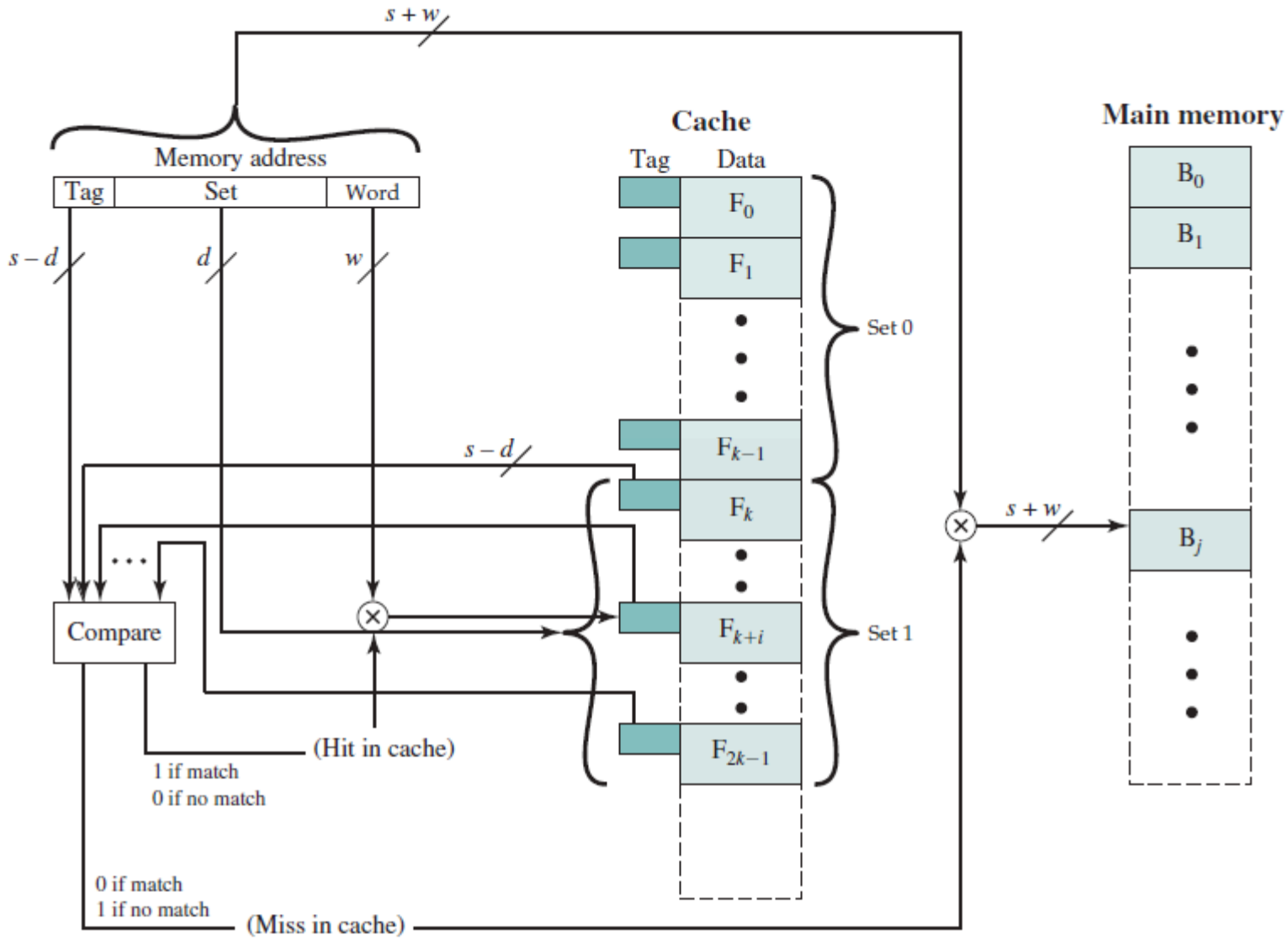
รูปที่ 4.10 Mapping from Main Memory to Cache: *k-Way Set Associative*

จากรูปที่ 4.10 เป็น set associative cache ซึ่งยังสามารถสร้างเป็นแบบ K direct mapping cache ดังรูปที่ 4.11 โดยแต่ละกลุ่มของ cache จะเรียกว่า way จากรูปมีทั้งหมด K way และในแต่ละกลุ่มจะมีทั้งหมด V line ซึ่งวิธีการนี้จะให้ K มีค่าต่ำ (small degree of associative)



K direct – mapped caches

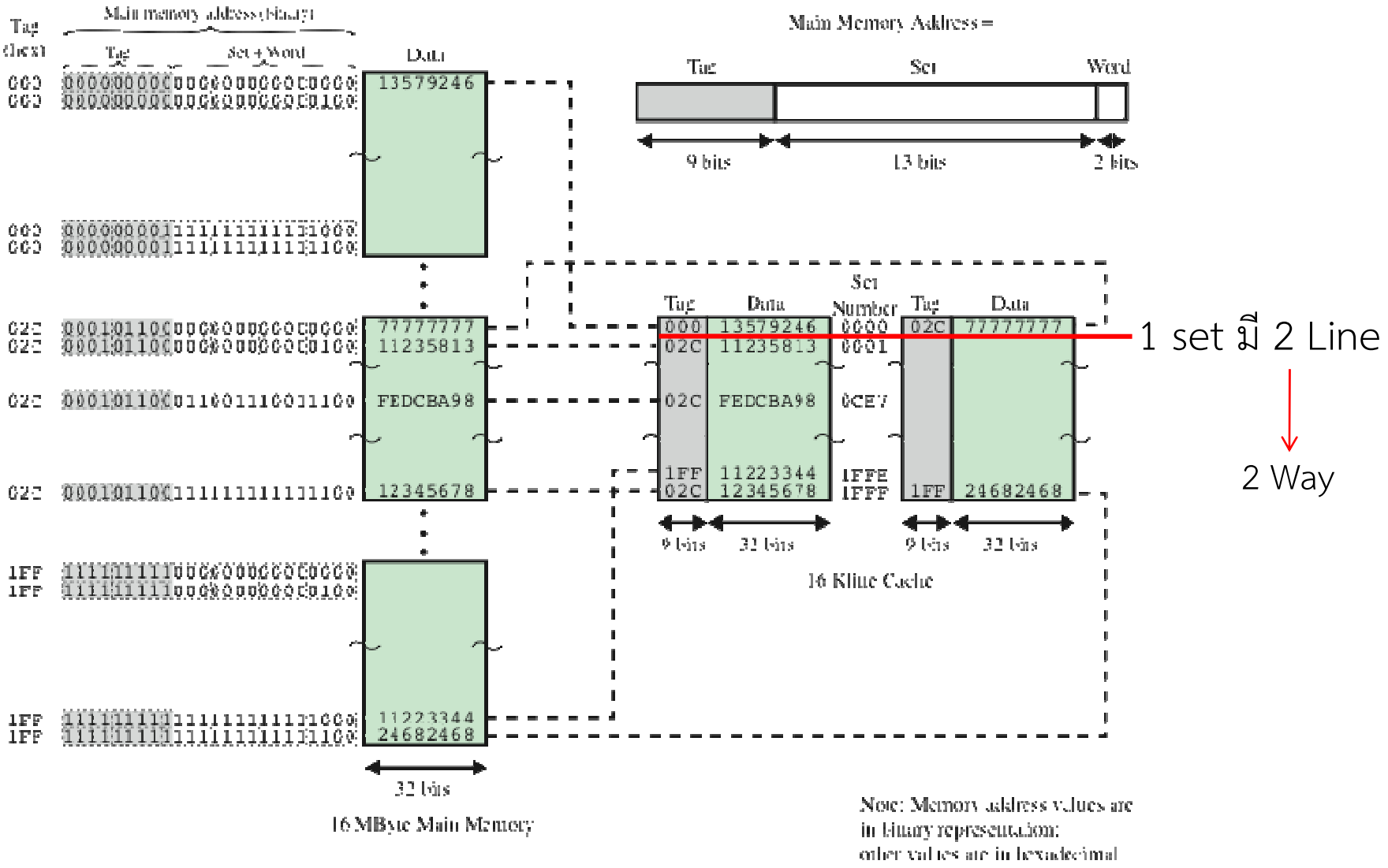
รูปที่ 4.11 Mapping from Main Memory to Cache: *k-Way Set Associative*



รูปที่ 4.12 K- Way Set Associate Cache Organization

- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s$
- Number of lines in set = k
- Number of sets = $v = 2^d$
- Number of lines in cache = $m = kv = k \times 2^d$
- Size of cache = $k \times 2^{d+w}$ words or bytes
- Size of tag = $(s - d)$ bits

จากรูปที่ 4.12 เมื่อ Processor ส่งแอดเดรสของหน่วยความจำที่ต้องการอ้างถึงออกมา ในส่วนของ cache logic จะตรวจจับในส่วนของบิตที่กำหนด Set เพื่อตรวจสอบว่าอ้างหน่วยความจำ Set ใด จากนั้นก็ไปตรวจสอบ Set นั้นใน Cache ว่ามี Line ใดที่มีหมายเลข Tag ที่ตรงกับ Processor อ้างถึงหรือไม่ถ้าไม่มีก็คือเกิด miss ต้องไปติดต่อเข้าถึง Main memory โดยตรงเพื่อโหลดข้อมูลที่ต้องการส่งไปยัง Processor และเก็บใน Cache แต่ถ้ากรณีพบใน Cache ก็จะโหลดข้อมูลส่งไปยัง Processor ทันที



รูปที่ 4.13 2- Way Set Associate Mapping

รูปนี้ชี้ชัดเพิ่มเติมการแสดงค่า Set number ซึ่งทั้ง 3 ตำแหน่งมี Set number ตรงกันแต่ Tag ต่างกัน ดังนั้นเมื่อโหลดไปยัง Cache จะอยู่ใน Set number เดียวกันซึ่งมีแค่ 2 Line เท่านั้น

รูปที่ 4.14 Memory block Set Number (2- Way Set Associate Mapping)

ใน case พิเศษเฉพาะที่จะเกิดขึ้นกับ K way Set Associative mapping อาจจะได้แก่

- **กรณี *v=m , k=1***

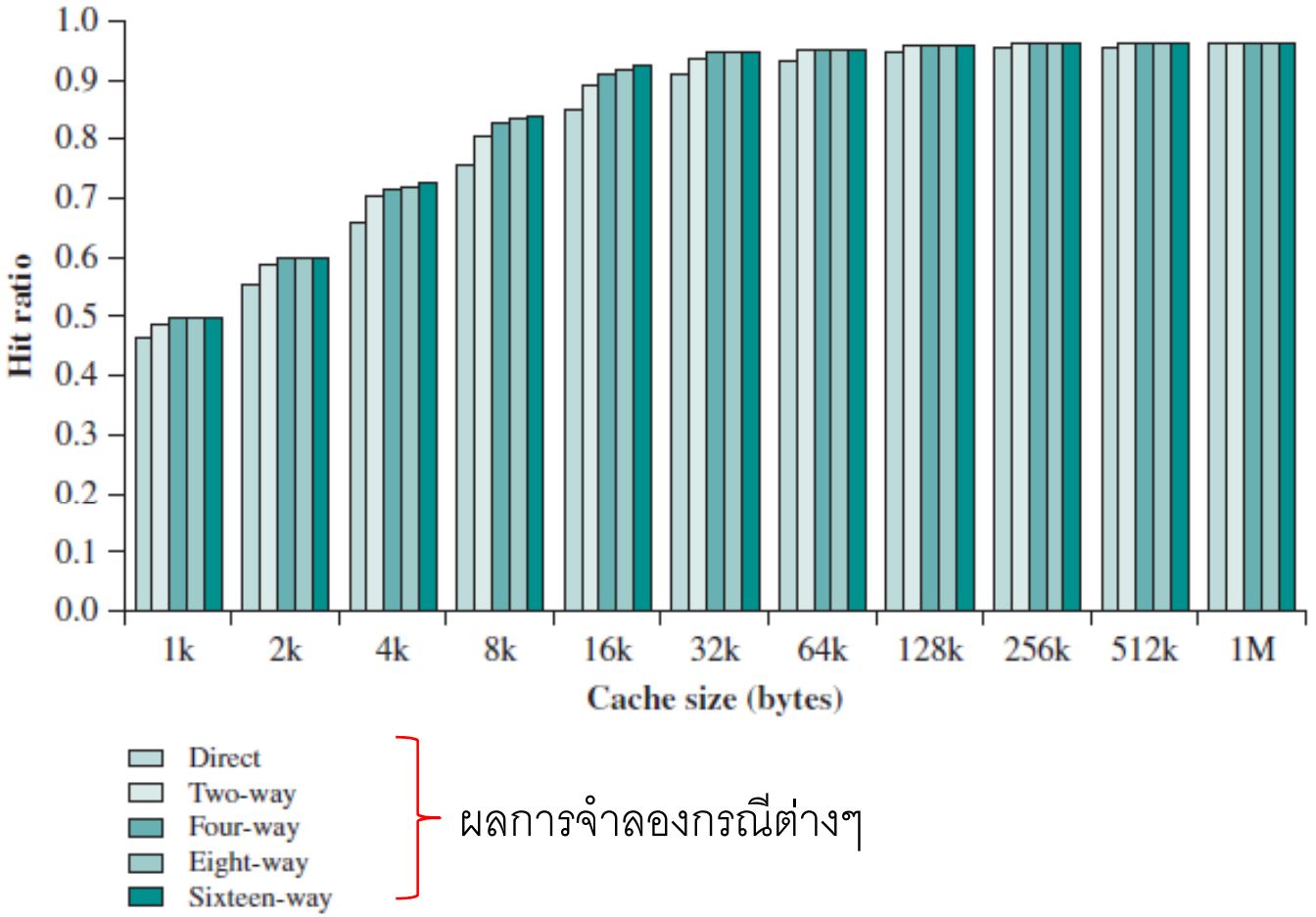
นั่นคือแบ่งกลุ่มของ Cache หรือ Set เท่ากับจำนวน Line ของ Cache

และ *k* = จำนวน Line ในแต่ละกลุ่มของ Cache มีค่าเท่ากับ 1 นั่นคือการ mapping จะกลายเป็นแบบ **Direct Cache Mapping**

- **กรณี *v= 1, k= m***

นั่นคือแบ่ง Cache เป็น 1 กลุ่ม(*v*=1) และในแต่ละกลุ่มมีจำนวน Line *k* เท่ากับจำนวน Line ของ Cache *m* ซึ่งนั่นก็คือเป็น **Cache Associative Mapping**

กรณีที่ใช้งานกันได้แก่ จำนวน Set *v* = 2 และจำนวน Line ต่อ Set *k* = 2 ซึ่งมี Hit ratio มากกว่ากรณี Direct Mapping สำหรับกรณี 4 way set Associative ให้ผลการทำงานดีที่สุดโดยเพิ่มต้นทุนเล็กน้อย แต่ถ้าเพิ่มไปมากกว่านี้แทบจะไม่มีผลมากนัก พิจารณาได้จากรูปที่ 4.15 ซึ่งเปรียบเทียบวิธีการต่างๆเมื่อมีการเปลี่ยนขนาด Cache



รูปที่ 4.15 Varying Associativity over cache size

การเพิ่มขนาดของ Cache เกิน 32kB ไม่ได้ส่งในเรื่องสมรรถนะระบบมากนัก

Replacement Algorithm

- เมื่อ cache ได้เก็บข้อมูลเอาไว้แล้ว และมีข้อมูล block ใหม่ที่ต้องการเก็บใน cache จะทำให้ข้อมูลเก่าที่เก็บไว้จะต้องถูกแทนที่
- สำหรับ direct mapping เป็นการเก็บข้อมูลแบบตรง ตาม line ที่กำหนดดังนั้นจึงไม่มีทางเลือกอื่นนอกจากทับข้อมูลเก่าเลย
- สำหรับ associative และ set-associative นั้นมีความจำเป็นที่จะต้องใช้อัลกอริธึมในการแทนที่ข้อมูล
- เพื่อความรวดเร็วในการทำงาน อัลกอริธึมนี้จะทำงานในรูปแบบของฮาร์ดแวร์

- **Least recently used (LRU)**
  - ถือว่าเป็นอัลกอริธึมที่ใช้งานกันอย่างแพร่หลายที่สุด
  - มีการประยุกต์ใช้งาน USE bit ถ้า block ไหนใน set ถูกใช้งานจะตั้งค่า USE bit เป็น 1 และ block ของ set เดียวกันที่เหลือจะตั้งค่า USE bit เป็น 0
  - เมื่อมี block ข้อมูลใหม่ จะนำไปแทนที่ข้อมูลของ block ใน set ที่มี USE bit เป็น 0
- **First-in First-out (FIFO)**
  - แทนที่ block ใน set ตัวที่อยู่ใน cache มานานที่สุด
  - สามารถทำได้ง่าย โดยใช้เทคนิค round-robin หรือ circular buffer
- **Least frequently used (LFU)**
  - แทนที่ block ใน set ตัวที่มีการอ้างอิงใช้งานน้อยที่สุด
  - สามารถทำได้ โดยใช้ counter ในแต่ละ line ของ cache
- **Random**
  - สุ่ม Block ที่จะถูกแทนที่เลย โดยไม่สนใจสิ่งอื่น

Write Policy

- เมื่อ Block ของหน่วยความจำอยู่ใน Cache และกำลังจะถูกแทนที่ มี 2 กรณีที่ต้องคำนึงถึง
  - ถ้าข้อมูลใน Block เก่าไม่ได้ถูกเปลี่ยนค่า Block ใหม่สามารถเข้ามาแทนที่ได้เลย
  - แต่ถ้ามีการเขียนอย่างน้อย 1 ครั้งใน Block เก่า จำเป็นจะต้อง update ค่านั้นกลับไปยังหน่วยความจำหลัก ก่อนที่จะนำ Block ใหม่เข้ามาแทนที่
- เพราะฉะนั้นปัญหาที่จะเกิดขึ้นตามมาคือ
  - ถ้ามีอุปกรณ์มากกว่า 1 อุปกรณ์ต้องการข้อมูลจากหน่วยความจำหลัก
  - ปัญหาที่ซับซ้อนมากกว่านั้นคือเมื่อ Processor หลายตัวที่ต่อบน bus เดียวกันและมี cache เป็นของตัวเอง ถ้า Processor หนึ่งมีการแก้ไขใน cache ของตัวเอง อาจจะทำให้ Processor อีกตัวหนึ่งมีข้อมูลใน cache ที่ไม่ตรงกัน

วิธีการที่ใช้เมื่อมีการแก้ไขข้อมูลใน Cache (Write Policy)

- **Write through**
  - เป็นเทคนิคที่ง่ายที่สุด
  - ทุกๆ การดำเนินการที่เป็นการ write จะเขียนทั้งใน cache และ หน่วยความจำหลัก
  - ข้อเสียหลักของเทคนิคนี้คือ จะทำให้เกิดการติดต่อกับหน่วยความจำหลักค่อนข้างเยอะ และทำให้เกิดปัญหาคอขวด
- **Write back**
  - พยายามลดปัญหาการติดต่อกับหน่วยความจำหลัก
  - มีการประยุกต์ใช้ Dirty bit ในแต่ละ line ของ cache
  - ถ้ามีการแทนที่ block จะตรวจสอบ dirty bit ถ้ามีถูก set จะเขียนข้อมูลกลับลงหน่วยความจำหลัก ก่อนที่จะแทนที่ block เก่าด้วย block ใหม่

ใน bus ถ้ามี processor มากกว่า 1 ตัว แต่ละตัวมี cache ของตนเองและใช้งาน Main memory ร่วมกัน จะมีปัญหาใหม่เกิดขึ้นมา คือ ถ้าข้อมูลใน cache มีการเปลี่ยนแปลงค่า ไม่เพียงแต่ข้อมูลในหน่วยความจำหลักที่ต้องจัดการ แต่ต้องจัดการกับข้อมูลใน cache ของ processor อื่นด้วย (ถ้าเป็นหน่วยความจำนั้นมีการโหลดไปยัง Cache อื่นด้วย) ถึงแม้จะใช้วีการแบบ Write Through แล้วก็ตามข้อมูลใน Cache อื่นอาจไม่ถูกต้องตรงกันก็ได้ ซึ่งระบบที่มีการป้องกันปัญหานี้จะเรียกว่ามีการรักษา Cache Coherency ซึ่งอาจมีวิธีการต่างๆดังนี้

- **Bus watching with write through**

ตัวควบคุม cache จะคอยตรวจสอบค่าแอดเดรสเพื่อตรวจจับการเขียนข้อมูลไปยังหน่วยความจำโดยอุปกรณ์อื่น (Bus Master) และกามีการเขียนข้อมูลในตำแหน่ง Shared Memory ซึ่งก็อยู่ใน Cache ด้วยก็จะทำให้ข้อมูลใน Cache นั้นไม่ถูกต้องเป็นปัจจุบัน ซึ่ง Cache Controller ก็จะรับรู้ว่าข้อมูลใน Cache ตำแหน่งนั้นไม่ถูกต้องแล้ว ซึ่งวิธีการนี้จะขึ้นกับนโยบาย Write Through ของ Cache Controller ทุกตัวในระบบ

Hardware transparency

วิธีนี้คือเพิ่ม hardware ในการทำงาน ถ้าข้อมูลหน่วยของความจำหลักที่โหลดไปยัง Cache ของProcessor มีการ update ข้อมูลใหม่ Hardware นี้จะไปทำการ update ข้อมูลนี้ที่อยู่ใน Main memory ด้วยและรวมถึงจะต้อง Update ในส่วน cache อื่นๆที่เกี่ยวข้องทั้งหมด

Non-cacheable memory

วิธีการนี้คือกำหนดให้ main memory บางส่วนเป็น Shared memory คือใช้งานร่วมกันระหว่าง Processor และข้อมูลส่วนนี้จะไม่มีการ copy ไปยัง cache วิธีการ Shared memory ถูกใช้เพื่อแก้ไขปัญหาการต้องไปตาม Update ข้อมูลใน Cache ของ Processor หลายๆตัว

Line Size

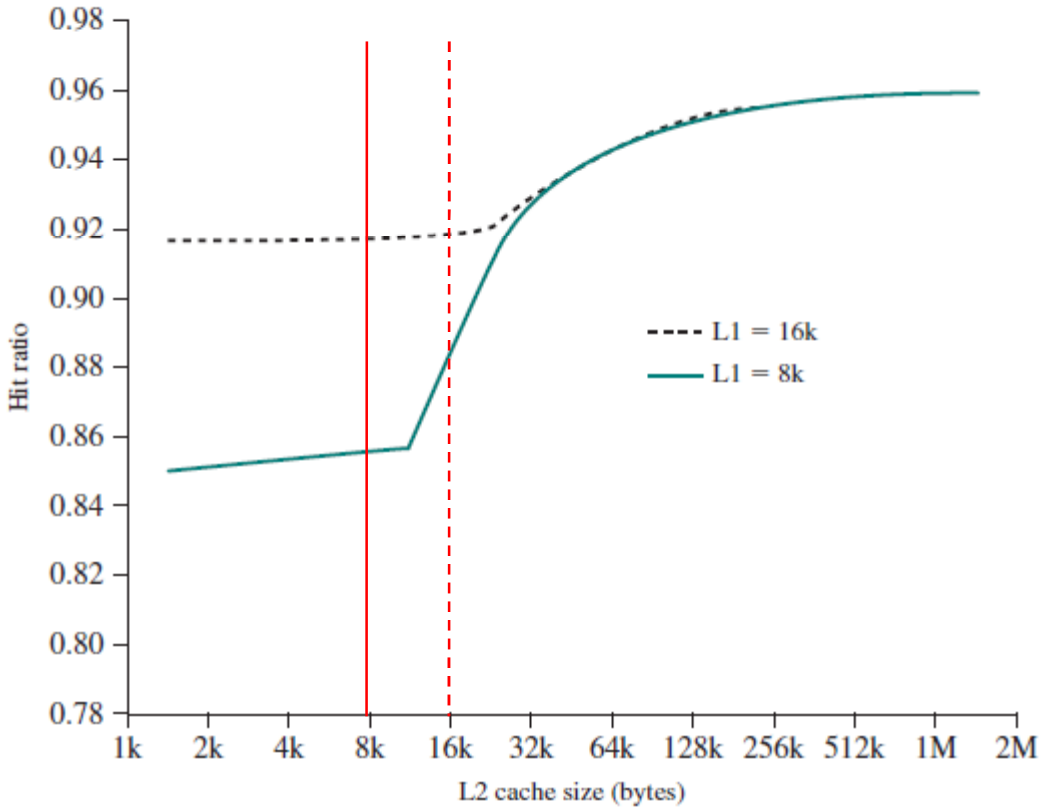
- เมื่อ Block ข้อมูลเข้ามาเก็บใน Cache จะเก็บเป็นจำนวนเท่ากับขนาดของ line ทำให้มีข้อมูลหรือชุดคำสั่งที่ติดกัน ถูกดึงเข้ามาด้วย
- เมื่อขนาดของ line ใหญ่ขึ้น จะเพิ่มอัตราส่วนของ cache hit มากขึ้น จากหลักการของ locality
- แต่อย่างไรก็ตามอัตราส่วนของ cache hit จะลดลงถ้า line ใหญ่เกินไป และความน่าจะเป็นที่ดึงข้อมูลที่ไม่ได้ใช้งานเข้ามามีเยอะมากขึ้น
- มีผลกระทบอยู่ 2 อย่างที่จะเกิดขึ้น
  - ถ้า line มีขนาดใหญ่มากจะทำให้จำนวน line ลดลง
  - ถ้า line ใหญ่มากไปจะทำให้ข้อมูลที่ต้องใช้งานอยู่ไกลเกินไป
- จากงานวิจัยพบว่าขนาดของ line ที่ 8 – 64 bytes เหมาะสมกับการทำงานทั่วไป และสำหรับ HPC จะใช้ line ขนาด 64 – 128 bytes

Number of Cache

Multi Level Cache

- ความหนาแน่นของ gate ที่เพิ่มขึ้นใน chip สามารถทำให้สามารถทำ cache ไว้ภายใน chip ของ processor ได้
- การที่ cache อยู่ภายใน chip จะทำให้ลดการใช้งาน bus ภายนอกและเพิ่มความเร็วให้การทำงานของ processor
  - ถ้าพบชุดคำสั่งใน cache จะทำให้ไม่จำเป็นต้องใช้งาน bus เพื่อดึงชุดคำสั่งจากหน่วยความจำหลัก
  - การดึงข้อมูลจาก cache ภายใน processor จะเร็วมากแทบจะไม่ต้องรอสัญญาณ clock เลย
  - เมื่อ bus ไม่ได้ถูกใช้งานจะทำให้ bus สามารถให้บริการกับอุปกรณ์อื่นแทนได้
- Two-level cache :
  - Internal cache ออกแบบมาอาจจะเรียกว่า level 1 (L1)
  - External cache ออกแบบมาทำงานถัดจาก level 1 เรียกว่า level 2 (L2)

รูปนี้แสดงให้เห็นความสัมพันธ์ระหว่างขนาด cache ใน Level1 และ Level 2 ที่มีผลต่อ hit ratio เช่นกรณี L1 kB ขนาด L2 ที่ทำให้ hit ratio สูงขึ้นอย่างมีนัยสำคัญก็คือ L2 ต้องมีขนาดมากกว่า 16 kB (จากรูปใกล้ๆ 32 kB)



รูปที่ 4.16 Total Hit Ratio (L1 and L2) for 8-Kbyte and 16-Kbyte L1

Unified & Split Cache

- ข้อดีของ Unified cache
  - มีอัตราการ hit ที่สูง
    - เนื่องจากเป็นการรวมกันระหว่างชุดคำสั่งและชุดข้อมูลที่ต้องการจะใช้งาน
    - การออกแบบและพัฒนาทำกับ cache เพียงแค่ตัวเดียว
- Split cache จะมี
  - Instruction cache สำหรับเก็บชุดคำสั่ง
  - Data cache สำหรับเก็บชุดข้อมูล
  - โดยทั้ง 2 cache นี้จะอยู่ในระดับเดียวกัน ปกติจะเป็น L1 ทั้งคู่
- ข้อดีของ Split cache
  - ลดความหนาแน่นของการใช้งาน cache ระหว่าง fetch, decode และ execute
    - ซึ่งจะสำคัญมากกับการทาง Pipelining
- แนวโน้มการออกแบบจะใช้ Split cache สำหรับ L1 และ Unified cache สำหรับ level ที่สูงกว่า

ตัวอย่าง Cache ใน Intel Processor และ ARM Processor

Table 4.4 Intel Cache Evolution

Problem	Solution	Processor on Which Feature First Appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip.	Add external L2 cache using faster technology than main memory.	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

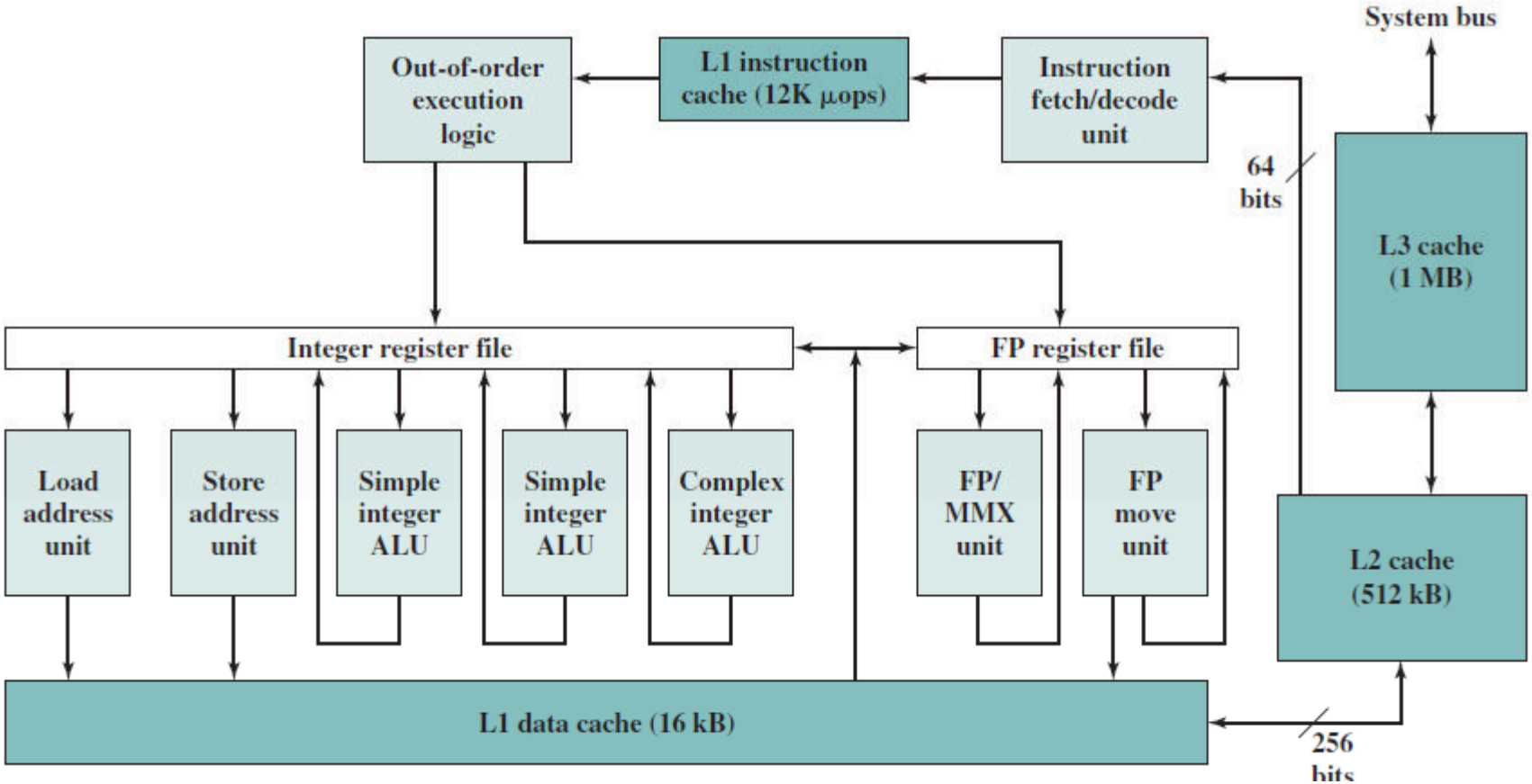


Table 4.5 Pentium 4 Cache Operating Modes

Control Bits		Operating Mode		
CD	NW	Cache Fills	Write Throughs	Invalidates
0	0	Enabled	Enabled	Enabled
1	0	Disabled	Enabled	Enabled
1	1	Disabled	Disabled	Disabled

Note: CD = 0; NW = 1 is an invalid combination.

Table 4.6 ARM Cache Features

Core	Cache Type	Cache Size (kB)	Cache Line Size (words)	Associativity	Location	Write Buffer Size (words)
ARM720T	Unified	8	4	4-way	Logical	8
ARM920T	Split	16/16 D/I	8	64-way	Logical	16
ARM926EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	16
ARM1022E	Split	16/16 D/I	8	64-way	Logical	16
ARM1026EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	8
Intel StrongARM	Split	16/16 D/I	4	32-way	Logical	32
Intel Xscale	Split	32/32 D/I	8	32-way	Logical	32
ARM1136-JF-S	Split	4-64/4-64 D/I	8	4-way	Physical	32

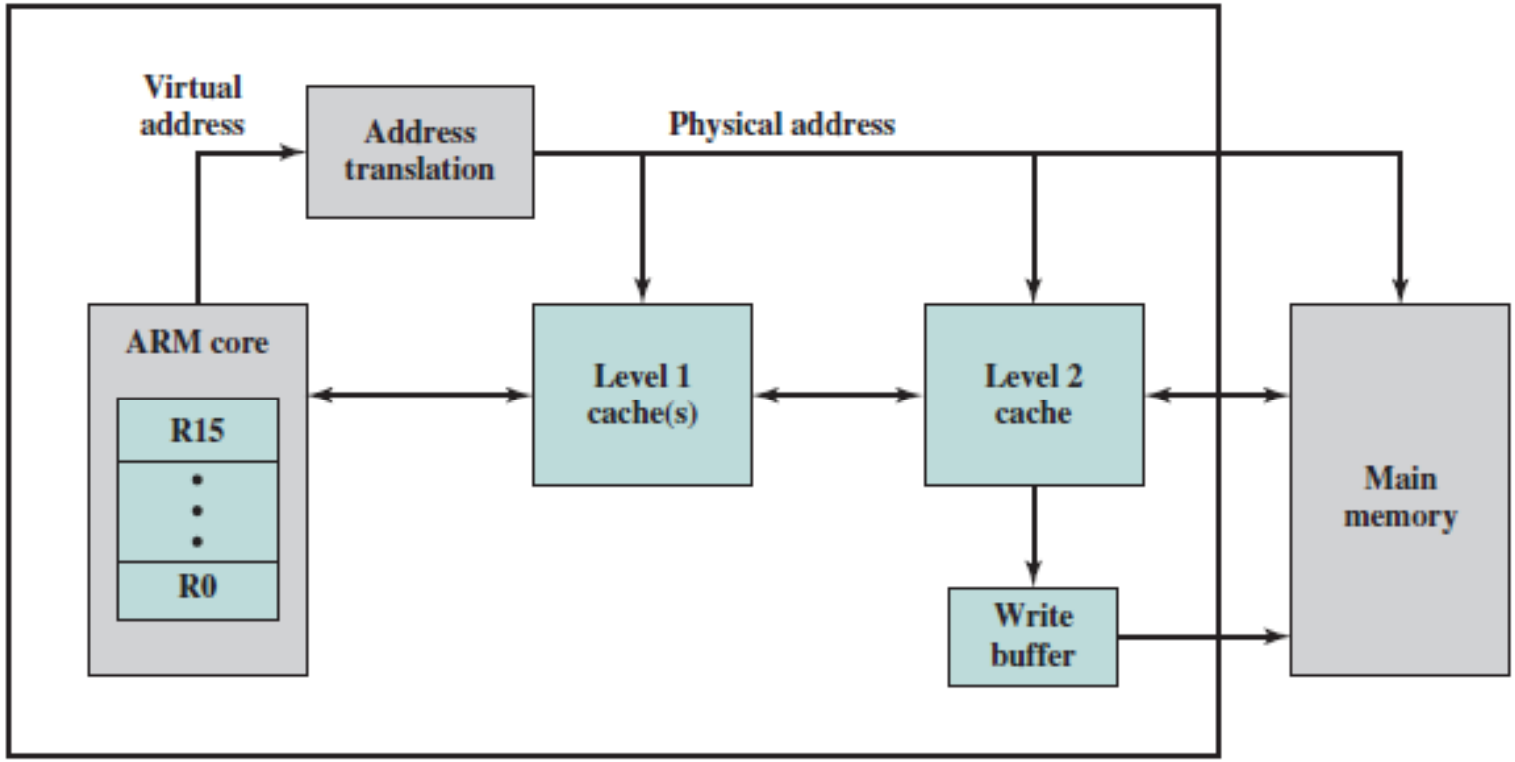


Figure 4.19 ARM Cache and Write Buffer Organization