Chapter 4

Cache Memory

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

Table 4.1
Key Characteristics of Computer Memory Systems

Characteristics of Memory Systems

- Location
  - Refers to whether memory is internal and external to the computer
  - Internal memory is often equated with main memory
  - Processor requires its own local memory, in the form of registers
  - Cache is another form of internal memory
  - External memory consists of peripheral storage devices that are accessible to the processor via I/O controllers
- Capacity
  - Memory is typically expressed in terms of bytes
- Unit of transfer
  - For internal memory the unit of transfer is equal to the number of electrical lines into and out of the memory module

참고) https://bournetocode.com/projects/GCSE_Computing_Fundamentals/pages/3-4-4-sys_arc.html

Location of memory

- Processor requires its own local memory(register)
- Internal memory : the main memory
- External memory : memory on peripheral storage devices (disk, tape, etc.)

Memory Capacity

- Memory capacity of internal memory is typically expressed in terms of bytes or words
  - Common word lengths are 8, 16, and 32 bits
- External memory capacity is typically expressed in terms of bytes

Unit of Transfer of Memory

- For internal memory,
  - The unit of transfer is equal to the word length, but is often larger, such as 64, 128, or 256 bits
  - Usually governed by data bus width
- For external memory,
  - The unit of transfer is usually a block which is much larger than a word
- For addressable unit,
  - The unit of transfer is the smallest location which can be uniquely addressed
  - Word internally

Access Methods (1)

- Sequential access
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
  - e.g. tape
- Direct access
  - Individual blocks have unique address
  - Access is by jumping to vicinity plus sequential search
  - Access time depends on location and previous location
  - e.g. disk

Access Methods (2)

- Random access
  - Individual addresses identify locations exactly
  - Access time is independent of location or previous access
  - e.g. RAM
- Associative
  - Data is located by a comparison with contents of a portion of the store
  - Access time is independent of location or previous access
  - e.g. cache

Access Methods (3) : Summary

Sequential access

- Memory is organized into units of data called records
- Access must be made in a specific linear sequence
- Access time is variable

Direct access

- Involves a shared read-write mechanism
- Individual blocks or records have a unique address based on physical location
- Access time is variable

Random access

- Each addressable location in memory has a unique, physically wired-in addressing mechanism
- The time to access a given location is independent of the sequence of prior accesses and is constant
- Any location can be selected at random and directly addressed and accessed
- Main memory and some cache systems are random access

Associative

- A word is retrieved based on a portion of its contents rather than its address
- Each location has its own addressing mechanism and retrieval time is constant independent of location or prior access patterns
- Cache memories may employ associative access

Performance

- Access time(latency)
  - For random access memory, this is the time it takes to perform a read or write operation
    - That is, the time between presenting the address and getting the valid data
  - For non-random access memory, access time is the time it takes to position the read-write mechanism at the desired location
- Memory Cycle time
  - Memory cycle time = access time + any additional time required before a second access can commence
  - Additional time ?
    - The time of transient signal dies out on signal lines
    - The time to regenerate data if they are read destructively
  - Memory cycle time >= Access time

Performance

- Transfer Rate (Bandwidth)
  - Rate at which data can be transferred into or out of a memory unit
  - For random access memory,
    - Transfer rate = (1/cycle time)
  - For non-random access memory,
    - the following relationship holds
    - $T_N = T_A + (N/R)$

      T_N : average time to read or write N bits

      T_A : average access time

      N : number of bits

      R : transfer rate (bps)
    - $R = N/(T_N - T_A)$

  Transfer Rate =

  Number of bits / (Avg Time to R/W N bits - Access Time)

Physical Types

- Semiconductor
  - RAM
- Magnetic
  - Disk & Tape
- Optical
  - CD & DVD

Memory

- The most common forms are:
  - Semiconductor memory
  - Magnetic surface memory
  - Optical
  - Magneto-optical
- Several physical characteristics of data storage are important:
  - Volatile memory
    - Information decays naturally or is lost when electrical power is switched off
  - Nonvolatile memory
    - Once recorded, information remains without deterioration until deliberately changed
    - No electrical power is needed to retain information
  - Magnetic-surface memories
    - Are nonvolatile
  - Semiconductor memory
    - May be either volatile or nonvolatile
  - Nonerasable memory
    - Cannot be altered, except by destroying the storage unit
    - Semiconductor memory of this type is known as read-only memory (ROM)
- For random-access memory the organization is a key design issue
  - Organization refers to the physical arrangement of bits to form words

Memory Hierarchy

- Design constraints on a computer's memory can be summed up by three questions:
  - How much, how fast, how expensive
- There is a trade-off among capacity, access time, and cost
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access time
- The way out of the memory dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy

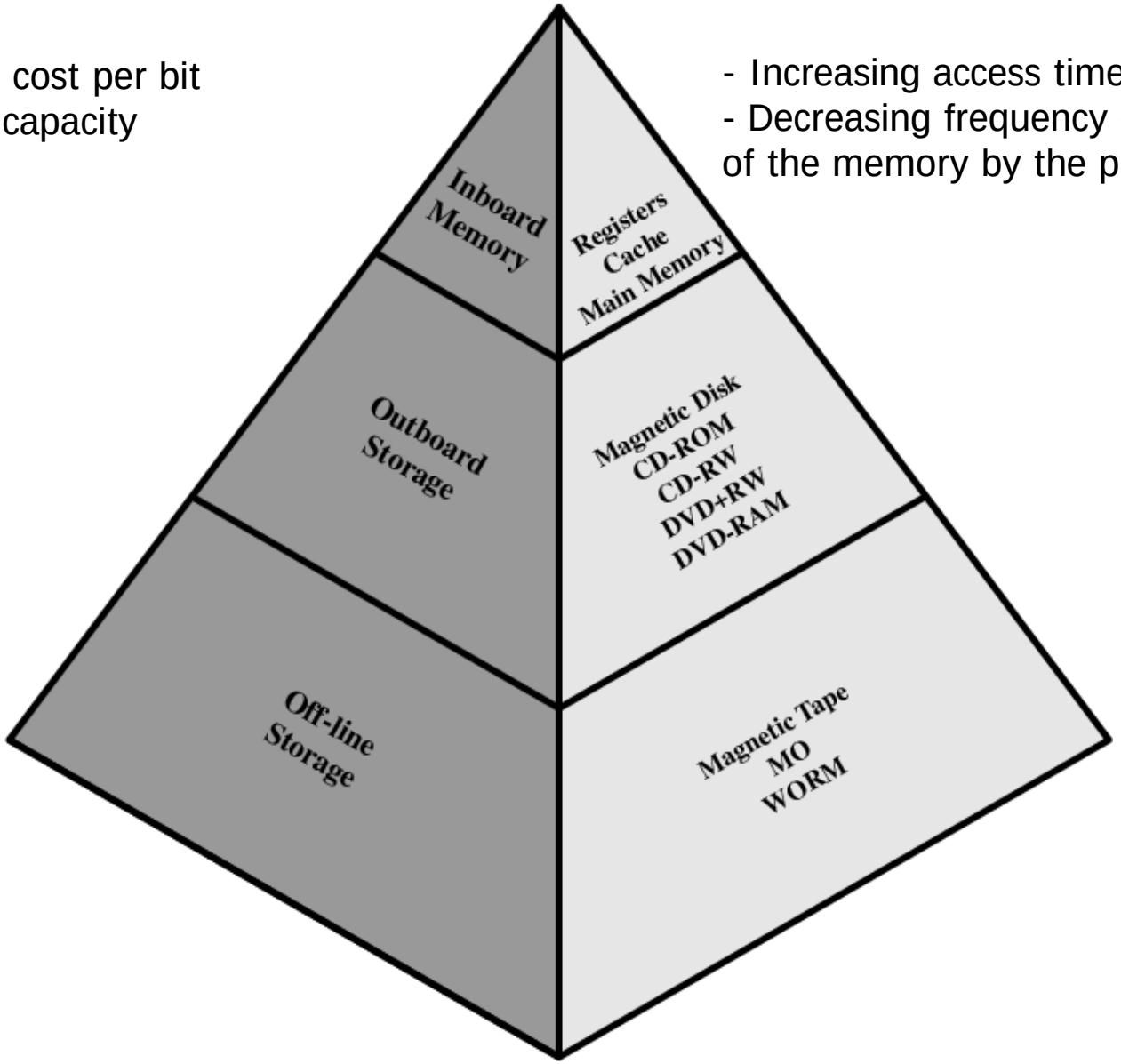
Memory Hierarchy

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - "RAM"
- External memory
  - Backing store

Memory Hierarchy - Diagram

- Decreasing cost per bit
- Increasing capacity

- Increasing access time
- Decreasing frequency of access of the memory by the processor



Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

So do you want fast memory architecture?

- It is possible to build a computer which uses only static RAM (see later)
- This would be very fast
- This would need no cache
- This would cost a very large amount

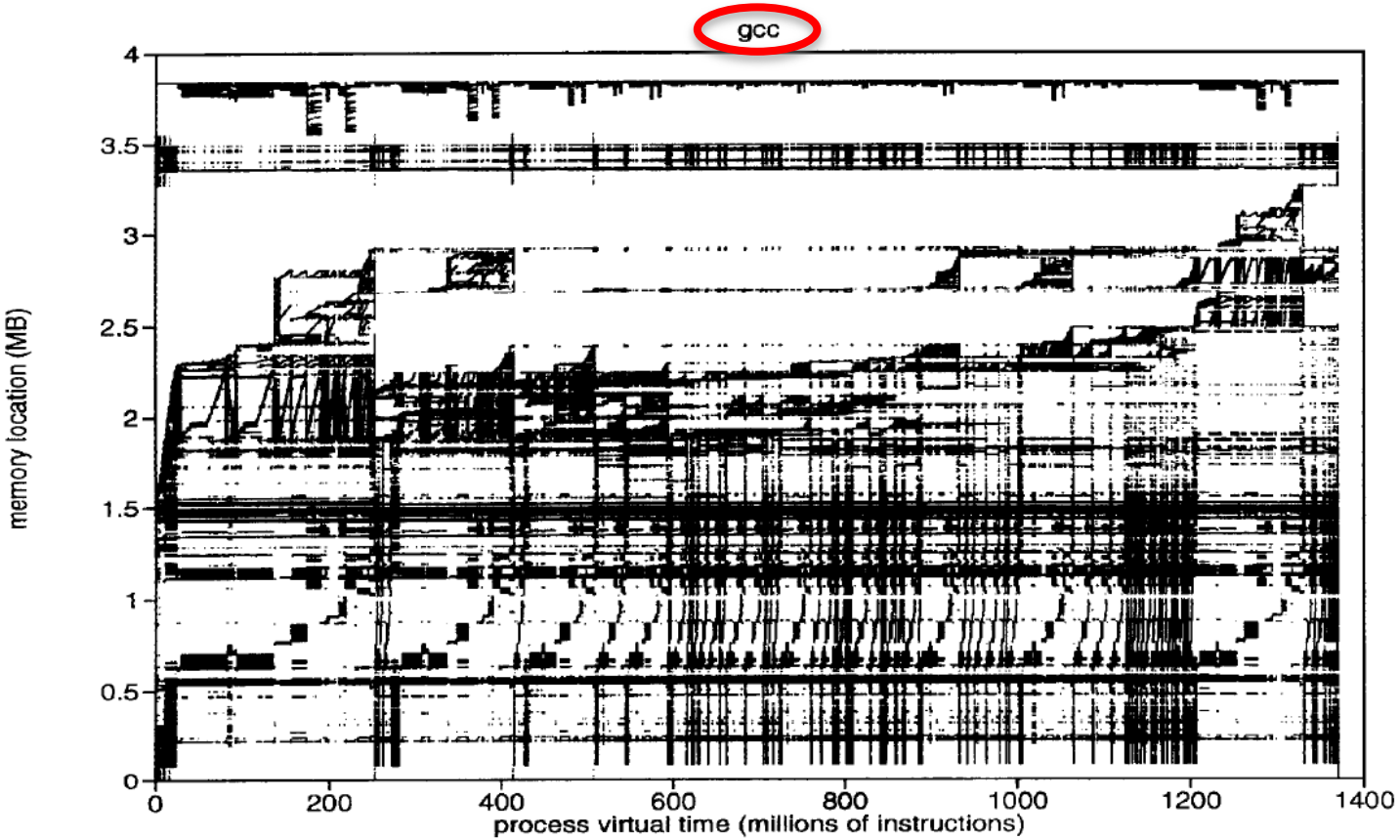
Locality of Reference

- Principle of Locality
  - Programs tend to reuse data & instructions near those they have used recently, or that were recently reference themselves
  - Temporal locality:
    - ✓ Recently referenced items are likely to be referenced in the near future
  - Spatial locality:
    - ✓ items with nearby addresses tend to be referenced close together in time

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

- e.g. loop
  - Data
    - ✓ Spatial locality: Reference a [ ] array elements in succession
    - ✓ Temporal locality: Reference sum in each iteration
  - Instructions
    - Spatial locality: Reference instructions in sequence
    - Temporal locality : Cycle through loop repeatedly

Locality of Reference 예 1



예) gcc compile가 실행될때의 memory reference 특성

그림 참고: Adaptive page replacement based on memory reference behavior by G. Glass, P. Cao, ACM SIGMETRICS 1997, available at https://dl.acm.org/doi/abs/10.1145/258612.258681

Cache

- Small amount of fast memory
- Sits between normal main memory and CPU
- May be located on CPU chip or module

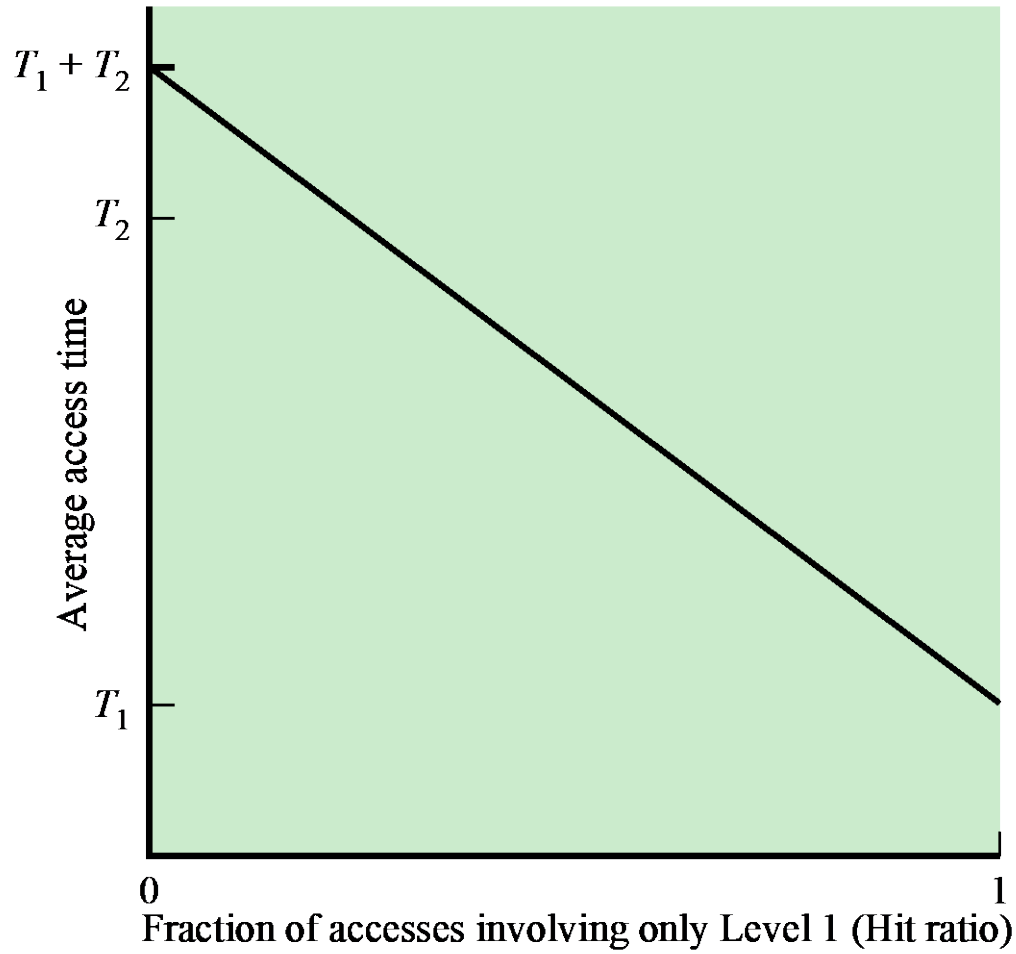
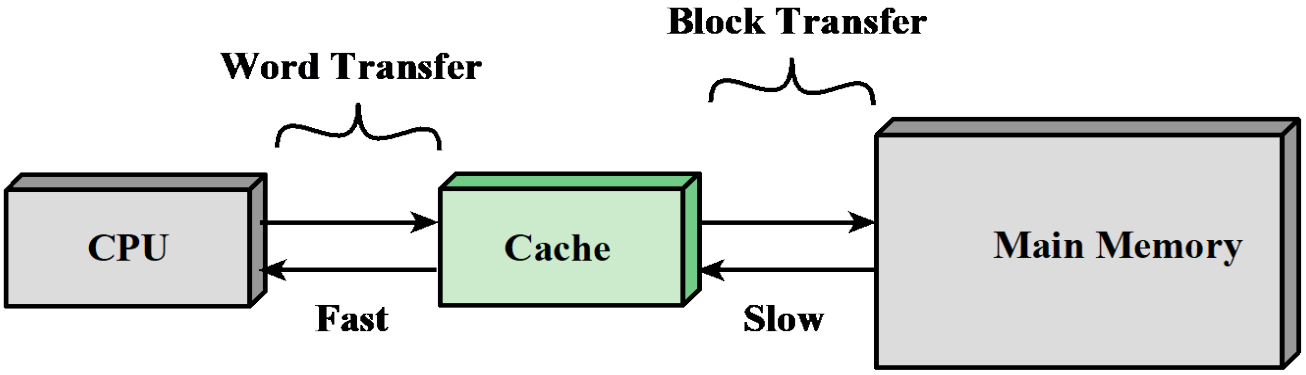


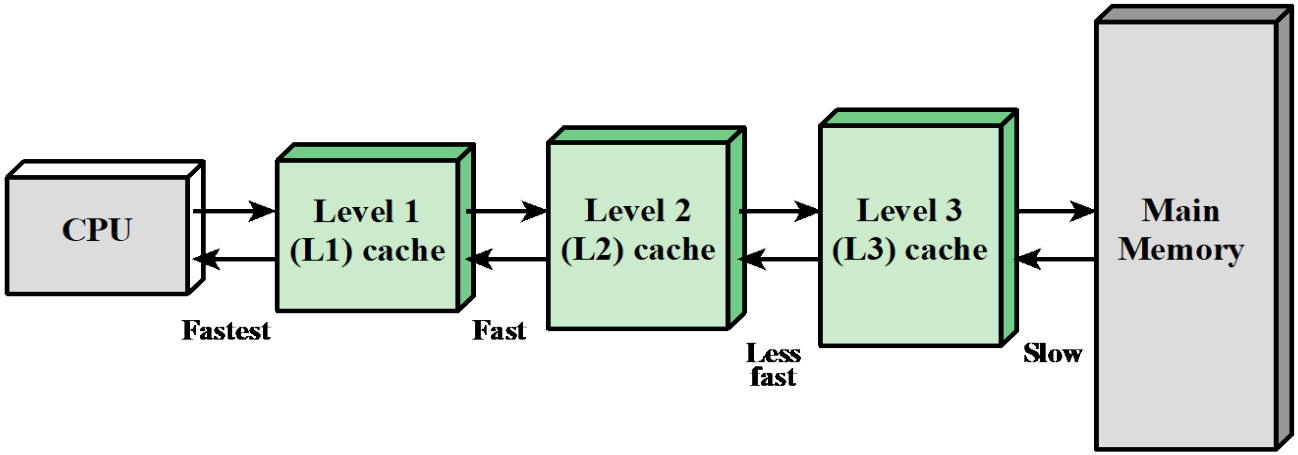
Figure 4.2 Performance of a Simple Two-Level Memory

Memory

- The use of three levels exploits the fact that semiconductor memory comes in a variety of types which differ in speed and cost
- Data are stored more permanently on external mass storage devices
- External, nonvolatile memory is also referred to as **secondary** memory or **auxiliary** memory
- Disk cache
  - A portion of main memory can be used as a buffer to hold data temporarily that is to be read out to disk
  - A few large transfers of data can be used instead of many small transfers of data
  - Data can be retrieved rapidly from the software cache rather than slowly from the disk



(a) Single cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache (slow)
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

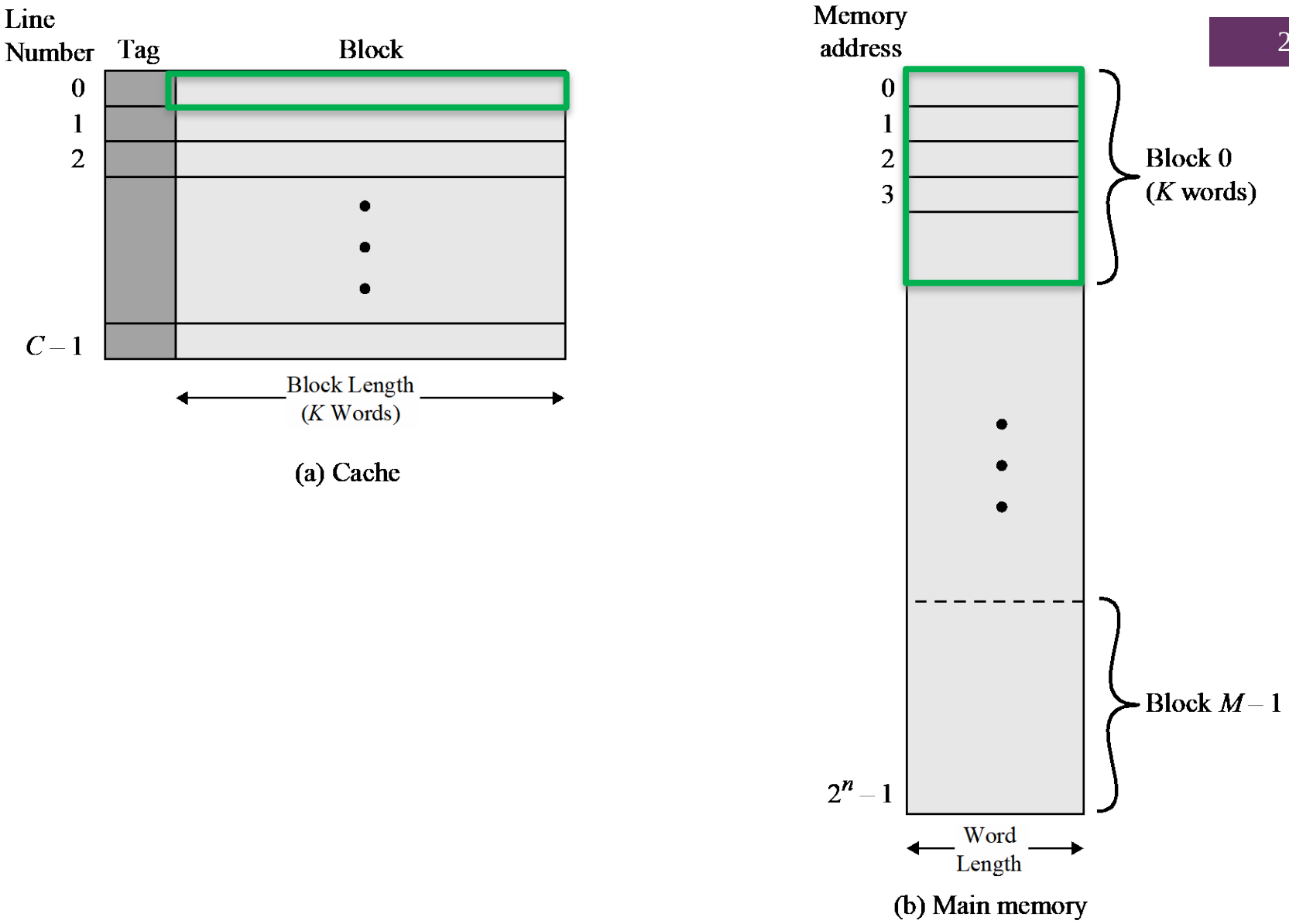


Figure 4.4 Cache/Main-Memory Structure

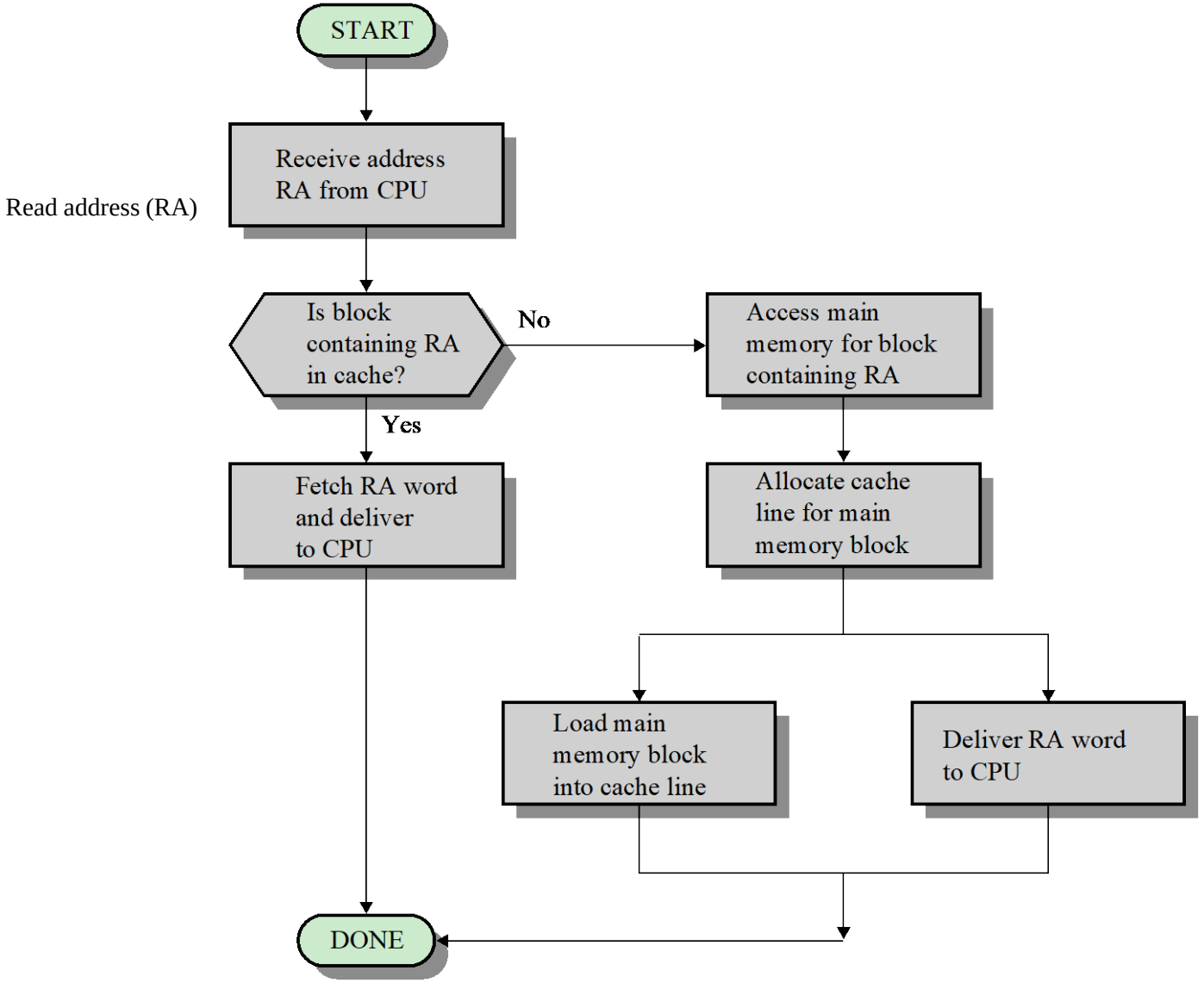


Figure 4.5 Cache Read Operation

Cache Design Parameters

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Size does matter

- Cost
  - More cache is expensive
- Speed
  - More cache is faster (up to a point)

Cf. The larger the cache, the larger the number of gates involved in addressing the cache. The result is that large caches tend to be slightly slower than small ones.

Cache Addressing

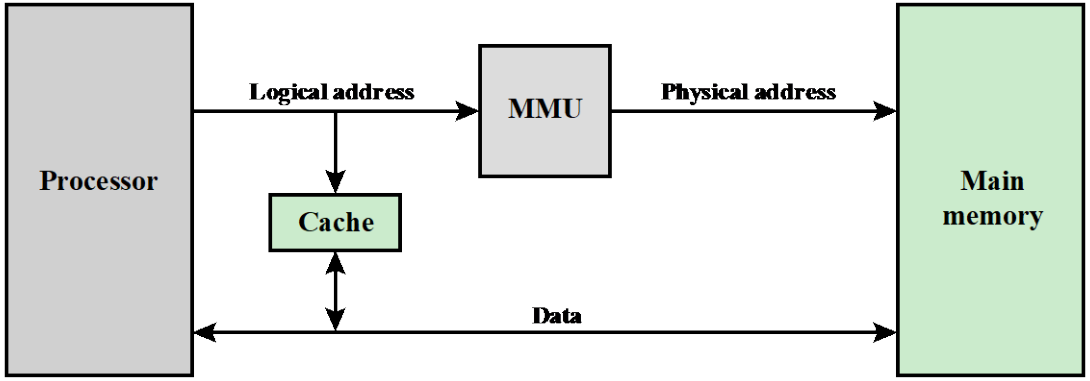
- Where does cache sit?
  - Between processor and virtual memory management unit
  - Between MMU and main memory

- Logical cache (virtual cache) stores data using virtual addresses
  - Stores data using virtual addresses
  - The processor accesses the cache directly, without going through the MMU.
  - Cf. a physical cache stores data using main memory physical addresses
  - Advantage of the logical cache is that cache access faster than for a physical cache, before MMU address translation
  - Disadvantage has to do with the fact that most virtual memory systems supply each application with the same virtual memory address space.
    - That is, each application sees a virtual memory that starts at address 0.
    - Thus, the same virtual address in two different applications refers to two different physical addresses.
    - The cache memory must therefore be completely flushed with each application context switch, or extra bits must be added to each line of the cache to identify with virtual address space this address refers to!

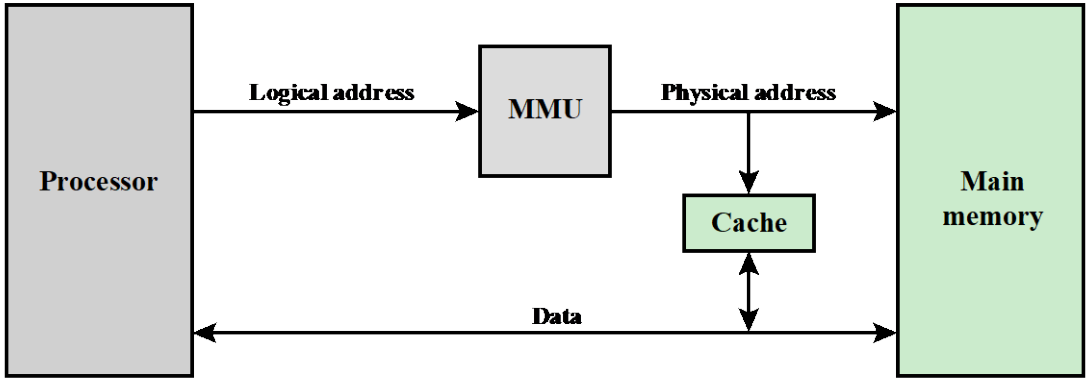
Cache Addresses

Virtual Memory

- **Virtual memory**
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory



(a) Logical Cache



(b) Physical Cache

Figure 4.7 Logical and Physical Caches

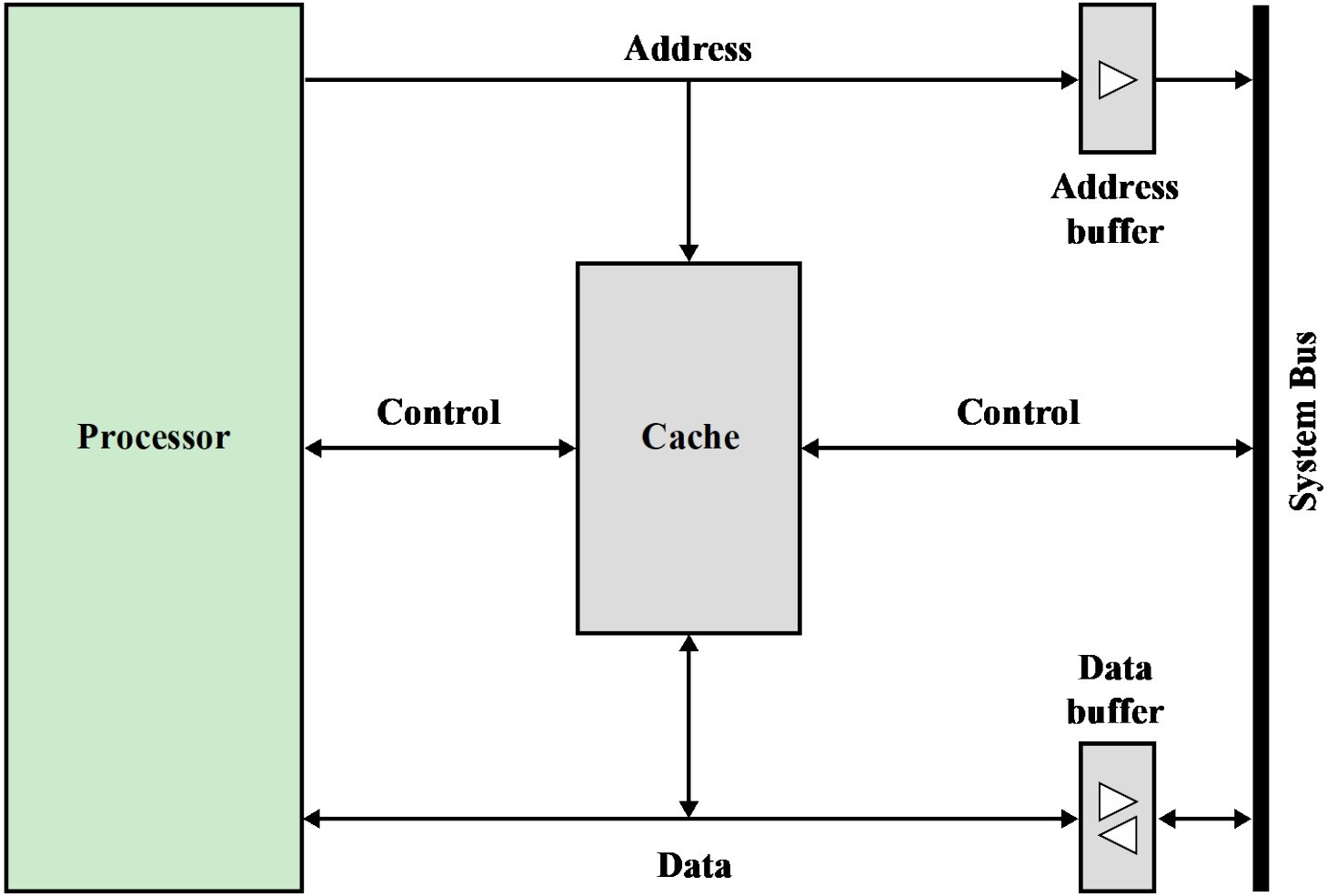


Figure 4.6 Typical Cache Organization

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	

Table 4.2
Elements of Cache Design

Table 4.3

Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 Cachea	L2 cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 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 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	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 MTAb	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	1MB	—
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	Workstaton/ 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

[a] Two values separated by a slash refer to instruction and data caches.

[b] Both caches are instruction only; no data caches.

(Table can be found on page 134 in the textbook.)

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines
- Three techniques can be used:

Direct

- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

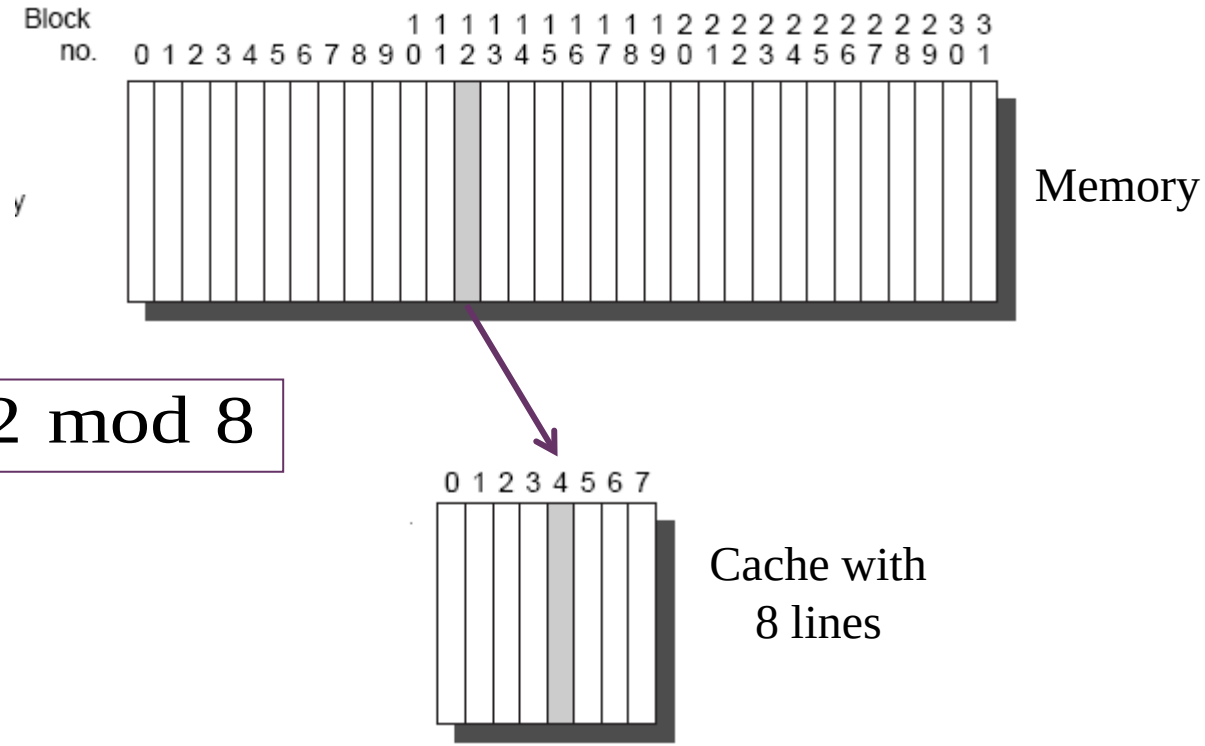
Mapping Function

- For designing the cache, we will use the following specification
  - Cache of 64kByte
  - Cache block of 4 bytes
    - i.e. cache is 16k (2^{14}) lines of 4 bytes
  - 16MBytes main memory
  - 24 bit address
    - (2^{24}=16M)

Mapping fn for "Direct Mapping"

i: Cache line number
j: block number
m : number of lines in the cache

- Since the mapping function of direct mapping is $i = j \bmod m$, we can see the direct mapped cache as following figure



Mapping fn for "Direct Mapping"

- The mapping is expressed as
- i ← j modulo m
  - i : cache line number
  - j : main memory block number
  - m : number of lines in the cache

Cache line (i)	Main Memory blocks assigned
0	0, m, 2m, 3m…2^s-m
1	1,m+1, 2m+1…2^s-m+1
…	…
m-1	m-1, 2m-1,3m-1…2^s-1

Direct Mapping

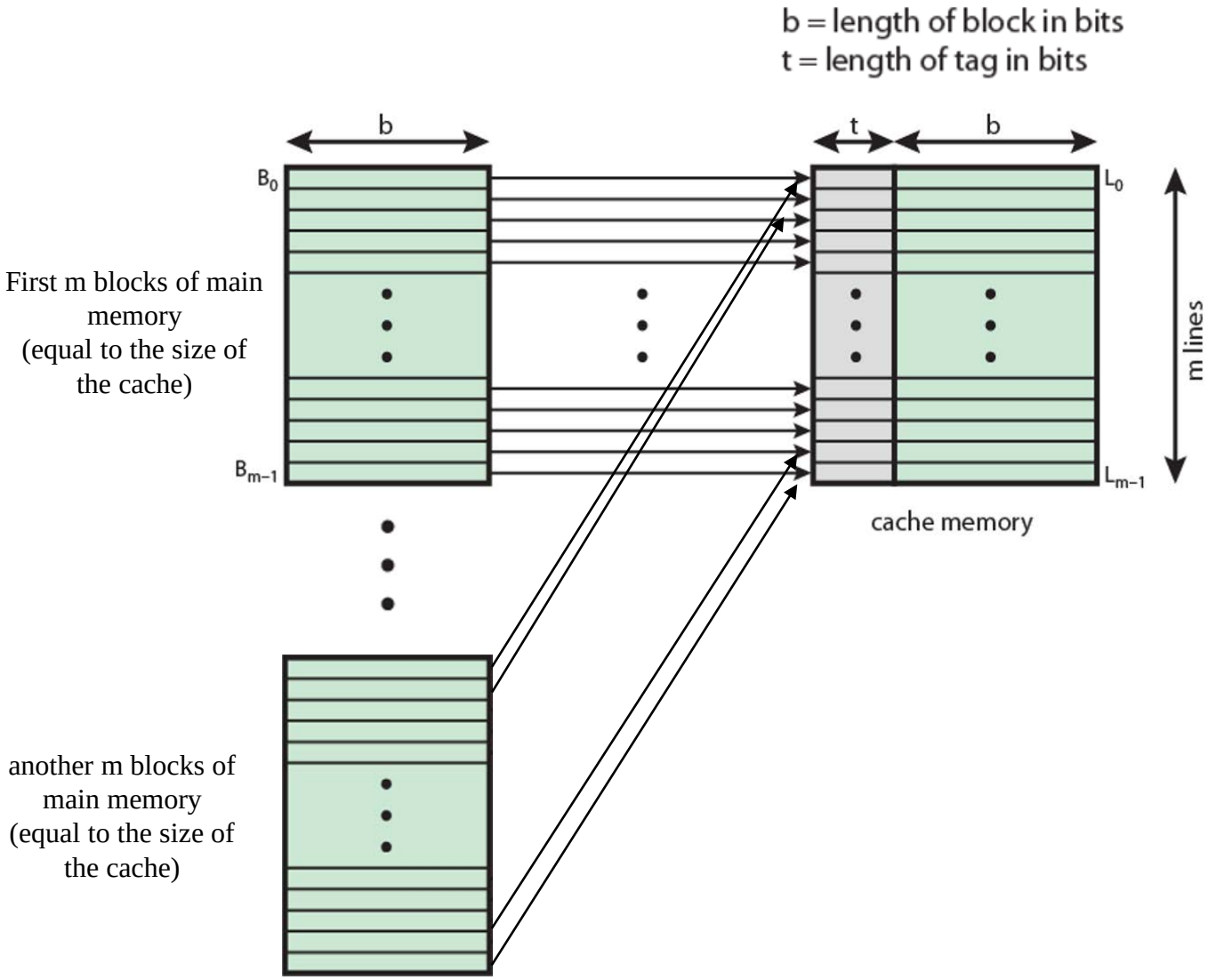
- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
  - Least Significant w bits identify unique word within a block of main memory
  - Most Significant s bits specify one of the 2^s blocks of main memory : Block identifier
    - The cache logic interprets these s bits as a tag of s-r bits (most significant) & line field of r bits
- Address length = (s + w) bits
  - Number of addressable units = 2^{s+w} words or bytes
- Size of a block = size of a line = 2^w words or bytes
- Number of blocks in main memory = $(2^{s+w})/(2^w) = 2^s$
- Number of lines in cache = $2^r = m$
- Size of tag = (s – r) bits

Direct Mapping Address Structure

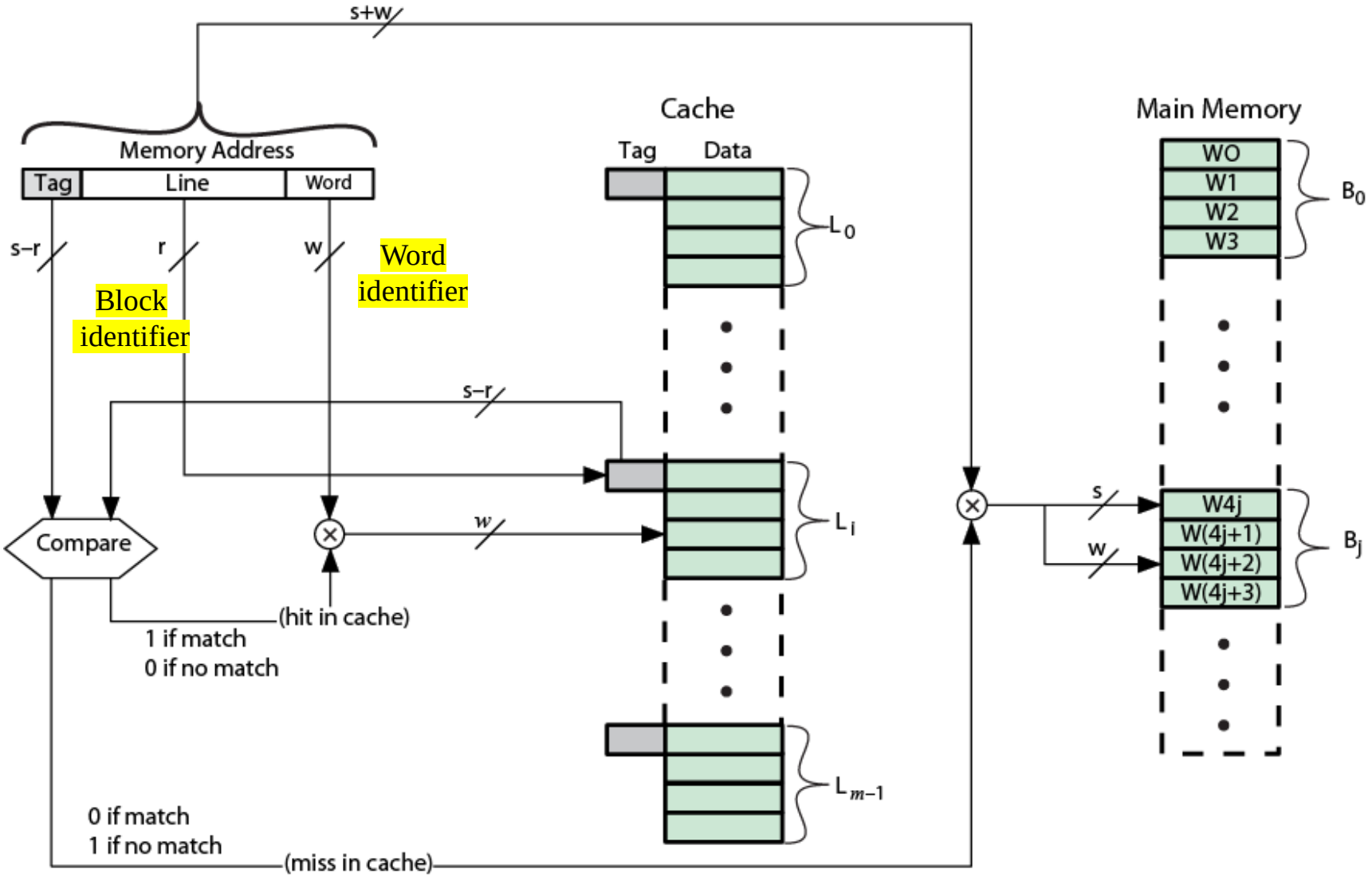
Tag s-r	Line or Slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

Direct Mapping from Cache to Main Memory



Direct Mapping Cache Organization

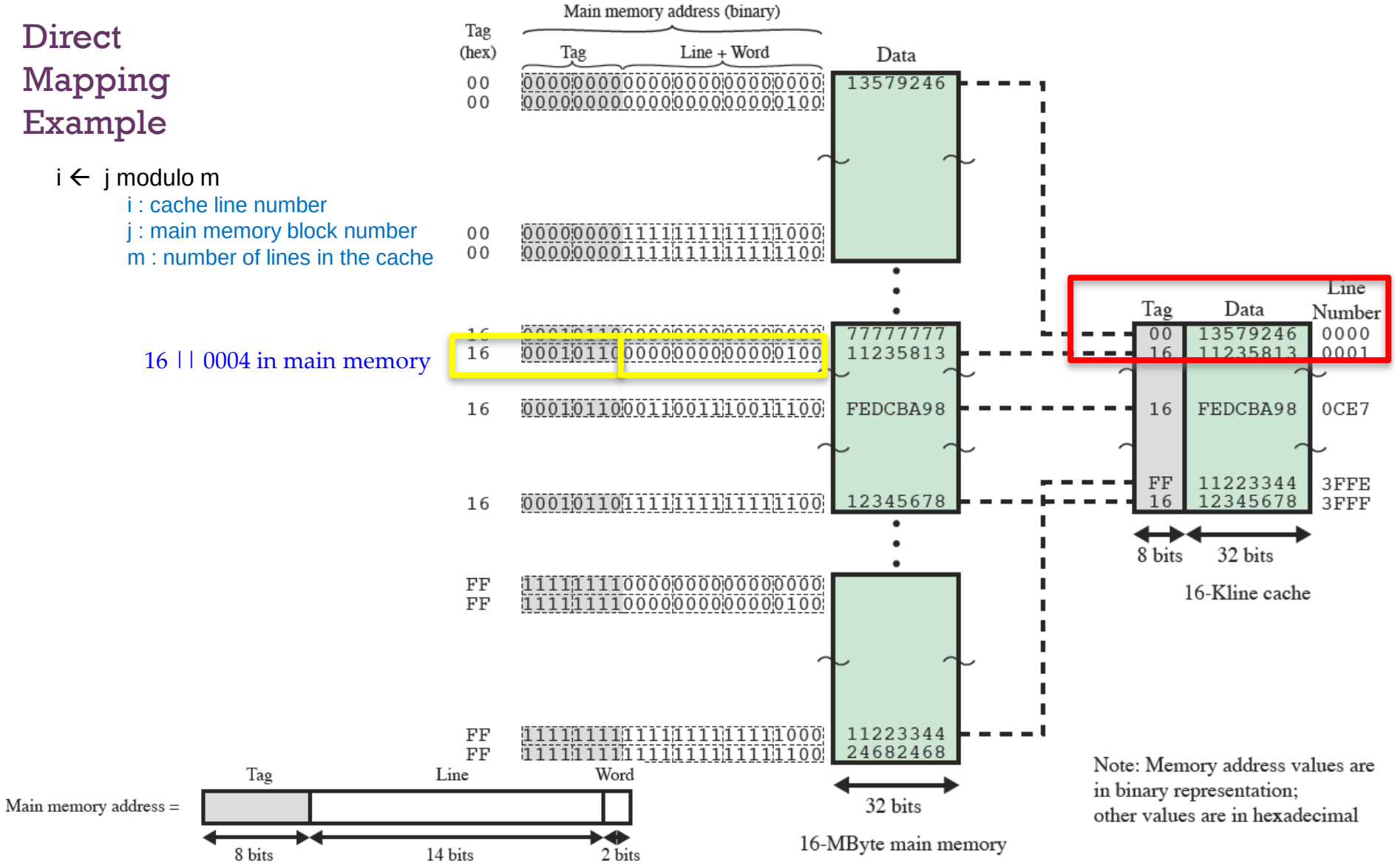


Direct Mapping Example

i ← j modulo m

- i : cache line number
- j : main memory block number
- m : number of lines in the cache

16 || 0004 in main memory



• In cache, tag(s-r) || line number (r)

•In 16K line cache

(1) Cache Tag (16) & Cache Line number (0001) → 16 || (0001*4) = 16 || 0004 in main memory

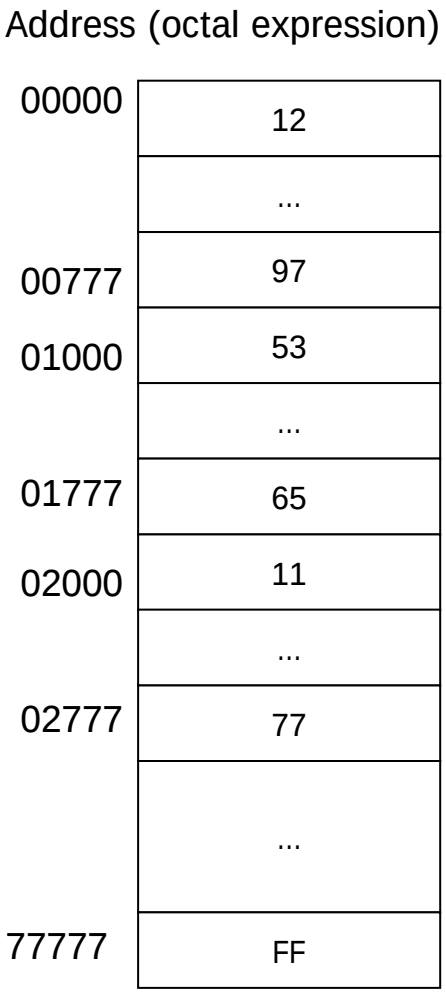
Word identifier

(2) Tag(FF) & line number (3FFE) → FF || (3FFE * 4) = FF || (EFF8) in main memory

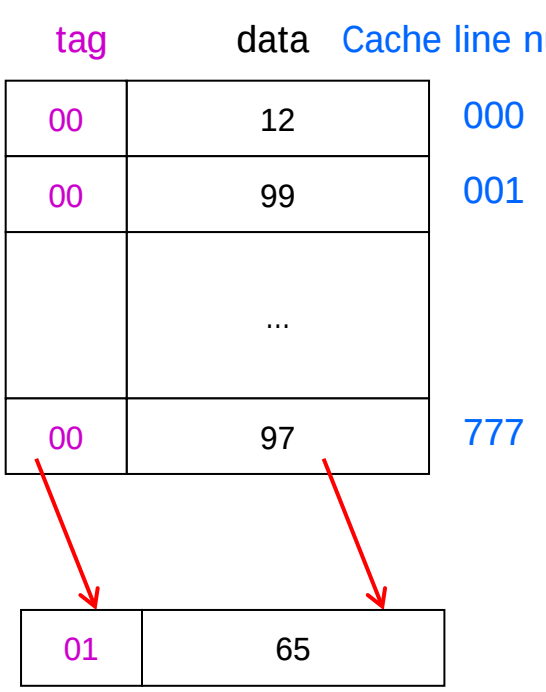
Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

More on Direct Mapping – cache miss



(a) Main Memory



From step (5),
The tag & data is updated

(b) Cache

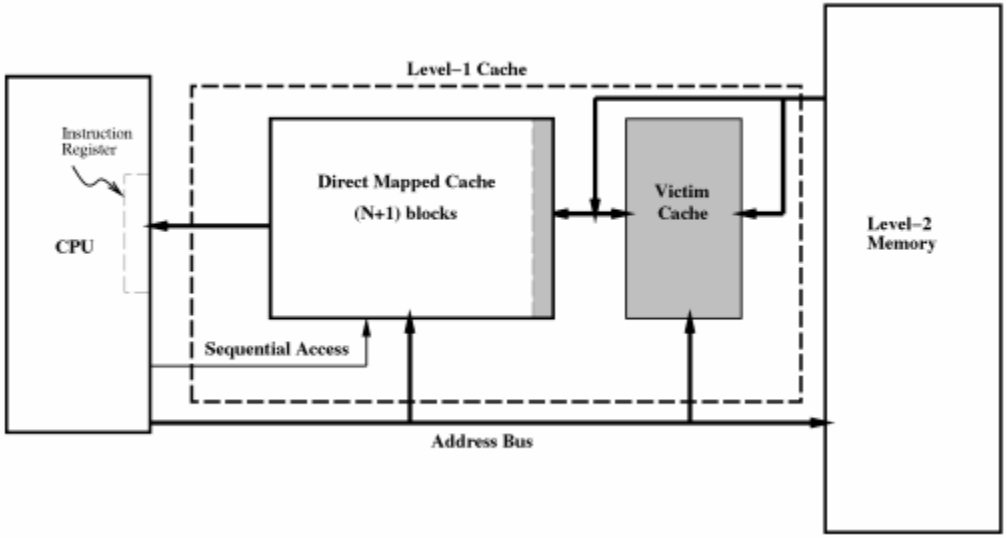
When the processor wants to read the data in memory address "01777"

(1) First, find the index 777 in the cache
(2) Compare the tag "01" with the tag value in the cache
(3) Since the tag value in the cache is "00", **the cache miss is occurred**
(4) Processor accesses the main memory, and then the word "65" is fetched.
(5) Also, the "65" value is updated to the cache line 777 with a new tag value "01"

In this example, we propose
- main Memory address = tag(6bits) + cache line index(9bits)
- Also, the address is expressed with an octal expression
- and 8 bit word size

Victim Cache

- An extension to a direct mapped cache that adds a small, secondary, fully associative cache to store cache blocks that have been ejected from the main cache due to a capacity or conflict miss.

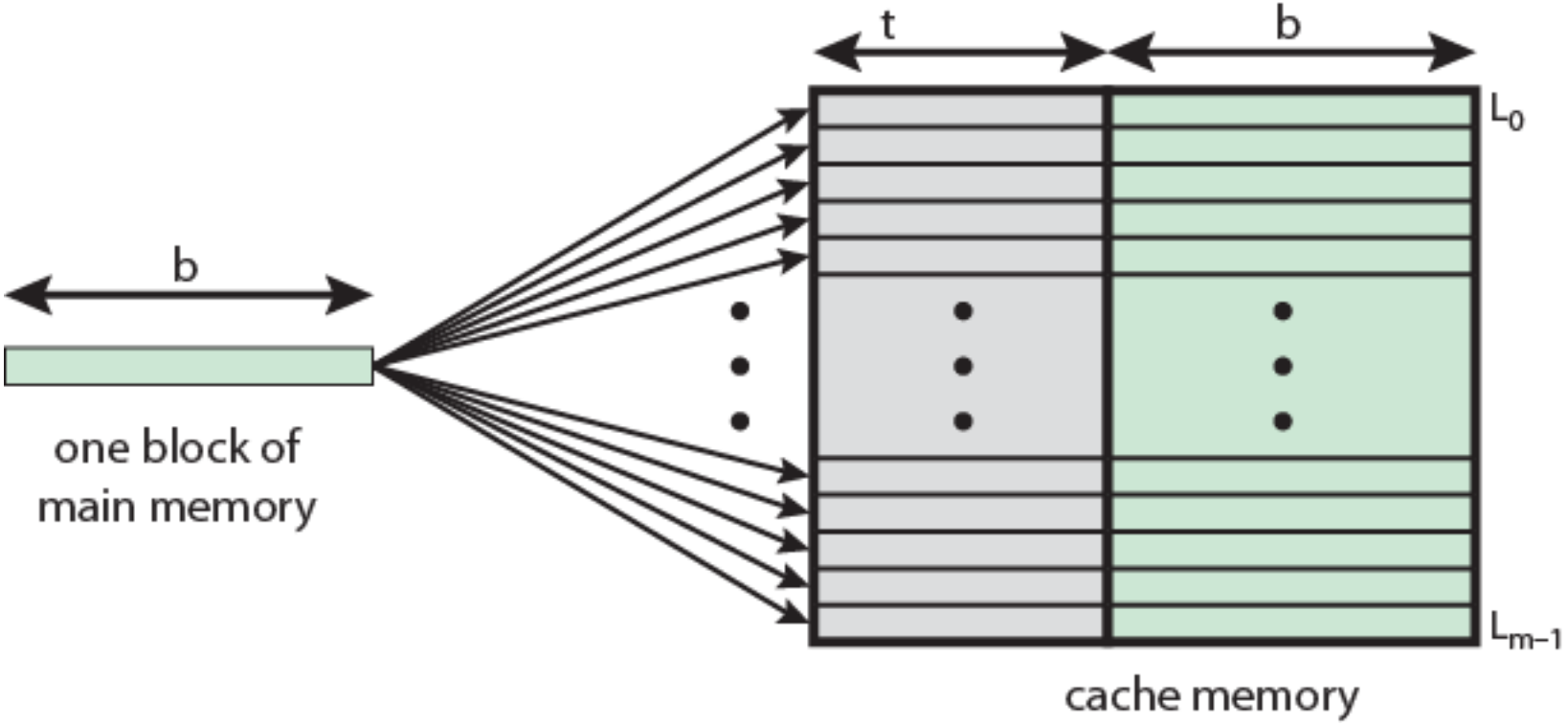


- Lower miss penalty
- Remember what was discarded
  - Already fetched
  - Use again with little penalty
- Fully associative
- 4 to 16 cache lines (작음 !!)
- Between direct mapped L1 cache and next memory level

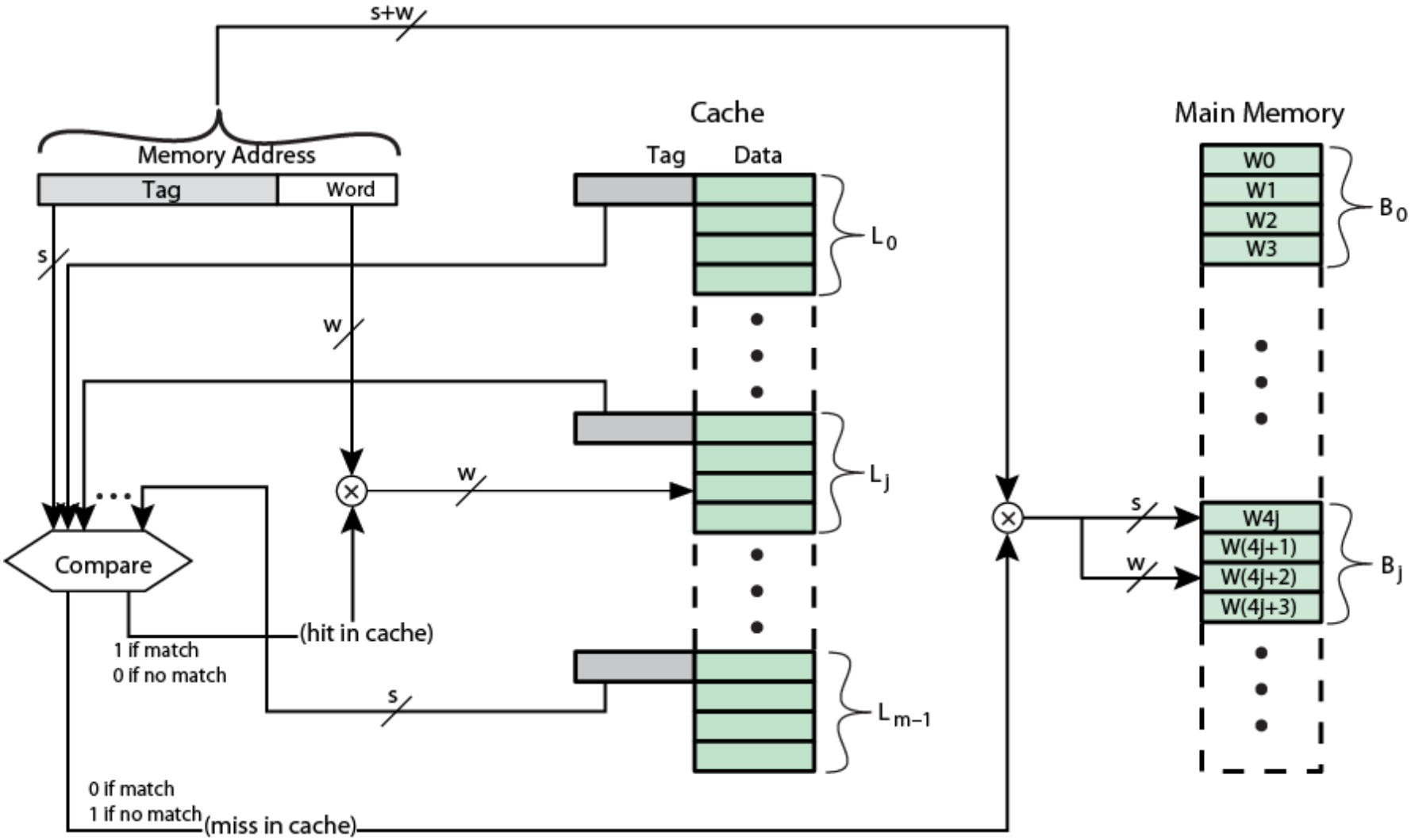
Associative Mapping

- Associative mapping overcomes the disadvantage of direct mapping
  - by permitting each main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
  - To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match
- Cache searching gets expensive

Associative Mapping from Cache to Main Memory



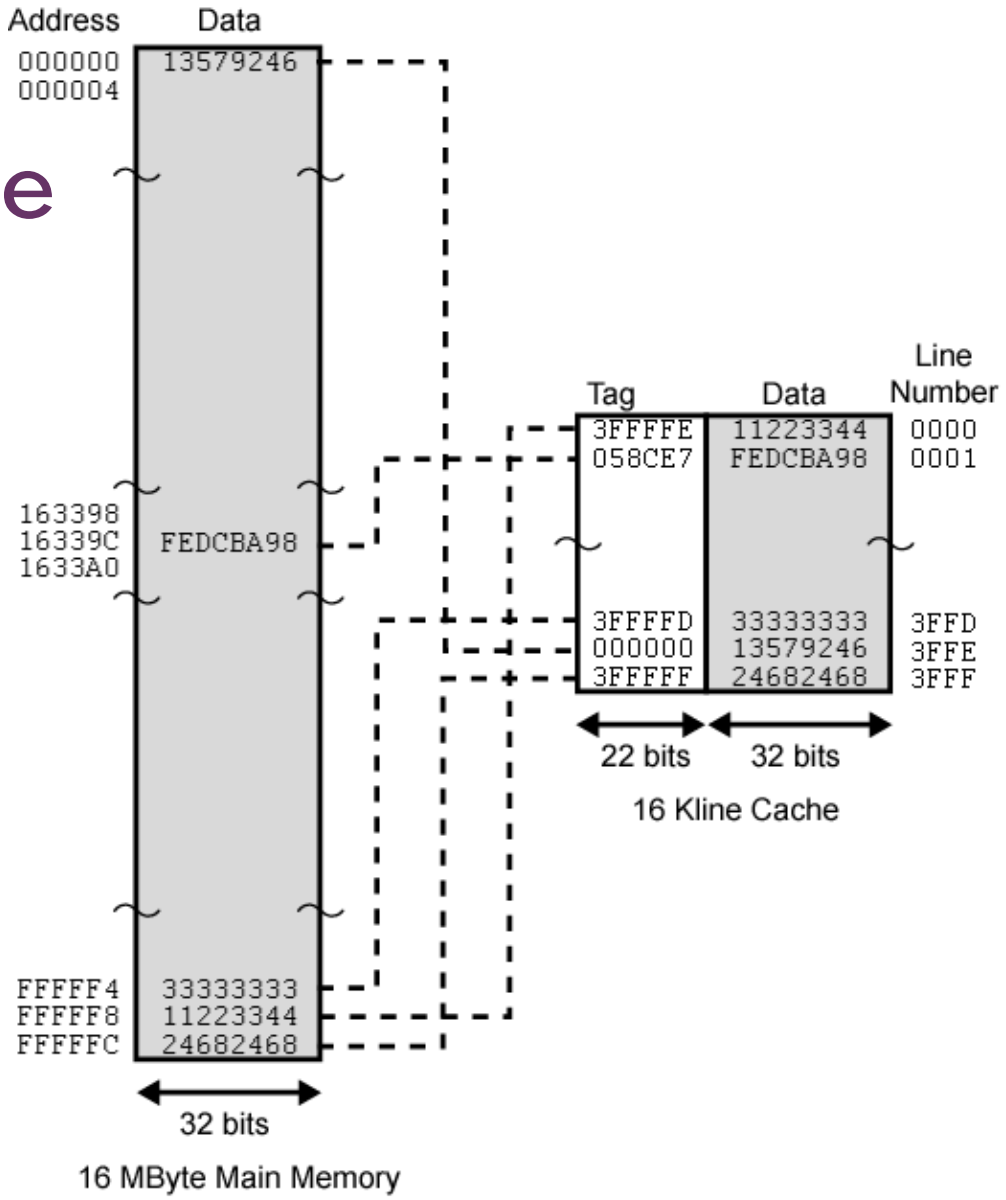
Fully Associative Cache Organization

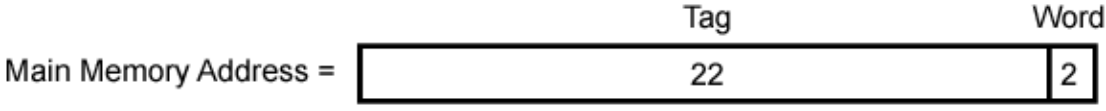


Associative Mapping Example

* 24bit sized main memory address : 16339C
→22bit sized tag value : 058CE7

* FFFFFC (in main memory) >> 2 bits
→3FFFFF (of tag value)





Associative Mapping Address Structure

Tag 22 bit	Word 2 bit

- 24 bit address
- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit

Associative Mapping Summary

- 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 = undetermined
- Size of tag = s bits

주소에 line number에 해당하는 내용없음 →
cache의 line number는 주소형식에 의해 결정되지 않음

- Advantage & Disadvantages of Associative Mapping
  - Advantage: flexible
  - Disadvantages : Cost, Complex circuit for simultaneous comparison

Set Associative Mapping

- Compromised to show the strengths of both the direct & associative mapping
- Cache is divided into a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set

Set Associative Mapping

- Cache is divided into *v* sets of *k* lines each
  - $m = v \times k$, where m: number of cache lines
  - $i = j \bmod v$, where

    i : cache set number

    j : main memory block number

    v : number of sets

- A given block maps to any line in a given set
- K-way set associate cache
  - 2-way and 4-way are common

Set Associative Mapping Example

- m = 16 lines, v = 8 sets

  $\Rightarrow k$ = 2 lines/set, 2 way set associative mapping

* Assume 32 blocks in memory, $i = j \bmod v$

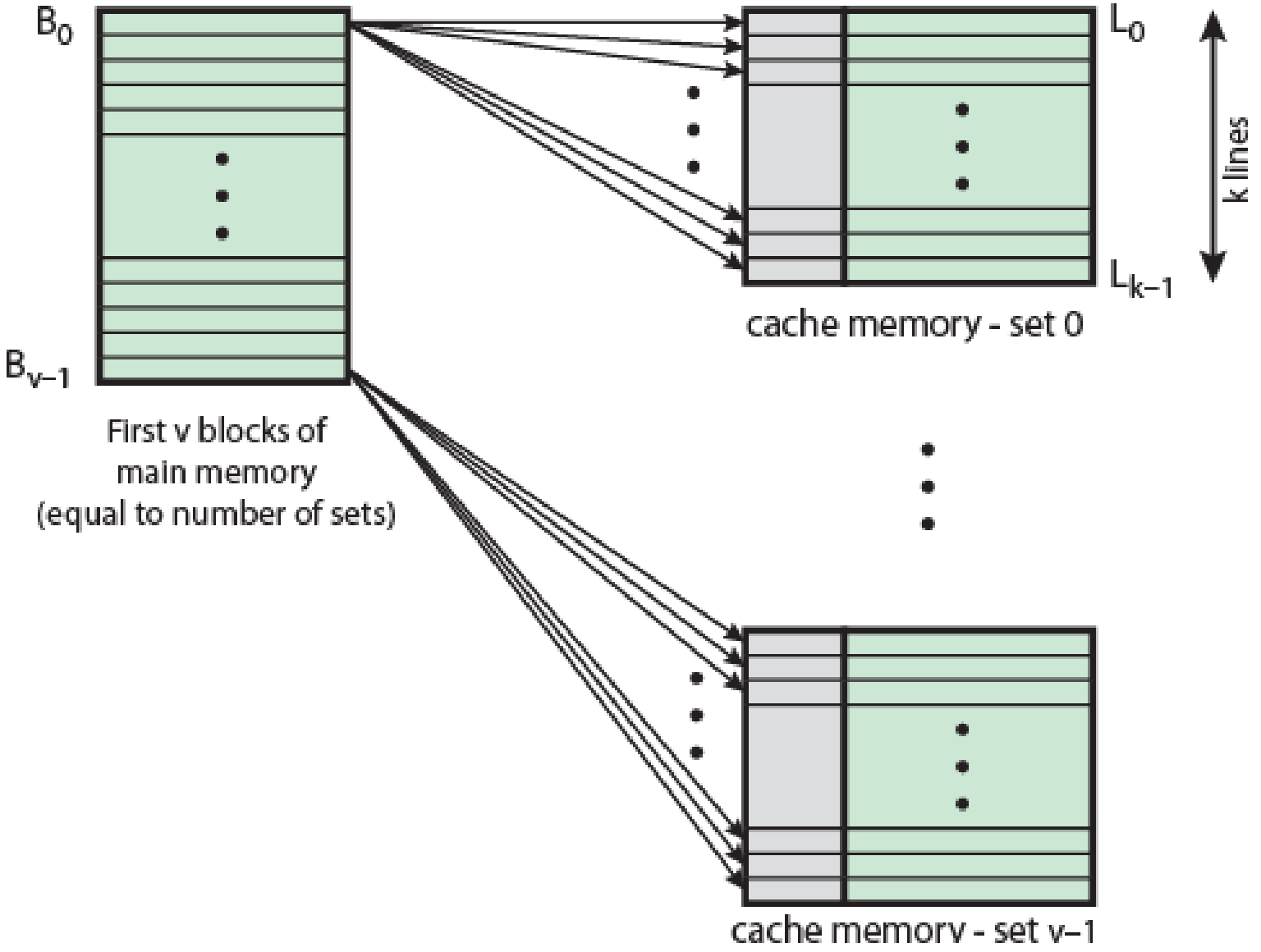
set	blocks
0	0, 8, 16, 24
1	1, 9, 17, 25
:	:
7	7, 15, 23, 31

- Since each set of cache has 2 lines, the memory block can be in one of 2 lines in the set
- e.g., block 17 can be assigned to either line 0 or line 1 in set 1

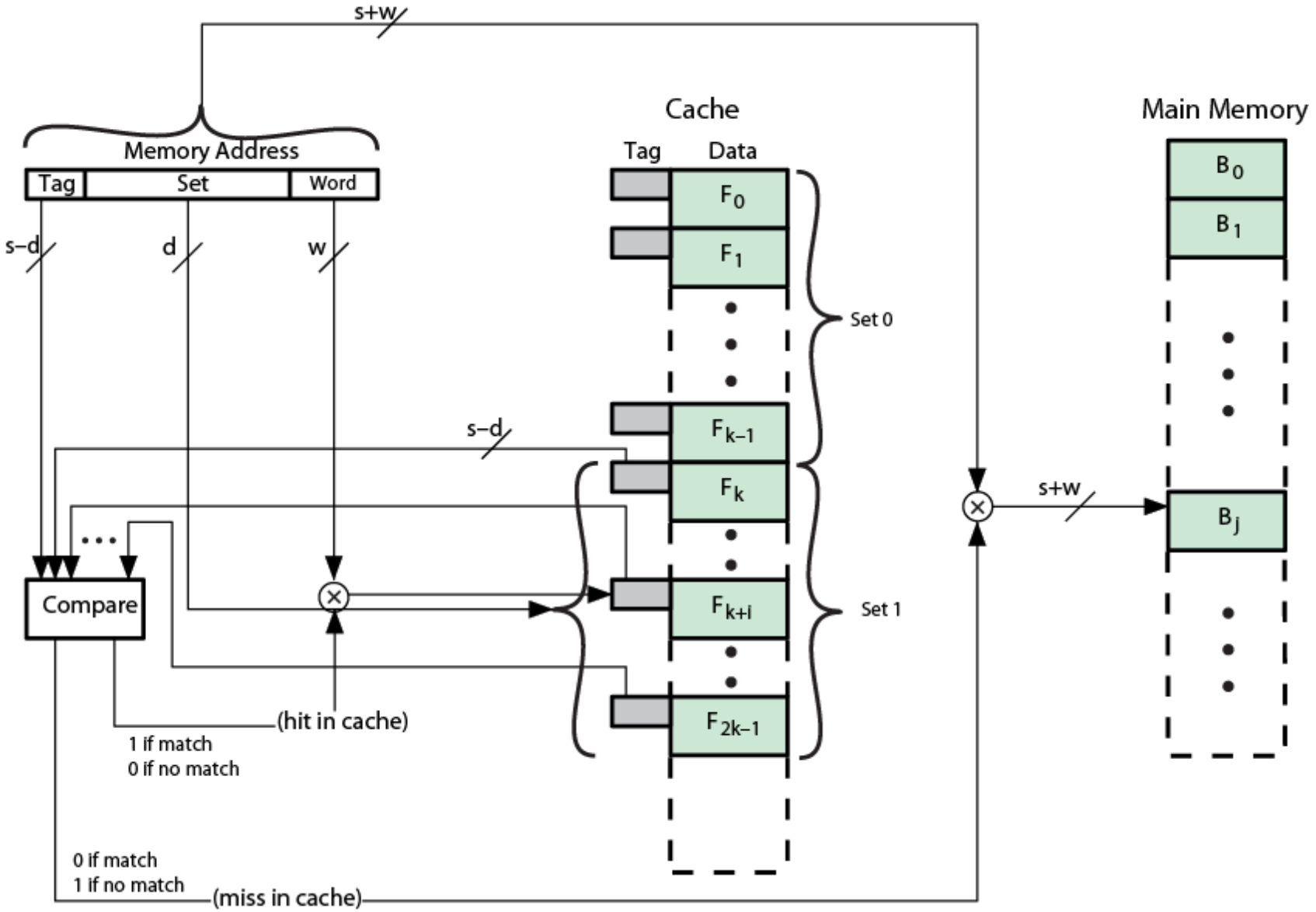
Set Associative Mapping Example

- Assume 13 bit set number
- Block number in main memory is modulo 2^{13}
  - (0010 0000 0000 0000 = 2000h)
  - 000000h, 002000, 004000, …, 00A000, 00C000 … map to same set. (← all of these examples have same values of least significant 13 bits)

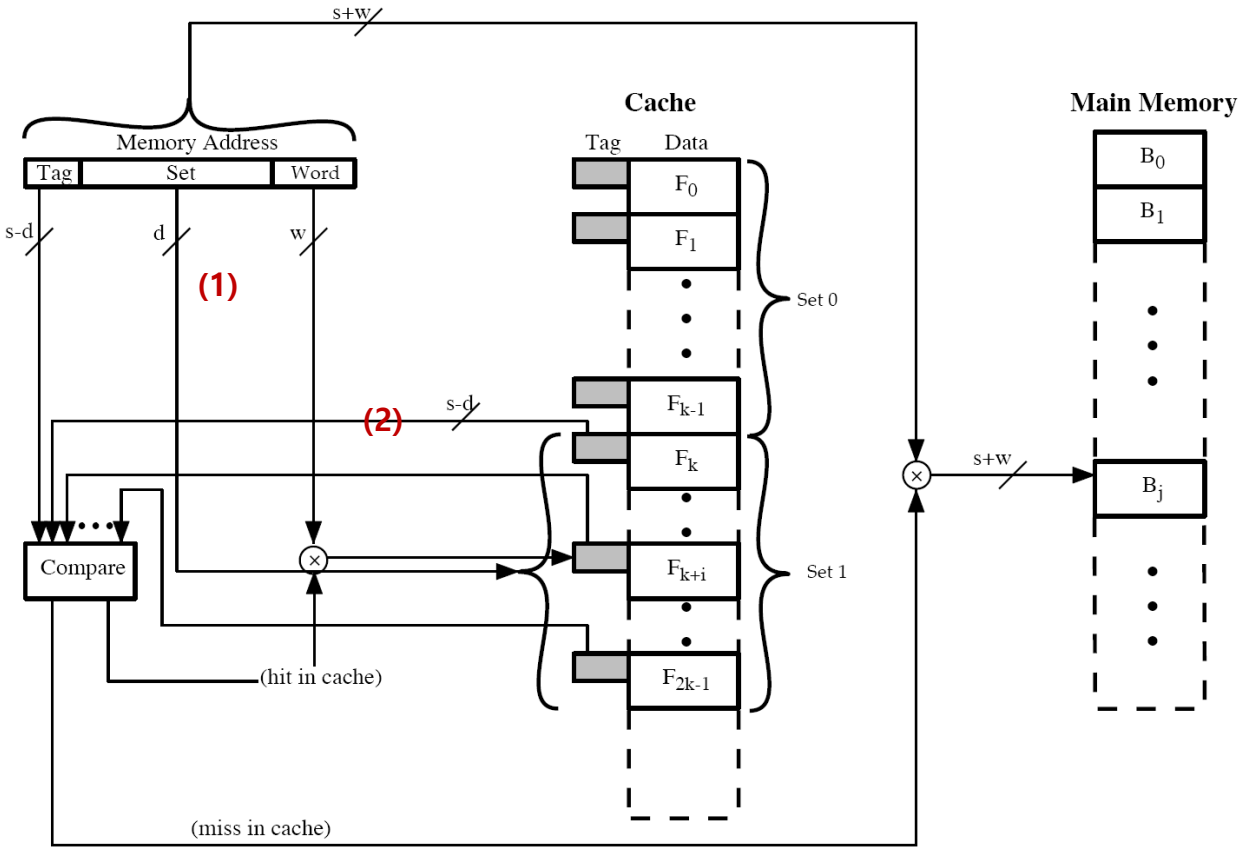
Mapping From Main Memory to Cache: v Associative

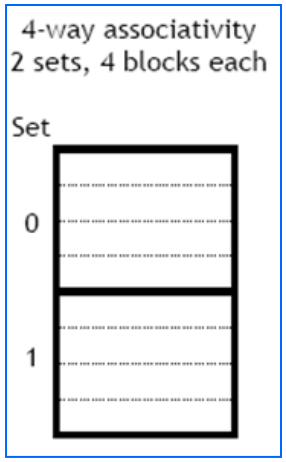


K-Way Set Associative Cache



k-Way Set Associative Cache Organization



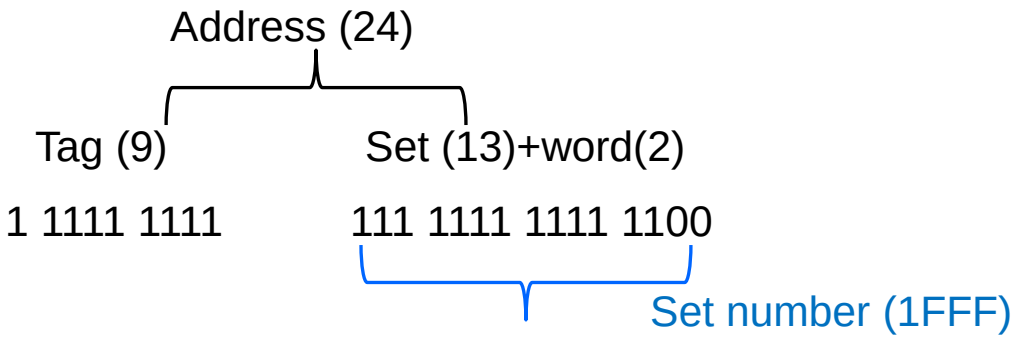


(1) 먼저 어느set에 있는지 확인
(2) 해당 set 중에서 어느 line에 해당하는지 tag값으로 확인

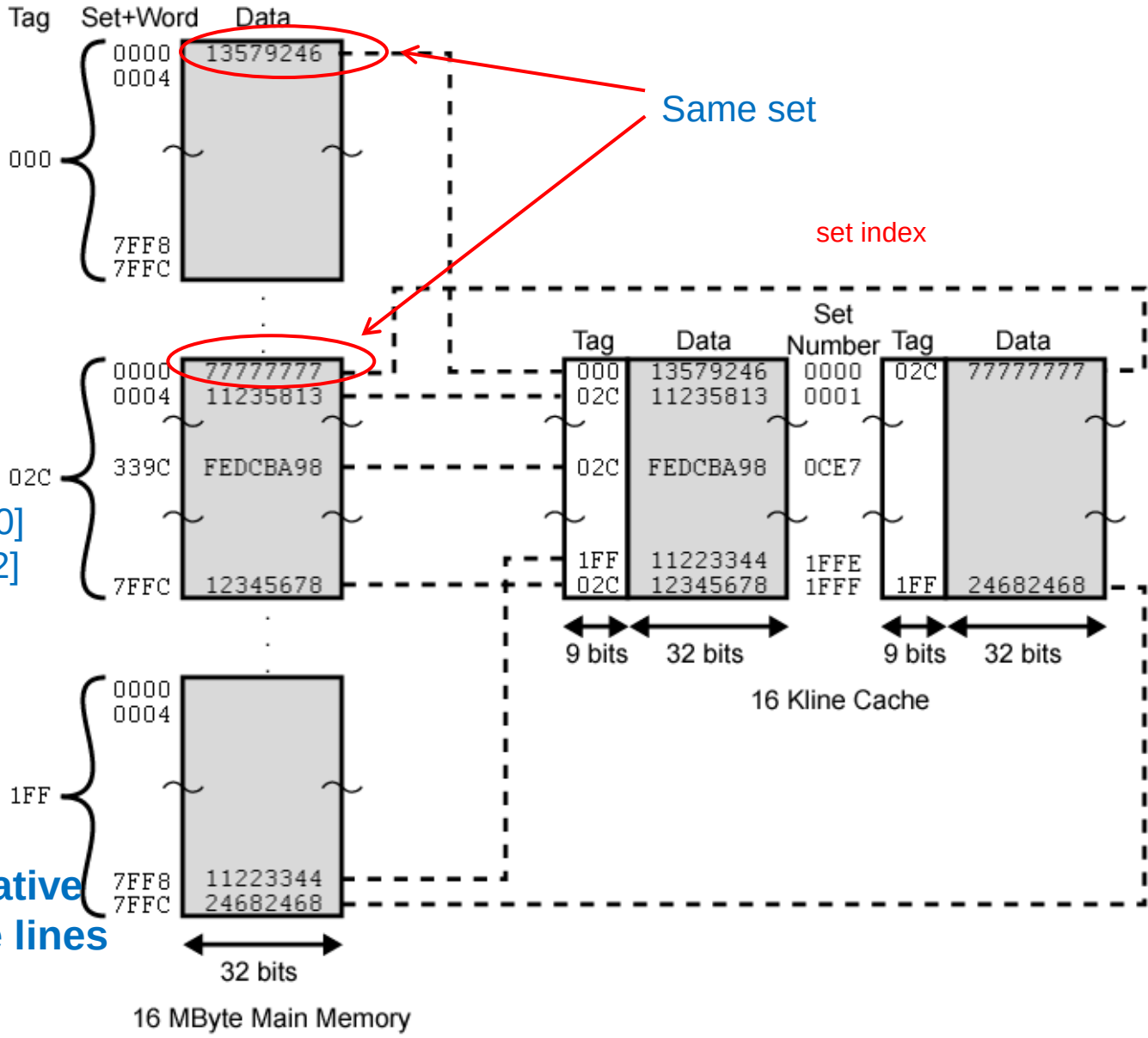
Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g
  - Address Tag Data Set number
  - 1FF 7FFC 1FF 12345678 1FFF
  - 001 7FFC 001 11223344 1FFF



Two Way Set Associative Mapping Example

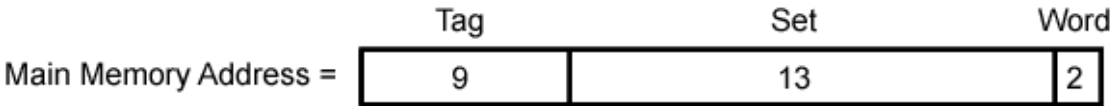


(1) Set Number :

mem addr (set + word) [14:0]
→ >> 2 → Set number [14:2]
0x0000 → 0000
0004 → 0001
339C → 0CE7
7FFC → 1FFF
7FF8 → 1FFE

(2) Since 2-way associative
→each set has 2 cache lines

(3) Tag : msb 9 bits
mem addr [23:15]



Set Associative Mapping Summary

- 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 set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k * 2^{d}
- Size of tag = (s – d) bits

Remarks

- Why is the simultaneous comparison cheaper here, compared to associate mapping?
  - Tag is much smaller
  - Only k tags within a set are compared
- Relationship between set associate and the first two: extreme cases of set associate
  - $k = 1 \Rightarrow v = m \Rightarrow$ direct mapping (1 line/set)
  - $k = m \Rightarrow v = 1 \Rightarrow$ associate mapping (one big set)

    k lines, v sets

Direct and Set Associative Cache Performance Differences

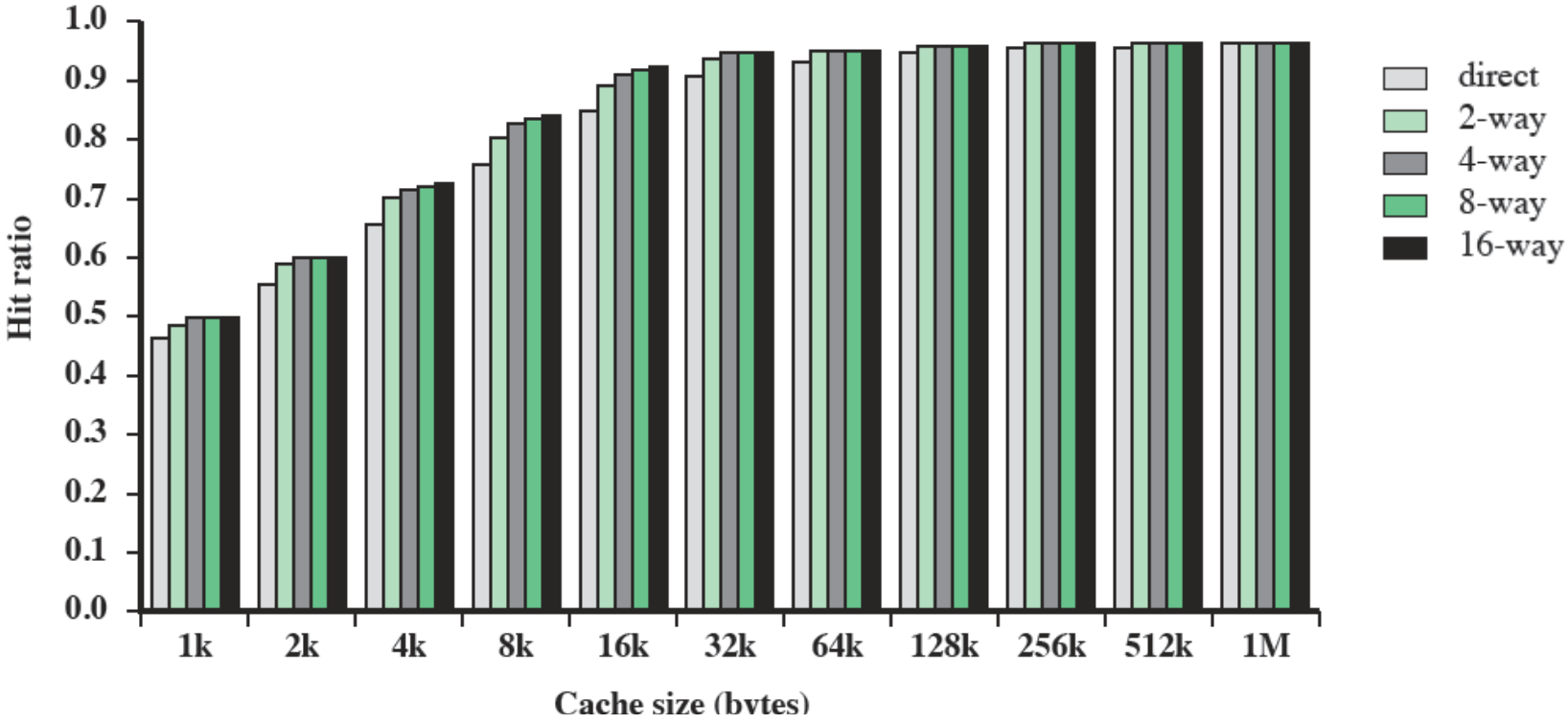
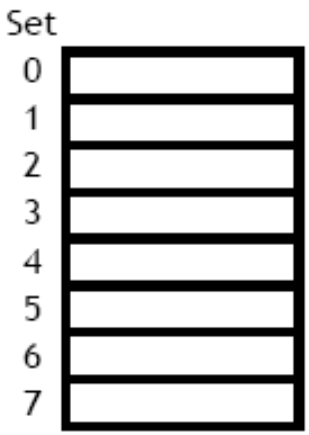


Figure 4.16 Varying Associativity over Cache Size

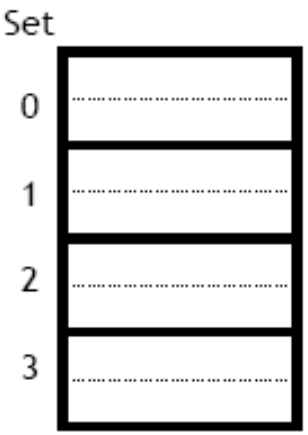
- Significant up to at least 64kB for 2-way
- Difference between 2-way and 4-way at 4kB much less than 4kB to 8kB
- The complexity of the cache increases in proportion to the associativity, and in this case would not be justifiable against increasing cache size to 8 or even 16 Kbytes.
- Above 32kB gives no improvement

Additional

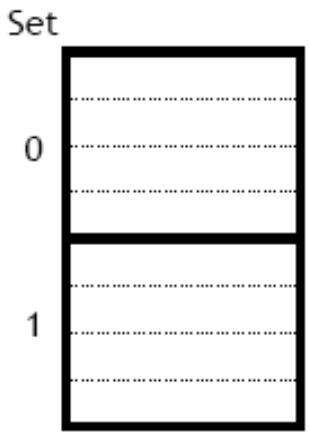
- 1-way associativity
  8 sets, 1 block each



2-way associativity
4 sets, 2 blocks each

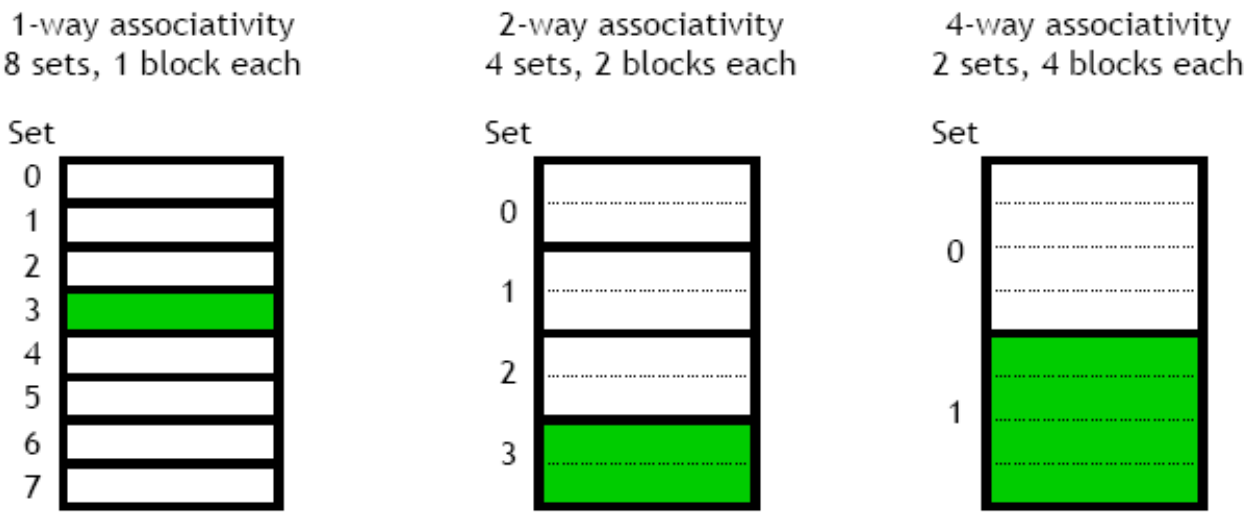


4-way associativity
2 sets, 4 blocks each



Additional

- Where would data from memory byte address 6195 be placed, assuming the eight-block cache designs below, with 16 bytes per block?
- 6195 in binary is 00...0110000 011 0011.
- Each block has 16 bytes, so the lowest 4 bits are the block offset.
- For the 1-way cache, the next three bits (011) are the set index.
  For the 2-way cache, the next two bits (11) are the set index.
  For the 4-way cache, the next one bit (1) is the set index.
- The data may go in *any* block, shown in green, within the correct set.



Replacement Algorithms (1) Direct mapping

- Replacement algorithm
  - When a new block is brought into cache, one of existing blocks must be replaced

- In direct mapping, the replacement alg has the following features:
  - No choice
  - Each block only maps to one line
  - Replace that line

Replacement Algorithms (2) Associative & Set Associative

- Hardware implemented algorithm (speed)
- Least Recently used (LRU)
  - e.g. in 2 way set associative
    - Which of the 2 block is LRU ?
- First in first out (FIFO)
  - replace block that has been in cache longest
- Least frequently used
  - replace block which has had fewest hits
- Random

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend with:

More than one device may have access to main memory

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Write Through and Write Back

- Write through
  - Simplest technique
  - All write operations are made to main memory as well as to the cache
  - The main disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

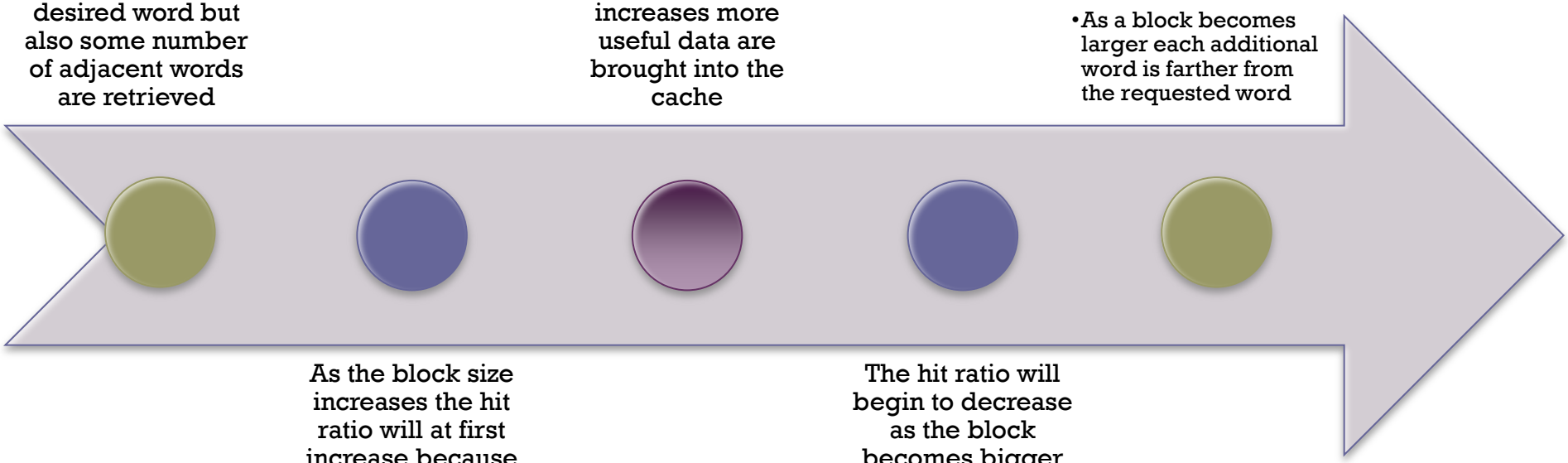
Line Size

When a block of data is retrieved and placed in the cache not only the desired word but also some number of adjacent words are retrieved

As the block size increases more useful data are brought into the cache

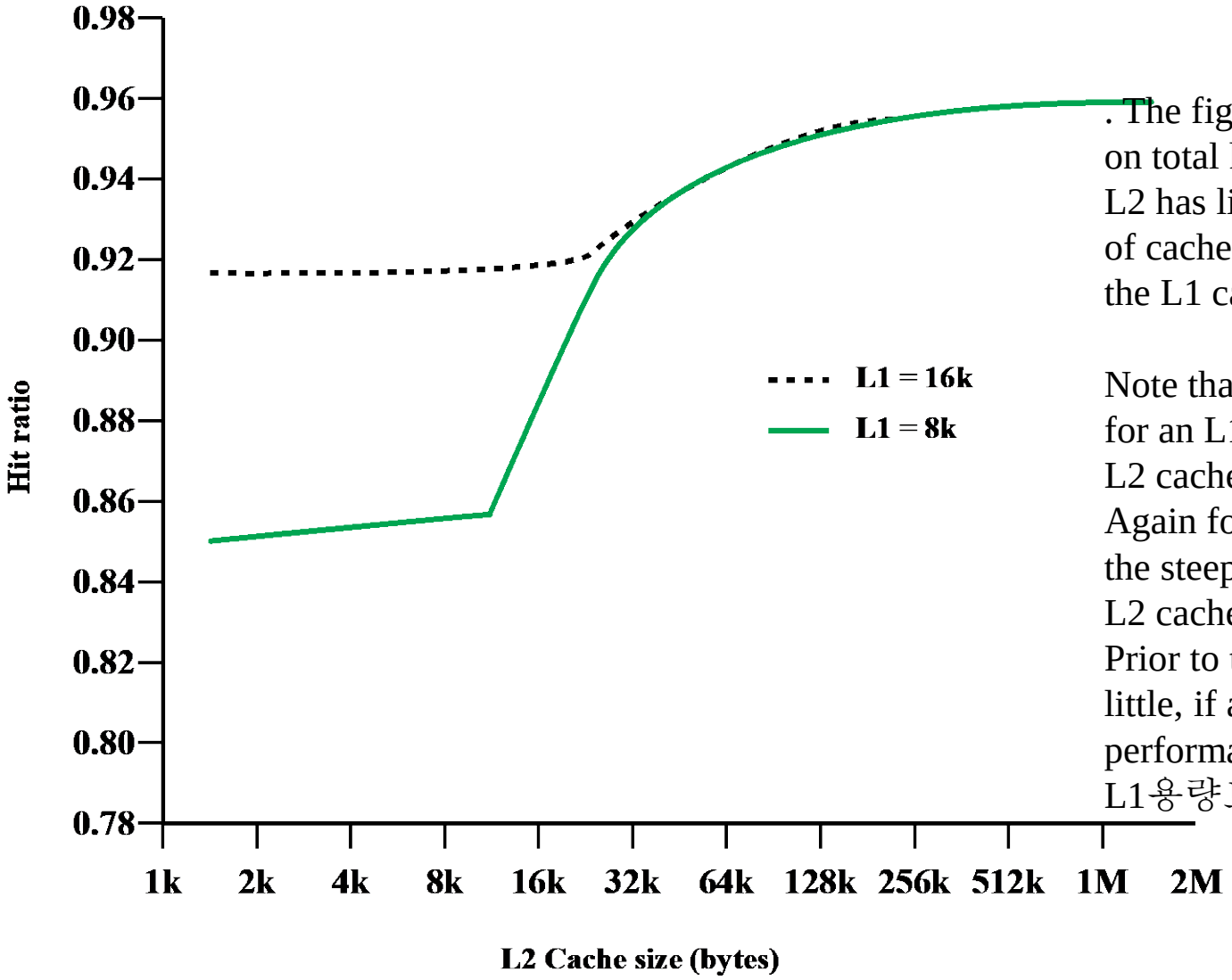
Two specific effects come into play:

- Larger blocks reduce the number of blocks that fit into a cache
- As a block becomes larger each additional word is farther from the requested word



As the block size increases the hit ratio will at first increase because of the principle of locality

The hit ratio will begin to decrease as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced



. The figure shows the impact of L2 on total hits with respect to L1 size. L2 has little effect on the total numbe of cache hits until it is at least double the L1 cache size.

Note that the steepest part of the slop for an L1 cache of 8 Kbytes is for an L2 cache of 16 Kbytes.
Again for an L1 cache of 16 Kbytes, the steepest part of the curve is for an L2 cache size of 32 Kbytes.
Prior to that point, the L2 cache has little, if any, impact on total cache performance. (L2 용량이 L1용량보다 적을때는 무의미함)

The need for the L2 cache to be larger than the L1 cache to affect performance makes sense.

Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

Multilevel Caches

- Modern CPU has on-chip cache (L1) that increases overall performance
  - e.g., 80486: 8KB
    Pentium: 16KB
    PowerPC: up to 64KB
- Secondary, off-chip cache (L2) provides high speed access to main memory
  - Generally 512KB or less
  - Current processor has the L2 cache in its processor

Multilevel Caches

- High logic density enables caches on chip
  - Faster than bus access
  - Frees bus for other transfers
- Common to use both on and off chip cache
  - L1 on chip, L2 off chip in static RAM
  - L2 access much faster than DRAM or ROM
  - L2 often uses separate data path
  - L2 may now be on chip
  - Resulting in L3 cache
    - Bus access or now on chip…

Unified vs. Split

- Unified cache
  - Stores data and instructions in one cache
  - Flexible and can balance the load between data and instruction fetches

    $\Rightarrow$ higher hit ratio
  - Only one cache to design and implement

- Split cache
  - Two caches, one for data and one for instructions
  - Trend toward split cache
  - Good for superscalar machines that support parallel execution, prefetch, and pipelining
    - Overcome cache contention

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 4.4

Intel Cache Evolution

(Table is on page 150 in the textbook.)

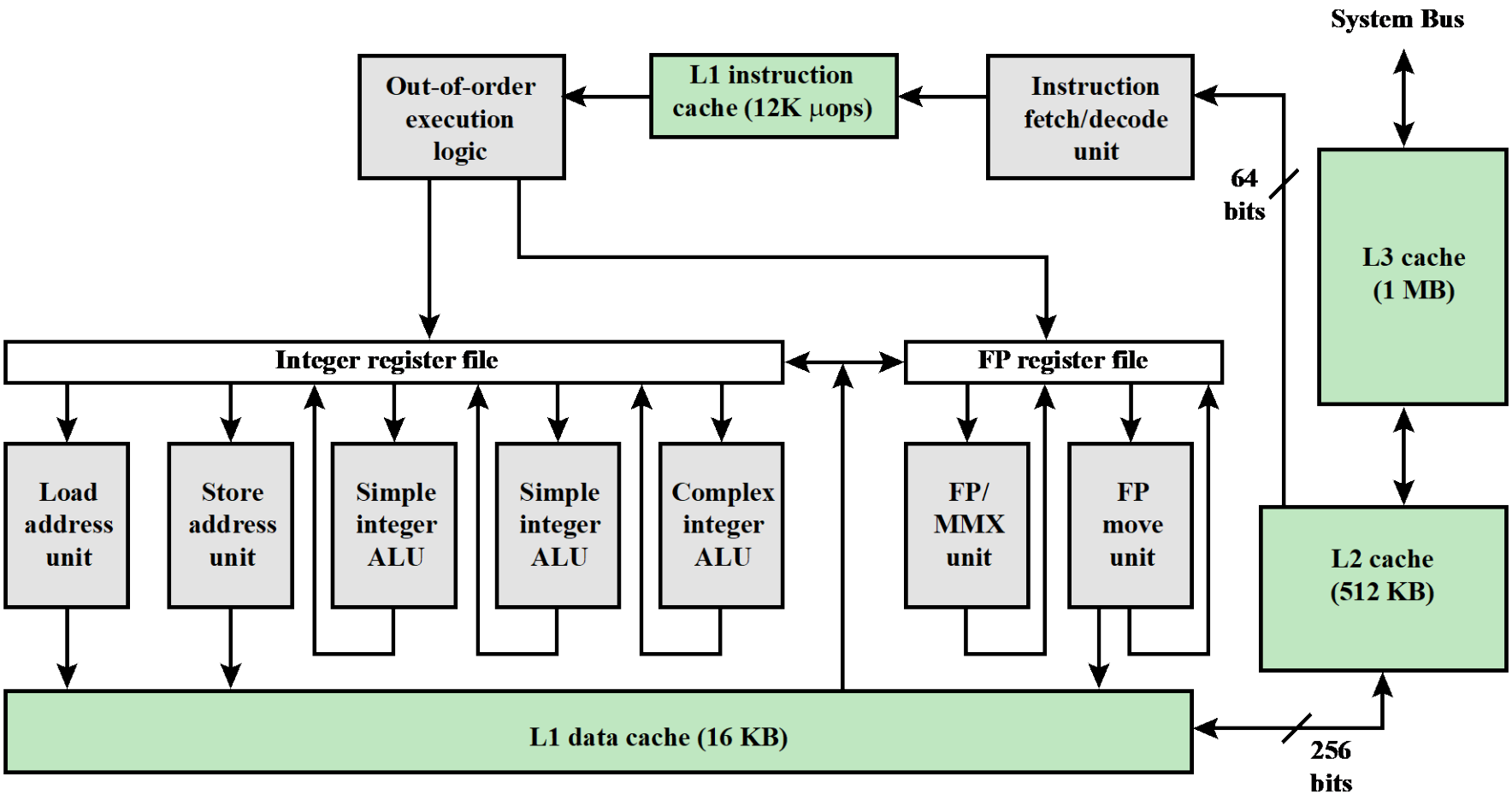


Figure 4.18 Pentium 4 Block Diagram

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.

Summary

Chapter 4

- Computer memory system overview
  - Characteristics of Memory Systems
  - Memory Hierarchy
- Cache memory principles
- Pentium 4 cache organization

Cache Memory

- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches