Lecture B - 05

Memory Architecture

Key Characteristics of Computer Memory Systems

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

Table 4.1 Key Characteristics of Computer Memory Systems

Characteristics of Memory Systems

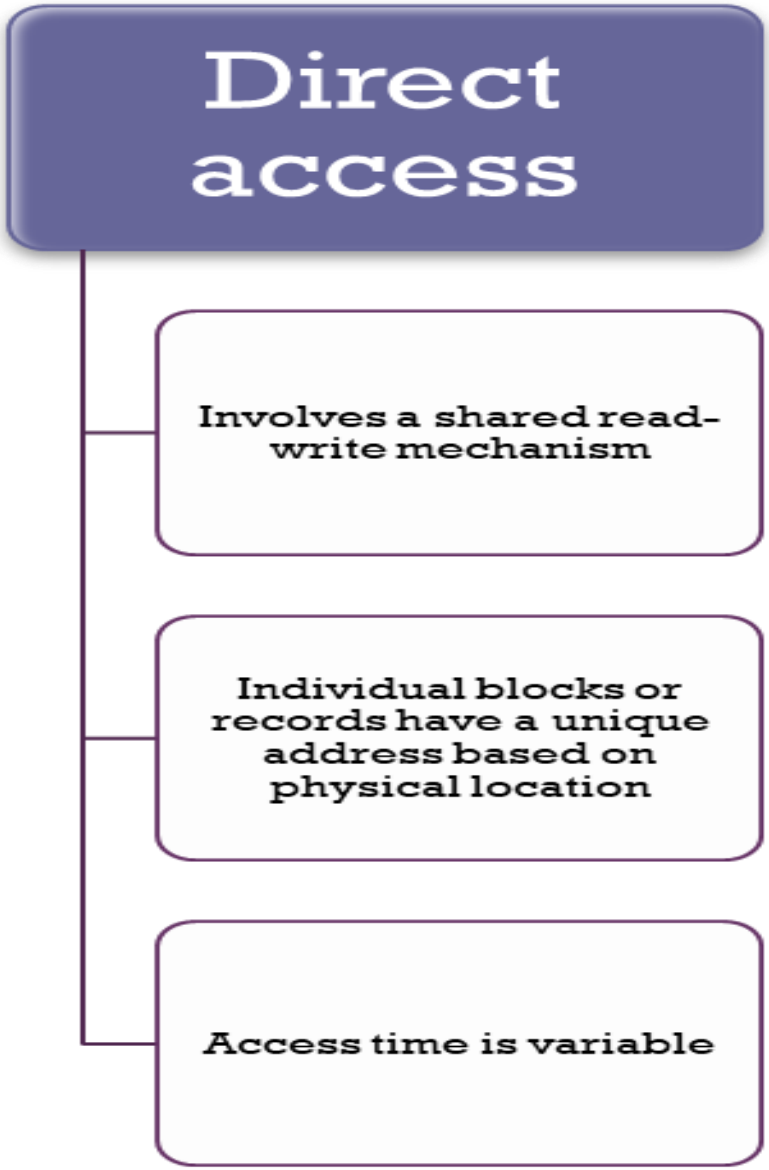
- Location
  - Refers to whether memory is internal and external to the computer
  - Internal memory is often equated with main memory
  - Processor requires its own local memory, in the form of registers
  - Cache is another form of internal memory
  - External memory consists of peripheral storage 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

Method of Accessing Units of Data

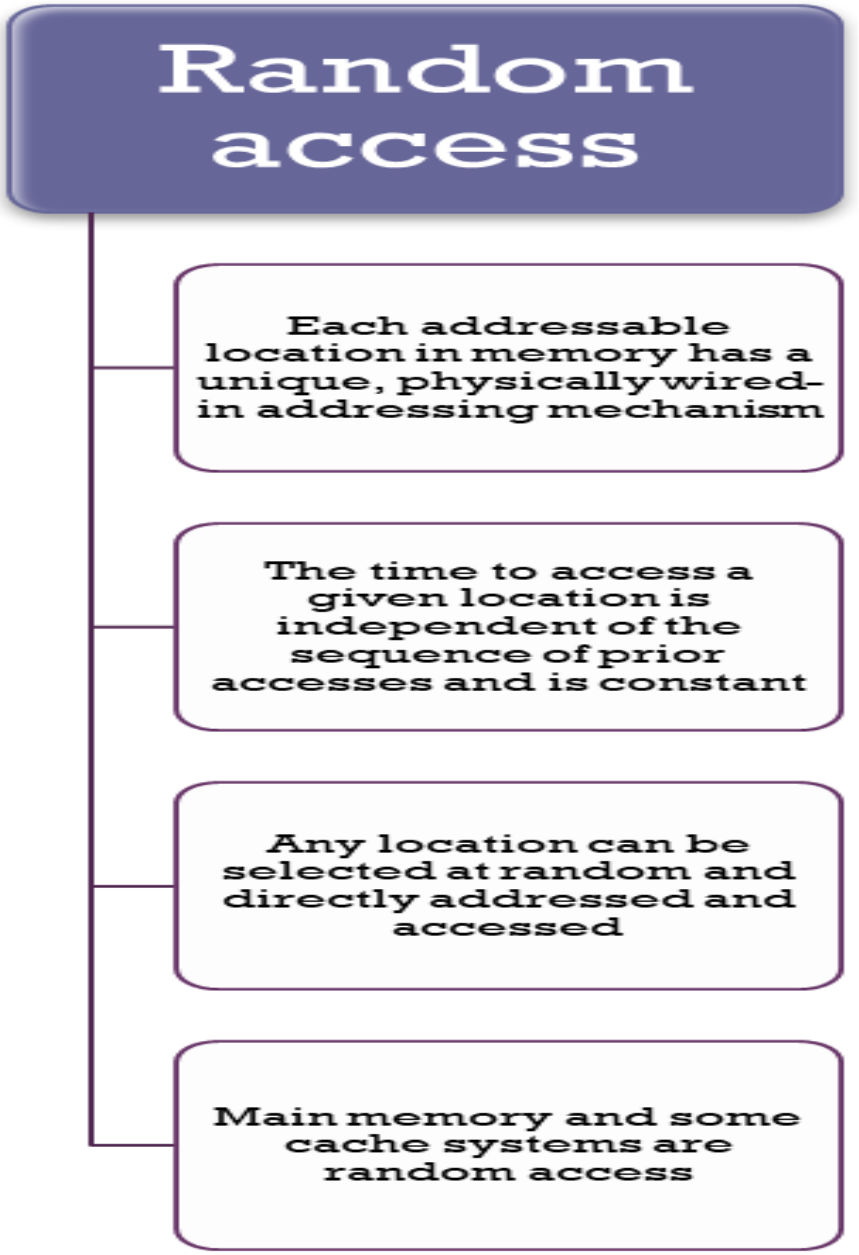
- Sequential access
- Random access
- Direct access
- Associative

Method of Accessing Units of Data

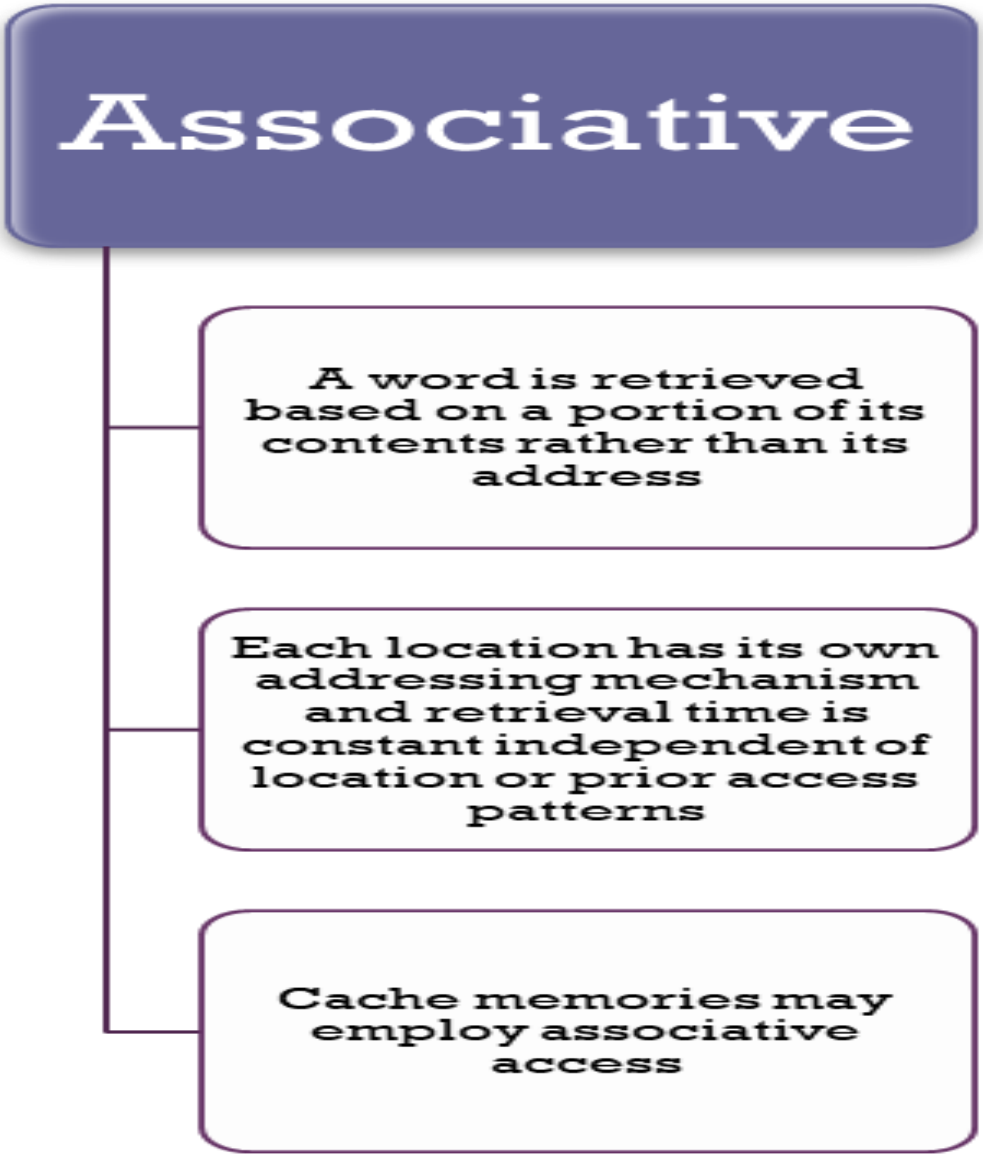
Method of Accessing Units of Data



Method of Accessing Units of Data



Method of Accessing Units of Data



Capacity and Performance:

The two most important characteristics of memory

Three performance parameters are used:

Access time (latency)

- For random-access memory it is the time it takes to perform a read or write operation
- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location

Memory cycle time

- Access time plus any additional time required before second access can commence
- Additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively
- Concerned with the system bus, not the processor

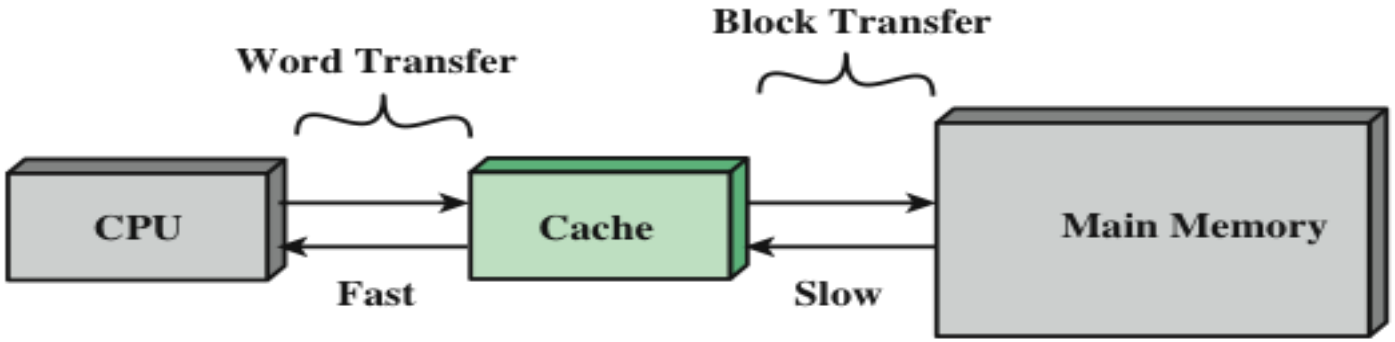
Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to 1/(cycle time)

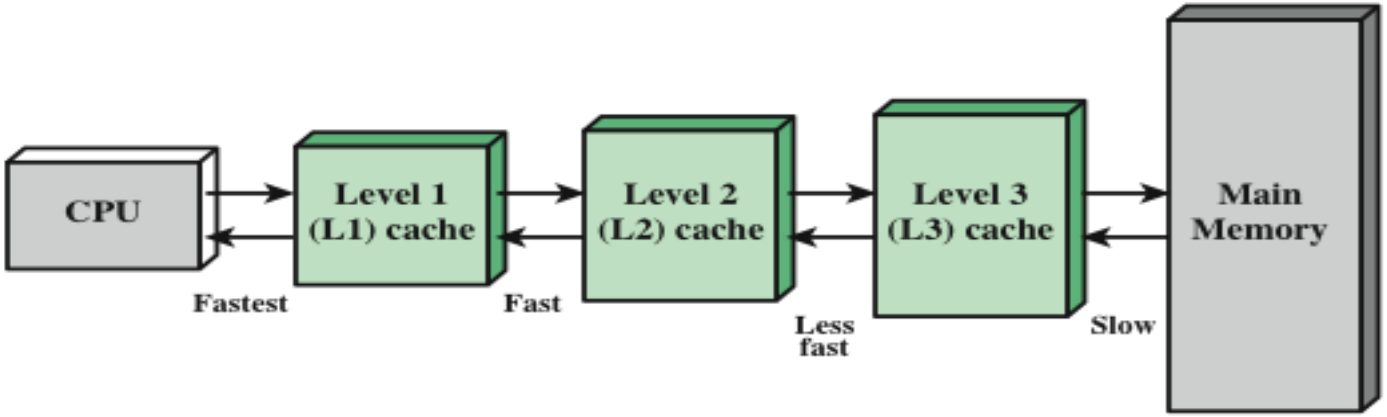
Memory

- 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

Cache and Main Memory

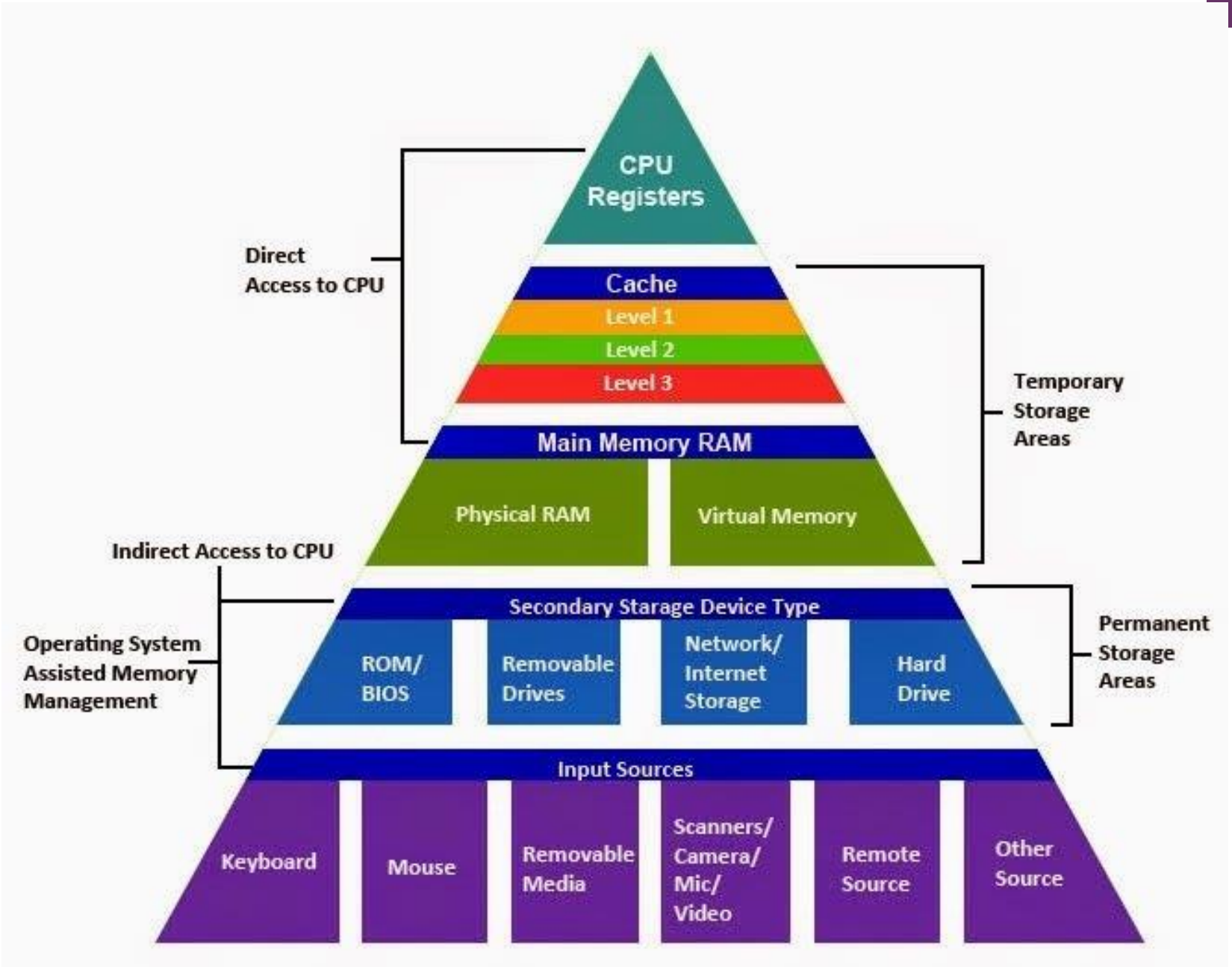


(a) Single cache



(b) Three-level cache organization

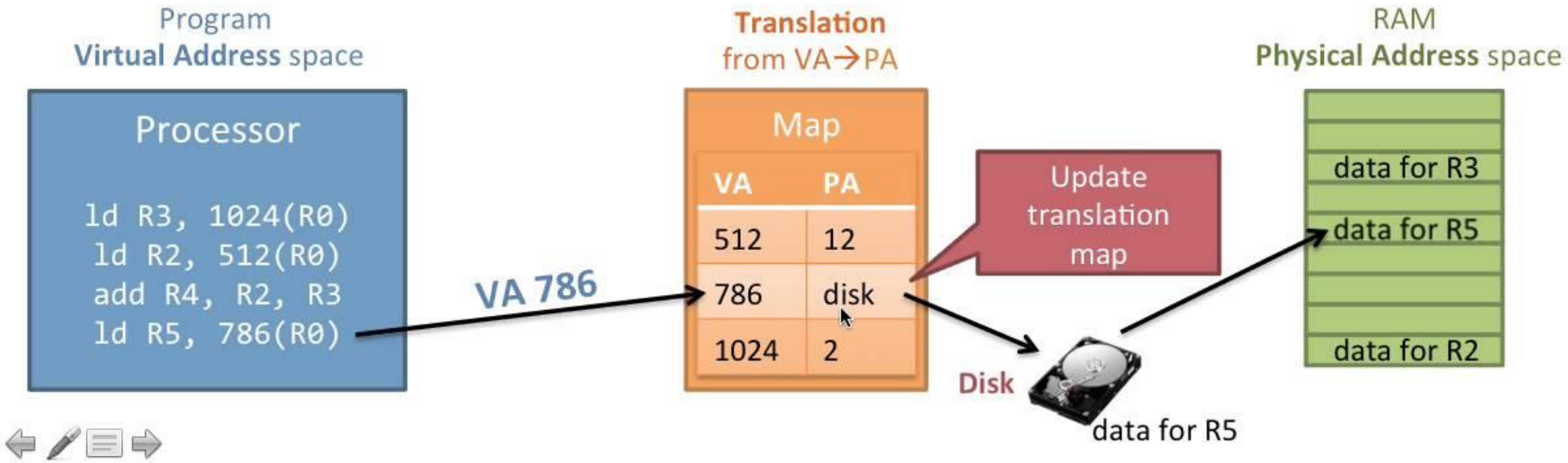
Figure 4.3 Cache and Main Memory


CPU Registers
Direct Access to CPU
Cache
Level 1
Level 2
Level 3
Main Memory RAM
Physical RAM
Virtual Memory
Temporary Storage Areas
Indirect Access to CPU
Secondary Starage Device Type
Operating System Assisted Memory Management
ROM/ BIOS
Removable Drives
Network/ Internet Storage
Hard Drive
Permanent Storage Areas
Input Sources
Keyboard
Mouse
Removable Media
Scanners/ Camera/ Mic/ Video
Remote Source
Other Source

Making VM work: translation

How does a program access memory?

1. Program executes a load with a **virtual address (VA)**
2. Computer **translates** the address to the **physical address (PA)** in memory
3. (If the **physical address (PA)** is not in memory, the operating system **loads it in from disk**)
4. The computer then **reads the RAM** using the **physical address (PA)** and returns the data to the program



Cache/Main Memory Structure

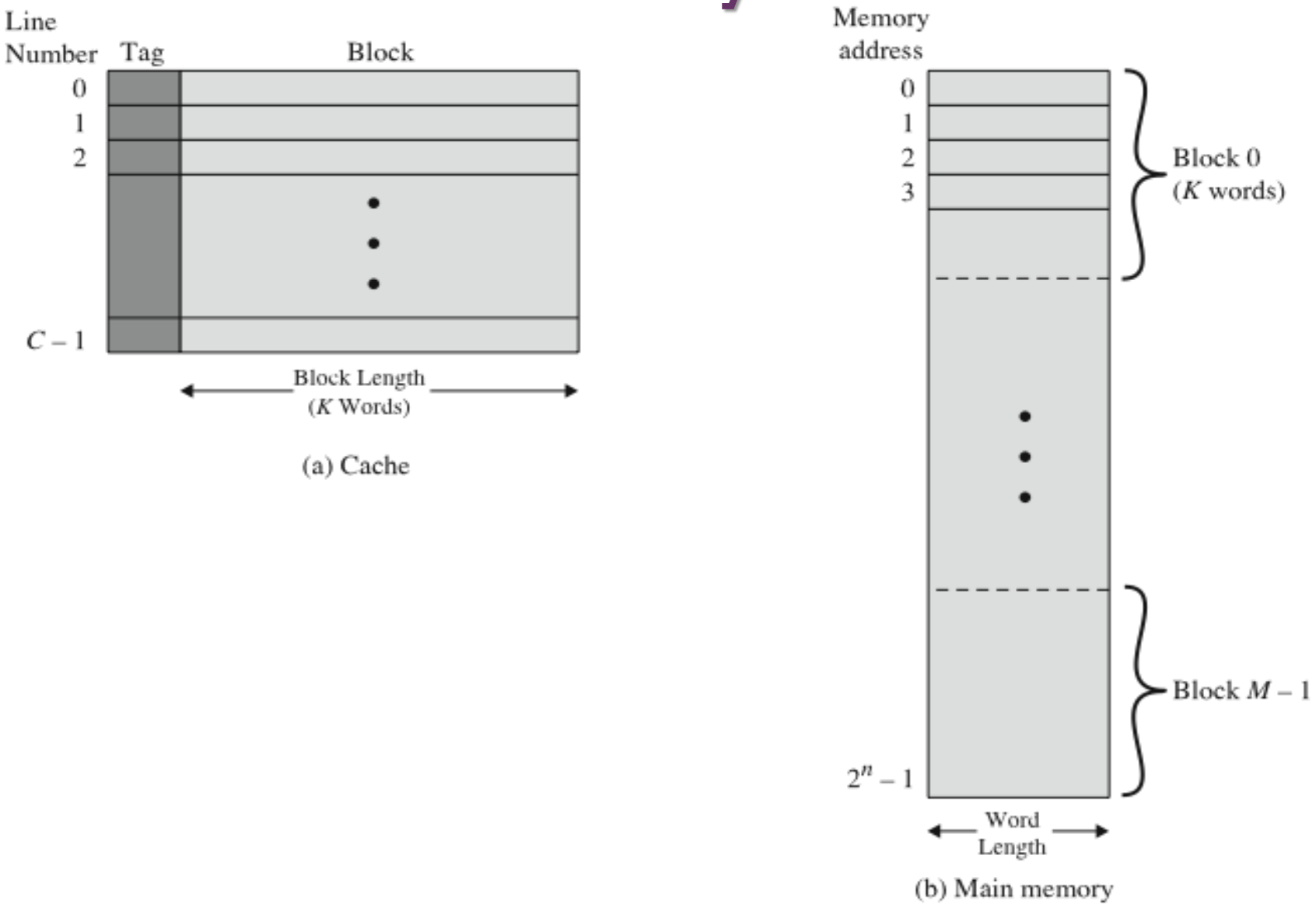
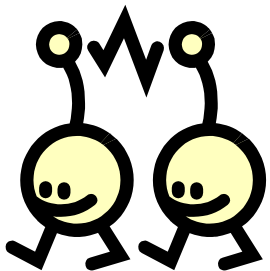


Figure 4.4 Cache/Main-Memory Structure

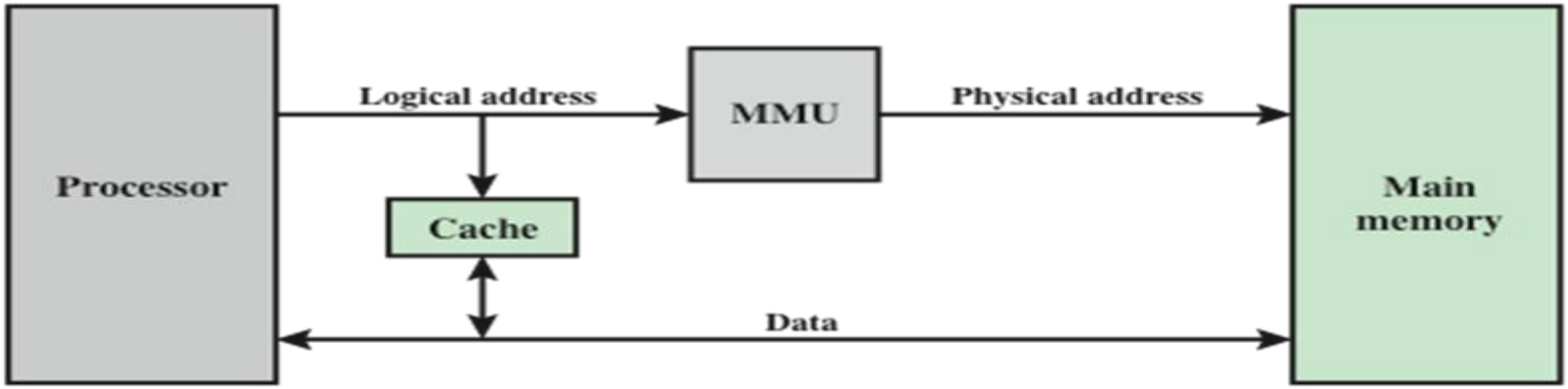
Cache Addresses

Virtual Memory

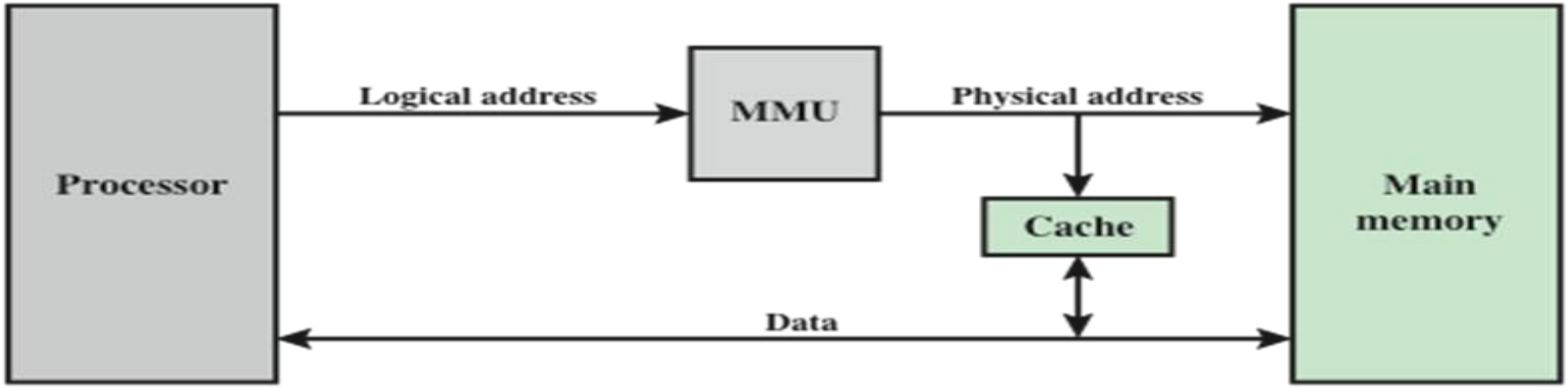
- Virtual memory
  - Facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
  - When used, the address fields of machine instructions contain virtual addresses
  - For reads to and writes from main memory, a hardware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



Logical and physical Caches



(a) Logical Cache



(b) Physical Cache

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

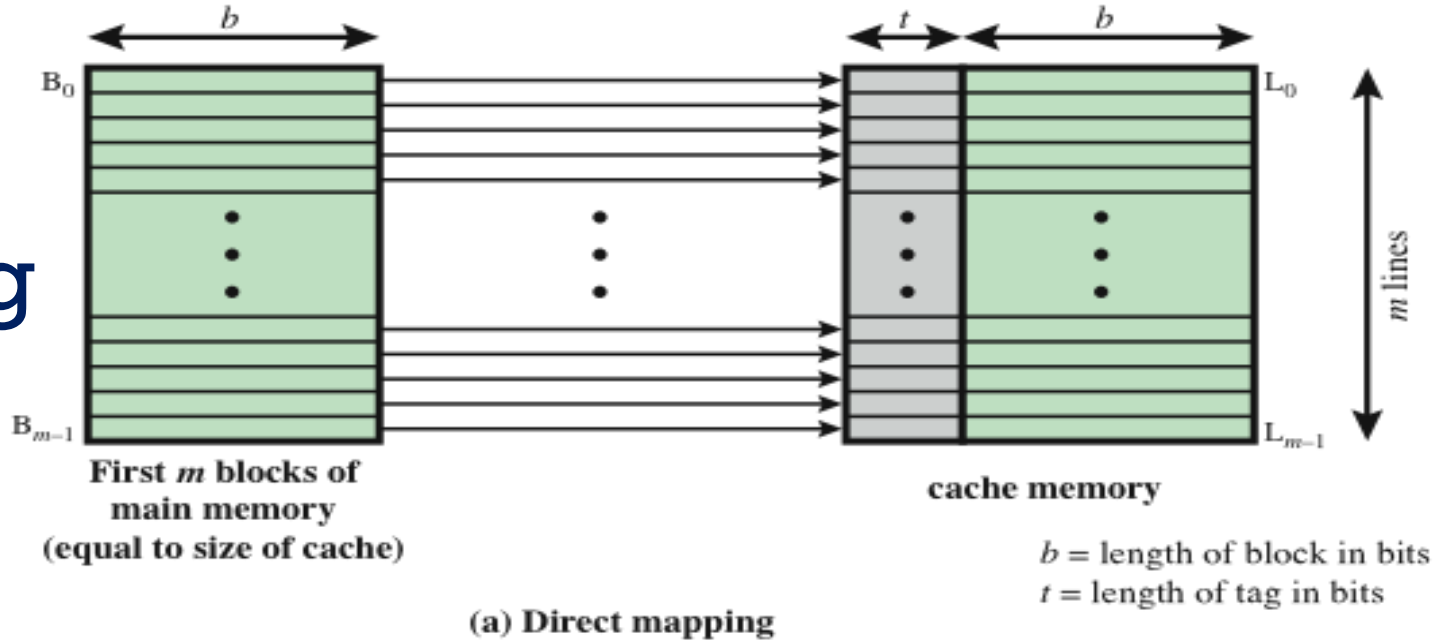
- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

- A person who has to live in one room is **direct mapping**. It is very easy to find him. It is the way in which memory blocks are stored/mapped to the lines in cache. Essentially, each block in main memory maps to a specific line in cache. But there is a restriction for him to not move anywhere.

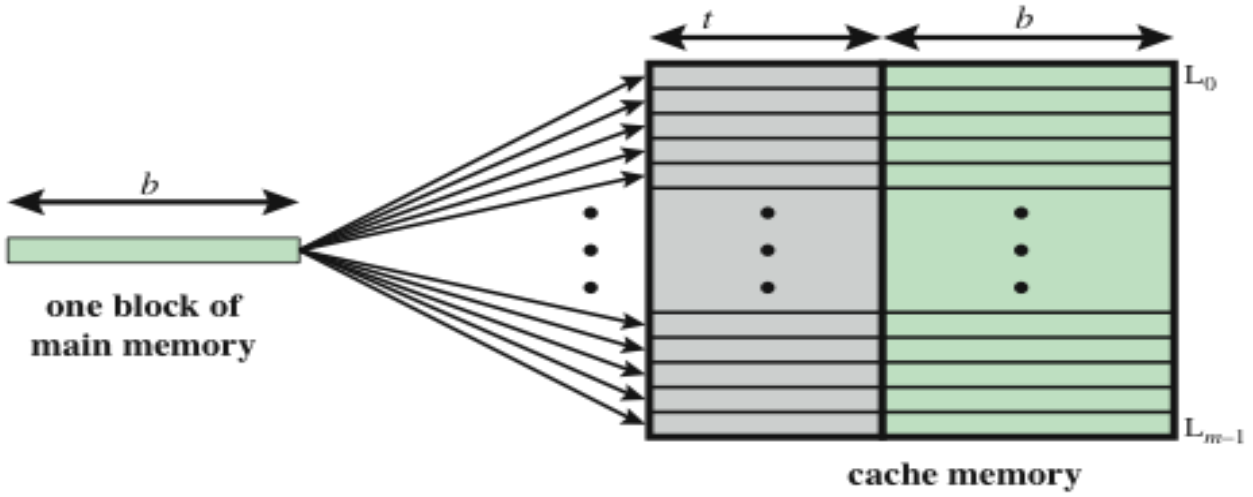
- **Fully associative** means a person can live anywhere in Malaysia so searching time is increase but restriction removed. The Tag is relatively long and must be compared to every line in the cache. A block of main memory can be mapped to any freely available cache line.

- Set associative mapping is a cache mapping technique that allows to map a block of main memory to only one particular set of cache.
- With k-way set associative mapping,. the Tag is much smaller and is only compared to the number of Tags within a single set. This technique is very robust, as the number of sets increases, the hit ratio increases.

Direct Mapping

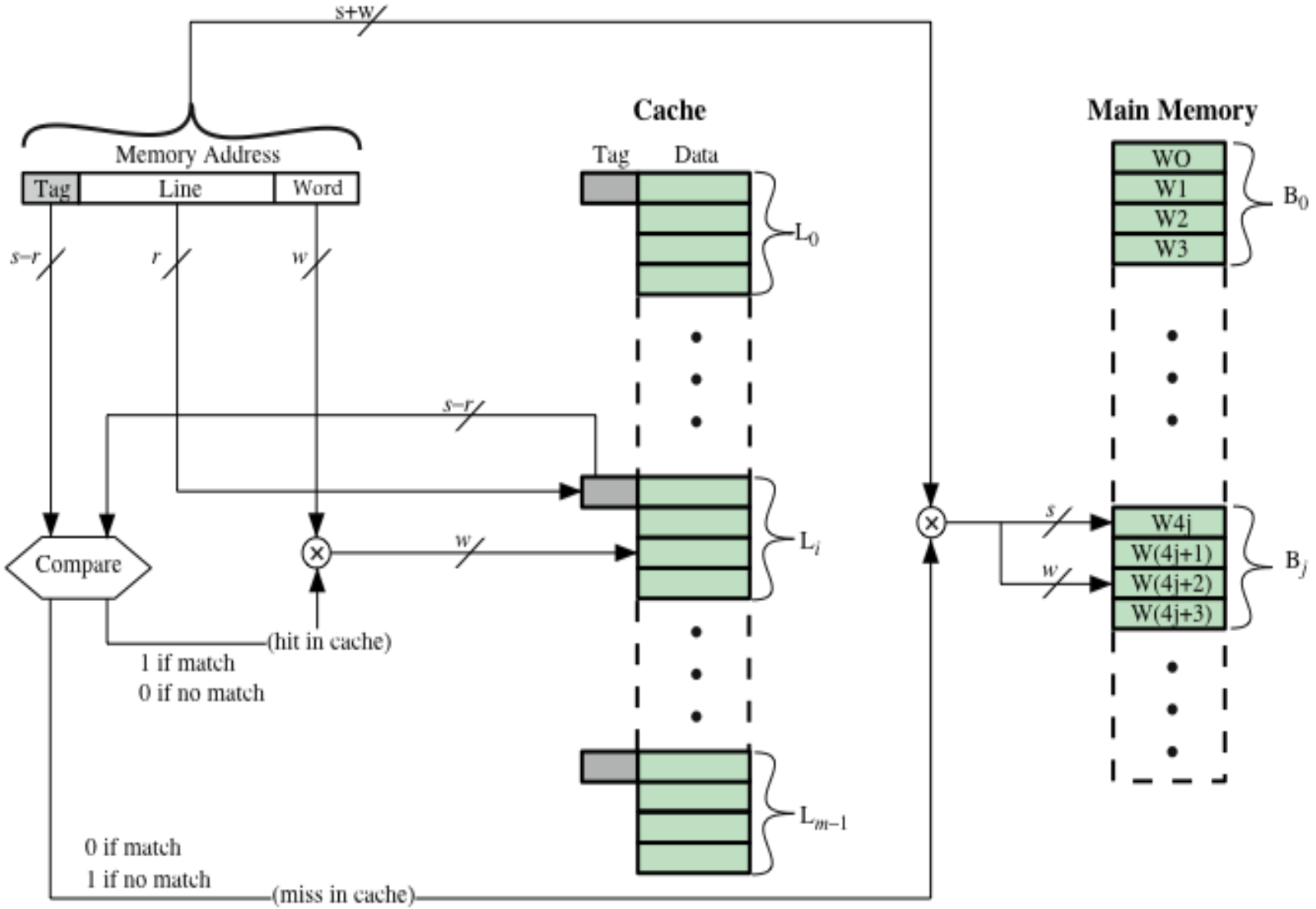


(a) Direct mapping

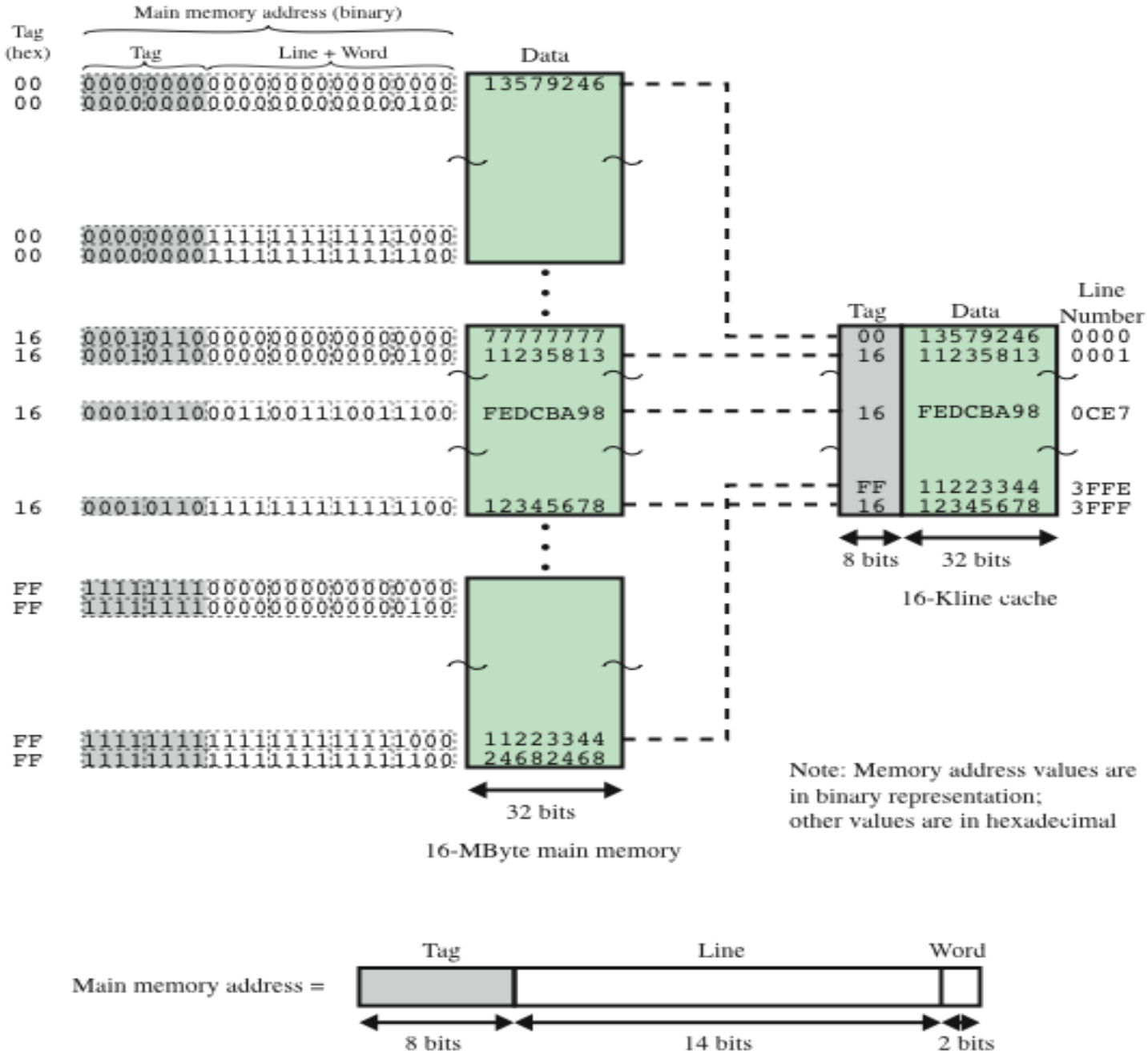


(b) Associative mapping

Direct Mapping Cache Organization

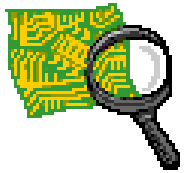


Direct Mapping Example

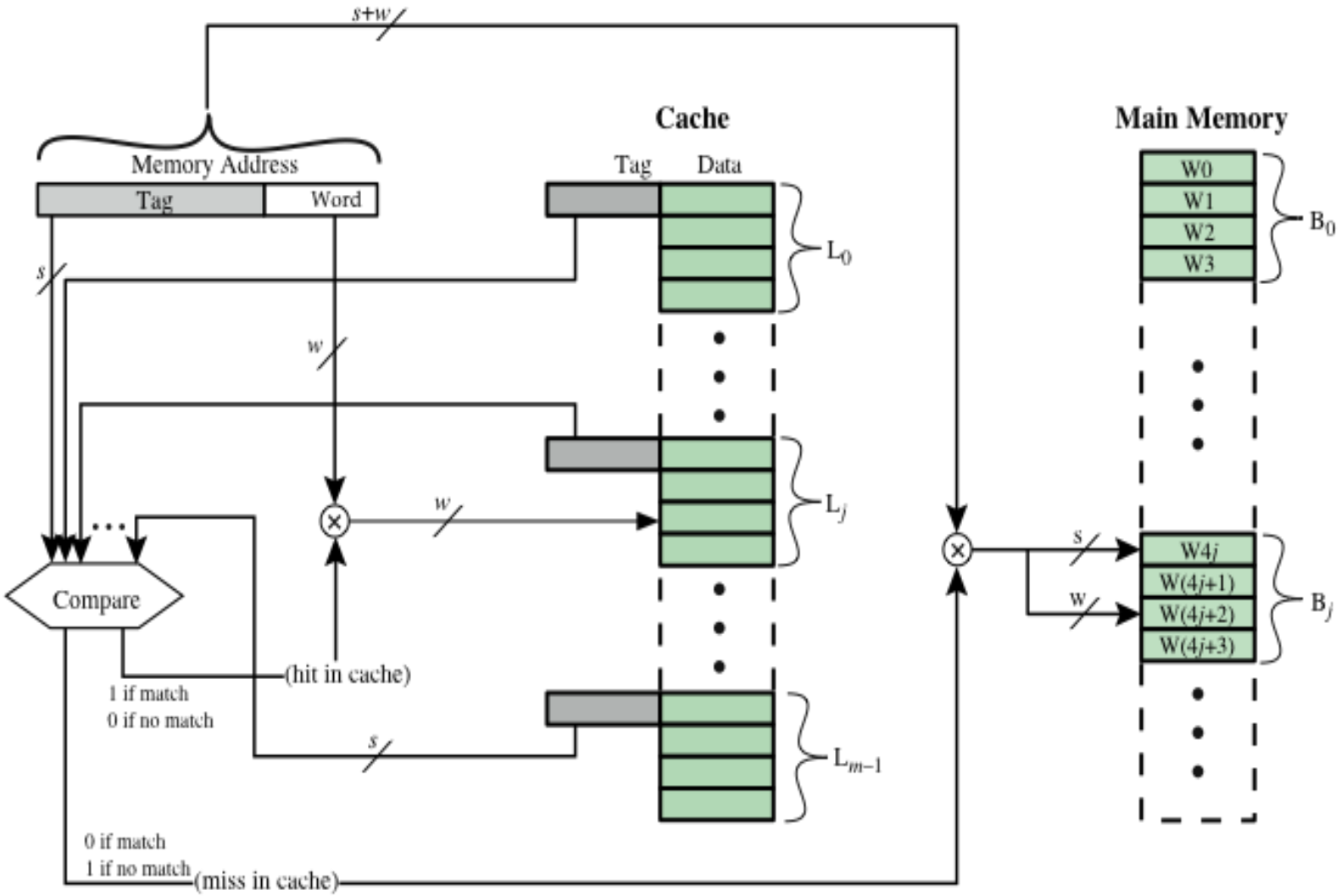


Direct Mapping Summary

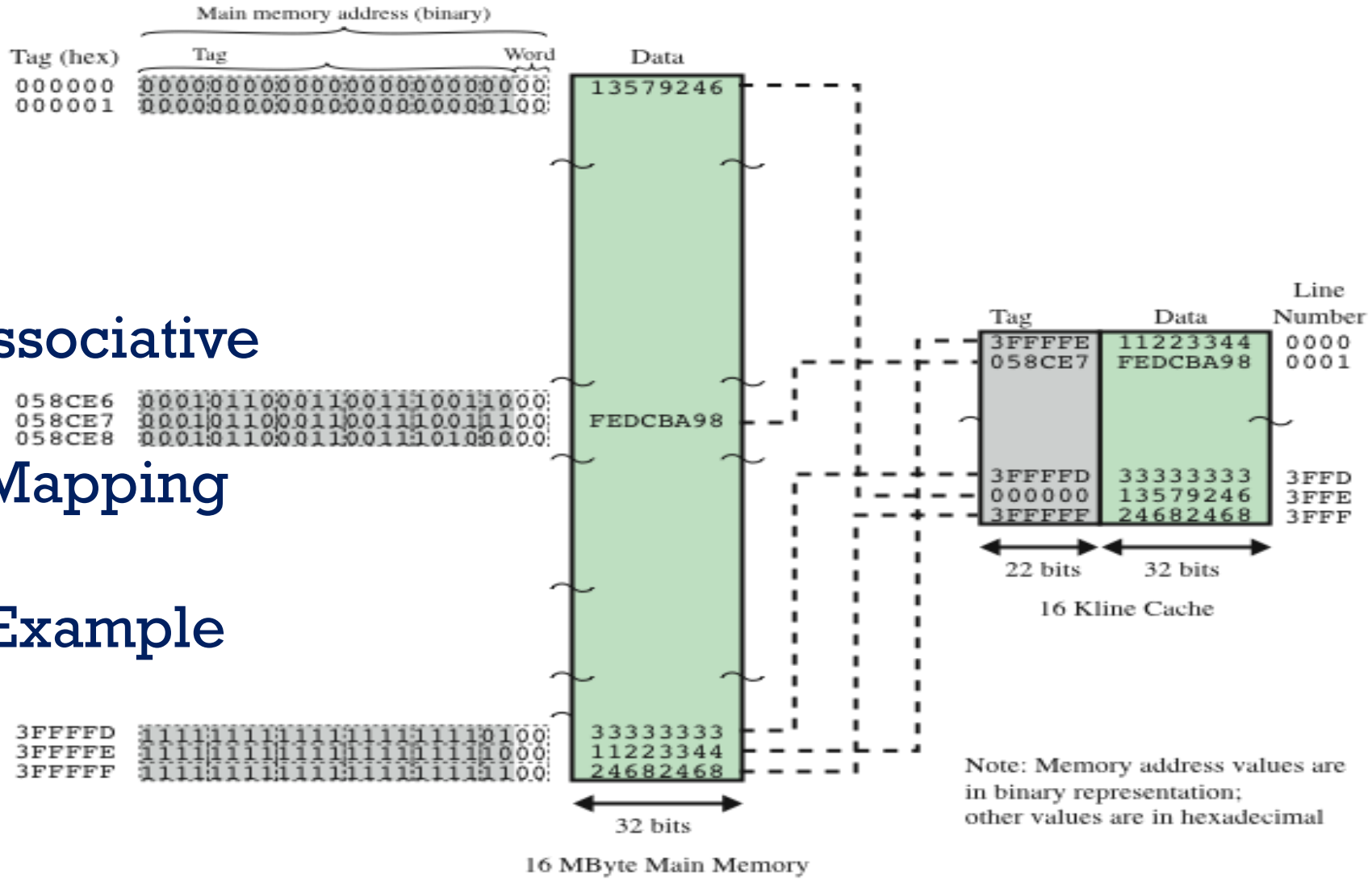
- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits



Fully Associative Cache Organization

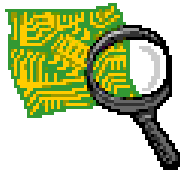


Associative Mapping Example



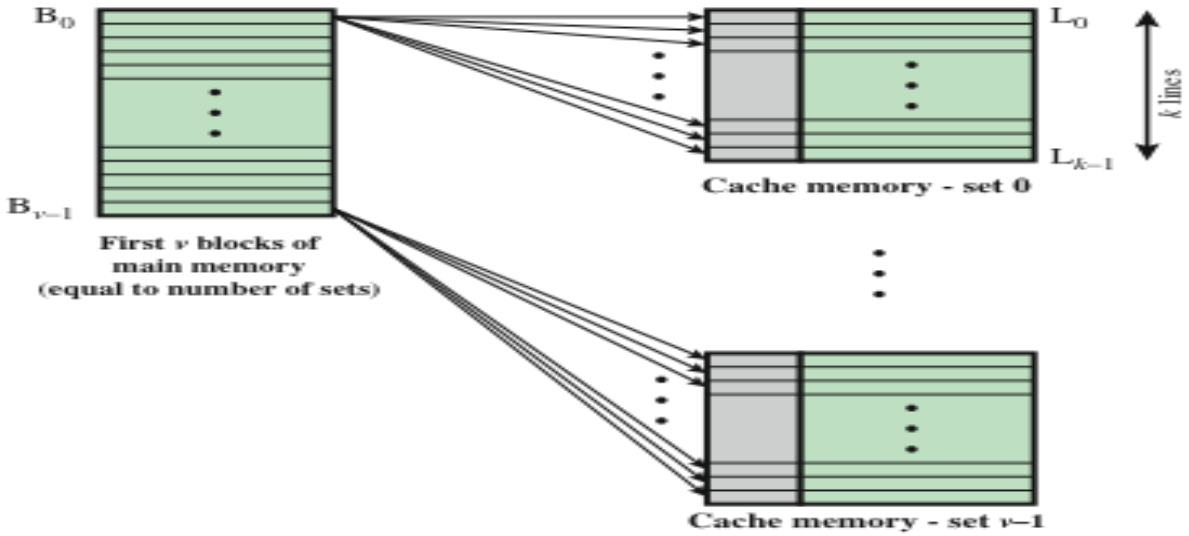
Associative Mapping Summary

- 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

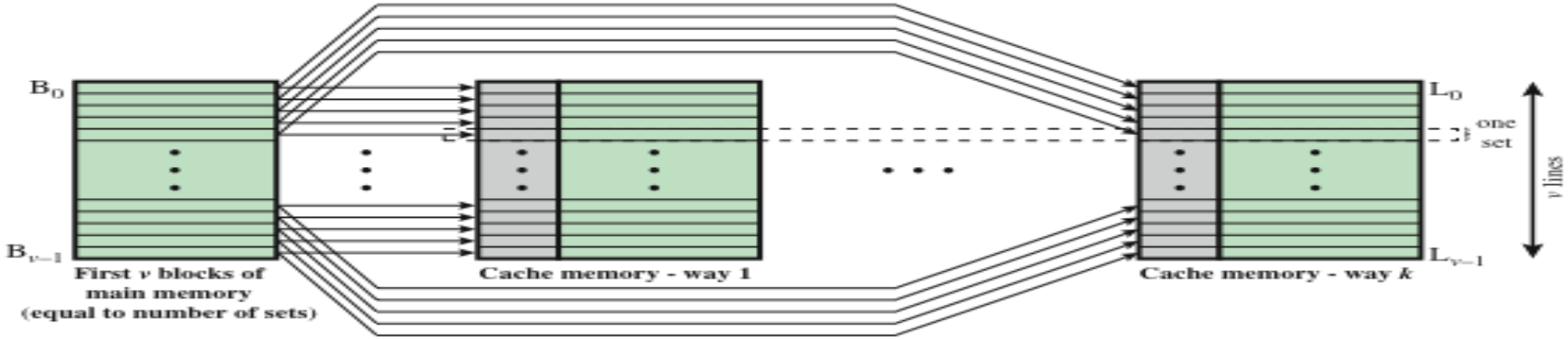


Set Associative Mapping

- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  - A given block can be in one of 2 lines in only one set



(a) v associative-mapped caches



(b) k direct-mapped caches

Figure 4.13 Mapping From Main Memory to Cache: *k*-way Set Associative

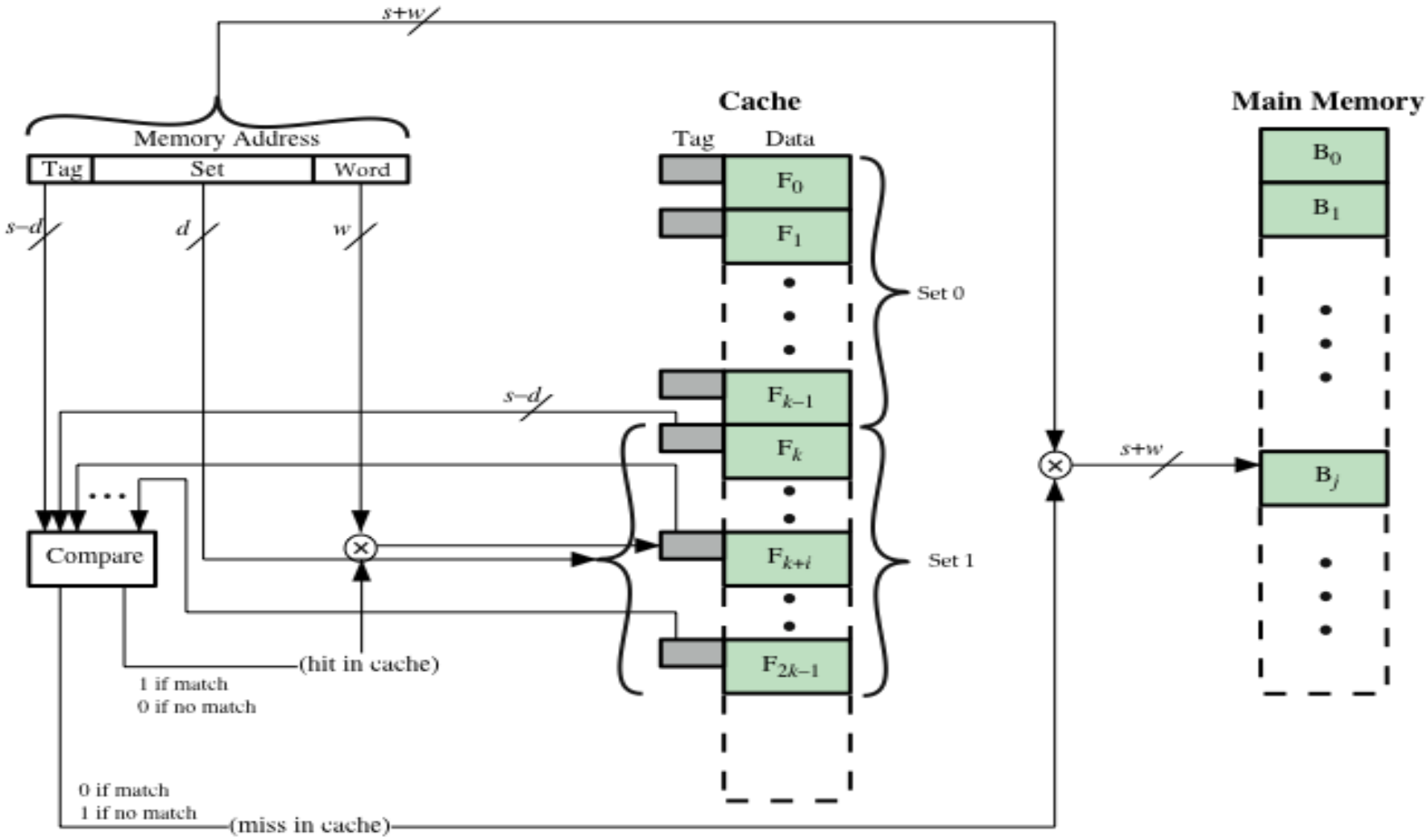


Figure 4.14 *k*-Way Set Associative Cache Organization

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 = m=kv = k * 2^{d}
- Size of cache = $k * 2^{d+w}$ words or bytes
- Size of tag = (s – d) bits

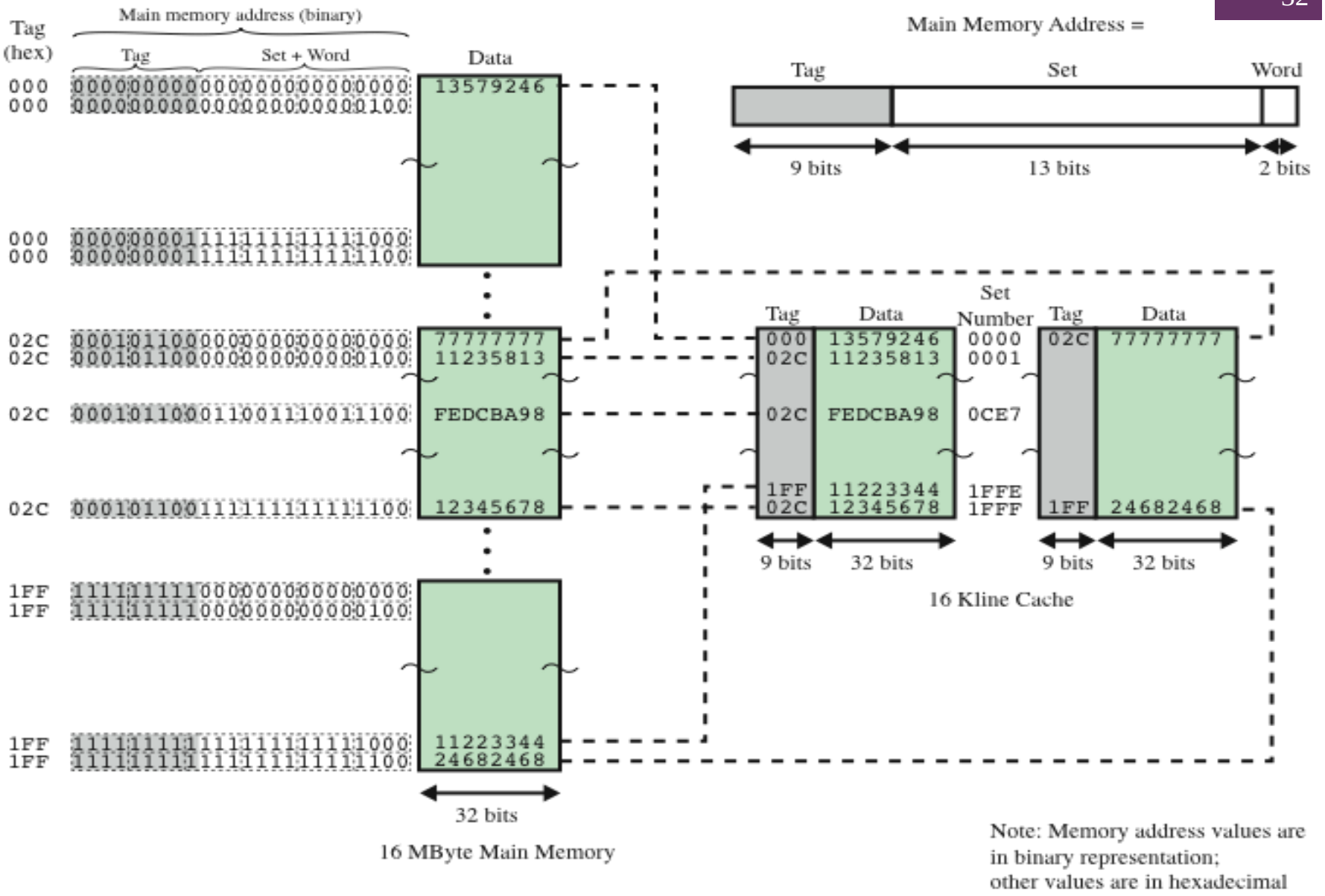


Figure 4.15 Two-Way Set Associative Mapping Example

Varying Associativity Over Cache Size

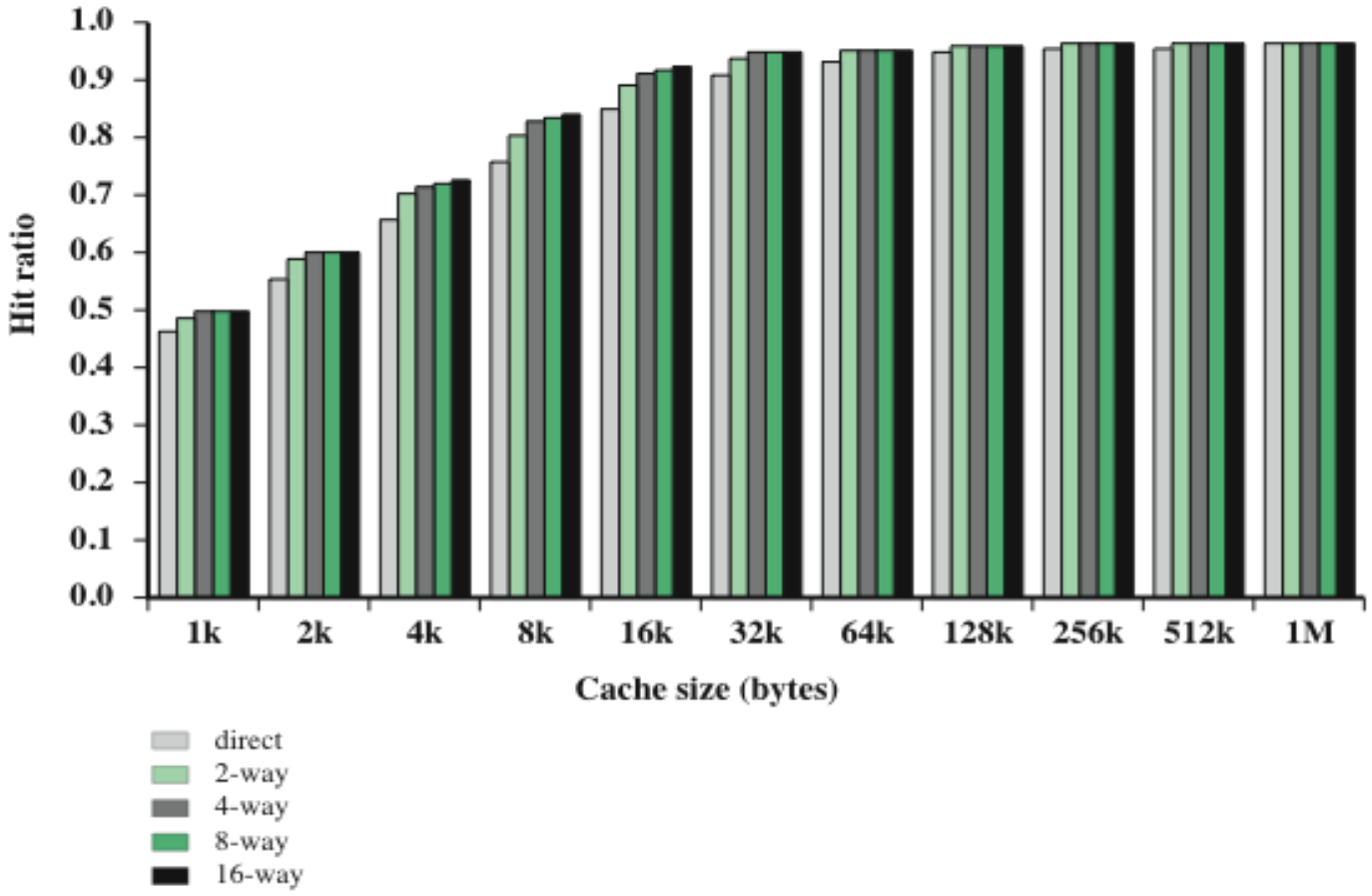


Figure 4.16 Varying Associativity over Cache Size

Example

a) A computer system has a memory architecture consists of main memory of **64GB** and cache of **32768KB**. In order to perform an efficient mapping function, the main memory is arranged in blocks of **512 bytes**. (Assuming 1 KB = 1024B, 1MB = 1024KB, 1GB = 1024MB.)

Draw the address structures for different memory mapping functions as stated below. Indicate the fields and the number of bits required for each field.

i. **Direct Mapping**
ii. **Thirty Two-Way Set Associative Mapping**
iii. **Associative Mapping**

1 KB = 1024B, 1MB = 1024KB, 1GB = 1024MB

$1024B = 2^{10}$

$1MB = 1024KB = (1024 \times 1024)\ B = 2^{10} \times 2^{10} = 2^{20}\ B$

$1GB = 1024MB = (1024 \times 1024 \times 1024)\ B = 2^{10} \times 2^{10} \times 2^{10} = 2^{30}\ B$

solution

i. Direct Mapping

a) Main Memory = 64GB = 64 X 1GB = 2^6 X 2^{30} = 2^{36}

Address length in Main Memory = 36-bit address length

Cache memory = 32768 Kbytes

Block size = 512 bytes

b) Total lines in Cache = cache memory / block size

= 32768 Kbytes/512 bytes

= 64KB = 64 x 1KB = (2^6 X 2^{10}) B = 2^{16} <-------- 16 bit of lines

c) Block size = 512 bytes = 2^9 <--------- 9-bit of word

Tag = 36 bits – 16 bits – 9 bits = 11 bits

Tag	Line	Word
11	16	9

ii) Thirty Two-Way Set Associative Mapping

a) Main Memory = 64GB = 2^{36}

Address length in Main Memory = 36-bit address length

Cache memory = 32768 Kbytes

Block size = 512 bytes

b) Total lines in Cache = cache memory / block size

= 32768 Kbytes/512 bytes

= 64K = 2^{16}

Total bits per set = total lines / K way

= 64K / 32 = 2K = 2^{11} <-------- 11 bit of set

c) Block size = 512 bytes = 2^{9} <--------- 9-bit of word

Tag = 36 bits – 11 bits – 9 bits = 16 bits

Tag	set	Word
16	11	9

iii. **Associative Mapping**

a) Main Memory = 64GB = 2^{36}

Address length in Main Memory = 36-bit address length

b) Block size = 512 bytes = 2^{9} <--------- 9-bit of word

Tag = 36 bits – 9 bits = 27 bits

Tag	Word
27	9

Example 2

- A computer system has a memory architecture consists of main memory of 4GB and cache of 8192KB. In order to perform an efficient mapping function, the main memory is arranged in blocks of 256 bytes. (Assuming 1 KB = 1024B, 1MB = 1024KB, 1GB = 1024MB.)

- Draw the address structures for different memory mapping functions as stated below. Indicate the fields and the number of bits required for each field.

i. Direct Mapping

ii. Sixteen -Way Set Associative Mapping

iii. Associative Mapping

solution

i. Direct Mapping

a) Main Memory = 4GB = 2^{32}

Address length in Main Memory = 32-bit address length

Cache memory = 8192 Kbytes

Block size = 256 bytes

b) Total lines in Cache = cache memory / block size

= 8192 Kbytes/256 bytes

= 32K = 2^{15} <-------- 15 bit of lines

c) Block size = 256 bytes = 2^{8} <--------- 8-bit of word

Tag = 32 bits – 15 bits – 8 bits = 9 bits

Tag	Line	Word
9	15	8

) sixteen way set Associative Mapping

a) Main Memory = 4GB = 2^{32}

Address length in Main Memory = 32-bit address length

Cache memory = 8192 Kbytes

Block size = 256 bytes

b) Total lines in Cache = cache memory / block size

= 8192 Kbytes/256 bytes

= 32K = 2^{15}

Total bits per set = total lines / K way

= 32K / 16 WAY = 2K = 2 X 1K = 2^1 X 2^{10} = 2^{11} <-------- 11 bit of set

c) Block size = 256 bytes = 2^8 <--------- 8-bit of word

Tag = 32 bits – 11 bits – 8 bits = 13 bits

Tag	Set	Word
13	11	8

iii. **Associative Mapping**

a) Main Memory = 4GB = 2^{32}

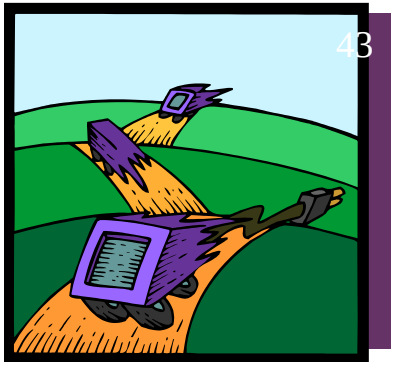
Address length in Main Memory = 32-bit address length

b) Block size = 256 bytes = 2^8 <--------- 8-bit of word

Tag = 32 bits – 8 bits = 24 bits

Tag	Word
24	8

Replacement Algorithms



- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
- For direct mapping there is only one possible line for any particular block and no choice is possible
- For the associative and set-associative techniques a replacement algorithm is needed
- To achieve high speed, an algorithm must be implemented in hardware

The three most common replacement algorithms are:

- Least recently used (LRU)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the most popular replacement algorithm
- First-in-first-out (FIFO)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Optimal page replacement (OPT)
  - Replace that block in the set that has experienced the fewest references
  - Could be implemented by associating a counter with each line

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

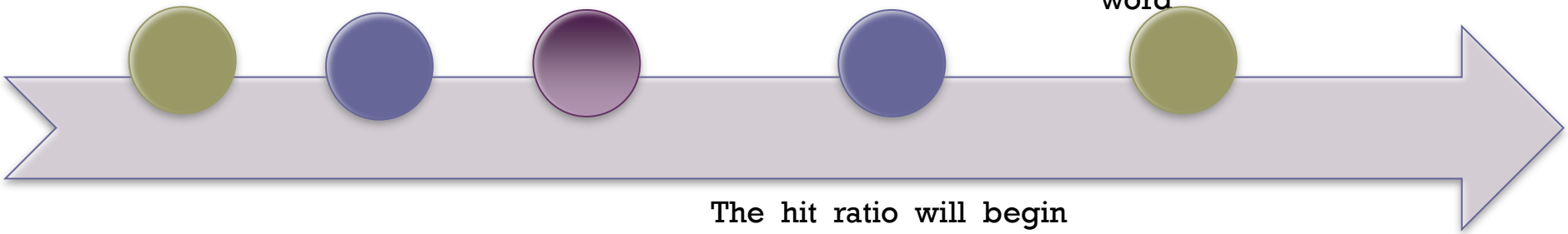
Line Size

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

Multilevel Caches

- As logic density has increased it has become possible to have a cache on the same chip as the processor
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete appreciably faster than would even zero-wait state bus cycles
  - During this period the bus is free to support other transfers
- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings due to the use of an L2 cache depends on the hit rates in both the L1 and L2 caches
- The use of multilevel caches complicates all of the design issues related to caches, including size, replacement algorithm, and write policy

Hit Ratio (L1 & L2)
For 8 Kbyte and 16 Kbyte L1

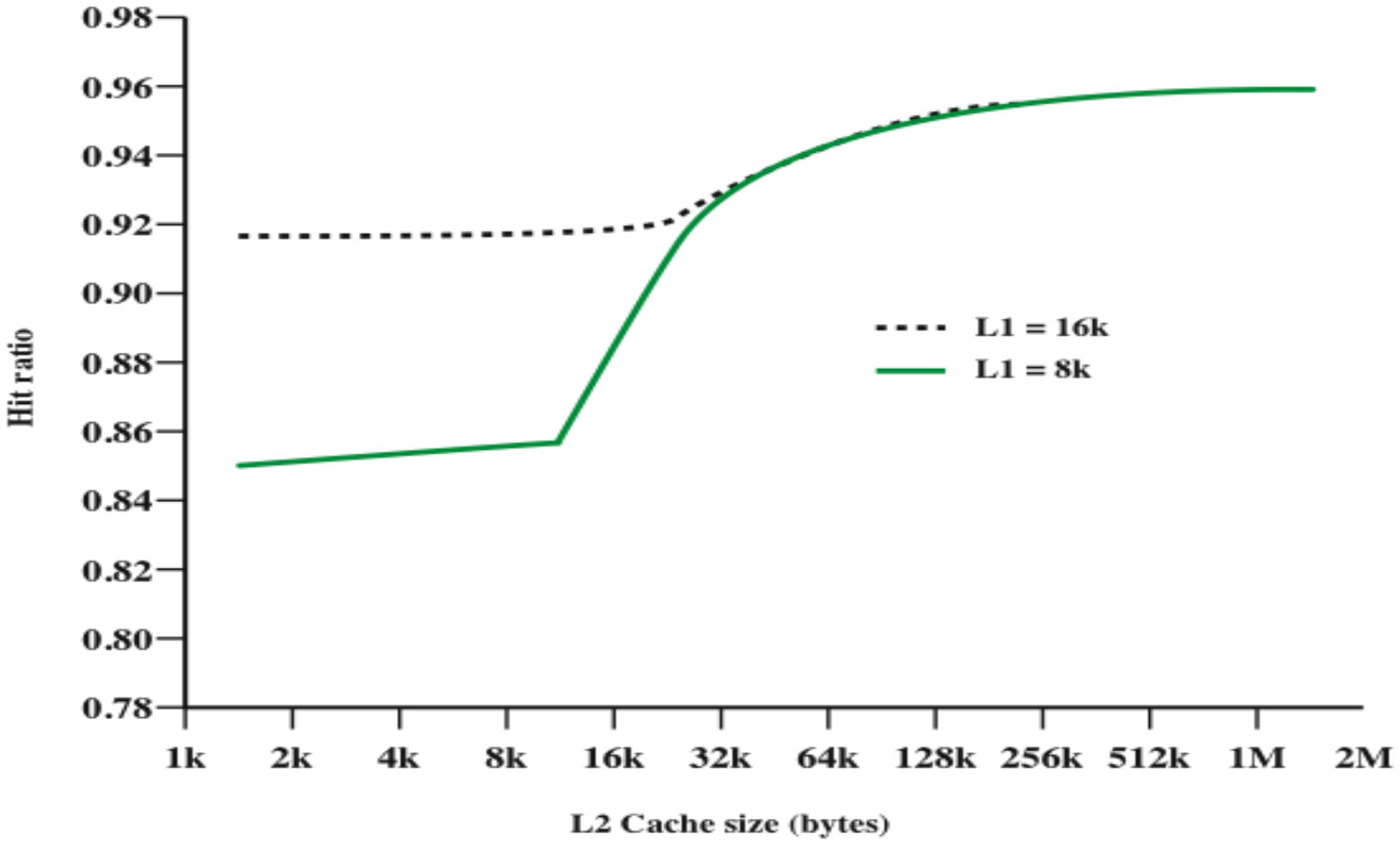


Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

Unified Versus Split Caches

- Has become common to split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load of instruction and data fetches automatically
    - Only one cache needs to be designed and implemented
- Trend is toward split caches at the L1 and unified caches for higher levels
- Advantages of split cache:
  - Eliminates cache contention between instruction fetch/decode unit and execution unit
    - Important in pipelining

Internal Memory - Memory Cell Operation

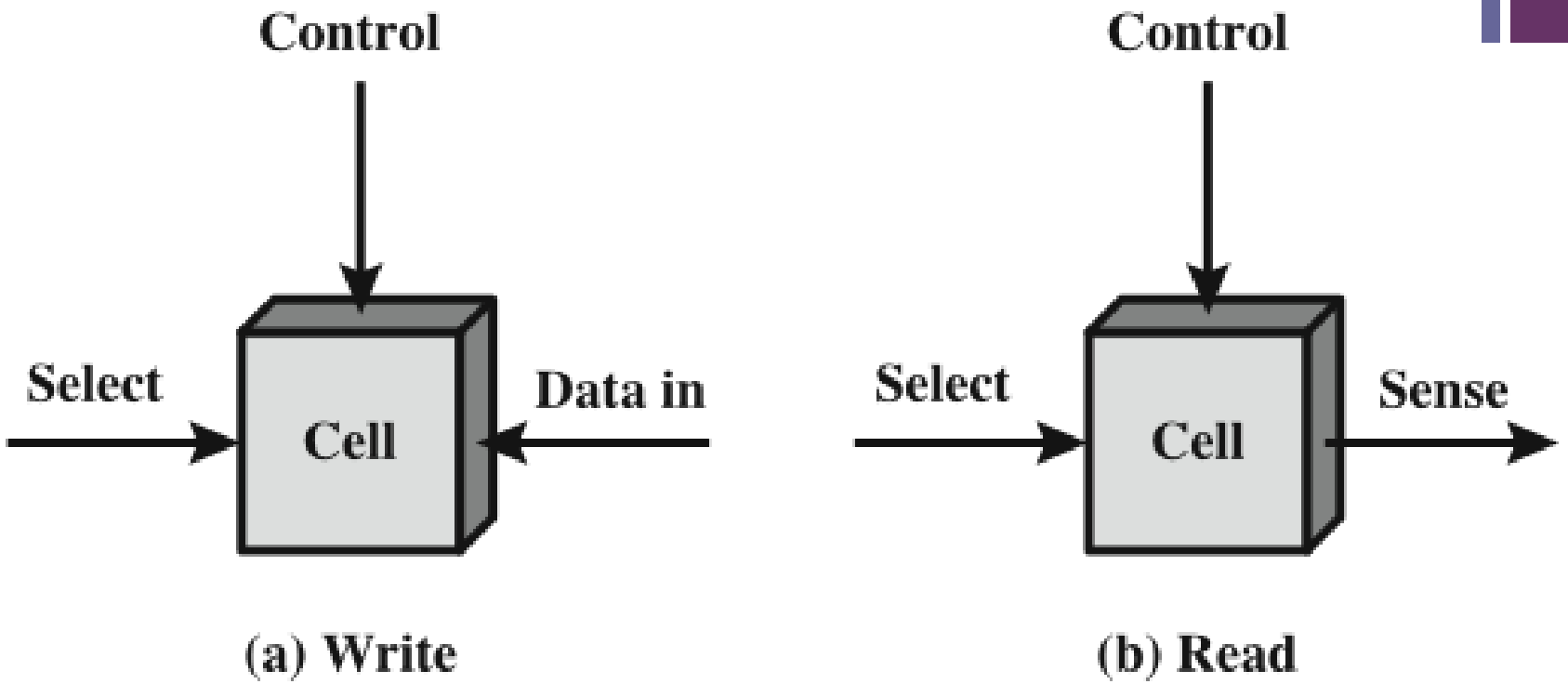


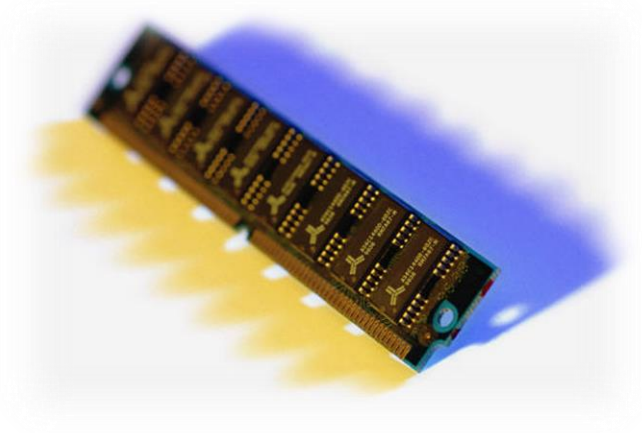
Figure 5.1 Memory Cell Operation

Dynamic RAM (DRAM)

- RAM technology is divided into two technologies:
  - Dynamic RAM (DRAM)
  - Static RAM (SRAM)
- DRAM
  - Made with cells that store data as charge on capacitors
  - Presence or absence of charge in a capacitor is interpreted as a binary 1 or 0
  - Requires periodic charge refreshing to maintain data storage
  - The term *dynamic* refers to tendency of the stored charge to leak away, even with power continuously applied

Static RAM (SRAM)

- Digital device that uses the same logic elements used in the processor
- Binary values are stored using traditional flip-flop logic gate configurations
- Will hold its data as long as power is supplied to it



SRAM versus DRAM

- Both volatile
  - Power must be continuously supplied to the memory to preserve the bit values
- Dynamic cell
  - Simpler to build, smaller
  - More dense (smaller cells = more cells per unit area)
  - Less expensive
  - Requires the supporting refresh circuitry
  - Tend to be favored for large memory requirements
  - Used for main memory
- Static
  - Faster
  - Used for cache memory (both on and off chip)

SRAM

DRAM

Performance Comparison

DRAM Alternatives

Table 5.3

	Clock Frequency (MHz)	Transfer Rate (GB/s)	Access Time (ns)	Pin Count
SDRAM	166	1.3	18	168
DDR	200	3.2	12.5	184
RDRAM	600	4.8	12	162

Table 5.3 Performance Comparison of Some DRAM Alternatives

Read Only Memory (ROM)

- Contains a permanent pattern of data that cannot be changed or added to
- No power source is required to maintain the bit values in memory
- Data or program is permanently in main memory and never needs to be loaded from a secondary storage device
- Data is actually wired into the chip as part of the fabrication process
  - Disadvantages of this:
    - No room for error, if one bit is wrong the whole batch of ROMs must be thrown out
    - Data insertion step includes a relatively large fixed cost

Programmable ROM (PROM)

- Less expensive alternative
- Nonvolatile and may be written into only once
- Writing process is performed electrically and may be performed by supplier or customer at a time later than the original chip fabrication
- Special equipment is required for the writing process
- Provides flexibility and convenience
- Attractive for high volume production runs

Read-Mostly Memory

EPROM

- Erasable programmable read-only memory
- Erasure process can be performed repeatedly
- More expensive than PROM but it has the advantage of the multiple update capability

EEPROM

- Electrically erasable programmable read-only memory
- Can be written into at any time without erasing prior contents
- Combines the advantage of non-volatility with the flexibility of being updatable in place
- More expensive than EPROM

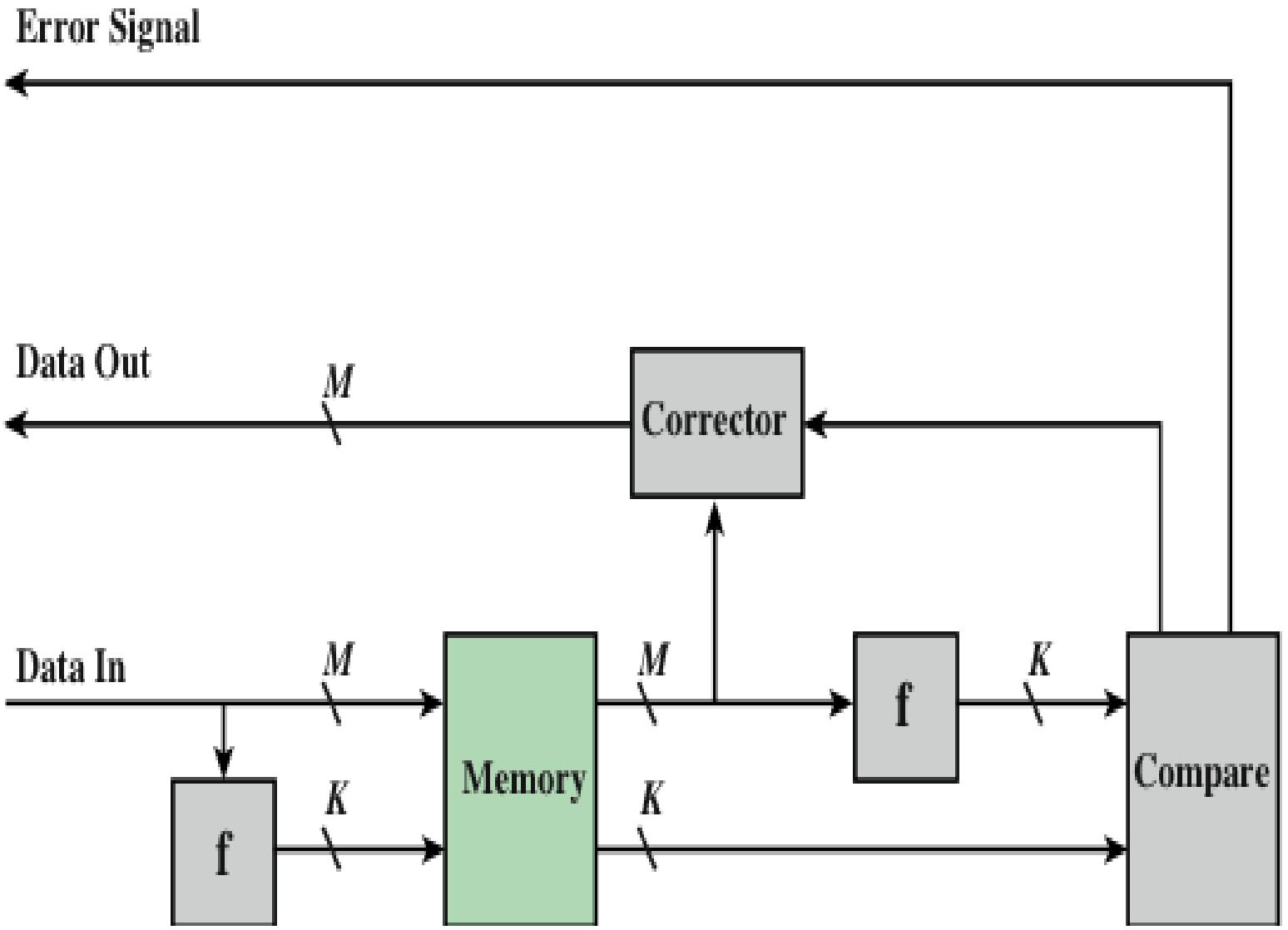
Flash Memory

- Intermediate between EPROM and EEPROM in both cost and functionality
- Uses an electrical erasing technology, does not provide byte-level erasure
- Microchip is organized so that a section of memory cells are erased in a single action or “flash”

Error Correction

- Hard Failure
  - Permanent physical defect
  - Memory cell or cells affected cannot reliably store data but become stuck at 0 or 1 or switch erratically between 0 and 1
  - Can be caused by:
    - Harsh environmental abuse
    - Manufacturing defects
    - Wear
- Soft Error
  - Random, non-destructive event that alters the contents of one or more memory cells
  - No permanent damage to memory
  - Can be caused by:
    - Power supply problems
    - Alpha particles - alpha particles emitted by traces of radioactive elements (such as thorium and uranium) present in the packaging materials of the device. These alpha particles manage to penetrate the die and generate a high density of holes and electrons in its substrate, which creates an imbalance in the device's electrical potential distribution that causes stored data to be corrupted.

Error Correcting Code Function



Error Correcting Code Function

- Additional information must be stored to detect these errors
- When M bits of data are stored, they are run through function *f* where a K bit code is created
- M+K bits are then stored in memory
- When data is read out, it is once again run through function f and the resulting K bits of code are compared with the stored K bits of code
- In some cases, the code can be corrected (error correcting codes)

Three possible results

- No errors detected – sent out data
- An error is detected and can be corrected – sent out corrected data
- An error is detected and can not be corrected – report error

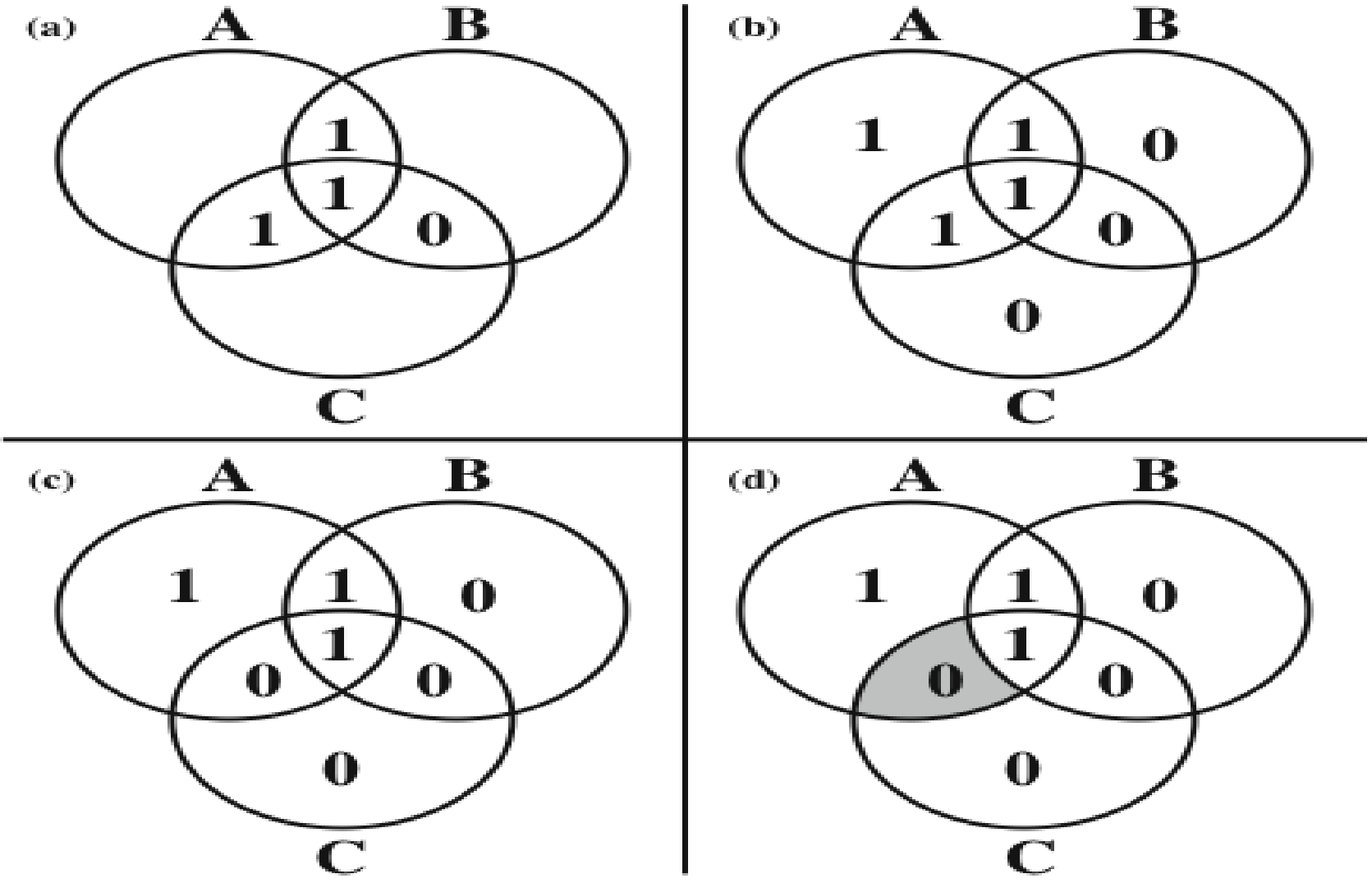


Figure 5.8 Hamming Error-Correcting Code

Layout of Data Bits and Check Bits

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data Bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check Bit					C8				C4		C2	C1

Figure 5.9 Layout of Data Bits and Check Bits

Check Bit Calculation example 1(a)

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1

Assuming the 8 bit word stored as 0011 1001 The check bits are in bit numbers 8, 4, 2, and 1.

Check bit 8 calculated by values in bit positions: 12, 11, 10 and 9→ $D8 \oplus D7 \oplus D6 \oplus D5$

$\rightarrow 0 \oplus 0 \oplus 1 \oplus 1 = 0$

Check bit 4 calculated by values in bit positions: 12, 7, 6, and 5 → $D8 \oplus D4 \oplus D3 \oplus D2$

$\rightarrow 0 \oplus 1 \oplus 0 \oplus 0 = 1$

Check bit 2 calculated by values in bit positions: 11, 10, 7, 6 and 3→ $D7 \oplus D6 \oplus D4 \oplus D3 \oplus D1$

$\rightarrow 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$

Check bit 1 calculated by values in bit positions: 11, 9, 7, 5 and 3 →
$D7 \oplus D5 \oplus D4 \oplus D2 \oplus D1 \rightarrow 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$

Thus, the check bits are: $0\ 1\ 1\ 1_2$

Bit position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Data bit	D8	D7	D6	D5		D4	D3	D2		D1		
Check bit					C8				C4		C2	C1
Word stored as	0	0	1	1	0	1	0	0	1	1	1	1
Word fetched as	0	0	1	1	0	1	1	0	1	1	1	1
Position Number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Check Bit					0				0		0	1

And assuming the 8 bit word fetched to another device as 0011 1101. Find out where is error bit

The new check bits are in bit numbers 8, 4, 2, and 1.

Check bit 8 calculated by values in bit positions: 12, 11, 10 and 9→ $D8 \oplus D7 \oplus D6 \oplus D5 \rightarrow 0 \oplus 0 \oplus 1 \oplus 1 = 0$

Check bit 4 calculated by values in bit positions: 12, 7, 6, and 5 → $D8 \oplus D4 \oplus D3 \oplus D2 \rightarrow 0 \oplus 1 \oplus 1 \oplus 0 = 0$

Check bit 2 calculated by values in bit positions: 11, 10, 7, 6 and 3→ $D7 \oplus D6 \oplus D4 \oplus D3 \oplus D1 \rightarrow 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$

Check bit 1 calculated by values in bit positions: 11, 9, 7, 5 and 3 → $D7 \oplus D5 \oplus D4 \oplus D2 \oplus D1 \rightarrow 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1$

Thus, the check bits are: $0\ 0\ 0\ 1_2$

Difference of the check bits : $0111_2 - 0001_2 = 0110_2 = 6$, bit no. 3 or D3 in error

Example 1

Suppose an 8-bit data word stored in memory is 00000101. Using the Hamming algorithm, determine what check bits would be stored in memory with the data word. Show how you got your answer.

solution

Bit Position	12	11	10	9	8	7	6	5	4	3	2	1
Position number	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
Bits	D8	D7	D6	D5	C8	D4	D3	D2	C4	D1	C2	C1
Word	0	0	0	0		0	1	0		1		
Check bit					0				1		0	1

The check bits are in bit numbers 8, 4, 2, and 1.

Check bit 8 calculated by values in bit positions: 12, 11, 10 and 9 $\rightarrow 0 \oplus 0 \oplus 0 \oplus 0 = 0$ [1]

Check bit 4 calculated by values in bit positions: 12, 7, 6, and 5 $\rightarrow 0 \oplus 0 \oplus 1 \oplus 0 = 1$ [1]

Check bit 2 calculated by values in bit positions: 11, 10, 7, 6 and 3$\rightarrow 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 = 0$ [1]

Check bit 1 calculated by values in bit positions: 11, 9, 7, 5 and 3 $\rightarrow 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$ [1]

Thus, the check bits are: 0 1 0 1

Summary

Lecture B - 05

Memory Architecture

- Characteristics of Memory Systems
- Memory Hierarchy
- Cache memory principles
- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function
  - Replacement algorithms
  - Write policy
  - Line size
  - Number of caches
- Semiconductor main memory
- Error correction
  - Hard failure
  - Soft error
- Hamming code

- Slides adopted from:
  - Computer Organization and Architecture, 9th Edition

    William Stallings

    ISBN-10: 013293633X | ISBN-13: 9780132936330