Memory Organization

Outline

- Semiconductor Memory
- CPU-Memory Interaction
- Organization of Memory Modules
- Cache Memory
- Cache Mapping and Replacement Policies
- Virtual Memory

- Books Recommended:

1. John L. Hannessy, David A. Patterson, "Computer organization and Design", 3/E, Morgan Kaufmaan, reprint -2003.

2. Andrew S. Tanenbaum, "Structured Computer Organization", 6/E, PHI EEE, reprint 1995.

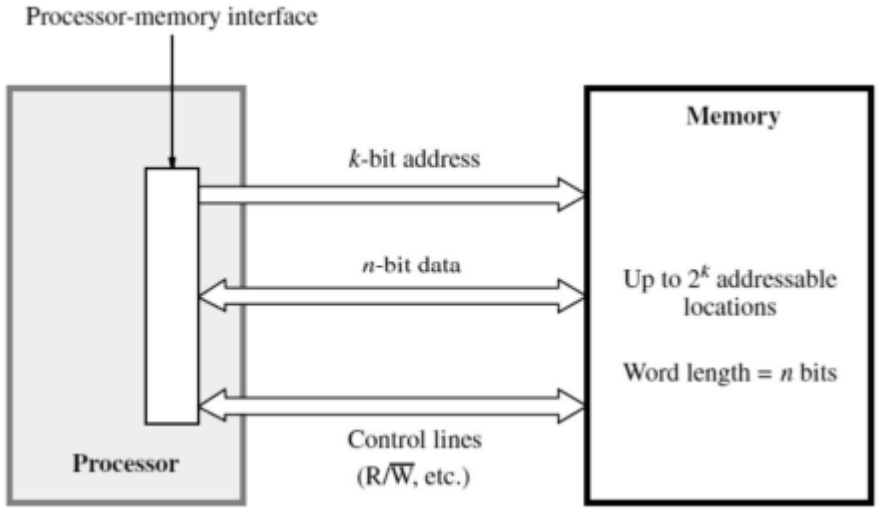
3**. William Stallings,"Computer Organization & Architecture: Designing For Performance", 6/E, PHI, 2002.**

4. Carl Hamacher, Zvonko Vranesic, Safwat Zaky, "Computer Organization", 5/E, McGraw-Hill, 2002.

5. Morris Mano, "Computer Systems Architecture", 3/E, PHI, reprint 1997.

Memory

- Programs and the data they operate on are held in the memory of the computer
- The maximum size of the memory that can be used in any computer is determined by the addressing scheme.
- For example, a computer that generates 16-bit addresses is capable of addressing up to 2^16 = 64K (kilo) memory locations.
- Machines whose instructions generate 32-bit addresses can utilize a memory that contains up to 2^32 = 4G (giga) locations,



Memory

- If memory is having 12 address lines and 8 data lines, then Number of registers/memory locations= 2^N = 2^12 = 4096 Word length = M bit = 8 bit

Memory capacity	Address Lines Required
1K = 1024 memory locations	10
2K = 2048 memory locations	11
4K=4096 memory locations	12
8K = 8192 memory locations	13
16K = 16384 memory locations	14
32K = 32768 memory locations	15
64K = 65536 memory locations	16

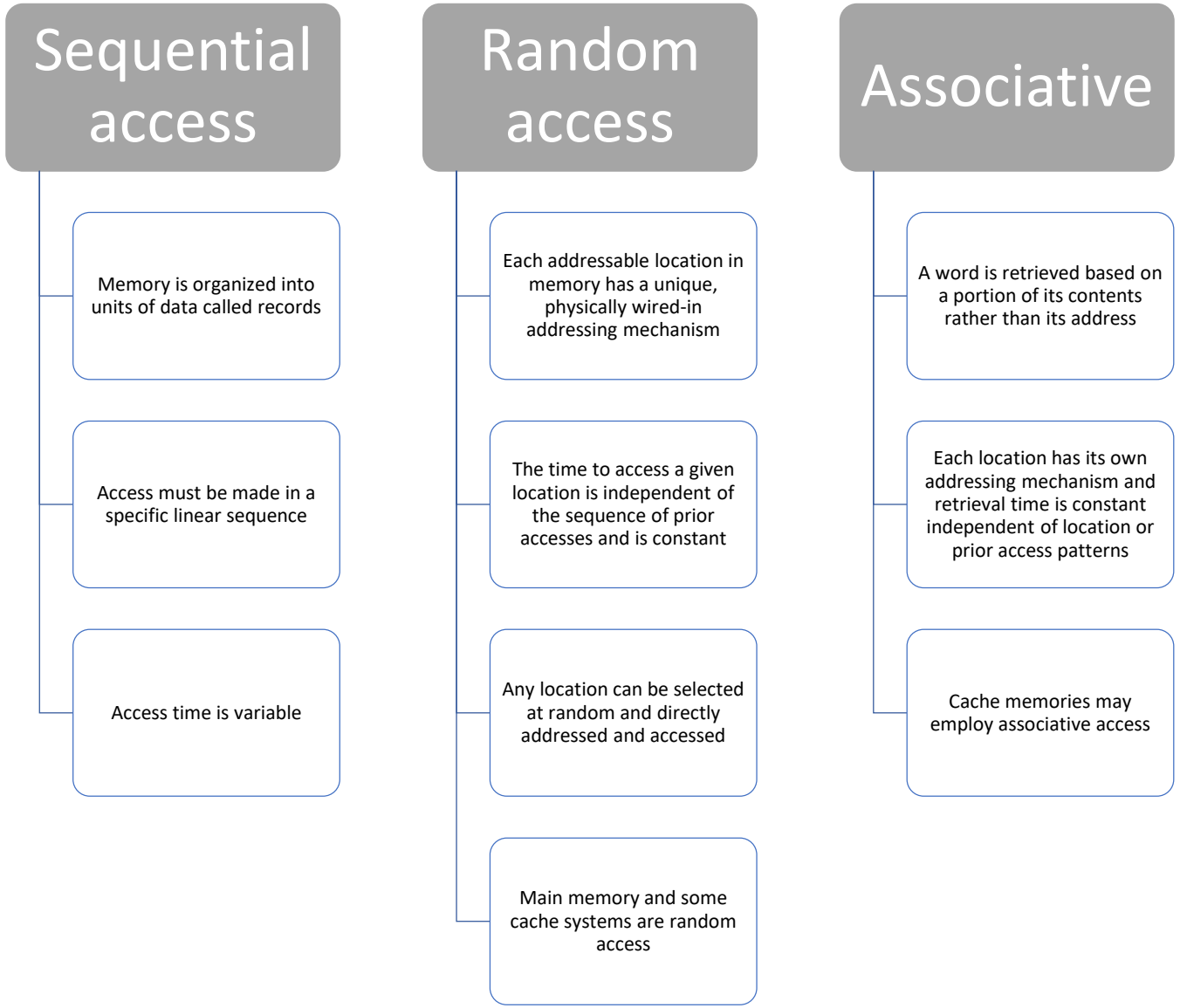
Characteristics of Memory Systems

- Location
  - Internal
  - External
- Capacity
  - Number of Words
  - Number of bytes
- Unit of Transfer
  - Word
  - Block
- Access Method
  - Sequential
  - Random
  - Associative
- Performance
  - Access Time
  - Cycle Time
  - Transfer Time
- Physical Type
  - Semi Conductor
  - Magnetic
  - Optical
  - Magneto-Optical
- Physical Characteristics
  - Volatile / Non volatile
  - Erasable/ Non Erasable
- Organization
  - Memory Modules

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 word size

Access Methods

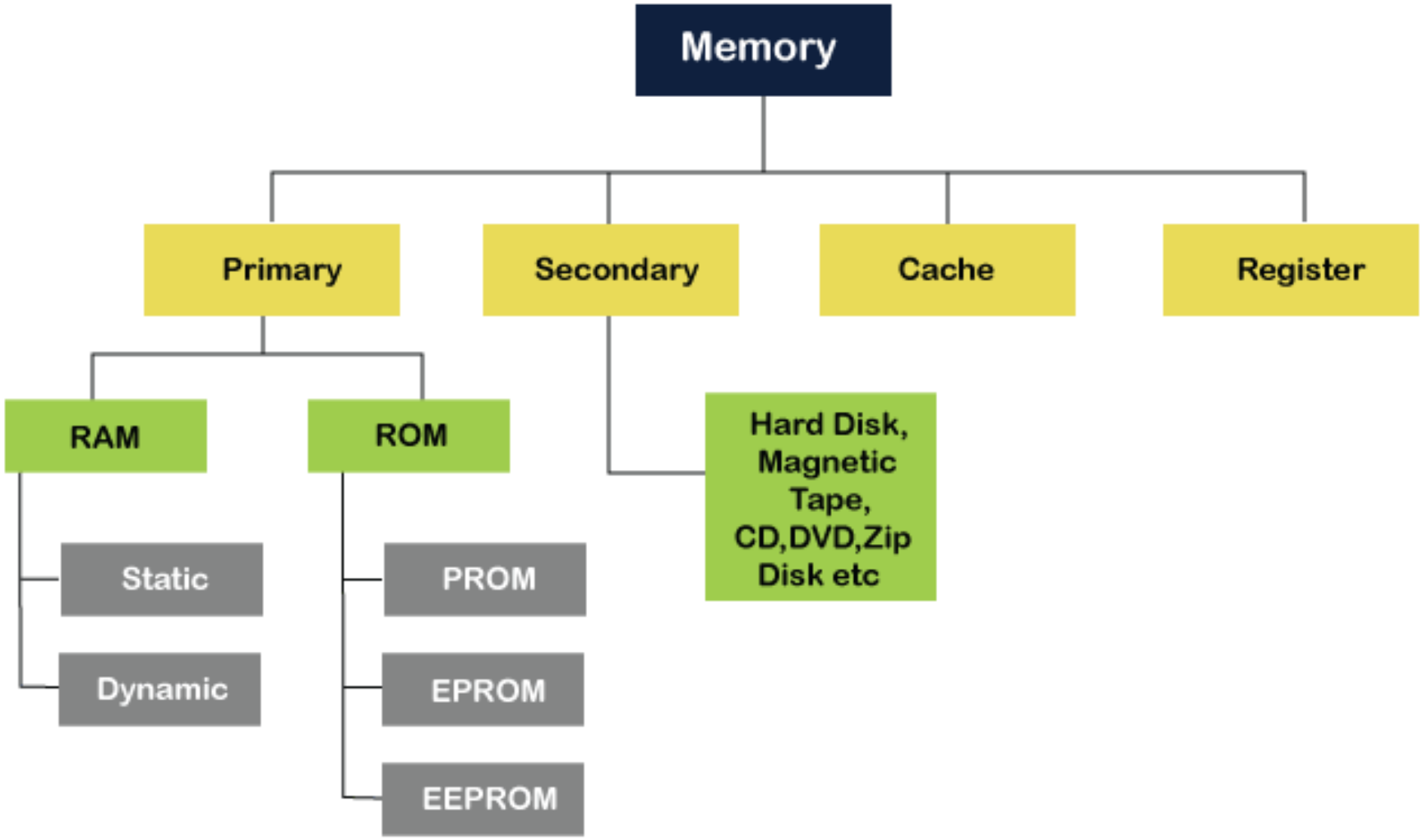


Performance

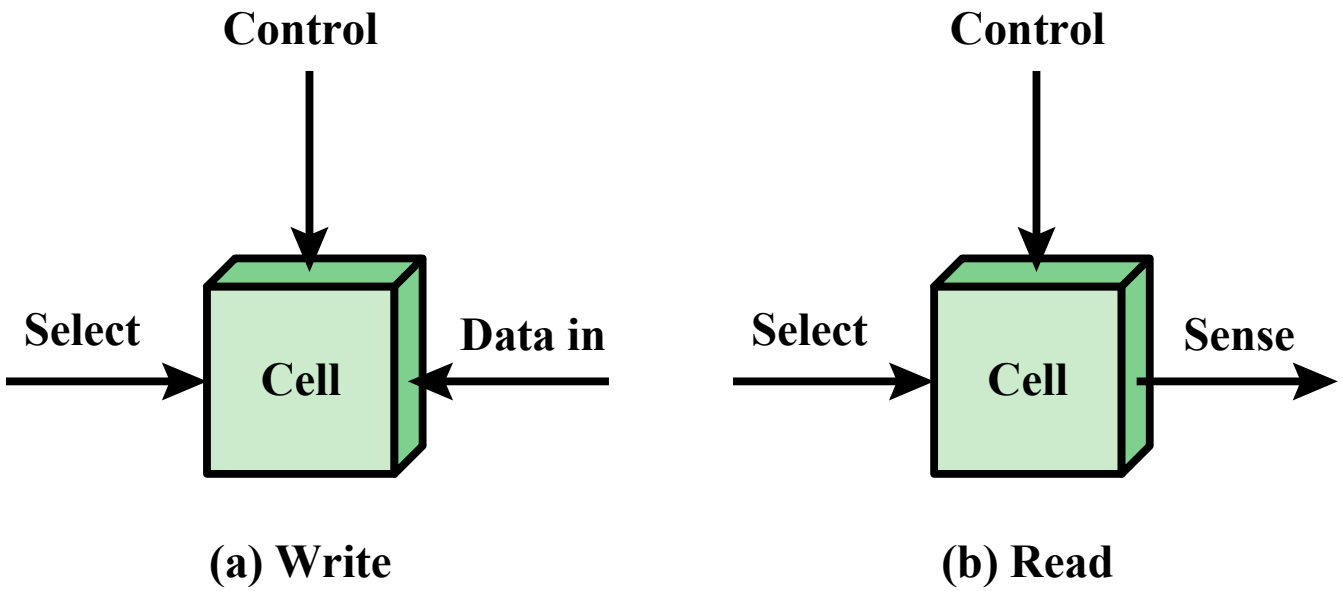
- The performance of the memory system is determined using three parameters:
  - Access time: In case of random access memory, it is the time taken by memory to complete read/ write operation from the instant that an address is sent to the memory. On the other hand, for nonrandom access memory, access time is the time it takes to position the read-write mechanism at the desired location.
  - Memory cycle time: This term is used only in concern with random access memory and it is defined as access time plus additional time required before a second access can commence.
  - Transfer rate: It is defined as the rate at which data can be transferred into or out of a memory unit.

- Physical Type : Two most common physical types used today are semiconductor memory and magnetic surface memory.
- Physical characteristics:
  - I. Volatile/Nonvolatile: If memory can hold data even if power is turned off, it is called as nonvolatile memory; otherwise it is called as volatile memory.
  - II. Erasable/Nonerasable: The memories in which data is once programmed cannot be erased are called as nonerasable memories. On the other hand, if data in the memory is erasable then memory is called as erasable memory

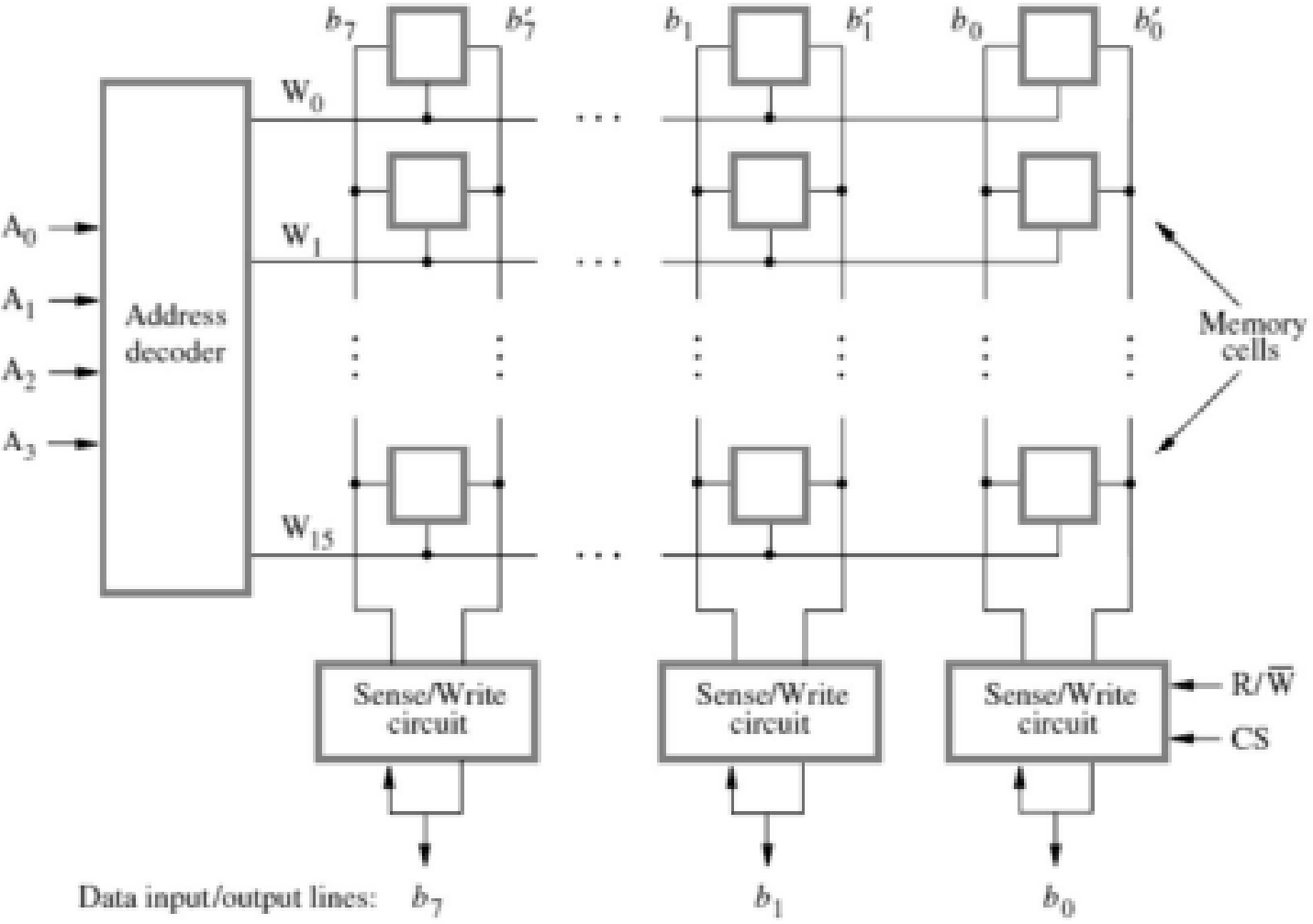
Memory


Memory
Primary
Secondary
Cache
Register
RAM
ROM
Hard Disk, Magnetic Tape, CD,DVD,Zip Disk etc
Static
Dynamic
PROM
EPROM
EEPROM

Memory Cell Operation



nternal Organization of Memory Chips



16 × 8 organization

- Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information
- . Each row of cells constitutes a memory word, and all cells of a row are connected to a common line referred to as the word line, which is driven by the address decoder on the chip.
- The cells in each column are connected to a Sense/Write circuit by two bit lines, and the Sense/Write circuits are connected to the data input/output lines of the chip.
- During a Read operation, these circuits sense, or read, the information stored in the cells selected by a word line and place this information on the output data lines.
- During a Write operation, the Sense/Write circuits receive input data and store them in the cells of the selected word.

Primary Memory

- Primary memory is also known as the computer system's main memory
- Random Access Memory
  - Volatile
  - 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
  - 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

Primary Memory

- Read Only Memory
- 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

Primary Memory

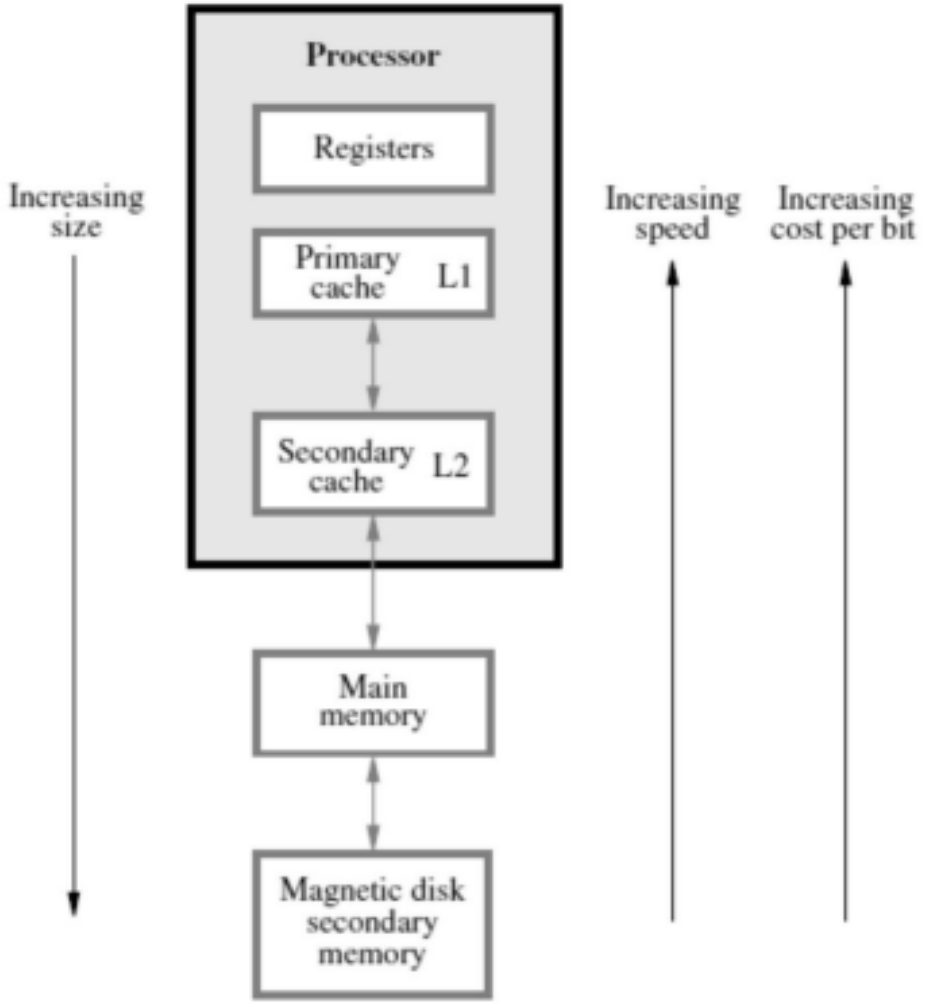
- Random Access Memory
- Read Only Memory

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)				
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level	Electrically	
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Secondary Memory

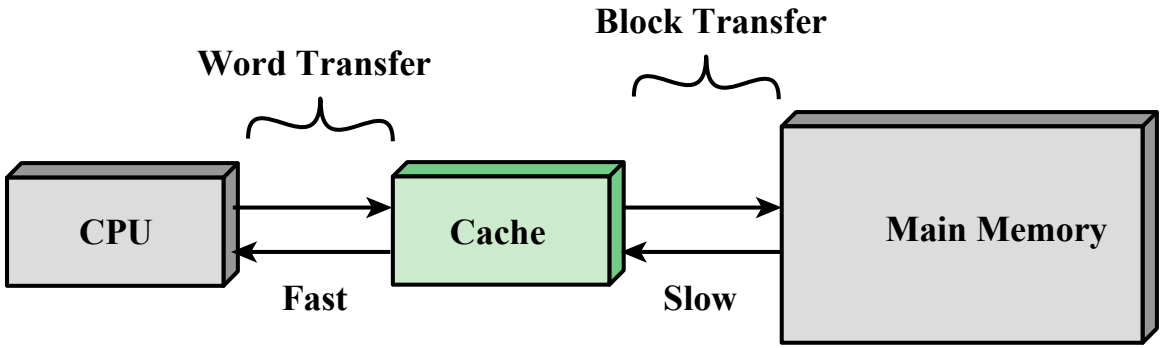
- is a **permanent storage** space to hold a large amount of data.
- It is also known as external memory that representing the various storage media (hard drives, USB, CDs, flash drives and DVDs) on which the computer data and program can be saved on a long term basis.
- However, it is cheaper and slower than the main memory. Unlike primary memory, secondary memory cannot be accessed directly by the CPU. Instead of that, secondary memory data is first loaded into the RAM (Random Access Memory) and then sent to the processor to read and update the data.
- Secondary memory devices also include magnetic disks like hard disk and floppy disks, an optical disk such as CDs and CDROMs, and magnetic tapes.

Memory Hierarchy

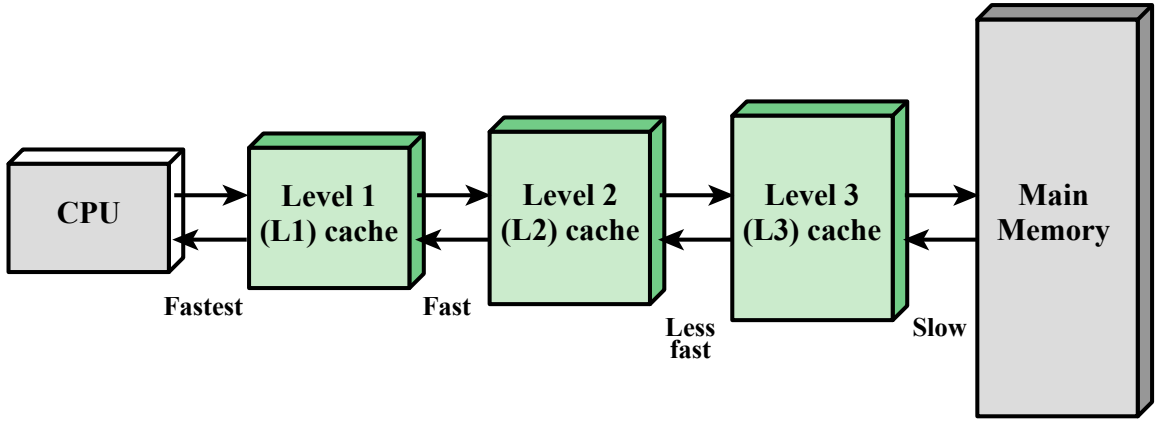


Cache Memory

- It is a small-sized chip-based computer memory that lies between the CPU and the main memory.
- It is a faster, high performance and temporary memory to enhance the performance of the CPU.
- It stores all the data and instructions that are often used by computer CPUs.
- Locality of Reference
  - Temporal Locality
    - This type of optimization includes bringing in the frequently accessed memory references to a nearby memory location for a short duration of time so that the future accesses are much faster.
  - Spatial Locality
    - if a memory location has been accessed it is highly likely that a nearby/consecutive memory location will be accessed as well and hence we bring in the nearby memory references too in a nearby memory location for faster access.



(a) Single cache



(b) Three-level cache organization

Cache

- Cache Hit
- Cache Miss

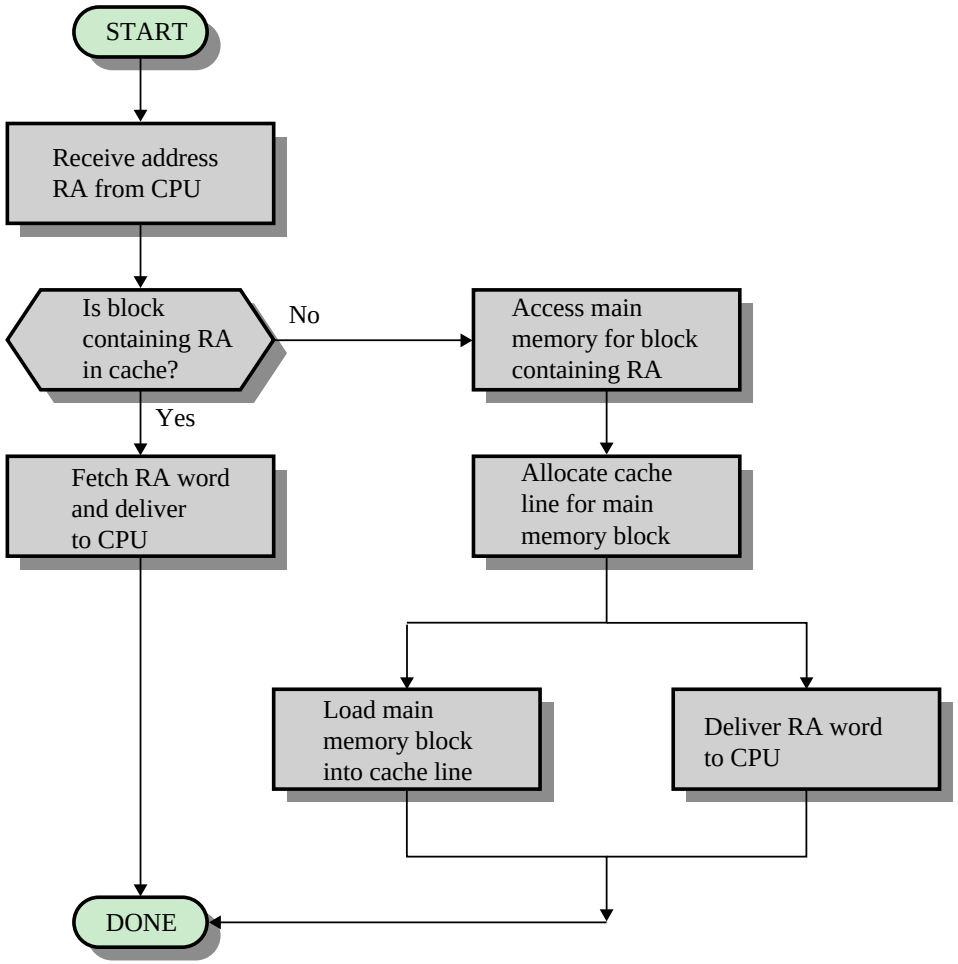
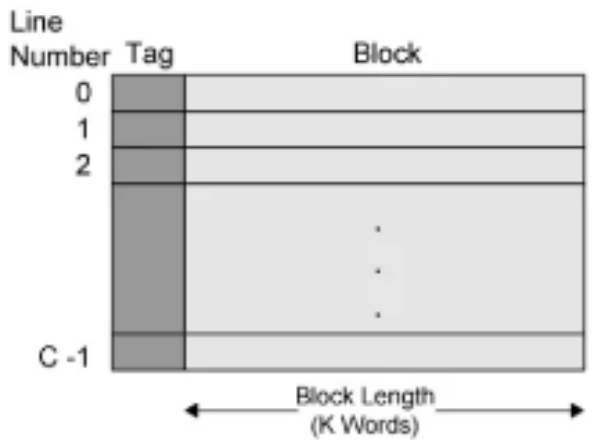


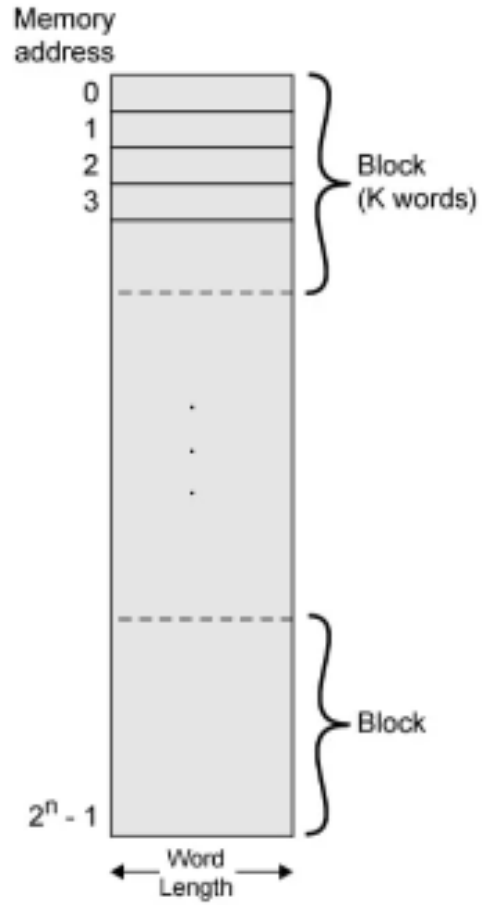
Figure 4.5 Cache Read Operation

Cache/Main Memory Structure

- Main memory is divided into equal size partitions called as **blocks** or **frames**.
- Cache memory is divided into partitions having same size as that of blocks called as **lines**.
- During cache mapping, block of main memory is simply copied to the cache and the block is not actually brought from the main memory.



(a) Cache



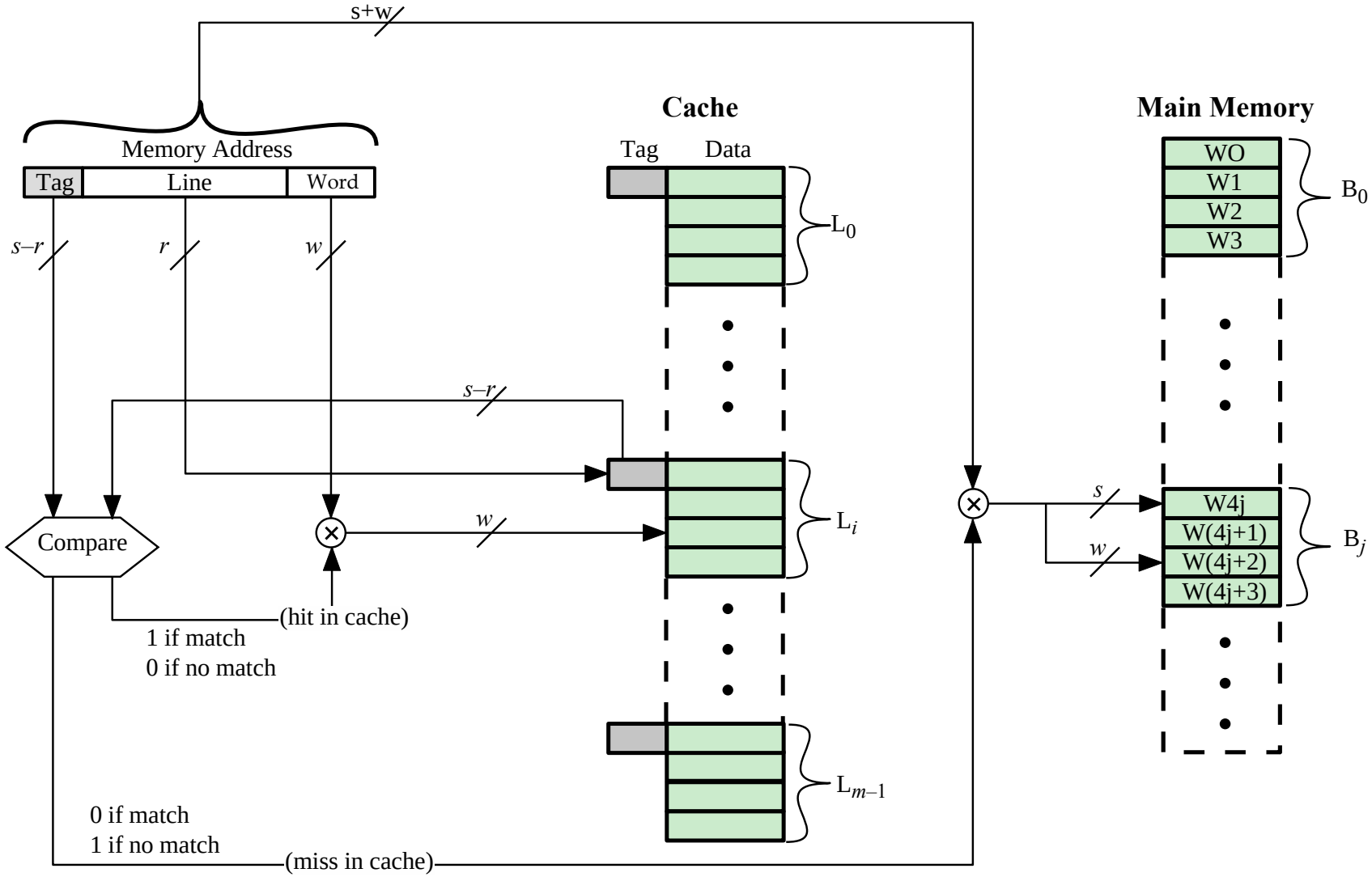
(b) Main memory

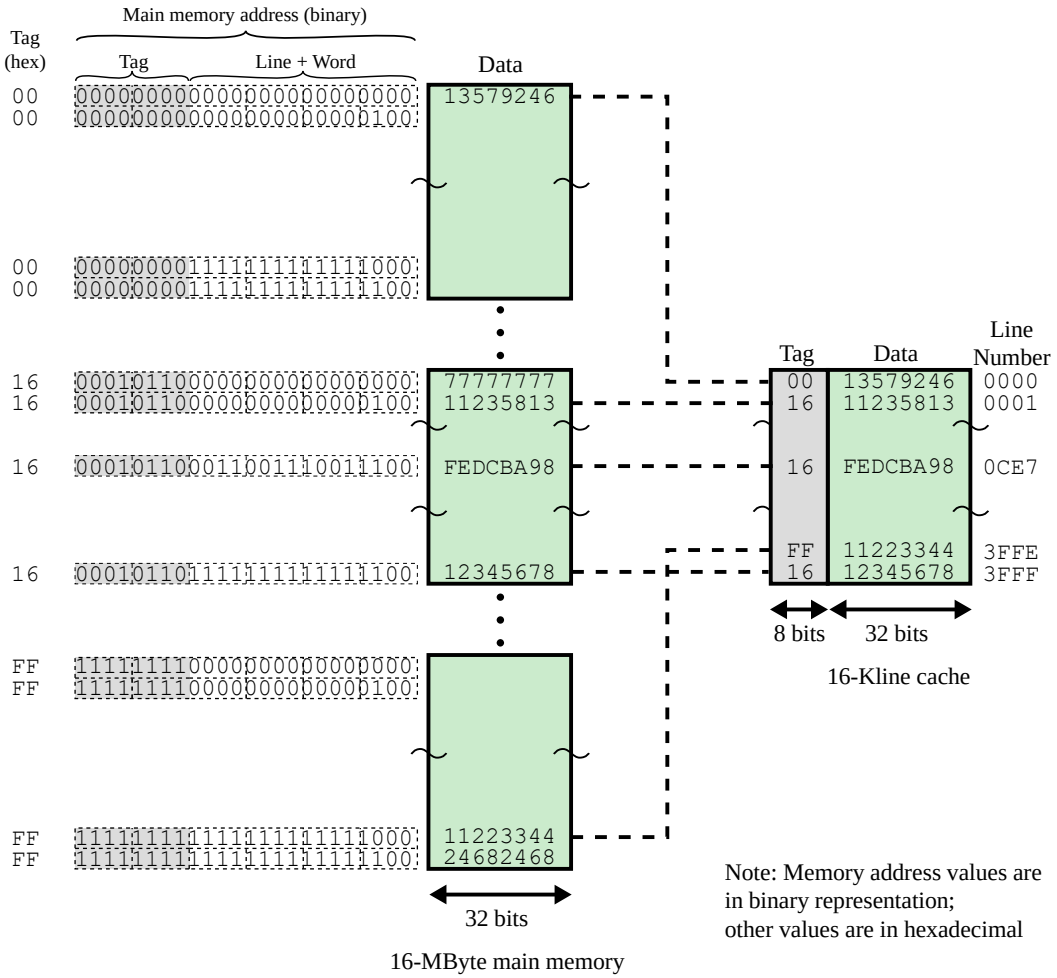
Mapping Function

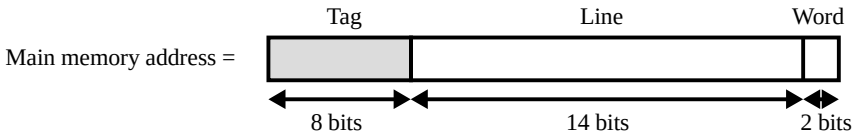
- Cache mapping defines how a block from the main memory is mapped to the cache memory in case of a cache miss.
- Cache includes tags to identify which block of main memory is in each cache slot
- Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

Cache Mapping Techniques

- Direct Mapping
- Fully Associative Mapping
- K-way Set Associative Mapping


s+w
Memory Address
Tag
Line
Word
s–r
r
w
Cache
Tag
Data
L0
Li
Lm–1
Main Memory
WO
W1
W2
W3
B0
W4j
W(4j+1)
W(4j+2)
W(4j+3)
Bj
s
w
Compare
(hit in cache)
1 if match
0 if no match
0 if match
1 if no match
(miss in cache)


Main memory address (binary)
Tag (hex)
Tag
Line + Word
Data
00 00000000000000000000000000 13579246
00 00000000000000000000000100
00 00000000111111111111111000
00 00000000111111111111111100
16 00010110000000000000000000 77777777
16 00010110000000000000000100 11235813
16 00010110001100111001111100 FEDCBA98
16 00010110111111111111111100 12345678
FF 11111111000000000000000000
FF 11111111000000000000000100
FF 11111111111111111111111000 11223344
FF 11111111111111111111111100 24682468
32 bits
16-MByte main memory
Tag
Data
Line Number
00 13579246 0000
16 11235813 0001
16 FEDCBA98 0CE7
FF 11223344 3FFE
16 12345678 3FFF
8 bits
32 bits
16-Kline cache
Note: Memory address values are in binary representation; other values are in hexadecimal

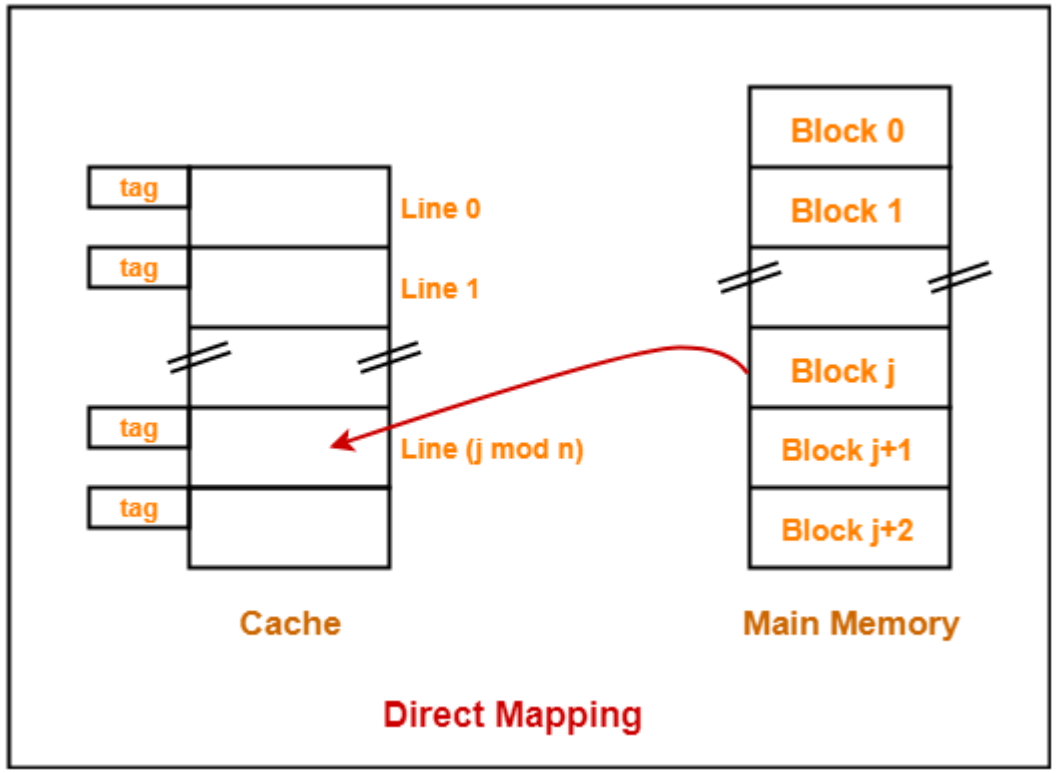

Main memory address =
Tag
Line
Word
8 bits
14 bits
2 bits

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

Direct Mapping

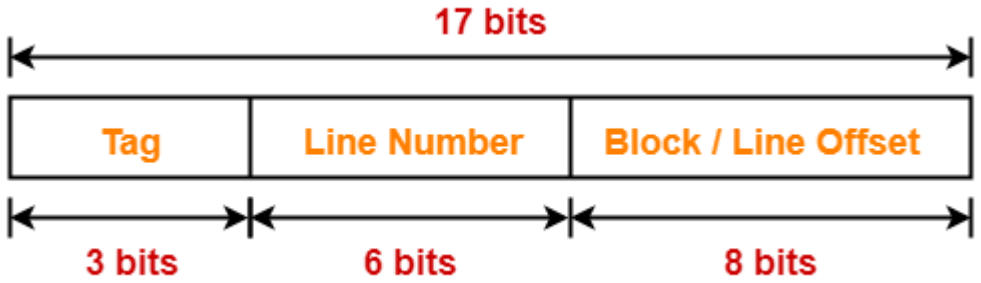
- A particular block of main memory can map only to a particular line of the cache.
- The line number of cache to which a particular block can map is given by-
- **Cache line number= (Main Memory Block Address) Modulo (Number of lines in Cache)**



Direct Mapping

- Cache memory size = 16 KB
- Block size = Frame size = Line size = 256 bytes
- Main memory size = 128 KB

Division of Physical Address in Direct Mapping

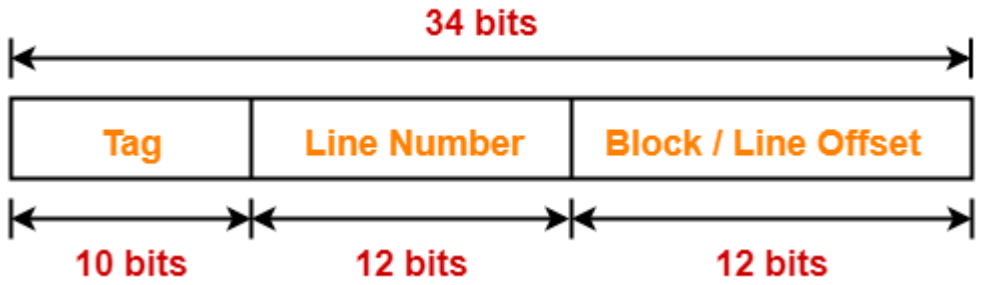


Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 2^6 x 3 bits
= 192 bits
= 24 bytes
Thus, size of tag directory = 24 bytes

- Block size = Frame size = Line size = 4 KB
- Size of main memory = 16 GB
- Number of bits in tag = 10 bits
- Find:
  - Size of cache memory
  - Tag directory size

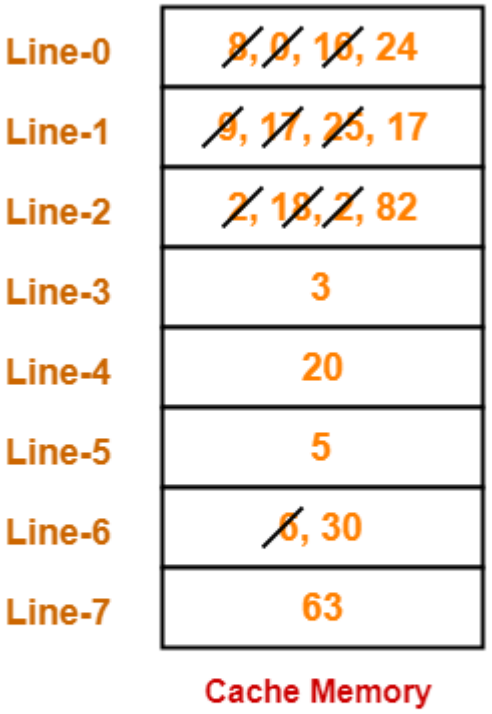
- **Number of Lines in Cache-**
- Number of bits in line number = 12 bits
- Total number of lines in cache = 2^{12} lines



- **Size of Cache Memory-**

- Size of cache memory
- = Total number of lines in cache x Line size
- = 2^{12} x 4 KB
- = 2^{14} KB
- = 16 MB
- Size of cache memory = 16 MB

- Consider a direct mapped cache with 8 cache blocks (0-7). If the memory block requests are in the order-3, 5, 2, 8, 0, 6, 3, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24. calculate the hit ratio and miss ratio.

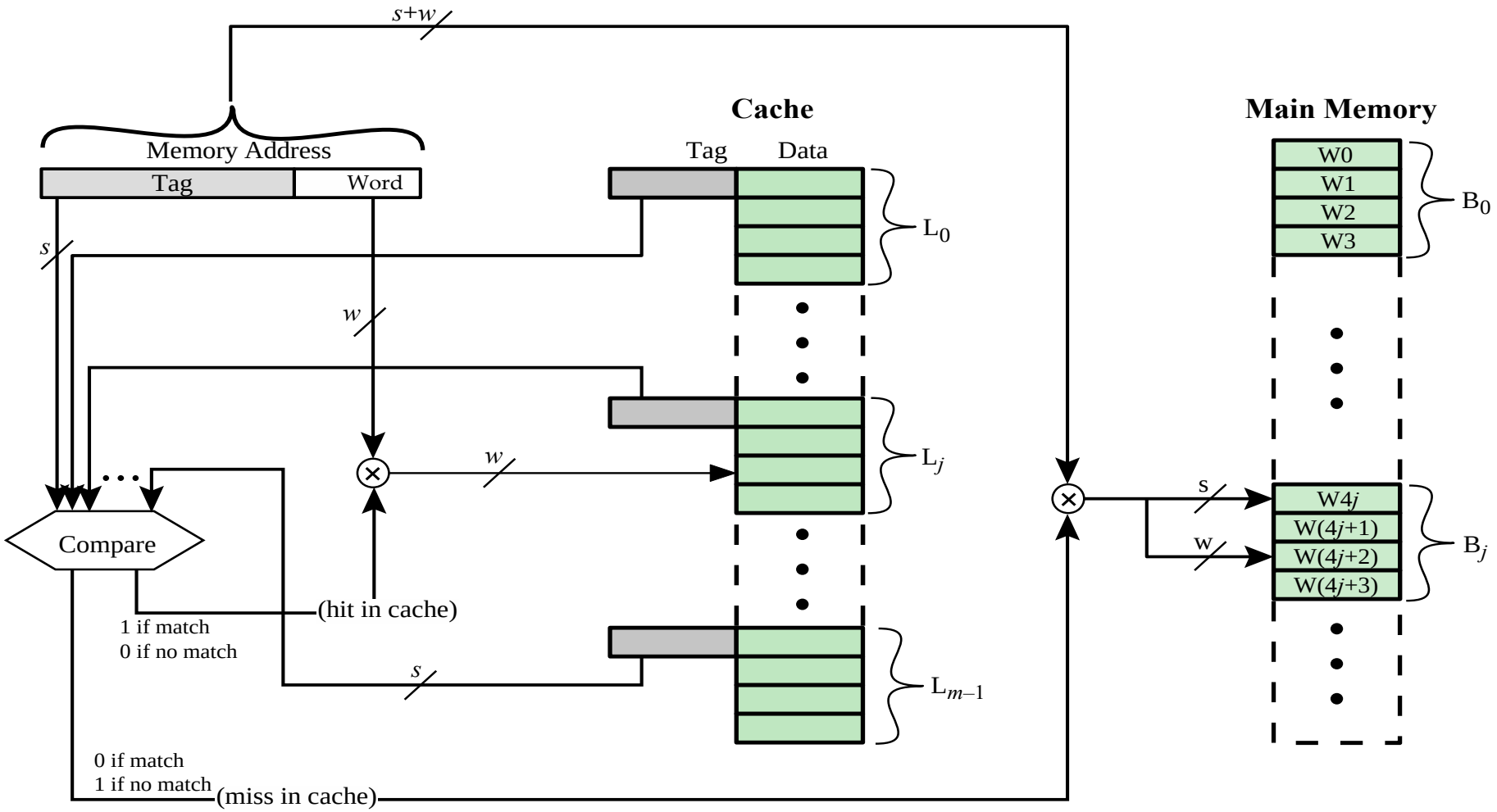


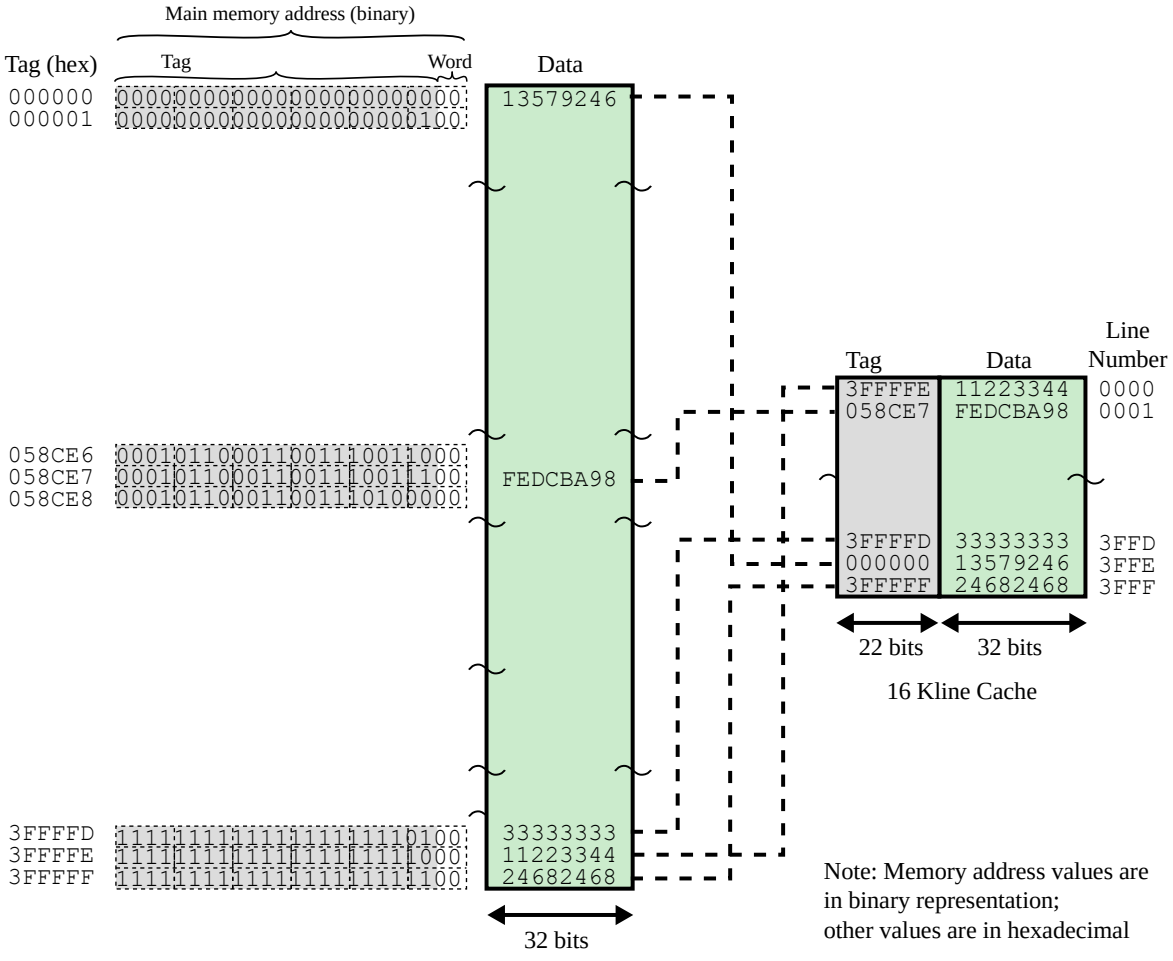
3 % 8 = 3 (Miss)
5 % 8 = 5 (Miss)
2 % 8 = 2 (Miss)
8 % 8 = 0 (Miss)
0 % 8 = 0 (Miss)
6 % 8 = 6 (Miss)
3 % 8 = 3 (Hit)
9 % 8 = 1 (Miss)
16 % 8 = 0 (Miss)
20 % 8 = 4 (Miss)
17 % 8 = 1 (Miss)
25 % 8 = 1 (Miss)
18 % 8 = 2 (Miss)
30 % 8 = 6 (Miss)
24 % 8 = 0 (Miss)
2 % 8 = 2 (Miss)
63 % 8 = 7 (Miss)
5 % 8 = 5 (Hit)
82 % 8 = 2 (Miss)
17 % 8 = 1 (Miss)
24 % 8 = 0 (Hit)

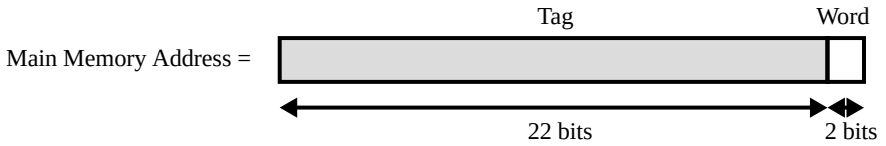
- Hit ratio = 3 / 20
- Miss ratio = 17 / 20

Associative mapping

- Associative mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of the cache.
- In this case, the cache control logic interprets a memory address simply as a Tag and a Word field. The Tag field uniquely identifies a block of main memory.


s+w
Memory Address
Tag
Word
s
w
Cache
Tag
Data
L0
Lj
Lm–1
Main Memory
W0
W1
W2
W3
B0
W4j
W(4j+1)
W(4j+2)
W(4j+3)
Bj
Compare
(hit in cache)
1 if match
0 if no match
0 if match
1 if no match
(miss in cache)

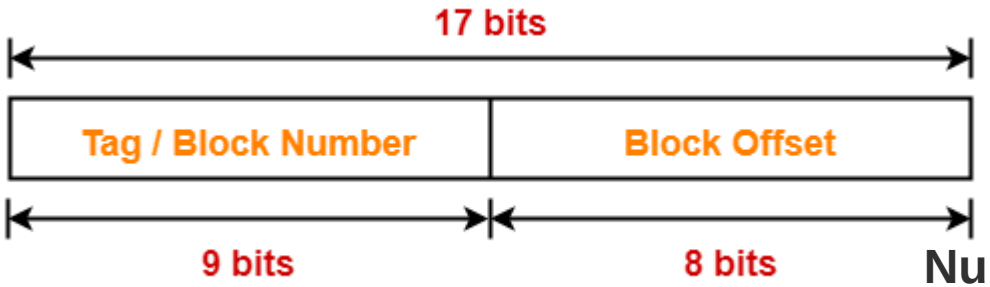

Main memory address (binary)
Tag (hex)
Tag
Word
Data
000000 00000000000000000000000000
000001 00000000000000000000000100
13579246
058CE6 00010110001100111001100000
058CE7 00010110001100111001100100
058CE8 00010110001100111010000000
FEDCBA98
3FFFFD 11111111111111111111110100
3FFFFE 11111111111111111111111000
3FFFFF 11111111111111111111111100
33333333
11223344
24682468
32 bits
16 MByte Main Memory
Line Number
Tag
Data
3FFFFE 11223344 0000
058CE7 FEDCBA98 0001
3FFFFD 33333333 3FFD
000000 13579246 3FFE
3FFFFF 24682468 3FFF
22 bits
32 bits
16 Kline Cache
Note: Memory address values are in binary representation; other values are in hexadecimal


Main Memory Address =
Tag
Word
22 bits
2 bits

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

- Cache memory size = 16 KB
- Block size = Frame size = Line size = 256 bytes
- Main memory size = 128 KB
- Number of bits in tag=?
- Tag directory size=?



Number of Lines in Cache-

Total number of lines in cache
= Cache size / Line size
= 16 KB / 256 bytes
= 2^{14} bytes / 2^{8} bytes
= 2^{6} lines

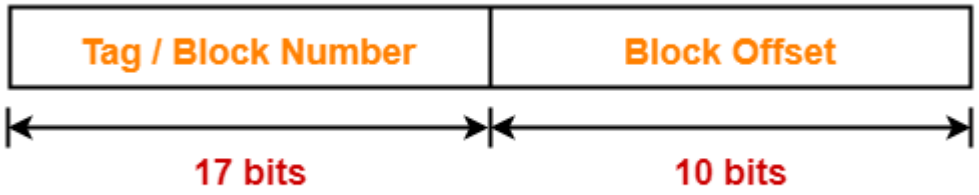
Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 2^{6} x 9 bits
= 576 bits
= 72 bytes
Thus, size of tag directory = 72 bytes

- Cache memory size = 512 KB
- Block size = Frame size = Line size = 1 KB
- Number of bits in tag = 17 bits

Size of main memory =?

Tag directory size=?



Size of Main Memory-

We have,
Number of bits in physical address = 27 bits
Thus, Size of main memory
= 2^{27} bytes
= 128 MB

Number of Lines in Cache-

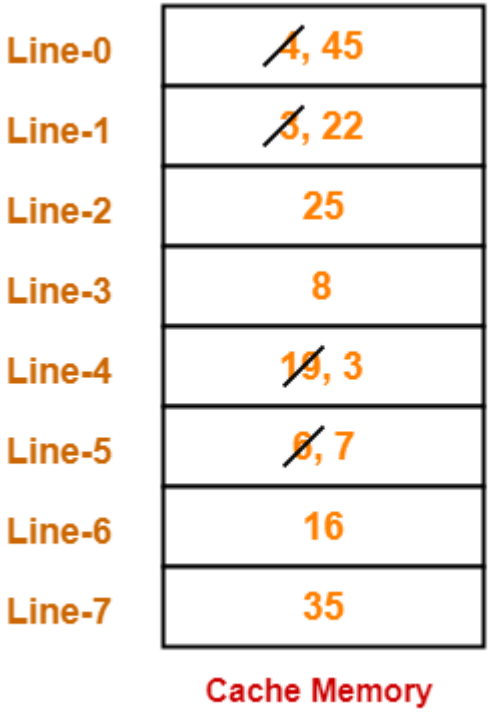
Total number of lines in cache
= Cache size / Line size
= 512 KB / 1 KB
= 512 lines
= 2^9 lines

Tag Directory Size-

Tag directory size
= Number of tags x Tag size
= Number of lines in cache x Number of bits in tag
= 2^9 x 17 bits
= 8704 bits
= 1088 bytes

- Consider a fully associative cache with 8 cache blocks (0-7). The memory block requests are in the order- 4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7
- If LRU replacement policy is used, which cache block will have memory block 7?Also, calculate the hit ratio and miss ratio.

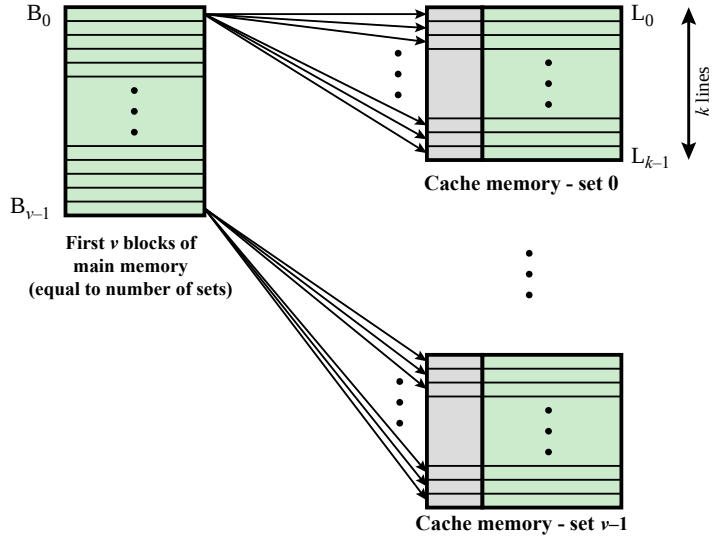
- There are 8 blocks in cache memory numbered from 0 to 7.
- In fully associative mapping, any block of main memory can be mapped to any line of the cache that is freely available.
- If all the cache lines are already occupied, then a block is replaced in accordance with the replacement policy



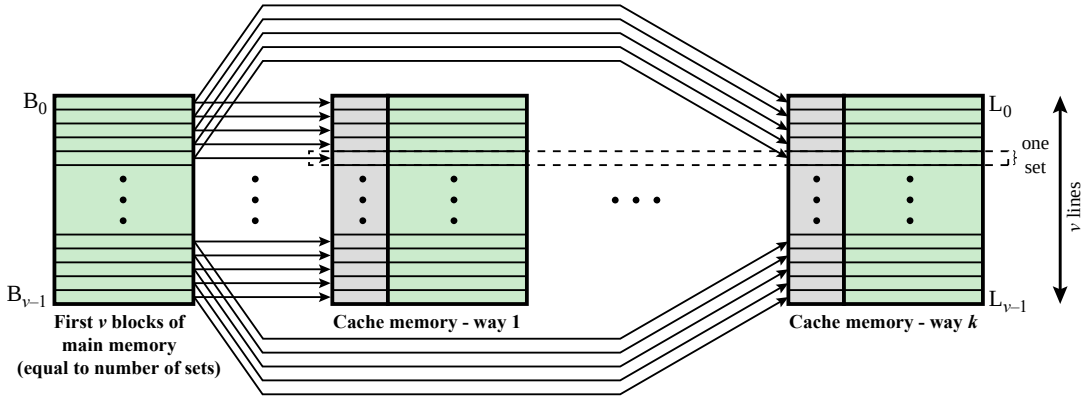
- Line-5 contains the block-7.
- Hit ratio = 5 / 17
- Miss ratio = 12 / 17

Set Associative Mapping

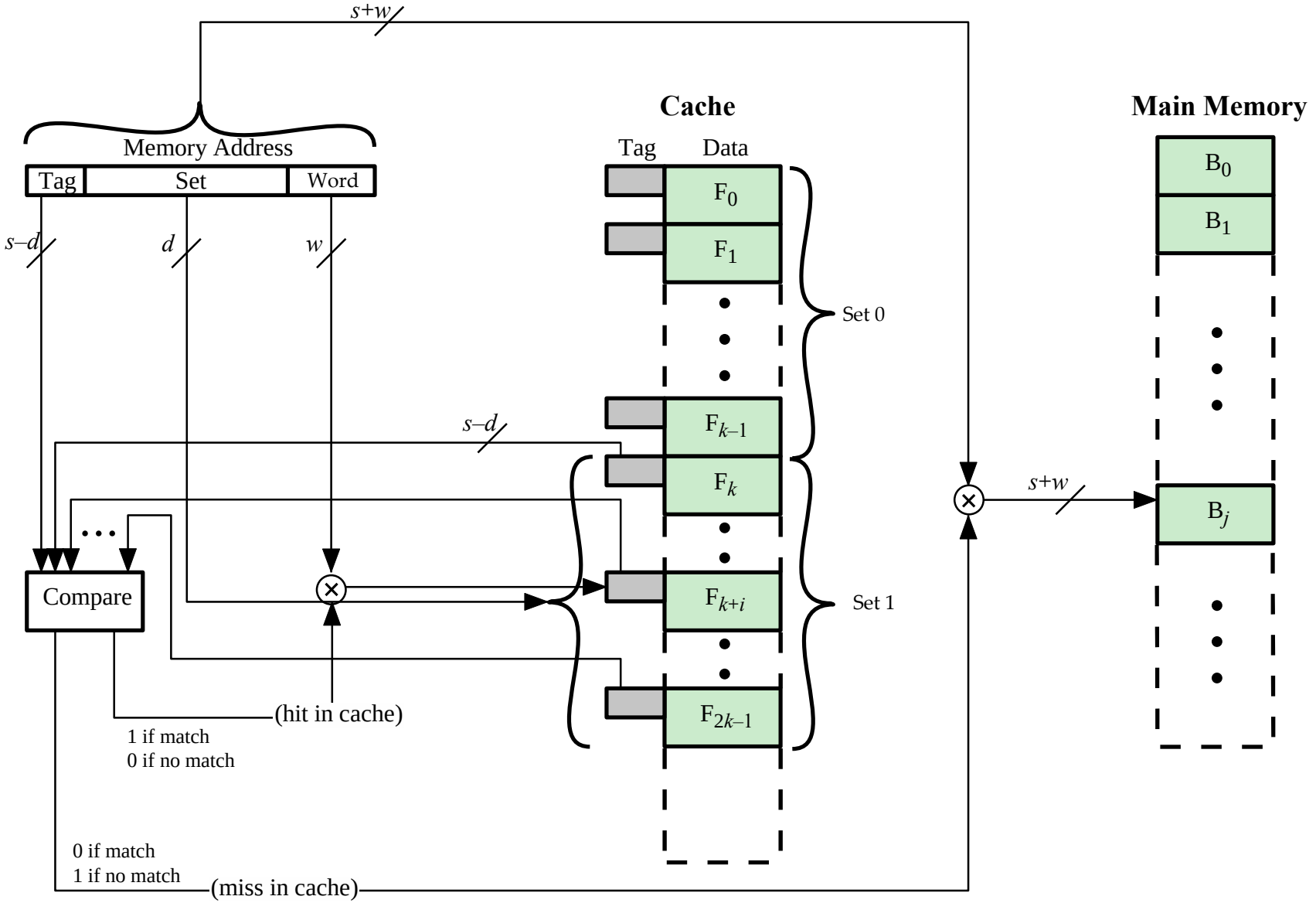
- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2 way associative mapping
  -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



(a) *v* associative-mapped caches

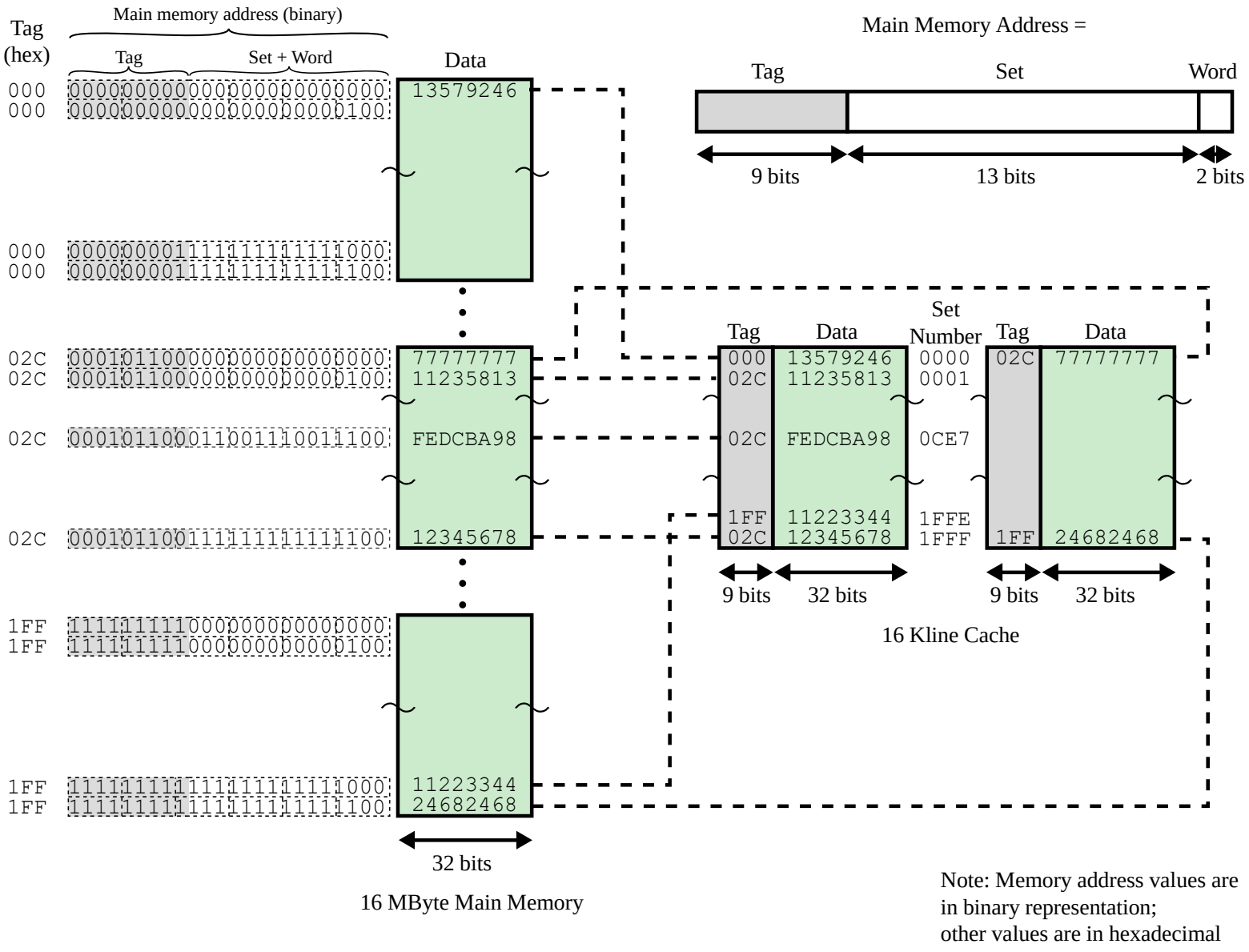


(b) *k* direct-mapped caches


s+w
Memory Address
Tag
Set
Word
s–d
d
w
Cache
Tag
Data
F0
F1
Set 0
Fk–1
Fk
s–d
Fk+i
Set 1
F2k–1
Compare
(hit in cache)
1 if match
0 if no match
0 if match
1 if no match
(miss in cache)
s+w
Main Memory
B0
B1
Bj

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

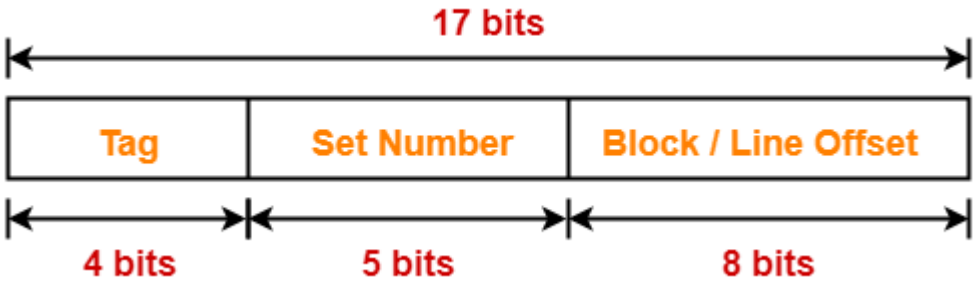

Tag (hex)
Main memory address (binary)
Tag
Set + Word
Data
000 00000000000000000000000000
000 00000000000000000000000100
13579246
000 00000000011111111111111000
000 00000000011111111111111100
02C 00010110000000000000000000
02C 00010110000000000000000100
77777777
11235813
02C 00010110000110011100011100
FEDCBA98
02C 00010110001111111111111100
12345678
1FF 11111111110000000000000000
1FF 11111111110000000000000100
1FF 11111111111111111111111000
1FF 11111111111111111111111100
11223344
24682468
32 bits
16 MByte Main Memory
Main Memory Address =
Tag
Set
Word
9 bits
13 bits
2 bits
Set Number
Tag
Data
Tag
Data
000 13579246
02C 11235813
02C FEDCBA98
1FF 11223344
02C 12345678
0000
0001
0CE7
1FFE
1FFF
02C 77777777
1FF 24682468
9 bits
32 bits
9 bits
32 bits
16 Kline Cache
Note: Memory address values are in binary representation;
other values are in hexadecimal

Set Associative Mapping Summary

- **Cache set number**
- **= (Main Memory Block Address) Modulo (Number of sets in Cache)**

Division of Physical Address in K-way Set Associative Mapping

- Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find-
- Number of bits in tag
- Tag directory size



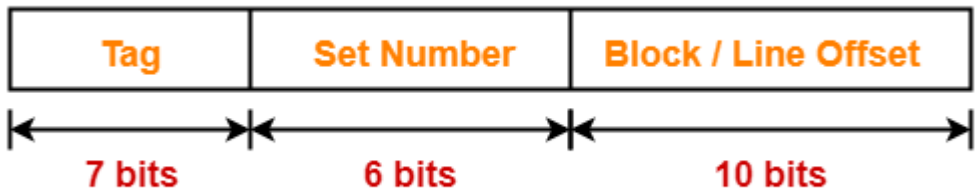
Number of Lines in Cache-

Total number of lines in cache
= Cache size / Line size
= 16 KB / 256 bytes
= 2^{14} bytes / 2^{8} bytes
= 64 lines
Thus, Number of lines in cache = 64 lines

Number of Sets in Cache-

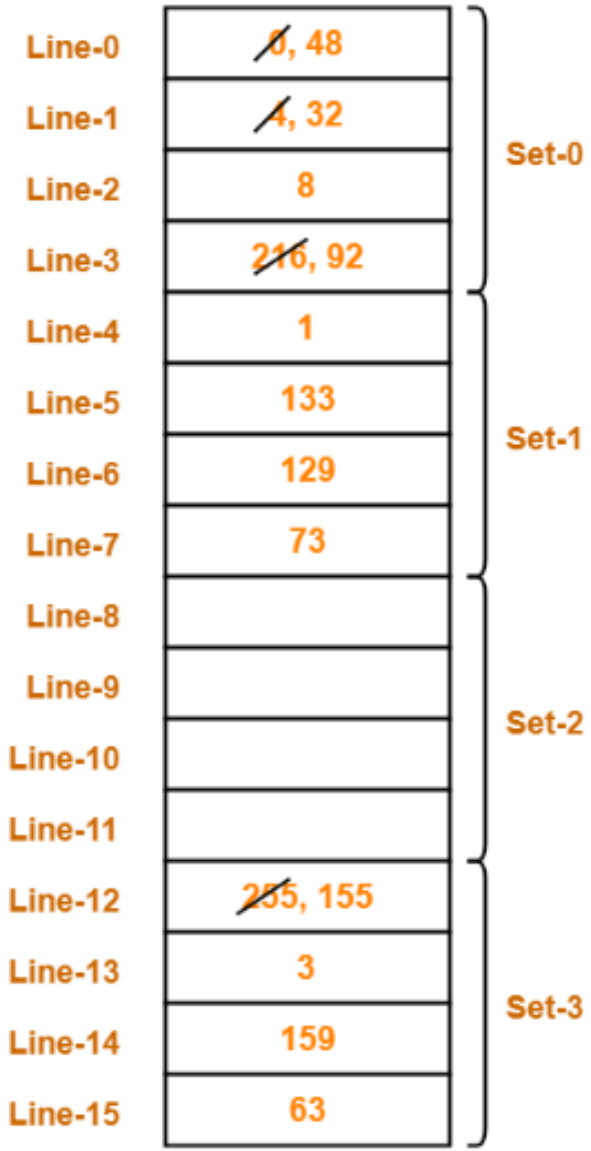
Total number of sets in cache
= Total number of lines in cache / Set size
= 64 / 2
= 32 sets
= 2^{5} sets
Thus, Number of bits in set number = 5 bits

- Consider a 8-way set associative mapped cache of size 512 KB with block size 1 KB. There are 7 bits in the tag. Find-
- Size of main memory
- Tag directory size


Tag
Set Number
Block / Line Offset
7 bits
6 bits
10 bits

Consider a 4-way set associative mapping with 16 cache blocks. The memory block requests are in the order-0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155

- If LRU replacement policy is used, which cache block will not be present in the cache?
- 3
- 8
- 129
- 216



Blocks requests in order
Calculation of set number

0 % 4 = 0	(Miss)
255 % 4 = 3	(Miss)
1 % 4 = 1	(Miss)
4 % 4 = 0	(Miss)
3 % 4 = 3	(Miss)
8 % 4 = 0	(Miss)
133 % 4 = 1	(Miss)
159 % 4 = 3	(Miss)
216 % 4 = 0	(Miss)
129 % 4 = 1	(Miss)
63 % 4 = 3	(Miss)
8 % 4 = 0	(Hit)
48 % 4 = 0	(Miss)
32 % 4 = 0	(Miss)
73 % 4 = 1	(Miss)
92 % 4 = 0	(Miss)
155 % 4 = 3	(Miss)

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 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
- Least frequently used (LFU)
  - 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

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

Memory Organization

- Simultaneous Access Memory Organization

  Average Memory Access Time

- Average time required to access memory per operation

  = H1 x T1 + (1 – H1) x H2 x T2

- Hierarchical Access Memory Organization

  Average time required to access memory per operation

  = H1 x T1 + (1 – H1) x H2 x (T1 + T2)