Cache Memory

COAL FALL 2020

Asma Ahmad

Characteristics of Memory Systems

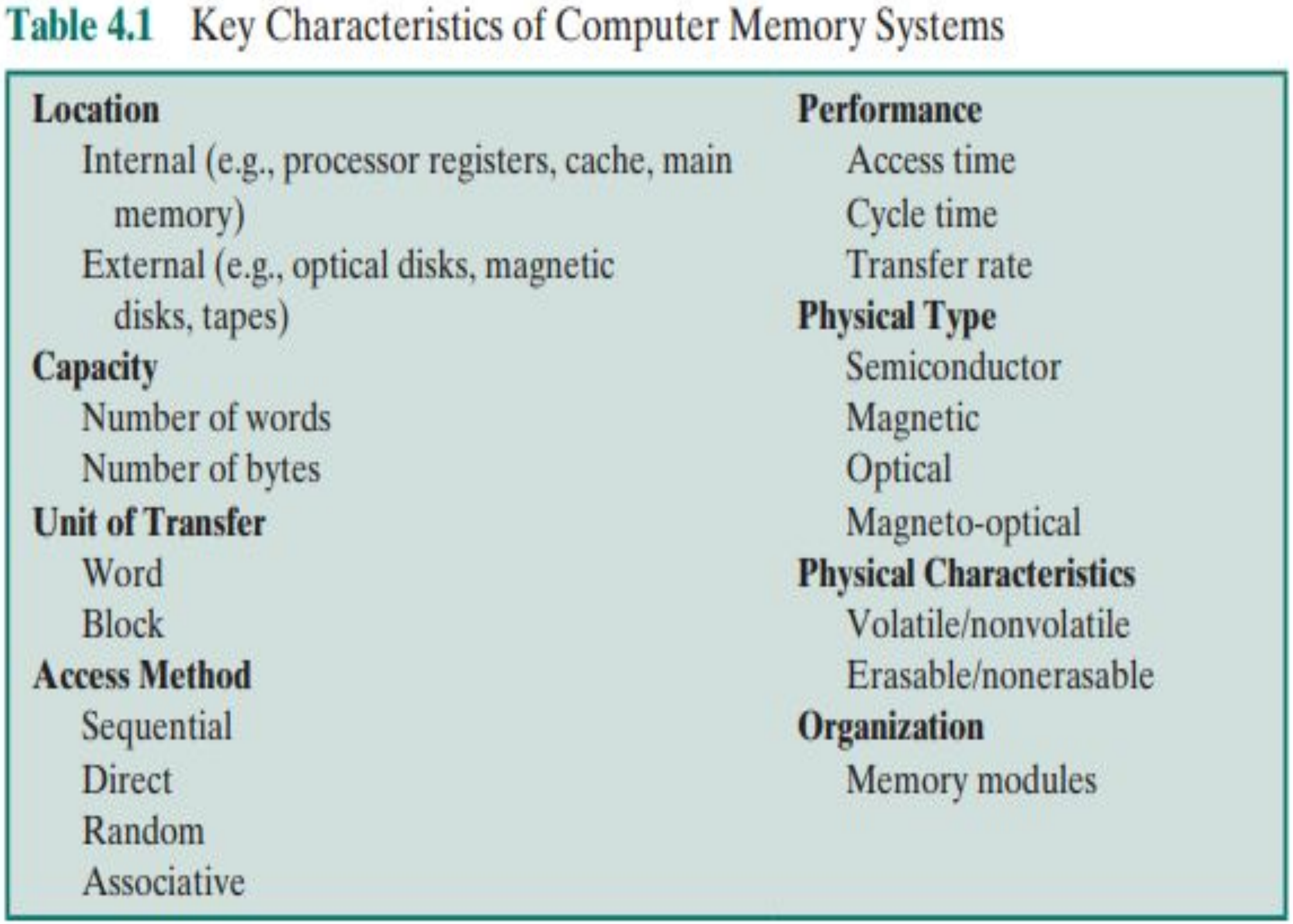


Table 4.1 Key Characteristics of Computer Memory Systems

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

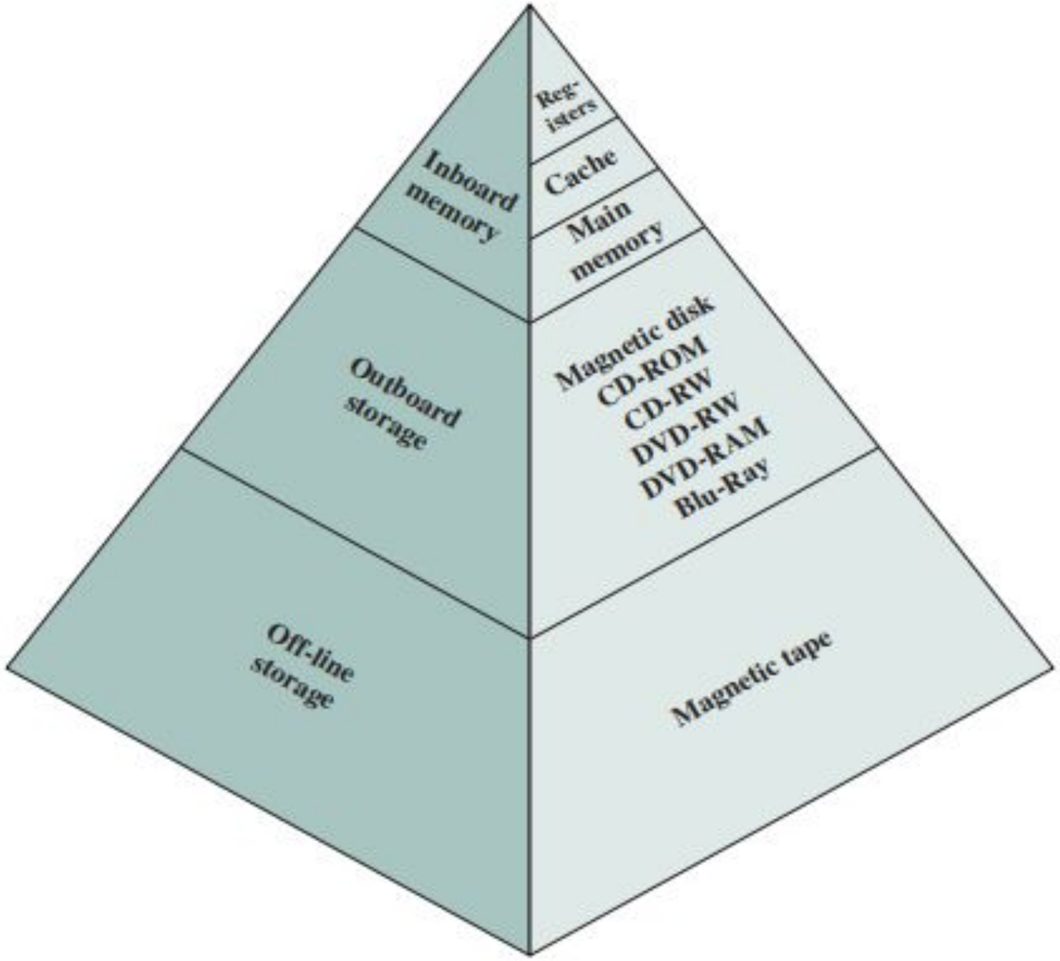
Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

Memory Hierarchy



Example of Sequential Access

Example of Direct Access

DIRECT ACCESS STORAGE DEVICES

CACHE MEMORY PRINCIPLES

- Cache memory is designed to combine the memory access time of expensive, high-speed memory combined with the large memory size of less expensive, lower-speed memory.
- The cache contains a copy of portions of main memory.
- When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache.
- If so, the word is delivered to the processor.
- If not, a block of main memory, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor.
- Because of the phenomenon of locality of reference, when a block of data is fetched into the cache to satisfy a single memory reference, it is likely that there will be future references to that same memory location or to other words in the block

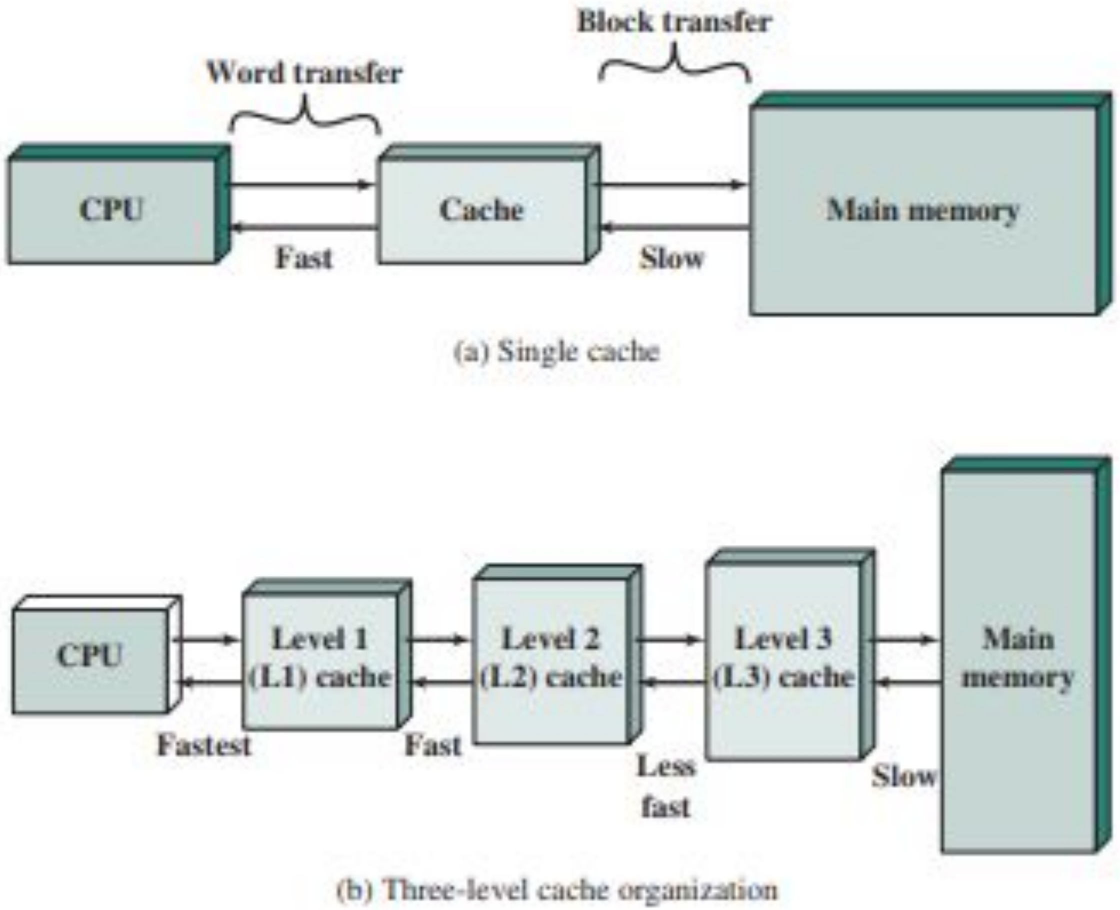


Figure 4.3 Cache and Main Memory

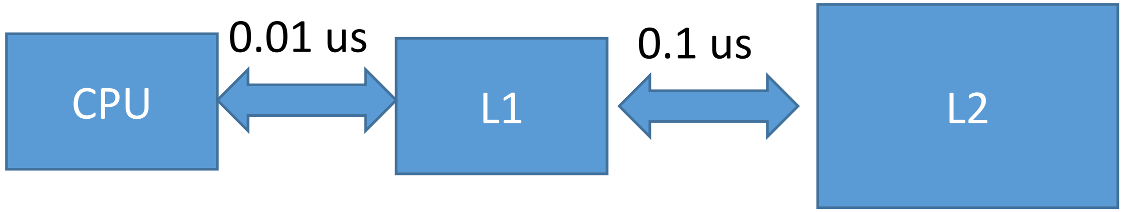
CACHE MEMORY PRINCIPLES

- Cache hit
    - Copy of data is in cache, quick access

- Cache miss
    - Copy of data is not in cache, read data on that address and possibly its neighbors into cache

Average Access Time

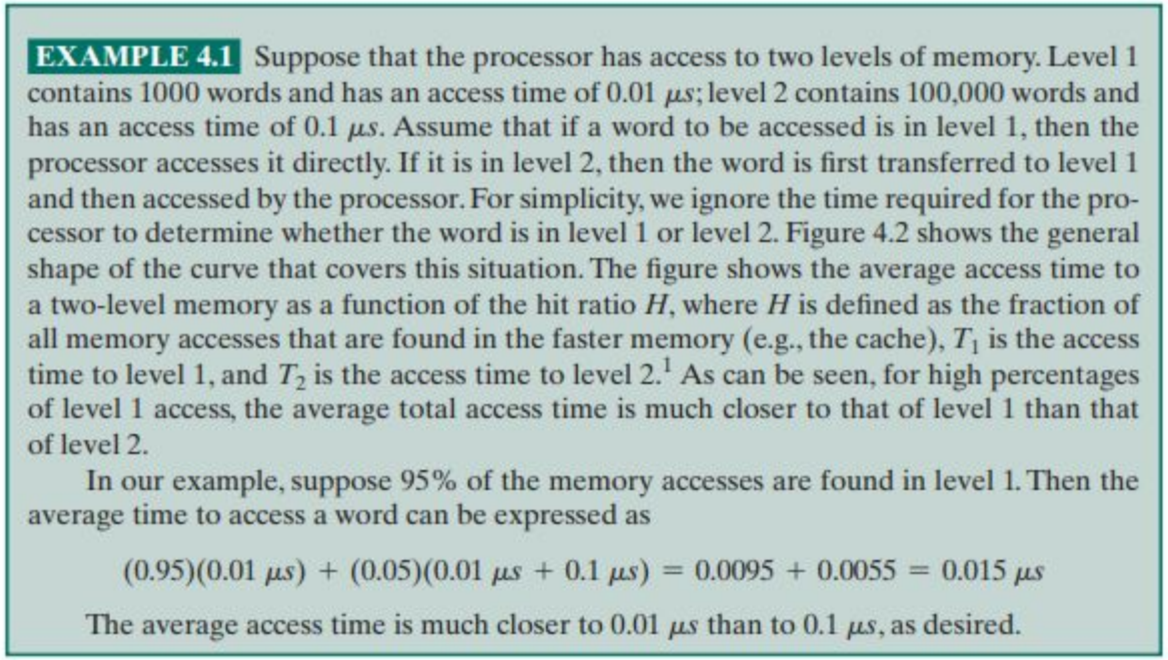
- Suppose we have two level memory L1 and L2
- L1 is on 1000 word capacity and L2 is on 10000 word capacity
- If required data is in L1 the processor will read it in 0.01 us
- If data was not in L1, then its in L2. To get the data to processor it has to be moved form L2 to L1 which will take 0.1 us
- How much total time is require to move data from L2 to processor?
  - **0.11 us**
- Hit rate of L1 is 95%, that is, 95% of the time data is found in L1. Rest of the 5% of time data is not in L1 and has to be read from L2.
- What is the average access time given the hit rate above?
  - **(0.95 * 0.01) + (0.5 * 0.11)= 0.015 us**



Average access time

- What will be the average access rate is hit rate of L1 drops to 50%?
  - (0.5 *0.01) + (0.5 * 0.11) = 0.06
- Which is much higher when hit rate was 95%
- What will be the average access rate is hit rate of L1 is 100%?
  - (1 *0.01) + (0 * 0.11) = 0.01 (perfect but not possible)
- What will be the average access rate is hit rate of L1 is 0%?
  - (0 *0.01) + (1 * 0.11) = 0.11 (worst case, no use of L1)

Average Access Time



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

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

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

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

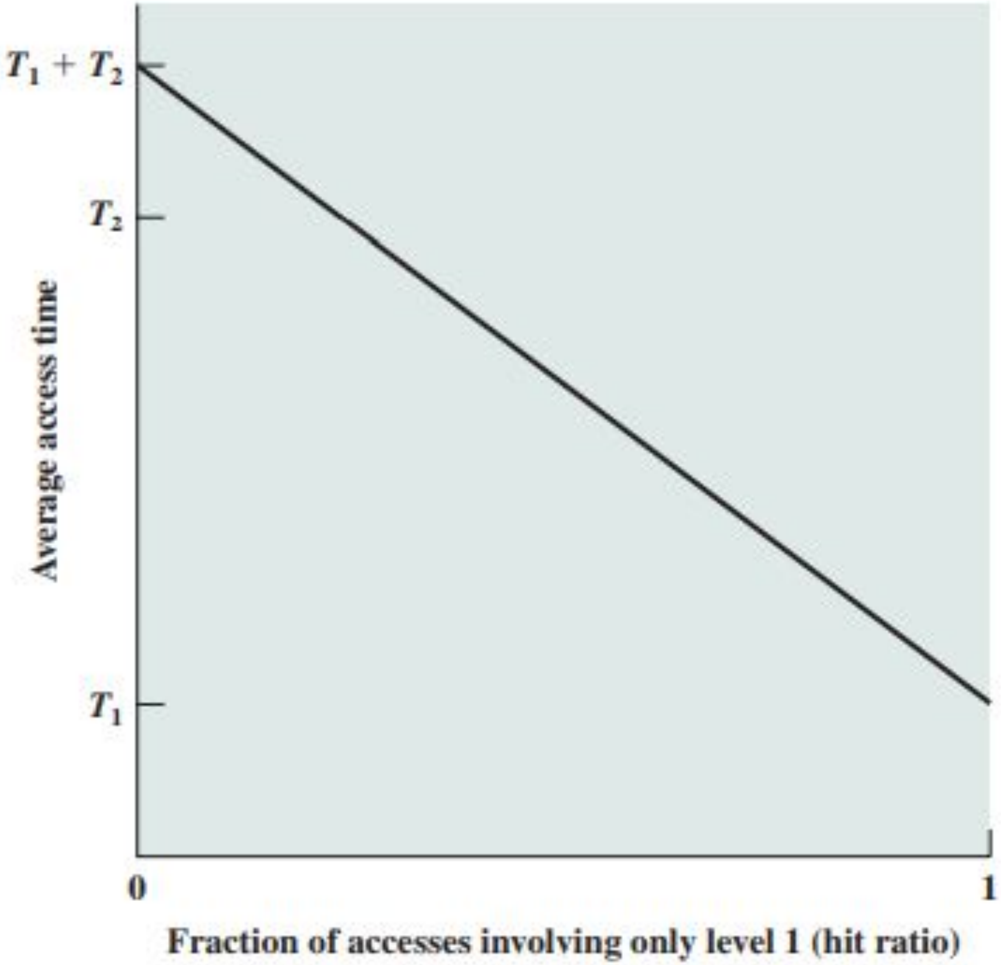
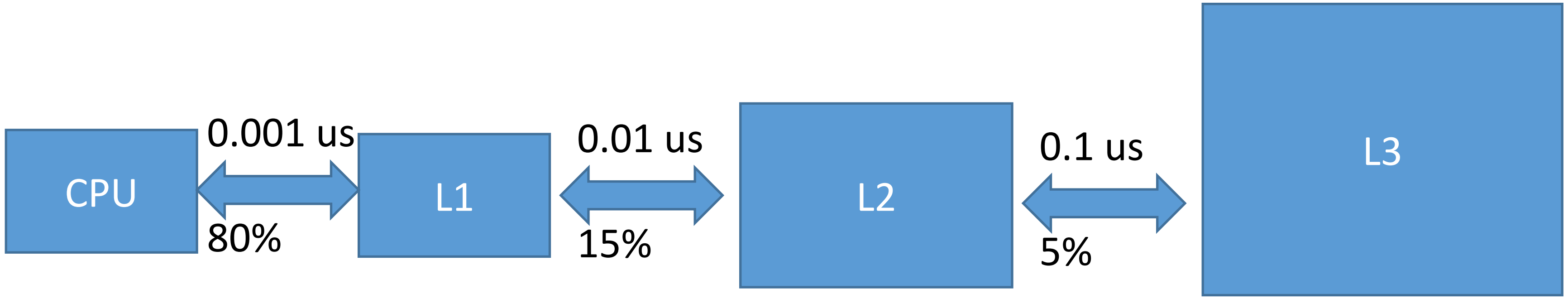


Figure 4.2 Performance of Accesses Involving only Level 1 (hit ratio)

Average Access Time

- Question:
- Consider a 3 level memory. It takes 0.001us to read from level 1, 0.01us to read from level 2 and 0.1us from level 3.
- Data is found in L1 80% of time, out the remaining 20% of time 15% it is found in L2 and 5% data is to be read from L3.
- What is the average access time?
  - (0.8 * 0.001) + (0.15 * [0.001+0.01]) + (0.05 * [0.001+0.01+0.1]) = 0.008



Elements of Cache Design

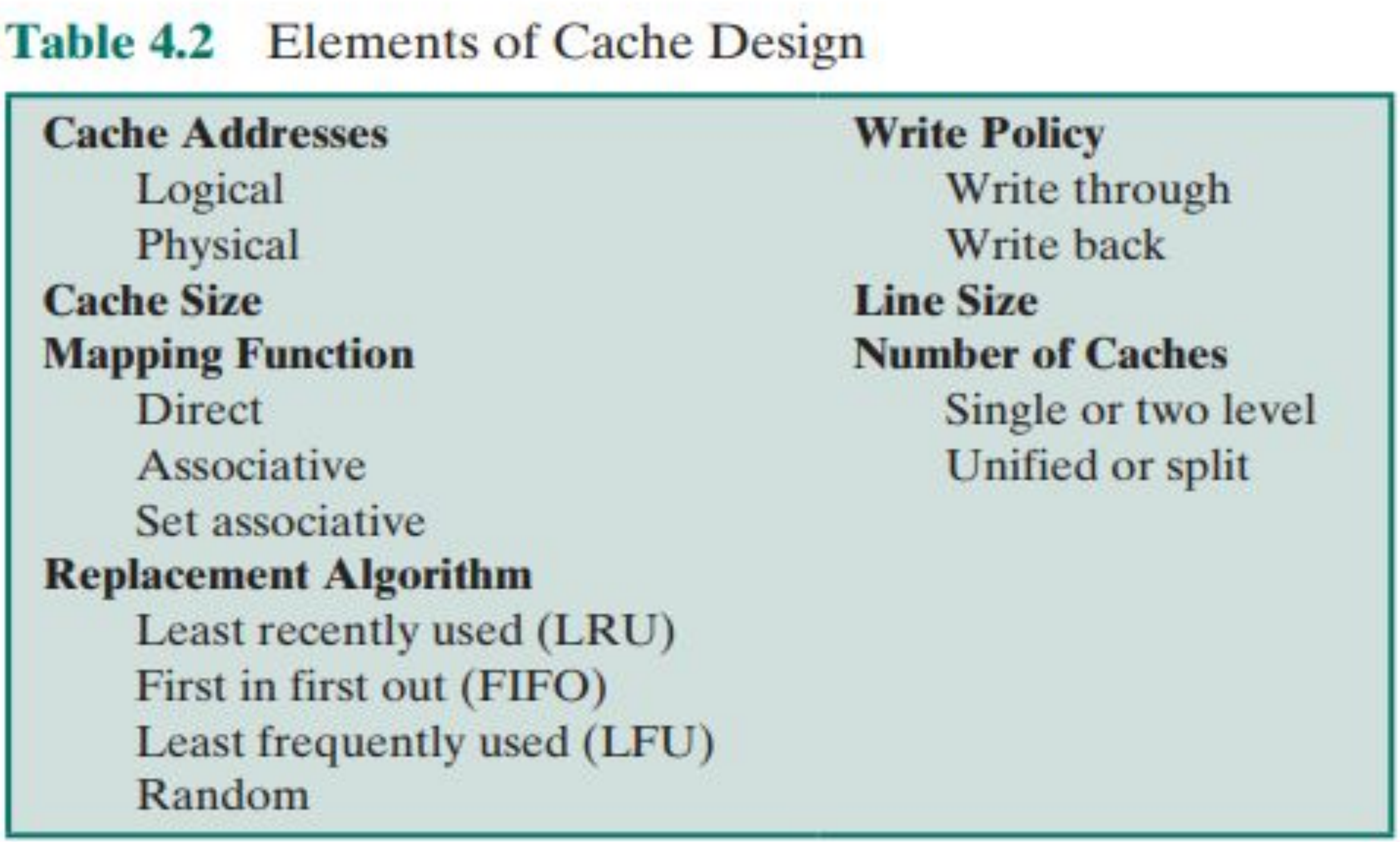


Table 4.2 Elements of Cache Design

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

Number of Caches
- Single or two level
- Unified or split

Cache Structure

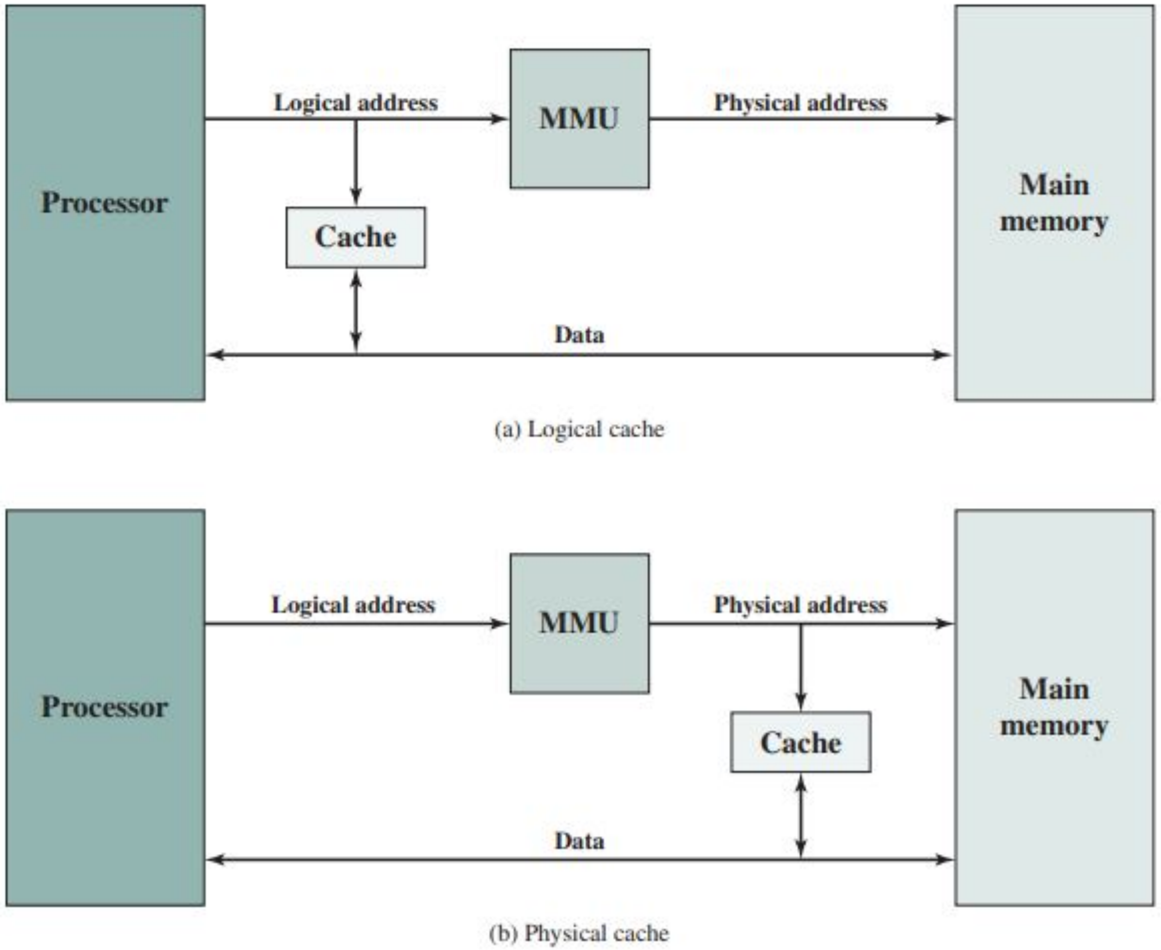
- **Index**: A number to each block/cell of cache
- **Tag**: Used to keep complete or part of address of data the was brought from main memory
- **Data**: To keep data brought from RAM
- **Value Bit**: A single bit use to show if the data is updated after bringing it in from RAM

- **Note** that **tag** kept is stored in cache so longer the tag the more space it requires in cache.
- Whereas **index** is just the cell number, doesn't need any storage.

Index	Tag	Data	Value Bit
0			
1			
2			
N			

Cache Addressing

- Depends on the location where cache is installed as shown in figures
- Logical caches that take Logical address is input
- Physical Caches that takes Physical Address is input



(a) Logical cache

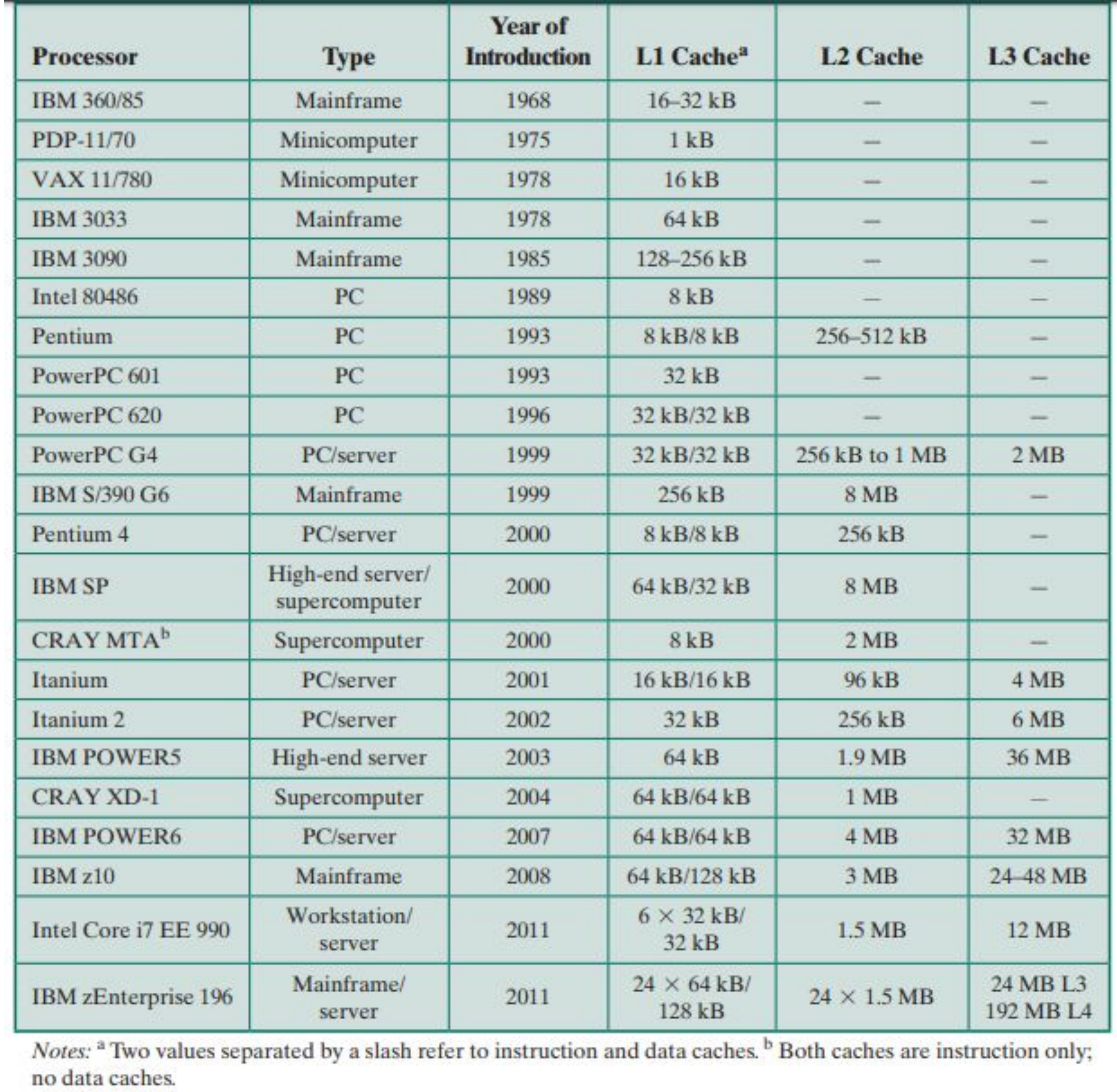
(b) Physical cache

Figure 4.7 Logical and Physical Caches

Cache Size

- Small enough so that the overall average cost per bit is close to that of main memory alone
- Large enough so that the overall average access time is close to that of the cache alone
- There are several other motivations for minimizing cache size.
  - 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—even when built with the same integrated circuit technology and put in the same place on chip and circuit board.
  - The available chip and board area also limits cache size. Because the performance of the cache is very sensitive to the nature of the workload, it is impossible to arrive at a single “optimum” cache size.
- Table 4.3 lists the cache sizes of some current and past processors.

Table 4.3 Cache Sizes of Some Processors



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

Notes: [a] Two values separated by a slash refer to instruction and data caches. [b] Both caches are instruction only; no data caches.

Mapping Function

- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- Further, a means is needed for determining which main memory block currently occupies a cache line.
- The choice of the mapping function dictates how the cache is organized. Three techniques can be used:
  - Direct
  - Associative
  - Set-associative

Associative Mapping

- Any cell of main memory can be loaded in any cell of Cache
- Addresses of RAM are stored as Tag
- The cache control logic must simultaneously examine every tag for a match
- Pros:
  - Flexibility
- Cons:
  - Larger value of tag
  - Number of bits required for tag= log2(size of RAM)
    - For 1MB RAM tag size is of 20 bits

Associative Mapping

Cache

Index	Tag	Data	Value Bit
00			
01			
11			
11			

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 0000

It’s a miss in cache

Cache

Index	Tag	Data	Value Bit
00			
01			
11			
11			

0000 will be brought to cache, on first available slot

Index	Tag	Data	Value Bit
00	0000	A	0
01			
11			
11			

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 0011, It’s a miss

Cache

Index	Tag	Data	Value Bit
00	0000	A	0
01			
10			
11			

Bring 0011 to cache

Index	Tag	Data	Value Bit
00	0000	A	0
01	0011	D	0
10			
11			

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 1000, It’s a miss

Cache

Index	Tag	Data	Value Bit
00	0000	A	0
01	0011	D	0
10			
11			

Bring 1000 to cache

Index	Tag	Data	Value Bit
00	0000	A	0
01	0011	D	0
10	1000	I	0
11			

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 1010, it’s a miss

Cache

Index	Tag	Data	Value Bit
00	0000	A	0
01	0011	D	0
10	1000	I	0
11			

Bring 1010 to cache

Index	Tag	Data	Value Bit
00	0000	A	0
01	0011	D	0
10	1000	I	0
11	1010	K	0

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 1110, It’s a miss,
there is no space in cache so we replace least recently used element (i.e., 0000)

Index	Tag	Data	Value Bit
00	0000	A	0
01	0011	D	0
10	1000	I	0
11	1010	K	0

Bring 1110 in place of least recently used

Index	Tag	Data	Value Bit
00	1110	O	0
01	0011	D	0
10	1000	I	0
11	1010	K	0

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 0000, It’s a miss
Cache is full so we replace the least recently used element (i.e 0011),

Index	Tag	Data	Value Bit
00	1110	O	0
01	0011	D	0
10	1000	I	0
11	1010	K	0

Bring 0000 in cache by replacing 0011
Note the now 0000 is on index 1 (Flexible)

Index	Tag	Data	Value Bit
00	1110	O	0
01	0000	A	0
10	1000	I	0
11	1010	K	0

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 0000
It’s a hit

Cache

Index	Tag	Data	Value Bit
00	1110	O	0
01	0000	A	0
10	1000	I	0
11	1010	K	0

Main Memory

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Associative Mapping

- We can see from last example that
  - Tag takes a lot of memory in cache
  - Index in not used at all
- Thus to save the space we can utilizing the index to save portion of address
  - This is done in **direct mapping**

Direct Mapping

- Main Memory address is divided into 2 fields
  - Index = Main Memory Address % Size of Cache
    - Data is placed in cache at this index
  - Tag = Main Memory Address / Size of Cache
    - As stored in cache along with data
  - Main Memory Address = Tag*Size of Cache + Index
- The drawback is that there is no flexibility, a certain address can only go in one designated slot in cache

Example

- If cache has 4 slots, indexes will be from 0 to 3
- Memory address 21 can only go on index 1 of cache (20%4=1)
- Tag will be 21/4 = 5
- From tag and index you can get the address back Tag * 4 + index
- = 5*4+1= 21

Example

- What address of ram is mapped on index 0, 1, 2 at this time?
- What will the address 31 go?

Index	Tag	Data	Value Bit
0	2		
1	6		
2	8		
3			
4			
5			

Index	Tag	Data	Value Bit
0			
1			
2			
3			

Cache

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Main Memory

Read 0000b (0d)
The tag and index of this address is
Tag=00b Index=00b
It's a miss

Index	Tag	Data	Value Bit
00			
01			
10			
11			

Cache

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

0000b (0d) will be placed on cache at index 00 and tag 00 will be stored

Index	Tag	Data	Value Bit
00	00	A	0
01			
10			
11			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Main Memory

Read 0100b (4d)
The tag and index of this address is
Tag=01b Index=00b
It’s a miss

Index	Tag	Data	Value Bit
00	00	A	0
01			
10			
11			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

0100b (4d) will be placed on cache at index 00 and tag 01 will be stored But index 00 already has some data, we will replace that

Note that even rest of the cache is free 0100 will go at index 00b

Index	Tag	Data	Value Bit
0	01	E	0
1			
2			
3			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Question:
See the tag at index 2 of cache. What data is placed over there?

Index	Tag	Data	Value Bit
0	00	A	0
1	10	J	0
2	11	?	0
3			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Question:
See the tag at index 2 of cache. What data is placed over there?
Answer: O i.e., data of address 1110

Index	Tag	Data	Value Bit
0	00	A	0
1	10	J	0
2	11	?	0
3			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Read 1001 (9d)
The tag and index of this address is
Tag=10b Index=01b
It’s a miss

Index	Tag	Data	Value Bit
0	00	A	0
1			
2			
3			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

1001 will be placed on cache at index 01 and tag 10 will be stored

Index	Tag	Data	Value Bit
0	00	A	0
1	10	J	0
2			
3			

Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Direct Mapping

- The size of tag will depend of size of cache and size of RAM
  - Number of bits required for tag

    =Number of bits required for Ram address – number of bits of cache index

    = log 2(Size of ram) - log 2(size of cache)

    =log2(size of Ram/ Size of Cache)

Direct Mapping

- Question:
  - If cache is of 8 words and Ram of 16 words
    - What is the size of tag?
  - If cache is of 1k Words and Ram is of 1M words
    - What is the size of tag?

Direct Mapping

- Question:
  - If cache is of 8 words and Ram of 16 words
    - What is the size of tag?
    - Answer log2(16/8)= 1
  - If cache is of 1k Words and Ram is of 1M words
    - What is the size of tag?
    - Answer = log 2(2^20/2^10)= 10

References

- William Stalling chapter 4 Cache Memory Section 4.1, 4.2, 4.3
- http://upscfever.com/upsc-fever/en/gatecse/en-gatecse-chp167.html
- http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/8-cache/dm.html