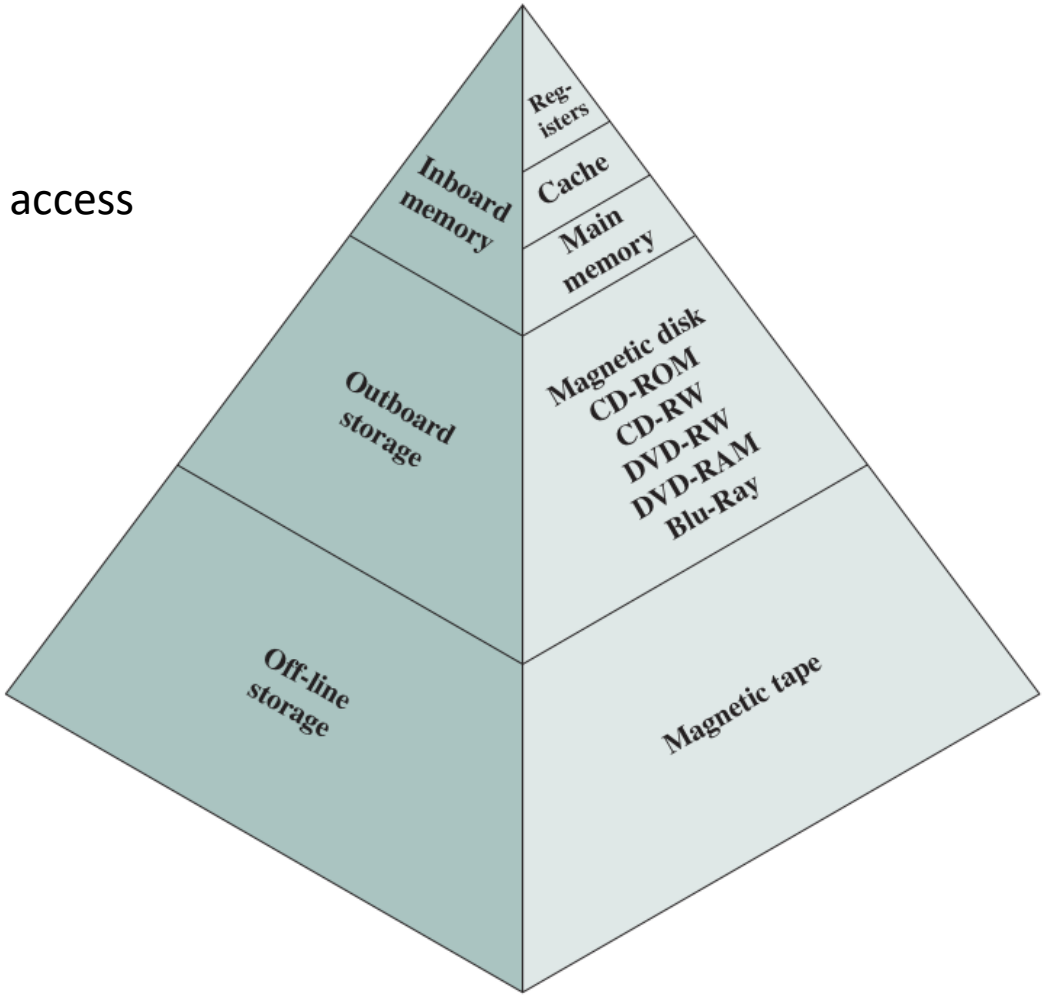
Cache Memory

Chapter 4, Stallings

Memory Hierarchy

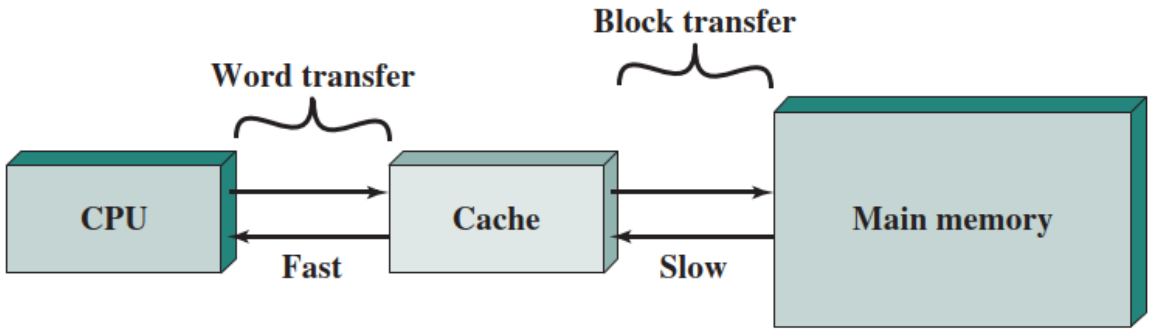
Increasing
- cost per bit
- frequency of access



Increasing
- capacity
- access time

Cache Memory Principles

- Cache memory is designed to combine the benefits of
  - low access time of expensive, high speed memory
  - large size of less expensive, lower-speed memory.
- The cache contains a copy of portions of main memory.



(a) Single cache

Cache Memory Principles

- 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

Locality of Reference

- During execution of a program, memory references by the processor, both instructions and data, tend to cluster.

(a) Locality of reference in instructions is observed due to

- Sequential instruction execution
- Loops in a program

Locality of Reference

(b) Locality of reference in data is observed due to

- Manipulation of data in arrays & tables
- Variables declared together will likely by co-located in memory

Cache Terminology

- Cache **Hit**
  - Desired value is in cache, quick access
- Cache **Miss**
  - Desired value is not in cache
  - Fetch value, and possibly its neighbors, from main memory and into cache
- **Hit Ratio**
  - What percentage of memory access are hits.

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.

Average Access Time

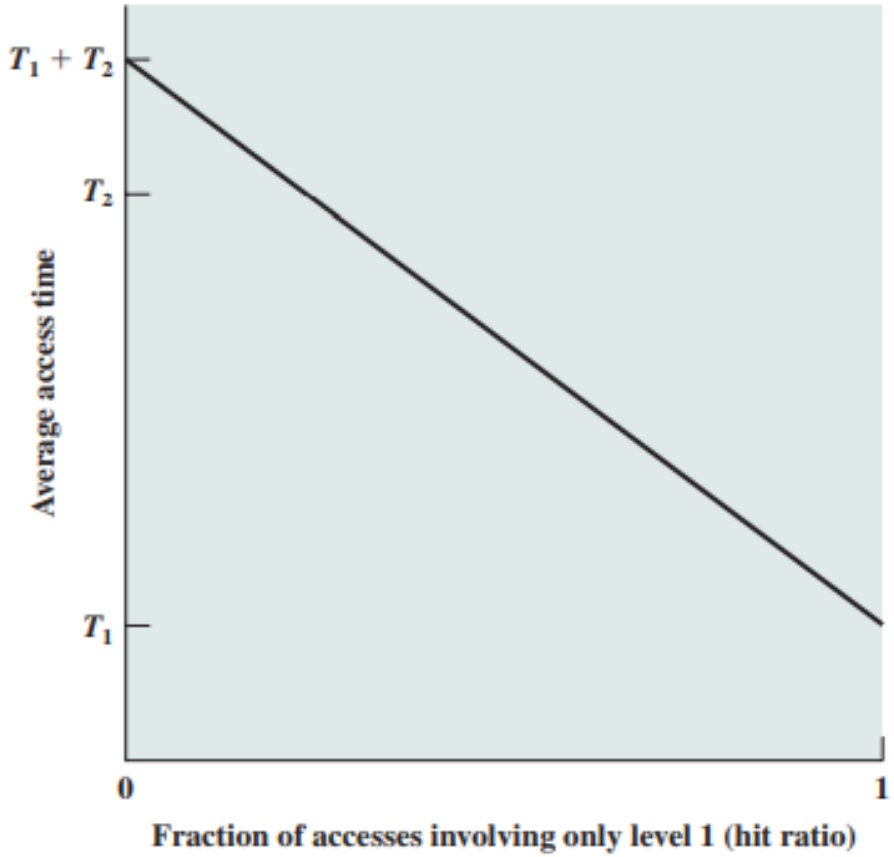
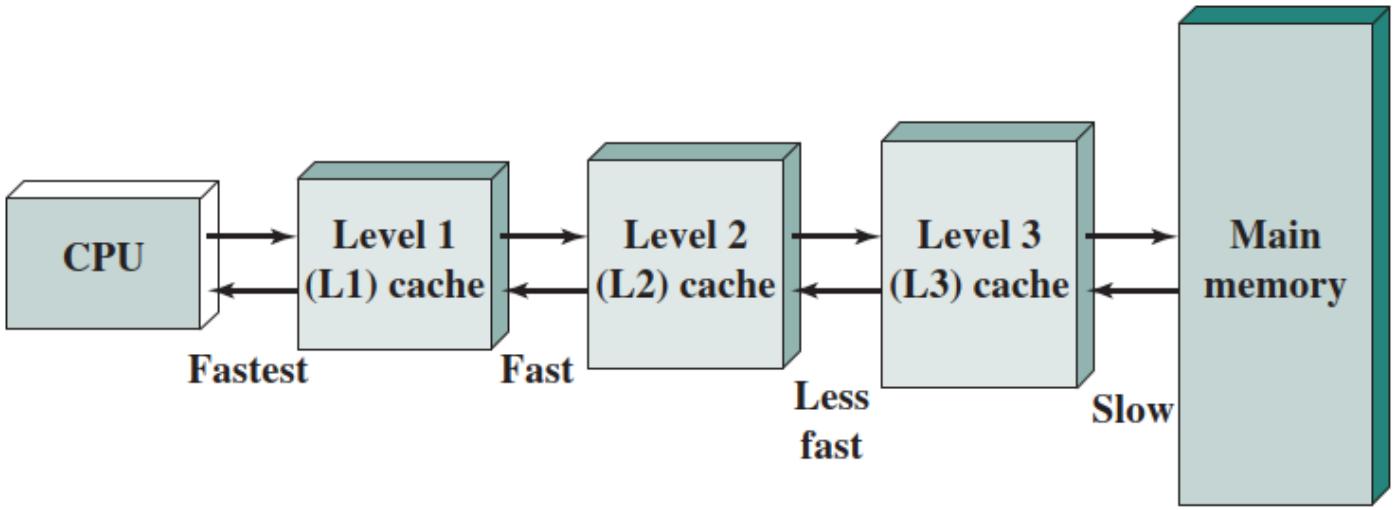


Figure 4.2 Performance of Accesses Involving only Level 1 (hit ratio)

Multi Level Cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

Average Access Time

Question:

- Consider a 3 level memory. It takes 0.001µs to read from level 1, 0.01µs to read from level 2 and 0.1µs from level 3.
- Data is found in L1 80% of time, it is found in L2 15% of the time, and in remaining 5% time data is to be read from L3.
- What is the average access time?

Average Access Time

Question:

- Consider a 3 level memory. It takes 0.001µs to read from level 1, 0.01µs to read from level 2 and 0.1µs from level 3.
- Data is found in L1 80% of time, it is found in L2 15% of the time, and in remaining 5% time data is to be read from L3.
- What is the average access time?

Answer

- Average Access time = 80%(0.001) + 15%(0.01+0.001) + 5%(0.1+0.01+0.001) = 0.008µs

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 available chip and board area also limits cache size.

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.

Cache Structure

- Index: A number to identify each block of cache
- Tag: Used to keep complete or part of address of data that was brought in from main memory
- Data: The value brought from RAM
- Valid Bit: A single bit use to show if cache block is in use (i.e. not empty).

Index	Tag	Data	Valid Bit
0			
1			
2			
N			

Mapping Function

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

(Fully) Associative Mapping

- Any block of main memory can be loaded in **any block** 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. Requires $\log_2$(size of RAM) bits
    - For 1MB RAM tag size is of 20 bits
  - Extra hardware required by CPU for searching a block in cache

Associative Mapping

Index	Tag	Data	Valid
0			0
1			0
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

For simplicity, these examples assume that block size is 1.
Real caches have a block size of 4, 8, 16 etc.

Index	Tag	Data	Valid
0			0
1			0
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

Read 0000
It's a miss

Index	Tag	Data	Valid
0			0
1			0
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

Bring in 0000 to cache

Index	Tag	Data	Valid
0	0000	A	1
1			0
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

Read 0011
It's a miss

Index	Tag	Data	Valid
0	0000	A	1
1			0
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

Bring in 0011 to cache

Index	Tag	Data	Valid
0	0000	A	1
1	0011	D	1
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

Read 1000
It's a miss

Index	Tag	Data	Valid
0	0000	A	1
1	0011	D	1
2			0
3			0

Cache

Address	Contents
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

Associative Mapping

Bring in 1000 to cache

Index	Tag	Data	Valid
0	0000	A	1
1	0011	D	1
2	1000	I	1
3			0

Cache

Address	Contents
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

Associative Mapping

Read 1010
It's a miss
Bring in 1010 to cache

Index	Tag	Data	Valid
0	0000	A	1
1	0011	D	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

Read 1110
It's a miss

Index	Tag	Data	Valid
0	0000	A	1
1	0011	D	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

Bring in 1110 to cache
There is no space in cache so we replace the least recently used element i.e. 0000

Index	Tag	Data	Valid
0	0000	A	1
1	0011	D	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

Bring in 1110 to cache
There is no space in cache so we replace the least recently used element i.e. 0000

Index	Tag	Data	Valid
0	~~0000~~	~~A~~	1
1	0011	D	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

Bring in 1110 to cache
There is no space in cache so we replace the least recently used element i.e. 0000

Index	Tag	Data	Valid
0	1110	O	1
1	0011	D	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

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

Index	Tag	Data	Valid
0	1110	O	1
1	0011	D	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

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

Index	Tag	Data	Valid
0	1110	O	1
1	~~0011~~	~~D~~	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

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

Index	Tag	Data	Valid
0	1110	O	1
1	0000	A	1
2	1000	I	1
3	1010	K	1

Cache

Address	Contents
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

Associative Mapping

Read 0000
It's a hit

Index	Tag	Data	Valid
0	1110	O	1
1	0000	A	1
2	1000	I	1
3	1010	K	1

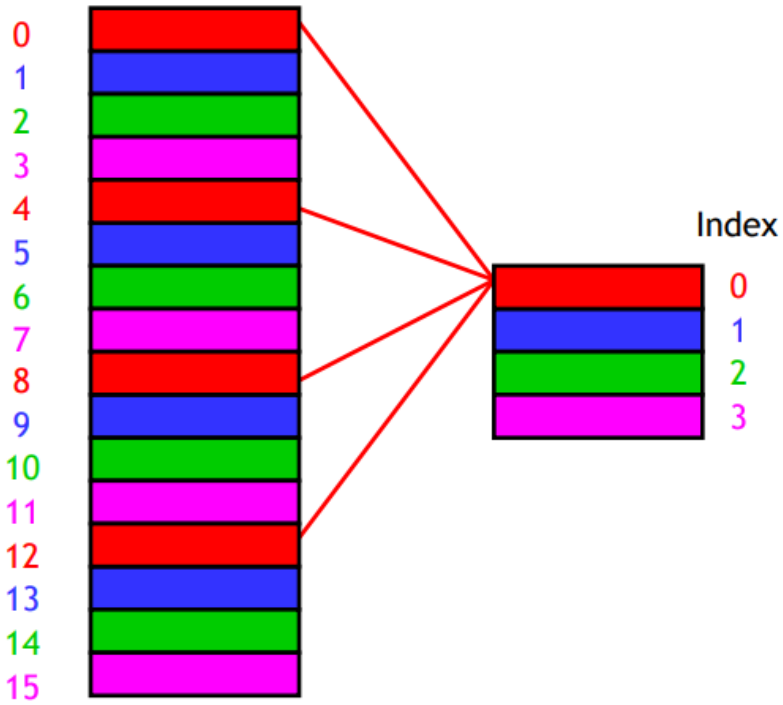
Cache

Address	Contents
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

Direct Mapping

- Each block of main memory can map into only one possible cache block

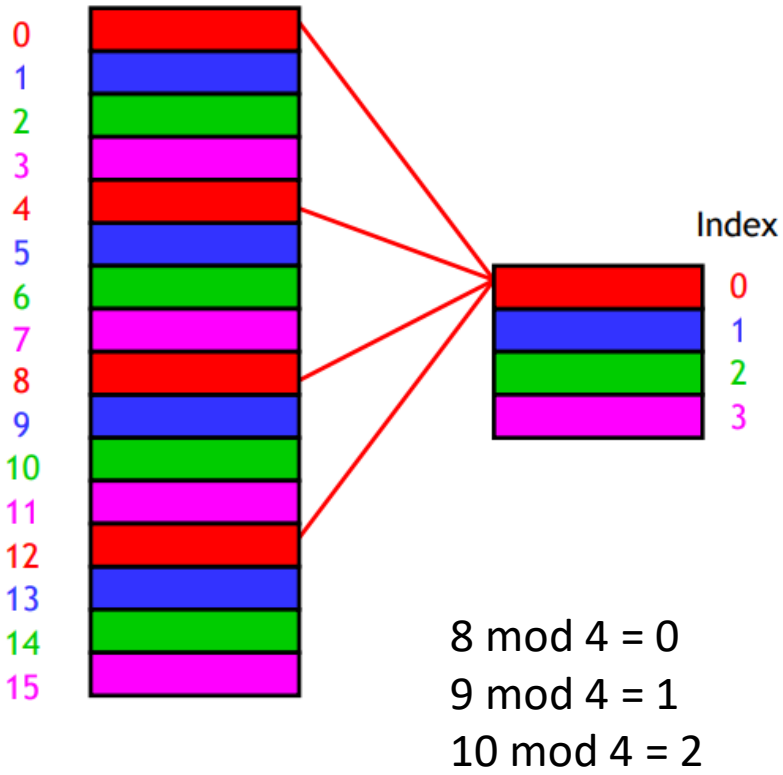


Direct Mapping

- Each block of main memory can map into only one possible cache block
- Can use mod (remainder) operation to figure out which cache location a memory block would go to.

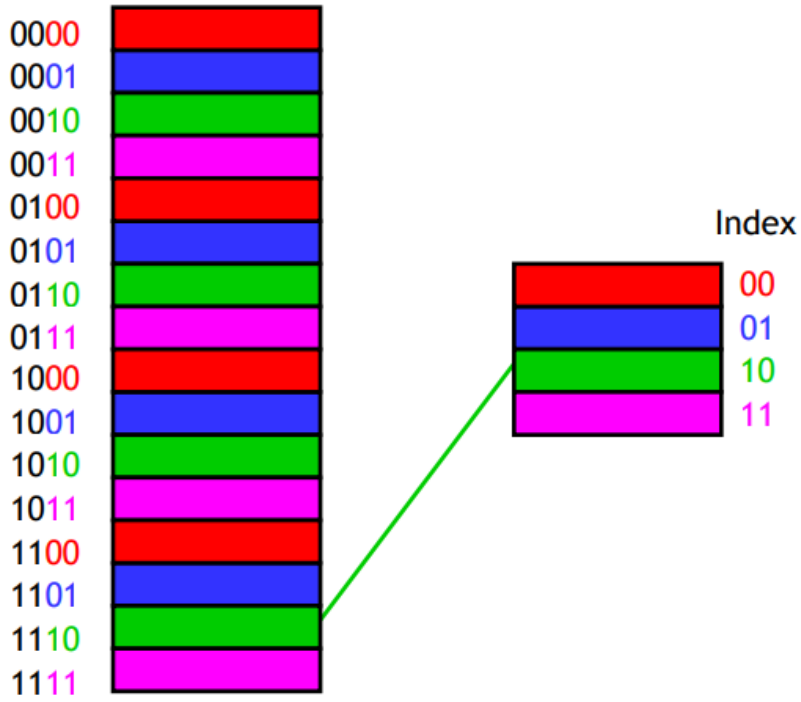
  Cache block = $j \ mod \ m$

- where j is block-number of main memory, m is number of blocks in cache



Direct Mapping

- ..... or even easier method is to use the k least significant bits where $2^k = m$ (number of block in cache)
- With a 4-block cache, take the main memory block number, divide it into two parts. Lower 2 bits indicate index, remaining bits become the tag.



Picking least significant k bits is the same as doing $mod\ 2^k$

Direct Mapping

- Main Memory address is divided into 2 fields
  - Index = Lower order bits
    - (Main memory address) mod (size of cache)
    - Data is placed in cache at this index
  - Tag = Higher order bits
    - (Main memory address) / (size of cache)
    - Stored in cache along with data

Direct Mapping

Index	Tag	Data	Valid
0			0
1			0
2			0
3			0

Cache

Address	Contents
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

Direct Mapping

Index	Tag	Data	Valid
00			0
01			0
10			0
11			0

Cache

Indexes written in binary for clarity

Address	Contents
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

Direct Mapping

Read 0000
The tag and index of this address is
Tag=00, Index=00
It's a miss

Index	Tag	Data	Valid
00			0
01			0
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

0000 will be placed in cache at index 00
tag 00 will be stored

Index	Tag	Data	Valid
00	00	A	1
01			0
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

Read 0100
The tag and index of this address is
Tag=01, Index=00
It's a miss

Index	Tag	Data	Valid
00	00	A	1
01			0
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

0100 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 00

Index	Tag	Data	Valid
00	~~00~~	~~A~~	1
01			0
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

0100 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 00

Index	Tag	Data	Valid
00	01	E	1
01			0
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

Read 1001
The tag and index of this address is
Tag=10, Index=01
It's a miss

Index	Tag	Data	Valid
00	01	E	1
01			0
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

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

Index	Tag	Data	Valid
00	01	E	1
01	10	J	1
10			0
11			0

Cache

Address	Contents
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

Direct Mapping

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

Index	Tag	Data	Valid
00	01	E	1
01	10	J	1
10	10	?	1
11			0

Cache

Address	Contents
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

Direct Mapping

Question
See the tag at index 2 of cache. What data is placed over there?
Answer: K i.e. data of address 1010

Index	Tag	Data	Valid
00	01	E	1
01	10	J	1
10	10	?	1
11			0

Cache

Address	Contents
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

Direct Mapping

- The size of tag will depend of size of cache and size of RAM
  - Number of bits required for tag = $\log_2\left(\frac{\text{RAM size}}{\text{cache size}}\right)$
  - Number of bits required for index = $\log_2(\text{cache size})$

Direct Mapping

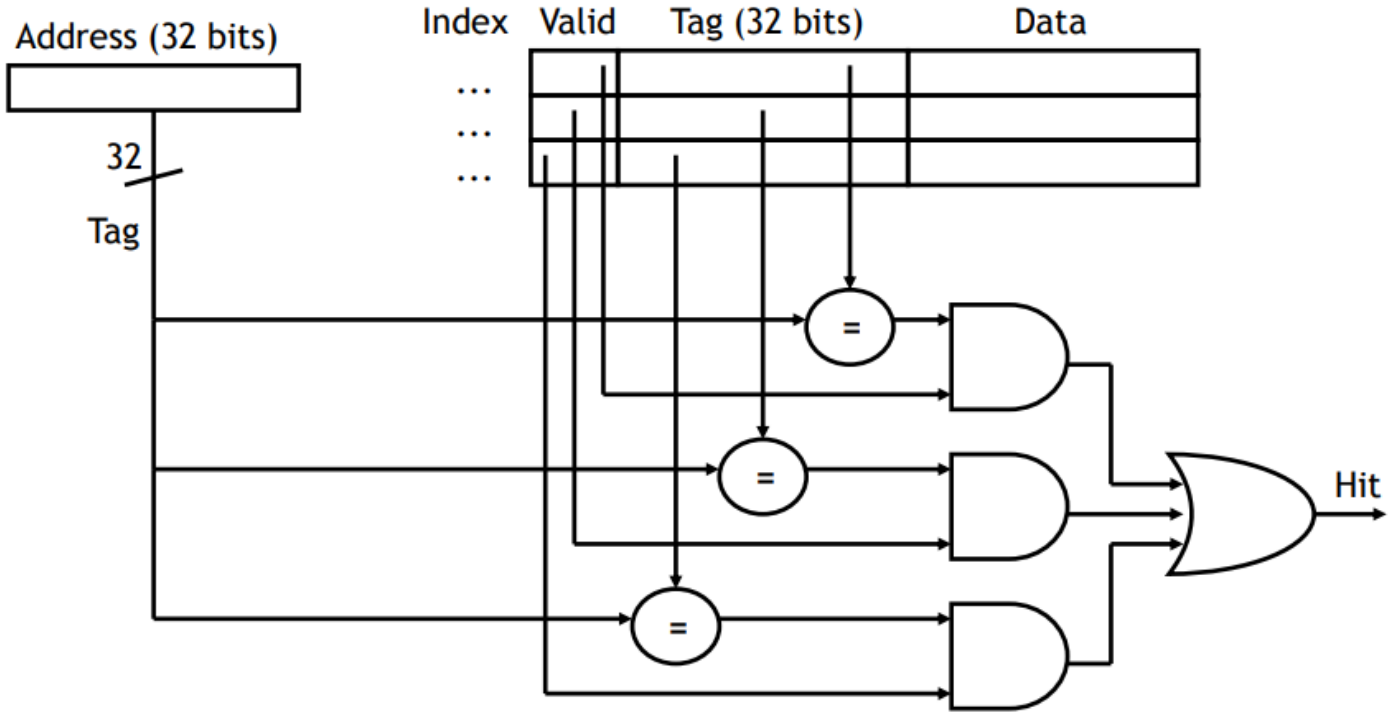
- Question:
  - If cache is of 8 words and RAM of 16 words
    - What is the size of tag?
  - If cache is of 1K words and RAM is of 1M words
    - What is the size of tag?

Direct Mapping

- Question:
  - If cache is of 8 words and RAM of 16 words
    - What is the size of tag?
    - Answer: $\log_2(16/8) = 1$
  - If cache is of 1K words and RAM is of 1M words
    - What is the size of tag?
    - Answer: $\log_2(2^{20}/2^{10}) = 10$

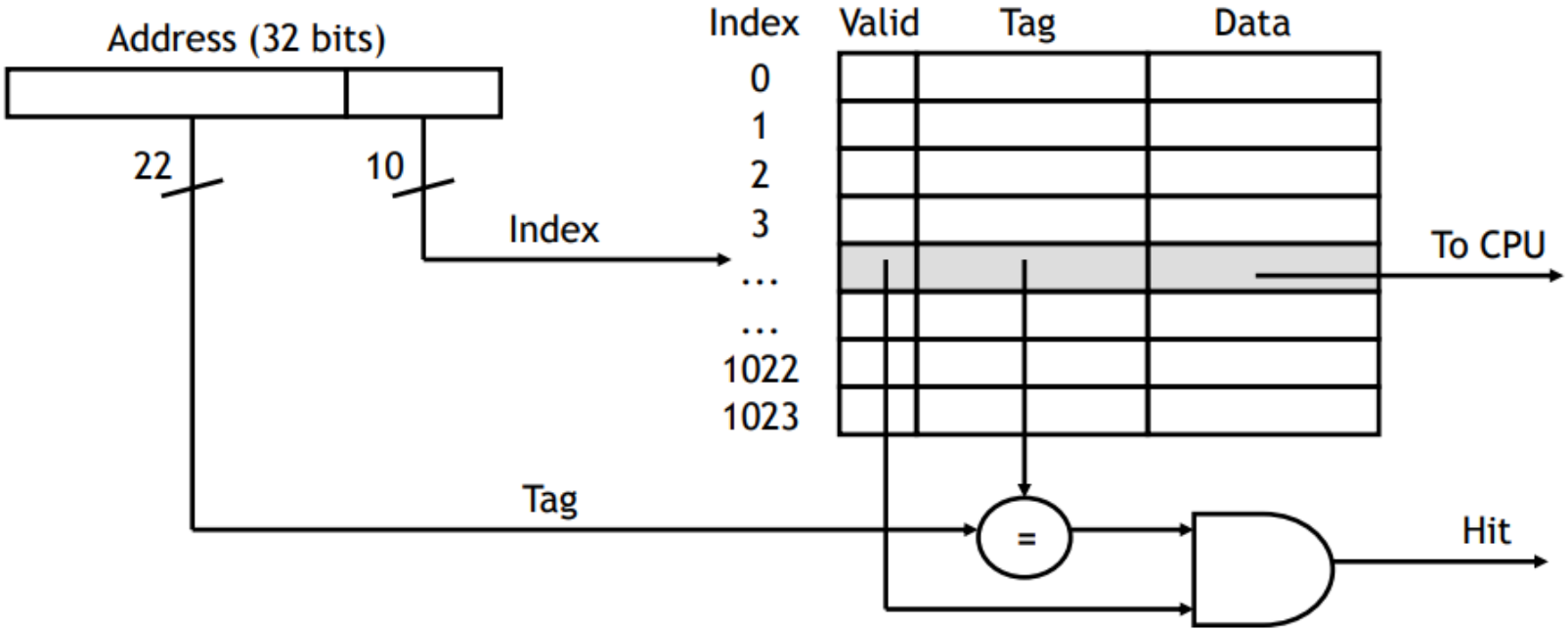
Direct vs Fully Associative Mapping

- Fully associative mapping requires one comparator for each cache block



Direct vs Fully Associative Mapping

- Direct mapping needs a single comparator



Set Associative Mapping

- In direct mapping two words with same index but different tags cannot reside in cache at same time
  - For figure given in slide 44, data of address 0000 and 0100 cannot reside in cache at same time
- In set associative mapping there are multiple sets and blocks with same index but different tags can reside in cache at same time
- If there are p sets then p elements with same index can reside in cache
- All the sets are checked simultaneously to see if the data is in cache

Set Associative Mapping

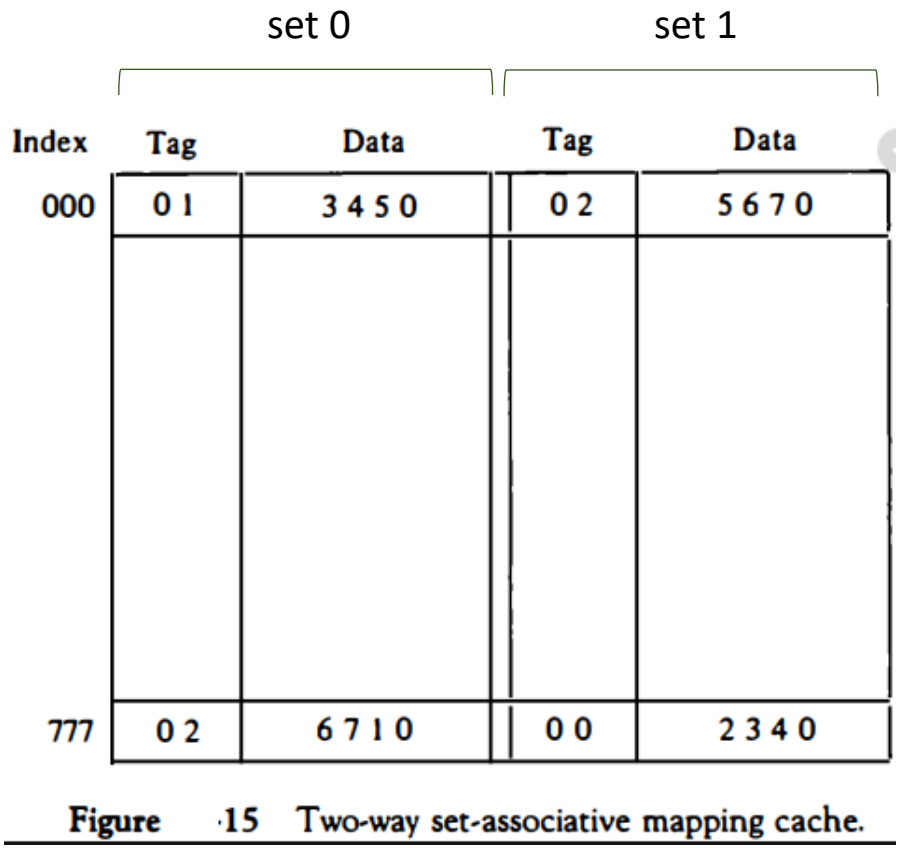
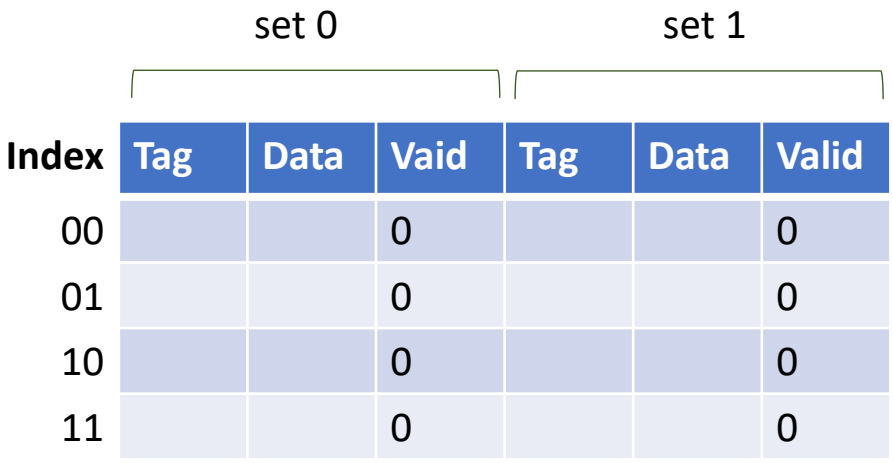


Figure 15 Two-way set-associative mapping cache.

Set Associative Mapping



Index	Tag	Data	Vaid	Tag	Data	Valid
00			0			0
01			0			0
10			0			0
11			0			0

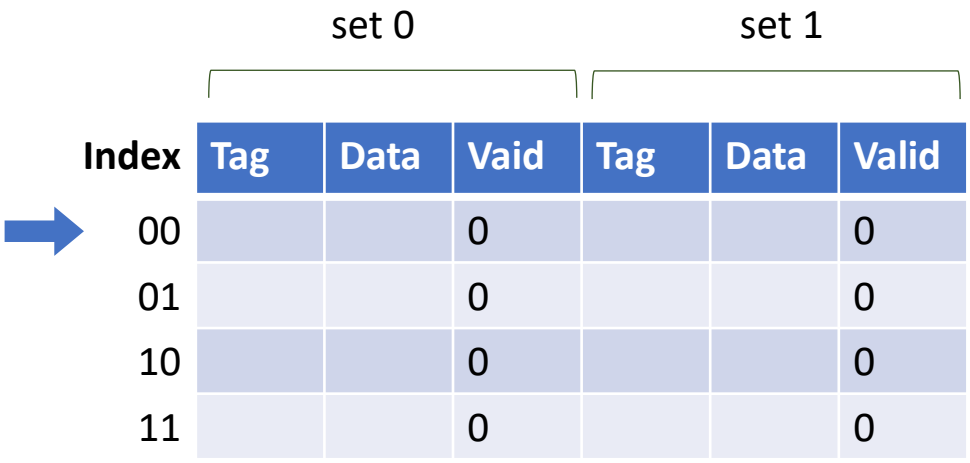
Cache

Address	Contents
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

Set Associative Mapping

Read 0000
Tag = 00, index = 00
It's a miss



Index	Tag	Data	Vaid	Tag	Data	Valid
00			0			0
01			0			0
10			0			0
11			0			0

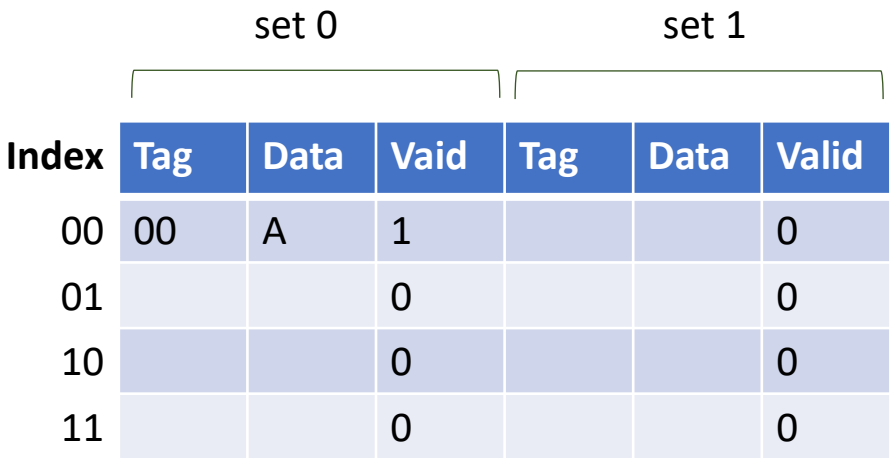
Cache

Address	Contents
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

Set Associative Mapping

0000 will be placed at index 00 and tag will be 00
We place it in set 0 because it was empty slot



Index	Tag	Data	Vaid	Tag	Data	Valid
00	00	A	1			0
01			0			0
10			0			0
11			0			0

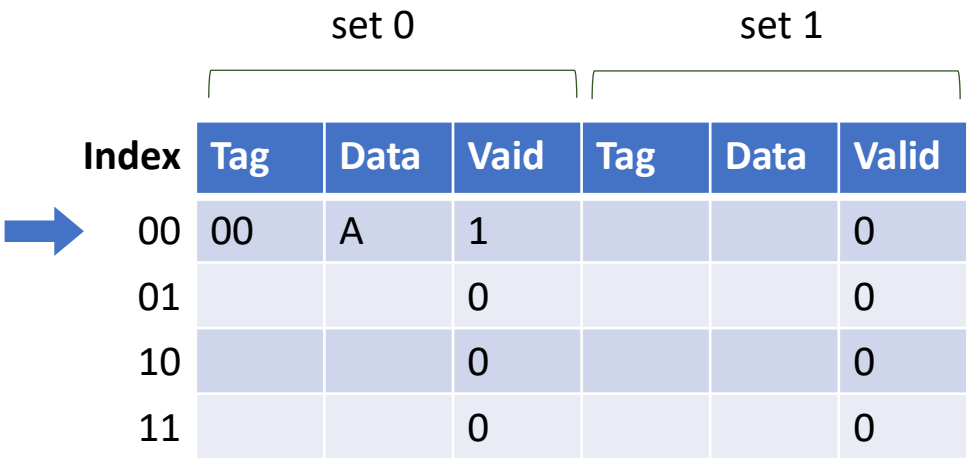
Cache

Address	Contents
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

Set Associative Mapping

Read 0100
Tag = 01, index = 00
It's a miss



Index	Tag	Data	Vaid	Tag	Data	Valid
00	00	A	1			0
01			0			0
10			0			0
11			0			0

Cache

Address	Contents
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

Set Associative Mapping

0100 will be placed at index 00 and tag will be 01
We place it in set 1 because it was empty slot

	set 0			set 1		
Index	**Tag**	**Data**	**Vaid**	**Tag**	**Data**	**Valid**
00	00	A	1	01	E	1
01			0			0
10			0			0
11			0			0

Cache

Address	Contents
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

What size of associativity set?

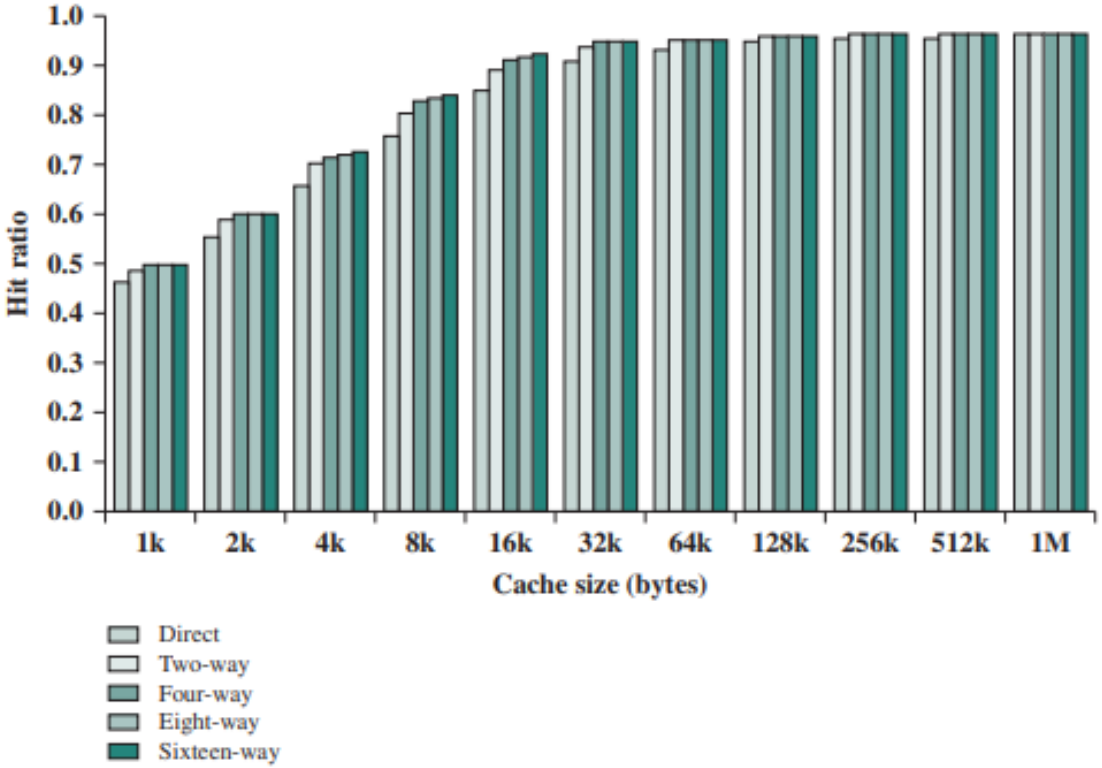


Figure 4.16 Varying Associativity over Cache Size

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, such an algorithm must be implemented in hardware.

LRU Replacement Algorithm

- Probably the most effective is least recently used (LRU):
  - Replace that block that has been in the cache longest with no reference to it.
- For two-way set associative, this is easily implemented.
  - Each set includes a USE bit. When a block is referenced, its USE bit is set to 1 and the USE bit of the other block in that set is made 0.
  - When a new block is to be read into the set, the block whose USE bit is 0 is ejected.
  - Because we are assuming that more recently used memory locations are more likely to be referenced, LRU should give the best hit ratio.

LRU Replacement Algorithm

- LRU is also relatively easy to implement for a fully associative cache.
  - The cache mechanism maintains a separate list (queue) of indexes to all the blocks in the cache.
  - When a blocks is referenced, it moves to the front of the list.
  - For replacement, the block at the back of the list is used.
- Because of its simplicity of implementation, LRU is the most popular replacement algorithm.

Other Replacement Policies

- First-in-first-out (FIFO):
  - Replace that block in the set that has been in the cache longest.
  - FIFO is 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.
  - LFU could be implemented by associating a counter with each line.
- Random replacement
  - Pick a line at random from among the candidate lines.
- Simulation studies have shown that random replacement provides only slightly inferior performance to an algorithm based on usage [SMIT82].

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 straight away.
  - 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.

Writing Policy

- The simplest technique is called **write through**.
  - Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid.
  - Disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck.

Writing Policy

- An alternative technique, known as **write back**, minimizes memory writes.
  - With write back, updates are made only in the cache.
  - When an update occurs, a dirty bit, or value bit, associated with the line is set.
  - When a block is replaced, it is written back to main memory if and only if the dirty bit is set.
  - The problem with write back is that portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache.

Exercise

Question

- Consider a cache that can store 4 words
- Main memory has a size of 64 words
- The cache is using **direct mapping**
- For the following addresses of RAM accesses, find the hits and misses:
  - 10, 12, 10, 12, 63, 58, 34, 10, 34, 63, 33, 10, 63

Question

- Consider a cache that can store 4 words
- Main memory has a size of 64 words
- The cache is using **associative mapping** and LRU is used as replacement policy
- For the following addresses of RAM accesses, find the hits and misses:
  - 10, 12, 10, 12, 63, 58, 34, 10, 34, 63, 33, 10, 63

Question

- Consider a cache that can store 4 words
- Main memory has a size of 64 words
- The cache is using **2-way set associative mapping** and LRU is used as replacement policy
- For the following addresses of RAM accesses, find the hits and misses:
  - 10, 12, 10, 12, 63, 58, 34, 10, 34, 63, 33, 10, 63

References

- William Stallings, Chapter 4 sections 4.1 to 4.3
- http://upscfever.com/upsc-fever/en/gatecse/en-gatecse-chp167.html
- https://web.archive.org/web/20210810084306/http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/8-cache/dm.html