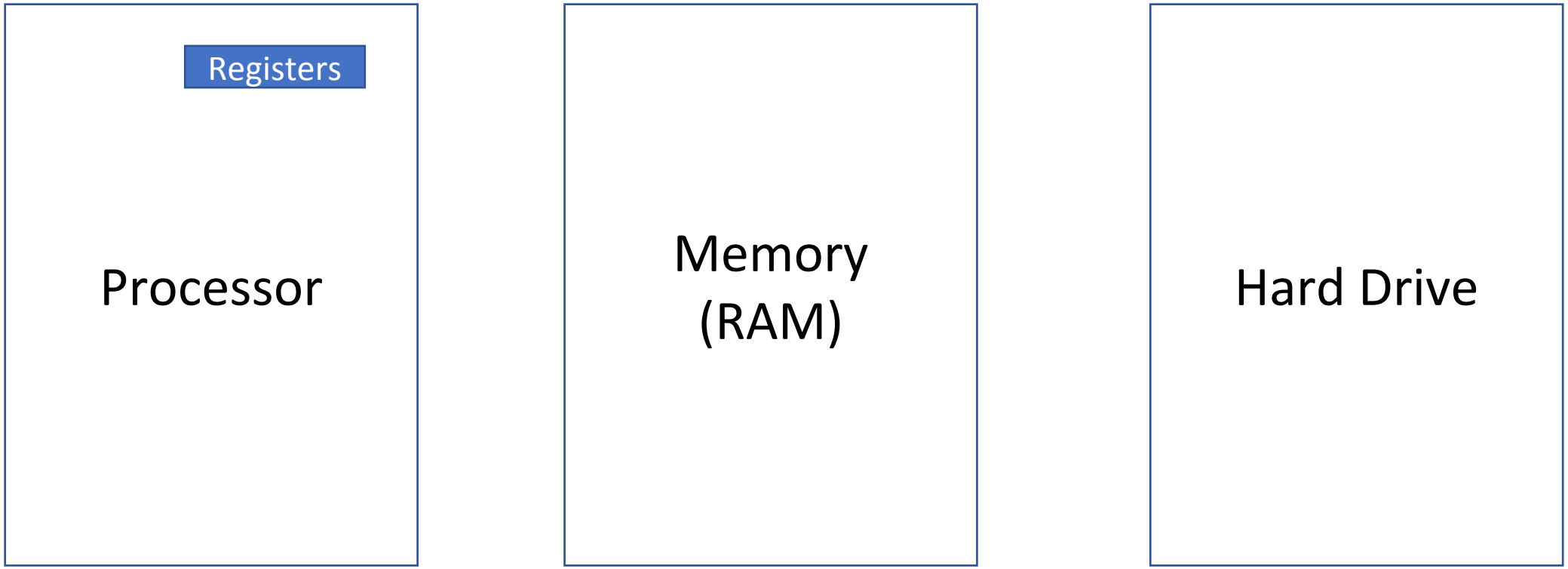
Cache Memory

By: Aatira Anum

Memory Organization



Key Characteristics of Computer Memory Systems

- Location
  - Internal
  - External
- Capacity
  - No of Words/Bytes
- Unit of Transfer
  - Word
  - Block
- Performance
  - Access Time

Memory Organization

- Registers
  - Temporary Storage inside processor
  - Very Fast
  - Very expensive
  - Small capacity/size
- Main Memory
  - Random access memory
  - Internal Memory
  - Normal fast
  - Affordable
  - Medium Capacity
  - Temporary Storage
- Hard Drive
  - Slow
  - Cheap
  - Large Size/capacity
  - Permanent Storage

Main Memory Latency is Long

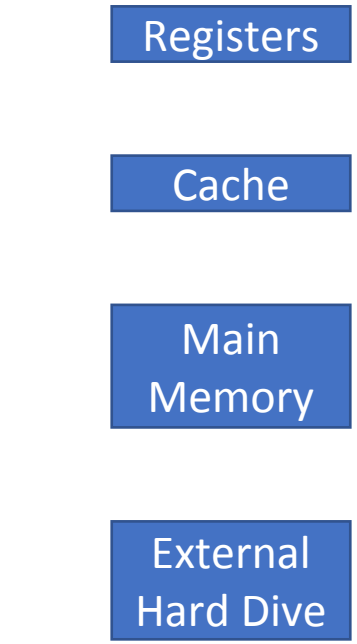
- 2.0 GHz CPU ➡ 0.5 ns cycle time

- 100 ns memory ➡ 200 cc memory latency

- Solution: Caches

Caches

- More closer to processor than main memory
- Very Fast and Very Expensive
- Smaller size and capacity



Memory Hierarchy

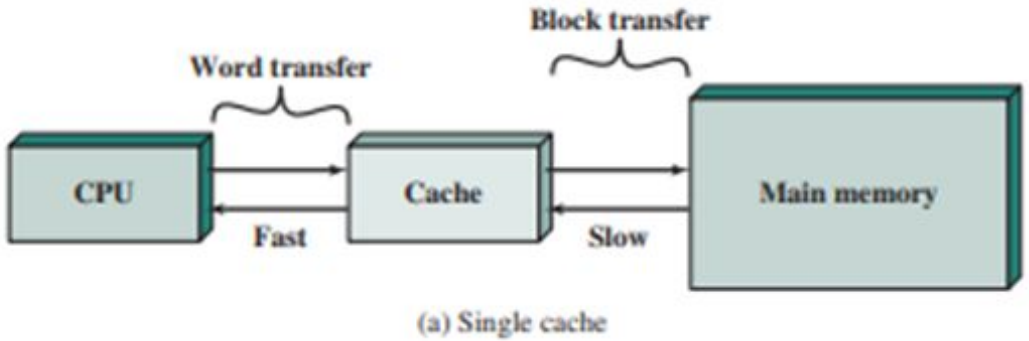
Cache Memory Principles

- 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.
- 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.

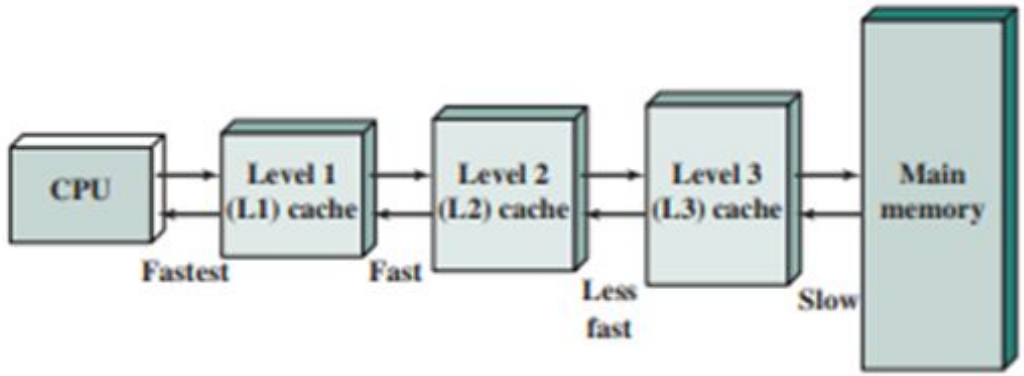
Cache Memory Principles

- Cache hit
  - copy is in cache, quick access
- Cache miss
  - copy not in cache, read address and possibly its neighbors into cache

Levels of Cache



(a) Single cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

Average Access Time

- Suppose it requires 100 ns to access memory
- Suppose the processor has two-level of cache memory
- It requires 10 ns to access level 1 cache and 90% of memory accesses are found in cache level 1
- It requires 30 ns to access level 2 cache and 10% of memory accesses are found in cache level 2
- What is the average time to access a location?
- Average Access Time = 0.9(10) + 0.1(10 + 30) = 13 ns

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

Index	Tag	Data	Value Bit
0			
1			
2			
N			

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
- 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.

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

Index	Tag	Data	Value Bit
0			
1			
2			
3			

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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 0000
It's a miss

Index	Tag	Data	Value Bit
0			
1			
2			
3			

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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

Bring 0000 to Cache

Index	Tag	Data	Value Bit
0	0000	A	0
1			
2			
3			

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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 0011
It's a miss

Index	Tag	Data	Value Bit
0	0000	A	0
1			
2			
3			

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000,0000

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

Bring 0011 to cache

Index	Tag	Data	Value Bit
0	0000	A	0
1	0011	D	0
2			
3			

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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 1000
It's a miss

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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 1010. It's a miss. Bring 1010 to cache

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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 1110
It’s a miss, there is no space.

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000,0000

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

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.

Replacement Algorithm (LRU)

- Probably the most effective is least recently used (LRU):
  - Replace that block in the set that has been in the cache longest with no reference to it.
- LRU is also relatively easy to implement for a fully associative cache.
  - The cache mechanism maintains a separate list of indexes to all the lines in the cache.
  - When a line is referenced, it moves to the front of the list.
  - For replacement, the line 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.

Read 1110
It’s a miss, there is no space.

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000,0000

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

Bring 1110 in place of least recently used, which is 0000

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000, 0000

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

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

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000,0000

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 0000
It's a hit

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

Cache

Addresses to access in memory:
0000, 0011, 1000, 1010, 1110, 0000,0000

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

Hit and Miss Rate

- 0000, 0011, 1000, 1010, 1110, 0000,0000
- Hit rate = 1/7
- Miss rate = 6/7

Direct Mapping

- Main Memory address is dived 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

Direct Mapping

- The size of tag will depend of size of cache and size of RAM
  - Number of bits required for tag= log2(size of Ram/ Size of Cache)
  - Number of bits required for index= log2(size of cache)

- Tip:
  - Tagb:Indexb= Main Memory Address b

Direct Mapping

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

- If cache is of 8 words and Ram of 16 words
    - What is the size of tag?
    - Answer log2(16/8)= 1

Direct Mapping

- Question:
  - If cache is of 1k Words and Ram is of 1M 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?
    - Answer = log 2(2^20/2^10)= 10

Index	Tag	Data	Value Bit
0			
1			
2			
3			

Cache

Addresses to access in memory:
0000, 0100, 1001

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

- Index = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

Index	Tag	Data	Value Bit
0			
1			
2			
3			

Cache

Addresses to access in memory:
0000, 0100, 1001

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

0000b will be placed on cache at index 00 and tag 00 will be stored

- Index = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

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

Cache

Addresses to access in memory:
0000, 0100, 1001

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 = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

Index	Tag	Data	Value Bit
0	00	A	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

Addresses to access in memory:
0000, 0100, 1001

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 = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

Index	Tag	Data	Value Bit
0	01	E	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

Addresses to access in memory:
0000, 0100, 1001

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

- Index = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

Index	Tag	Data	Value Bit
0	01	E	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

Addresses to access in memory:
0000, 0100, 1001

- Index = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

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

Index	Tag	Data	Value Bit
0	01	E	0
1	10	J	0
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

Addresses to access in memory:
0000, 0100, 1001

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

- Index = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache
- Main Memory Address= Tag*Size of Cache + Index

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

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

- Index = Main Memory Address% Size of Cache
- Tag= Main Memory Address / Size of Cache

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

Cache

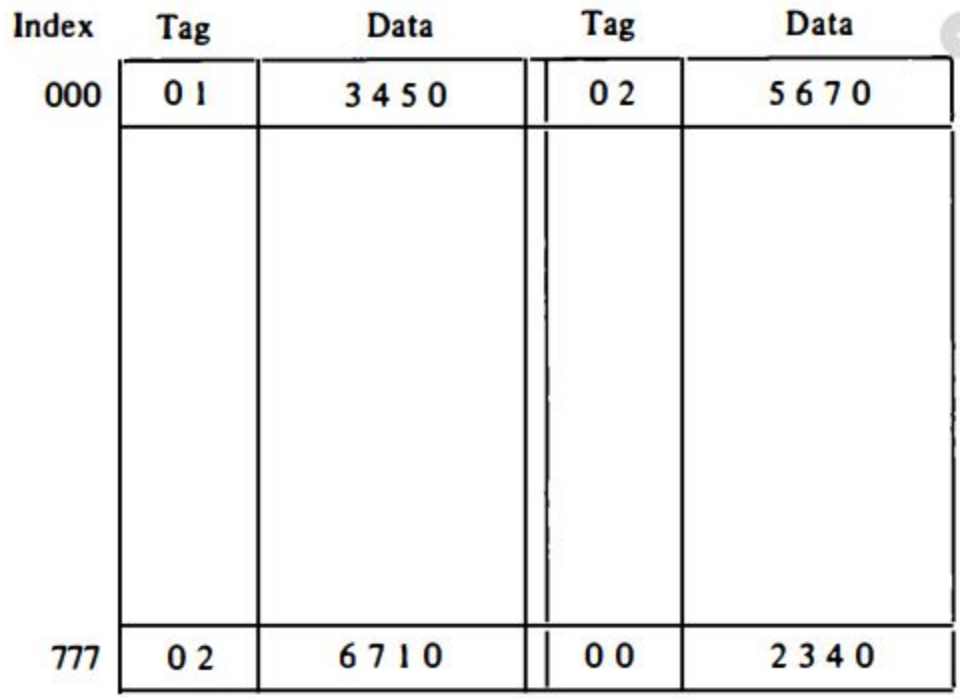
Address	Word
0000	A
0001	B
0010	C
0011	D
0100	E
0101	F
0110	G
0111	H
1000	I
1001	J
1010	K
1011	L
1100	M
1101	N
1110	O
1111	P

Main Memory

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 39, data of address 0000 and 0100 cannot reside in cache at same time
- In set associative mapping there are multiple sets and words with same index but different tags can reside in cache at same time
- If there are k sets then k elements with same index can reside in cache
- All the sets are checked simultaneously to see if the data is in cache

- set 0 set 1



Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Figure 15 Two-way set-associative mapping cache.

	set 0			set 1		
Index	**Tag**	**Data**	**Value Bit**	**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 0000
Tag= 00 index =00
It’s a miss

set 0 set 1

Index	Tag	Data	Value Bit	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

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

	set 0			set 1		
Index	**Tag**	**Data**	**Value Bit**	**Tag**	**Data**	**Value Bit**
0	00	A	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 0100
Tag= 01 index =00
It's a miss

set 0 set 1

Index	Tag	Data	Value Bit	Tag	Data	Value Bit
0	00	A	0	01	E	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

0100 will be placed it in 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**	**Value Bit**	**Tag**	**Data**	**Value Bit**
0	00	A	0	01	E	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

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.

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.
- 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.