SET ASSOCIATIVE MAPPING

In direct Mapping two words with same index but different tags cannot reside in cache at same time

- Previously we saw that data of address 0000 and 0100 cannot reside in cache at same time (for two bit index cache)

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**	**Value Bit**	**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

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**
00						
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

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

		set 0			set	
Index	**Tag**	**Data**	**Value Bit**	**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

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

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

1

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

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**
00	00	A	0	01	E	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

SET ASSOCIATIVE

We can see that set associative combines the good of both the direct and the associative mapping

Graph shows that how hit ratio changes with the size of cache and number of sets

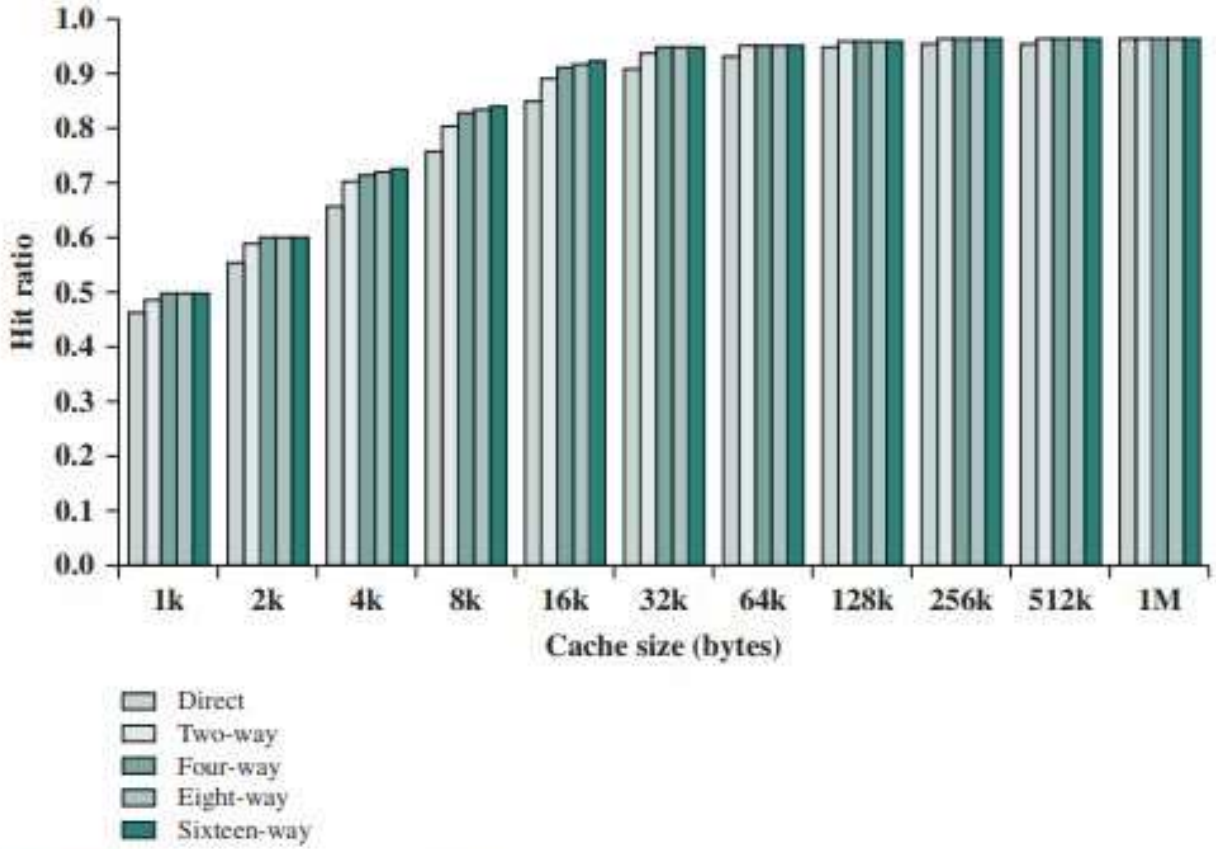

Figure 4.16 Varying Associativity over Cache Size

CACHE LINE AND MAIN MEMORY BLOCK

Recall that we said earlier that whole **block** is read from cache instead of one byte or 2 bytes

If memory contains 2^n addressable words

Memory can be broken up into blocks with K words per block.

Number of blocks = 2^n / K

Cache consists of C lines or slots, each consisting of K words

Size of block and line is same

And in reality blocks are mapped to lines. But process is same as we have seen up till now. Block number is used to find Tag and index

Whole block is copied to the designated index

LINE Index	Tag	Data	Value Bit
00			
01			
10			
11			

Cache

Main Memory

Block #	Block data
0000	Data of block 0000
0001	Data of block 0001
0010	Data of block 0010
…	…
1110	Data of block 1110
1111	Data of block 1111

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

LINE Index	Tag	Data	Value Bit
00			
01			
10			
11			

Cache

Main Memory

Block #	Block data
0000	Data of block 0000
0001	Data of block 0001
0010	Data of block 0010
…	…
1110	Data of block 1110
1111	Data of block 1111

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

LINE Index	Tag	Data	Value Bit
00	00	Data of block 0000	
01			
10			
11			

Cache

Main Memory

Block #	Block data
0000	Data of block 0000
0001	Data of block 0001
0010	Data of block 0010
…	…
1110	Data of block 1110
1111	Data of block 1111

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

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.

For two-way set associative, this is easily implemented.

- Each line includes a USE bit. When a line is referenced, its USE bit is set to 1 and the USE bit of the other line in that set is set to 0.
- When a block is to be read into the set, the line whose USE bit is 0 is used.
- Because we are assuming that more recently used memory locations are more likely to be referenced, LRU should give the best hit ratio.

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.

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 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 (AKA 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.