CDA 3103: Study Set 8

MEMORY HIERARCHY, CACHE HITS AND MISSES, ASSOCIATIVITY, BLOCK REPLACEMENT POLICIES, WRITE POLICIES

Review: Memory Hierarchy

To give the illusion of large amounts of fast memory, we a build a memory hierarchy:

- Fast memory is expensive to build, so we usually create small segments to serve the processor (caches)
- Slow memory is relatively cheap, so we usually use it for permanent storage (hard drives)
- In between, computers have “main memory” which is faster than the hard drive, but slower than the caches. It is cheaper than caches, but more expensive than hard drives in terms of cost per byte of storage. (RAM)

For the processor to access a piece of information, that information has to be loaded into every part of the hierarchy.

- Instructions for a program are copied from permanent storage to main memory in “pages”
- Sections of each page (“blocks”) are copied from main memory to the instruction cache
- The processor retrieves instructions (individual words) from the correct, valid block of the instruction cache and runs them on the datapath

Review: Cache Hits and Misses

Whenever anything is accessed, we begin by checking the highest level cache to see if the information is present there. For example, we will check the data cache for a piece of information first before checking main memory or permanent storage.

- If the information is found in the cache, it is called a cache hit
- If the information is not found in the cache, it is called a cache miss

Review: Direct-Mapped Caches

The simplest organization for caches is direct mapped. In these caches any piece of information has exactly one valid location. To find that location, we take the address of the information and mod it by the number of blocks in the cache.

Each block has two additional pieces of information:

- Tag: the upper portion of the address to distinguish separate pieces of information that map to the same location
- Valid bit: a single bit to identify whether there is any information currently in the cache block

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 1:

To set up the cache, we need 16 blocks that can each contain one word. These will have index values 0-15. Since the cache is initially empty, all of the valid bits are 0.

Solution 1:

Cache Index	Valid Bit	Tag	Contents
0000	0		
0001	0		
0010	0		
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	0		
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	0		
1111	0		

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 2:

The first address we want to read from is 0. We would represent this as (0000 0000). The lower 4 bits are the cache index. The upper 4 bits are the tag.

We will first look in the cache at index 0000. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[0] from a lower level of memory and add it to our cache at index 0000. We will set the valid bit to 1 to show there is information in the cache.

Solution 2:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	0		
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	0		
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	0		
1111	0		

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, **15**, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 3:

The next address is 15 or (0000 1111). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1111. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[15] from a lower level of memory and add it to our cache at index 1111. We will set the valid bit to 1 to show there is information in the cache.

Solution 3:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	0		
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	0		
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	0		
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, **2**, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 4:

The next address is 2 or (0000 0010). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 0010. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[2] from a lower level of memory and add it to our cache at index 0010. We will set the valid bit to 1 to show there is information in the cache.

Solution 4:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	0		
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	0		
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, **8**, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 5:

The next address is 8 or (0000 1000). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1000. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[8] from a lower level of memory and add it to our cache at index 1000. We will set the valid bit to 1 to show there is information in the cache.

Solution 5:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	0		
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, **14**, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 6:

The next address is 14 or (0000 1110). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1110. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[14] from a lower level of memory and add it to our cache at index 1110. We will set the valid bit to 1 to show there is information in the cache.

Solution 6:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given: Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, **15**, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 7: The next address is 15 or (0000 1111). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1111. The valid bit is 1, which indicates there is information in this cache block. We need to compare the tag in the cache with the tag of our address. Since they are the same (both are 0000), we can assert that information we are looking for is already in the cache. This is a cache hit.

Solution 7:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	0		
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, **26**, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 8:

The next address is 26 or (0001 1010). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1010. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[26] from a lower level of memory and add it to our cache at index 1010. We will set the valid bit to 1 to show there is information in the cache.

Solution 8:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	1	0001	[26]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, **2**, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 9:

The next address is 2 or (0000 0010). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 0010. The valid bit is 1, which indicates there is information in this cache block. We need to compare the tag in the cache with the tag of our address. Since they are the same (both are 0000), we can assert that information we are looking for is already in the cache. This is a cache hit.

Solution 9:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	1	0001	[26]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, **0**, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 10:

The next address is 0 or (0000 0000). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 0000. The valid bit is 1, which indicates there is information in this cache block. We need to compare the tag in the cache with the tag of our address. Since they are the same (both are 0000), we can assert that information we are looking for is already in the cache. This is a cache hit.

Solution 10:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	0		
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	1	0001	[26]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, **19**, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 11:

The next address is 19 or (0001 0011). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 0011. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[19] from a lower level of memory and add it to our cache at index 0011. We will set the valid bit to 1 to show there is information in the cache.

Solution 11:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	1	0001	[19]
0100	0		
0101	0		
0110	0		
0111	0		
1000	1	0000	[8]
1001	0		
1010	1	0001	[26]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, **7**, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 12:

The next address is 7 or (0000 0111). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 0111. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[7] from a lower level of memory and add it to our cache at index 0111. We will set the valid bit to 1 to show there is information in the cache.

Solution 12:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	1	0001	[19]
0100	0		
0101	0		
0110	0		
0111	1	0000	[7]
1000	1	0000	[8]
1001	0		
1010	1	0001	[26]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, **10**, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 13:

The next address is 10 or (0000 1010). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1010. The valid bit is 1, which indicates there is information in this cache block. We need to compare the tag in the cache with the tag of our address. These tags are different (0001 vs 0000), we must replace the current contents of block 1010 with a copy of Mem[10] from a lower level of memory and update the tag. The valid bit does not change.

Solution 13:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	1	0001	[19]
0100	0		
0101	0		
0110	0		
0111	1	0000	[7]
1000	1	0000	[8]
1001	0		
1010	1	0000	[10]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, **8**, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 14:

The next address is 8 or (0000 1000). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1000. The valid bit is 1, which indicates there is information in this cache block. We need to compare the tag in the cache with the tag of our address. Since they are the same (both are 0000), we can assert that information we are looking for is already in the cache. This is a cache hit.

Solution 14:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	1	0001	[19]
0100	0		
0101	0		
0110	0		
0111	1	0000	[7]
1000	1	0000	[8]
1001	0		
1010	1	0000	[10]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, **14**, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 15:

The next address is 14 or (0000 1110). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1110. The valid bit is 1, which indicates there is information in this cache block. We need to compare the tag in the cache with the tag of our address. Since they are the same (both are 0000), we can assert that information we are looking for is already in the cache. This is a cache hit.

Solution 15:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	1	0001	[19]
0100	0		
0101	0		
0110	0		
0111	1	0000	[7]
1000	1	0000	[8]
1001	0		
1010	1	0000	[10]
1011	0		
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

Example: Direct-Mapped Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, **11**

The content at address 0 can be shown as [0]. Assume the cache is initially empty. Show the contents of the cache at the end of the above reading operations if the cache is direct mapped. Use 8 bits for any addresses.

Partial Credit 16:

The final address is 1 or (0000 1011). The lower 4 bits are the cache index. The upper 4 bits are the tag.

First , we check the cache at index 1011. Since the valid bit is 0, we know there is no information in the cache. We must retrieve a copy of Mem[11] from a lower level of memory and add it to our cache at index 1011. We will set the valid bit to 1 to show there is information in the cache.

Solution 16:

Cache Index	Valid Bit	Tag	Contents
0000	1	0000	[0]
0001	0		
0010	1	0000	[2]
0011	1	0001	[19]
0100	0		
0101	0		
0110	0		
0111	1	0000	[7]
1000	1	0000	[8]
1001	0		
1010	1	0000	[10]
1011	1	0000	[11]
1100	0		
1101	0		
1110	1	0000	[14]
1111	1	0000	[15]

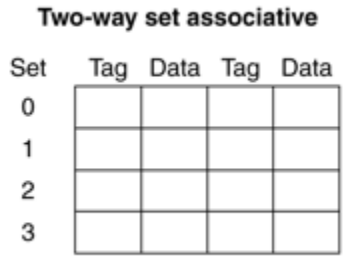
Review: Associativity

To reduce the number of conflicts in the cache, we can increase the cache associativity. This will allow multiple blocks that map to the same address to be in the cache at the same time.

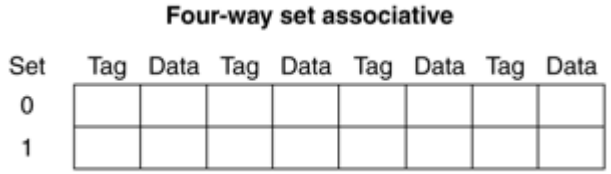
We create "sets" that are defined by the number of blocks in the set. For example, a "2-way Set Associative" cache has sets that have 2 blocks in them. Each block of memory can be found in one of two locations. To determine the number of sets, we take the full size of the cache (the total number of blocks it can contain) and divide it by the number of blocks in a set. To find the correct set for a block, we mod the block's address by the number of sets.

Review: Associativity

Suppose we have a total of 8 blocks. If we created a 2-way Set Associative cache, we would have 4 sets of 2 blocks:



If we created a 4-way Set Associative cache with the same number of total blocks, we would end up with 2 sets of 4 blocks:

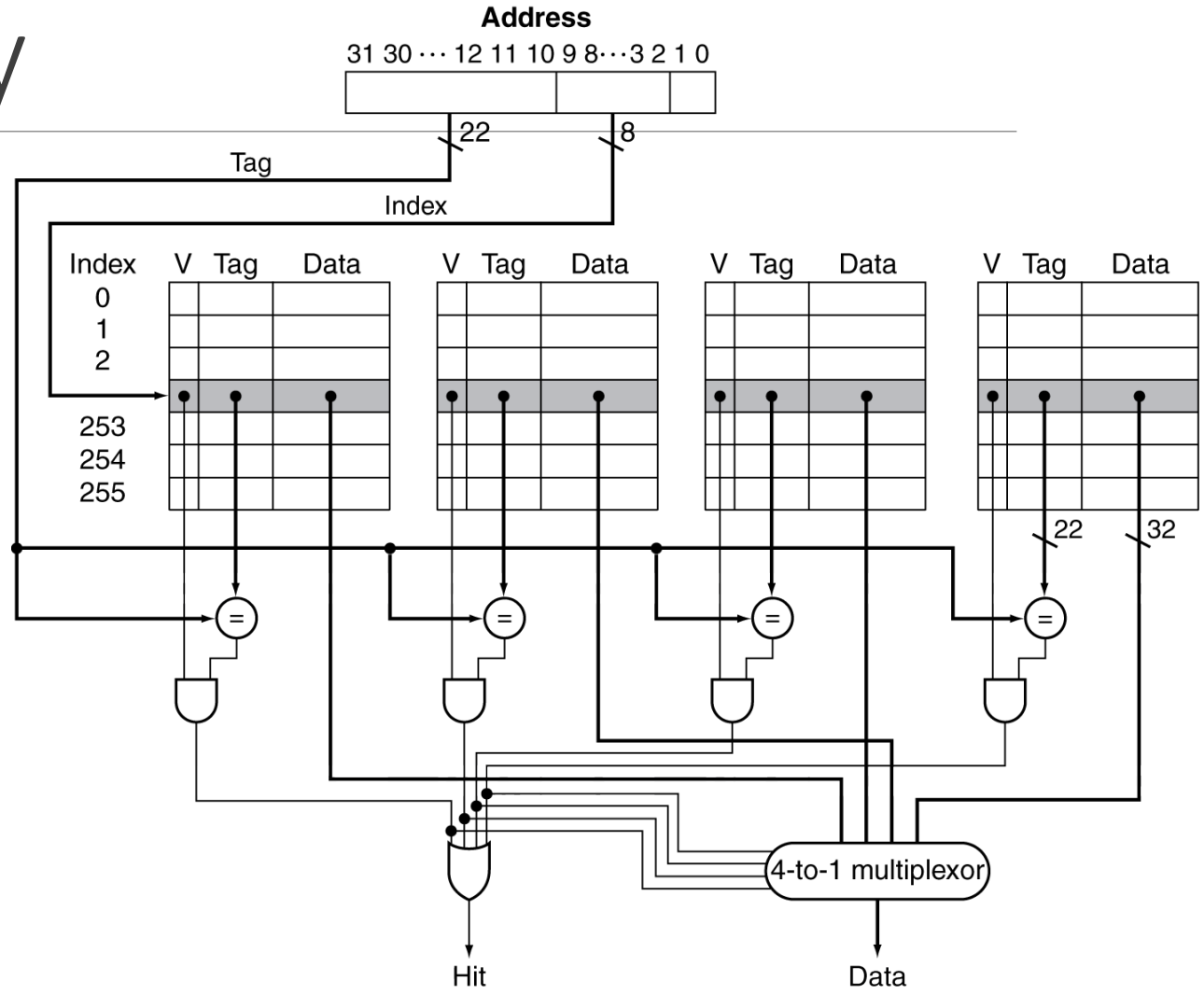


Review: Associativity

Increasing the associativity has a hardware cost: We need a comparator for each block in a set, an or gate to determine the hit, and an X-to-1 multiplexor where X is the degree of associativity.

This diagram represents a 4-way Set Associative cache that has a total of 1024 blocks of memory. It is organized into 256 sets of 4 blocks.

When we look for a piece of information, we will check all 4 blocks of the expected set simultaneously.



Review: Block Replacement Policies

With higher degree of associativity, we shouldn't have to replace blocks as often. When the set is full, however, and a new block needs to be stored we will have to choose which existing block to replace.

There are two strategies for block replacement:

- Least-recently used (LRU): Choose the one unused for the longest time
- Random: Gives approximately the same performance as LRU for degrees > 4

Example: Set-Associative Cache Reads

Given: Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 1:

To set up the cache, we need 16 blocks arranged into sets in such a way that each set has 2 blocks. This will give us 8 sets, with index values 0-7. Since the cache is initially empty, all of the valid bits are 0.

Solution 1:

Cache Index	Valid Bit	Tag	Contents
000	0		
	0		
001	0		
	0		
010	0		
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	0		
	0		
111	0		
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 2:

The first address we want to read from is 0. We would represent this as (0000 0000). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 000. Both blocks have individual valid bits, but both are 0. We know there is no information in either block of this cache index. We must retrieve a copy of Mem[0] from a lower level of memory and add it to one of the cache blocks at index 000. We can choose either block since both are empty. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 2:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	0		
001	0		
	0		
010	0		
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	0		
	0		
111	0		
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, **15**, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 3:

The next address is 15 or (0000 1111). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 111. Both blocks have individual valid bits, but both are 0. We know there is no information in either block of this cache index. We must retrieve a copy of Mem[15] from a lower level of memory and add it to one of the cache blocks at index 111. We can choose either block since both are empty. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 3:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	0		
001	0		
	0		
010	0		
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	0		
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, **2**, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 4:

The next address is 2 or (0000 0010). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index010. Both blocks have individual valid bits, but both are 0. We know there is no information in either block of this cache index. We must retrieve a copy of Mem[2] from a lower level of memory and add it to one of the cache blocks at index 010. We can choose either block since both are empty. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 4:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	0		
001	0		
	0		
010	1	00000	[2]
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	0		
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, **8**, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 5:

The next address is 8 or (0000 1000). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index000. One block has valid information, so we should first check to see if 8 is already in the cache. The tags for 8 and the valid block of the cache do not match (00000 vs 00001) so we must add 8 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[8] from a lower level of memory and add it to the currently empty cache block at index 000. We will set the valid bit of the this block to 1 to show there is information in the cache.

Solution 5:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	0		
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, **14**, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 6:

The next address is 14 or (0000 1110). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 110. Both blocks have individual valid bits, but both are 0. We know there is no information in either block of this cache index. We must retrieve a copy of Mem[14] from a lower level of memory and add it to one of the cache blocks at index 110. We can choose either block since both are empty. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 6:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, **15**, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 7:

The next address is 15 or (0000 1111). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 111. One block has valid information, so we should first check to see if 15 is already in the cache. The tags for 15 and the valid block of the cache match (both are 00001). This is a cache hit.

Solution 7:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	0		
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, **26**, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 8:

The next address is 26 or (0001 1010). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index010. One block has valid information, so we should first check to see if 26 is already in the cache. The tags for 26 and the valid block of the cache do not match (00000 vs 00011) so we must add 26 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[26] from a lower level of memory and add it to the currently empty cache block at index 010. We will set the valid bit of the this block to 1 to show there is information in the cache.

Solution 8:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00011	[26]
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, **2**, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 9:

The next address is 2 or (0000 0010). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 010. Both blocks have valid information, so we should first check to see if 2 is already in the cache. The tags for 2 and one of the valid blocks of the cache match (both are 00000). This is a cache hit.

Solution 9:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00011	[26]
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, **0**, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 10:

The next address is 0 or (0000 0000). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 000. Both blocks have valid information, so we should first check to see if 0 is already in the cache. The tags for 0 and one of the valid blocks of the cache match (both are 00000). This is a cache hit.

Solution 10:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00011	[26]
011	0		
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, **19**, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 11:

The next address is 19 or (0001 0011). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 011. Both blocks have valid bits that are equal to 0. We know there is no information in either block of this cache index. We must retrieve a copy of Mem[19] from a lower level of memory and add it to one of the cache blocks at index 011. We can choose either block since both are empty. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 11:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00011	[26]
011	1	00010	[19]
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	0		

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, **7**, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 12:

The next address is 7 or (0000 0111). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 111. One block has valid information, so we should first check to see if 7 is already in the cache. The tags for 7 and the valid block of the cache do not match (00000 vs 00001) so we must add 7 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[7] from a lower level of memory and add it to the currently empty cache block at index 111. We will set the valid bit of the this block to 1 to show there is information in the cache.

Solution 12:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00011	[26]
011	1	00010	[19]
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	1	00000	[7]

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, **10**, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 13:

The next address is 10 or (0000 1010). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 010. Both blocks have valid information, so we should first check to see if 10 is already in the cache. The tag for 10 does not match either tag in the set (00001, 00000, and 00011, respectively). We must add 10 to the cache. Neither block is invalid so we must choose the least recently used block to replace. Looking back through the sequence of reads, 2 was used more recently than 26 so we will replace 26. We will retrieve a copy of Mem[10] from a lower level of memory and add it to the block currently occupied by 26, updating the tag. The valid bit does not change.

Solution 13:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00001	[10]
011	1	00010	[19]
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	1	00000	[7]

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, **8**, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 14:

The next address is 8 or (0000 1000). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 000. Both blocks have valid information, so we should first check to see if 8 is already in the cache. The tags for 8 and one of the valid blocks of the cache match (both are 00001). This is a cache hit.

Solution 14:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00001	[10]
011	1	00010	[19]
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	1	00000	[7]

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, **14**, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 15:

The next address is 14 or (0000 1110). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index 110. One block has valid information, so we should first check to see if 14 is already in the cache. The tags for 14 and the valid block match (both are 00001). This is a cache hit.

Solution 15:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00001	[10]
011	1	00010	[19]
	0		
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	1	00000	[7]

Example: Set-Associative Cache Reads (2-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, **11**

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 2-way Set Associative. Use 8 bits for any addresses.

Partial Credit 16:

The final address is 11 or (0000 1011). The lower 3 bits are the cache index. The upper 5 bits are the tag.

We will first look in the cache at index011. One block has valid information, so we should first check to see if 11 is already in the cache. The tags for 11 and the valid block of the cache do not match (00010 vs 00001) so we must add 11 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[11] from a lower level of memory and add it to the currently empty cache block at index 011. We will set the valid bit of the this block to 1 to show there is information in the cache.

Solution 16:

Cache Index	Valid Bit	Tag	Contents
000	1	00000	[0]
	1	00001	[8]
001	0		
	0		
010	1	00000	[2]
	1	00001	[10]
011	1	00010	[19]
	1	00001	[11]
100	0		
	0		
101	0		
	0		
110	1	00001	[14]
	0		
111	1	00001	[15]
	1	00000	[7]

Example: Set-Associative Cache Reads

Given: Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 1:

To set up the cache, we need 16 blocks arranged into sets in such a way that each set has 4 blocks. This will give us 4 sets, with index values 0-3. Since the cache is initially empty, all of the valid bits are 0.

Solution 1:

Cache Index	Valid Bit	Tag	Contents
00	0		
	0		
	0		
	0		
01	0		
	0		
	0		
	0		
10	0		
	0		
	0		
	0		
11	0		
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 2:

The first address we want to read from is 0. We would represent this as (0000 0000). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 00. All blocks have individual valid bits and all are 0. We know there is no information in any block of this cache index. We must retrieve a copy of Mem[0] from a lower level of memory and add it to one of the cache blocks at index 00. We can choose any of the available blocks.
We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 2:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	0		
	0		
	0		
01	0		
	0		
	0		
	0		
10	0		
	0		
	0		
	0		
11	0		
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, **15**, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 3:

The next address we want to read from is 15. We would represent this as (0000 1111). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 11. All blocks have individual valid bits and all are 0. We know there is no information in any block of this cache index. We must retrieve a copy of Mem[15] from a lower level of memory and add it to one of the cache blocks at index 11. We can choose any of the available blocks.
We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 3:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	0		
	0		
	0		
01	0		
	0		
	0		
	0		
10	0		
	0		
	0		
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, **2**, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 4:

The next address we want to read from is 2. We would represent this as (0000 0010). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 10. All blocks have individual valid bits and all are 0. We know there is no information in any block of this cache index. We must retrieve a copy of Mem[2] from a lower level of memory and add it to one of the cache blocks at index 10. We can choose any of the available blocks.
We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 4:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	0		
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	0		
	0		
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, **8**, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 5:

The next address we want to read from is 8. We would represent this as (0000 1000). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 00. One block has valid information, so we should first check to see if 8 is already in the cache. The tags for 8 and the valid block of the cache do not match (000000 vs 000010) so we must add 8 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[8] from a lower level of memory and add it to a currently empty cache block at index 00. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 5:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	0		
	0		
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, **14**, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 6:

The next address we want to read from is 14. We would represent this as (0000 1110). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 10. One block has valid information, so we should first check to see if 14 is already in the cache. The tags for 14 and the valid block of the cache do not match (000000 vs 000011) so we must add 14 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[14] from a lower level of memory and add it to a currently empty cache block at index 10. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 6:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	0		
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, **15**, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 7:

The next address we want to read from is 15. We would represent this as (0000 1111). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 11. One block has valid information, so we should first check to see if 15 is already in the cache. The tags for 15 and the valid block of the cache match (both are 000011). This is a cache hit.

Solution 7:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	0		
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, **26**, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 8:

The next address we want to read from is 26. We would represent this as (0011 0010). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 10. Two blocks have valid information, so we should first check to see if 26 is already in the cache. The tags for 26 and the valid blocks of the cache do not match (000000 vs 000010 vs 001100) so we must add 26 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[26] from a lower level of memory and add it to a currently empty cache block at index 10. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 8:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, **2**, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 9:

The next address we want to read from is 2. We would represent this as (0000 0010). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 10. Three blocks have valid information, so we should first check to see if 2 is already in the cache. The tags for 2 and one of the valid blocks of the cache match (both are 000000). This is a cache hit.

Solution 9:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, **0**, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 10:

The next address we want to read from is 0. We would represent this as (0000 0000). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 00. Two blocks have valid information, so we should first check to see if 0 is already in the cache. The tags for 0 and one of the valid blocks of the cache match (both are 000000). This is a cache hit.

Solution 10:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	0		
11	1	000011	[15]
	0		
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, **19**, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 11:

The next address we want to read from is 19. We would represent this as (0001 0011). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 11. One block has valid information, so we should first check to see if 19 is already in the cache. The tags for 19 and the valid block of the cache do not match (000011 vs 000100) so we must add 19 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[19] from a lower level of memory and add it to a currently empty cache block at index 11. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 11:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	0		
11	1	000011	[15]
	1	000100	[19]
	0		
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, **7**, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 12:

The next address we want to read from is 7. We would represent this as (0000 0111). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 11. Two blocks have valid information, so we should first check to see if 7 is already in the cache. The tags for 7 and the valid blocks of the cache do not match (000011 vs 000100 vs 000001) so we must add 7 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[7] from a lower level of memory and add it to a currently empty cache block at index 11. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 12:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	0		
11	1	000011	[15]
	1	000100	[19]
	1	000001	[7]
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, **10**, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 13:

The next address we want to read from is 10. We would represent this as (0000 1010). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 10. Three blocks have valid information, so we should first check to see if 10 is already in the cache. The tags for 10 and the valid blocks of the cache do not match (000011 vs 000100 vs 000001 vs 000010) so we must add 10 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[10] from a lower level of memory and add it to the final empty cache block at index 10. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 13:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	1	000010	[10]
11	1	000011	[15]
	1	000100	[19]
	1	000001	[7]
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, **8**, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 14:

The next address we want to read from is 8. We would represent this as (0000 1000). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 00. Two blocks have valid information, so we should first check to see if 8 is already in the cache. The tags for 8 and one of the valid blocks of the cache match (both are 000010). This is a cache hit.

Solution 14:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	1	000010	[10]
11	1	000011	[15]
	1	000100	[19]
	1	000001	[7]
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, **14**, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 15:

The next address we want to read from is 14. We would represent this as (0000 1110). The lower 2 bits are the cache index. The upper 6 bits are the tag.

We will first look in the cache at index 10. All blocks have valid information, so we should first check to see if 14 is already in the cache. The tags for 14 and one of the valid blocks of the cache match (both are 000011). This is a cache hit.

Solution 15:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	1	000010	[10]
11	1	000011	[15]
	1	000100	[19]
	1	000001	[7]
	0		

Example: Set-Associative Cache Reads (4-way)

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, **11**

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 16:

The final address we want to read from is 11. We would represent this as (0000 1011). The lower 2 bits are the cache index. The upper 6 bits are the tag.

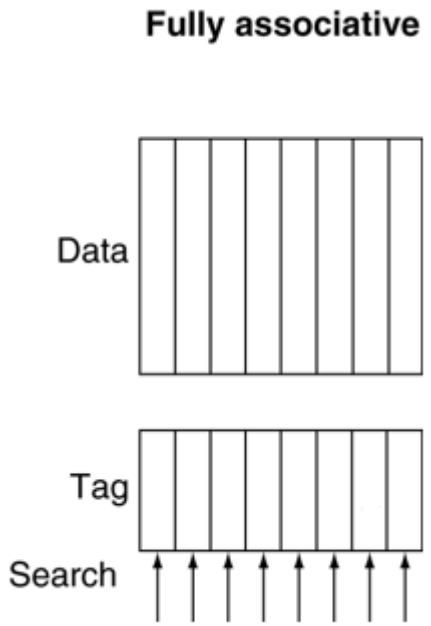
We will first look in the cache at index 11. Three blocks have valid information, so we should first check to see if 11 is already in the cache. The tags for 11 and the valid blocks of the cache do not match (000011 vs 000100 vs 000001 vs 000010) so we must add 11 to the cache. We prefer to overwrite an invalid block rather than a valid block, so we will retrieve a copy of Mem[11] from a lower level of memory and add it to the final empty cache block at index 10. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 16:

Cache Index	Valid Bit	Tag	Contents
00	1	000000	[0]
	1	000010	[8]
	0		
	0		
01	0		
	0		
	0		
	0		
10	1	000000	[2]
	1	000011	[14]
	1	001100	[26]
	1	000010	[10]
11	1	000011	[15]
	1	000100	[19]
	1	000001	[7]
	1	000010	[11]

Review: Associativity

When the degree of associativity is equal to the number of blocks in the cache, we say that the cache is Fully Associative. Any piece of information can be found in any block. To look for data we need to search every block simultaneously.



Example: Fully-Associative Cache Reads

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is Fully Associative. Use 8 bits for any addresses.

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 1:

We can think of a fully associative cache as one with a single, large set that contains 16 blocks. Any given data can go anywhere within the set. All blocks start with the valid bit initialized to 0 to indicate there is no valid information.

Solution 1:

Cache Index	Valid Bit	Tag	Contents
0	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		
	0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 2:

The first address we want to read from is 0. We would represent this as (0000 0000). There are no index bits in a fully associative cache. All bits would represent the tag.

To check a fully associative cache, we need to check all locations for validity. All blocks have individual valid bits and all are 0. We know there is no information in any block of this cache. We must retrieve a copy of Mem[0] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 2:

Valid Bit	Tag	Contents
1	0000 0000	[0]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, **15**, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 3:

The next address we want to read from is 15. We would represent this as (0000 1111). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There is currently one valid block, but the tag for it and the tag for 15 do not match. We must retrieve a copy of Mem[15] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 3:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, **2**, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 4:

The next address we want to read from is 2. We would represent this as (0000 0010). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently two valid blocks, but the tags for them and the tag for 2 do not match. We must retrieve a copy of Mem[2] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 4:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, **8**, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 5:

The next address we want to read from is 8. We would represent this as (0000 1000). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently three valid blocks, but the tags for them and the tag for 8 do not match. We must retrieve a copy of Mem[8] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 5:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, **14**, 15, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 6:

The next address we want to read from is 14. We would represent this as (0000 1110). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently four valid blocks, but the tags for them and the tag for 14 do not match. We must retrieve a copy of Mem[14] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 6:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, **15**, 26, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 7:

The next address we want to read from is 15. We would represent this as (0000 1111). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently five valid blocks, so we would compare the tag of 15 with the tags from all of them to see that there is a match. This is a cache hit.

Solution 7:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, **26**, 2, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 8:

The next address we want to read from is 26. We would represent this as (0011 0010). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently five valid blocks, but the tags for them and the tag for 26 do not match. We must retrieve a copy of Mem[26] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 8:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, **2**, 0, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 9:

The next address we want to read from is 2. We would represent this as (0000 0010). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently five valid blocks, so we would compare the tag of 2 with the tags from all of them to see that there is a match. This is a cache hit.

Solution 9:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, **0**, 19, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 10:

The next address we want to read from is 0. We would represent this as (0000 0000). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently five valid blocks, so we would compare the tag of 0 with the tags from all of them to see that there is a match. This is a cache hit.

Solution 10:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
0		
0		
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, **19**, 7, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 11:

The next address we want to read from is 19. We would represent this as (0001 0011). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently six valid blocks, but the tags for them and the tag for 19 do not match. We must retrieve a copy of Mem[19] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 11:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
1	0001 0011	[19]
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, **7**, 10, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 12:

The next address we want to read from is 7. We would represent this as (0000 0111). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently seven valid blocks, but the tags for them and the tag for 7 do not match. We must retrieve a copy of Mem[7] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 12:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
1	0001 0011	[19]
1	0000 0111	[7]
0		
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, **10**, 8, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 13:

The next address we want to read from is 10. We would represent this as (0000 1010). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently eight valid blocks, but the tags for them and the tag for 10 do not match. We must retrieve a copy of Mem[10] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 13:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
1	0001 0011	[19]
1	0000 0111	[7]
1	0000 1010	[10]
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, **8**, 14, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 14:

The next address we want to read from is 8. We would represent this as (0000 1000). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently five valid blocks, so we would compare the tag of 8 with the tags from all of them to see that there is a match. This is a cache hit.

Solution 14:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
1	0001 0011	[19]
1	0000 0111	[7]
1	0000 1010	[10]
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, **14**, 11

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 15:

The next address we want to read from is 14. We would represent this as (0000 1110). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently five valid blocks, so we would compare the tag of 14 with the tags from all of them to see that there is a match. This is a cache hit.

Solution 15:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
1	0001 0011	[19]
1	0000 0111	[7]
1	0000 1010	[10]
0		
0		
0		
0		
0		
0		
0		

Example: Fully Associative Cache Reads

Given:

Suppose we have a 16 block cache. Each block of the cache is one word wide. When a given program is executed, the processor reads data from the following sequence of decimal addresses:

0, 15, 2, 8, 14, 15, 26, 2, 0, 19, 7, 10, 8, 14, **11**

The content at address 0 can be shown as [0]. Assume the cache is initially empty and that the replacement policy is LRU. Show the contents of the cache at the end of the above reading operations if the cache is 4-way Set Associative. Use 8 bits for any addresses.

Partial Credit 16:

The next address we want to read from is 11. We would represent this as (0000 1011). There are no index bits in a fully associative cache. All bits represent the tag.

To check a fully associative cache, we need to check all locations for validity. There are currently nine valid blocks, but the tags for them and the tag for 11 do not match. We must retrieve a copy of Mem[11] from a lower level of memory and add it to one of the cache blocks. We can choose any of the available blocks. We will set the valid bit of the chosen block to 1 to show there is information in the cache.

Solution 16:

Valid Bit	Tag	Contents
1	0000 0000	[0]
1	0000 1111	[15]
1	0000 0010	[2]
1	0000 1000	[8]
1	0000 1110	[14]
1	0011 0010	[26]
1	0001 0011	[19]
1	0000 0111	[7]
1	0000 1010	[10]
1	0000 1011	[11]
0		
0		
0		
0		
0		
0		

Example Follow-Up: Other Common Cache Questions

Some other common cache questions include:

- Record the number of hits. / What is the hit ratio?
- Record the number of conflict or capacity misses. / What is the miss ratio?
- What is the miss ratio if you include cold start misses?

	Direct-Mapped			2-way Set Associative			4-way Set Associative			Fully Associative		
	Hits	Misses	Misses+	Hits	Misses	Misses+	Hits	Misses	Misses+	Hits	Misses	Misses+
#	5	1	10	5	1	10	5	0	10	5	0	10
ratio	1/3	1/15	2/3	1/3	1/15	2/3	1/3	0	2/3	1/3	0	2/3

Review: Write Policies

Reads are easier to manage than writes, because reads do not modify memory. When writing to memory, we need to consider ways to ensure all levels of memory remain consistent. There are two primary write policies:

Write-Through: whenever you write to the cache, also write to all other levels of memory.

- Pro: memory remains consistent at all times
- Con: requires a lot of time to update all levels, can be mitigated but not fixed with write buffers

Write-Back: whenever you write to the cache, assert a bit that indicates the block has been modified. When a block is removed from the cache, we now need to check if it has been modified. Any modified ("dirty") blocks have to be written down to the next level of memory.

- Pro: only need to write to one location
- Con: each block requires an additional bit to track modifications.