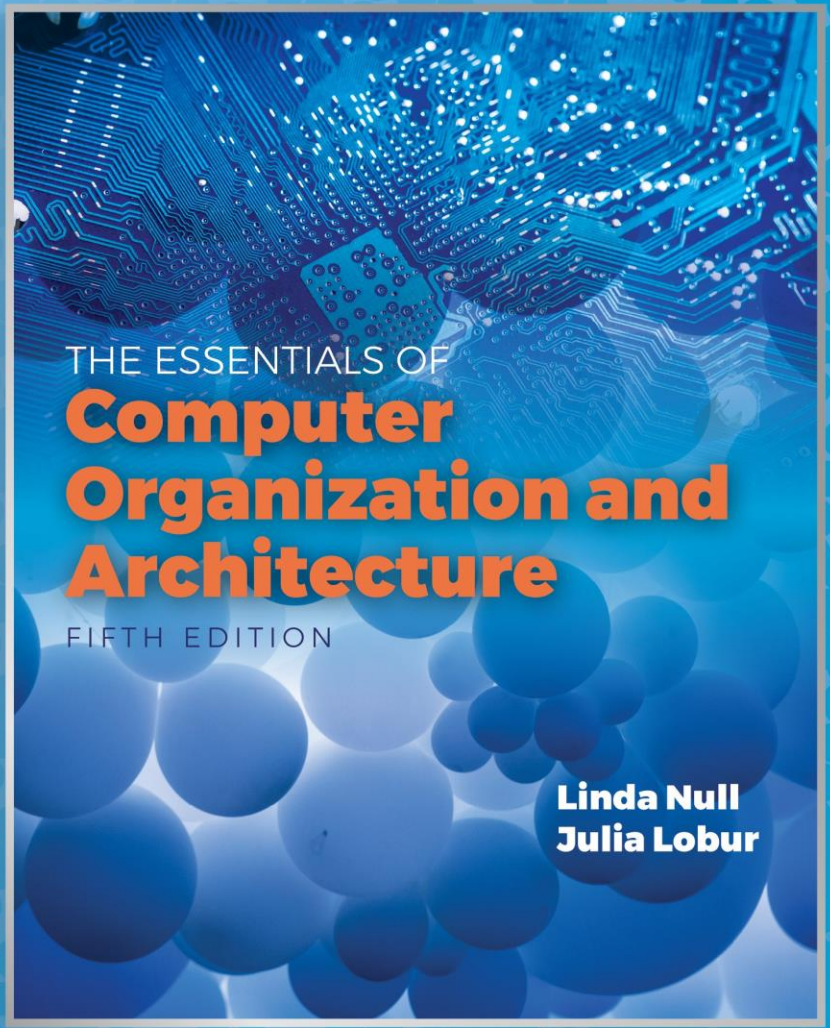
This is the third lecture of Chapter 6

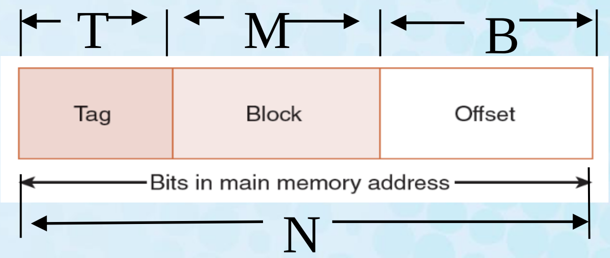
Chapter 6

Memory (C)



Quick review of last lecture

- Cache Placement
  - Direct Mapping
    - Memory Address format
    - Memory_size = Block_size * #_of_blocks

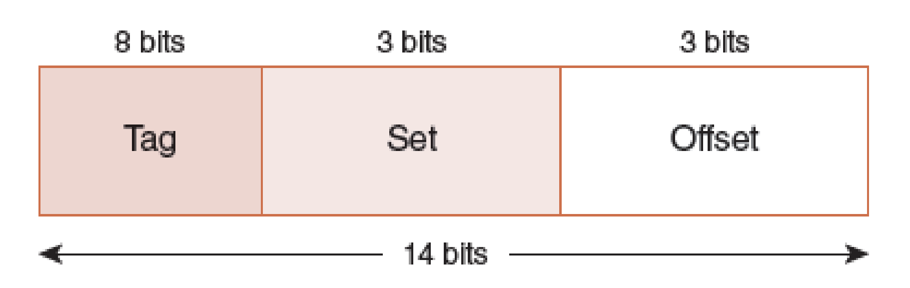


  - Fully associative mapping
    - Memory Address format
    - Cache replacement

  - N-way set associative mapping

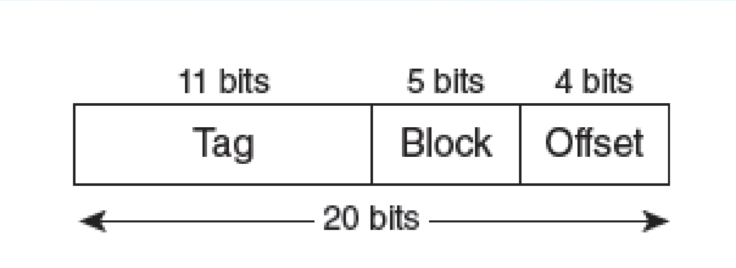
6.4 Cache Memory (21 of 45)

- Example 6.5: Suppose we are using 2-way set associative mapping with a byte-addressable main memory of 2^{14} bytes and a cache with 16 blocks, where each block contains 8 bytes.
  - Cache has a total of 16 blocks, and each set has 2 blocks, then there are 8 sets in cache.
  - Thus, the set field is 3 bits, the offset field is 3 bits, and the tag field is 8 bits.



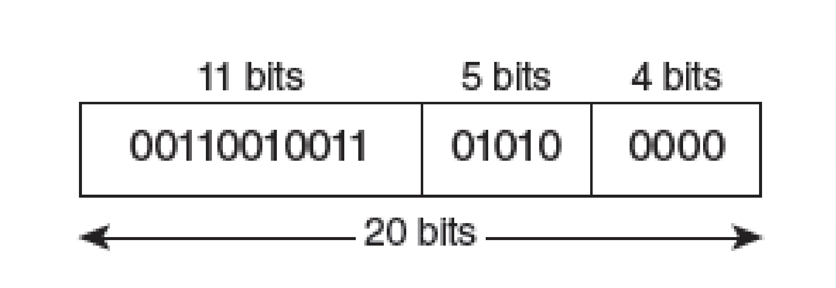
6.4 Cache Memory (22 of 45)

- Example 6.6: Suppose a byte-addressable memory contains 1MB and cache consists of 32 blocks, where each block contains 16 bytes. Using direct mapping, fully associative mapping, and a 4-way set associative mapping, determine where the main memory address 0x326A0 maps to in cache.
  - First note that a main memory address has 20 bits. The main memory address for direct mapped cache is shown below.



6.4 Cache Memory (23 of 45)

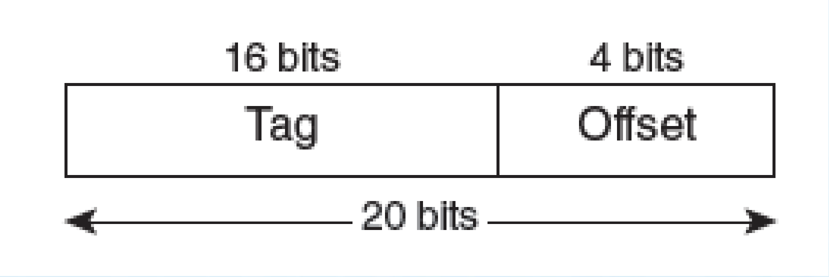
- Example 6.6:
  - If we represent our main memory address 0x326A0 in binary and place the bits into the format, we get:



  - So this address maps to cache block 01010 (or block 10).

6.4 Cache Memory (24 of 45)

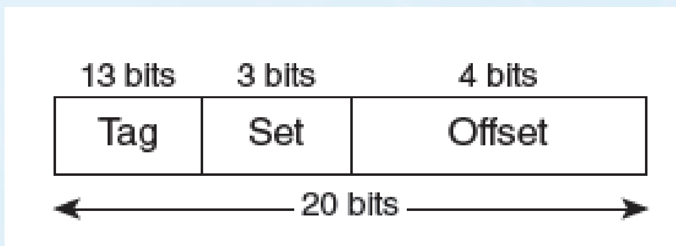
- Example 6.6: Cont'd.
  - If we are using fully associative cache, we have:



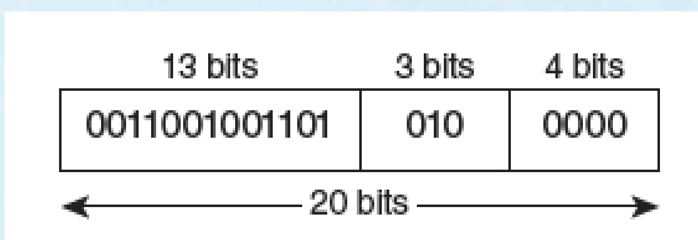
  - But because it is fully associative, it could map anywhere.

6.4 Cache Memory (25 of 45)

- Example 6.6: Cont'd.
  - If we are using 4-way set associative cache, we have:

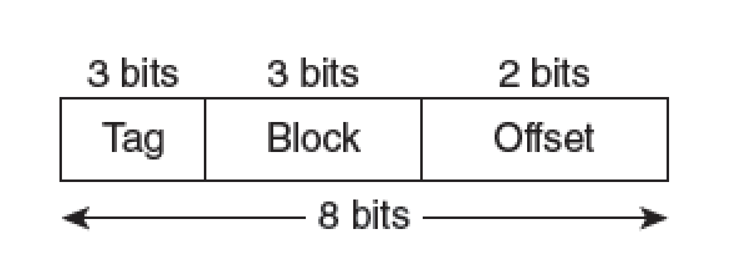


  - If we divide the main memory address into these fields, we get:



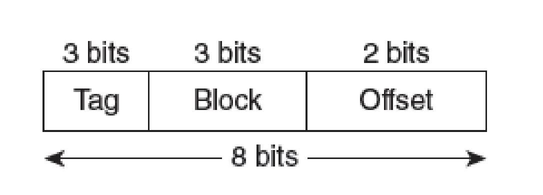
6.4 Cache Memory (26 of 45)

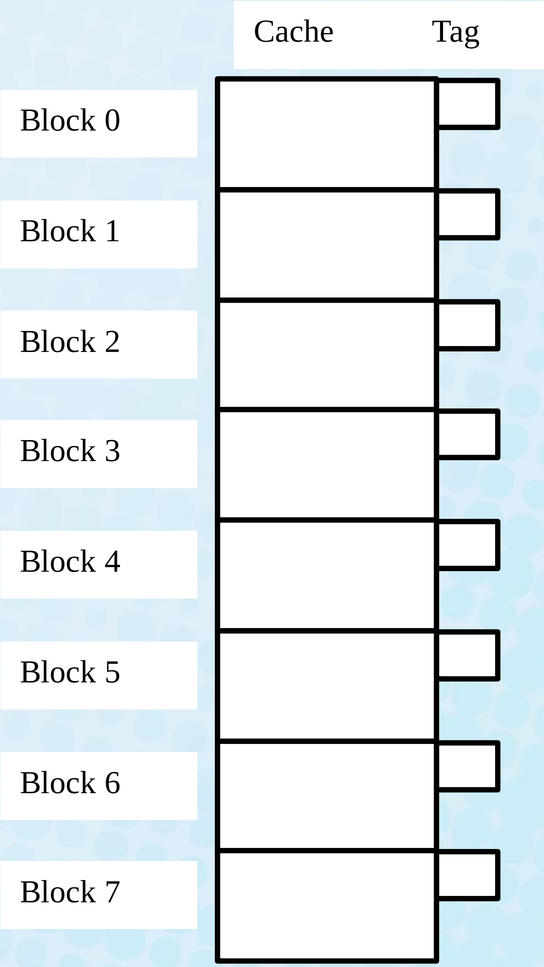
- Example 6.7: A byte-addressable computer with an 8-block cache of 4 bytes each. Assuming each memory address has 8 bits and cache initially is empty.
- Trace memory accesses: 0x01, 0x04, 0x09, 0x05, 0x14, 0x21, and 0x01 for each mapping approach.
- The address format for direct mapped cache is:



Our trace is on the next slide.

Address Reference	Binary Address (divided into fields)	Hit or Miss
0x01	000 000 01	Miss
0x04	000 001 00	Miss
0x09	000 010 01	Miss
0x05	000 001 01	Hit
0x14	000 101 00	Miss
0x21	001 000 01	Miss
0x01	000 000 01	Miss



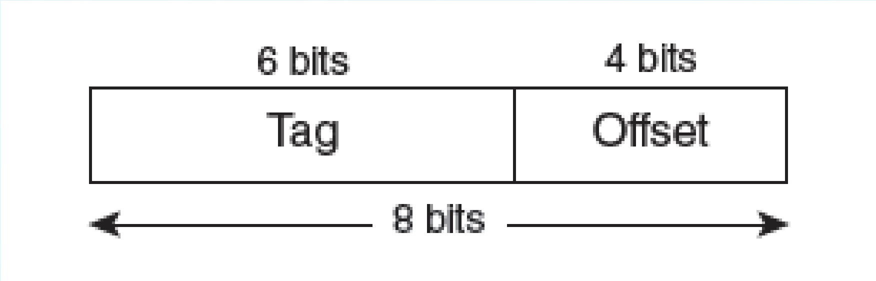


6.4 Cache Memory (27 of 45)

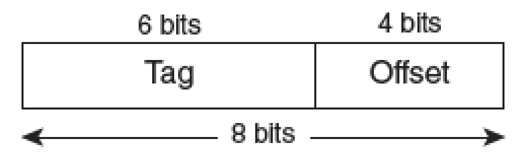
Address Reference	Binary Address (divided into fields)	Hit or Miss	Comments
0x01	000 000 01	Miss	If we check cache block 000 for the tag 000, we find that it is not there. So we copy the data from addresses 0x00, 0x01, 0x02, and 0x03 into cache block 0 and store the tag 000 for that block.
0x04	000 001 00	Miss	We check cache block 001 for the tag 000, and on finding it missing, we copy the data from addresses 0x04, 0x05, 0x06, and 0x07 into cache block 1 and store the tag 000 for that block.
0x09	000 010 01	Miss	A check of cache block 010 (2) for the tag 000 reveals a miss, so we copy the data from addresses 0x08, 0x09, 0x0A, and 0x0B into cache block 2 and store the tag 000 for that block.
0x05	000 001 01	Hit	We check cache block 001 for the tag 000, and we find it. We then use the offset value 01 to get the exact byte we need.
0x14	000 101 00	Miss	We check cache block 101 (5) for the tag 000, but it is not present. We copy addresses 0x14, 0x15, 0x16, and 0x17 to cache block 5 and store the tag 000 with that block.
0x21	001 000 01	Miss	We check cache block 000 for the tag 001; we find tag 000 (which means this is not the correct block), so we overwrite the existing contents of this cache block by copying the data from addresses 0x20, 0x21, 0x22, and 0x23 into cache block 0 and storing the tag 001.
0x01	000 000 01	Miss	Although we have already fetched the block that contains address 0x01 once, it was overwritten when we fetched the block containing address 0x21 (if we look at block 0 in cache, we can see that its tag is 001, not 000). Therefore, we must overwrite the contents of block 0 in cache with the data from addresses 0x00, 0x01, 0x02, and 0x03, and store a tag of 000.

6.4 Cache Memory (28 of 45)

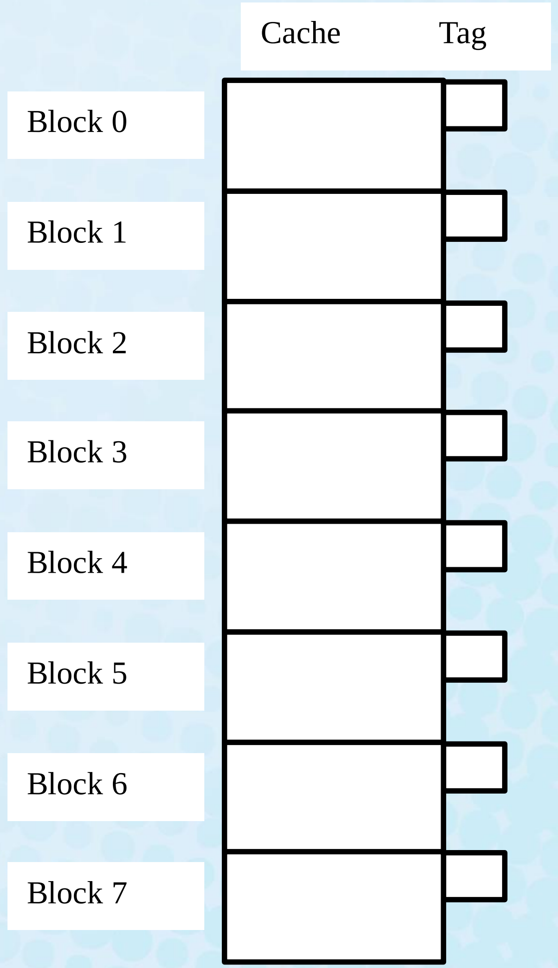
- Example 6.7: Cont'd. A byte-addressable computer with an 8-block cache of 4 bytes each, trace memory accesses: 0x01, 0x04, 0x09, 0x05, 0x14, 0x21, and 0x01 for each mapping approach.
- The address format for fully associative cache is:



Our trace is on the next slide.



Address Reference	Binary Address (divided into fields)	Hit or Miss
0x01	000000 01	Miss
0x04	000001 00	Miss
0x09	000010 01	Miss
0x05	000001 01	Hit
0x14	000101 00	Miss
0x21	001000 01	Miss
0x01	000000 01	Hit

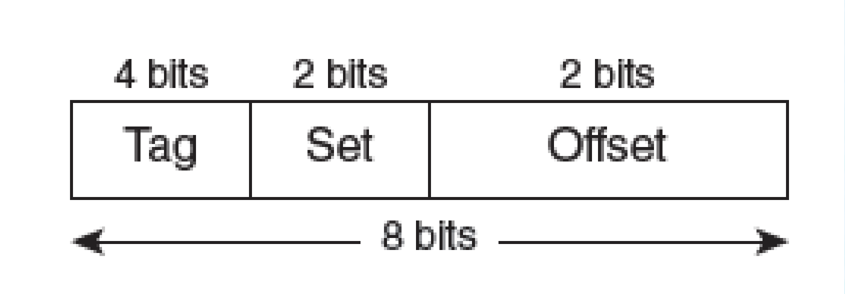


6.4 Cache Memory (29 of 45)

Address Reference	Binary Address (divided into fields)	Hit or Miss	Comments
0x01	000000 01	Miss	We search all of cache for the tag 000000, and we don't find it. So we copy the data from addresses 0x00, 0x01, 0x02, and 0x03 into cache block 0 and store the tag 000000 for that block.
0x04	000001 00	Miss	We search all of cache for the tag 000001, and on finding it missing, we copy the data from addresses 0x04, 0x05, 0x06, and 0x07 into cache block 1 and store the tag 000001 for that block.
0x09	000010 01	Miss	We don't find the tag 000010 in cache, so we copy the data from addresses 0x08, 0x09, 0x0A, and 0x0B into cache block 2 and store the tag 000010 for that block.
0x05	000001 01	Hit	We search all of cache for the tag 000001, and we find it stored with cache block 1. We then use the offset value 01 to get the exact byte we need.
0x14	000101 00	Miss	We search all of cache for the tag 000101, but it is not present. We copy addresses 0x14, 0x15, 0x16, and 0x17 to cache block 3 and store the tag 000101 with that block.
0x21	001000 01	Miss	We search all of cache for the tag 001000; we don't find it, so we copy the data from addresses 0x20, 0x21, 0x22, and 0x23 into cache block 4 and store the tag 001000.
0x01	000000 01	Hit	We search cache for the tag 000000 and find it with cache block 0. We use the offset of 1 to find the data we want.

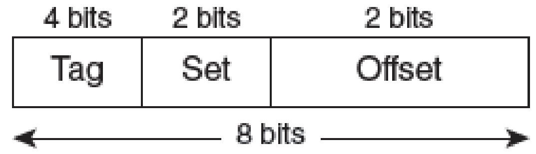
6.4 Cache Memory (30 of 45)

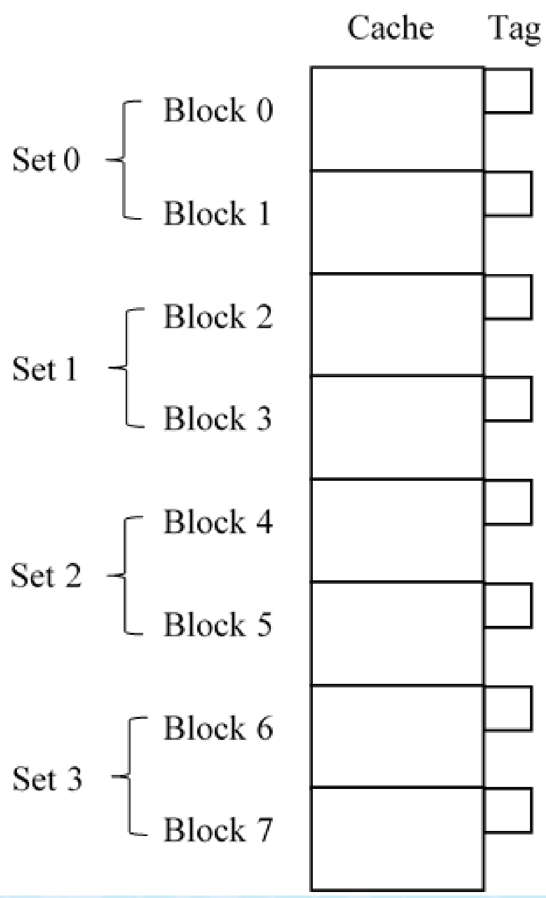
- EXAMPLE 6.7: Cont'd. A byte-addressable computer with an 8-block cache of 4 bytes each, trace memory accesses: 0x01, 0x04, 0x09, 0x05, 0x14, 0x21, and 0x01 for each mapping approach.
- The address format for 2-way set-associative cache is:



Our trace is on the next slide.

Address Reference	Binary Address (divided into fields)	Hit or Miss
0x01	0000 00 01	Miss
0x04	0000 01 00	Miss
0x09	0000 10 01	Miss
0x05	0000 01 01	Hit
0x14	0001 01 00	Miss
0x21	0010 00 01	Miss
0x01	0000 00 01	Hit





6.4 Cache Memory (31 of 45)

Address Reference	Binary Address (divided into fields)	Hit or Miss	Comments
0x01	0000 00 01	Miss	We search in set 0 of cache for a block with the tag 0000, and we find it is not there. So we copy the data from addresses 0x00, 0x01, 0x02, and 0x03 into set 0 (so now set 0 has one used block and one free block) and store the tag 0000 for that block. It does not matter which set we use; for consistency, we put the data in the first set.
0x04	0000 01 00	Miss	We search set 1 for a block with the tag 0000, and on finding it missing, we copy the data from addresses 0x04, 0x05, 0x06, and 0x07 into set and store the tag 0000 for that block.
0x09	0000 10 01	Miss	We search set 2 (10) for a block with the tag 0000, but we don't find one, so we copy the data from addresses 0x08, 0x09, 0x0A, and 0x0B into set 2 and store the tag 0000 for that block.
0x05	0000 01 01	Hit	We search set 1 for a block with the tag 0000, and we find it. We then use the offset value 01 within that block to get the exact byte we need.
0x14	0001 01 00	Miss	We search set 1 for a block with the tag 0001, but it is not present. We copy addresses 0x14, 0x15, 0x16, and 0x17 to set 1 and store the tag 0001 with that block. Note that set 1 is now full.
0x21	0010 00 01	Miss	We search cache set 0 for a block with the tag 0010; we don't find it, so we copy the data from addresses 0x20, 0x21, 0x22, and 0x23 into set 0 and store the tag 0010. Note that set 0 is now full.
0x01	0000 00 01	Hit	We search cache set 0 for a block with the tag 0000, and we find it. We use the offset of 1 within that block to find the data we want.