

L15-CS8421-Cache-P2

CS8421 Computing Systems Dr. Ken Hoganson

Class

Will

Start

Momentarily...

- Review of Cache Principles
- Example Cache Design
- Cache Homework Problem

- Memory performance has not kept pace with processor performance
- The Von-Neuman architecture requires multiple memory accesses for many instructions
- The use of pipelines (covered later) to increase the number of instructions processed per unit time, further increases the memory/bus bandwidth
- bandwidth is often used to talk about the communication requirements in terms of bits per second, in addition to its more traditional sense of a range of frequencies

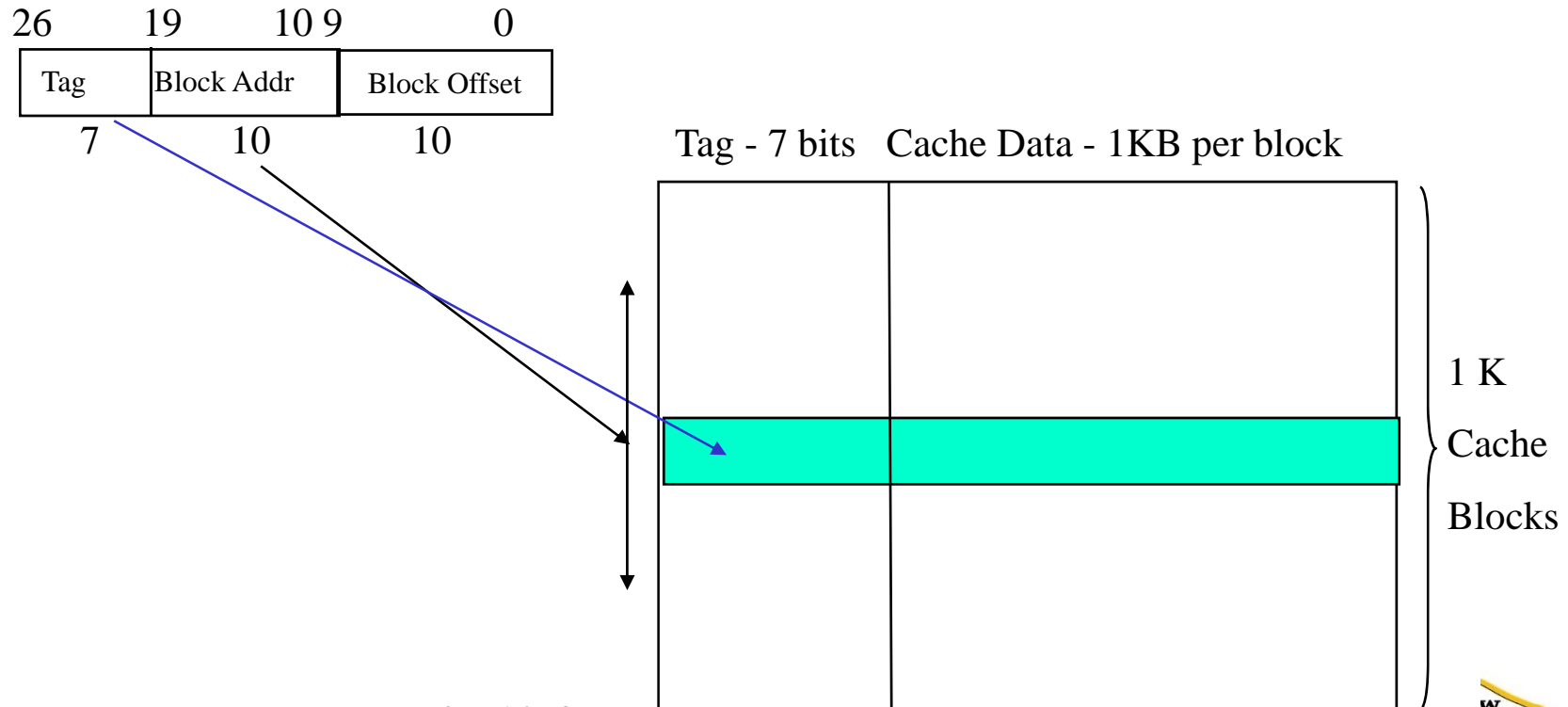
Memory access requires an address.

Caching works by examining the address generated by the CPU, to check if data is currently in the cache.

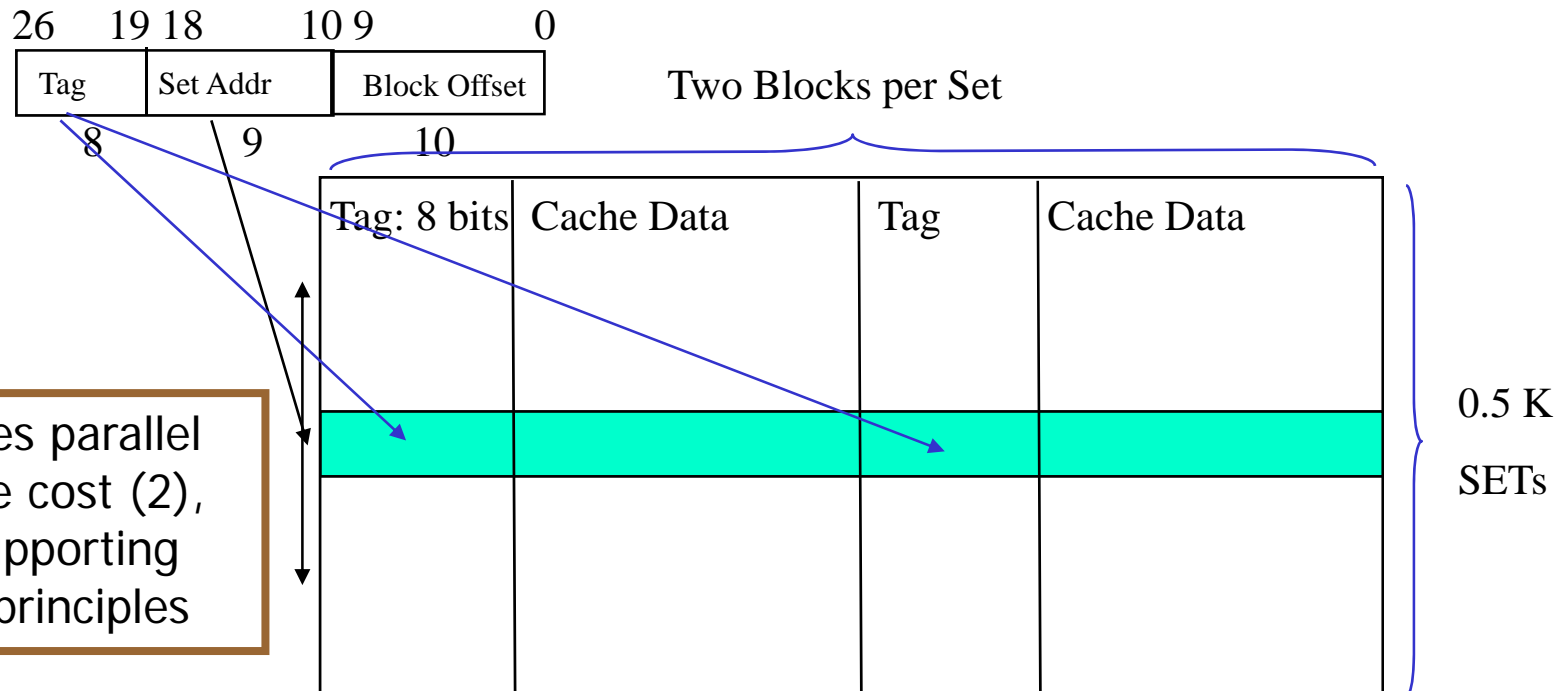
The address is subdivided into parts, in the process or decoding the address.

This address “decoding” is the key mechanism behind caching.

- Blocks can be cached in **one location only**
- Example: 128Mbytes of Mem (2^{27}), 1MB of cache (2^{20}), 1KB block size (2^{10}),
- Number of blocks in cache = $2^{20} / 2^{10} = 2^{10} = 1\text{K}$
- Fraction of memory in cache = $1/128 = 0.78\%$
- Must compare Tag to see which block is cached (many go to same location)
- 128 blocks map to the same cache location in this example
- Restrictive block location undermines locality principles



- Blocks can go anywhere within a particular **SET**
- A set is a grouping of blocks - **2 way** set associative means **2 blocks make a set**
- Example: 128Mbytes of Mem (2^{27}), 1MB of cache (2^{20}), 1KB block size (2^{10}),
- Number of blocks in cache = $2^{20} / 2^{10} = 2^{10} = 1\text{K}$, number of sets is $2^9 = 0.5\text{K}$
- Fraction of memory in cache = $1/128 = 0.78\%$
- Must compare both **tags** to see which block is cached (many go to same location)
- 128 blocks map to the same set, but two of those in set at a time



Writes of memory in a cached system pose an interesting problem

- Are the writes themselves cached, and written back later?
 - **WRITE-BACK**
 - A crash would lose data (caching disk in memory)
 - Best performance - do not wait for slow storage, write to cache is fast.
 - Write back to slower storage when system bandwidth is available

OR

- Write information through to the memory (or disk if caching disk)
 - **WRITE -THROUGH**
 - Slower, must wait for slower memory (or wait for disk)
 - Safer, changed data goes to disk - non-volatile

- Direct-Mapped
 - Not an issue, since each block can go only one place in the cache, and overwrites current contents (must force a write if using Write-Back)
- Fully-Associate
 - Which block to replace?
 - Perhaps Least Recently Used (LRU)
 - How to know which was LRU?
 - Track accesses to the block: reference bits, exponential averaging?
 - Random? - Actually works reasonably well!
- Set-Associative
 - Replace the LRU block within the set

Work out the following cache design: that is, do the analysis required, draw (and label) a diagram that shows the cache organization as was done in class:

128MB of main memory, 64KB of direct-mapped cache, 512byte cache block

Analysis: 128MB of main memory, 64KB of direct-mapped cache, 512byte cache block.

1. Express specs as power of two:
2. Number of blocks in the cache:
3. Number of blocks in memory:
4. Fraction of memory in the cache:
5. Block Offset bits:
6. Tag bits:
7. Number of blocks mapped to the same cache block:

Draw (and label) a diagram that shows the cache organization:
128MB of main memory, 64KB of direct-mapped cache,
512byte cache block

Work out the following cache designs: that is, do the analysis required, draw (and label) a diagram that shows the cache organization as was done in class:

1. 64MB of main memory, 128KB of direct-mapped cache, 512byte cache block
2. 256MB of main memory, 2MB of 4-way set-associative cache, 2KB cache block
3. 128MB of main memory, 2KB of fully-associative cache, 256 byte cache block

Cache Homework: Turn in for grade.

For a tiny memory system with 32 bytes of memory, 4 byte cache blocks of fully-associative cache, and 8 bytes of cache:

- How many bits required for the address?
- How many blocks in the cache?
- How many blocks in the memory?
- Draw a diagram of the cache memory system like the last lecture in class that shows:
 - Memory Addresses for each of the 32 bytes (in binary)
 - The cache structure
 - The memory address structure with the number of bits for each piece of the address
 - What is compared to what when checking for an address in the cache.
- Now populate your memory with the lower case letters of the alphabet, followed by digits, both in ascending order. Start the population with the first (lowest address) memory location.
- Now load the cache with the first and fourth memory blocks.
- The CPU generates address 01101. Is it in the cache? Explain what the CPU does to determine this.
- If the memory block is currently in the cache, what character is found at the above address?

**End
Of
Today's
Lecture.**

