Cache Memories

Lecture 15

Memory "Access method"

- Based on the hardware implementation of the storage device
- Four types
  - Sequential
  - Direct
  - Random
  - Associative

Memory “Access method”

- Sequential Access Method
  - Start at the beginning and read through in order
  - Access time depends on location of data and previous location
    - Example: tape
- Direct Access Method
  - Individual blocks have unique address
  - Access is by jumping to vicinity then performing a sequential search
  - Access time depends on location of data within "block" and previous location
    - Example: hard disk

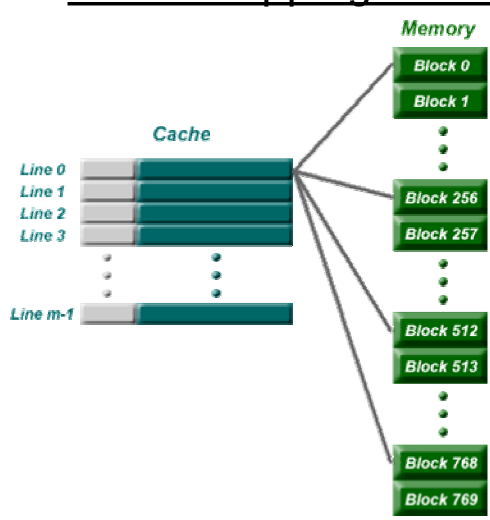
Memory “Access method”

- Random Access Method
  - Individual addresses identify locations exactly
  - Access time is consistent across all locations and is independent previous access
    - Example: RAM
- Associative Access Method
  - Addressing information must be stored with data in a general data location
  - A specific data element is located by a comparing desired address with address portion of stored elements
  - Access time is independent of location or previous access
    - Example: cache

Mapping Functions

- A mapping function is the method used to locate a memory address within a cache
  - It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Full Associative
  - Set Associative

Direct mapping Technique



Tag	bits identifying line in cache (block)	word id bits

Direct mapping Cache

- Direct mapped cache is not as expensive as other caches because:
  - The mapping scheme does not require any searching
    - Each main memory block has a specific location to which it maps in cache;
  - When a main memory address is converted to a cache address,
    - the CPU knows exactly where to look in the cache for that memory block by simply examining the bits in the block field.
      - Example: Dictionary

Fully Associative Cache

- Allowing a main memory block to be placed anywhere in cache.
  - The only way to find a block mapped this way is to search all of cache
    - requires the entire cache to be built from *associative memory* so it can be searched in parallel.
  - Compare the requested tag to *all* tags in cache to determine whether the desired data block is present in cache.
- The main memory address is partitioned into two pieces, the tag and the word.

Tag	word id bits

Fully Associative

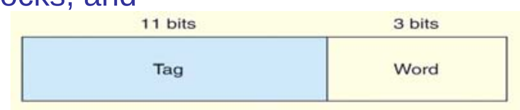
- Any block from main memory can map to any location in the cache
  - word id bits are used to identify which word in the block is needed,
  - The tag becomes all of the remaining bits.
- we could allow a block to go anywhere in cache.

- In this way, cache would have to fill up before any blocks are evicted.
  - This is how *fully associative* cache works.

Tag	word id bits

Fully Associative Cache

- For example,
  - Memory configuration with 2^{14} words,
  - A cache with 16 blocks, and
  - Blocks of 8 words,



- The word field is 3 bits, but the tag field is 11 bits.
  - This tag must be stored with each block in cache
  - When the cache is searched for a specific main memory block, the tag field of the main memory address is compared to all the valid tag fields in cache;
    - if a match is found, the block is found
    - If there is no match, we have a cache miss and the block must be transferred from main memory

Fully Associative Example

- Suppose a computer using fully associative cache has 216 words of main memory and a cache of 64 blocks, where each cache block contains 32 words.
  - a. How many blocks of main memory are there?
  - b. What is the format of a memory address as seen by the cache, i.e., what are the sizes of the tag and word fields?
  - c. To which cache block will the memory reference F8C9 map?
- *Ans.*
  - a. $2^{16}/2^{5} = 2^{11}$
  - b. 16 bit addresses with 11 bits in the tag field and 5 in the word field
  - c. Since it is associative cache, it can map anywhere

Problem with Direct & Fully Associative

- Owing to its speed and complexity, associative cache is very expensive.
- Direct mapping is inexpensive, it is very restrictive.
- To see how direct mapping limits cache usage:
  - suppose we are running a program on the architecture described in our class examples.
    - the program is using block 0, then block 16, then 0, then 16, and so on as it executes instructions.
    - which means the program would repeatedly throw out 0 to bring in 16, then throw out 16 to bring in 0,
      - Additional blocks in cache not being used
  - Fully associative cache remedies this problem by allowing a block from main memory to be placed anywhere.
    - However, it requires a larger tag to be stored with the block
    - requiring special hardware for searching of all blocks
      - more expensive cache

N-way set associative cache mapping

- A combination of Direct and Fully Associative Mapping approaches.
  - Scheme is similar to direct mapped cache,
    - The address is used to map the block to a certain cache location.
  - The important difference:
    - Instead of mapping to a single cache block, an address maps to a *set* of several cache blocks.
  - All sets in cache must be the same size.
- similar to the way in which fully associative cache works.
  - Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.

Set Associative Mapping

- The number of cache blocks per set in set associative cache varies according to overall system design.
  - For example, in a 2-way set associative cache, there are two cache blocks per set,
  - Each set contains two different memory blocks.
    - set 0 contains two blocks, one that is valid and holds the data A, B, C, . . . , and another that is not valid.

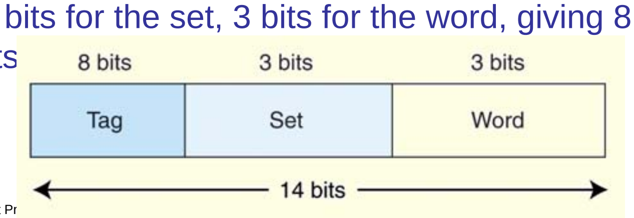
Set	Tag	Block 0 of set	Valid	Tag	Block 1 of set	Valid
0	00000000	Words A, B, C, . . .	1	-------------		0
1	11110101	Words L, M, N, . . .	1	-------------		0
2	-------------		0	10111011	P, Q, R, . . .	1
3	-------------		0	11111100	T, U, V, . . .	1

Set Associative cache

- In set associative cache mapping, a memory reference is divided into three fields:
  - Tag, set, and word.
- As with direct-mapped cache,
  - The word field chooses the word within the cache block, and
  - The tag field uniquely identifies the memory address.
- The set field determines the set to which the memory block maps.

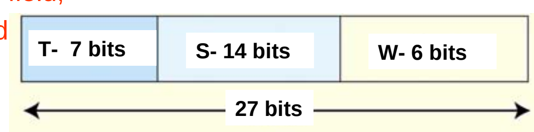
Set Associative Example

- Suppose we have a main memory of 2^{14} bytes.
  - This memory is mapped to a 2-way set associative cache having 16 blocks where each block contains 8 words.
  - Since this is a 2-way cache,
    - each set consists of 2 blocks, and there are 8 sets.
- If cache consists of a total of 16 blocks, and each set has 2 blocks, then there are 8 sets in cache.
- Thus, we need
  - 3 bits for the set, 3 bits for the word, giving 8 leftover bits



Example

- Suppose
  - A memory has 128M words. → 2^{27}
  - Blocks are 64 words in length and →2^{6}
  - The cache consists of 32K blocks. → 2^{15}
- Show the format for a main memory address assuming a 2-way set associative cache mapping scheme.
- *Ans.*
  - Each address has 27 bits, and
    - there are 7 in the tag field,
    - 14 in the set field and
    - 6 in the word field.



Next Class

- Block Replacement Policies