The Memory Hierarchy

Tom Kelliher, CS 220

Dec. 4, 2009

1 Administrivia

Announcements

Assignment

Read 5.4.

From Last Time

Introduction to pipelining.

Outline

1. Terminology.
2. Basics of caches.
3. Direct mapped caches.

Coming Up

Virtual memory.

2 Introduction

1. Memory is a hierarchy: registers, cache, main memory, disk, tape.

   How does cost, speed, size vary over this hierarchy?

2. Key idea: Trick the processor into believing it has a large, fast memory. How can we accomplish this?

3. Principle of locality allows us to keep a subset of the memory space high in the hierarchy, where it fits and where access is fast. Because, a program uses just a small part of its address space at any instant.

   (You know you need more RAM when you hear your disks seeking a lot.)

   Two types of locality:

   (a) Temporal locality.

   (b) Spatial locality.

   We concentrate upon two points in the memory hierarchy: cache/main memory and main memory/disk:

   (a) Cache/main memory: handled by the hardware. Not a part of the system architecture.

   (b) Main memory/disk: handled by the OS. The virtual memory system.

   (Actually, the register file can be thought of as a compiler/programmer-controlled cache.)

4. Terminology: block, hit, hit rate, miss, miss penalty. (Define.)

   Block sizes for caches, disks.

3 Basics of Caches

1. Think of memory being partitioned into blocks, called *lines* when in the cache.

2. Think of the cache as being partitioned into lines and lines being collected into sets, where the set size can be:

   (a) One line: direct mapped cache.

   (b) All lines: fully associative cache.

   (c) Between the first two, but a power of two: set-associative cache.

   A given memory block is always loaded into the same set.

   With a direct mapped cache, two blocks in the same set can't be in cache at the same time.

3. Think of a memory address being partitioned into two pieces: the *tag* and the *set selector* (low order bits)

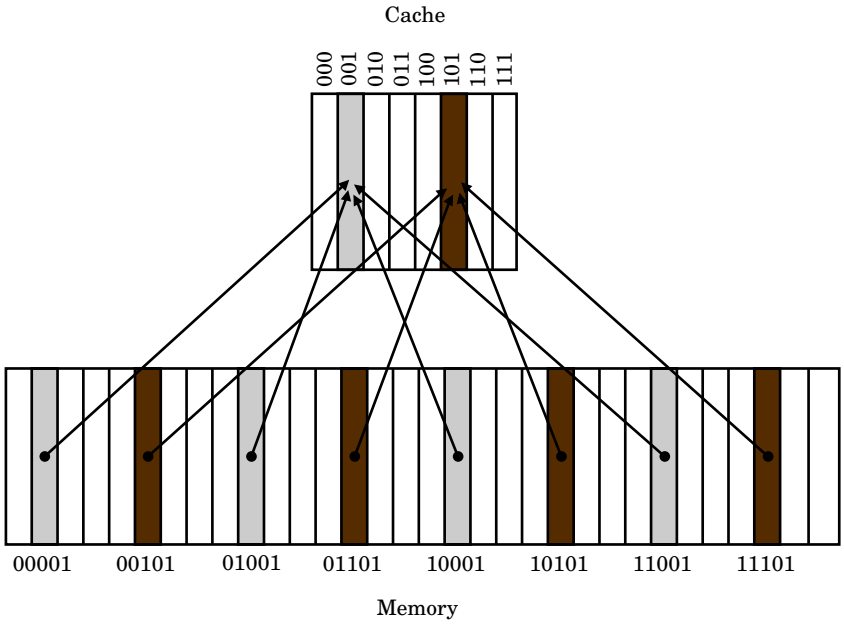
4. Basic parameters of cache design: size of cache (may include data, tag, valid bit), size of line, set size.

4 Direct Mapped Caches

Direct mapped cache:

> *A cache structured such that a memory block is associated with exactly one cache block, based upon the address of the memory block. The usual algorithm is:*
>
> cache address = memory address *modulo* number of cache blocks

By definition, multiple memory blocks map to the same cache block:



4.1 First Cache

Let's design our first cache. Parameters:

1. Direct mapped — a memory block maps to exactly one cache block.
2. Block size is one word.
3. Eight blocks.
4. Memory size is 64 words, byte addressable (always the case).

Questions:

1. How many bits for the cache?
2. Consider the memory trace: 22, 26, 22, 26, 16, 4, 16, 18, run on a cold cache. How many hits, misses? What's the hit rate? What are we missing from the cache design (tag bits).
3. Why don't we need to store the entire memory address?

4. What about a valid bit?

5. Now, how many bits for the cache?

4.2 Second Cache

Let's design another cache. Parameters:

1. Direct mapped.

2. Block size is four words. Why would we want a larger block size?

3. 4K blocks.

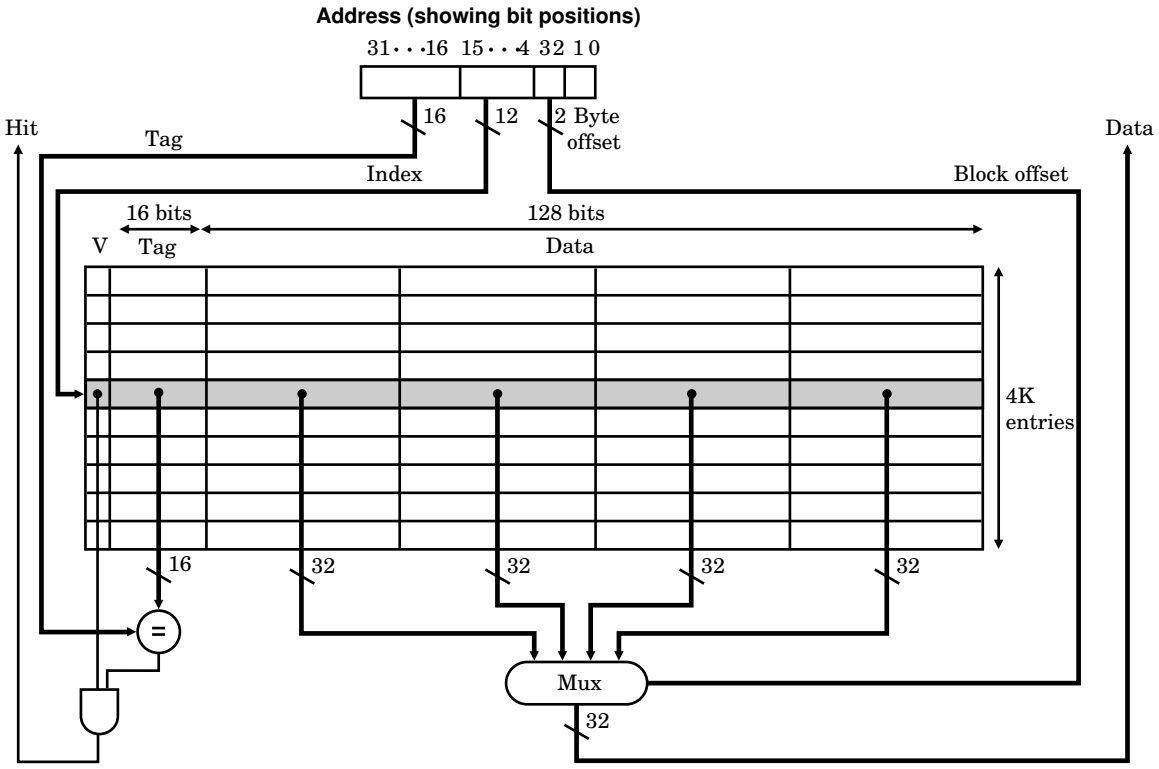
4. 32 bit memory space.

Questions:

1. How many bits/block? Total size of the cache?

2. Size of the data portion of the cache, in bytes? (This is the number you see quoted.)

Organization of the cache:


Address (showing bit positions)
31 . . . 16 15 . . . 4 3 2 1 0
16
12
2 Byte offset
Hit
Tag
Index
Block offset
Data
16 bits
128 bits
V
Tag
Data
4K entries
16
32
32
32
32
=
Mux
32