Lecture 12 – Cache Memory Systems

Karl R. Wilcox
Karl@cs.rhul.ac.uk

Objectives

- **In this lecture we will cover**
  - **Cache Memory Systems**
  - **The hardware for implementing cache memory**

(Diagrams from Patterson & Hennessy)

Recall:

- Cache systems (of all kinds) take advantage of:
  - Temporal locality
    - If something has been referenced recently it is likely to be referenced again soon
  - Spatial locality
    - If something has been referenced, things nearby (in the address space) are likely to be referenced soon
  - Memory technology
    - Fast = expensive, slow = cheap

CPU Memory Hierarchy

- SRAM on the same die as the CPU
  - 5ns / Relatively large area of silicon per bit / v. high cost
- Separate SRAM between the CPU and the bus
  - 20ns / additional chips / high cost
- DRAM, attached to the system bus
  - 60ns / less silicon per bit (higher density) / medium cost
- Magnetic disk etc. (mass storage)
  - 1s / extremely high density / very low cost

Caching Terms

- **Objective**
  - To make as much memory as is available on the cheapest medium, at the speed of fastest
- **Cache Hit**
  - Finding an item in the cache
- **Hit time**
  - Time to access an item in the cache (including search time)
- **Cache miss**
  - Not finding an item in the cache
- **Miss penalty**
  - Time to search cache and load missing item

Compiler Considerations

- **Although caches are in theory "invisible" to the programmer (and the compiler writer) programs need to be compiled in such a way as to maximise cache hits**
  - **I.e. need to demonstrate spatial and temporal locality**
  - **This will achieve best performance**

A Simple Memory Cache

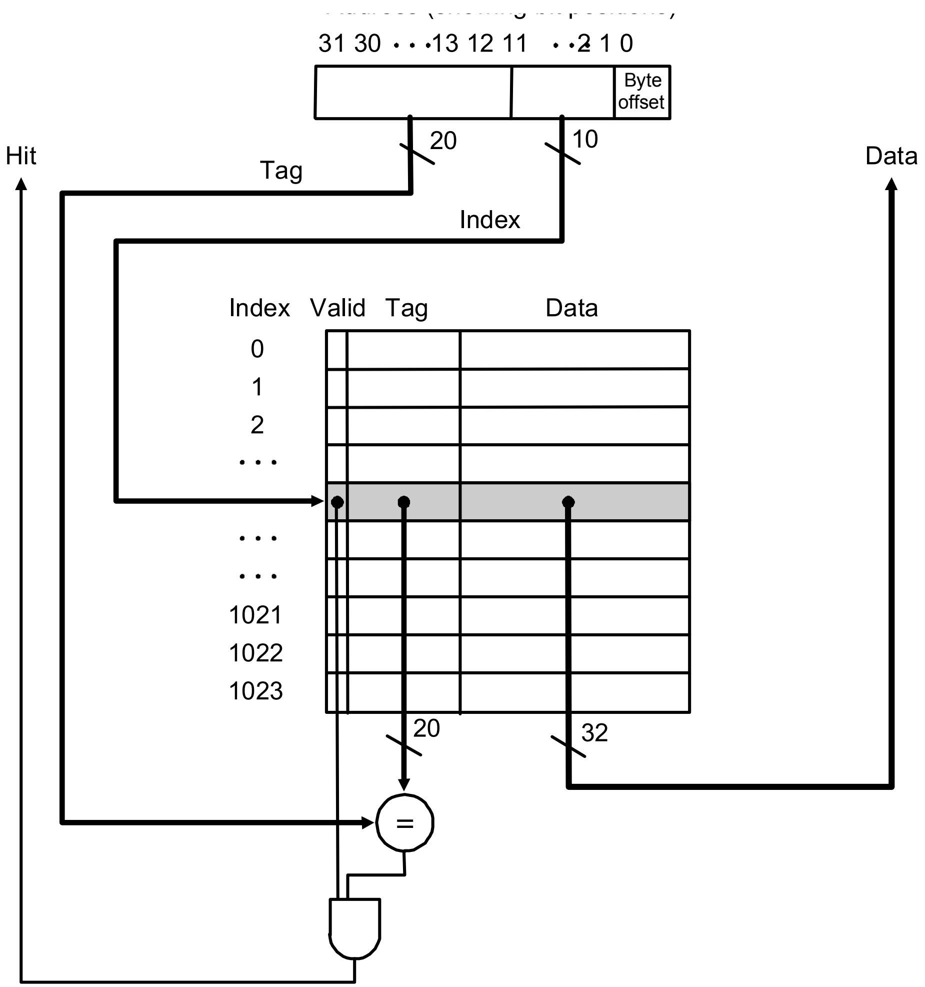
- A cache has *n* blocks of storage
  - *n* is usually a power of 2
  - The block may be as small as a word

- *Direct Mapping*
  - Each block goes into the storage location given be the block address modulus the number of blocks in cache
  - Equivalent to the lowest *m* bits of the address
  - The cache must also hold the high bits to check address
  - And it requires a “valid” bit to distinguish unused locations

Hardware for a Direct Mapped Cache

- **32 bit words**
- **1024, 1 word locations in cache**
- **Therefore**
  - **2 bits for byte offset**
  - **10 bits used for location**
  - **leaving 20 bits for tag**
  - **“Hit” logic needs 20 bit comparator + AND gate**

— **How much overhead?**



The CPU and Cache Misses

- We must fetch the missing block from the next level of the memory hierarchy

- What about the CPU in the meantime?
  - If this is an instruction miss we must stall the pipeline until the instruction is available
  - If it is a data miss we could look for other instructions to execute
    - May not be worth the extra hardware…?

One Cache or Two?

- Should we have one cache for both data and instructions?
- Or separate caches for each?

- A single cache gives a better hit rate
  - More flexibility

- Two caches (accessible simultaneously) increase the cache *bandwidth*
  - This usually gives better overall performance

What About Writing Data?

- Simplest approach – write all data to memory and cache
  - Known as *write-through*
- Both are always valid
- But memory writes are slow
- Could write to the cache but buffer the write to memory
  - Buffer depth can increase as required
- Alternatively, only update cache
  - Update memory when cache block is replaced
  - Better performance, but memory not always valid

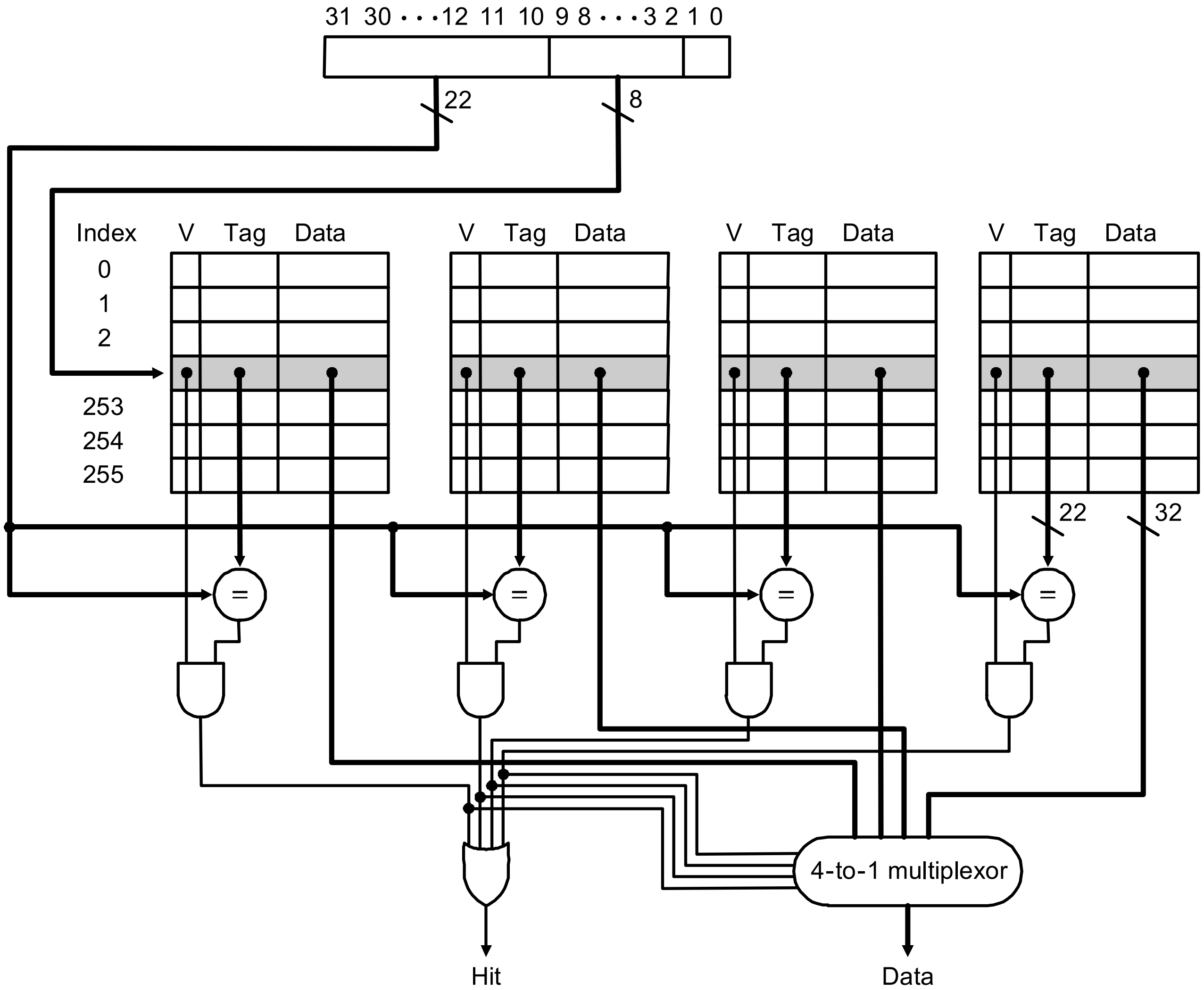
Bigger Blocks

- One word blocks only take advantage of temporal locality
- Using a larger block size and loading adjacent addresses can improve performance
- There should be a higher hit rate
- But there will be a higher miss penalty
- We can organise the memory to read a block of data at one time
  - Or at least more quickly – interleaving
- As always, there is a trade-off
  - No single answer

Other Cache Organisations

- The direct mapped cache can be wasteful
  - May leave unused locations
- Use an associative cache
  - Block can go in any location
  - Need to search all locations in parallel for target
- Compromise – the set associative cache
  - Address determines which set
  - Block can go anywhere in the set
  - Set is then searched in parallel
- Which block to replace
  - LRU is common
  - Other algorithms as discussed in CS263 Op Sys

Hardware for Set Associative Cache



Multi-level Caches

- Modern processors often have a second level cache
  - Separate SRAM chips “near” the processor
  - Typically 10 times as large as the primary cache
- First level cache design should minimise hit time
  - To give short cycle time for instructions
- Second level cache should focus on hit rate
  - To minimise long access times to main memory
  - Typically have bigger block sizes

Caching Vs Virtual Memory

- Caching works over the space of a few thousand instructions
  - I.e. "within" the quantum of time that a process gets on the processor
- Virtual Memory operates on entire processes
  - Which pages of a process should be in memory, ready to run
- Caching and virtual memory are independent
  - But both increase processor performance

Summary

- We have looked at simple caching schemes
- And the hardware to implement them
- Some cache extensions and improvements
  - Multi-level, set associative, write-back
- Real caches tend to be more sophisticated
  - But based on the same principles

Next Lecture

- Virtual memory systems
  - Implementing a Virtual Memory system
  - (Related to CS263, Operating Systems)
    - Not concerned with page replacement algorithms
    - Interested with actual hardware implementation of virtual memory