Lecture 12: Memory hierarchy & caches

- How a modern memory subsystem is built
- Memory hierarchy – locality of references
- Cache memory – introduction

Memory requirements

- Programmer's view of memory
  - Large
  - Flat (uniform access)
  - Fast
- This is impossible with any known memory technology
  - Even supercomputers cannot have it
  - The main problem is size: *large memory is slow*

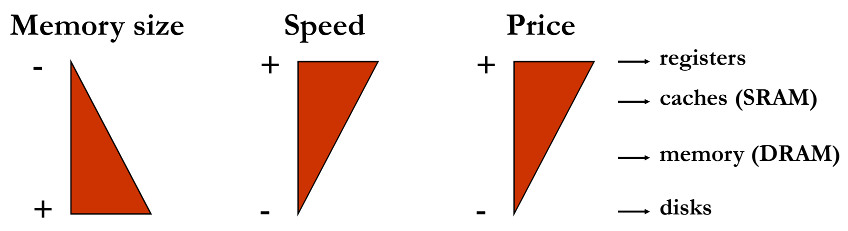
Locality of memory references

- Temporal locality: a recently accessed memory location (instruction or data) is likely to be accessed again in the near future
- Spatial locality: memory locations (instruction or data) close to a recently accessed location are likely to be accessed in the near future

Memory Hierarchies

Ideally one would desire an indefinitely large memory capacity such that any particular … word would be immediately available… we are … forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

A. W. Burks, H. H. Goldstine, and J. von Neumann - 1946



Memory hierarchy

- Fast, small memory located close to the processor
  - – Called cache
- Slow, large memory at the bottom of the hierarchy
  - – Main memory, build with DRAM technology
- A number of intermediate memory levels
  - – Currently one such stage is necessary: 2nd-level cache
- Inclusivity: each level copies information stored at the lower levels

Memory hierarchy terminology

- Block (or line): the minimum amount of data transferred between 2 adjacent memory levels
- Hit: data is found – the ideal case
  - – Operation performed quickly
- Miss: data not found
  - – Must continue the search at the next level down
  - – After data is eventually located, it is copied at the memory level where the miss happened
    - Exploit temporal locality
- Hit/Miss ratio: proportion of accesses that hit/miss, respectively

Cache basics – Tag, valid bit

- Data are identified in (main) memory by their address
- Problem: how can this work in a much smaller memory, such as the 1st level cache?
- We need to store the address as well as the data
  - – In a field called the tag
- A cache location looks like:

tag - address	data

- When unoccupied, need to indicate that a location is unused – valid bit

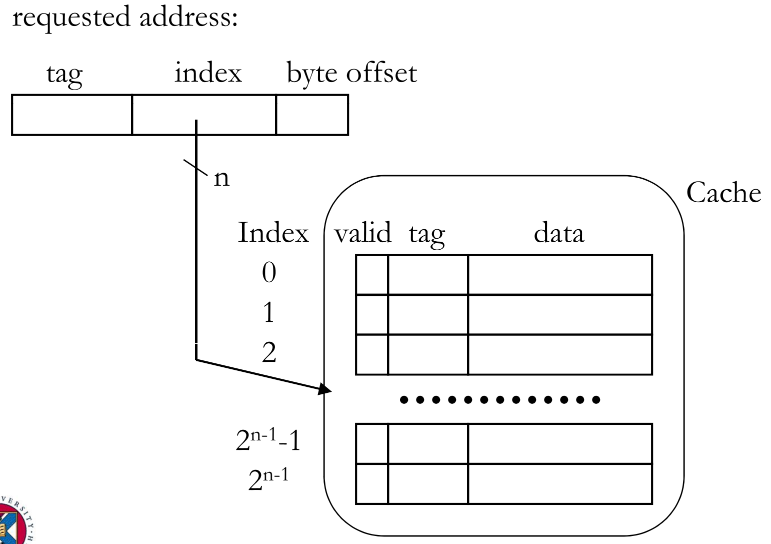
Fully-associative cache

- The cache just described is called fully-associative
- To check if requested data is in the cache
  - – Compare the requested address to all the tag fields
  - – If there is a match, the data is found – hit
- Two problems:
  - – The search is either very slow, or requires a very expensive memory type called Content Addressable Memory (CAM)
  - – Storing the full address as tag is wasteful

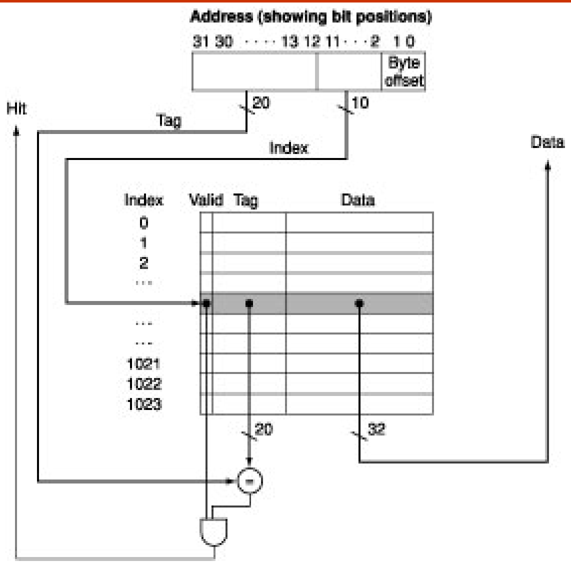
Direct-mapped cache

- By restricting the cache location where a data item can be stored, we can simplify the cache
- In a direct-mapped cache, a data item can be stored in one location only, determined by its address
  - Use some of the address bits as index to the cache array

Address mapping



Is it a hit?



Writing to caches

- Write through – write both in cache and next level down
- Write back – write to cache only
  - If a dirty entry is removed, it must be written to memory