

Lecture 12:

Memory hierarchy & caches

A modern memory subsystem combines

- fast small memory,
- slower larger memories

This lecture looks at why and how

Focus today mostly on electronic memories. Next lecture looks at supplementing electronic memory with disk storage.



Memory requirements

- Programmers wish for memory to be
 - Large
 - Fast
 - Random access
- Wish not achievable with 1 kind of memory
 - Issues of cost and technical feasibility



Memory examples

Technology	Typical access time	\$ per GB in 2008
SRAM	0.5 - 2.5 ns	\$2000 - \$5000
DRAM	50 - 70 ns	\$20 - \$75
Magnetic disk	5 - 20 ms	\$0.2 - \$2



Locality of memory references

- Can approximate programmers' wishes because of useful properties of memory references
 - **Temporal locality**: a recently accessed memory location (instruction or data) is likely to be accessed again
 - **Spatial locality**: memory locations (instruction or data) close to a recently accessed location are likely to be accessed in the near future
- Properties exploited by having a **memory hierarchy**



Idea of a memory hierarchy

- Use combination of memory kinds
 - Larger amounts of cheaper slower memory
 - Smaller amounts of more expensive faster memory
- Take advantage of temporal locality
 - If access data from slower memory, move it to faster memory
 - If data in faster memory unused recently, move it to slower memory
- Take advantage of spatial locality
 - If need to move a word from slower to faster memory, move adjacent words at same time



Idea is an old one

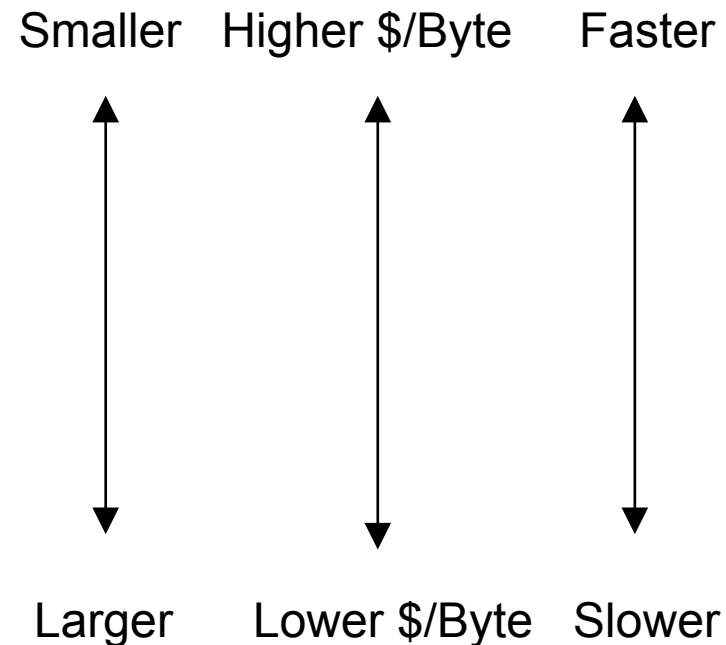
Ideally one would desire an indefinitely large memory capacity such that any particular ... word would be immediately available... we are ... forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

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



Levels of a typical memory hierarchy

- Registers in CPU
- Level 1 Cache (SRAM)
- Level 2 Cache (SRAM)
- Main memory (DRAM)
- Magnetic disk



Control of data transfers in hierarchy

- Q. Should the programmer explicitly copy data between levels of memory hierarchy?
- A. It depends: there is a trade-off between ease of programming and performance.
 - *Yes*: between registers and caches/main memory
 - *No*: between caches and main memory
 - *Sometimes*: between main memory and disk
 - *No*: when use disk area as virtual memory
 - *Yes*: when read and write files



Automatic data transfers between levels

- Happens between cache memory and main memory levels
- Programmers oblivious to where data held
 - Just see readable and writable locations at range of addresses
- Hardware manages transfers between levels
 - Lowest level holds master version of all data
 - Data copied to/from higher levels as needed



Memory hierarchy terminology

- **Block** (or **line**): the minimum amount of data transferred between 2 adjacent memory levels
 - E.g. in range 16-256 bytes
- **Hit**: data is found at higher level – 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



More memory hierarchy terminology

- **Hit rate (hit ratio):** fraction of accesses that are hits at a given level of the hierarchy
- **Hit time:** Time required to access a level of the hierarchy, including time to determine whether access is a hit or miss
- **Miss rate (miss ratio):** fraction of accesses that are misses at a given level ($= 1 - \text{hit rate}$)
- **Miss penalty:** Extra time required to fetch a block into some level from the next level down



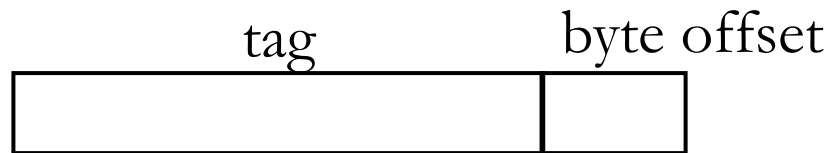
Cache basics – Tag, valid bit

- 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?
- Answer: associate with each data block in cache
 - a **tag** word, indicating the address of the main memory block it holds
 - a **valid bit**, indicating the block is in use



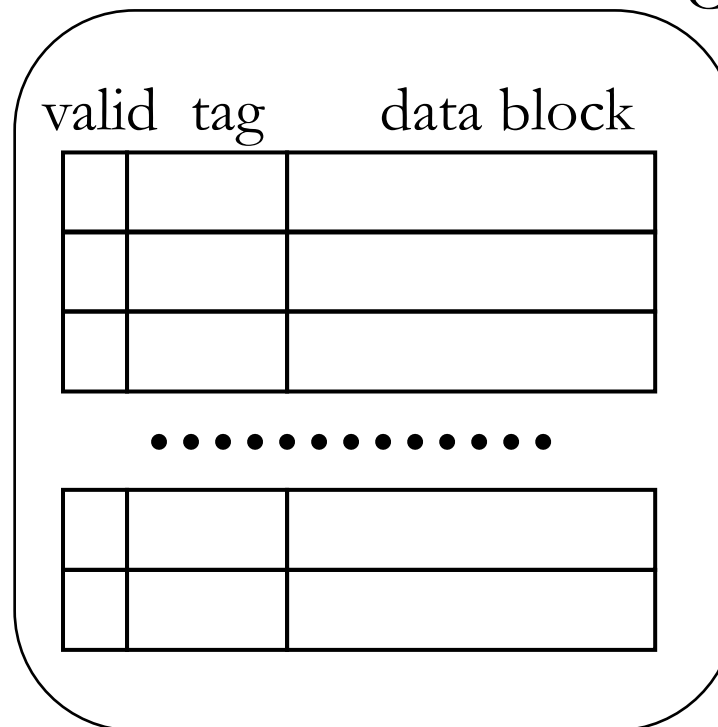
Fully-associative cache

requested address:



- Cache block selected by matching tags
- Byte offset selects word/byte within block
- Address tag can potentially match tag of *any* cache block

Cache



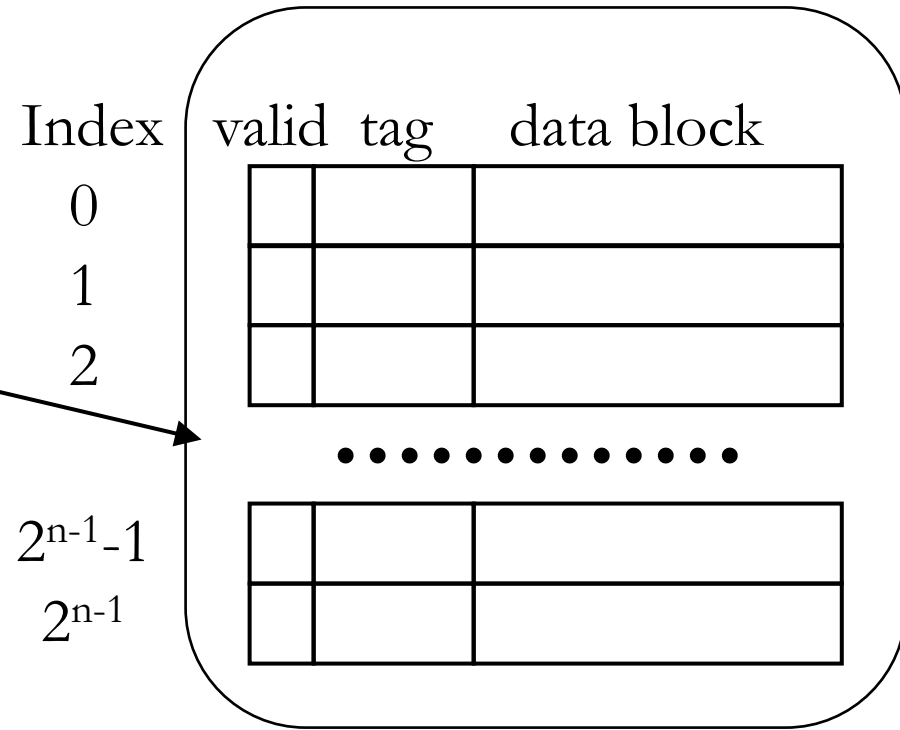
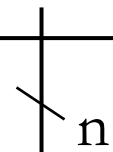
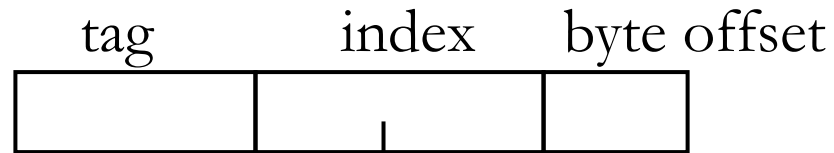
Direct-mapped cache

- In a fully-associative cache, search for matching tags is either very slow, or requires a very expensive memory type called Content Addressable Memory (CAM)
- 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

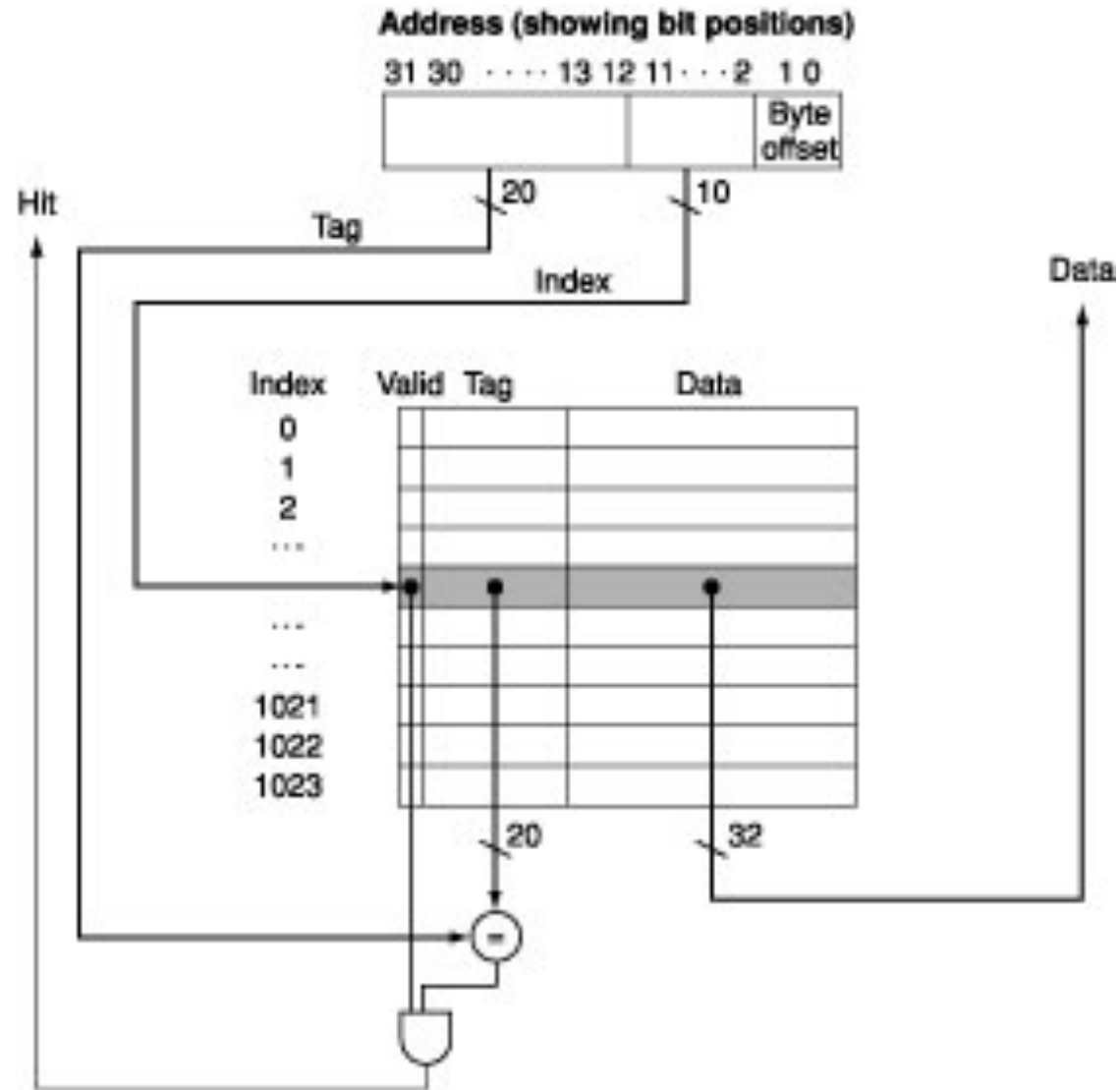


Address mapping for direct-mapped cache

requested address:



Is it a hit?



Writing to caches

- **Write through** – write both in cache and next level down
- **Write back** – write to cache only
 - Each cache block has a **dirty bit**, set if the block has been written to
 - When a **dirty** cache block is replaced, it is written to memory

