The Memory Hierarchy

Chap. 7 in P+H

"Memory hierarchy" refers to several layers of memory:

- fast, small, expensive cache (primary and secondary)
- main memory
- slow, large, cheap disk

- hopefully, data and instructions that are about to be needed are already in fast memory (i.e., cache)
- if a needed datum or instruction is found in cache, it's called a "cache hit"
- otherwise, it's a "cache miss", and slower memory is accessed.

- notice analogy: cache hits/misses \(\equiv\) paging/faults in V.M.
- typically hierarchy contains 4 levels, but could be any number (in theory) \(4, 12,\) main, disk

Purpose: Exploit temporal and spatial locality
Cache Organization - 3 major categories:
- Direct mappable
- Fully associative
- Set associative

Direct Mapped Cache
- simplest organization

- Word is placed in cache according to its address in main memory

- Since cache is smaller than main memory, many addresses in memory get mapped to same location in cache typically:

$$\text{MEM address} \mod \text{num words in cache}$$

If number of words in cache is a power of 2, then mapping is easy!

$$\frac{32-2}{\log(N)} = \frac{26}{2} = 26 \text{ bits}$$

$$N = \text{num words in cache}$$

- Index into cache

- Just look at bits 2 - $\left(\log(N) + 1\right)$

- Example: 256 KB primary cache, $\log(N) = 18 \div 2 = 16$

  - use bits 0-17 to index into cache

- Ignore, all in same word
Note that each word in cache is shared among
\[ \frac{2^{32}}{4} = \frac{2^{32}}{N} \] addresses
- of course, not all \( 2^{30} \) addresses will be in use!

How do we know which address an entry in the cache currently corresponds to?
- need a tag to identify the address
- just use the upper \( 32 - 2 \cdot \log(N) \) bits as the tag
- the rest of the bits are implicit in the cache index

---

![Diagram of cache structure](image)

- How do we know that a cache entry contains anything at all?
  - need a "valid" bit

So, a cache entry consumes \((32 - \log(N) + 1)\) extra bits as compared to memory.
Cache Placement

Associative caches

- "N-way set associative" - each block of memory can be mapped into N different possible locations in cache.

  1-way - direct mapped

  N-way, where N = # blocks in cache - fully associative

- Associative caches reduce the number of "conflict misses" block
- where a block in use must be removed from cache because another block in use maps to some cache location.
- even though there might be plenty of room in the cache
- blocks are N words apart, where N is number of words in the cache.
  - example: summing two arrays that are N words apart.

- Other kinds of misses are "compulsory misses" - occur the first time a block is needed
  and "capacity misses" - where the number of blocks in use exceeds the cache size.

- Associativity doesn't help these last two.
For n-way associative caches, the larger the n:
1) more tag bits needed in the cache
2) the increased number of comparators
   (comparing tags)
3) the larger the mux.

\[ 30 - (\log(N) - \log(n)) - \log(M) = (30 - \log(N) - \log(M)) + \log(n) \]
- thus, extra \( \log(n) \) bits per block.
- in fully associative case, that is \( \log(n) \) bits!

You also need \( n \) comparators and an \( n \) to-1 mux.