Review of Caching

In light of how significant memory reference is to the effective use of pipelining, cache design may be more important than any other aspect of computer design.

The goal of discussing caching is to establish what can and cannot be expected from a cache.

Why Caching Works

Programs exhibit locality of memory reference, which takes 2 forms:

- **Spacial locality** -- if memory location *a* is referenced, there is a high probability that the next location referenced will be "near" *a.*
- **Temporal locality** -- if memory location *a* is referenced, it is likely to be referenced again "soon."

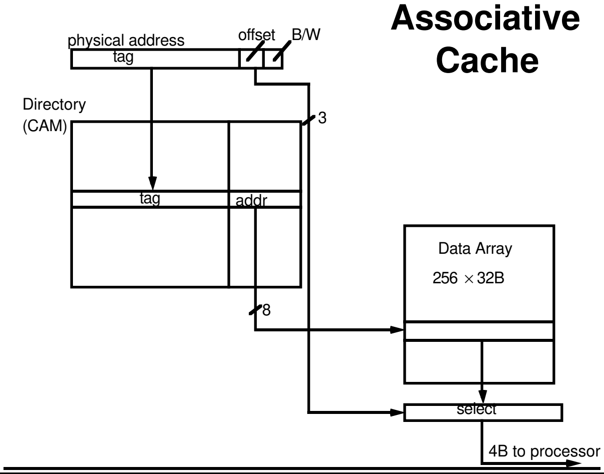
Many properties of programs cause the locality:

- Programs generally execute sequentially, i.e. executing an instruction at location *pc* generally implies the next instruction is at *pc+4*.
- Data structures are traversed in "order", i.e. data elements are not selected at random, usually, but rather have a notion of a "successor" which is often the next element referenced.
- Looping, the fundamental mechanism for leveraging the application of instructions against "new" data, re-references instructions.
- . . .

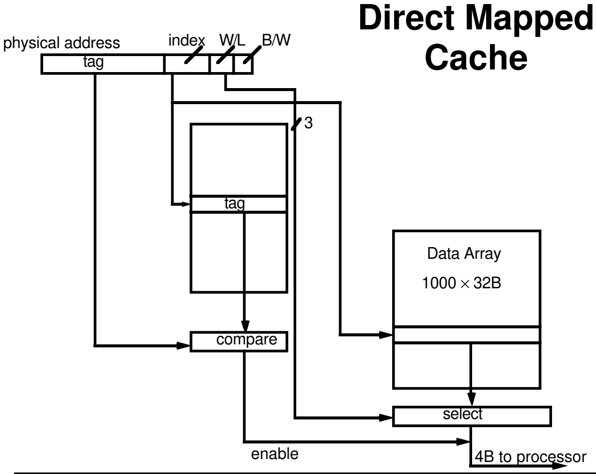
Caching Terminology

Hit -- finding the referenced word in the cache.
Miss -- not finding the referenced word in the cache.
Access time -- time to reference word on cache hit.
Miss time -- time to reference word on cache miss.
Compulsory miss -- first access to word
Capacity miss -- referencing word evicted because space is short.
Conflict miss -- referencing word that has been displaced by reference to same address.
Block or Line -- unit of cache storage.
CAM -- content addressable memory.

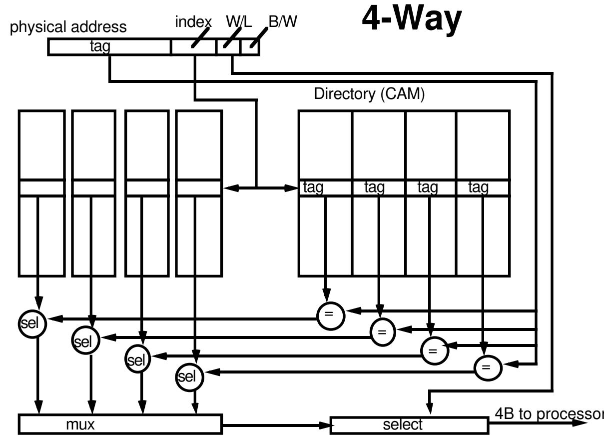
Associative Cache



Direct Mapped Cache



Set Associative 4-Way



Writing to a Cache

- There are two basic mechanisms for writing to cache:
  - -Write through -- each write to the cache also goes to the memory.
  - -Copy back -- only cache copy written until cache block is replaced when value written back to memory.
    - Requires a "dirty" bit.
- Write through is simpler but causes more memory traffic than copy back, which is somewhat more complex.
- Significant issue -- to allocate or not to allocate on write to nonresident line?
  - -Write through usually doesn't.
  - -Copy back usually does.
- Note, copyback must replace whole line on write.

Fetching a Line

Must select line to replace if cache is full --

Strategies: LRU, FIFO or random (only for assoc. caches)

In copy back, dirty lines must be written -- write buffer

Two basic strategies :

Fill cache starting from zero byte, and restarting processor when the whole line is present.

Return faulted word to processor and cache, filling with following words, cyclically.

Another strategy is "nonblocking caches" or prefetching.