Design Tradeoffs

Virtual or physical addressing

Virtually-addressed caches:

- access with a virtual address (index & tag)
- do address translation *only* on a cache miss

+ faster for hits because no address translation
- need to flush the cache on a context switch or
  process identification (PID) as part of the tag
- synonyms
  - **"the synonym problem"**
    - if 2 processes are sharing data, two (different) virtual addresses map to the same physical address
    - 2 copies of the same data in the cache
    - on a write, only one will be updated; so the other has old data
  + there are a few solutions (which we won't study)

Design Tradeoffs

Virtual or physical addressing

Physically addressed caches

- access with a physical address (index & tag)
- do address translation on *every* access

- increase in hit time because must translate the virtual address before access the cache
  + increase in hit time can be avoided if address translation is done in parallel with the cache access
    - access the cache with a virtual index:
      restrict cache size so that cache index bits are in the page offset (virtual = physical)
    - access the TLB with the virtual address bits
    - compare the physical tag from the cache to the physical address (page frame #) from the TLB
    - can increase cache size, but still use page offset bits for the index, by increasing associativity
+ no cache flushing on a context switch
+ no synonym problem

Cache Hierarchies

Cache hierarchy

- different caches with different sizes & access times & purposes
- \+ decrease effective memory access time:
  - many misses in the L1 cache will be satisfied by the L2 cache
  - can avoid going all the way to memory

Cache Hierarchies

Level 1 cache

- **goal: fast access**
  - so minimize hit time (the common case)
- small
  - so can access it in one CPU cycle (also chip real estate constraints)
- virtually addressed
  - so cache accesses can be fast without constraints on cache size
- direct mapped or set associative?
  - direct mapped: faster access
  - set associative: better hit ratio
- separate caches for instructions & data
  - each is smaller than a unified cache, so the access time is lower
  - can be configured differently
- write through

Cache Hierarchies

Level 2 cache

- **goal: keep traffic off the system bus**
  - the processor-memory bottleneck

- big cache
  - so it will have a high hit ratio
- physically addressed
  - more time to do address translation
  - no flushing on a context switch
- direct mapped
  - big direct-mapped caches have almost the same hit ratio as big set associative caches
  - slightly less hardware cost
- unified
  - its hit ratio is higher than that of two separate caches (I&D) half the size
- write back

Pentium Pro Memory Hierarchy

L1 on-chip instruction cache

- 8KB
- 32B block
- 4-way set associative
- physically-addressed cache: physical index, physical tags
  - TLB lookup in parallel

L1 on-chip data cache

- 8KB
- 32B block
- 2-way set associative
- physically-addressed cache: physical index, physical tags
  - TLB lookup in parallel
- write-back

Pentium Pro Memory Hierarchy

L2 on-chip cache

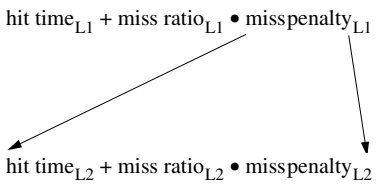
- 256KB
- 4-way set associative
- 32B block
- physically indexed

TLBs

- separate instruction & data TLBs
- 4-way set associative
- 32 entries (instruction); 64 entries (data)
- TLB misses handled in hardware

Measuring Cache Hierarchy Performance

Effective Access Time:



Comparing Caches & Paging

Timing aspects

- cache miss takes 5 to 50 cycles
- TLB miss takes 100s of cycles
- page fault takes milliseconds (millions of cycles)

How a miss/fault is handled

- cache miss: in hardware
- TLB miss: either in hardware or software
  if software, often there is no trap
- page fault: in software
  trap to the operating system

Mapping

- caches: direct-mapped or set associative
- TLBs: usually fully associative
- paging: fully associative

Comparing Caches & Paging

Page/block size

- cache block: 8 to 128 bytes
- TLB entry: size of a PTE (typically 4 to 8 bytes)
- page: 4KB - 4MB

Memory update policy

- caches: write through or write back to memory
- TLBs: write back to memory
- pages: write back to disk

Replacement policy

- TLBs & caches: LRU if 2-way set associative, but not as important
- paging: important to be LRU (why?)

All are demand-driven

Be sure you know why all these choices were made!