Memory Organization

- Memory hierarchy
  - processor registers
    - fewer in number (typically 16/32/128)
    - sub-cycle access time (1 nSec)
  - caches (L1, L2, L3, …)
    - on-chip memory
    - 10's of kBytes (to a few MBytes) of locations.
    - Access time: L1 : 1 – 2; L2: 2-5; L3:5-10 cycles.
  - main memory
    - 100's of MBytes storage
    - access time 10's of cycles
  - secondary storage
    - 100's of Giga bytes storage, access time ~ 10's mSec.

Cache Organizations

- Where can a block be placed?
  - Direct mapped, Set Associative, Associative
- How to identify a block in cache?
  - Tag, valid bit, …
- Replacement policy
  - FIFO, Random, LRU, PseudoLRU, …
- What happens on writes?
  - Write-back vs. write-through.
  - Write-allocate vs. write no-allocate

Cache Miss Classification

- Cold or Compulsory Misses: The first access to a block is a miss in Cache.
  - Increasing block size reduces compulsory misses.
- Capacity Misses: Misses due to limited size of cache.
  - Increasing cache size reduces capacity misses.
- Conflict Misses: Misses due to block placement strategy.
  - Increasing associativity reduces conflict misses.

Improving Cache Performance

- Avg. Mem. Access Time =

  Hit Time + MissRate*Miss Penalty

- To improve performance, reduce
  - Hit time
  - Miss Ratio
  - Miss Penalty

Improving Cache Performance

- Reduce Hit Time
  - smaller cache
  - direct mapped cache
  - for writes
    - write no-allocate
- Reduce Miss Rate
  - increase associativity
  - larger blocks (16 to 64 bytes typical)
  - victim cache – small buffer holding most recently discarded blocks
  - Prefetching
  - Compiler transformation: Interchange, Tiling, …

Improving Cache Performance

- Reduce Miss Penalty
  - smaller blocks
  - use a write buffer to hold dirty blocks being replaced so don't have to wait for the write to complete before reading
  - Sub-blocking with large blocks
    - for large blocks fetch critical word first
  - use multiple cache levels – L2 cache not tied to CPU clock rate
  - non-blocking caches

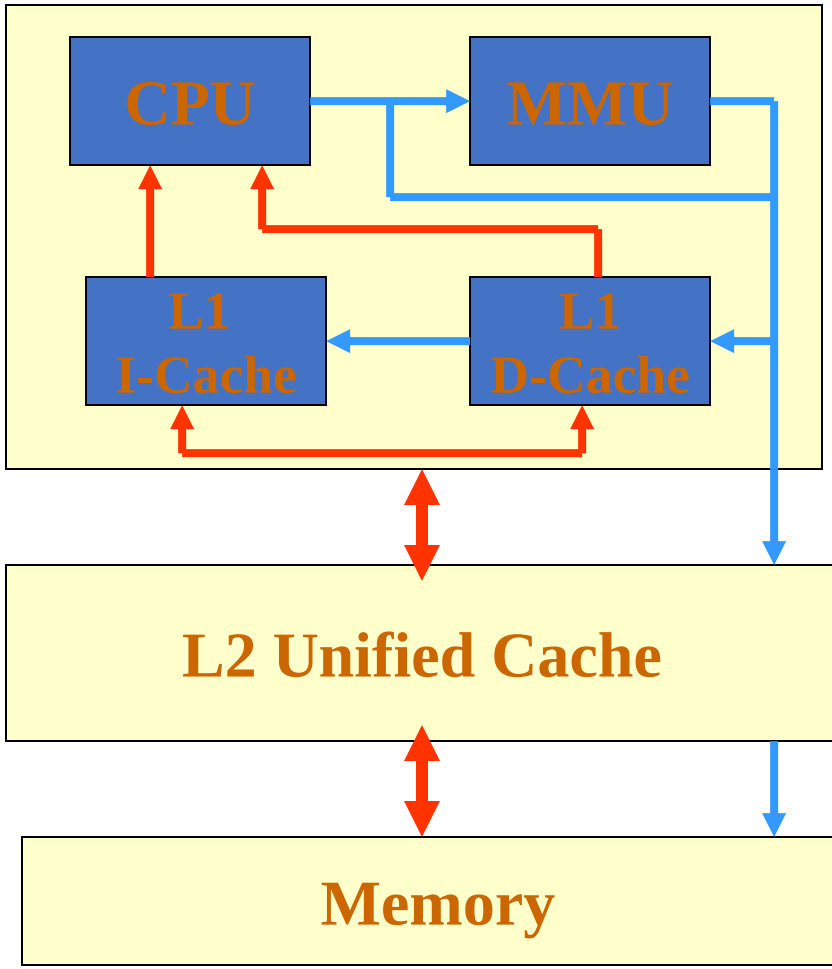
Non-blocking Caches

- *Non-blocking cache* or *lockup-free cache* allows data cache to continue to supply cache hits during a miss
- *Hit under miss* reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- *Hit under multiple misses* or *miss under miss* may further lower the effective miss penalty by overlapping multiple misses
  - miss status holding registers (MSHR) -- hardware structure for tracking outstanding misses
    - physical address of the block and word location in the block
    - destination register number
    - mechanism to merge requests to the same block
    - mechanism to ensure accesses to the same location happen in program order

Reducing Miss Penalty (contd.)

- Multi-level Caches:
  - small L1 cache -- to give a low hit time, and hence faster CPU cycle time .
  - large L2 cache to reduce L1 cache miss penalty.
  - L2 cache is typically set-associative to reduce L2-cache miss ratio!
  - Typically, L1 cache is direct mapped (or small associativity), separate I and D cache orgn.
  - L2 is unified and set-associative (8 or 16 way).
  - Intel Skylake Processor
    - L1-I / L1-D Cache : 32KB, 8-way, 64B line (Private)
    - L2 : 256 KB, 4-way, 64B line (Private)
    - L3 : 2MB (per core), 16-way, 64B line (Shared)

Putting Together



Acc. Time
1 - 2 cycles

Acc. Time
5 -10 cycles

Acc. Time
100's of cycles

Inclusion vs Exclusion

- Inclusive caches: Every block existing in the a lower level also exists in all subseq. higher levels
  - When fetching a block, place it in all cache levels.
  - When a block is evicted from higher level (L2), evict it from lower levels (L1) – back invalidation
  - Tradeoffs:
    - Leads to duplication of data in the hierarchy: less efficient
    - Maintaining inclusion takes effort (forced evictions)
    - ✓ Makes cache coherence in multiprocessors easier

Inclusion vs Exclusion

- Exclusive caches: The blocks contained in cache levels are mutually exclusive
  - When a block is fetched from memory, it is directly fetched only to that level
  - When a block is fetched from higher level (L2) to L1, it is evicted from L2.
  - When a block is evicted from lower level (L1), it may be written to the next level (L2).
  - Eviction at higher levels (L2), no action is required
    - ✓ More efficient utilization of cache space
    - ✓ (Potentially) More flexibility in replacement/placement
    - ○ More blocks/levels to keep track of to ensure cache coherence; takes effort

Inclusion vs Exclusion

- Non-inclusive and Non Exclusive caches: No guarantees for inclusion or exclusion. Simpler design
  - When a block is fetched from memory, it is allocated at all levels (L1 and L2)
  - When a block is evicted from higher levels (L2), no action is required
  - When a block is evicted from lower level (L1), if it missed in the higher level (L2), it may be allocated there (L2 serves as a victim)
  - Design followed in most Intel processors

Maintaining Inclusion and Exclusion

- When does maintaining inclusion take effort?
  - When a block is evicted from L2, need to evict all corresponding subblocks from L1
    - L1 block size < L2 block size; keep 1 bit per subblock in L2
  - When a block is inserted, make sure all higher levels also have it
- When does maintaining exclusion take effort?
  - When a block is inserted into any level, ensure it is not in any other
  - Cache invalidation requires requests percolating to all levels

Virtual Memory and Virtual Caches

- Is the address used for indexing/tag matching in cache virtual or physical?
- If index and tags bits used are both part of physical address (PIPT), then virtual to physical address translation must happen before cache access.
  - Address translation time in the critical path of cache access time $\Rightarrow$ increases hit time!
  - Our discussion so far considered PIPT
- Other organizations (VIPT, VIVT, PIVT) will be covered later in the course !