CS 61C:
Great Ideas in Computer Architecture
Caches

Instructors:
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

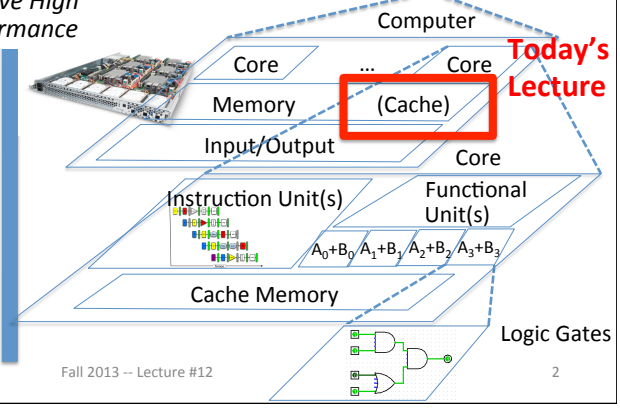
New-School Machine Structures (It's a bit more complicated!)

Software

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance

Hardware



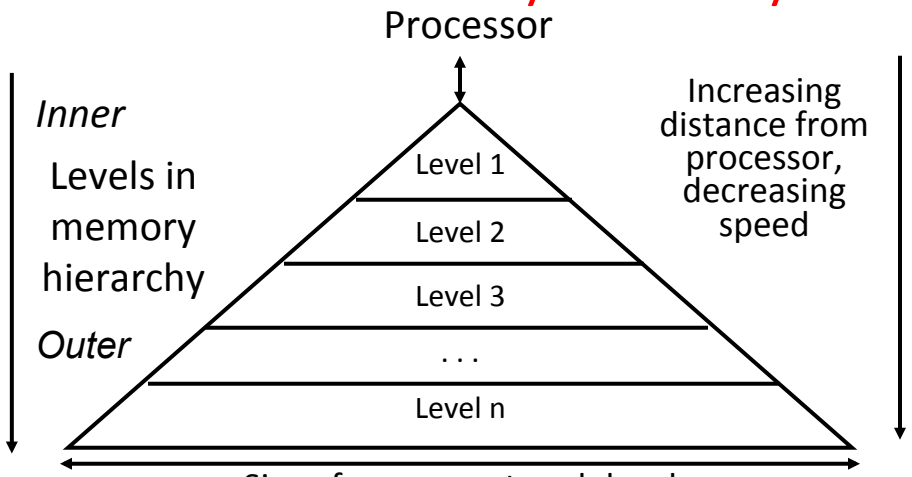
Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy and AMAT
- Technology Break
- Multiple Cache Levels
- And in Conclusion, …

Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy and AMAT
- Technology Break
- Multilevel Caches
- And in Conclusion, …

Review: Memory Hierarchy



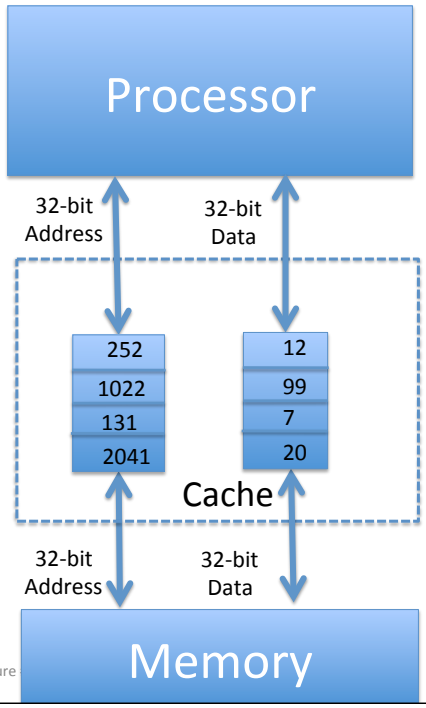
As we move to outer levels the latency goes up and price per bit goes down. Why?

Review: Cache Philosophy

- *Principle of Locality*: Programs access small portion of address space at any instant of time
  - *Temporal Locality* (locality in time)
  - *Spatial Locality* (locality in space)
- Programmer-invisible hardware mechanism to give illusion of speed of fastest memory with size of largest memory

Anatomy of a 16 Byte Cache, 4 Byte Block

- Operations:
  1. Cache Hit
  2. Cache Miss
  3. Refill cache from memory
- Cache needs Address Tags to decide if Processor Address is a Cache Hit or Cache Miss
  - Compares all 4 tags



Anatomy of a 32B Cache, 8B Block

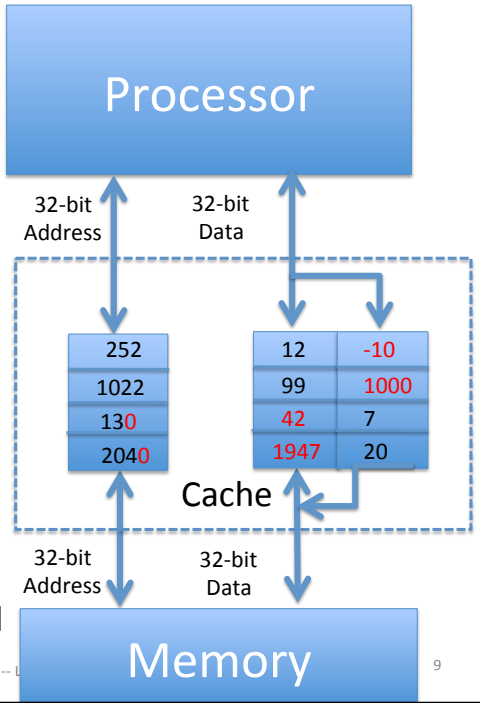
- Blocks must be aligned in pairs, otherwise could get same word twice in cache

⇒Tags only have even-numbered words

⇒ Last 3 bits of address always 000_{two}

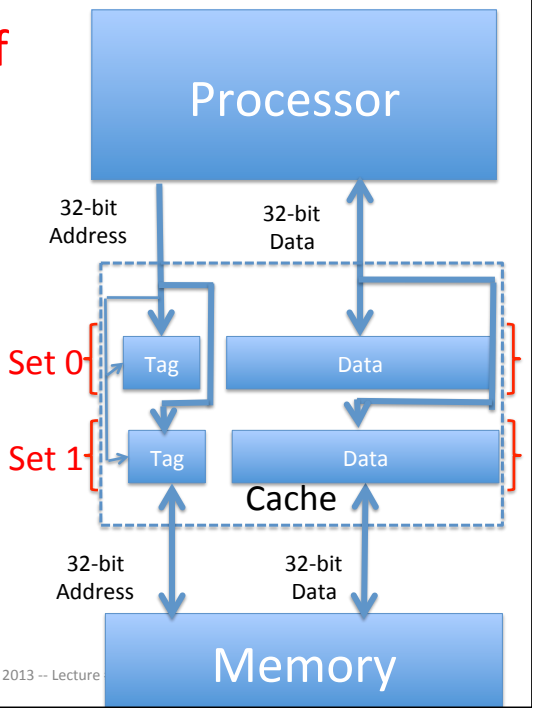
⇒Tags, comparators can be narrower

- Can get hit for either word in block



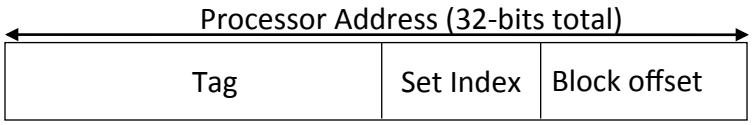
Hardware Cost of Cache

- Need to compare every tag to the Processor address
- Comparators are expensive
- Optimization: 2 sets => ½ comparators
- 1 Address bit selects which set



Processor Address Fields used by Cache Controller

- Block Offset: Byte address within block
- Set Index: Selects which set
- Tag: Remaining portion of processor address



- Size of Index = log2 (number of sets)
- Size of Tag = Address size – Size of Index – log2 (number of bytes/block)

What is Limit to # of Sets?

- Can save more comparators if have more than 2 sets
- Limit: As Many Sets as Cache Blocks – only needs one comparator!
- Called "Direct-Mapped" Design

One More Detail: Valid Bit

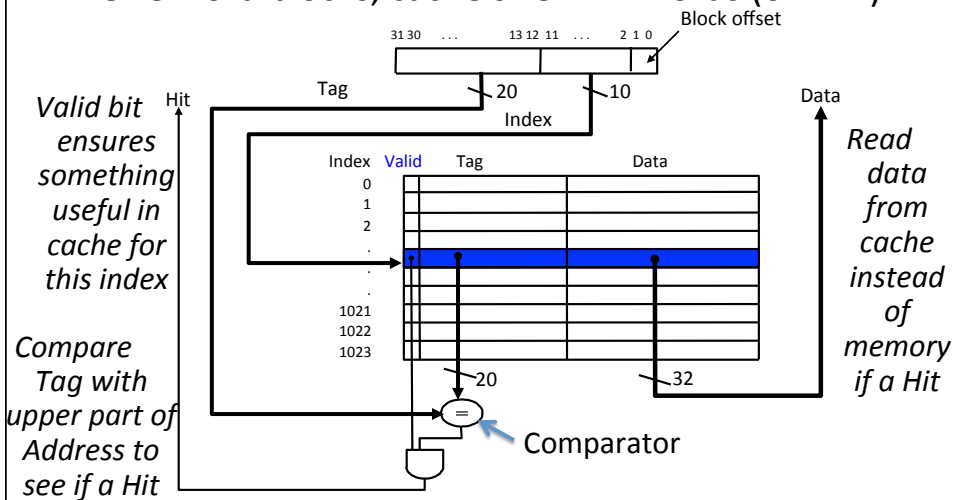
- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a "valid bit" to the cache tag entry
  - 0 => cache miss, even if by chance, address = tag
  - 1 => cache hit, if processor address = tag

Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy and AMAT
- Technology Break
- Multilevel Caches
- And in Conclusion, …

Direct-Mapped Cache Example

- One word blocks, cache size = 1K words (or 4KB)



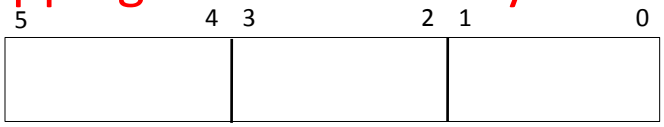
What kind of locality are we taking advantage of?

Cache Terms

- Hit rate: fraction of access that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

- Abbreviation: "$" = cache (A Berkeley innovation!)

Mapping a 6-bit Memory Address



Mem Block Within $ Block
Tag

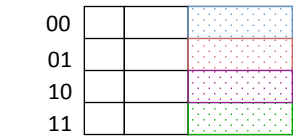
Block Within $
Index

Byte Offset Within Block
(e.g., Word)

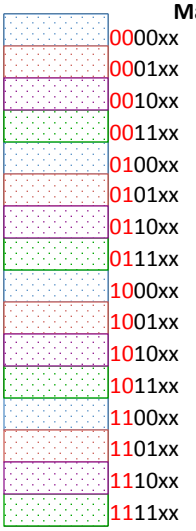
- In example, block size is 4 bytes/1 word (it could be multi-word)
- Memory and cache blocks are the same size, unit of transfer between memory and cache
- # Memory blocks >> # Cache blocks
  - 16 Memory blocks/16 words/64 bytes/6 bits to address all bytes
  - 4 Cache blocks, 4 bytes (1 word) per block
  - 4 Memory blocks map to each cache block
- Byte within block: low order two bits, ignore! (nothing smaller than a block)
- Memory block to cache block, aka *index*: middle two bits
- Which memory block is in a given cache block, aka *tag*: top two bits

Caching: A Simple First Example

Cache



Main Memory



One word blocks. Two low-order bits define the byte in the block (32b words).

Q: Where in the cache is the memory block?

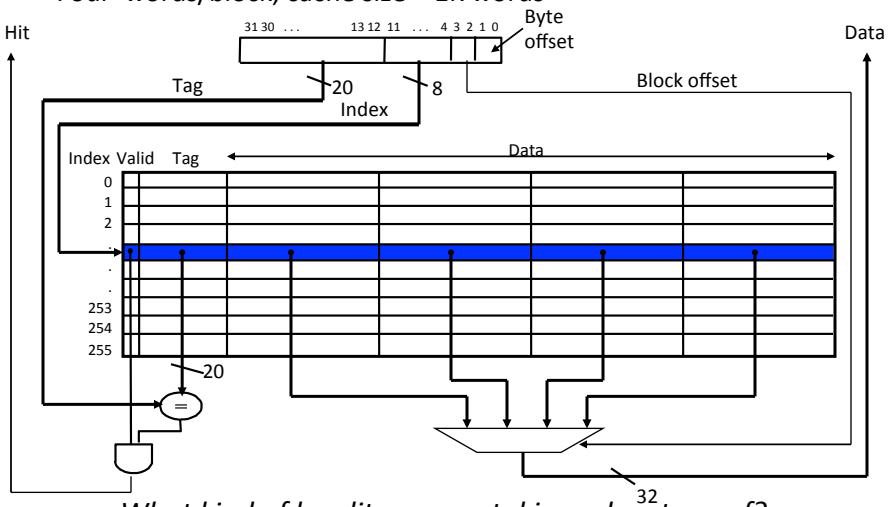
Use next 2 low-order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Q: Is the mem block in cache?

Compare the cache tag to the high-order 2 memory address bits to tell if the memory block is in the cache (provided valid bit is set)

Multiword-Block Direct-Mapped Cache

- Four words/block, cache size = 1K words



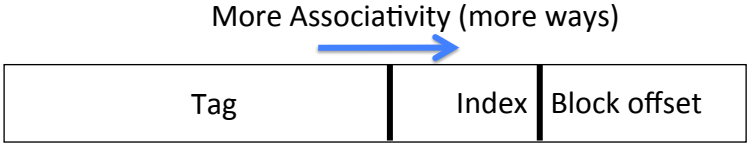
What kind of locality are we taking advantage of?

Cache Names for Each Organization

- "Fully Associative": Block can go anywhere
  - First design in lecture
  - Note: No Index field, but 1 comparator/block
- "Direct Mapped": Block goes one place
  - Note: Only 1 comparator
  - Number of sets = number blocks
- "N-way Set Associative": N places for a block
  - Number of sets = number of blocks / N
  - Fully Associative: N = number of blocks
  - Direct Mapped: N = 1

Range of Set-Associative Caches

- For a fixed-size cache, each increase by a factor of 2 in associativity doubles the number of blocks per set (i.e., the number of "ways") and halves the number of sets –
  - Decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Note: IBM persists in calling sets "ways" and ways "sets". They're wrong.

For S sets, N ways, B blocks, which statements hold?

A) The cache has B tags
B) The cache needs N comparators
C) B = N x S
D) Size of Index = $\text{Log}_2(S)$

- ☐ A only
- ☐ A and B only
- ☐ A, B, and C only
- ☐ All four statements are true

23

Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy and AMAT
- Technology Break
- Multilevel Caches
- And in Conclusion, …

Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy
- Technology Break
- AMAT
- And in Conclusion, …

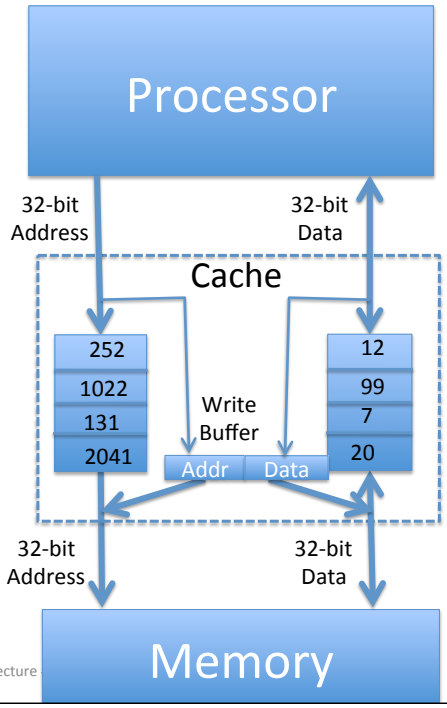
Handling Stores with Write-Through

- Store instructions write to memory, changing values
- Need to make sure cache and memory have same values on writes: 2 policies

1) Write-Through Policy: write cache and write *through* the cache to memory
   - Every write eventually gets to memory
   - Too slow, so include Write Buffer to allow processor to continue once data in Buffer
   - Buffer updates memory in parallel to processor

Write-Through Cache

- Write both values in cache and in memory
- Write buffer stops CPU from stalling if memory cannot keep up
- Write buffer may have multiple entries to absorb bursts of writes
- What if store misses in cache?

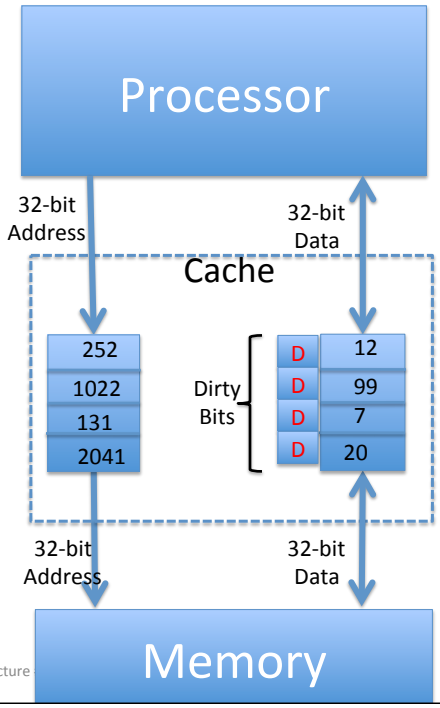


Handling Stores with Write-Back

2) Write-Back Policy: write only to cache and then write cache block *back* to memory when evict block from cache
   - Writes collected in cache, only single write to memory per block
   - Include bit to see if wrote to block or not, and then only write back if bit is set
     - Called "Dirty" bit (writing makes it "dirty")

Write-Back Cache

- Store/cache hit, write data in cache *only* & set dirty bit
  - Memory has stale value
- Store/cache miss, read data from memory, then update and set dirty bit
  - "Write-allocate" policy
- Load/cache hit, use value from cache
- On any miss, write back evicted block, only if dirty. Update cache with new block and clear dirty bit.



Write-Through vs. Write-Back

- Write-Through:
  - Simpler control logic
  - More predictable timing simplifies processor control logic
  - Easier to make reliable, since memory always has copy of data
- Write-Back
  - More complex control logic
  - More variable timing (0,1,2 memory accesses per cache access)
  - Usually reduces write traffic
  - Harder to make reliable, sometimes cache has only copy of data

Average Memory Access Time (AMAT)

- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses in the cache

AMAT = Time for a hit
+ Miss rate x Miss penalty

Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

Given a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache hit time of 1 clock cycle, what is AMAT?

- ☐ ≤200 psec
- ☐ 400 psec
- ☐ 600 psec
- ☐ ≥ 800 psec

Average Memory Access Time (AMAT)

- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

- What is the AMAT for a processor with a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache access time of 1 clock cycle?

1 + 0.02 x 50 = 2 clock cycles
Or 2 x 200 = 400 psecs

Average Memory Access Time (AMAT)

- Average Memory Access Time (AMAT) is the average to access memory considering both hits and misses

AMAT = Time for a hit + Miss rate x Miss penalty

- How calculate if separate instruction and data caches?

Impact of Cache on CPI

- Assume cache hit time included in normal CPU execution time, then

  CPU time = Instruction Count (IC) × Cycles Per Instruction (CPI) × Cycle Time (CT)

  = IC × (CPI_{ideal} + CPI_{miss}) × CT

  (CPI_{stalls})

  - A simple model for cache miss impact on CPI

  CPI_{miss}= accesses/instruction × miss rate × miss penalty

Impacts of Cache Performance

- Relative $ penalty increases as processor performance improves (faster clock rate and/or lower CPI)
  - When calculating CPI_{stalls}, cache miss penalty is measured in processor clock cycles needed to handle a miss
  - Lower the CPI_{ideal}, more pronounced impact of stalls
- Processor with a CPI_{ideal} of 2, a 100-cycle miss penalty, 36% load/store instr's, and 2% I$ and 4% D$ miss rates
  - CPI_{miss}= 2% × 100 + 36% × 4% × 100 = 3.44
  - So CPI_{stalls} = 2 + 3.44 = 5.44
  - More than twice the CPI_{ideal} !
- What if the CPI_{ideal} is reduced to 1?
- What if the D$ miss rate went up by 1%?

Impact of Larger Cache on AMAT?

- 1) Lower Miss rate
- 2) Longer Access time (Hit time): smaller is faster
  - Increase in hit time will likely add another stage to the pipeline
- At some point, increase in hit time for a larger cache may overcome the improvement in hit rate, yielding a decrease in performance
- Computer architects expend considerable effort optimizing organization of cache hierarchy – big impact on performance and power!

How to Reduce Miss Penalty?

- Could there be locality on misses from a cache?
- Use multiple cache levels!
- With Moore's Law, more room on die for bigger L1 caches and for second-level (L2)cache
- And in some cases even an L3 cache!
- IBM mainframes have ~1GB L4 cache off-chip.

Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy and AMT
- Technology Break
- Multilevel Caches
- And in Conclusion, …

Agenda

- Review
- Direct Mapped Cache
- Administrivia
- Write Policy and AMAT
- Technology Break
- Multilevel Caches
- And in Conclusion, …

Multiple Cache Levels

- E.g., CPI_{ideal} of 2,
  100 cycle miss penalty (to main memory),
  25 cycle miss penalty (to L2$),
  36% load/stores,
  a 2% (4%) L1 I$ (D$) miss rate,
  add a 0.5% L2$ miss rate
  - CPI_{stalls} = 2 + 0.02×25 + 0.36×0.04×25
    + 0.005×100 + 0.36×0.005×100
    = 3.54 (vs. 5.44 with no L2$)

Local vs. Global Miss Rates

- *Local miss rate* – the fraction of references to one level of a cache that miss
- Local Miss rate L2$ = $L2 Misses / L1$ Misses
- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache
  - L2$ local miss rate >> than the global miss rate
  - Often as high as 50% local miss rate – still useful?

For L1 cache

AMAT = Time for a hit + Miss rate x Miss penalty

What is AMAT for system with L1 and L2 cache (L2 miss rate is *local* miss rate)?

- ☐ Time for L2 hit + L2 Miss rate x L2 Miss penalty
- ☐ Time for L1 hit + L1 Miss rate x L2 Miss rate x Miss penalty
- ☐ Time for L1 hit + L1 Miss rate x (Time for L2 hit + L2 Miss rate x Miss Penalty)
- ☐ **Time for L1 hit + L1 Miss rate x Miss penalty + Time for L2 hit + L2 Miss rate x Miss penalty**

45

Local vs. Global Miss Rates

- *Local miss rate* – the fraction of references to one level of a cache that miss
- Local Miss rate L2$ = $L2 Misses / L1$ Misses
- *Global miss rate* – the fraction of references that miss in all levels of a multilevel cache
  - L2$ local miss rate >> than the global miss rate
- Global Miss rate = L2$ Misses / Total Accesses
  = L2$ Misses / L1$ Misses x L1$ Misses / Total Accesses
  = Local Miss rate L2$ x Local Miss rate L1$
- AMAT = Time for a hit + Miss rate x Miss penalty
- AMAT = Time for a L1$ hit + (local) Miss rateL1$ x (Time for a L2$ hit + (local) Miss rate L2$ x L2$ Miss penalty)

Improving Cache Performance (1 of 3)

AMAT = Hit Time + Miss rate x Miss penalty

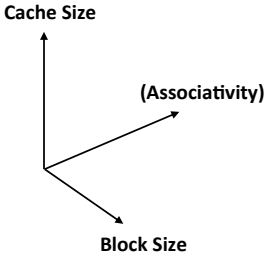
1. Reduce the time to hit in the cache
   - Smaller cache
2. Reduce the miss rate
   - Bigger cache
   - Larger blocks (16 to 64 bytes typical)
   - (Later in semester: More flexible placement by increasing associativity)

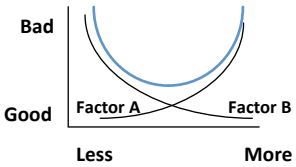
Improving Cache Performance (2 of 3)

3. Reduce the miss penalty
   - Smaller blocks
   - Use multiple cache levels
     - L2 cache size not tied to processor clock rate
   - Higher DRAM memory bandwidth (faster DRAMs)
   - Use a write buffer to hold dirty blocks being replaced so don't have to wait for the write to complete before reading

The Cache Design Space (3 of 3)

- Several interacting dimensions
  - Cache size
  - Block size
  - Write-through vs. write-back
  - Write allocation
  - (Later Associativity)
  - (Later Replacement policy)
- Optimal choice is a compromise
  - Depends on access characteristics
    - Workload
    - Use (I-cache, D-cache)
  - Depends on technology / cost
- Simplicity often wins





Multilevel Cache Design Considerations

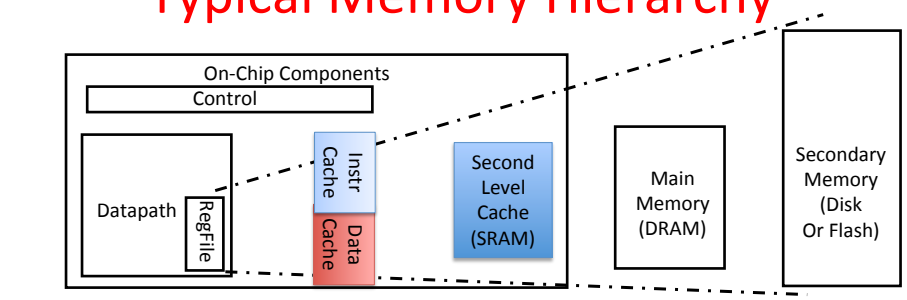
- Different design considerations for L1$ and L2$
  - L1$ focuses on minimizing hit time for shorter clock cycle: Smaller $ with smaller block sizes
  - L2$(s) focus on reducing miss rate to reduce penalty of long main memory access times: Larger $ with larger block sizes
- Miss penalty of L1$ is significantly reduced by presence of L2$, so can be smaller/faster but with higher miss rate
- For the L2$, hit time is less important than miss rate
  - L2$ hit time determines L1$'s miss penalty

Characteristic	Intel Nehalem	AMD Opteron X4 (Barcelona)
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KB each for instructions/data per core	64 KB each for instructions/data per core
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L1 hit time (load-use)	Not Available	3 clock cycles
L2 cache organization	Unified (instruction and data) per core	Unified (instruction and data) per core
L2 cache size	256 KB (0.25 MB)	512 KB (0.5 MB)
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L2 hit time	Not Available	9 clock cycles
L3 cache organization	Unified (instruction and data)	Unified (instruction and data)
L3 cache size	8192 KB (8 MB), shared	2048 KB (2 MB), shared
L3 block size	64 bytes	64 bytes
L3 write policy	Write-back, Write-allocate	Write-back, Write-allocate
L3 hit time	Not Available	38 (?)clock cycles

CPI/Miss Rates/DRAM Access SpecInt2006

Name	CPI	Data Only: L1 D cache misses/1000 instr	Data Only: L2 D cache misses/1000 instr	Instructions and Data: DRAM accesses/1000 instr
perl	0.75	3.5	1.1	1.3
bzip2	0.85	11.0	5.8	2.5
gcc	1.72	24.3	13.4	14.8
mcf	10.00	106.8	88.0	88.5
go	1.09	4.5	1.4	1.7
hmmer	0.80	4.4	2.5	0.6
sjeng	0.96	1.9	0.6	0.8
libquantum	1.61	33.0	33.1	47.7
h264avc	0.80	8.8	1.6	0.2
omnetpp	2.94	30.9	27.7	29.8
astar	1.79	16.3	9.2	8.2
xalancbmk	2.70	38.0	15.8	11.4
Median	1.35	13.6	7.5	5.4

Typical Memory Hierarchy



Speed (cycles):	½'s	1's	10's	100's	1,000,000's
Size (bytes):	100's	10K's	M's	G's	T's
Cost/bit:	highest	⟷			lowest

- Principle of locality + memory hierarchy presents programmer with ≈ as much memory as is available in the *cheapest* technology at the ≈ speed offered by the *fastest* technology

And in Conclusion, …

- Great Ideas: Principle of Locality and Memory Hierarchy
- Cache – copy of data lower level in memory hierarchy
- Direct Mapped to find block in cache using Tag field and Valid bit for Hit
- AMAT balances Hit time, Miss rate, Miss penalty
- Larger caches reduce Miss rate via Temporal and Spatial Locality, but can increase Hit time
- Multilevel caches reduce the Miss penalty
- Write-through versus write-back caches