CS152
Computer Architecture and Engineering
Lecture 20

Caches

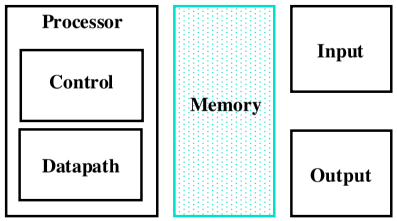
April 14, 2003

John Kubiatowicz (www.cs.berkeley.edu/~kubitron)

lecture slides: http://inst.eecs.berkeley.edu/~cs152/

Recap: The Big Picture: Where are We Now?

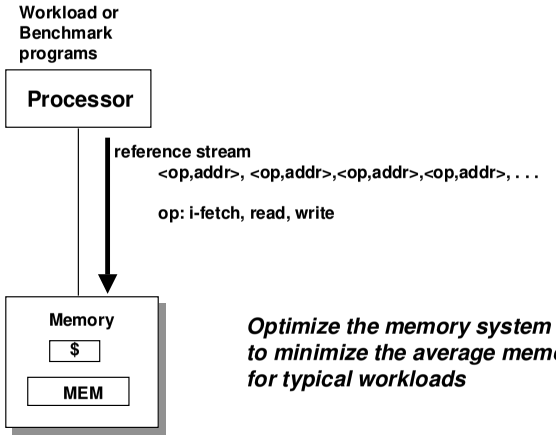
° The Five Classic Components of a Computer



° Today's Topics:

- Recap last lecture
- Simple caching techniques
- Many ways to improve cache performance
- Virtual memory?

The Art of Memory System Design



Optimize the memory system organization to minimize the average memory access time for typical workloads

Recap: Cache Performance

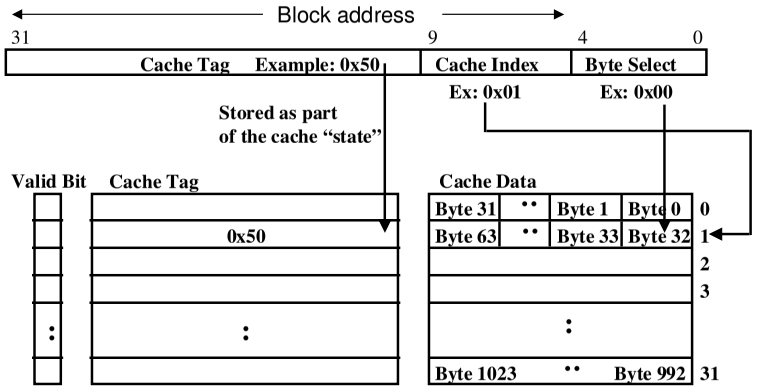
Execution_Time =
Instruction_Count x Cycle_Time x
(ideal CPI + Memory_Stalls/Inst + Other_Stalls/Inst)

Memory_Stalls/Inst =
Instruction Miss Rate x Instruction Miss Penalty +
Loads/Inst x Load Miss Rate x Load Miss Penalty +
Stores/Inst x Store Miss Rate x Store Miss Penalty

Average Memory Access time (AMAT) =
Hit Time_{L1} + (Miss Rate_{L1} x Miss Penalty_{L1}) =
(Hit Rate_{L1} x Hit Time_{L1}) + (Miss Rate_{L1} x Miss Time_{L1})

Example: 1 KB Direct Mapped Cache with 32 B Blocks

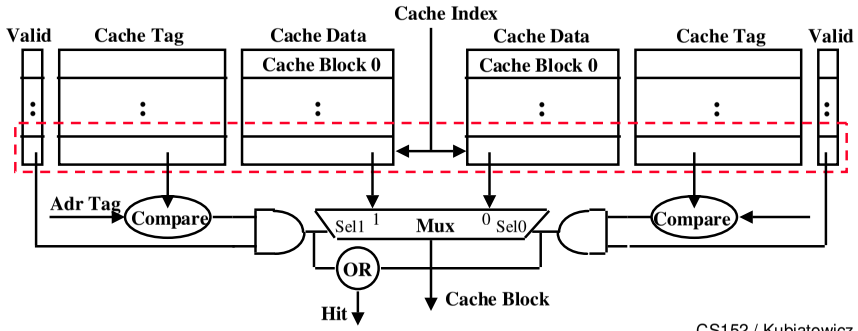
° For a 2 ** N byte cache:
- The uppermost (32 - N) bits are always the Cache Tag
- The lowest M bits are the Byte Select (Block Size = 2^M)
- One cache miss, pull in complete "Cache Block" (or "Cache Line")



Set Associative Cache

° N-way set associative: N entries for each Cache Index
- N direct mapped caches operates in parallel

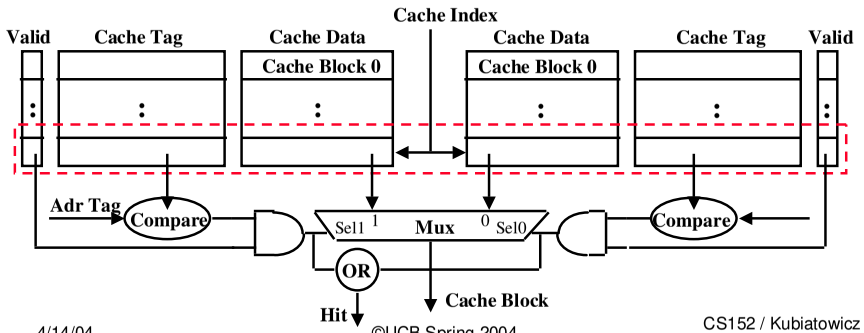
° Example: Two-way set associative cache
- Cache Index selects a "set" from the cache
- The two tags in the set are compared to the input in parallel
- Data is selected based on the tag result



Disadvantage of Set Associative Cache

° N-way Set Associative Cache versus Direct Mapped Cache:
- N comparators vs. 1
- Extra MUX delay for the data
- Data comes AFTER Hit/Miss decision and set selection

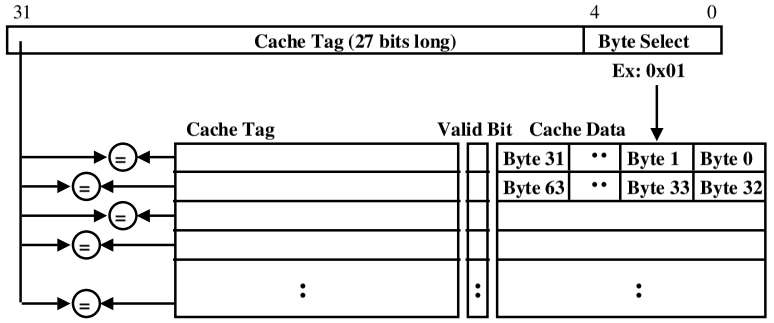
° In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
- Possible to assume a hit and continue. Recover later if miss.



Example: Fully Associative

° Fully Associative Cache
- Forget about the Cache Index
- Compare the Cache Tags of all cache entries in parallel
- Example: Block Size = 32 B blocks, we need N 27-bit comparators

° By definition: Conflict Miss = 0 for a fully associative cache



A Summary on Sources of Cache Misses

° **Compulsory** (cold start or process migration, first reference): first access to a block
- "Cold" fact of life: not a whole lot you can do about it
- Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant

° **Capacity**:
- Cache cannot contain all blocks access by the program
- Solution: increase cache size

° **Conflict** (collision):
- Multiple memory locations mapped to the same cache location
- Solution 1: increase cache size
- Solution 2: increase associativity

° **Coherence** (Invalidation): other process (e.g., I/O) updates memory

Design options at constant cost

	Direct Mapped	N-way Set Associative	Fully Associative
Cache Size	Big	Medium	Small
Compulsory Miss	Same	Same	Same
Conflict Miss	High	Medium	Zero
Capacity Miss	Low	Medium	High
Coherence Miss	Same	Same	Same

Note:
If you are going to run "billions" of instruction, Compulsory Misses are insignificant (except for streaming media types of programs).

Four Questions for Caches and Memory Hierarchy

° Q1: Where can a block be placed in the upper level? *(Block placement)*

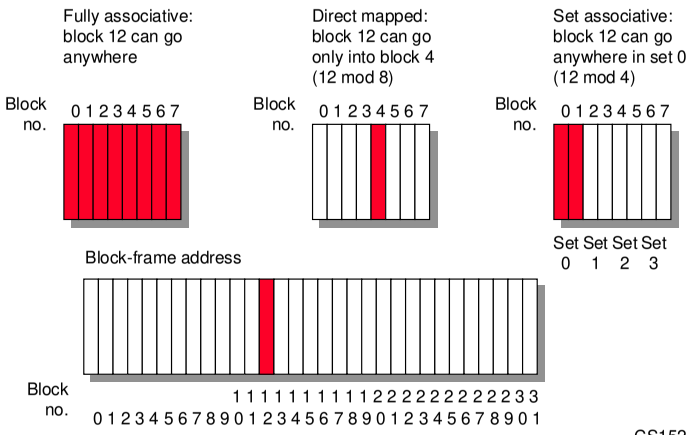
° Q2: How is a block found if it is in the upper level? *(Block identification)*

° Q3: Which block should be replaced on a miss? *(Block replacement)*

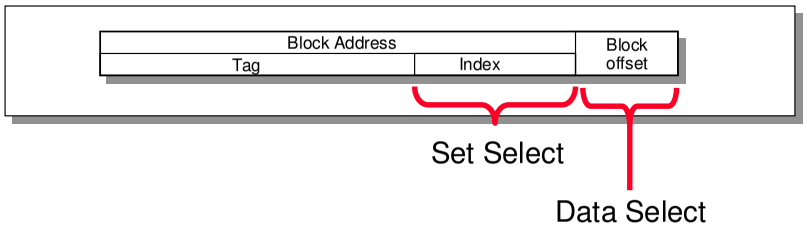
° Q4: What happens on a write? *(Write strategy)*

Q1: Where can a block be placed in the upper level?

° Block 12 placed in 8 block cache:
- Fully associative, direct mapped, 2-way set associative
- S.A. Mapping = Block Number Modulo Number Sets



Q2: How is a block found if it is in the upper level?



° Direct indexing (using index and block offset), tag compares, or combination

° Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss?

° Easy for Direct Mapped

° Set Associative or Fully Associative:
- Random
- LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

° ***Write through***—The information is written to both the block in the cache and to the block in the lower-level memory.

° ***Write back***—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
- is block clean or dirty?

° Pros and Cons of each?
- WT:
  - PRO: read misses cannot result in writes
  - CON: Processor held up on writes unless writes buffered
- WB:
  - PRO: repeated writes not sent to DRAM
    processor not held up on writes
  - CON: More complex
    Read miss may require writeback of dirty data

° WT always combined with write buffers so that don't wait for lower level memory

New Question: How does a store to the cache work?

° Must update cache data, but *only if cached!*
- Otherwise may overwrite unrelated cache data!

° Two-cycle solution (old SPARC processor, for instance):

```
Fetch Decode Ex  Mem(check) Mem(write/stall) Wr
```

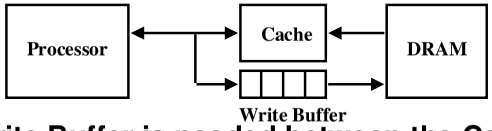
- Memory stage for store introduces a stall when writing
- First cycle, check tags
- Second cycle, (optional) write

° One Cycle solution

```
Fetch Decode Ex  Mem(check/store old) Wr
```

- Separate "Store buffer" always holds written value + Address + valid
- On load, check for match address to store buffer and return data
- In memory stage of store:
  - (1) Check tag for new store
  - (2) Write previous value to cache and clear valid bit
  - (3) if tag match, will set valid bit and write new value to store buffer
- Never stalls for the write, since defers to next store cycle

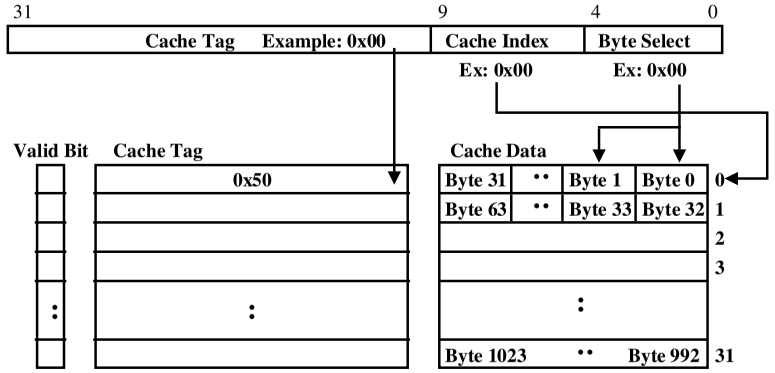
Write Buffer for Write Through



- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Must handle bursts of writes
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Can Write-buffer help with "Store Buffer"?
  - Yes, if careful, but don't want to overwrite entry until written to cache
  - This can be made to work, but must be careful

Write-miss Policy: Write Allocate versus Not Allocate

- Assume: a 16-bit write to memory location 0x0 and causes a miss
  - Do we allocate space in cache and possibly read in the block?
    - Yes: Write Allocate
    - No: Not Write Allocate

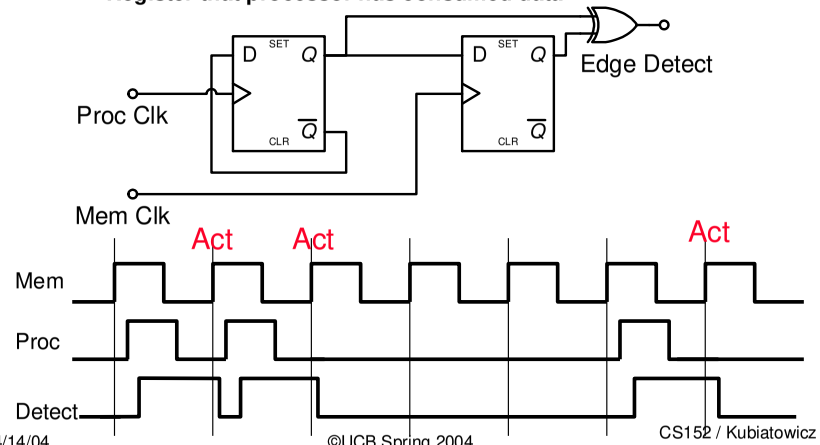


Administrative Issues

- Should be reading Chapter 7 of your book
  - Some of the advanced things in the graduate text
- Second midterm: Monday May 5th
  - Pipelining
    - Hazards, branches, forwarding, CPI calculations
    - (may include something on dynamic scheduling)
  - Memory Hierarchy
  - Possibly something on I/O (see where we get in lectures)
  - Possibly something on power (Broderson Lecture)
- Homework 5 is up
  - Due next Wednesday

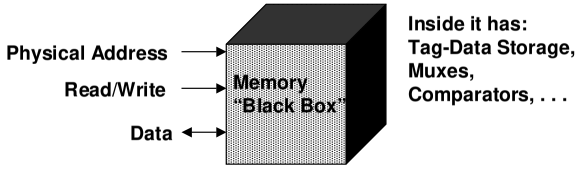
Administrivia: Edge Detection For Lab 5

- Lab 5 Hint: Edge Detection For Memory Controller
  - Detect rising edges of processor clock
  - Use detection signal to drive circuits synchronized to memory clock
    - Accept new requests
    - Register that processor has consumed data

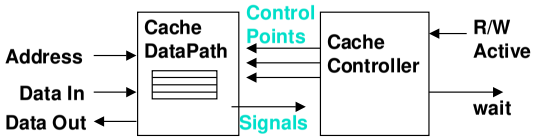


How Do you Design a Memory System?

° Set of Operations that must be supported
- read: data <= Mem[Physical Address]
- write: Mem[Physical Address] <= Data



° Determine the internal register transfers
° Design the Datapath
° Design the Cache Controller



Review: Stall Methodology in Memory Stage

° Freeze pipeline in Mem stage:

```
IF0 ID0  EX0 Mem0 Wr0  Noop  … Noop  Noop
    IF1  ID1 EX1  Mem1 stall … stall Mem1 Wr1
         IF2 ID2  EX2  stall … stall Ex2  Mem2 Wr2
             IF3  ID3  stall … stall ID3  Ex3   Mem3 Wr3
                  IF4  stall … stall IF4  ID4   Ex4  Mem4 Wr4
                                     IF5  ID5   Ex5  Mem5
```

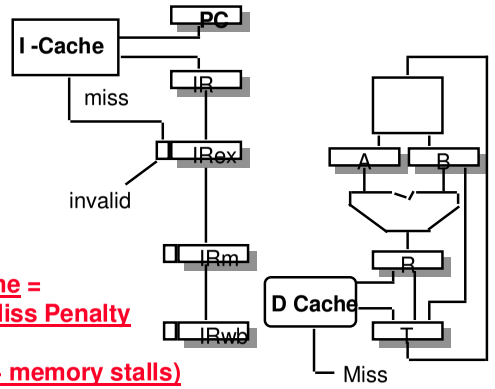
° Stall detected by end of Mem1 stage
- Think of caching system as providing a "result invalid" signal
- Because of stall, this invalid result doesn't make it to WR stage

° Really means:
- Freeze up, Bubble down
- During stall, Mem1, Ex2, ID3, and IF4 keep repeating!
- WR0 goes forward
- Bubbles (noops) passed from memory stage to WR stage

° As a result, cache fill can take an arbitrary time
- When pipeline finally released by cache, last cycle repeats Mem1 and performs good load/store

Impact on Cycle Time

Cache Hit Time:
directly tied to clock rate
increases with cache size
increases with associativity



Average Memory Access time =
Hit Time + Miss Rate x Miss Penalty

Time = IC x CT x (ideal CPI + memory stalls)

Improving Cache Performance: 3 general options

Time = IC x CT x (ideal CPI + memory stalls)

Average Memory Access time =
Hit Time + (Miss Rate x Miss Penalty) =

(Hit Rate x Hit Time) + (Miss Rate x Miss Time)

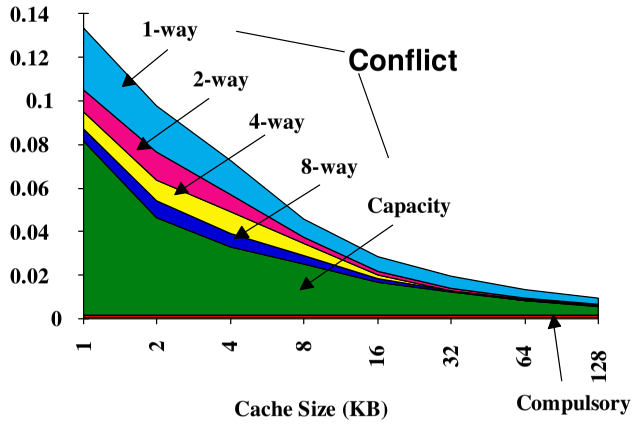
Options to reduce AMAT:

1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

Improving Cache Performance

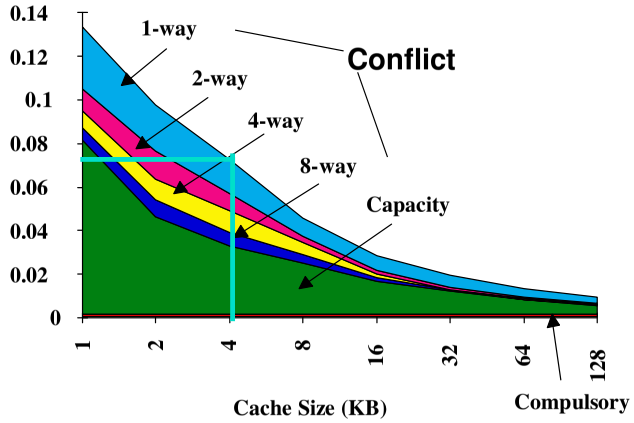
1. Reduce the miss rate,
2. Reduce the miss penalty, or
3. Reduce the time to hit in the cache.

3Cs Absolute Miss Rate (SPEC92)

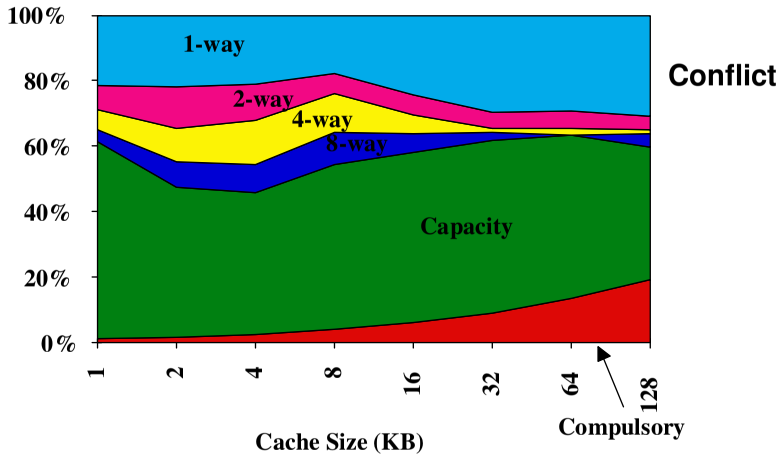


2:1 Cache Rule

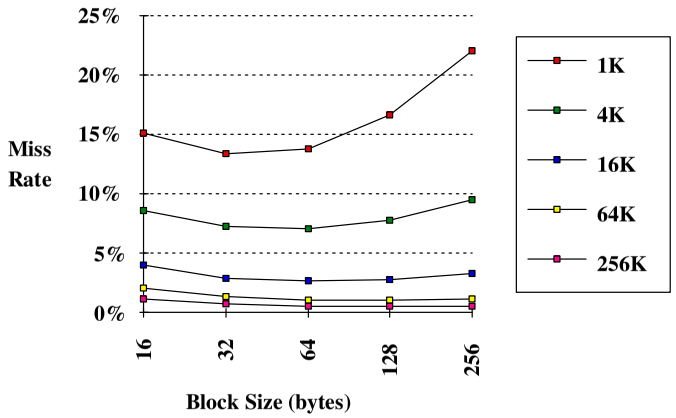
miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



3Cs Relative Miss Rate



1. Reduce Misses via Larger Block Size



2. Reduce Misses via Higher Associativity

- 2:1 Cache Rule:
  - Miss Rate DM cache size N - Miss Rate 2-way cache size N/2
- Beware: Execution time is only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%

Example: Avg. Memory Access Time vs. Miss Rate

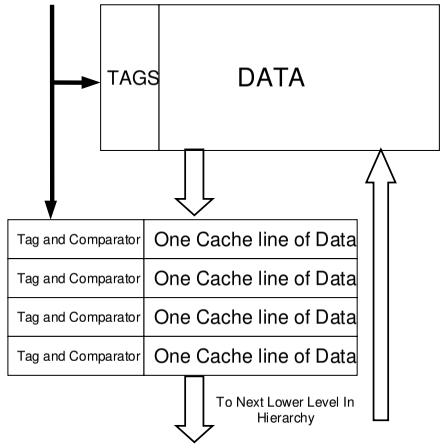
- Assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	1.48	1.47	1.43
16	1.29	1.32	1.32	1.32
32	1.20	1.24	1.25	1.27
64	1.14	1.20	1.21	1.23
128	1.10	1.17	1.18	1.20

(Red means A.M.A.T. not improved by more associativity)

3. Reducing Misses via a "Victim Cache"

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines



4. Reducing Misses by Hardware Prefetching

° E.g., Instruction Prefetching
- Alpha 21064 fetches 2 blocks on a miss
- Extra block placed in "stream buffer"
- On miss check stream buffer

° Works with data blocks too:
- Jouppi [1990] 1 data stream buffer got 25% misses from 4KB cache; 4 streams got 43%
- Palacharla & Kessler [1994] for scientific programs for 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches

° Prefetching relies on having extra memory bandwidth that can be used without penalty
- Could reduce performance if done indiscriminantly!!!

5. Reducing Misses by Software Prefetching Data

° Data Prefetch
- Load data into register (HP PA-RISC loads)
- Cache Prefetch: load into cache (MIPS IV, PowerPC, SPARC v. 9)
- Special prefetching instructions cannot cause faults; a form of speculative execution

° Issuing Prefetch Instructions takes time
- Is cost of prefetch issues < savings in reduced misses?
- Higher superscalar reduces difficulty of issue bandwidth

6. Reducing Misses by Compiler Optimizations

° McFarling [1989] reduced caches misses by 75% on 8KB direct mapped cache, 4 byte blocks in software

° Instructions
- Reorder procedures in memory so as to reduce conflict misses
- Profiling to look at conflicts(using tools they developed)

° Data
- *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
- *Loop Interchange*: change nesting of loops to access data in order stored in memory
- *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
- *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Improving Cache Performance (Continued)

1. Reduce the miss rate,
2. *Reduce the miss penalty,* or
3. Reduce the time to hit in the cache.

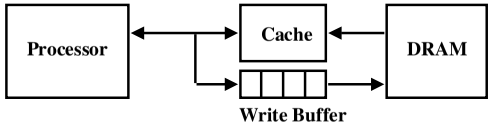
0. Reducing Penalty: Faster DRAM / Interface

° New DRAM Technologies
- RAMBUS - same initial latency, but much higher bandwidth
- Synchronous DRAM
- TMJ-RAM (Tunneling magnetic-junction RAM) from IBM??
- Merged DRAM/Logic - IRAM project here at Berkeley

° Better BUS interfaces

° CRAY Technique: only use SRAM

1. Reducing Penalty: Read Priority over Write on Miss

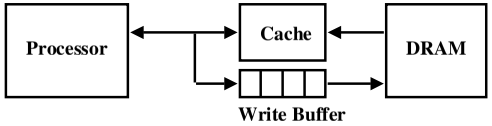


° A Write Buffer Allows Reordering of Requests
- Processor: writes data into the cache and the write buffer
- Memory controller: write contents of the buffer to memory

° Writes go to DRAM only when idle
- This allows the so-called "Read Around Write" capability
- Why important? Reads hold up the processor, writes do not

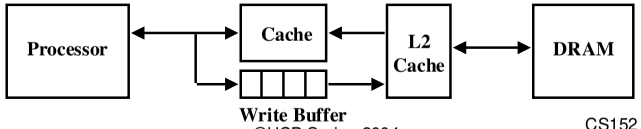
° Write buffer is just a FIFO:
- Typical number of entries: 4
- Must handle bursts of writes
- Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

Write Buffer Saturation



° Store frequency (w.r.t. time) > 1 / DRAM write cycle
- If this condition exist for a long period of time (CPU cycle time too quick and/or too many store instructions in a row):
  - Store buffer will overflow no matter how big you make it
  - The CPU Cycle Time <= DRAM Write Cycle Time

° Solution for write buffer saturation:
- Use a write back cache
- Install a second level (L2) cache: (does this always work?)



RAW Hazards from Write Buffer!

° Write-Buffer Issues: Could introduce RAW Hazard with memory!
- Write buffer may contain *only* copy of valid data ⇒ Reads to memory may get wrong result if we ignore write buffer

° Solutions:
- Simply wait for write buffer to empty before servicing reads:
  - Might increase read miss penalty (old MIPS 1000 by 50%)
- Check write buffer contents before read ("fully associative");
  - If no conflicts, let the memory access continue
  - Else grab data from buffer

° Can Write Buffer help with Write Back?
- Read miss replacing dirty block
  - Copy dirty block to write buffer *while* starting read to memory
- CPU stall less since restarts as soon as do read

2. Reduce Penalty: Early Restart and Critical Word First

° Don't wait for full block to be loaded before restarting CPU
- *Early restart*—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
- *Critical Word First*—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called *wrapped fetch* and *requested word first*
- *DRAM FOR LAB 6 can do this in burst mode! (Check out sequential timing)*

° Generally useful only in large blocks,

° Spatial locality a problem; tend to want next sequential word, so not clear if benefit by early restart

block

3. Reduce Penalty: Non-blocking Caches

° *Non-blocking cache* or *lockup-free cache* allow data cache to continue to supply cache hits during a miss
- requires F/E bits on registers or out-of-order execution
- requires multi-bank memories

° "*hit under miss*" reduces the effective miss penalty by working during miss vs. ignoring CPU requests

° "*hit under multiple miss*" or "*miss under miss*" may further lower the effective miss penalty by overlapping multiple misses
- Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
- Requires muliple memory banks (otherwise cannot support)
- Penium Pro allows 4 outstanding memory misses

Reprise: What happens on a Cache miss?

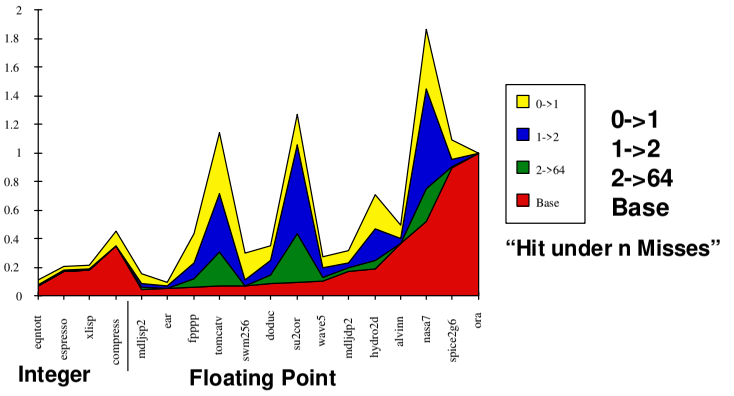
° For in-order pipeline, 2 options:
- Freeze pipeline in Mem stage (popular early on: Sparc, R4000)

```
IF  ID  EX  Mem stall stall stall … stall Mem   Wr
    IF  ID  EX  stall stall stall … stall stall Ex Wr
```

- Use Full/Empty bits in registers + MSHR queue
  - MSHR = "Miss Status/Handler Registers" (Kroft)
    Each entry in this queue keeps track of status of outstanding memory requests to one complete memory line.
    - Per cache-line: keep info about memory address.
    - For each word: register (if any) that is waiting for result.
    - Used to "merge" multiple requests to one memory line
  - New load creates MSHR entry and sets destination register to "Empty". Load is "released" from pipeline.
  - Attempt to use register before result returns causes instruction to block in decode stage.
  - Limited "out-of-order" execution with respect to loads.
    Popular with in-order superscalar architectures.

° Out-of-order pipelines already have this functionality built in… (load queues, etc).

Value of Hit Under Miss for SPEC

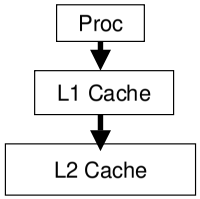


° FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26

° Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19

° 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

4. Reduce Penalty: Second-Level Cache



° L2 Equations

AMAT = Hit Time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$

Miss Penalty$_{L1}$ = Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$

AMAT = Hit Time$_{L1}$ +

Miss Rate$_{L1}$ x (Hit Time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$)

° Definitions:

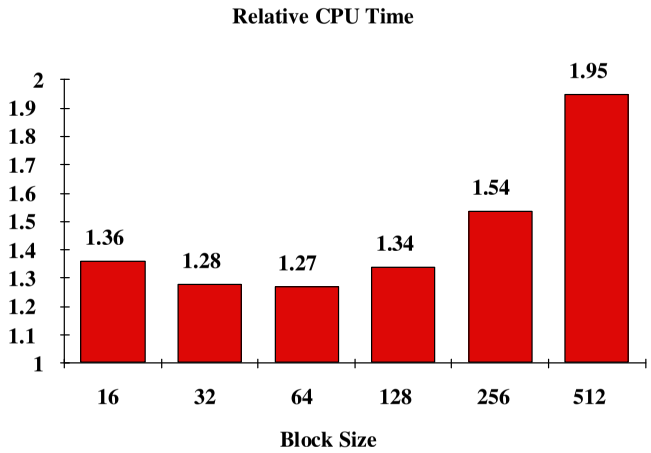
- *Local miss rate*— misses in this cache divided by the total number of memory accesses *to this cache* (Miss rate$_{L2}$)
- *Global miss rate*—misses in this cache divided by the total number of memory accesses *generated by the CPU* (Miss Rate$_{L1}$ x Miss Rate$_{L2}$)
- Global Miss Rate is what matters

Reducing Misses: which apply to L2 Cache?

° Reducing Miss Rate

1. Reduce Misses via Larger Block Size
2. Reduce Conflict Misses via Higher Associativity
3. Reducing Conflict Misses via Victim Cache
4. Reducing Misses by HW Prefetching Instr, Data
5. Reducing Misses by SW Prefetching Data
6. Reducing Capacity/Conf. Misses by Compiler Optimizations

L2 cache block size & A.M.A.T.



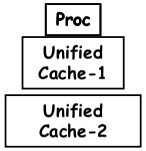
° 32KB L1, 8 byte path to memory

Improving Cache Performance (Continued)

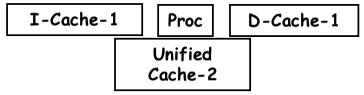
1. Reduce the miss rate,
2. *Reduce the miss penalty,* or
3. Reduce the time to hit in the cache:

Lower Associativity (+victim caching or 2nd-level cache)?
Multiple cycle Cache access (e.g. R4000)
Harvard Architecture
Careful Virtual Memory Design (rest of lecture!)

Example: Harvard Architecture



Unified



Harvard Architecture

° Sample Statistics:
- 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
- 32KB unified: Aggregate miss rate=1.99%

° Which is better (ignore L2 cache)?
- Assume 33% loads/store, hit time=1, miss time=50
- Note: *data* hit has 1 stall for unified cache (only one port)

$AMAT_{Harvard}$=(1/1.33)x(1+0.64%x50)+(0.33/1.33)x(1+6.47%x50) = 2.05
$AMAT_{Unified}$=(1/1.33)x(1+1.99%x50)+(0.33/1.33)X(1+1+1.99%x50)= 2.24

Summary #1/ 2:

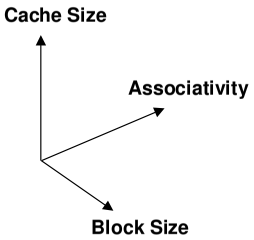
° The Principle of Locality:
- Program likely to access a relatively small portion of the address space at any instant of time.
  - Temporal Locality: Locality in Time
  - Spatial Locality: Locality in Space

° Three (+1) Major Categories of Cache Misses:
- Compulsory Misses: sad facts of life. Example: cold start misses.
- Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- Capacity Misses: increase cache size
- Coherence Misses: Caused by external processors or I/O devices

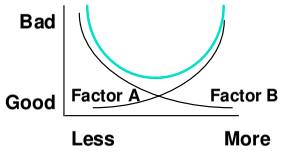
° Cache Design Space
- total size, block size, associativity
- replacement policy
- write-hit policy (write-through, write-back)
- write-miss policy

Summary #2 / 2: The Cache Design Space

° Several interacting dimensions
- cache size
- block size
- associativity
- replacement policy
- write-through vs write-back
- write allocation



° The optimal choice is a compromise
- depends on access characteristics
  - workload
  - use (I-cache, D-cache, TLB)
- depends on technology / cost



° Simplicity often wins