Caches (Writing)

Hakim Weatherspoon
CS 3410, Spring 2012
Computer Science
Cornell University

P & H Chapter 5.2-3, 5.5
Goals for Today

Cache Parameter Tradeoffs
Cache Conscious Programming
Writing to the Cache
  • Write-through vs Write back
Cache Design Tradeoffs
Cache Design

Need to determine parameters:

- Cache size
- Block size (aka line size)
- Number of ways of set-associativity (1, N, $\infty$)
- Eviction policy
- Number of levels of caching, parameters for each
- Separate I-cache from D-cache, or Unified cache
- Prefetching policies / instructions
- Write policy
A Real Example

Cache Information
- Configuration: Enabled, Not Socketed, Level 1
- Operational Mode: Write Back
- Installed Size: 128 KB
- Error Correction Type: None

Cache Information
- Configuration: Enabled, Not Socketed, Level 2
- Operational Mode: Varies With Memory Address
- Installed Size: 6144 KB
- Error Correction Type: Single-bit ECC

> cd /sys/devices/system/cpu/cpu0; grep cache/*/*

cache/index0/level:1
- type: Data
- ways_of_associativity: 8
- number_of_sets: 64
- coherency_line_size: 64
- size: 32K

cache/index1/level:1
- type: Instruction
- ways_of_associativity: 8
- number_of_sets: 64
- coherency_line_size: 64
- size: 32K

cache/index2/level:2
- type: Unified
- shared_cpu_list: 0-1
- ways_of_associativity: 24
- number_of_sets: 4096
- coherency_line_size: 64
- size: 6144K

Dual-core 3.16GHz Intel (purchased in 2011)
A Real Example

Dual 32K L1 Instruction caches
- 8-way set associative
- 64 sets
- 64 byte line size

Dual 32K L1 Data caches
- Same as above

Single 6M L2 Unified cache
- 24-way set associative (!!!)
- 4096 sets
- 64 byte line size

4GB Main memory

1TB Disk
Basic Cache Organization

Q: How to decide block size?
A: Try it and see
But: depends on cache size, workload, associativity, ...

Experimental approach!
Experimental Results

Miss rate vs Block size for different cache sizes:
- DM
- 2-way
- 8-way
- FA

Cache sizes:
- 16K
- 64K
- 256K
Tradeoffs

For a given total cache size, larger block sizes mean....

- fewer lines
- so fewer tags (and smaller tags for associative caches)
- so less overhead
- and fewer cold misses (within-block “prefetching”)

But also...

- fewer blocks available (for scattered accesses!)
- so more conflicts
- and larger miss penalty (time to fetch block)
Cache Conscious Programming
Cache Conscious Programming

// H = 12, W = 10
int A[H][W];

for(x=0; x < W; x++)
    for(y=0; y < H; y++)
        sum += A[y][x];

Every access is a cache miss!
(unless entire matrix can fit in cache)
Cache Conscious Programming

// H = 12, W = 10
int A[H][W];

for(y=0; y < H; y++)
    for(x=0; x < W; x++)
        sum += A[y][x];

Block size = 4 → 75% hit rate
Block size = 8 → 87.5% hit rate
Block size = 16 → 93.75% hit rate

And you can easily prefetch to warm the cache.
Writing with Caches
Eviction

Which cache line should be evicted from the cache to make room for a new line?

- **Direct-mapped**
  - no choice, must evict line selected by index

- **Associative caches**
  - random: select one of the lines at random
  - round-robin: similar to random
  - FIFO: replace oldest line
  - LRU: replace line that has not been used in the longest time
Q: How to write data?

If data is already in the cache...

**No-Write**
- writes invalidate the cache and go directly to memory

**Write-Through**
- writes go to main memory and cache

**Write-Back**
- CPU writes only to cache
- cache writes to main memory later (when block is evicted)
What about Stores?

Where should you write the result of a store?

• If that memory location is in the cache?
  – Send it to the cache
  – Should we also send it to memory right away? (write-through policy)
  – Wait until we kick the block out (write-back policy)

• If it is not in the cache?
  – Allocate the line (put it in the cache)? (write allocate policy)
  – Write it directly to memory without allocation? (no write allocate policy)
Write Allocation Policies
Q: How to write data?

If data is not in the cache...

Write-Allocate
• allocate a cache line for new data (and maybe write-through)

No-Write-Allocate
• ignore cache, just go to main memory
Handling Stores (Write-Through)

Using byte addresses in this example! Addr Bus = 5 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume write-allocate policy</td>
<td>Fully Associative Cache</td>
<td></td>
</tr>
<tr>
<td>LB $1 ← M[ 1 ]</td>
<td>2 cache lines</td>
<td>0 [78]</td>
</tr>
<tr>
<td>LB $2 ← M[ 7 ]</td>
<td>2 word block</td>
<td>1 [29]</td>
</tr>
<tr>
<td>SB $2 → M[ 0 ]</td>
<td>3 bit tag field</td>
<td>2 [120]</td>
</tr>
<tr>
<td>SB $1 → M[ 5 ]</td>
<td>1 bit block offset field</td>
<td>3 [123]</td>
</tr>
<tr>
<td>LB $2 ← M[ 10 ]</td>
<td>V tag</td>
<td>4 [71]</td>
</tr>
<tr>
<td>SB $1 → M[ 5 ]</td>
<td>data</td>
<td>5 [150]</td>
</tr>
<tr>
<td>SB $1 → M[ 10 ]</td>
<td>Misses: 0</td>
<td>6 [162]</td>
</tr>
<tr>
<td>$0</td>
<td></td>
<td>7 [173]</td>
</tr>
<tr>
<td>$1</td>
<td></td>
<td>8 [18]</td>
</tr>
<tr>
<td>$2</td>
<td></td>
<td>9 [21]</td>
</tr>
<tr>
<td>$3</td>
<td></td>
<td>10 [33]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 [28]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 [19]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 [200]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 [210]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 [225]</td>
</tr>
</tbody>
</table>
### Write-Through (REF 1)

#### Processor
- **Load Block (LB)**:
  - $1 \leftarrow M[1]
  - $2 \leftarrow M[7]
- **Store Block (SB)**:
  - $2 \rightarrow M[0]
  - $1 \rightarrow M[5]
  - $2 \leftarrow M[10]
  - $1 \rightarrow M[5]
  - $1 \rightarrow M[10]

#### Cache
- **V tag data**
  - Cache-000
  - Tag: 78
  - Data: 29

#### Memory
- **Misses**: 0
- **Hits**: 0

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>4</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>6</td>
<td>162</td>
</tr>
<tr>
<td>7</td>
<td>173</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>200</td>
</tr>
<tr>
<td>14</td>
<td>210</td>
</tr>
<tr>
<td>15</td>
<td>225</td>
</tr>
</tbody>
</table>
Write-Through (REF 1)

Processor

- LB $1 \leftarrow M[1]
- LB $2 \leftarrow M[7]
- SB $2 \rightarrow M[0]
- SB $1 \rightarrow M[5]
- LB $2 \leftarrow M[10]
- SB $1 \rightarrow M[5]
- SB $1 \rightarrow M[10]

Cache

- V tag data
- Misses: 1
- Hits: 0

Memory

- Misses: 1
- Hits: 0

- LB $1 \leftarrow M[1]
- LB $2 \leftarrow M[7]
- SB $2 \rightarrow M[0]
- SB $1 \rightarrow M[5]
- LB $2 \leftarrow M[10]
- SB $1 \rightarrow M[5]
- SB $1 \rightarrow M[10]
Write-Through (REF 2)

Processor:
- LB $1 ← M[1]
- LB $2 ← M[7]
- SB $2 → M[0]
- SB $1 → M[5]
- LB $2 ← M[10]
- SB $1 → M[5]
- SB $1 → M[10]

Cache:
- V tag data
  - Misses: 1
  - Hits: 0

Memory:
- Misses: 1
- Hits: 0

- 0: 78
- 1: 29
- 2: 120
- 3: 123
- 4: 71
- 5: 150
- 6: 162
- 7: 173
- 8: 18
- 9: 21
- 10: 33
- 11: 28
- 12: 19
- 13: 200
- 14: 210
- 15: 225
Write-Through (REF 2)

Processor

Memory

Cache

<table>
<thead>
<tr>
<th>V</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>000</td>
<td>78</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>162</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>173</td>
</tr>
</tbody>
</table>

Misses: 2
Hits: 0

LB $1 \leftarrow M[1]
LB $2 \leftarrow M[7]
SB $2 \rightarrow M[0]
SB $1 \rightarrow M[5]
LB $2 \leftarrow M[10]
SB $1 \rightarrow M[5]
SB $1 \rightarrow M[10]
Write-Through (REF 3)

Processor

Cache

Memory

Misses: 2

Hits: 0
Write-Through (REF 3)

Processor

Cache

Memory

Misses: 2

Hits: 1

LB $1 \leftarrow M[ 1 ]
LB $2 \leftarrow M[ 7 ]
SB $2 \rightarrow M[ 0 ]
SB $1 \rightarrow M[ 5 ]
LB $2 \leftarrow M[ 10 ]
SB $1 \rightarrow M[ 5 ]
SB $1 \rightarrow M[ 10 ]

$0
$1
$2
$3
Write-Through (REF 4)

Processor

- LB $1 ← M[ 1 ]
- LB $2 ← M[ 7 ]
- SB $2 → M[ 0 ]
- SB $1 → M[ 5 ]
- LB $2 ← M[ 10 ]
- SB $1 → M[ 5 ]
- SB $1 → M[ 10 ]

Cache

- Misses: 2
- Hits: 1

Memory

- Hits: 1
- Misses: 2

0101

000173
00129
010173
011162
10173
11173
1219
13200
14210
15225

0129

71150
Write-Through (REF 4)

Processor

Cache

Memory

Misses: 3
Hits: 1
Write-Through (REF 5)

**Processor**
- LB $1 \leftarrow M[1]
- LB $2 \leftarrow M[7]
- SB $2 \rightarrow M[0]
- SB $1 \rightarrow M[5]
- LB $2 \leftarrow M[10]
- SB $1 \rightarrow M[5]
- SB $1 \rightarrow M[10]

**Cache**
- Misses: 3
- Hits: 1

**Memory**
- Misses: 3
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
- Hits: 1
Write-Through (REF 5)

Processor

<table>
<thead>
<tr>
<th>$0</th>
<th>$1</th>
<th>$2</th>
<th>$3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>29</td>
<td>33</td>
<td></td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>V</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>010</td>
<td>71</td>
</tr>
</tbody>
</table>

Misses: 4

Hits: 1

Memory

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>
Write-Through (REF 6)

Processor:
- LB $1 \leftarrow M[1]
- LB $2 \leftarrow M[7]
- SB $2 \rightarrow M[0]
- SB $1 \rightarrow M[5]
- LB $2 \leftarrow M[10]
- SB $1 \rightarrow M[5]

Cache:
- Misses: 4
- Hits: 1

Memory:
- 0: 173
- 1: 29
- 2: 120
- 3: 123
- 4: 71
- 5: 29
- 6: 162
- 7: 173
- 8: 18
- 9: 21
- 10: 33
- 11: 28
- 12: 19
- 13: 200
- 14: 210
- 15: 225
Write-Through (REF 6)

Processor

<table>
<thead>
<tr>
<th>Operation</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB $1 \leftrightarrow M[1] M</td>
<td>V</td>
<td>tag</td>
</tr>
<tr>
<td>LB $2 \leftrightarrow M[7] M</td>
<td>1</td>
<td>101</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB $2 \leftrightarrow M[10] M</td>
<td>1</td>
<td>010</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10] H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Misses: 4
Hits: 2
Write-Through (REF 7)

Processor:
- LB $1 \leftarrow M[1]
- LB $2 \leftarrow M[7]
- SB $2 \rightarrow M[0]
- SB $1 \rightarrow M[5]
- LB $2 \leftarrow M[10]
- SB $1 \rightarrow M[5]
- SB $1 \rightarrow M[10]

Cache:
- Misses: 4
- Hits: 2

Memory:
- 0: 173
- 1: 29
- 2: 120
- 3: 123
- 4: 71
- 5: 29
- 6: 162
- 7: 173
- 8: 18
- 9: 21
- 10: 33
- 11: 28
- 12: 19
- 13: 200
- 14: 210
- 15: 225
Write-Through (REF 7)

Processor

Cache

Memory

<table>
<thead>
<tr>
<th>Misses:</th>
<th>Hits:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
How Many Memory References?

Write-through performance

Each miss (read or write) reads a block from mem
• 4 misses $\rightarrow$ 8 mem reads

Each store writes an item to mem
• 4 mem writes

Evictions don’t need to write to mem
• no need for dirty bit
Write-Through (REF 8,9)

Processor:
- LB $1 \leftarrow M[1]$
- LB $2 \leftarrow M[7]$
- SB $2 \rightarrow M[0]$
- SB $1 \rightarrow M[5]$
- LB $2 \leftarrow M[10]$
- SB $1 \rightarrow M[5]$
- SB $1 \rightarrow M[10]$

Cache:
- Misses: 4
- Hits: 3
- V tag data
  - 1 101 29
  - 1 010 71

Memory:
- 0 173
- 1 29
- 2 120
- 3 123
- 4 71
- 5 29
- 6 162
- 7 173
- 8 18
- 9 21
- 10 29
- 11 28
- 12 19
- 13 200
- 14 210
- 15 225
Write-Through (REF 8, 9)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB $1 \leftarrow M[ 1 ] M</td>
<td>V tag data</td>
<td></td>
</tr>
<tr>
<td>LB $2 \leftarrow M[ 7 ] M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $2 \rightarrow M[ 0 ] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 5 ] M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LB $2 \leftarrow M[ 10 ] M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 10 ] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 5 ] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 10 ] H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 5 ] H</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Misses: 4
Hits: 5

$0$ 29
$1$ 29
$2$ 33
$3$ 123

0 173
1 29
2 120
3 123
4 71
5 29
6 162
7 173
8 18
9 21
10 29
11 28
12 19
13 200
14 210
15 225
Write-Through vs. Write-Back

Can we also design the cache NOT to write all stores immediately to memory?

Keep the most current copy in cache, and update memory when that data is evicted (write-back policy)

Do we need to write-back all evicted lines?

No, only blocks that have been stored into (written)
### Write-Back Meta-Data

<table>
<thead>
<tr>
<th>V</th>
<th>D</th>
<th>Tag</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>...</th>
<th>Byte N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V = 1 means the line has valid data

D = 1 means the bytes are newer than main memory

#### When allocating line:
- Set V = 1, D = 0, fill in Tag and Data

#### When writing line:
- Set D = 1

#### When evicting line:
- If D = 0: just set V = 0
- If D = 1: write-back Data, then set D = 0, V = 0
Handling Stores (Write-Back)

Using **byte addresses** in this example! Addr Bus = 4 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume write-allocate policy</strong></td>
<td><strong>Fully Associative Cache</strong></td>
<td></td>
</tr>
<tr>
<td>LB $1 \leftarrow M[1]$</td>
<td><strong>V d tag data</strong></td>
<td>0</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]$</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>$2$</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>$3$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$4$</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>$5$</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>$6$</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>$7$</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>$8$</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

Using **byte addresses** in this example! Addr Bus = 4 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume write-allocate policy</strong></td>
<td><strong>Fully Associative Cache</strong></td>
<td></td>
</tr>
<tr>
<td>LB $1 \leftarrow M[1]$</td>
<td><strong>V d tag data</strong></td>
<td>0</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]$</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>$2$</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>$3$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$4$</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>$5$</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>$6$</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>$7$</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>$8$</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

Using **byte addresses** in this example! Addr Bus = 4 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume write-allocate policy</strong></td>
<td><strong>Fully Associative Cache</strong></td>
<td></td>
</tr>
<tr>
<td>LB $1 \leftarrow M[1]$</td>
<td><strong>V d tag data</strong></td>
<td>0</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]$</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>$2$</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>$3$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$4$</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>$5$</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>$6$</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>$7$</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>$8$</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

Using **byte addresses** in this example! Addr Bus = 4 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume write-allocate policy</strong></td>
<td><strong>Fully Associative Cache</strong></td>
<td></td>
</tr>
<tr>
<td>LB $1 \leftarrow M[1]$</td>
<td><strong>V d tag data</strong></td>
<td>0</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]$</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>$2$</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>$3$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$4$</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>$5$</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>$6$</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>$7$</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>$8$</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

Using **byte addresses** in this example! Addr Bus = 4 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume write-allocate policy</strong></td>
<td><strong>Fully Associative Cache</strong></td>
<td></td>
</tr>
<tr>
<td>LB $1 \leftarrow M[1]$</td>
<td><strong>V d tag data</strong></td>
<td>0</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]$</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>$2$</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>$3$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$4$</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>$5$</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>$6$</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>$7$</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>$8$</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>

Using **byte addresses** in this example! Addr Bus = 4 bits

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assume write-allocate policy</strong></td>
<td><strong>Fully Associative Cache</strong></td>
<td></td>
</tr>
<tr>
<td>LB $1 \leftarrow M[1]$</td>
<td><strong>V d tag data</strong></td>
<td>0</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]$</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]$</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]$</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]$</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]$</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>$0$</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>$1$</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>$2$</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>$3$</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>$4$</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>$5$</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>$6$</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>$7$</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>$8$</td>
<td></td>
<td>15</td>
</tr>
</tbody>
</table>
Write-Back (REF 1)

Processor

Cache

Memory

$0
$1
$2
$3

Misses: 0
Hits: 0

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>29</td>
<td>120</td>
<td>123</td>
<td>71</td>
<td>162</td>
<td>173</td>
<td>18</td>
<td>21</td>
<td>225</td>
<td>33</td>
<td>19</td>
<td>200</td>
<td>210</td>
<td>225</td>
<td></td>
</tr>
</tbody>
</table>
Write-Back (REF 1)
### Write-Back (REF 2)

**Processor**
- LB $1 \leftarrow M[1]$
- LB $2 \leftarrow M[7]$
- SB $2 \rightarrow M[0]$
- SB $1 \rightarrow M[5]$
- LB $2 \leftarrow M[10]$
- SB $1 \rightarrow M[5]$
- SB $1 \rightarrow M[10]$

**Cache**

<table>
<thead>
<tr>
<th></th>
<th>V</th>
<th>d</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>000</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Misses**: 1
- **Hits**: 0

**Memory**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>78</td>
<td>29</td>
<td>120</td>
<td>123</td>
<td>71</td>
<td>150</td>
<td>162</td>
<td>173</td>
<td>18</td>
<td>21</td>
<td>33</td>
<td>28</td>
<td>19</td>
<td>200</td>
<td>210</td>
<td>225</td>
</tr>
</tbody>
</table>
Write-Back (REF 2)

Processor:
- LB $1 \leftrightarrow M[1]$
- LB $2 \leftrightarrow M[7]$
- SB $2 \rightarrow M[0]$
- SB $1 \rightarrow M[5]$
- LB $2 \leftrightarrow M[10]$
- SB $1 \rightarrow M[5]$
- SB $1 \rightarrow M[10]$

Cache:
- V d tag data
  - 1 0 000 78
  - 1 0 011 162
  - Misses: 2
  - Hits: 0

Memory:
- 0: 78
- 1: 29
- 2: 120
- 3: 123
- 4: 71
- 5: 150
- 6: 162
- 7: 173
- 8: 18
- 9: 21
- 10: 33
- 11: 28
- 12: 19
- 13: 200
- 14: 210
- 15: 225
Write-Back (REF 3)

Processor

Cache

Memory

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>78</td>
<td>29</td>
<td>120</td>
<td>123</td>
<td>71</td>
<td>150</td>
<td>162</td>
<td>173</td>
<td>18</td>
<td>21</td>
<td>33</td>
<td>28</td>
<td>19</td>
<td>200</td>
<td>210</td>
<td>225</td>
</tr>
</tbody>
</table>

Misses: 2

Hits: 0
Write-Back (REF 3)

Processor

Cache

Memory

Misses: 2

Hits: 1

$0  29
$1  29
$2  173
$3  173

V d tag data

0  1  000  173
1  1  29
1  0  011  162
1  0  011  173

78  29  120  123  71  150  162  173  18  21  33  28  19  200  210  225
Write-Back (REF 4)

Processor

Cache

Memory

LRU

Misses: 2

Hits: 1

$0

$1

$2

$3

$0

$1

$2

$3

V d tag data

1 1 000 173

1 0 011 173
Write-Back (REF 4)

<table>
<thead>
<tr>
<th>Processor</th>
<th>Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB $1 \leftarrow M[ 1 ]</td>
<td>M</td>
<td>0 [78]</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[ 7 ]</td>
<td>M</td>
<td>1 [29]</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[ 0 ]</td>
<td>M</td>
<td>2 [120]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 5 ]</td>
<td>H</td>
<td>3 [123]</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[ 10 ]</td>
<td>M</td>
<td>4 [71]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 5 ]</td>
<td>M</td>
<td>5 [150]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[ 10 ]</td>
<td>M</td>
<td>6 [162]</td>
</tr>
<tr>
<td>$0</td>
<td>1 [1]</td>
<td>7 [173]</td>
</tr>
<tr>
<td>$1</td>
<td>1 [1]</td>
<td>8 [173]</td>
</tr>
<tr>
<td>$2</td>
<td>1 [1]</td>
<td>9 [29]</td>
</tr>
<tr>
<td>$3</td>
<td>1 [1]</td>
<td>10 [71]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 [29]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 [100]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 [200]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 [210]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 [225]</td>
</tr>
</tbody>
</table>

Misses: 3
Hits: 1
Write-Back (REF 5)

Processor

- LB $1 \leftarrow M[1]
- LB $2 \leftarrow M[7]
- SB $2 \rightarrow M[0]
- SB $1 \rightarrow M[5]
- LB $2 \leftarrow M[10]
- SB $1 \rightarrow M[5]
- SB $1 \rightarrow M[10]

$0$
- $29$

$1$
- $33$

$2$
- $173$

$3$
- $1$

Cache

V d tag data

- $1 1 000 173$
- $1 1 010 71$
- $1 1 010 29$

Misses: 3

Hits: 1

Memory

- Misses: 3
- Hits: 1

- $0$
  - 78
- $1$
  - 29
- $2$
  - 120
- $3$
  - 123
- $4$
  - 71
- $5$
  - 150
- $6$
  - 162
- $7$
  - 173
- $8$
  - 18
- $9$
  - 21
- $10$
  - 33
- $11$
  - 28
- $12$
  - 19
- $13$
  - 200
- $14$
  - 210
- $15$
  - 225
Write-Back (REF 5)

Processor

Memory

Cache

Misses: 3
Hits: 1
Write-Back (REF 5)

**Processor**
- LB $1 \leftarrow M[1]$
- LB $2 \leftarrow M[7]$
- SB $2 \rightarrow M[0]$
- SB $1 \rightarrow M[5]$
- LB $2 \leftarrow M[10]$
- SB $1 \rightarrow M[5]$
- SB $1 \rightarrow M[10]$

**Cache**
- Misses: 4
- Hits: 1
- V d tag data
  - $1 \ 0 \ 101 \ 33$
  - $1 \ 1 \ 010 \ 71$
  - $1 \ 1 \ 010 \ 29$

**Memory**
- 0: 78
- 1: 29
- 2: 120
- 3: 123
- 4: 71
- 5: 150
- 6: 162
- 7: 173
- 8: 18
- 9: 21
- 10: 33
- 11: 28
- 12: 19
- 13: 200
- 14: 210
- 15: 225
Write-Back (REF 6)

Processor

Cache

Memory

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

Misses: 4

Hits: 1
### Write-Back (REF 6)

**Processor**

- LB $1 \leftarrow M[ 1 ]$
- LB $2 \leftarrow M[ 7 ]$
- SB $2 \rightarrow M[ 0 ]$
- SB $1 \rightarrow M[ 5 ]$
- LB $2 \leftarrow M[ 10 ]$
- SB $1 \rightarrow M[ 5 ]$
- SB $1 \rightarrow M[ 10 ]$

**Memory**

<table>
<thead>
<tr>
<th>Page</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>4</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>6</td>
<td>162</td>
</tr>
<tr>
<td>7</td>
<td>173</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>200</td>
</tr>
<tr>
<td>14</td>
<td>210</td>
</tr>
<tr>
<td>15</td>
<td>225</td>
</tr>
</tbody>
</table>

**Cache**

- V d tag data
- Misses: 4
- Hits: 2
Write-Back (REF 7)
Write-Back (REF 7)

Processor:
- LB $1 \leftarrow M[1]$
- LB $2 \leftarrow M[7]$
- SB $2 \rightarrow M[0]$
- SB $1 \rightarrow M[5]$
- LB $2 \leftarrow M[10]$
- SB $1 \rightarrow M[5]$
- SB $1 \rightarrow M[10]$

Cache:
- V d tag data
- Misses: 4
- Hits: 3

Memory:
- 0: 78
- 1: 29
- 2: 120
- 3: 123
- 4: 71
- 5: 150
- 6: 162
- 7: 173
- 8: 18
- 9: 21
- 10: 33
- 11: 28
- 12: 19
- 13: 200
- 14: 210
- 15: 225
How Many Memory References?

Write-back performance

Each miss (read or write) reads a block from mem
- 4 misses $\rightarrow$ 8 mem reads

Some evictions write a block to mem
- 1 dirty eviction $\rightarrow$ 2 mem writes
- (+ 2 dirty evictions later $\rightarrow$ +4 mem writes)
How many memory references?

Each miss reads a block
  Two words in this cache
Each evicted dirty cache line writes a block
Total reads: six words
Total writes: 4/6 words (after final eviction)
Write-Back (REF 8,9)

Processor

<table>
<thead>
<tr>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB $1 \leftarrow M[1]</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[7]</td>
</tr>
<tr>
<td>SB $2 \rightarrow M[0]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]</td>
</tr>
<tr>
<td>LB $2 \leftarrow M[10]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[5]</td>
</tr>
<tr>
<td>SB $1 \rightarrow M[10]</td>
</tr>
<tr>
<td>$0</td>
</tr>
<tr>
<td>$1</td>
</tr>
<tr>
<td>$2</td>
</tr>
<tr>
<td>$3</td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>V</th>
<th>d</th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>101</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>010</td>
<td>71</td>
</tr>
</tbody>
</table>

Misses: 4

Hits: 5

Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>1</td>
<td>29</td>
</tr>
<tr>
<td>2</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>123</td>
</tr>
<tr>
<td>4</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>150</td>
</tr>
<tr>
<td>6</td>
<td>162</td>
</tr>
<tr>
<td>7</td>
<td>173</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>28</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>13</td>
<td>200</td>
</tr>
<tr>
<td>14</td>
<td>210</td>
</tr>
<tr>
<td>15</td>
<td>225</td>
</tr>
</tbody>
</table>
How Many Memory References?

Write-back performance

Each miss (read or write) reads a block from mem

- 4 misses $\rightarrow$ 8 mem reads

Some evictions write a block to mem

- 1 dirty eviction $\rightarrow$ 2 mem writes
- (+ 2 dirty evictions later $\rightarrow$ +4 mem writes)

By comparison write-through was

- Reads: eight words
- Writes: 4/6/8 etc words
- Write-through or Write-back?
Write-through vs. Write-back

Write-through is slower
• But cleaner (memory always consistent)

Write-back is faster
• But complicated when multi cores sharing memory
Performance: Write-back versus Write-through

Assume: large associative cache, 16-byte lines

for (i=1; i<n; i++)
    A[0] += A[i];

for (i=0; i<n; i++)
    B[i] = A[i]
Performance Tradeoffs

Q: Hit time: write-through vs. write-back?
A: Write-through slower on writes.

Q: Miss penalty: write-through vs. write-back?
A: Write-back slower on evictions.
Write Buffering

Q: Writes to main memory are **slow**!

A: Use a **write-back buffer**
  - A small queue holding dirty lines
  - Add to end upon eviction
  - Remove from front upon completion

Q: What does it help?

A: short bursts of writes (but not sustained writes)

A: fast eviction reduces miss penalty
Write-through vs. Write-back

Write-through is slower

• But simpler (memory always consistent)

Write-back is almost always faster

• write-back buffer hides large eviction cost
• But what about multiple cores with separate caches but sharing memory?

Write-back requires a cache coherency protocol

• Inconsistent views of memory
• Need to “snoop” in each other’s caches
• Extremely complex protocols, very hard to get right
Cache-coherency protocol

- May need to **snoop** on other CPU’s cache activity
- **Invalidate** cache line when other CPU writes
- **Flush** write-back caches before other CPU reads
- Or the reverse: Before writing/reading...
- Extremely complex protocols, very hard to get right
Prelim2 results

- Mean 68.9 (median 71), standard deviation 13.0

- Prelims available in Upson 360 after today
- Regrade requires written request
  - **Whole test is regraded**
Administrivia

Lab3 due next Monday, April 9\textsuperscript{th}

HW5 due next Tuesday, April 10\textsuperscript{th}
Summary

Caching assumptions

• small working set: 90/10 rule
• can predict future: spatial & temporal locality

Benefits

• (big & fast) built from (big & slow) + (small & fast)

Tradeoffs:

associativity, line size, hit cost, miss penalty, hit rate
Summary

Memory performance matters!

- often more than CPU performance
- ... because it is the bottleneck, and not improving much
- ... because most programs move a LOT of data

Design space is huge

- Gambling against program behavior
- Cuts across all layers:
  users $\rightarrow$ programs $\rightarrow$ os $\rightarrow$ hardware

Multi-core / Multi-Processor is complicated

- Inconsistent views of memory
- Extremely complex protocols, very hard to get right