

CS 316: Caches-III

Kavita Bala
Fall 2007
Computer Science
Cornell University

Announcements

- HW 1 grades are out
- HW 2 is due on Friday
- PA 4 is out on Friday

Cache Organization

- Three common designs
 - Fully associative: Block can be anywhere in the cache
 - Direct mapped: Block can only be in one line in the cache
 - Set-associative: Block can be in a few (2 to 8) places in the cache

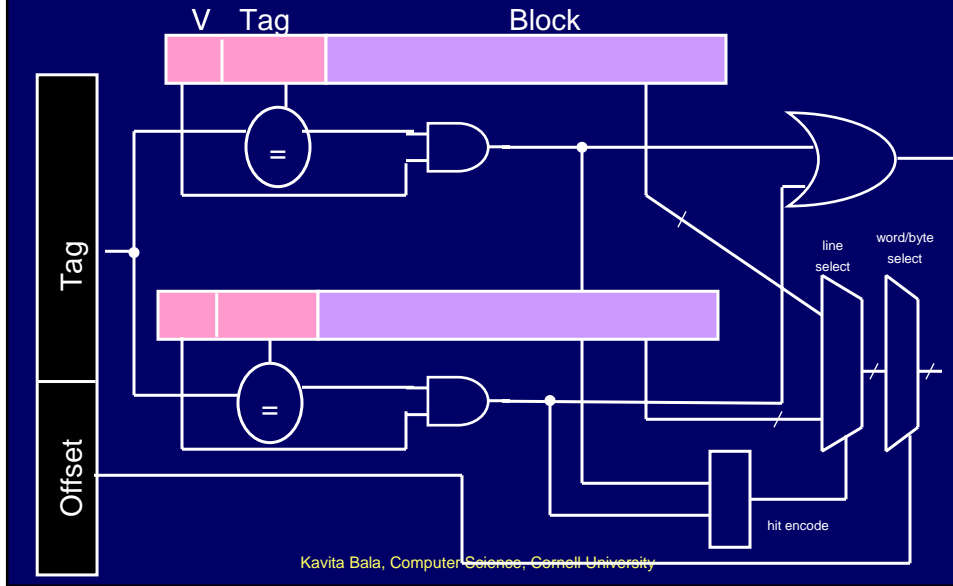
Kavita Bala, Computer Science, Cornell University

Misses

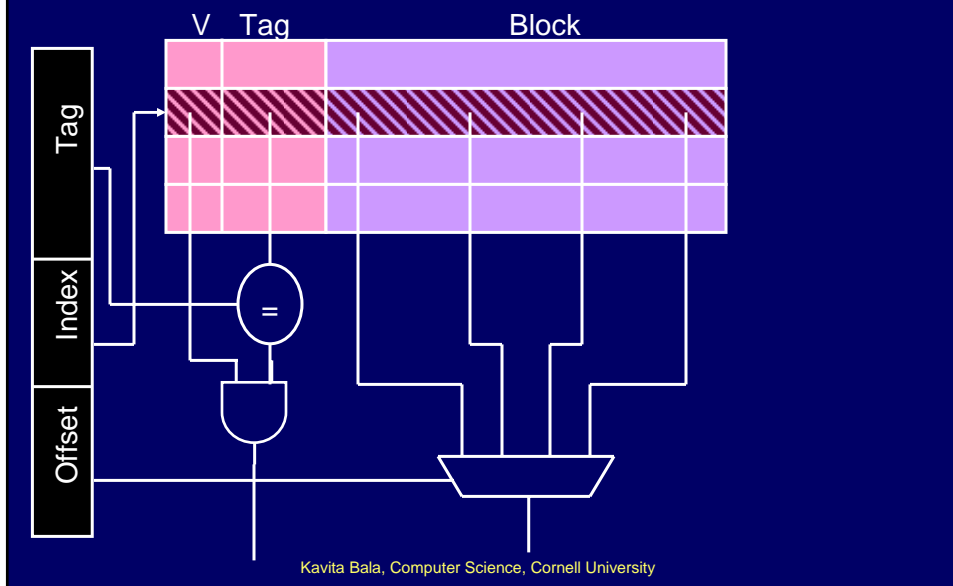
- Three types of misses
 - Cold
 - The line is being referenced for the first time
 - Capacity
 - The line was evicted because the cache was not large enough
 - Conflict
 - The line was evicted because of another access whose index conflicted

Kavita Bala, Computer Science, Cornell University

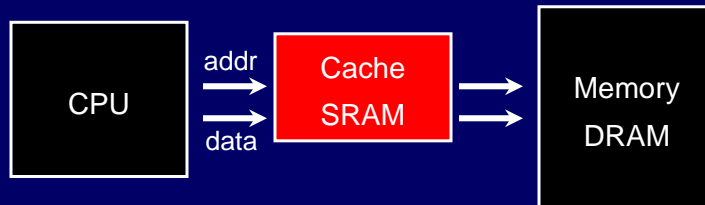
Fully Associative Cache



Comparison: Direct Mapped Cache



Cache Writes



- No-Write
 - writes invalidate the cache and go to memory
- Write-Through
 - writes go to cache and to main memory
- Write-Back
 - write cache, write main memory only when block is evicted

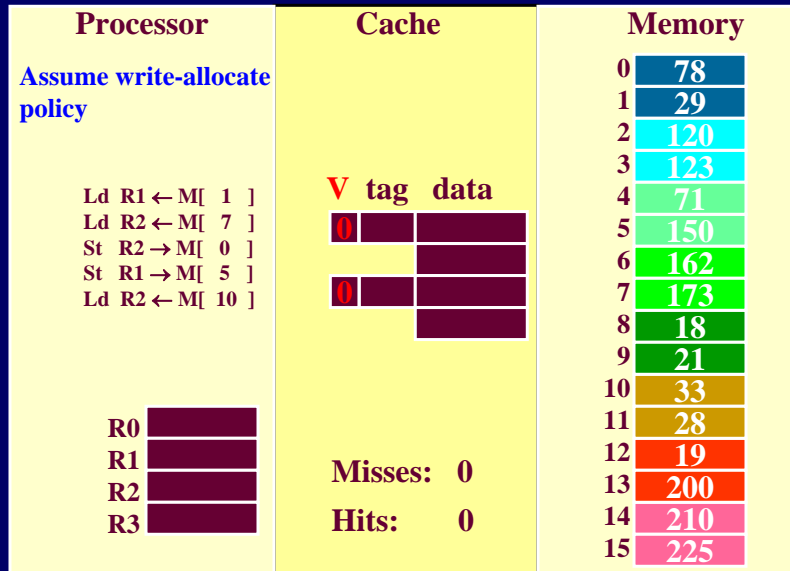
Kavita Bala, Computer Science, Cornell University

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)

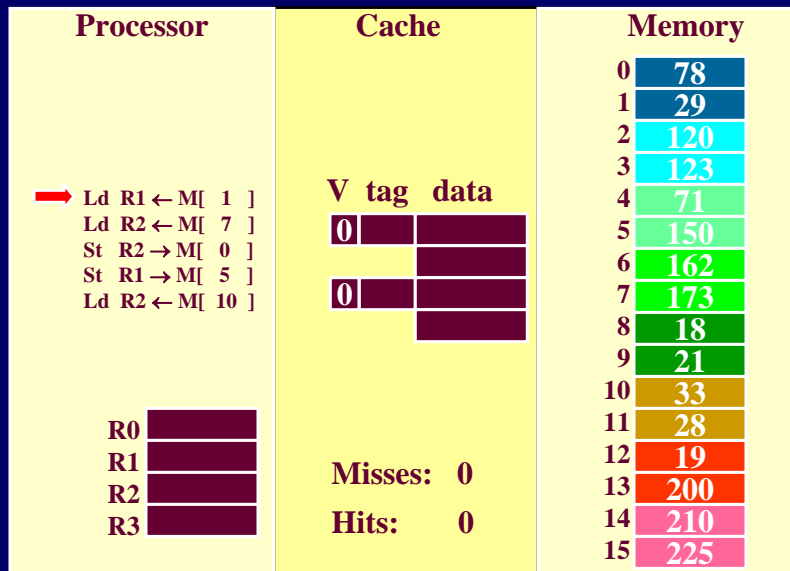
Kavita Bala, Computer Science, Cornell University

Handling Stores (Write-Through)



Kavita Bala, Computer Science, Cornell University

Write-Through (REF 1)



Kavita Bala, Computer Science, Cornell University

Write-Through (REF 1)

Processor	Cache	Memory																																									
<p>→ Ld R1 ← M[1]</p> <p>Ld R2 ← M[7]</p> <p>St R2 → M[0]</p> <p>St R1 → M[5]</p> <p>Ld R2 ← M[10]</p>	<p>V tag data</p> <table border="1"> <tr> <td>1</td> <td>0</td> <td>78</td> </tr> <tr> <td></td> <td></td> <td>29</td> </tr> <tr> <td>lru</td> <td>0</td> <td></td> </tr> </table> <p>Misses: 1</p> <p>Hits: 0</p>	1	0	78			29	lru	0		<table border="1"> <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> </table>	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
1	0	78																																									
		29																																									
lru	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																																										
<p>R0</p> <p>R1 29</p> <p>R2</p> <p>R3</p>																																											

Kavita Bala, Computer Science, Cornell University

Write-Through (REF 2)

Processor	Cache	Memory																																									
<p>Ld R1 ← M[1]</p> <p>→ Ld R2 ← M[7]</p> <p>St R2 → M[0]</p> <p>St R1 → M[5]</p> <p>Ld R2 ← M[10]</p>	<p>V tag data</p> <table border="1"> <tr> <td>1</td> <td>0</td> <td>78</td> </tr> <tr> <td></td> <td></td> <td>29</td> </tr> <tr> <td>lru</td> <td>0</td> <td></td> </tr> </table> <p>Misses: 1</p> <p>Hits: 0</p>	1	0	78			29	lru	0		<table border="1"> <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> </table>	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
1	0	78																																									
		29																																									
lru	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																																										
<p>R0</p> <p>R1 29</p> <p>R2</p> <p>R3</p>																																											

Kavita Bala, Computer Science, Cornell University

Write-Through (REF 2)

Processor	Cache	Memory
Ld R1 ← M[1]	V tag data	0 78
→ Ld R2 ← M[7]	lru 1 0	1 29
St R2 → M[0]		2 120
St R1 → M[5]	1 3	3 123
Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 173	Misses: 2	12 19
R3	Hits: 0	13 200
		14 210
		15 225

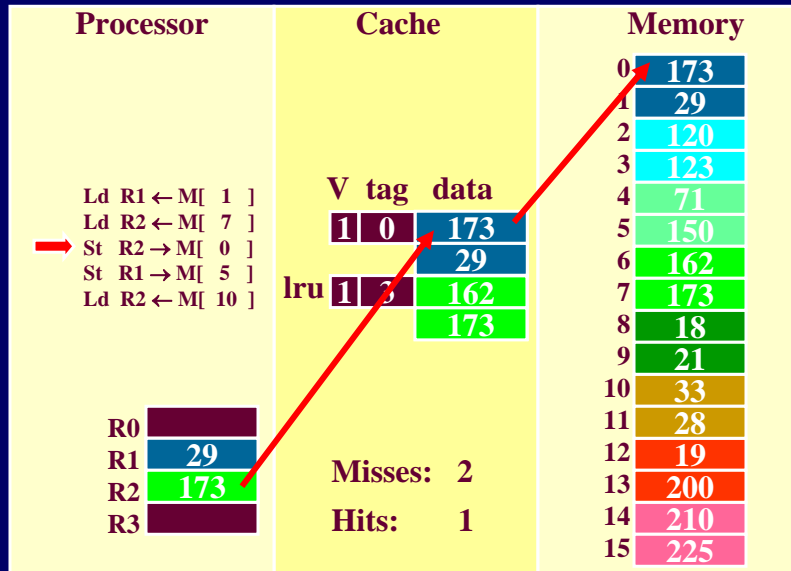
Kavita Bala, Computer Science, Cornell University

Write-Through (REF 3)

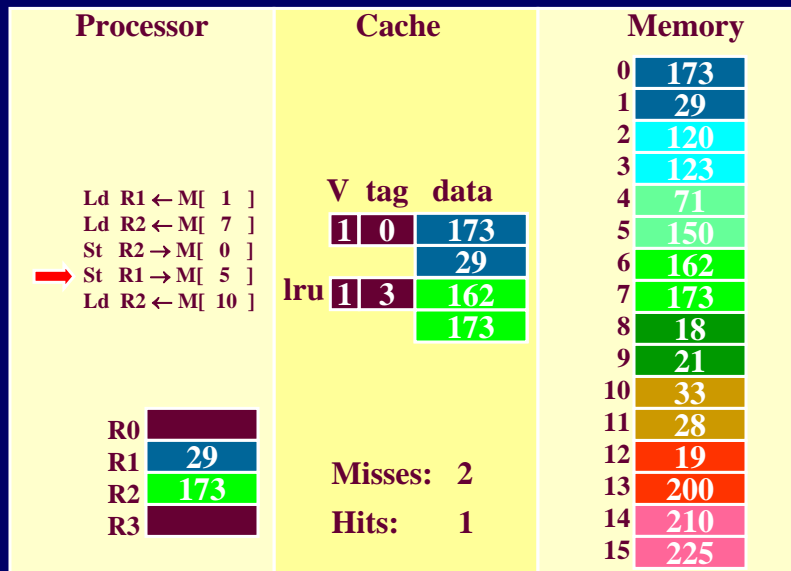
Processor	Cache	Memory
Ld R1 ← M[1]	V tag data	0 78
Ld R2 ← M[7]	lru 1 0	1 29
→ St R2 → M[0]		2 120
St R1 → M[5]	1 3	3 123
Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 173	Misses: 2	12 19
R3	Hits: 0	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

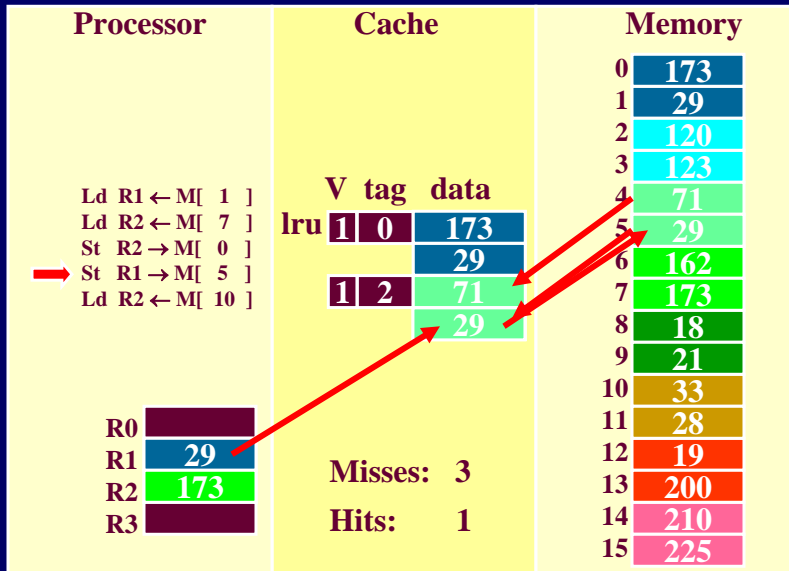
Write-Through (REF 3)



Write-Through (REF 4)

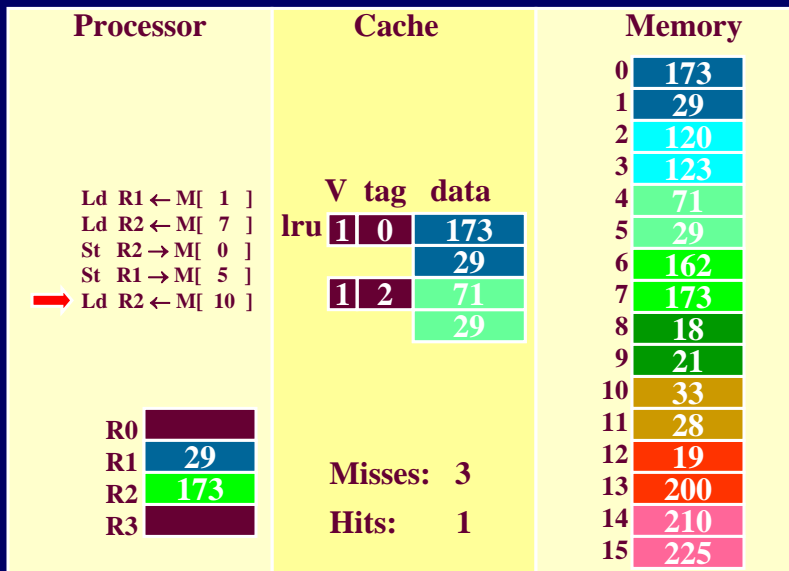


Write-Through (REF 4)



Kavita Bala, Computer Science, Cornell University

Write-Through (REF 5)



Kavita Bala, Computer Science, Cornell University

Write-Through (REF 5)

Processor	Cache	Memory
Ld R1 ← M[1]	V tag data	0 173
Ld R2 ← M[7]	1 5 33	1 29
St R2 → M[0]	28	2 120
St R1 → M[5]	lru 1 2 71	3 123
→ Ld R2 ← M[10]	29	4 71
		5 29
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 33	Misses: 4	12 19
R3	Hits: 1	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Through (REF 6,7,8)

Processor	Cache	Memory
Ld R1 ← M[1]	V tag data	0 173
Ld R2 ← M[7]	1 5 33	1 29
St R2 → M[0]	28	2 120
St R1 → M[5]	lru 1 2 71	3 123
Ld R2 ← M[10]	29	4 71
St R0 → M[5]		5 29
St R3 → M[5]		6 162
→ St R2 → M[5]		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 33	Misses: 4	12 19
R3	Hits: 1+3	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

How Many Memory References?

- Each miss reads a block (only two words in this cache)
- Each store writes a word (or a block, depends)
- Total reads: eight words
- Total writes:
 - Before last 3 stores: two words
 - After last 3 stores: five

Kavita Bala, Computer Science, Cornell University

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)
- Keep a “**dirty bit**”, reset when the line is allocated, set when the block is written
- If a block is “dirty” when evicted, write its data back into memory

Kavita Bala, Computer Science, Cornell University

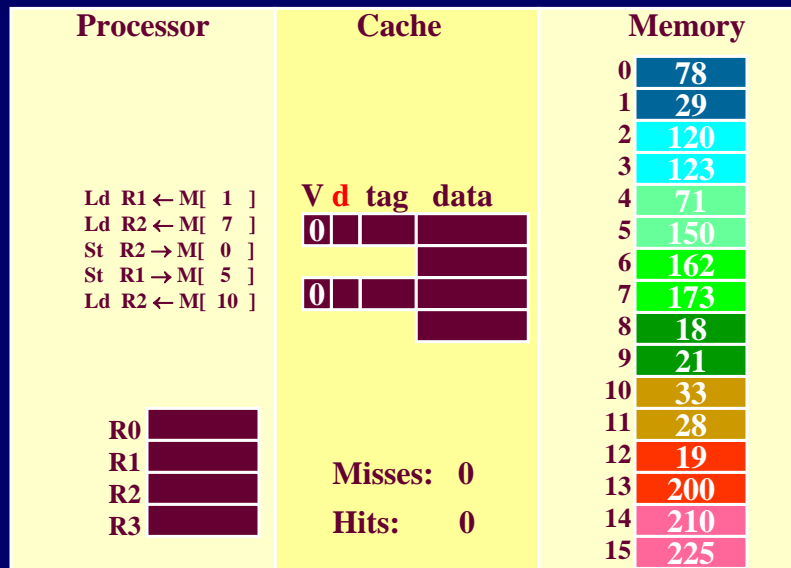
Dirty Bits and Write-Back Buffers

V	D	Tag	Data Byte 0, Byte 1 ... Byte N	Line
1	0			
1	1			
1	0			

- Dirty bits indicate which lines have been written
- Dirty bits enable the cache to handle multiple writes to the same cache line without having to go to memory
- Write-back buffer
 - A queue where dirty lines are placed

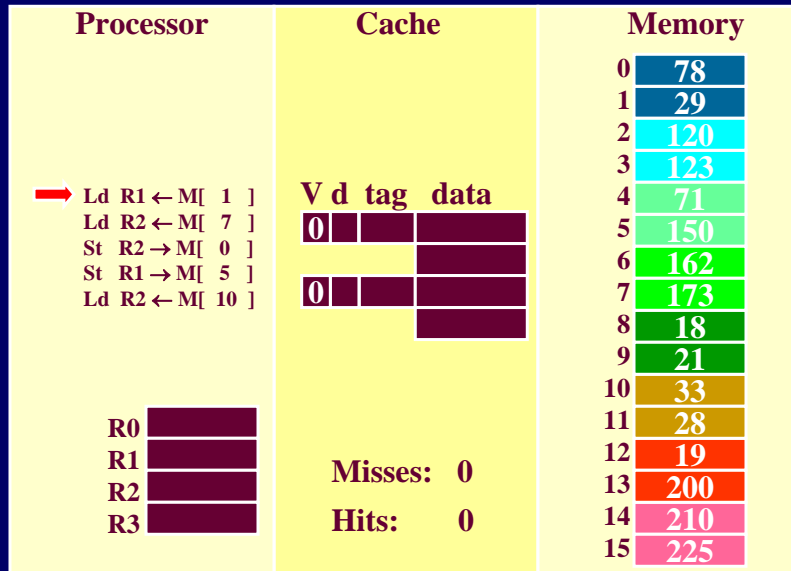
Kavita Bala, Computer Science, Cornell University

Handling Stores (Write-Back)



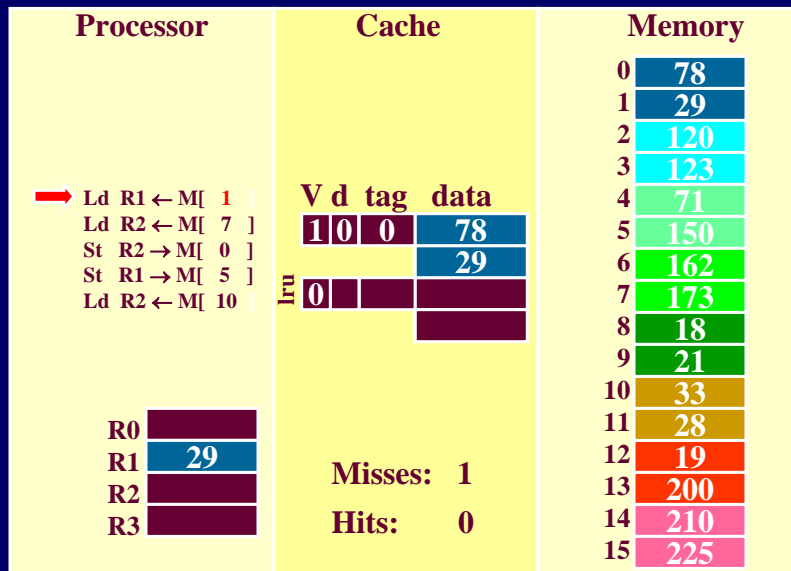
Kavita Bala, Computer Science, Cornell University

Write-Back (REF 1)



Kavita Bala, Computer Science, Cornell University

Write-Back (REF 1)



Kavita Bala, Computer Science, Cornell University

Write-Back (REF 2)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data 1 0 0 78	0 78
→ Ld R2 ← M[7]		29
St R2 → M[0]	lru 0	3 120
St R1 → M[5]		4 71
Ld R2 ← M[10]		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2	Misses: 1	12 19
R3	Hits: 0	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 2)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data 1 0 0 78	0 78
→ Ld R2 ← M[7]		29
St R2 → M[0]	lru 1 0 3	3 120
St R1 → M[5]		4 71
Ld R2 ← M[10]		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 173	Misses: 2	12 19
R3	Hits: 0	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 3)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data lru 1 0 0 78	0 78
Ld R2 ← M[7]		1 29
St R2 → M[0]	1 0 3 162	2 120
St R1 → M[5]	173	3 123
Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 173	Misses: 2	12 19
R3	Hits: 0	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 3)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data lru 1 1 0 173	0 78
Ld R2 ← M[7]		1 29
St R2 → M[0]	1 0 3 162	2 120
St R1 → M[5]	173	3 123
Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 173	Misses: 2	12 19
R3	Hits: 1	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 4)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data	0 78
Ld R2 ← M[7]		1 29
St R2 → M[0]	1 1 0 173	2 120
→ St R1 → M[5]	lru 1 0 3 162	3 123
Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29		11 28
R2 173	Misses: 2	12 19
R3	Hits: 1	13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 4)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data	0 78
Ld R2 ← M[7]		lru 1 1 0 173
St R2 → M[0]		2 120
→ St R1 → M[5]	1 1 3 71	3 123
Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
R0		10 33
R1 29	Misses: 3	11 28
R2 173	Hits: 1	12 19
R3		13 200
		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 5)

Processor	Cache	Memory																				
Ld R1 ← M[1]	<table border="1"> <thead> <tr> <th>lru</th> <th>V</th> <th>d</th> <th>tag</th> <th>data</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>0</td> <td></td> <td>173</td> </tr> <tr> <td>1</td> <td>1</td> <td>3</td> <td></td> <td>71</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>29</td> </tr> </tbody> </table>	lru	V	d	tag	data	1	1	0		173	1	1	3		71					29	0 78
lru		V	d	tag	data																	
1		1	0		173																	
1		1	3		71																	
				29																		
Ld R2 ← M[7]		1 29																				
St R2 → M[0]		2 120																				
St R1 → M[5]		3 123																				
→ Ld R2 ← M[10]		4 71																				
		5 150																				
		6 162																				
		7 173																				
		8 18																				
		9 21																				
R0		10 33																				
R1 29		11 28																				
R2 173	Misses: 3	12 19																				
R3	Hits: 1	13 200																				
		14 210																				
		15 225																				

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 5)

Processor	Cache	Memory																				
Ld R1 ← M[1]	<table border="1"> <thead> <tr> <th>lru</th> <th>V</th> <th>d</th> <th>tag</th> <th>data</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>1</td> <td>0</td> <td></td> <td>173</td> </tr> <tr> <td>1</td> <td>1</td> <td>3</td> <td></td> <td>71</td> </tr> <tr> <td></td> <td></td> <td></td> <td></td> <td>29</td> </tr> </tbody> </table>	lru	V	d	tag	data	1	1	0		173	1	1	3		71					29	0 173
lru		V	d	tag	data																	
1		1	0		173																	
1		1	3		71																	
				29																		
Ld R2 ← M[7]		1 29																				
St R2 → M[0]		2 120																				
St R1 → M[5]		3 123																				
→ Ld R2 ← M[10]		4 71																				
		5 150																				
		6 162																				
		7 173																				
		8 18																				
		9 21																				
R0		10 33																				
R1 29		11 28																				
R2 173	Misses: 4	12 19																				
R3	Hits: 1	13 200																				
		14 210																				
		15 225																				

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 5)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data	0 78
Ld R2 ← M[7]	1 0 5 33	1 29
St R2 → M[0]	lru 1 1 3 71	2 120
St R1 → M[5]		3 123
→ Ld R2 ← M[10]		4 71
		5 150
		6 162
		7 173
		8 18
		9 21
		10 33
		11 28
R0	Misses: 4	12 19
R1 29	Hits: 1	13 200
R2 33		14 210
R3		15 225

Kavita Bala, Computer Science, Cornell University

Write-Back (REF 6, 7, 8)

Processor	Cache	Memory
Ld R1 ← M[1]	V d tag data	0 78
Ld R2 ← M[7]	1 0 5 33	1 29
St R2 → M[0]	lru 1 1 3 71	2 120
St R1 → M[5]		3 123
Ld R2 ← M[10]		4 71
St R0 → M[5]		5 150
St R3 → M[5]		6 162
→ St R2 → M[5]		7 173
		8 18
		9 21
		10 33
R0 11	Misses: 4	11 28
R1 29	Hits: 2+1+1	12 19
R2 33		13 200
R3		14 210
		15 225

Kavita Bala, Computer Science, Cornell University

How many memory references?

- Each miss reads a block
Two words in this cache
- Each evicted dirty cache line writes a block
- Total reads: eight words
- Total writes: four after final eviction

Choose write-back or write-through?

Kavita Bala, Computer Science, Cornell University

Cache Design

- Need to determine parameters
 - Block size
 - Number of ways
 - Eviction policy
 - Write policy
 - Separate I-cache from D-cache

Kavita Bala, Computer Science, Cornell University

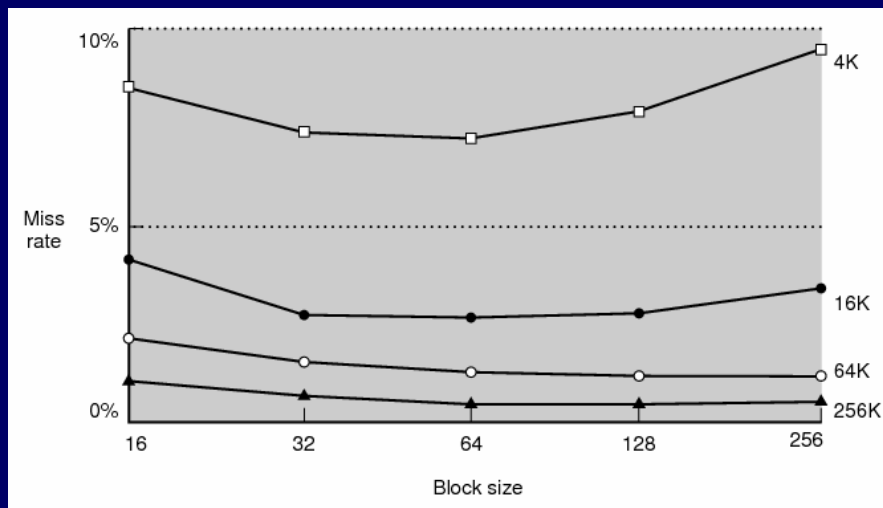
Basic Cache Organization

Decide on the block size

- How? Simulate lots of different block sizes and see which one gives the best performance
- Most systems use a block size between 32 bytes and 128 bytes



Kavita Bala, Computer Science, Cornell University



Kavita Bala, Computer Science, Cornell University

Tradeoff

- Larger sizes reduce the overhead by
 - Reducing the number of tags
 - Reducing the size of each tag
- But
 - Have fewer blocks available
 - And the time to fetch the block on a miss is longer

Kavita Bala, Computer Science, Cornell University

Short Performance Discussion

- Complicated
 - Time from start-to-end (wall-clock time)
 - System time, user time
 - CPI (Cycles per instruction)
- Ideal CPI?

Kavita Bala, Computer Science, Cornell University

Cache Performance

- Consider hit (H) and miss ratio (M)
- $H \times AT_{\text{cache}} + M \times AT_{\text{memory}}$
- Hit rate = 1 – Miss rate
- Access Time is given in cycles
- Ratio of Access times, 1:50

- 90% : $.90 + .1 \times 50 = 5.9$
- 95% : $.95 + .05 \times 50 = .95 + 2.5 = 3.45$
- 99% : $.99 + .01 \times 50 = 1.49$
- 99.9%: $.999 + .001 \times 50 = 0.999 + 0.05 = 1.049$

Kavita Bala, Computer Science, Cornell University

Cache Hit/Miss Rate

- Consider processor that is 2x times faster
 - But memory is same speed

- Since AT is access time in terms of cycle time: it doubles 2x
- $H \times AT_{\text{cache}} + M \times AT_{\text{memory}}$
- Ratio of Access times, 1:100
- 99% : $.99 + .01 \times 100 = 1.99$

Kavita Bala, Computer Science, Cornell University

Cache Hit/Miss Rate

- Original is 1GHz, 1ns is cycle time
- CPI (cycles per instruction): 1.49
- Therefore, 1.49 ns for each instruction

- New is 2GHz, 0.5 ns is cycle time.
- CPI: 1.99, 0.5ns. 0.995 ns for each instruction.

- So it doesn't go to 0.745 ns for each instruction.
- Speedup is 1.5x (not 2x)

Kavita Bala, Computer Science, Cornell University

Misses

- Three types of misses
 - Cold
 - The line is being referenced for the first time
 - Capacity
 - The line was evicted because the cache was not large enough
 - Conflict
 - The line was evicted because of another access whose index conflicted

Kavita Bala, Computer Science, Cornell University

Cache Conscious Programming

```
int a[NCOL][NROW];
int sum = 0;

for(j = 0; j < NCOL; ++j)
  for(i = 0; i < NROW; ++i)
    sum += a[j][i];
```

- Speed up this program!

Kavita Bala, Computer Science, Cornell University

Cache Conscious Programming

```
int a[NCOL][NROW];
int sum = 0;

for(j = 0; j < NCOL; ++j)
  for(i = 0; i < NROW; ++i)
    sum += a[j][i];
```

1	11								
2	12								
3	13								
4	14								
5	15								
6									
7									
8									
9									
10									

- Every access is a cache miss!

Kavita Bala, Computer Science, Cornell University

Cache Conscious Programming

```
int a[NCOL][NROW];
int sum = 0;

for(i = 0; i < NROW; ++i)
    for(j = 0; j < NCOL; ++j)
        sum += a[j][i];
```

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15					

- Same program, trivial transformation, 3 out of four accesses hit in the cache