

## Caching (part 2)

1

## exercise

address (hex)	result
00000000 (00)	
00000001 (01)	
01100011 (63)	
01100001 (61)	
01100010 (62)	
00000000 (00)	
01100100 (64)	

4 byte blocks, 4 sets

index	valid	tag	value
00			
01			
10			
11			

3

## exercise

4 byte blocks, 4 sets

address (hex)	result	index	valid	tag	value
00000000 (00)		00			
00000001 (01)					
01100011 (63)		01			
01100001 (61)					
01100010 (62)		10			
00000000 (00)					
01100100 (64)		11			

how is the address 61 (01100001) split up into tag/index/offset?

$b$  block offset bits;  
 $B = 2^b$  byte block size;  
 $s$  set index bits;  $S = 2^s$  sets ;  
 $t = m - (s + b)$  tag bits (leftover)

3

## exercise

4 byte blocks, 4 sets

address (hex)	result
00000000 (00)	
00000001 (01)	
01100011 (63)	
01100001 (61)	
01100010 (62)	
00000000 (00)	
01100100 (64)	

index	valid	tag	value
00			
01			
10			
11			

$m = 8$  bit addresses  
 $S = 4 = 2^s$  sets  
 $s = 2$  (set) index bits

$B = 4 = 2^b$  byte block size  
 $b = 2$  (block) offset bits  
 $t = m - (s + b) = 4$  tag bits

3

## exercise

4 byte blocks, 4 sets

address (hex)	result	index	valid	tag	value
00000000 (00)		00			
00000001 (01)		00			
01100011 (63)		01			
01100001 (61)		01			
01100010 (62)		10			
00000000 (00)		10			
01100100 (64)		11			

tag index offset

$m = 8$  bit addresses  
 $S = 4 = 2^s$  sets  
 $s = 2$  (set) index bits

$B = 4 = 2^b$  byte block size  
 $b = 2$  (block) offset bits  
 $t = m - (s + b) = 4$  tag bits

3

## exercise

4 byte blocks, 4 sets

address (hex)	result	index	valid	tag	value
00000000 (00)		00			
00000001 (01)		00			
01100011 (63)		01			
01100001 (61)		01			
01100010 (62)		10			
00000000 (00)		10			
01100100 (64)		11			

tag index offset

exercise: how many accesses are hits?

3

## example access pattern (1)

2 byte blocks, 4 sets

address (hex)	result	index	valid	tag	value
00000000 (00)	miss	00	1	000000	mem[0x00]
00000001 (01)	hit	00			mem[0x01]
01100011 (63)	miss	01	1	011000	mem[0x62]
01100001 (61)	miss	01			mem[0x63]
01100010 (62)	hit	10	1	011000	mem[0x64]
00000000 (00)	miss	10			mem[0x65]
01100100 (64)	miss	11	0		

miss caused by conflict

tag index offset

$m = 8$  addresses  
 $S = 4 = 2^s$  sets  
 $s = 2$  (set) index bits

$B = 2 = 2^b$  byte block size  
 $b = 1$  (block) offset bits  
 $t = m - (s + b) = 5$  tag bits

4

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	0			0		
1	0			0		

multiple places to put values with same index  
 avoid conflict misses

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	0		set 0	0		
1	0		set 1	0		

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	0			0		
1	0			0		

way 0                      way 1

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	0			0		
1	0			0		

$m = 8$  bit addresses       $B = 2 = 2^b$  byte block size  
 $S = 2 = 2^s$  sets               $b = 1$  (block) offset bits  
 $s = 1$  (set) index bits       $t = m - (s + b) = 6$  tag bits

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	0		
1	0			0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	
01100011 (63)	
01100001 (61)	
01100010 (62)	
00000000 (00)	
01100100 (64)	

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	0		
1	0			0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	
01100001 (61)	
01100010 (62)	
00000000 (00)	
01100100 (64)	

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	0		
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	
01100010 (62)	
00000000 (00)	
01100100 (64)	

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	
00000000 (00)	
01100100 (64)	

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	
01100100 (64)	

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

needs to replace block in set 0!

tag index offset

5

## adding associativity

2-way set associative, 2 byte blocks, 2 sets

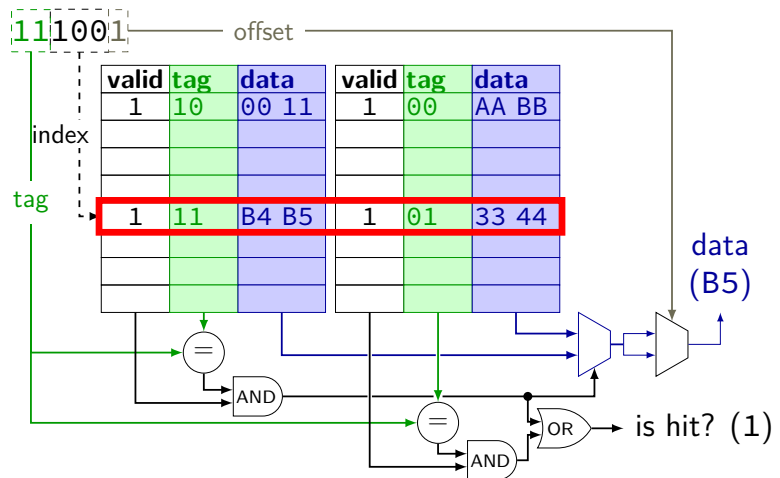
index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

tag index offset

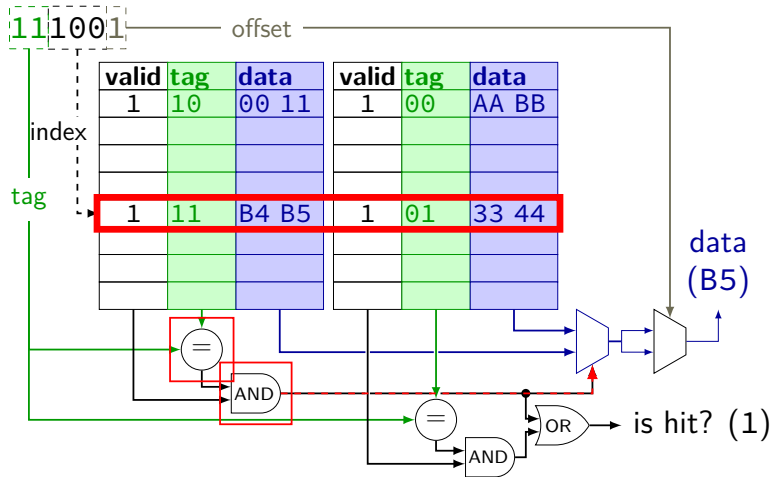
5

## cache operation (associative)



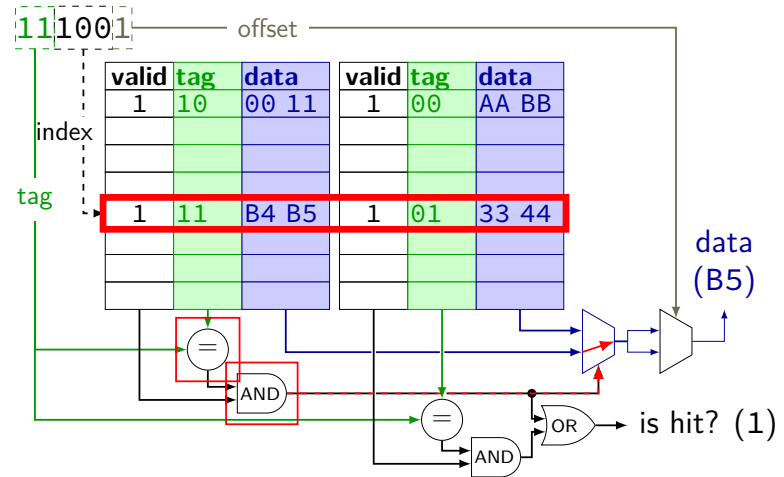
6

## cache operation (associative)



6

## cache operation (associative)



6

## associative lookup possibilities

none of the blocks for the index are valid

none of the valid blocks for the index match the tag  
something else is stored there

one of the blocks for the index is valid and matches the tag

7

## associativity terminology

**direct-mapped** — one block per set

**$E$ -way set associative** —  $E$  blocks per set  
 $E$  ways in the cache

**fully associative** — one set total (everything in one set)

8

## Tag-Index-Offset formulas (complete)

$m$	memory addresses bits (Y86-64: 64)
$E$	number of blocks per set ("ways")
$S = 2^s$	number of sets
$s$	(set) index bits
$B = 2^b$	block size
$b$	(block) offset bits
$t = m - (s + b)$	tag bits
$C = B \times S \times E$	cache size (excluding metadata)

9

## Tag-Index-Offset exercise

$m$	memory addresses bits (Y86-64: 64)
$E$	number of blocks per set ("ways")
$S = 2^s$	number of sets
$s$	(set) index bits
$B = 2^b$	block size
$b$	(block) offset bits
$t = m - (s + b)$	tag bits
$C = B \times S \times E$	cache size (excluding metadata)

### My desktop:

L1 Data Cache: 32 KB, 8 blocks/set, 64 byte blocks

L2 Cache: 256 KB, 4 blocks/set, 64 byte blocks

L3 Cache: 8 MB, 16 blocks/set, 64 byte blocks

Divide the address 0x34567 into **tag**, **index**, **offset** for each cache.

10

## T-I-O exercise: L1

quantity	value for L1
block size (given)	$B = 64\text{Byte}$
	$B = 2^b$ ( $b$ : block offset bits)

11

## T-I-O exercise: L1

quantity	value for L1
block size (given)	$B = 64\text{Byte}$
	$B = 2^b$ ( $b$ : block offset bits)
block offset bits	$b = 6$

11

## T-I-O exercise: L1

quantity	value for L1
block size (given)	$B = 64\text{Byte}$
	$B = 2^b$ ( $b$ : block offset bits)
block offset bits	$b = 6$
blocks/set (given)	$E = 8$
cache size (given)	$C = 32\text{KB} = E \times B \times S$

11

## T-I-O exercise: L1

quantity	value for L1
block size (given)	$B = 64\text{Byte}$
	$B = 2^b$ ( $b$ : block offset bits)
block offset bits	$b = 6$
blocks/set (given)	$E = 8$
cache size (given)	$C = 32\text{KB} = E \times B \times S$
	$S = \frac{C}{B \times E}$ ( $S$ : number of sets)

11

## T-I-O exercise: L1

quantity	value for L1
block size (given)	$B = 64\text{Byte}$
	$B = 2^b$ ( $b$ : block offset bits)
block offset bits	$b = 6$
blocks/set (given)	$E = 8$
cache size (given)	$C = 32\text{KB} = E \times B \times S$
	$S = \frac{C}{B \times E}$ ( $S$ : number of sets)
number of sets	$S = \frac{32\text{KB}}{64\text{Byte} \times 8} = 64$

11

## T-I-O exercise: L1

quantity	value for L1
block size (given)	$B = 64\text{Byte}$
	$B = 2^b$ ( $b$ : block offset bits)
block offset bits	$b = 6$
blocks/set (given)	$E = 8$
cache size (given)	$C = 32\text{KB} = E \times B \times S$
	$S = \frac{C}{B \times E}$ ( $S$ : number of sets)
number of sets	$S = \frac{32\text{KB}}{64\text{Byte} \times 8} = 64$
	$S = 2^s$ ( $s$ : set index bits)
set index bits	$s = \log_2(64) = 6$

11



## T-I-O results

	L1	L2	L3
sets	64	1024	8192
block offset bits	6	6	6
set index bits	6	10	13
tag bits		(the rest)	

12

## T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	13
tag bits	(the rest)		

0x34567:      3          4          5          6          7  
                 0011 0100 0101 0110 0111

bits 0-5 (all offsets): 100111 = 0x27

13

## T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	13
tag bits	(the rest)		

0x34567:      3          4          5          6          7  
                 0011 0100 0101 0110 0111

bits 0-5 (all offsets): 100111 = 0x27

13

## T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	13
tag bits	(the rest)		

0x34567:      3          4          5          6          7  
                 0011 0100 0101 0110 0111

bits 0-5 (all offsets): 100111 = 0x27

L1:  
bits 6-11 (L1 set): 01 0101 = 0x15  
bits 12- (L1 tag): 0x34

13

## T-I-O: splitting

	L1	L2	L3		
block offset bits	6	6	6		
set index bits	6	10	13		
tag bits	(the rest)				
0x34567:	3	4	5	6	7
	0011	0100	0101	0110	0111

bits 0-5 (all offsets): 100111 = 0x27

L1:

bits 6-11 (L1 set): 01 0101 = 0x15

bits 12- (L1 tag): 0x34

13

## T-I-O: splitting

	L1	L2	L3		
block offset bits	6	6	6		
set index bits	6	10	13		
tag bits	(the rest)				
0x34567:	3	4	5	6	7
	0011	0100	0101	0110	0111

bits 0-5 (all offsets): 100111 = 0x27

L2:

bits 6-15 (set for L2): 01 0001 0101 = 0x115

bits 16-: 0x3

13

## T-I-O: splitting

	L1	L2	L3		
block offset bits	6	6	6		
set index bits	6	10	13		
tag bits	(the rest)				
0x34567:	3	4	5	6	7
	0011	0100	0101	0110	0111

bits 0-5 (all offsets): 100111 = 0x27

L2:

bits 6-15 (set for L2): 01 0001 0101 = 0x115

bits 16-: 0x3

13

## T-I-O: splitting

	L1	L2	L3		
block offset bits	6	6	6		
set index bits	6	10	13		
tag bits	(the rest)				
0x34567:	3	4	5	6	7
	0011	0100	0101	0110	0111

bits 0-5 (all offsets): 100111 = 0x27

L3:

bits 6-18 (set for L3): 0 1101 0001 0101 = 0xD15

bits 18-: 0x0

13

## cache miss types

*compulsory (or cold)* — **first time** accessing something  
doesn't matter how big/flexible the cache is

*conflict* — sets aren't big/flexible enough  
a fully-associative (1-set) cache of the same size would have done better

*capacity* — cache was not big enough

14

## replacement policies

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]
1	1	011000	mem[0x62] mem[0x63]	0		

address (hex)	result
000	hit
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

how to decide where to insert 0x64?

15

## replacement policies

2-way set associative, 2 byte blocks, 2 sets

index	valid	tag	value	valid	tag	value	LRU
0	1	000000	mem[0x00] mem[0x01]	1	011000	mem[0x60] mem[0x61]	1
1	1	011000	mem[0x62] mem[0x63]	0			1

address (hex)	result
00000000 (00)	hit
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

track which block was read least recently updated on **every access**

15

## example replacement policies

least recently used and approximations

take advantage of **temporal locality**

exact:  $\lceil \log_2(E!) \rceil$  bits per set for  $E$ -way cache

good approximations:  $E$  to  $2E$  bits

first-in, first-out

counter per set — where to replace next

(pseudo-)random

no extra information!

16

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	0			0			
1	0			0			

address (hex)	hit?
00000000 (00)	
00000001 (01)	
00001010 (0A)	
00100001 (21)	
00001100 (0C)	
00000011 (02)	
00100011 (23)	

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	0			0			
1	0			0			

address (hex)	hit?
00000000 (00)	
00000001 (01)	
00001010 (0A)	
00100001 (21)	
00001100 (0C)	
00000011 (02)	
00100011 (23)	

how is the address 21 (00100001) split up into tag/index/offset?

$b$  block offset bits;  
 $B = 2^b$  byte block size;  
 $s$  set index bits;  $S = 2^s$  sets;  
 $t = m - (s + b)$  tag bits (leftover)

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	0			0			
1	0			0			

address (hex)	hit?
00000000 (00)	
00000001 (01)	
00001010 (0A)	
00100001 (21)	
00001100 (0C)	
00000011 (02)	
00100011 (23)	

tag index offset

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	0			0			
1	0			0			

address (hex)	hit?
00000000 (00)	
00000001 (01)	
00001010 (0A)	
00100001 (21)	
00001100 (0C)	
00000011 (02)	
00100011 (23)	

tag index offset

exercise: how many accesses are hits?  
 what is the final state of the cache?

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	1	000000	M[0x00] M[0x01] M[0x02] M[0x03]	0			way 1
1	0			0			

address (hex)	hit?
00000000 (00)	miss
00000001 (01)	
00001010 (0A)	
00100001 (21)	
00001100 (0C)	
00000011 (02)	
00100011 (23)	

exercise: how many accesses are hits?  
what is the final state of the cache?

tag index offset

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	1	000000	M[0x00] M[0x01] M[0x02] M[0x03]	1	000001	M[0x08] M[0x09] M[0x0A] M[0x0B]	way 0
1	0			0			

address (hex)	hit?
00000000 (00)	miss
00000001 (01)	hit
00001010 (0A)	miss
00100001 (21)	
00001100 (0C)	
00000011 (02)	
00100011 (23)	

tag index offset

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	1	001000	M[0x20] M[0x21] M[0x22] M[0x23]	1	000001	M[0x08] M[0x09] M[0x0A] M[0x0B]	way 1
1	0			0			

address (hex)	hit?
00000000 (00)	miss
00000001 (01)	hit
00001010 (0A)	miss
00100001 (21)	miss
00001100 (0C)	miss
00000011 (02)	
00100011 (23)	

tag index offset

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	1	001000	M[0x20] M[0x21] M[0x22] M[0x23]	1	000000	M[0x00] M[0x01] M[0x02] M[0x03]	way 0
1	1	000000	M[0x0C] M[0x0D] M[0x0E] M[0x0F]	0			way 1

address (hex)	hit?
00000000 (00)	miss
00000001 (01)	hit
00001010 (0A)	miss
00100001 (21)	miss
00001100 (0C)	miss
00000011 (02)	miss
00100011 (23)	

tag index offset

17

## exercise

4 byte blocks, 2 sets

index	V	tag	value	V	tag	value	LRU
0	1	00100	M[0x20] M[0x21] M[0x22] M[0x23]	1	00000	M[0x00] M[0x01] M[0x02] M[0x03]	way 1
1	1	00000	M[0x0C] M[0x0D] M[0x0E] M[0x0F]	0			way 1

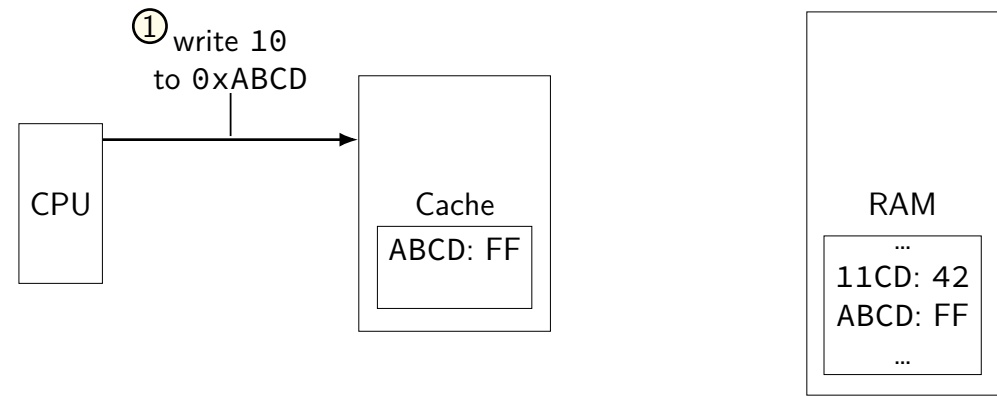
address (hex)	hit?
00000000 (00)	miss
00000001 (01)	hit
00001010 (0A)	miss
00100001 (21)	miss
00001100 (0C)	miss
00000011 (02)	miss
00100011 (23)	hit

tag index offset

17

## write-through v. write-back

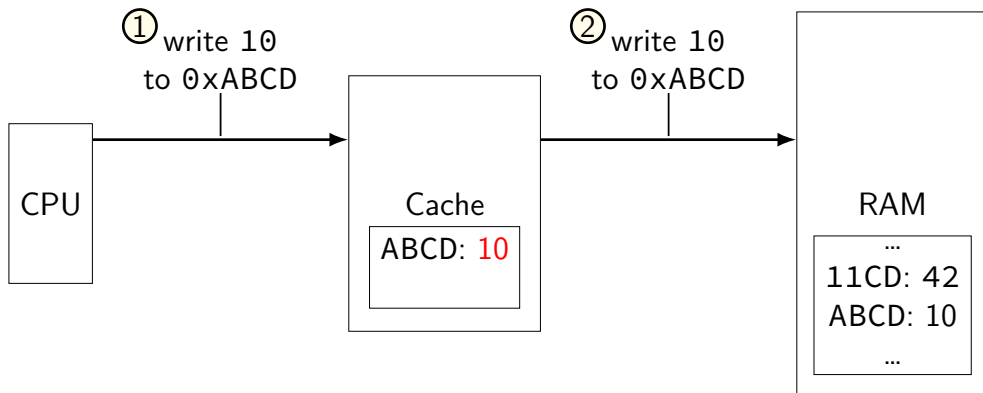
option 1: write-through



18

## write-through v. write-back

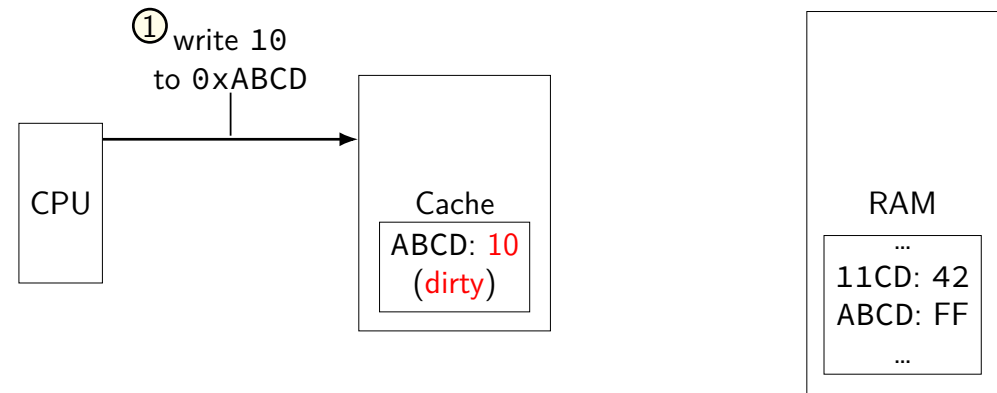
option 1: write-through



18

## write-through v. write-back

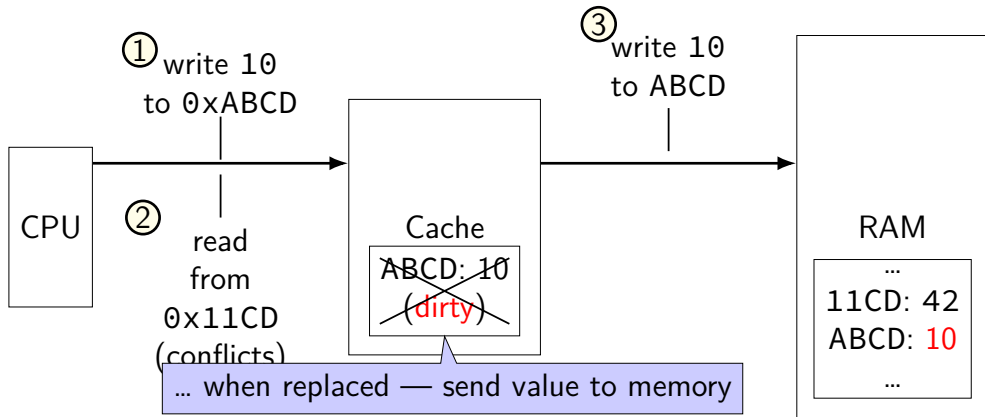
option 2: write-back



18

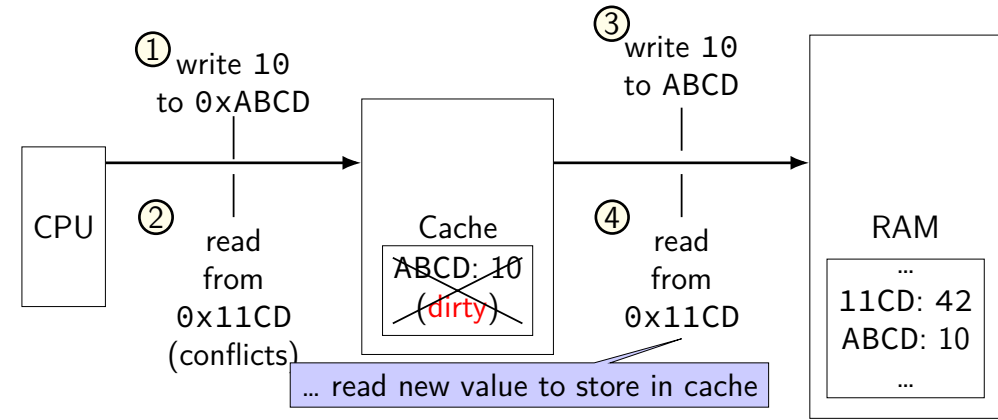
## write-through v. write-back

### option 2: write-back



18

## write-through v. write-back



18

## writeback policy

changed value!

2-way set associative, 4 byte blocks, 2 sets

index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	mem[0x60]* mem[0x61]*	1	1
1	1	011000	mem[0x62] mem[0x63]	0	0				0

1 = dirty (different than memory)  
needs to be written if evicted

19

## allocate on write?

processor writes **less than whole** cache block

block not yet in cache

two options:

**write-allocate**

fetch rest of cache block, replace written part

**write-no-allocate**

send write through to memory

guess: not read soon?

20

## write-allocate

2-way set associative, LRU, writeback

index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	mem[0x60]* mem[0x61]*	1	1
1	1	011000	mem[0x62] mem[0x63]	0	0				0

writing 0xFF into address 0x04?  
index 0, tag 000001

21

## write-allocate

2-way set associative, LRU, writeback

index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	mem[0x60]* mem[0x61]*	1	1
1	1	011000	mem[0x62] mem[0x63]	0	0				0

writing 0xFF into address 0x04?  
index 0, tag 000001  
step 1: find **least recently used** block

21

## write-allocate

2-way set associative, LRU, writeback

index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	mem[0x60]* mem[0x61]*	1	1
1	1	011000	mem[0x62] mem[0x63]	0	0				0

writing 0xFF into address 0x04?  
index 0, tag 000001  
step 1: find **least recently used** block  
step 2: possibly writeback old block

21

## write-allocate

2-way set associative, LRU, writeback

index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	0xFF mem[0x05]	1	0
1	1	011000	mem[0x62] mem[0x63]	0	0				0

writing 0xFF into address 0x04?  
index 0, tag 000001  
step 1: find **least recently used** block  
step 2: possibly writeback old block  
step 3a: read in new block – to get mem[0x05]  
step 3b: update LRU information

21



## write-no-allocate

2-way set associative, LRU, writeback

index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	mem[0x60] mem[0x61]	1	1
1	1	011000	mem[0x62] mem[0x63]	0	0				0

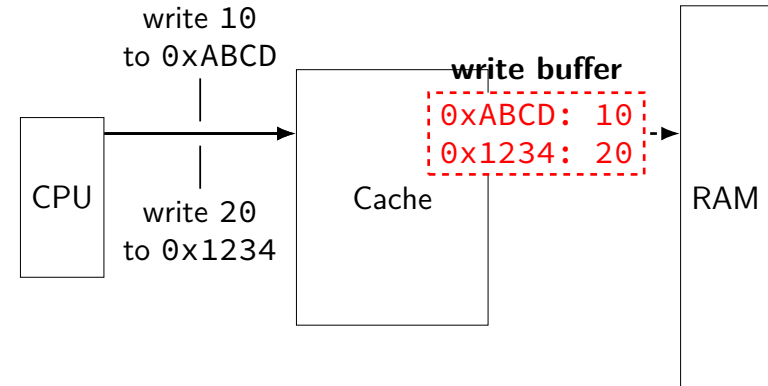
writing 0xFF into address 0x04?

step 1: is it in cache yet?

step 2: no, just send it to memory

22

## fast writes



write appears to complete immediately when placed in buffer  
memory can be much slower

23