CS3330: Caching

2004 CPU

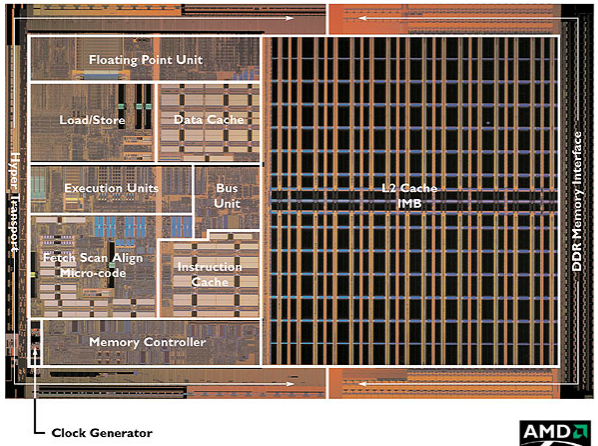
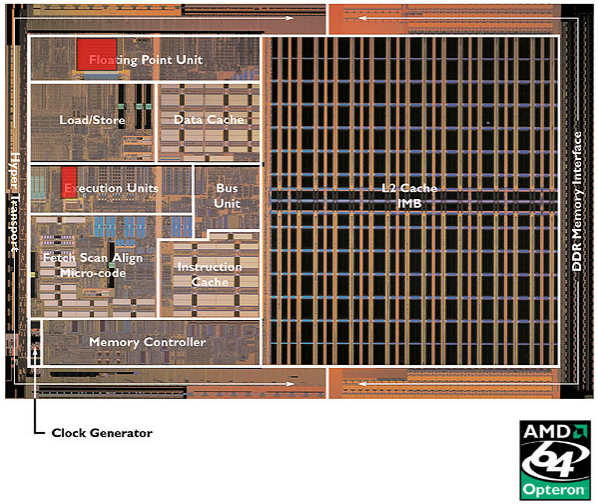


Image: approx 2004 AMD press image of Opteron die

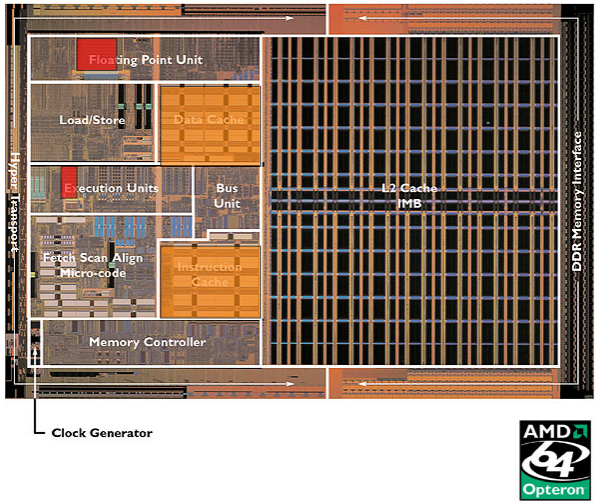
2004 CPU



Registers

Image: approx 2004 AMD press image of Opteron die

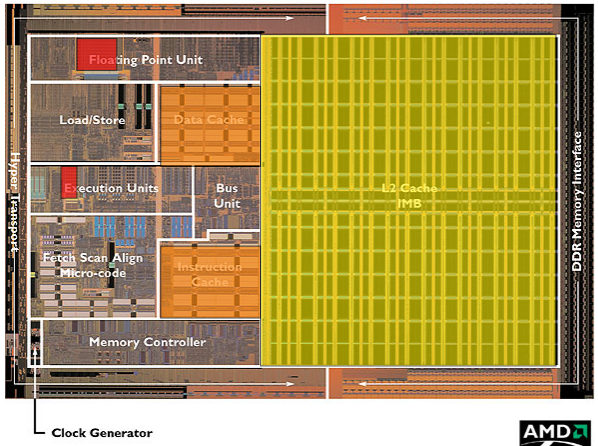
2004 CPU



Registers
L1 cache

Image: approx 2004 AMD press image of Opteron die

2004 CPU



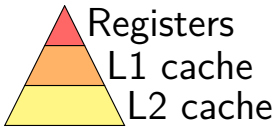
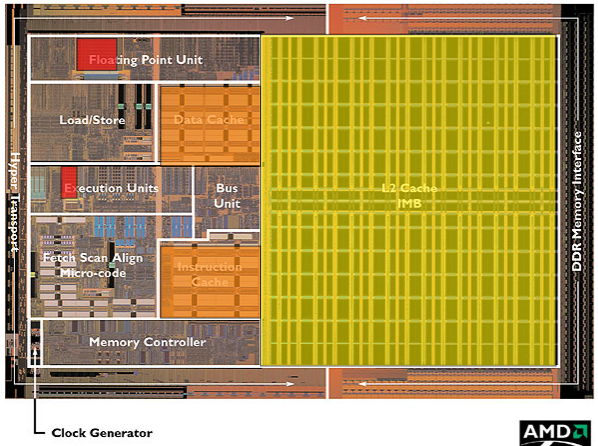


Image: approx 2004 AMD press image of Opteron die

2004 CPU



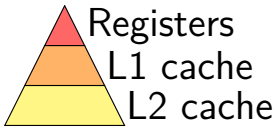
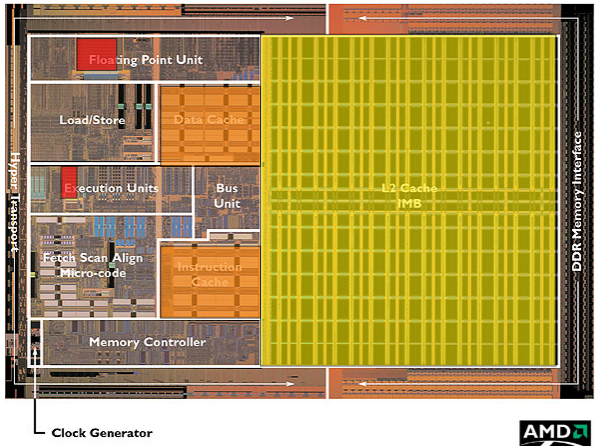


Image: approx 2004 AMD press image of Opteron die

2004 CPU



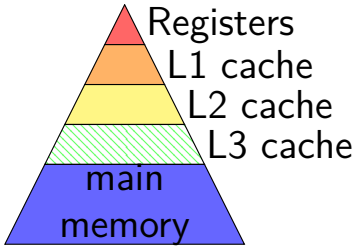
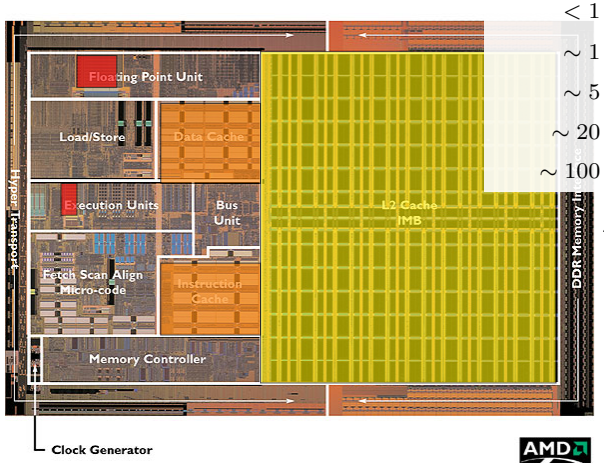


Image: approx 2004 AMD press image of Opteron die

2004 CPU



< 1 ns
~ 1 ns
~ 5 ns
~ 20 ns
~ 100 ns

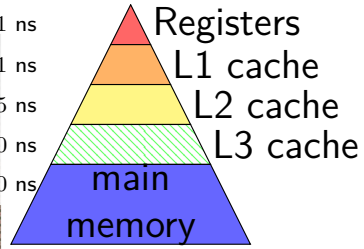
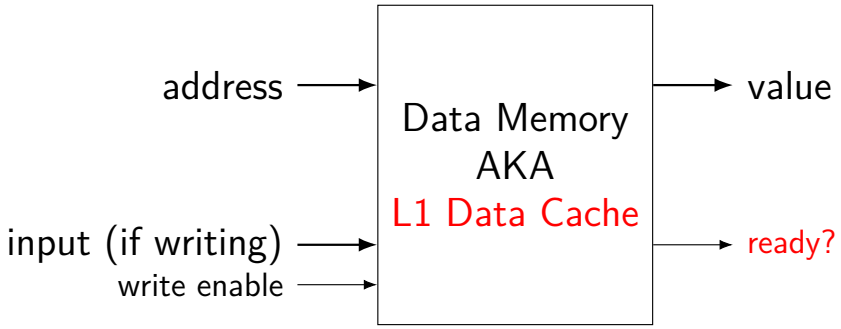
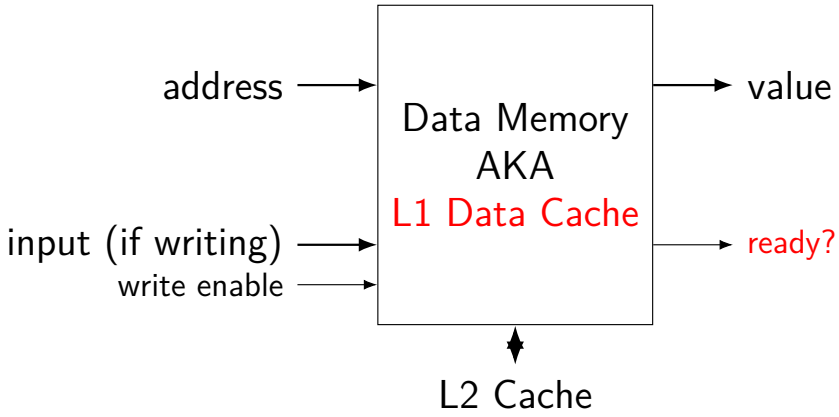


Image: approx 2004 AMD press image of Opteron die

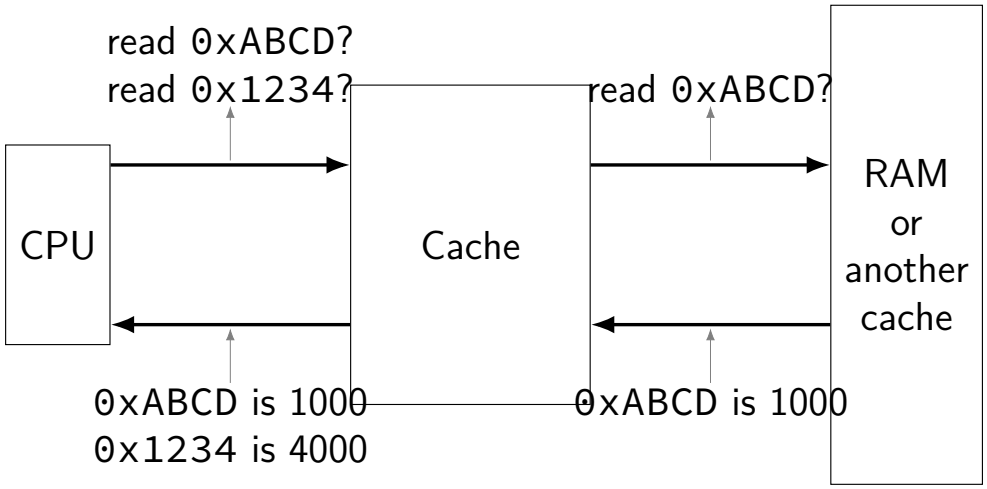
cache: real memory



cache: real memory



the place of cache



memory hierarchy goals

performance of the fastest memory

- hide 100x latency difference? 99+% hit rate

capacity of the largest memory

building a (direct-mapped) cache

Cache

value
00 00
00 00
00 00
00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

cache block: 2 bytes

building a (direct-mapped) cache

read byte at 01011?

Cache

value
00 00
00 00
00 00
00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

cache block: 2 bytes

building a (direct-mapped) cache

read byte at 01011?

exactly one place for each address
spread out what can go in a block

Cache

index	value
00	00 00
01	00 00
10	00 00
11	00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

cache block: 2 bytes

direct-mapped

building a (direct-mapped) cache

read byte at 01011?

exactly one place for each address
spread out what can go in a block

Cache

index	value
00	00 00
01	00 00
10	00 00
11	00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

cache block: 2 bytes

direct-mapped

building a (direct-mapped) cache

read byte at 01011?

exactly one place for each address
spread out what can go in a block

Cache

index	value
00	00 00
01	00 00
10	00 00
11	00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

cache block: 2 bytes
direct-mapped

building a (direct-mapped) cache

read byte at 01011?

Cache

index	valid	value
00	0	00 00
01	0	00 00
10	0	00 00
11	0	00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

is this even a value?

need extra bit to know

cache block: 2 bytes
direct-mapped

building a (direct-mapped) cache

read byte at 01011?
invalid, fetch

Cache

index	valid	value
00	0	00 00
01	1	AA BB
10	0	00 00
11	0	00 00

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

cache block: 2 bytes
direct-mapped

building a (direct-mapped) cache

read byte at 01011?
invalid, fetch

Cache

index	valid	tag	value
00	0	00	00 00
01	1	01	AA BB
10	0	00	00 00
11	0		

Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

value from 01010 or 00010?

need tag to know

cache block: 2 bytes
direct-mapped

building a (direct-mapped) cache

read byte at 01011?
invalid, fetch

Cache

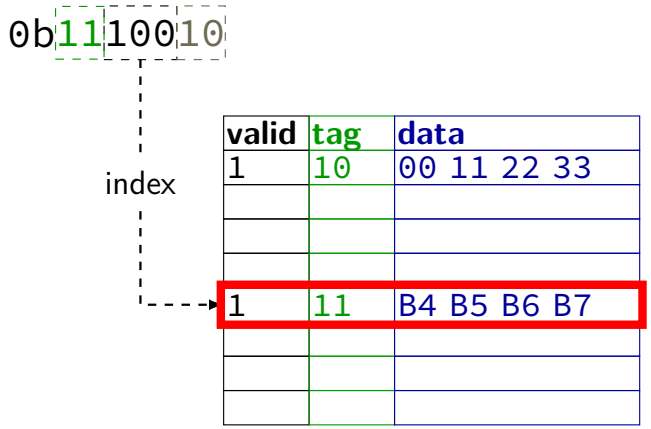
index	valid	tag	value
00	0	00	00 00
01	1	01	AA BB
10	0	00	00 00
11	0	00	00 00

cache block: 2 bytes
direct-mapped

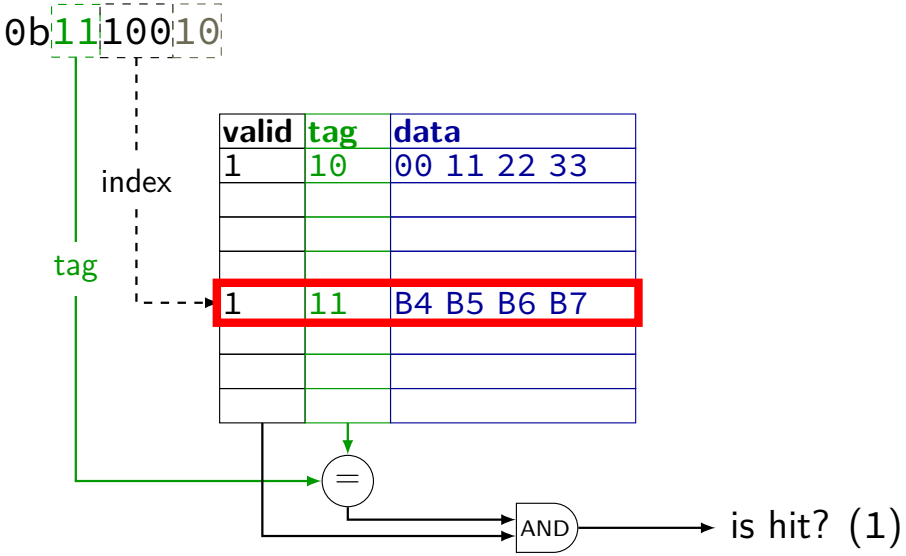
Memory

addresses	bytes
00000–00001	00 11
00010–00011	22 33
00100–00101	55 55
00110–00111	66 77
01000–01001	88 99
01010–01011	AA BB
01100–01101	CC DD
01110–01111	EE FF
10000–10001	F0 F1
…	…

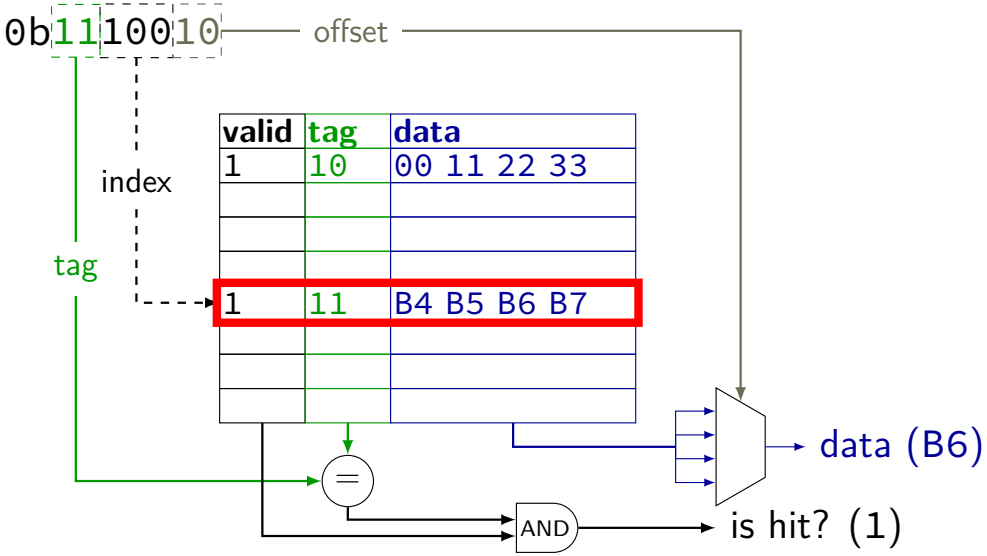
cache operation (read)



cache operation (read)



cache operation (read)



Tag-Index-Offset (TIO)

address 001111 = FF

cache	tag	index	offset
2 byte blocks, 4 sets	???	???	???
2 byte blocks, 8 sets	???	???	???
4 byte blocks, 2 sets	???	???	???

2 byte blocks, 4 sets

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0	--	-- --
11	1	001	EE FF

4 byte blocks, 2 sets

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
001	1	01	F1 F2
010	0	--	-- --
011	0	--	-- --
100	0	--	-- --
101	1	00	AA BB
110	0	--	-- --
111	1	00	EE FF

Tag-Index-Offset (TIO)

address 001111 = FF

cache	tag	index	offset
2 byte blocks, 4 sets	???	???	1
2 byte blocks, 8 sets	???	???	1
4 byte blocks, 2 sets	???	???	???

2 byte blocks, 4 sets

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0	--	-- --
11	1	001	EE FF

4 byte blocks, 2 sets

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
011	0	--	-- --
100	0	--	-- --
101	1	00	AA BB
110	0	--	-- --
111	1	00	EE FF

$2 = 2^1$ bytes in block
1 bit to say which byte

Tag-Index-Offset (TIO)

address 001111 = FF

cache	tag	index	offset
2 byte blocks, 4 sets	???	???	1
2 byte blocks, 8 sets	???	???	1
4 byte blocks, 2 sets	???	???	11

2 byte blocks, 4 sets

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0		
11	1		

4 byte

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
001	1	01	F1 F2
0	--	-- --	
0	--	-- --	
0	--	-- --	
1	00	AA BB	
110	0	--	-- --
111	1	00	EE FF

$4 = 2^2$ bytes in block
2 bits to say which byte

Tag-Index-Offset (TIO)

address 001111 = FF

cache	tag	index	offset
2 byte blocks, 4 sets	???	11	1
2 byte blocks, 8 sets	???		1
4 byte blocks, 2 sets	???	1	11

2 byte blocks, 4 sets

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0	--	-- --
11	1	001	EE FF

4 byte blocks, 2 sets

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
			F1 F2
			-- --
			-- --
100	0	--	-- --
101	1	00	AA BB
110	0	--	-- --
111	1	00	EE FF

$2^2 = 4$ sets
2 bits to index set

Tag-Index-Offset (TIO)

address `001111` = `FF`

cache	tag	index	offset
2 byte blocks, 4 sets	???	11	1
2 byte blocks, 8 sets	???	111	1
4 byte blocks, 2 sets	???	1	11

2 byte blocks, 4 sets

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0	--	-- --
11	1	001	EE FF

4 byte blocks, 2 sets

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
001	1	01	F1 F2
010	0	--	-- --
011	0	--	-- --
100	0	--	-- --
101	1	00	AA BB
110	0	--	-- --
111	1	00	EE FF

$2^3 = 8$ sets
3 bits to index set

Tag-Index-Offset (TIO)

address `001111` = `FF`

cache	tag	index	offset
2 byte blocks, 4 sets	???	11	1
2 byte blocks, 8 sets	???	111	1
4 byte blocks, 2 sets	???	1	11

2 byte blocks, 4 sets

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0	--	-- --
11	1	001	EE FF

4 byte blocks, 2 sets

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
001	1	01	F1 F2
010	0	--	-- --
011	[illegible]	[illegible]	[illegible]
100	[illegible]	[illegible]	[illegible]
101	[illegible]	[illegible]	BB
110	0	--	-- --
111	1	00	EE FF

$2^1 = 2$ sets
1 bit to index set

Tag-Index-Offset (TIO)

address `001111` = `FF`

cache	tag	index	offset
2 byte blocks, 4 sets	001	11	1
2 byte blocks, 8 sets	00	111	1
4 byte blocks, 2 sets	001	1	11

tag — whatever is left over

index	valid	tag	value
00	1	000	00 11
01	1	001	AA BB
10	0	--	-- --
11	1	001	EE FF

4 byte blocks, 2 sets

index	valid	tag	value
0	1	00	00 11 22 33
1	1	01	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
001	1	01	F1 F2
010	0	--	-- --
011	0	--	-- --
100	0	--	-- --
101	1	00	AA BB
110	0	--	-- --
111	1	00	EE FF

Tag-Index-Offset formulas

m	memory addreses bits (Y86-64: 64)
$S = 2^s$	number of sets
s	(set) index bits
$B = 2^b$	block size
b	(block) offset bits
$t = m - (s + b)$	tag bits
$C = B \times S$	cache size (if direct-mapped)

example access pattern (1)

2 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	0		
01	0		
10	0		
11	0		

example access pattern (1)

2 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	0		
01	0		
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	0		
01	0		
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	0		
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	0		
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	1	01100	mem[0x62] mem[0x63]
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	01100	mem[0x60] mem[0x61]
01	1	01100	mem[0x62] mem[0x63]
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

index	valid	tag	value
00	1	01100	mem[0x60] mem[0x61]
01	1	01100	mem[0x62] mem[0x63]
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	1	01100	mem[0x62] mem[0x63]
10	0		
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	1	01100	mem[0x62] mem[0x63]
10	1	01100	mem[0x64] mem[0x65]
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	1	01100	mem[0x62] mem[0x63]
10	1	01100	mem[0x64] mem[0x65]
11	0		

$m = 8$ bit 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

example access pattern (1)

2 byte blocks, 4 sets

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

tag index offset

index	valid	tag	value
00	1	00000	mem[0x00] mem[0x01]
01	1	01100	mem[0x62] mem[0x63]
10	1	01100	mem[0x64] mem[0x65]
11	0		

miss caused by conflict

$m = 8$ bit 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

adding associativity

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

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

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		

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
$S = 2 = 2^s$ sets
$s = 1$ (set) index bits

$B = 2 = 2^b$ byte block size
$b = 1$ (block) offset bits
$t = m - (s + b) = 6$ tag bits

adding associativity

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

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

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

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)	

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)	

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)	

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)	

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)	

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)	

adding associativity

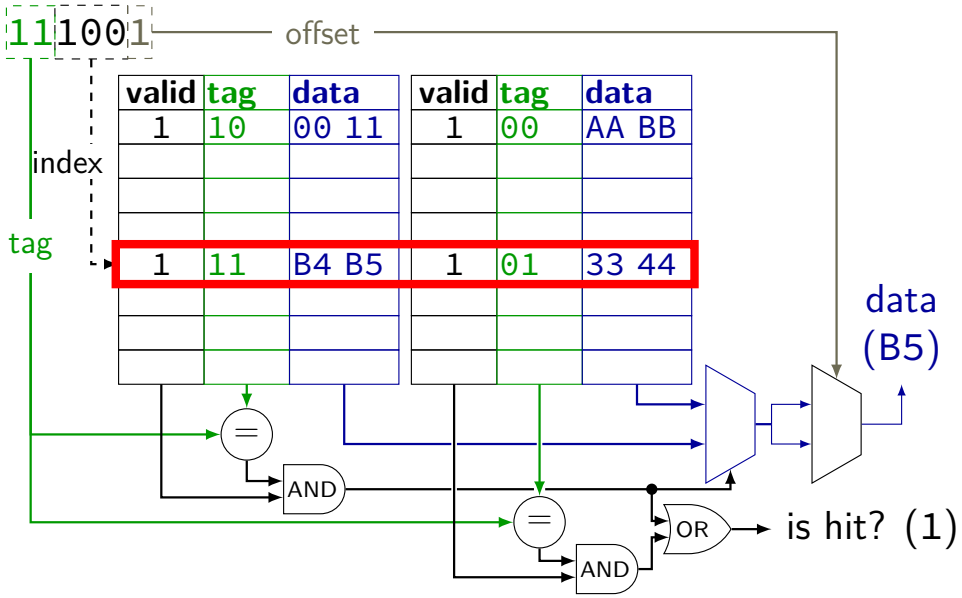
2-way set associative, 2 byte blocks, 2 sets

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		

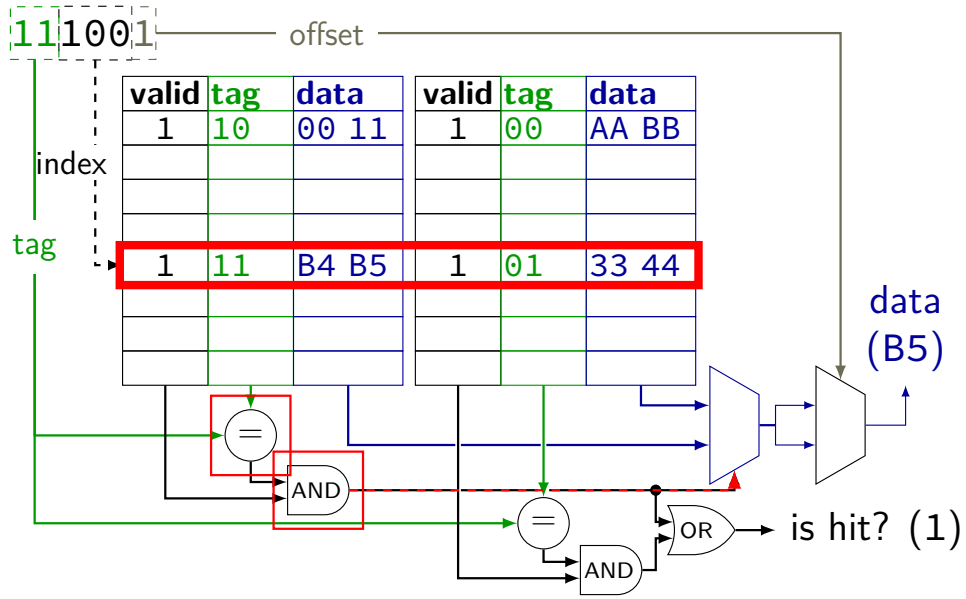
address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

needs to replace block in set 0!

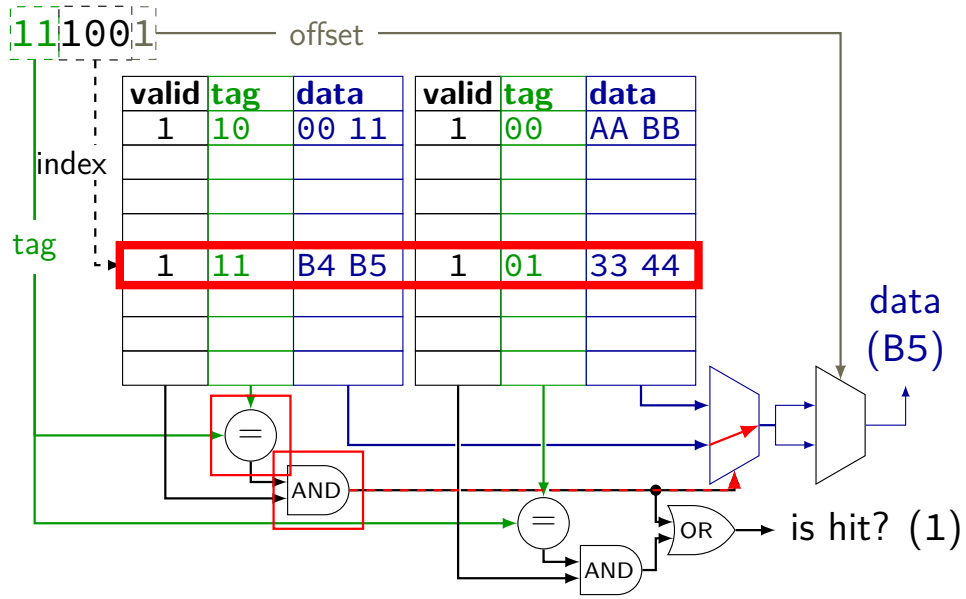
cache operation (associative)



cache operation (associative)



cache operation (associative)



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	
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?

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

track which block was read least recently
updated on every access

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!

associativity terminology

direct-mapped — one block per set

E-way set associative — E blocks per set

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

Tag-Index-Offset formulas (2)

m	memory addreses 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)

Tag-Index-Offset exercise

m	memory addreses 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 laptop:

- L1 Cache: 32KB, 8 blocks/set, 64 byte blocks
- L2 Cache: 256KB, 4 blocks/set, 64 byte blocks
- L3 Cache: 4096KB, 16 blocks/set, 64 byte blocks

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

T-I-O exercise: L1

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

T-I-O exercise: L1

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

T-I-O exercise: L1

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

T-I-O exercise: L1

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

T-I-O exercise: L1

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

T-I-O exercise: L1

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

T-I-O results

	L1	L2	L3
sets	64	1024	4096
block offset bits	6	6	6
set index bits	6	10	12
tag bits		(the rest)	

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits		(the rest)	

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

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

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits	(the rest)		

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

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

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits	(the rest)		

	3	4	5	6	7
0x34567:	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

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits	(the rest)		

	3	4	5	6	7
0x34567:	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

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits	(the rest)		

	3	4	5	6	7
0x34567:	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

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits	(the rest)		

	3	4	5	6	7
`0x34567`:	`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`

T-I-O: splitting

	L1	L2	L3
block offset bits	6	6	6
set index bits	6	10	12
tag bits	(the rest)		

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

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

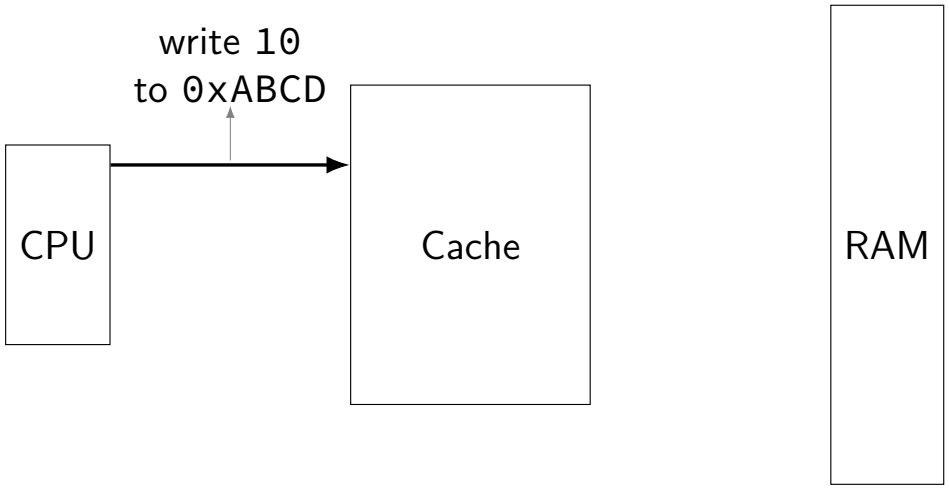
L3:

bits 6-17 (set for L3): `1101 0001 0101` = `0xD15`
bits 18-: `0x0`

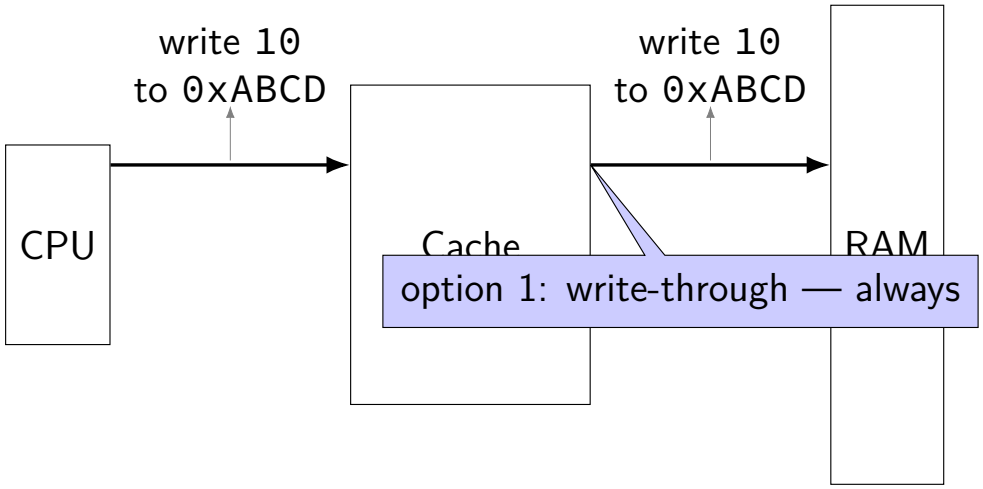
T-I-O: answers

	L1	L2	L3
offset	`0x27`	`0x27`	`0x27`
index	`0x15`	`0x115`	`0xD15`
tag	`0x34`	`0x3`	`0x0`

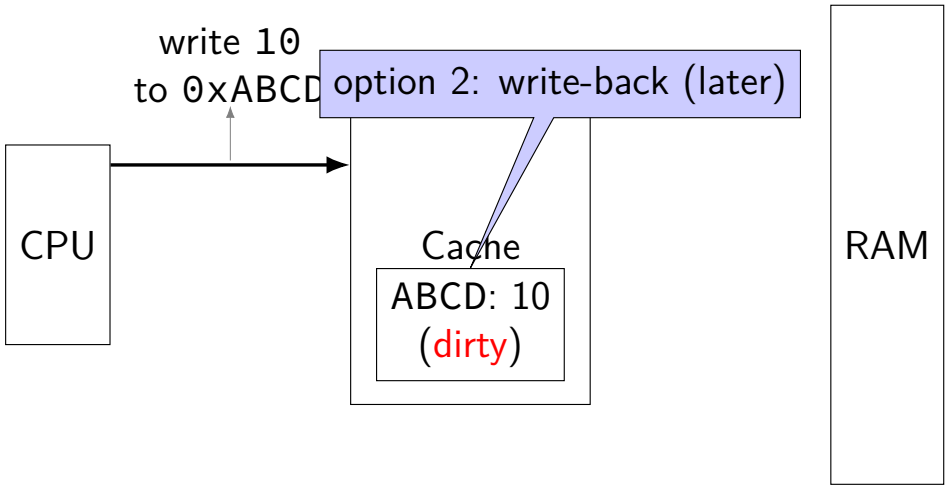
writing to caches



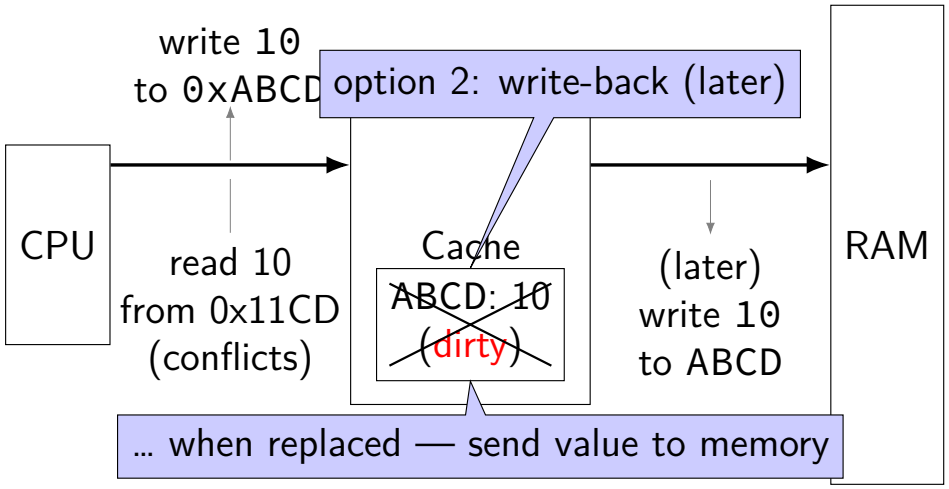
writing to caches



writing to caches



writing to caches



writeback policy

changed value!

2-way set associative, 2 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

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?

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

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

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

write-allocate

2-way set associative, LRU, writeback

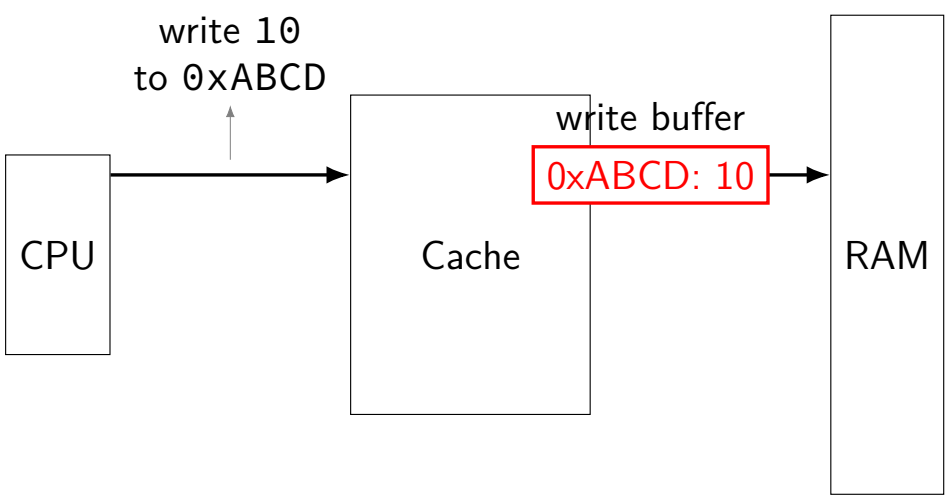
index	valid	tag	value	dirty	valid	tag	value	dirty	LRU
0	1	000000	mem[0x00] mem[0x01]	0	1	011000	0xFF mem[0x05]	1	0
1	1	011000	mem[0x62] mem[0x63]	0	0				0

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

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

fast writes