last time

branch prediction

- instead of stalling, do something
- if that something is wrong, undo it

squashing (undoing) instructions

- if instruction hasn't done too much, just clear pipeline registers
- more complicated if it's changed other things

implementing stalling and squashing

- keep insturction in Y: `stall_Y` write disable for xY registers
- send NOP to Y: `bubble_Y` reset for xY registers

using icodes, etc. in pipeline to set `stall/bubble_Y`

recall: data/instruction memory

model in CPU: one cycle per access

but earlier — had to talk to memory on different chip

can't do that in one cycle

solution: keep copies of part of memory ("cache")

- copy can be accessed quickly
- hope: almost always use copy?

2004 CPU

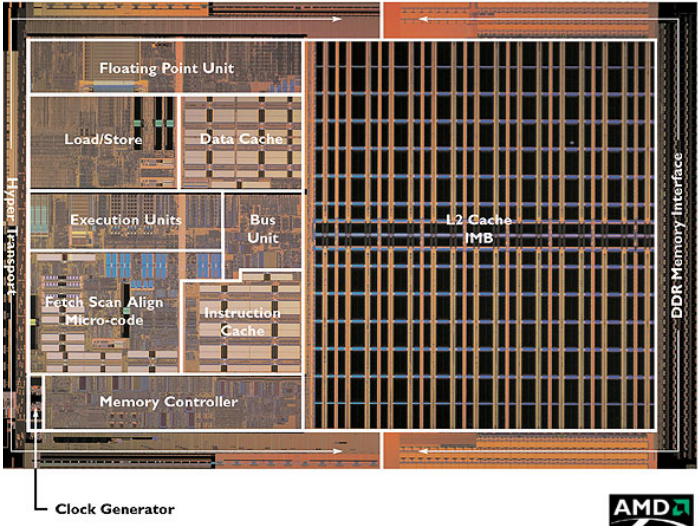
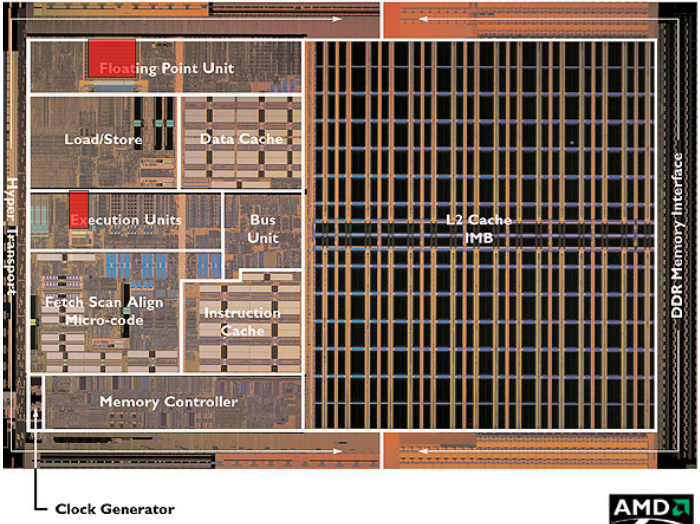


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



Registers

AMD 64 Opteron

Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU

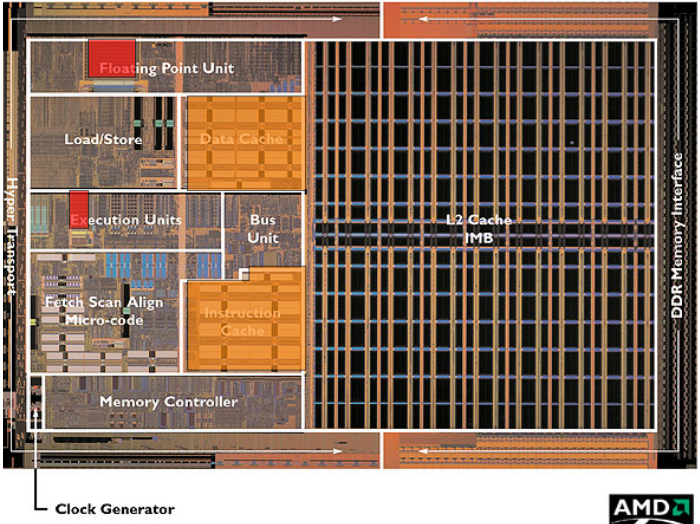
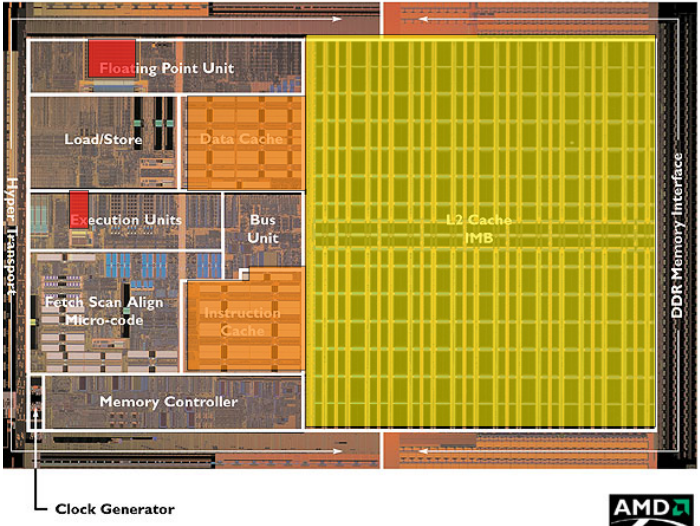


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



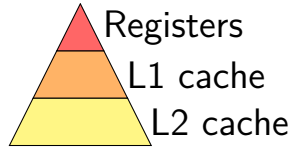
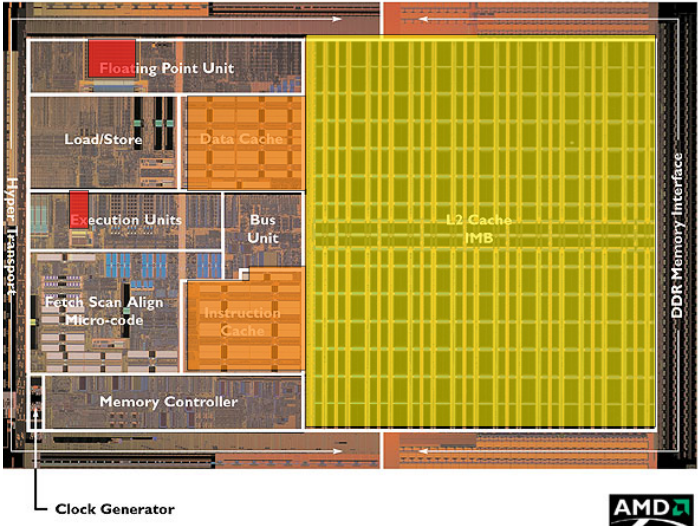


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



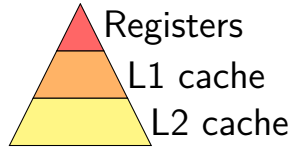
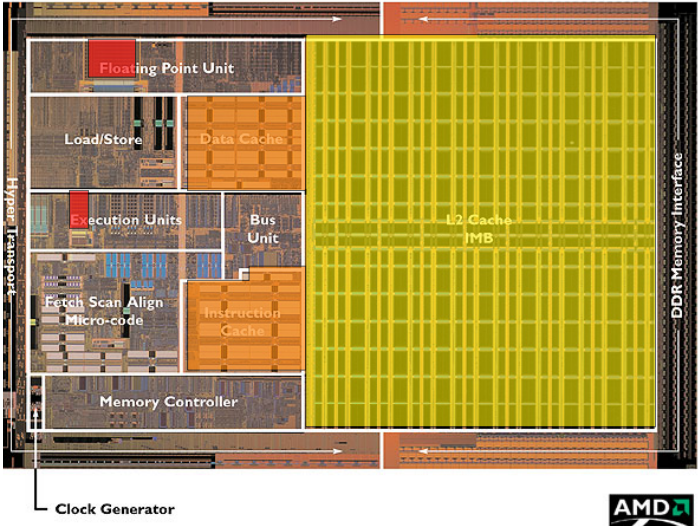


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



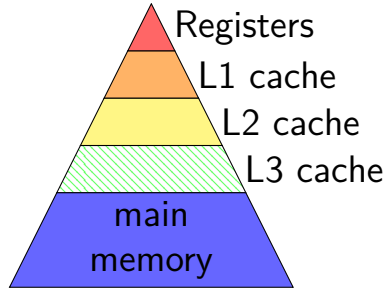
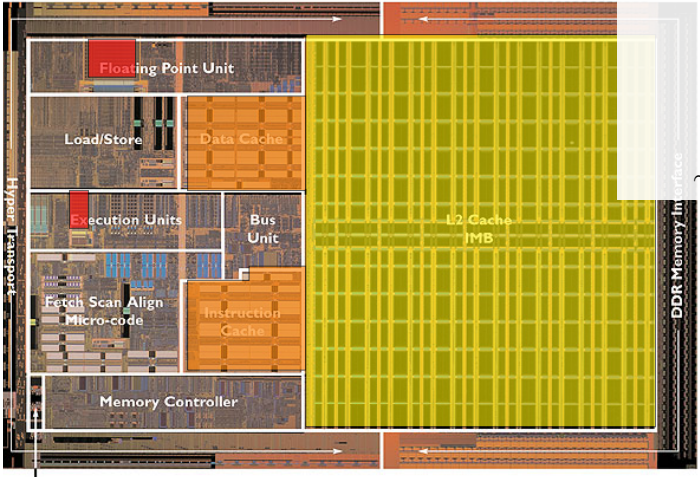


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

2004 CPU



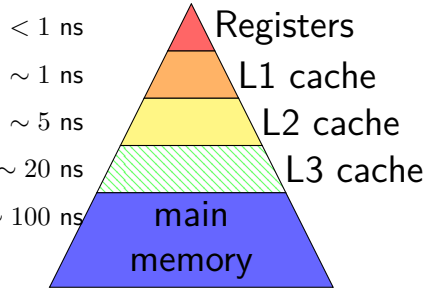
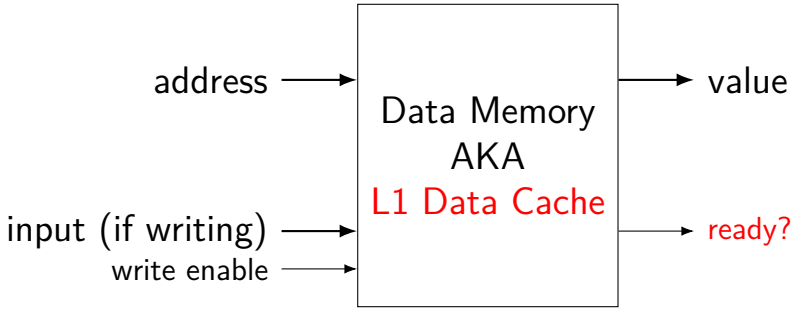
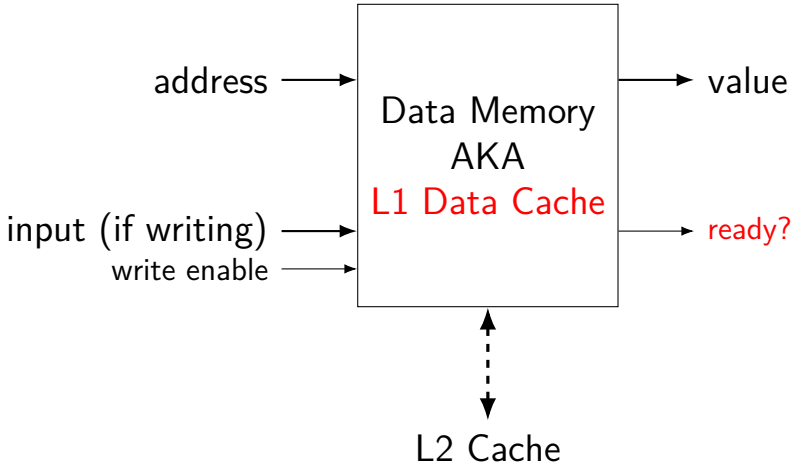


Image: approx 2004 AMD press image of Opteron die;
approx register location via chip-architect.org (Hans de Vries)

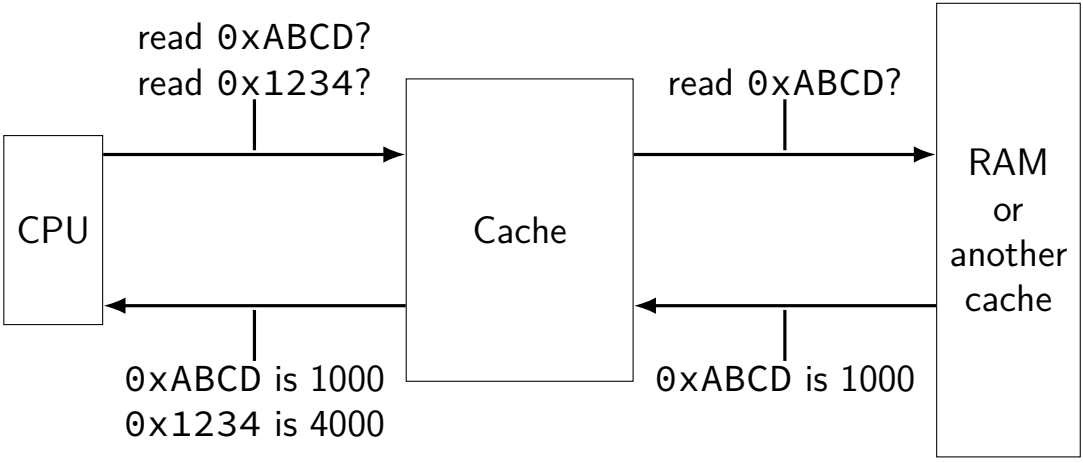
cache: real memory



cache: real memory



the place of cache



memory hierarchy goals

performance of the fastest (smallest) memory

- hide 100x latency difference? 99+% hit (= value found in cache) rate

capacity of the largest (slowest) memory

memory hierarchy assumptions

temporal locality

"if a value is accessed now, it will be accessed again soon"

caches should keep recently accessed values

spatial locality

"if a value is accessed now, adjacent values will be accessed soon"

caches should store adjacent values at the same time

natural properties of programs — think about loops

locality examples

```
double computeMean(int length, double *values) {
    double total = 0.0;
    for (int i = 0; i < length; ++i) {
        total += values[i];
    }
    return total / length;
}
```

temporal locality: machine code of the loop

spatial locality: machine code of most consecutive instructions

temporal locality: `total`, `i`, `length` accessed repeatedly

spatial locality: `values[i+1]` accessed after `values[i]`

building a (direct-mapped) cache

Cache

value
00 00
00 00
00 00
00 00

cache block: 2 bytes

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
…	…

building a (direct-mapped) cache

read byte at 01011?

Cache

value
00 00
00 00
00 00
00 00

cache block: 2 bytes

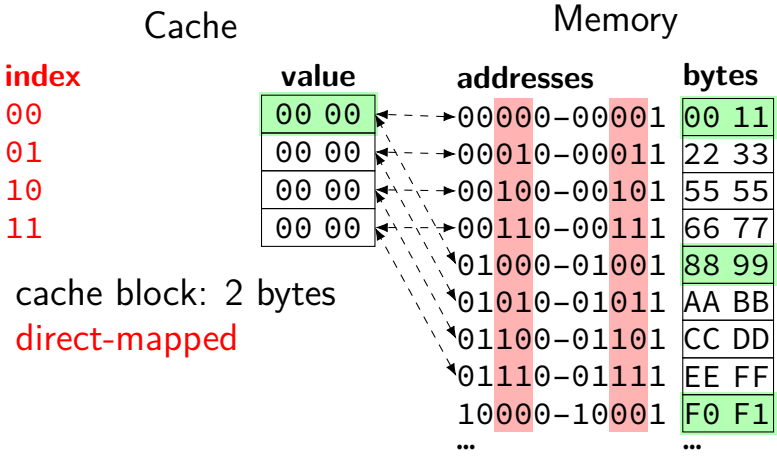
Memory

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
…	…

building a (direct-mapped) cache

read byte at 01011?

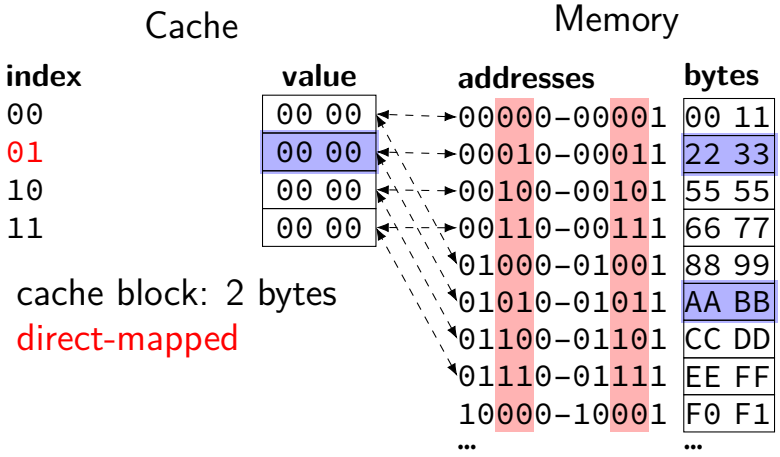
exactly one place for each address
spread out what can go in a block



building a (direct-mapped) cache

read byte at 01011?

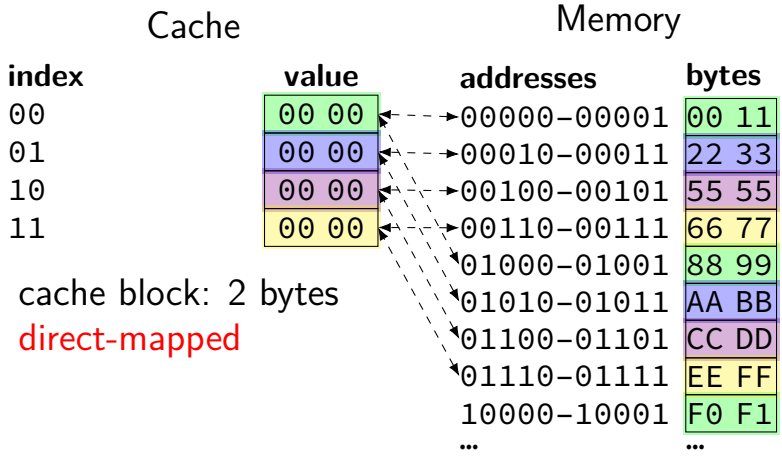
exactly one place for each address
spread out what can go in a block



building a (direct-mapped) cache

read byte at 01011?

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

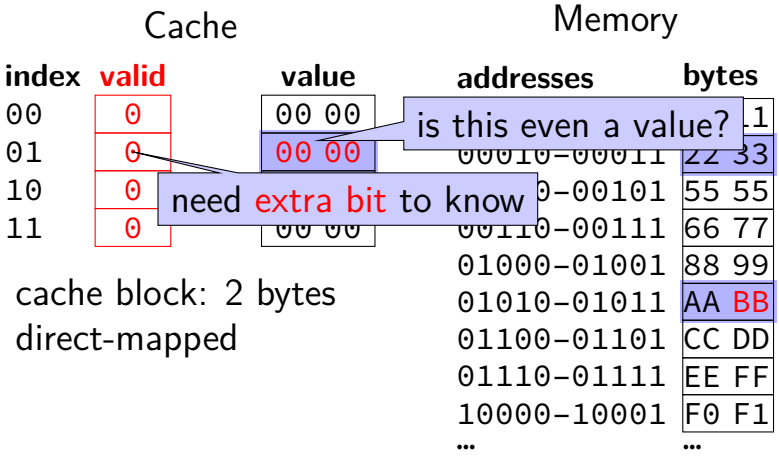


cache block: 2 bytes

direct-mapped

building a (direct-mapped) cache

read byte at 01011?



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

cache block: 2 bytes

direct-mapped

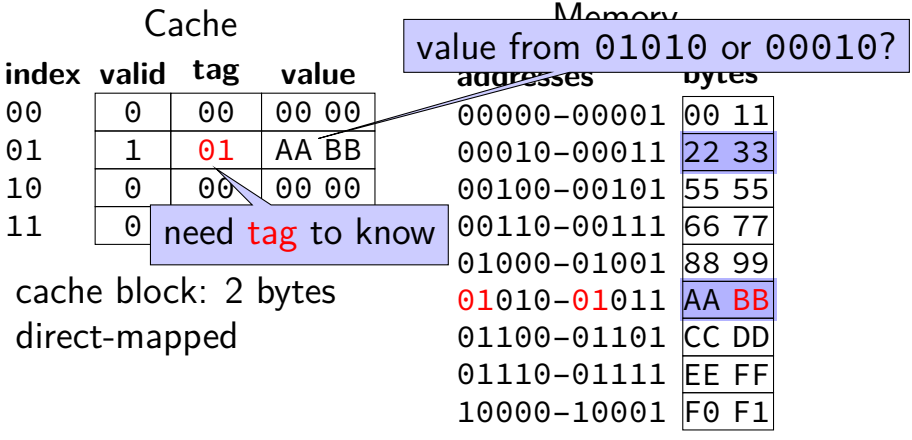
Memory

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
…	…

building a (direct-mapped) cache

read byte at 01011?

invalid, fetch



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

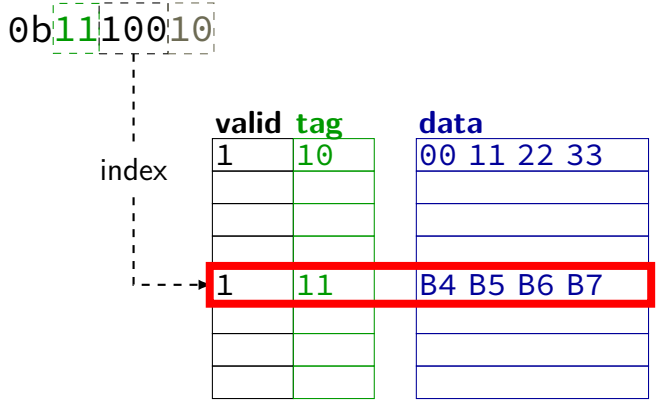
cache block: 2 bytes

direct-mapped

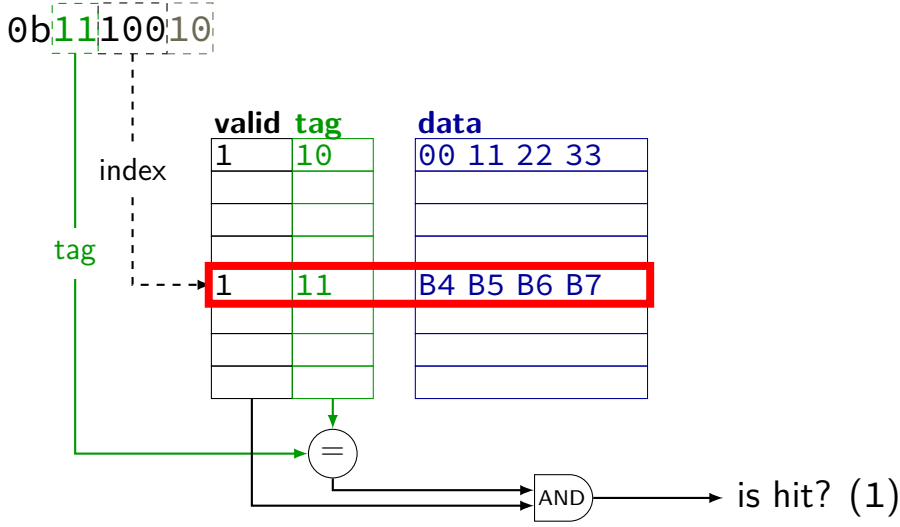
Memory

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
…	…

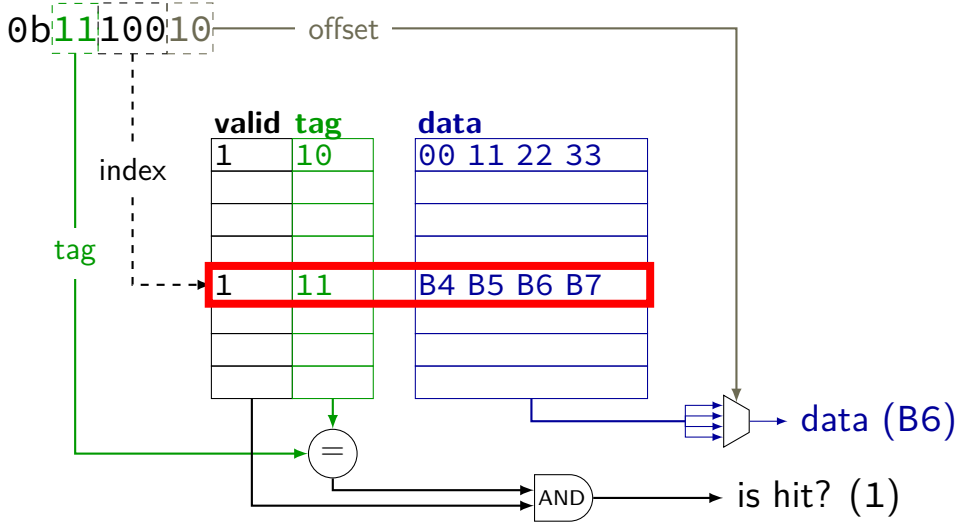
cache operation (read)



cache operation (read)



cache operation (read)



terminology

row = set

preview: change how much is in a row

Tag-Index-Offset (TIO)

address `001111` (stores value `0xFF`)

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	000	00 11 22 33
1	1	001	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` (stores value `0xFF`)

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	000	00 11 22 33
1	1	001	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	000	00 11
001	1	001	F1 F2
010	0	--	-- --
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` (stores value `0xFF`)

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 blocks, 2 sets

index	valid	tag	value
0	1	000	00 11 22 33
1	1	001	CC DD EE FF

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

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` (stores value `0xFF`)

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	000	00 11 22 33
1	1	001	CC DD EE FF

2 byte blocks, 8 sets

index	valid	tag	value
000	1	00	00 11
001	1	00	F1 F2
010	0	--	-- --
011	0	--	-- --
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` (stores value `0xFF`)

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		

4

index	valid	tag	value
0	1	000	00 11 22 33
1	1	001	CC DD EE FF

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

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` (stores value `0xFF`)

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	000	00 11 22 33
1	1	001	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	--	-- --
01			--
10			--
10			BB
110	0	--	-- --
111	1	00	EE FF

$2^1 = 2$ sets

1 bit to index set

Tag-Index-Offset (TIO)

address `001111` (stores value `0xFF`)

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

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	000	00 11 22 33
1	1	001	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 (direct-mapped only)

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)

TIO: exercise

64-byte blocks, 128 set cache

stores $64 \times 128 = 8192$ bytes (of data)

if addresses 32-bits, then how many tag/index/offset bits?

which bytes are stored in the same block as byte from `0x1037`?

A. byte from `0x1011`
B. byte from `0x1021`
C. byte from `0x1035`
D. byte from `0x1041`

example access pattern (1)

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

2 byte blocks, 4 sets

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

example access pattern (1)

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

2 byte blocks, 4 sets

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)

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

tag index offset

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

tag index offset

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

tag index offset

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

tag index offset

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

tag index offset

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

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

2 byte blocks, 4 sets

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

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

example access pattern (1)

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

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

2 byte blocks, 4 sets

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		

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

example access pattern (1)

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

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

2 byte blocks, 4 sets

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		

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

example access pattern (1)

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

2 byte blocks, 4 sets

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

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			

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			

how is the 8-bit 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 (lefto

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			

$m = 8$ 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

exercise

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

tag index offset

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

4 byte blocks, 4 sets

index	valid	tag	value
00			
01			
10			
11			

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

exercise

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

tag index offset

4 byte blocks, 4 sets

index	valid	tag	value
00			
01			
10			
11			

exercise: which accesses are hits?

cache accesses and C code (1)

```
int scaleFactor;

int scaleByFactor(int value) {
    return value * scaleFactor;
}
```

```
scaleByFactor:
    movl scaleFactor, %eax
    imull %edi, %eax
    ret
```

---

exericse: what data cache accesses does this function do?

cache accesses and C code (1)

```
int scaleFactor;

int scaleByFactor(int value) {
    return value * scaleFactor;
}
```

```
scaleByFactor:
    movl scaleFactor, %eax
    imull %edi, %eax
    ret
```

---

exericse: what data cache accesses does this function do?

- 4-byte read of scaleFactor
- 8-byte read of return address

possible scaleFactor use

```
for (int i = 0; i < size; ++i) {
    array[i] = scaleByFactor(array[i]);
}
```

misses and code (2)

```
scaleByFactor:
    movl scaleFactor, %eax
    imull %edi, %eax
    ret
```

suppose each time this is called in the loop:

return address located at address `0x7ffffffe43b8`

scaleFactor located at address `0x6bc3a0`

with direct-mapped 32KB cache w/64 B blocks, what is their:

	return address	scaleFactor
tag		
index		
offset		

misses and code (2)

```
scaleByFactor:
    movl scaleFactor, %eax
    imull %edi, %eax
    ret
```

suppose each time this is called in the loop:

return address located at address `0x7ffffffe43b8`

scaleFactor located at address `0x6bc3a0`

with direct-mapped 32KB cache w/64 B blocks, what is their:

	return address	scaleFactor
tag	0xffffffc	0xd7
index	0x10e	0x10e
offset	0x38	0x20

misses and code (2)

```
scaleByFactor:
    movl scaleFactor, %eax
    imull %edi, %eax
    ret
```

suppose each time this is called in the loop:

return address located at address `0x7ffffffe43b8`

scaleFactor located at address `0x6bc3a0`

with direct-mapped 32KB cache w/64 B blocks, what is their:

	return address	scaleFactor
tag	0xffffffc	0xd7
index	0x10e	0x10e
offset	0x38	0x20

conflict miss coincidences?

obviously I set that up to have the same index

- have to use exactly the right amount of stack space...

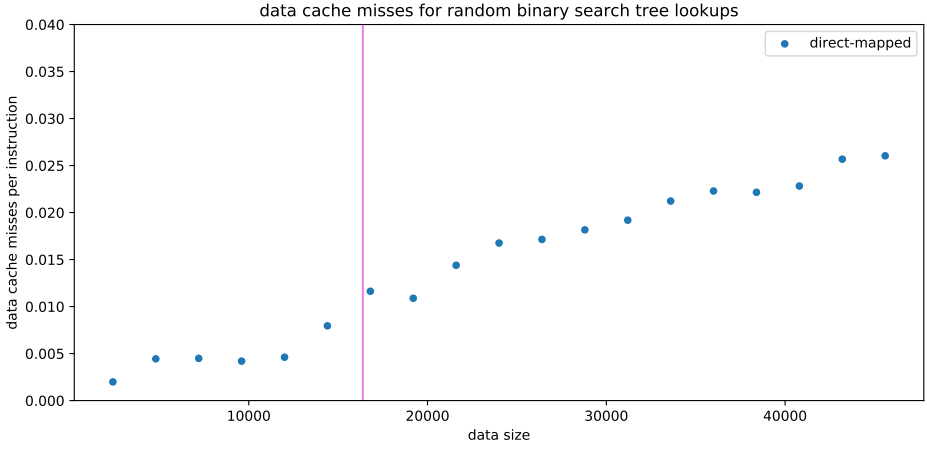
but gives one possible reason for conflict misses:

bad luck giving the same index for unrelated values

more direct reason: values related by power of two

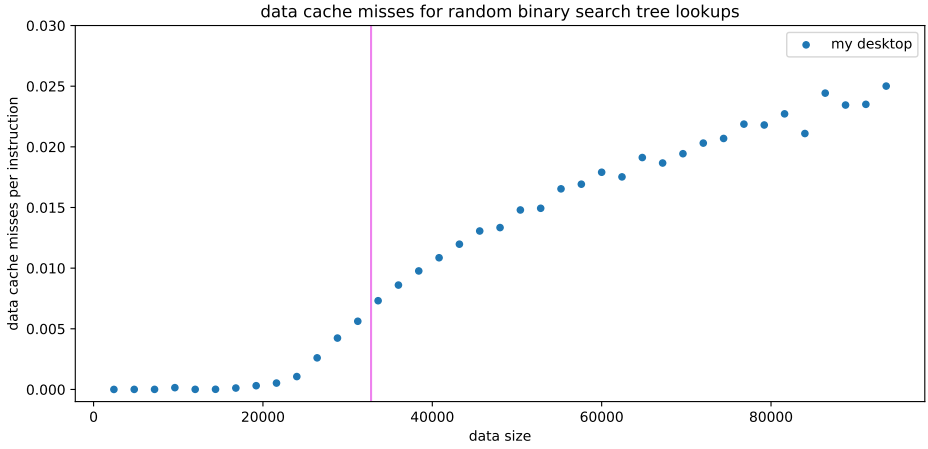
- some examples later, probably

simulated misses: BST lookups



(simulated 16KB direct-mapped data cache; excluding BST setup)

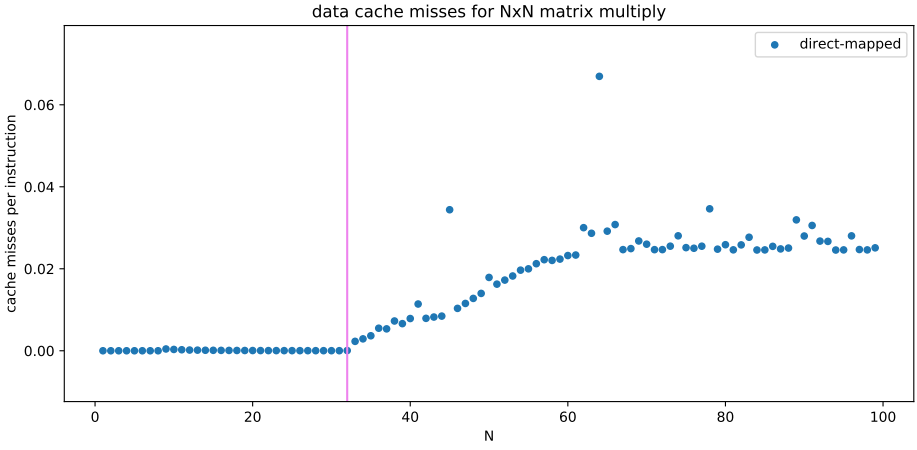
actual misses: BST lookups



(actual 32KB more complex data cache)

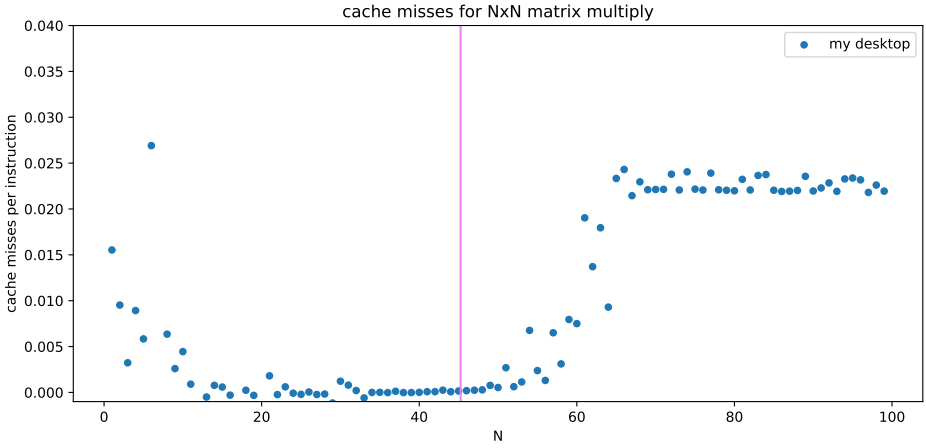
(only one set of measurements + other things on machine + excluding initial load)

simulated misses: matrix multiplies



(simulated 16KB direct-mapped data cache; excluding initial load)

actual misses: matrix multiplies



(actual 32KB more complex data cache; excluding matrix initial load)
(only one set of measurements + other things on machine)

example access pattern (1)

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

2 byte blocks, 4 sets

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		

multiple places to put values with same index
avoid conflict misses

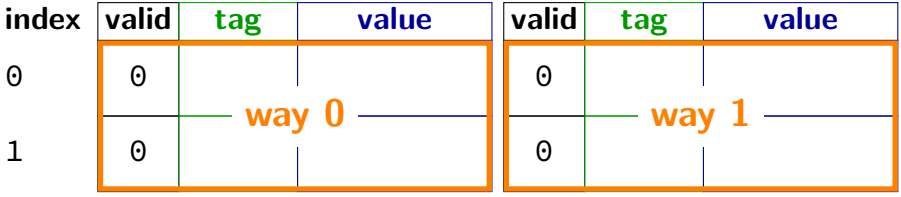
adding associativity

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



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	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

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

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

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

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

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)	

tag index offset

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

needs to replace block in set 0!

adding associativity

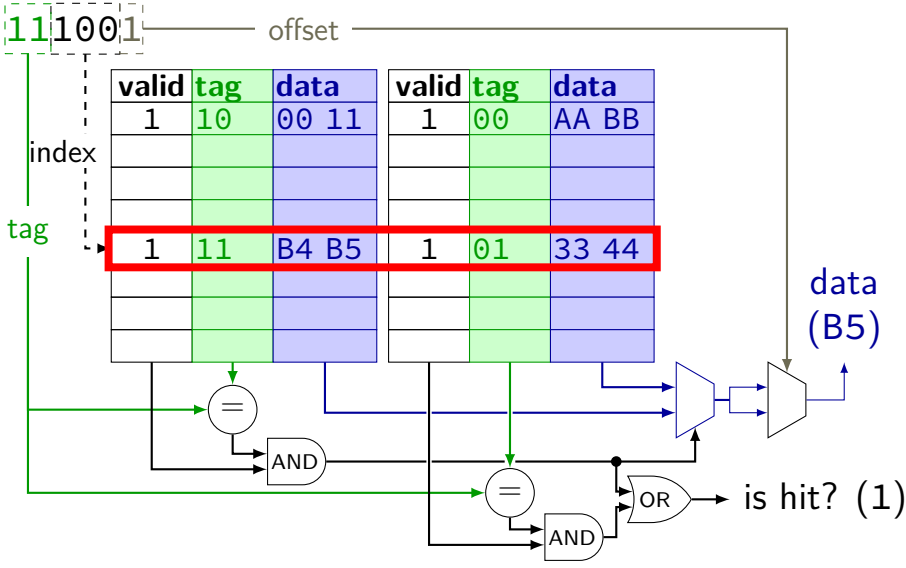
2-way set associative, 2 byte blocks, 2 sets

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		

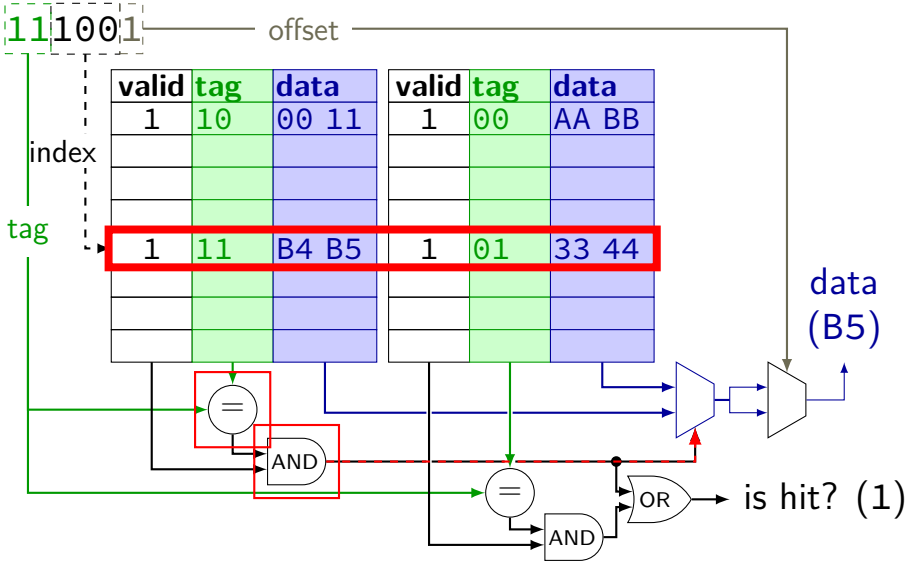
address (hex)	result
00000000 (00)	miss
00000001 (01)	hit
01100011 (63)	miss
01100001 (61)	miss
01100010 (62)	hit
00000000 (00)	hit
01100100 (64)	miss

tag index offset

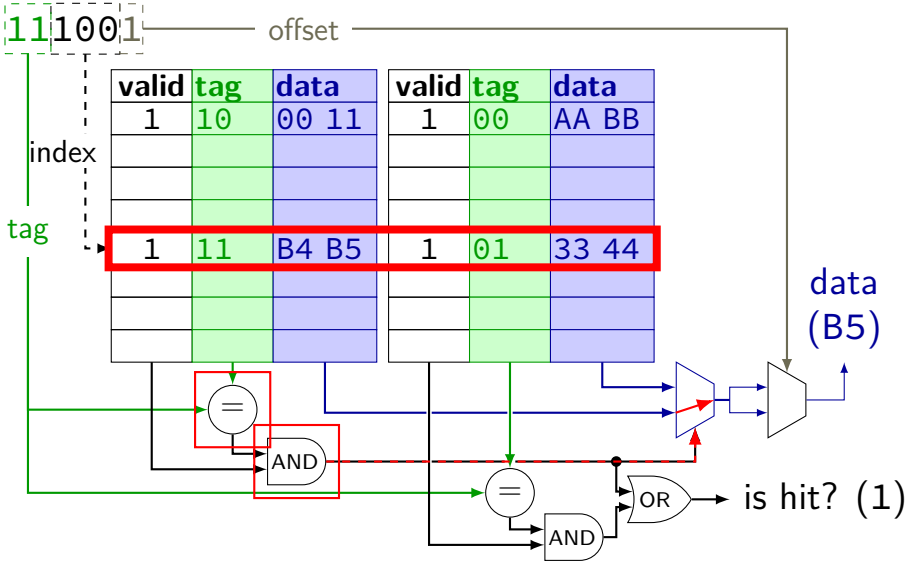
cache operation (associative)



cache operation (associative)



cache operation (associative)



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