CSCI 210: Computer Architecture
Lecture 35: Caches 3

Stephen Checkoway

Oberlin College

May 18, 2022

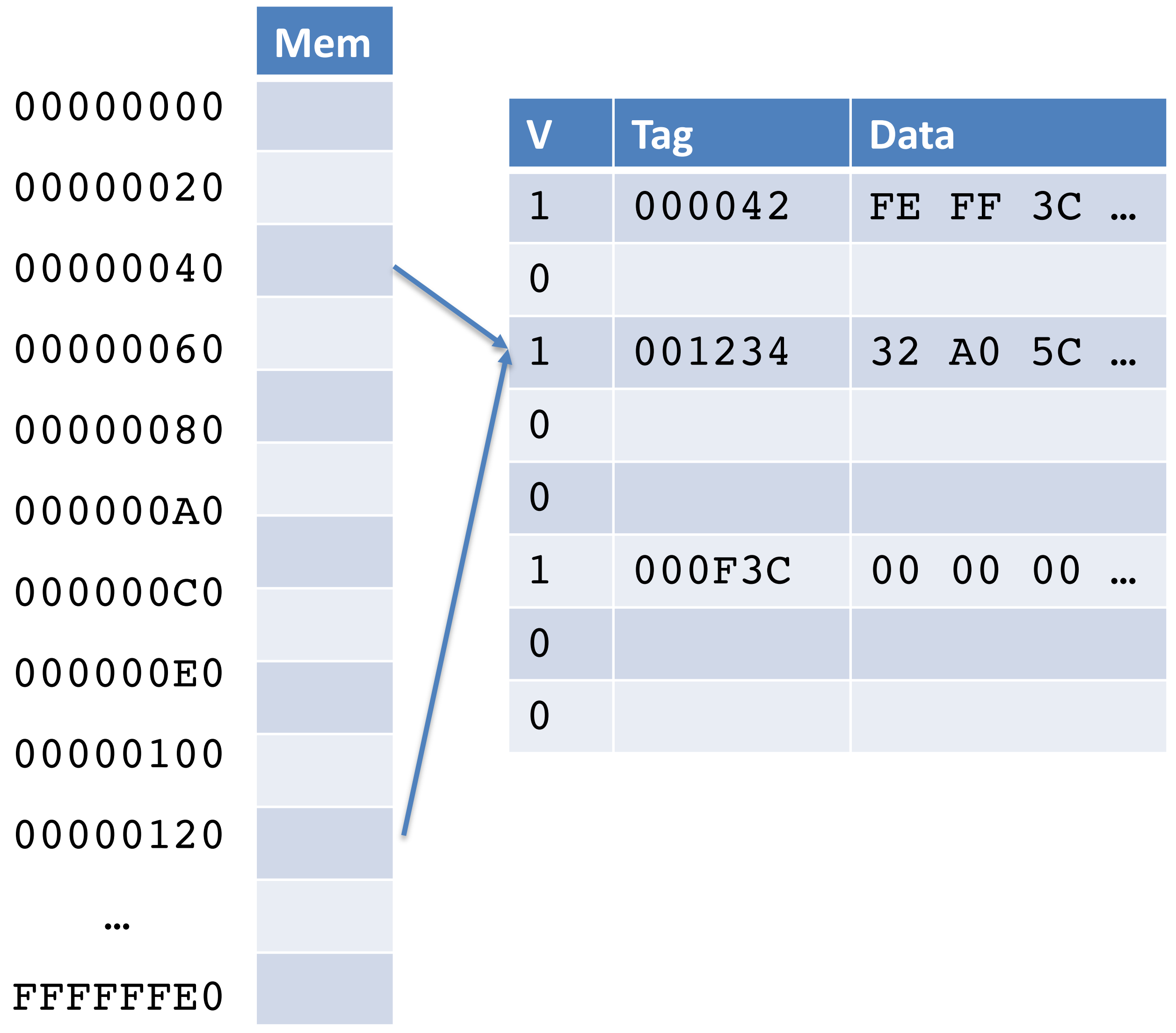
Slides from Cynthia Taylor

Announcements

- Problem Set 12 due one week from Thursday
- Cache Lab (final project) due Wednesday, Jun. 1 at 21:00
- Office Hours Friday 13:30 – 14:30

High-level cache strategy

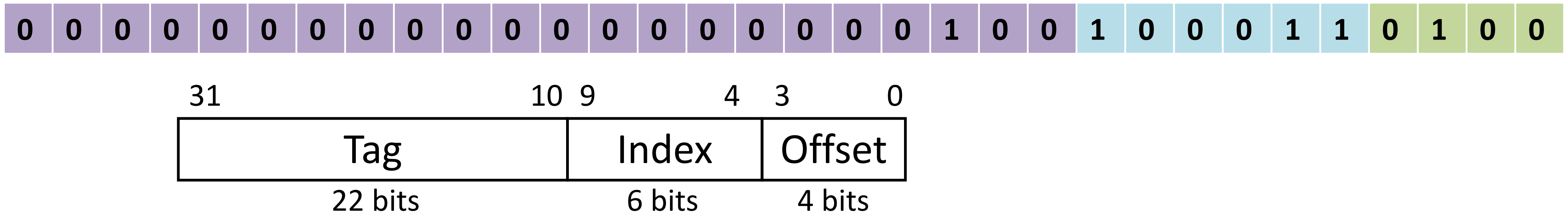
- Divide all of memory into consecutive blocks
- Copy data (memory ↔ cache) one block at a time
- Cache lookup:
  - Get the index of the block in the cache from the address
  - Check the valid bit; compare the tag to the address



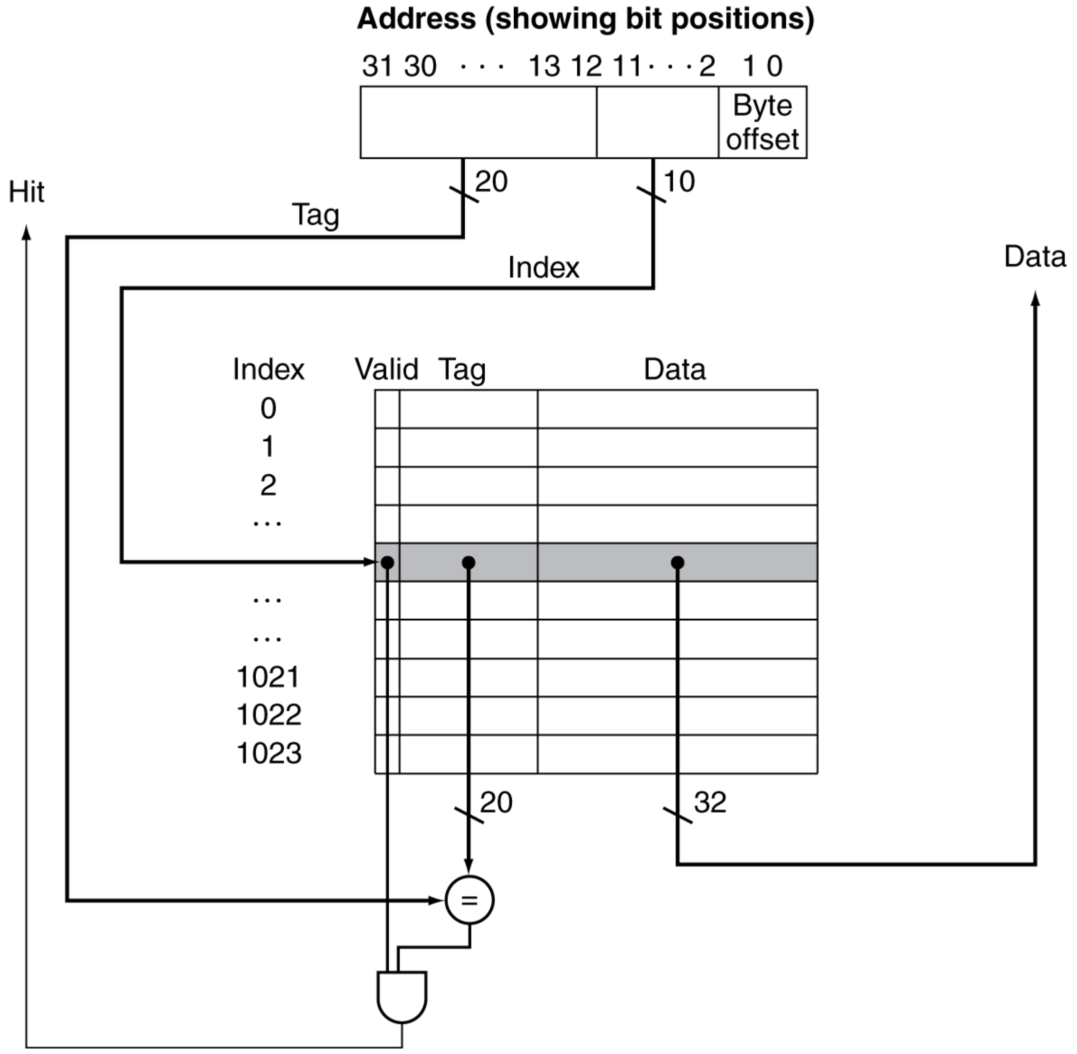
V	Tag	Data
1	000042	FE FF 3C …
0		
1	001234	32 A0 5C …
0		
0		
1	000F3C	00 00 00 …
0		
0		

Example

- 64 blocks, 16 bytes/block
  - To what cache index does address 0x1234 map?
- Block address = $\lfloor 0x1234/16 \rfloor$ = 0x123
- Index = 0x123 modulo 64 = 0x23
- No actual math required: just select appropriate bits from address!



Memory access



Direct Mapped Cache

data	byte addresses	A	B	C	D
x	00 00 01 00	M	M	M	M
y	00 00 10 00	M	M	M	H
z	00 00 11 00	M	M	M	M
x	00 00 01 00	H	H	H	H
y	00 00 10 00	H	H	H	H
w	00 01 01 00	M	M	M	M
x	00 00 01 00	M	M	H	H
y	00 00 10 00	H	H	H	H
w	00 01 01 00	H	M	H	H
u	00 01 10 00	M	M	M	M
z	00 00 11 00	H	H	M	H
y	00 00 10 00	H	M	H	H
x	00 00 01 00	H	M	M	M

E None are correct

	tag	data
00		
01		
10		
11		

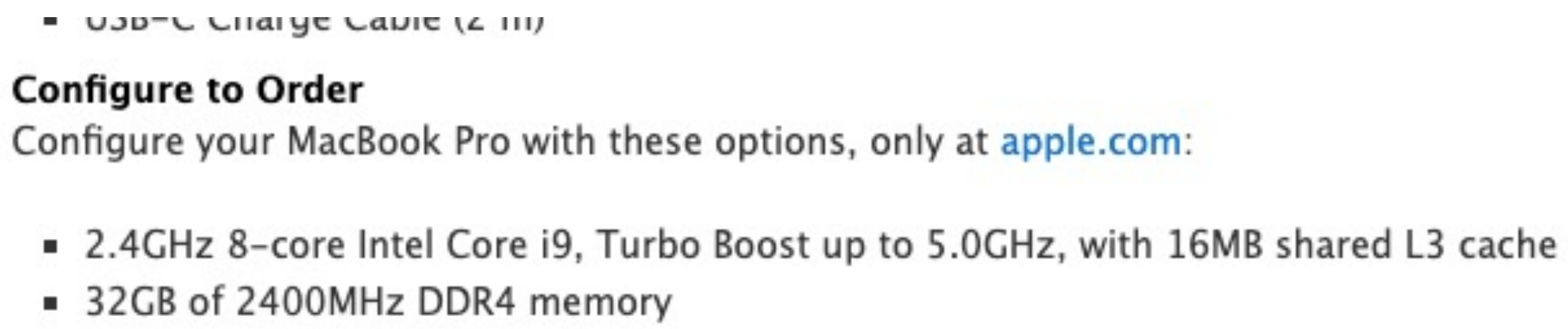
Four blocks, each block holds four bytes

How do we know how big a block in cache is?

A. Each block in the cache stores its size
B. The length of the tag in the cache determines the block size
C. The most significant bits of the address determine the block size
D. The least significant bits of the address determine the block size
E. For any given cache, the block size is constant

CACHE REPLACEMENT POLICIES

Cache Size vs Memory Size



- USB-C Charge Cable (2 m)

Configure to Order

Configure your MacBook Pro with these options, only at apple.com:

- 2.4GHz 8-core Intel Core i9, Turbo Boost up to 5.0GHz, with 16MB shared L3 cache
- 32GB of 2400MHz DDR4 memory

Memory is 2048 times bigger than cache

Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Cache replacement policy

- On a hit, return the requested data
- On a miss, load block from lower level in the memory hierarchy and write in cache; return the requested data
- Policy: Where in cache should the block be written? (With direct-mapped caches, there’s only one possible location: block_address % number_of_blocks_in_cache)

Cache policy for stores

- Policy choice for a hit: Where do we write the data?
  - Write-back: Write to cache only
  - Write-through: Write to cache and also to the next lowest level of the memory hierarchy
- Policy choice for a miss
  - Write-allocate: Bring the block into cache and then do the write-hit policy
  - Write-around: Write only to memory

Store-hit policy: write-through

- Update cache block AND memory
- Makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Store-hit policy: write-back

- Only update the block in cache
  - Keep track of whether each block is "dirty" (i.e., it has a different value than in memory)
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first
- Faster than write-through, but more complex

V	D	Tag	Data
1	0	000042	FE FF 3C …
0			
1	1	001234	65 82 5C …
0			
0			
1	0	000F3C	00 00 00 …
0			
0			

Store-miss policy: write-around

- Only write the data to memory

- Good for initialization where lots of memory is written at once but won’t be read again soon

Store-miss policy: write-allocate

- Read a block from memory (just like a load miss)
- Perform the write according to the store-hit policy (i.e., write in cache or write in both cache and memory)

- Good for when data is likely to be read shortly after being written (temporal locality)

Store Policies

- Given either high store locality or low store locality, which policies might you expect to find?
- Write-allocate: create block in cache. Write-around: don't create block. Write-through: update cache + memory. Write-back: update cache only.

Selection	**High Locality**		**Low Locality**	
	Miss Policy	**Hit Policy**	**Miss Policy**	**Hit Policy**
A	Write-allocate	Write-through	Write-around	Write-back
B	Write-around	Write-through	Write-allocate	Write-back
C	Write-allocate	Write-back	Write-around	Write-through
D	Write-around	Write-back	Write-allocate	Write-through
E	None of the above			

Common policy choices

- Write-back + write-allocate
  - Dirty blocks are written to memory only when replaced
  - Stores bring block into cache
  - Subsequent loads/stores will cause cache hits (unless the block is evicted)
- Write-through + write-around
  - Writes always go to memory
  - Cache is mostly for loads

Reading

- Next lecture: More Caches!
  - Section 6.4

- Cache Lab (final project) due at the time of the final exam (which this class doesn’t have)