

CENG 3420

Computer Organization & Design



Lecture 13: Cache

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Textbook: Chapters 5.3–5.4)

2024 Spring

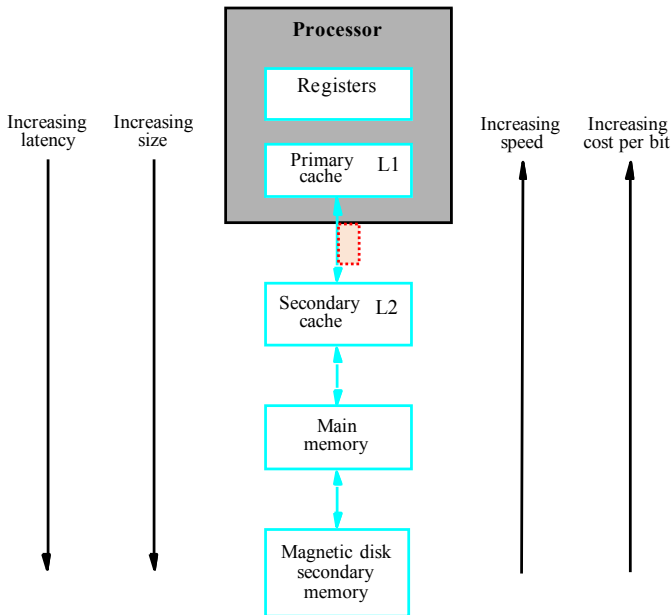


- ① Introduction
- ② Direct Mapping
- ③ Associative Mapping
- ④ Replacement
- ⑤ Conclusion



Introduction

- **Aim:** to produce fast, big and cheap memory
- L1, L2 cache are usually SRAM
- Main memory is DRAM
- Relies on **locality of reference**





- A way to record which part of the Main Memory is now in cache
- Synonym: Cache **line** == Cache **block**
- **Design concerns:**
 - Be **Efficient**: fast determination of cache hits/ misses
 - Be **Effective**: make full use of the cache; increase probability of cache hits

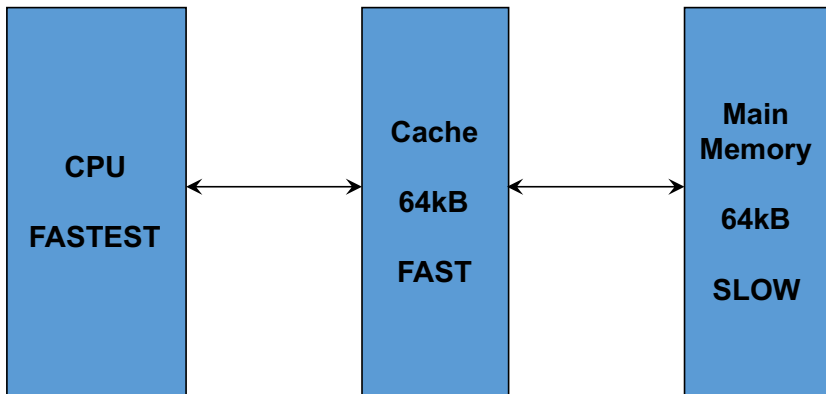
Two questions to answer (in hardware)

Q1 How do we know if a data item is in the cache?

Q2 If it is, how do we find it?

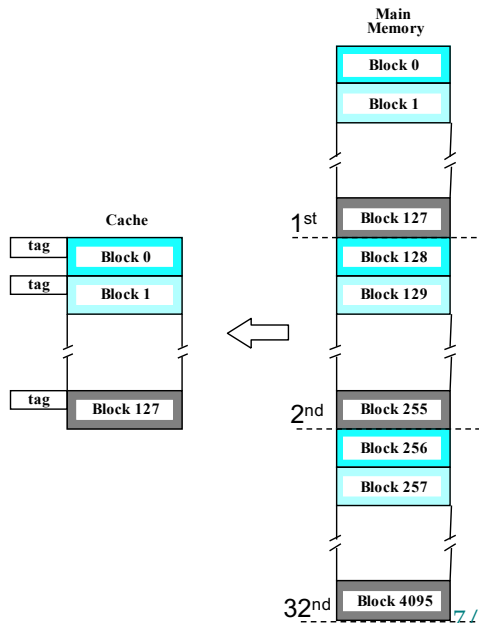


- Cache size == Main Memory size
- Trivial **one-to-one mapping**
- Do we need Main Memory any more?



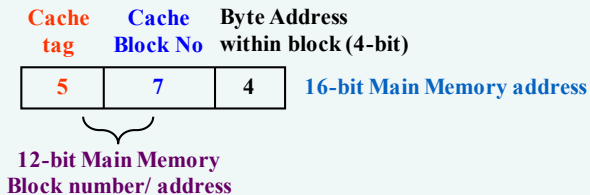


- Cache size is much smaller than the Main Memory size
- A block in the Main Memory maps to a block in the Cache
- **Many-to-One** Mapping

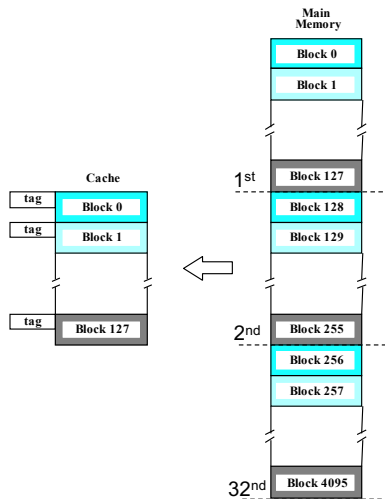


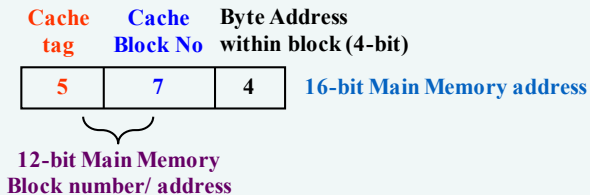


Direct Mapping

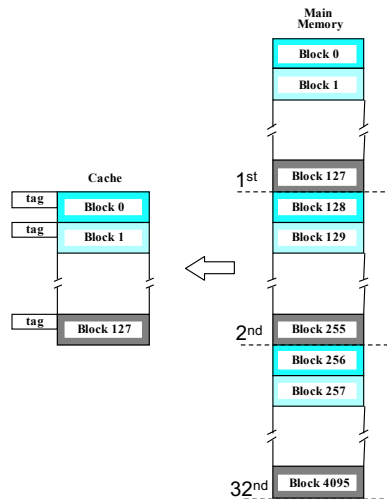


- $2^4 = 16$ bytes in a block
- $2^7 = 128$ Cache blocks
- $2^{(7+5)} = 4096$ main memory blocks





- $2^4 = 16$ bytes in a block
- $2^7 = 128$ Cache blocks
- $2^{(7+5)} = 4096$ main memory blocks



- Block j of main memory maps to block $(j \bmod 128)$ of Cache (same colour in figure)
- Cache **hit** occurs if **tag** matches desired address



Q: The cache block ID is determined by “red” or “blue” part on address?

Blue one (take advantage of spatial locality)

Q: what's the meaning of “tag” field on each cache block?

Indicate whether on cache block contains the correct data.



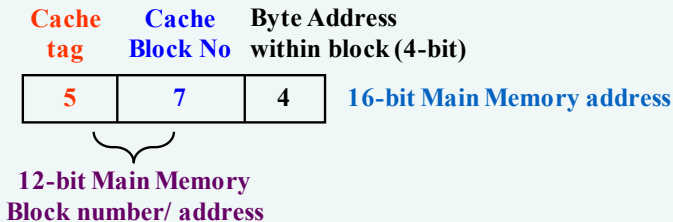
Memory address divided into 3 fields

- Main Memory **Block number** determines position of block in cache
- **Tag** used to keep track of which block is in cache (as many MM blocks can map to same position in cache)
- The **last bits** in the address selects target word in the block

Example: given an address (**t**,**b**,**w**) (16-bit)

- 1 See if it is already in cache by comparing **t** with the tag in block **b**
- 2 If not, cache miss! Replace the current block at **b** with a new one from memory block (**t**,**b**) (12-bit)

Direct Mapping Example 1



- 1 CPU is looking for [A7B4] MAR = 1010011110110100
- 2 Go to cache block 1111011, see if the tag is 10100
- 3 If YES, cache hit!
- 4 Otherwise, get the block into cache row 1111011



Q: what's the meaning using 4-bit byte address?

It means one block contains $2^4 = 16$ bytes.

Q: what's the meaning if all address is red + black (i.e. no cache block#)?

Only 1 Block & cache size $<$ memory size

Q: what's the meaning if all address is black (i.e. no cache tag or cache block#)?

Only 1 Block & cache size = memory size

Direct Mapping Example 2



Cache

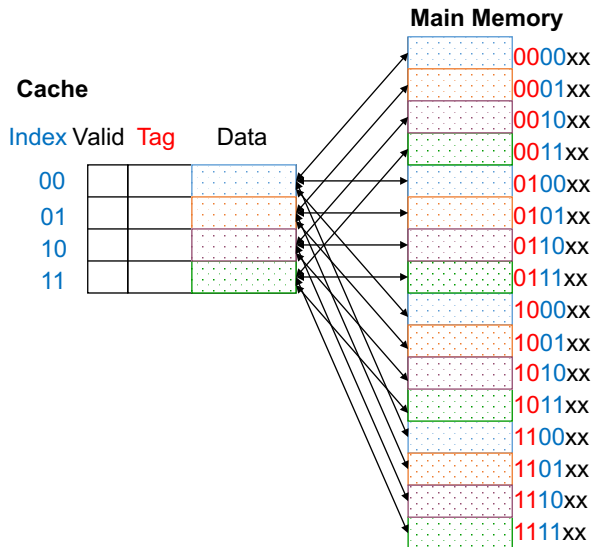
Index Valid Tag Data

00			
01			
10			
11			

Main Memory

	0000xx
	0001xx
	0010xx
	0011xx
	0100xx
	0101xx
	0110xx
	0111xx
	1000xx
	1001xx
	1010xx
	1011xx
	1100xx
	1101xx
	1110xx
	1111xx

Direct Mapping Example 2





Discussions

- In direct mapping: a block in main memory is always mapped to the same cache location.
- 0000xx & 0100xx conflict with each other. At most one of them can be mapped to cache.
- if a cache block is NOT mapped, valid=0.
- Initial state of the cache? All data 0; valid = 0
- why we want to use blue filed as cache index? Continuous blocks can be mapped to different cache blocks!







Question: Direct Mapping Cache Hit Rate

Consider a 4-block empty Cache, and all blocks initially marked as not valid. Given the main memory word addresses "0 1 2 3 4 3 4 15", calculate Cache hit rate.

Cache

Index Valid Tag Data

00			
01			
10			
11			



Discussions

0 00 00

1 00 01

2 00 10

3 00 11

4 01 00

15 11 11

Q: how we know whether address 3 is in cache?

We look at block index '11' and check whether its tag field is '00'



0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

11

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

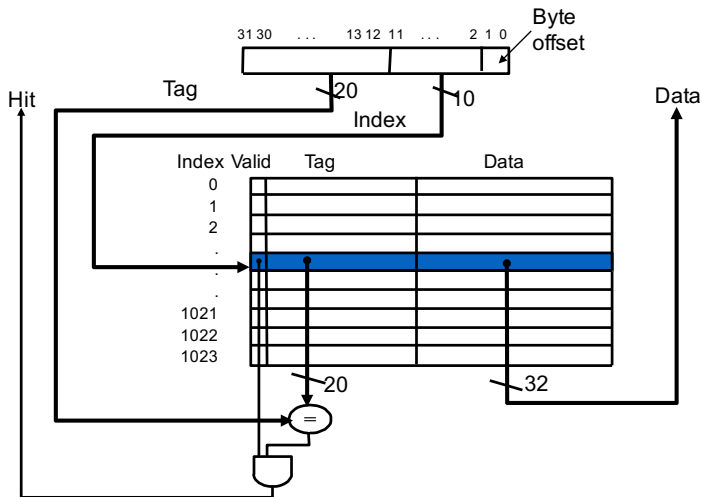
15

● 8 requests, 6 misses

Example 3: MIPS



- One word blocks, cache size = 1K words (or 4KB)
- What kind of locality are we taking advantage of?





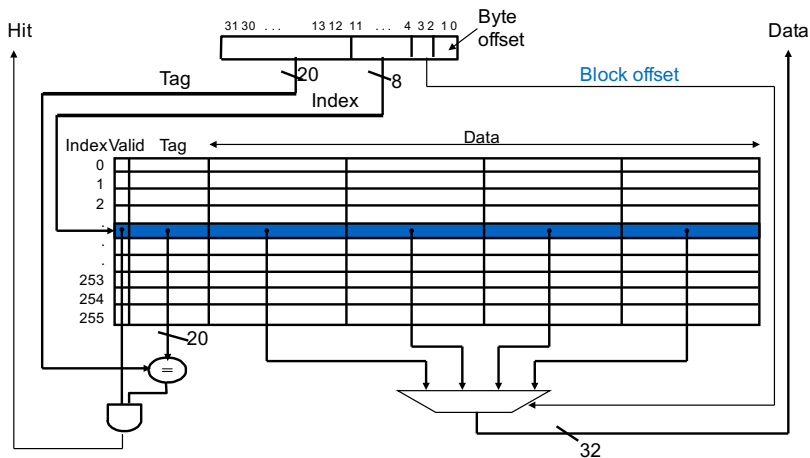
Notes:

- 1K lines \rightarrow 10 bits in address for block index
- 4 bytes in one block \rightarrow 2 bits in address for byte address
- all other bits are used for tag

Example 4: MIPS w. Multiword Block



- Four words/block, cache size = 1K words
- What kind of locality are we taking advantage of?





Question: Multiword Direct Mapping Cache Hit Rate

Consider a 2-block empty Cache, and each block is with 2-words. All blocks initially marked as `not valid`. Given the main memory word addresses “0 1 2 3 4 3 4 15”, calculate Cache hit rate.

Cache

Index	Tag	Data
00		
01		



Discussions

0 00 0 0

1 00 0 1

2 00 1 0

3 00 1 1

4 01 0 0

15 11 1 1



0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

● 8 requests, 4 misses



The number of bits includes both the storage for data and for the tags

- For a direct mapped cache with 2^n blocks, n bits are used for the index
- For a block size of 2^m words (2^{m+2} bytes), m bits are used to address the word within the block
- 2 bits are used to address the byte within the word



The number of bits includes both the storage for data and for the tags

- For a direct mapped cache with 2^n blocks, n bits are used for the index
- For a block size of 2^m words (2^{m+2} bytes), m bits are used to address the word within the block
- 2 bits are used to address the byte within the word

Size of the tag field?

$$32 - (n + m + 2)$$



The number of bits includes both the storage for data and for the tags

- For a direct mapped cache with 2^n blocks, n bits are used for the index
- For a block size of 2^m words (2^{m+2} bytes), m bits are used to address the word within the block
- 2 bits are used to address the byte within the word

Size of the tag field?

$$32 - (n + m + 2)$$

Total number of bits in a direct-mapped cache

$$2^n \times (\text{block size} + \text{tag field size} + \text{valid field size})$$



Question: Bit number in a Cache

How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?



Question: Bit number in a Cache

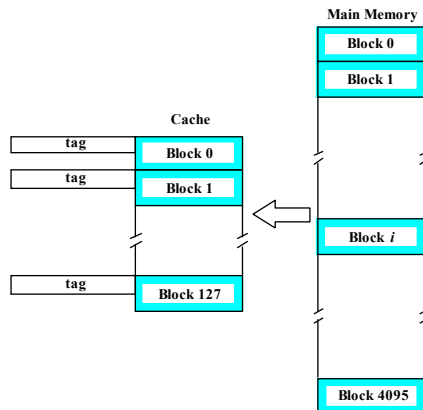
How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?

Solution

- 16K bytes == 4K words == 1K blocks
- Tag field size = $32 - (10 + 2 + 2) = 18$
- $2^{10} \times [4 \times 32 + 18 + 1] = 2^{10} \times 147 = 147 \text{ Kbits}$



Associative Mapping



- An MM block can be in **arbitrary** Cache block location
- In this example, all 128 tag entries must be compared with the address **Tag** in parallel (by hardware)

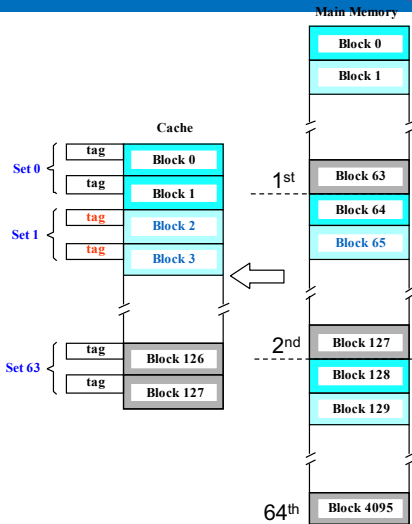
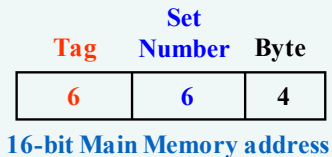


Tag	Byte
12	4

16-bit Main Memory address

- 1 CPU is looking for [A7B4] MAR = 1010011110110100
- 2 See if the tag 101001111011 matches one of the 128 cache tags
- 3 If YES, cache hit!
- 4 Otherwise, get the block into BINGO cache row

Set Associative Mapping



Example: 2-way set associative

- Combination of direct and associative
- $(j \bmod 64)$ derives the Set Number
- A cache with k-blocks per set is called a **k-way** set associative cache.



Notations

Set : how to partition the cache (e.g. 2-set means cache is divided into 2 parts)

Way : # of block in one set (e.g. 2-way means one set has 2 blocks)



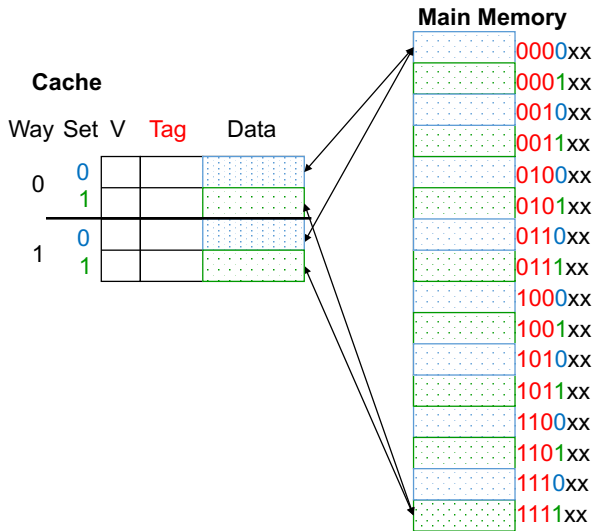
Tag	Set Number	Byte
6	6	4

16-bit Main Memory address

E.g. 2-Way Set Associative:

- 1 CPU is looking for [A7B4] MAR = 1010011110110100
- 2 Go to cache Set 111011 (59_{10})
 - Block 1110110 (118_{10})
 - Block 1110111 (119_{10})
- 3 See if ONE of the TWO tags in the Set 111011 is 101001
- 4 If YES, cache hit!
- 5 Get the block into BINGO cache row

Set Associative Mapping Example 2





Question: Direct Mapping v.s. 2-Way Set Associate

Consider the following two empty caches, calculate Cache hit rates for the reference word addresses: "0 4 0 4 0 4 0 4"

Cache

Index	Valid	Tag	Data
00			
01			
10			
11			

(a)

Cache

Set	Tag	Data
0		
1		
0		
1		

(b)

(a) Direct Mapping; (b) 2-Way Set Associative.



Question: Direct Mapping v.s. 2-Way Set Associate

Consider the following two empty caches, calculate Cache hit rates for the reference word addresses: "0 4 0 4 0 4 0 4"

Cache

Index	Valid	Tag	Data
00			
01			
10			
11			

(a)

Cache

Set	Tag	Data
0		
1		
0		
1		

(b)

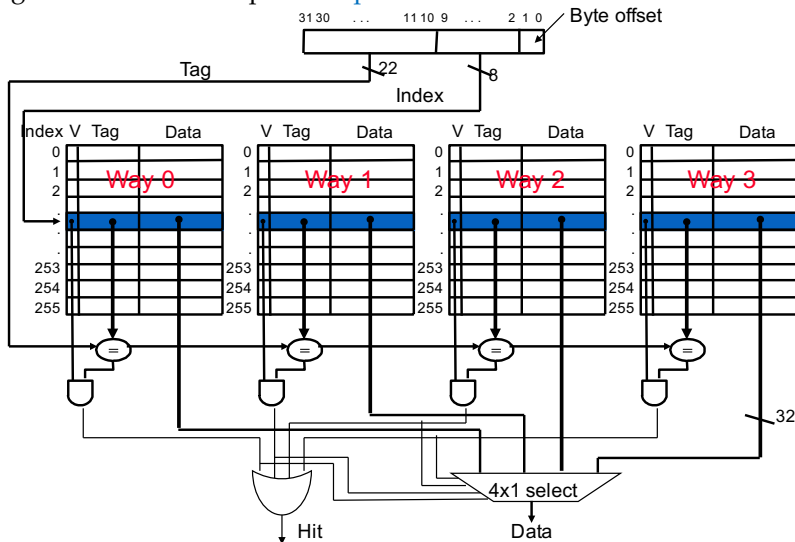
(a) Direct Mapping; (b) 2-Way Set Associative.

- Direct mapping: 0 hit (Ping pong effect)
- 2-Way Set Associative: 6 hits

Set Associative Mapping Example 3: MIPS

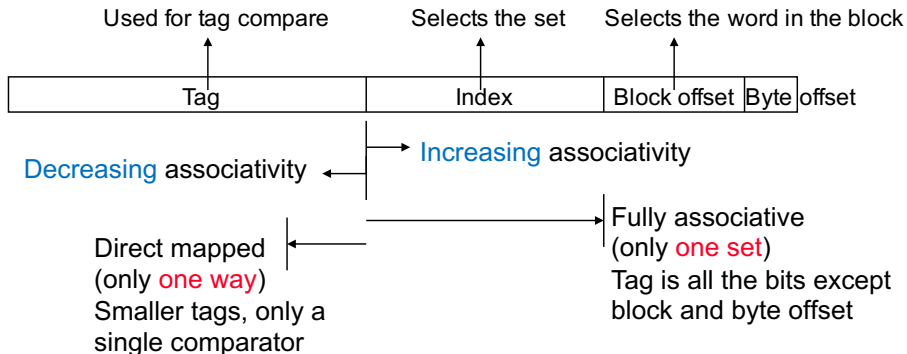


- $2^8 = 256$ sets each with **four** ways (each with one block).
- four tags in the set are compared in **parallel**.





For a fixed size cache:





Replacement



- I\$ and D\$
- Read **hit**: what we want!
- Read **miss**: **stall** the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume.



Only D\$

Case 1: Write-Through

- Cache and memory to be **consistent**
- always write the data into both the cache block and the next level in the memory hierarchy
- Speed-up: use **write buffer** and stall only when buffer is full

Case 2: Write-Back

- Write the data **only** into the cache block
- Write to memory hierarchy when that cache block is “**evicted**”
- Need a **dirty** bit for each data cache block



Case 1: Write-Through caches with a write buffer

- No-write allocate¹
- skip cache write (but must invalidate that cache block since it now holds stale data)
- just write the word to the write buffer (and eventually to the next memory level)
- no need to stall if the write buffer isn't full

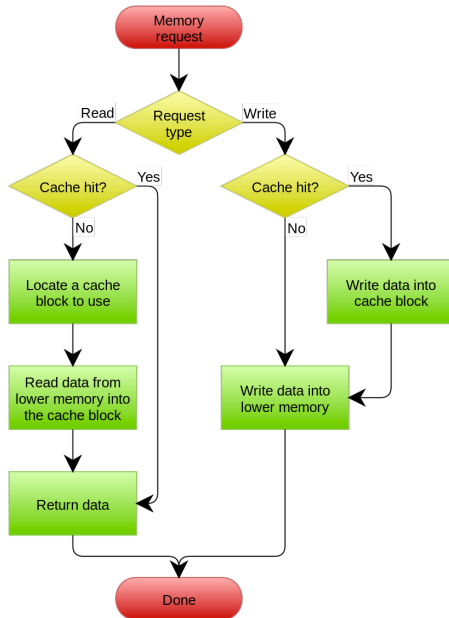
Case 2: Write-Back caches

- Write allocate²
- Just write the word into the cache updating both the tag and data
- no need to stall

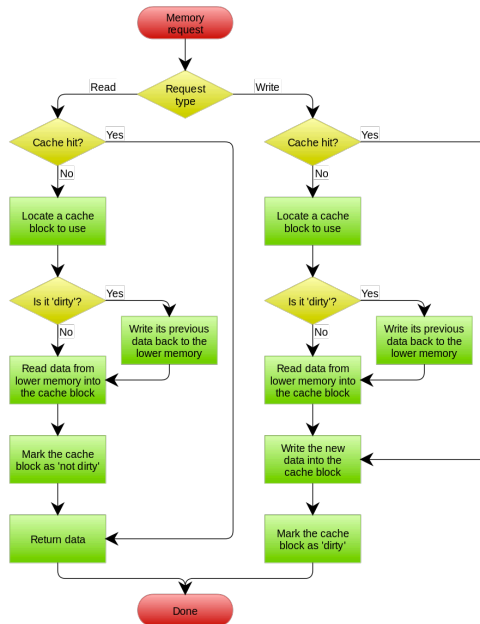
¹The block is modified in the main memory and not loaded into the cache.

²The block is loaded on a write miss, followed by the write-hit action.

Write-Through Cache with No-Write Allocation



Write-Back Cache with Write Allocation





Direct Mapping

- Position of each block fixed
- Whenever replacement is needed (i.e. cache miss \rightarrow new block to load), the choice is obvious and thus **no** “replacement algorithm” is needed

Associative and Set Associative

- Need to decide which block to replace
- Keep/retain ones likely to be used in near future again



Strategy 1: Least Recently Used (LRU)

- e.g. for a 4-block/set cache, use a $\log_2 4 = 2$ bit counter for each block
- Reset the counter to 0 whenever the block is accessed
- counters of other blocks in the same set should be incremented
- On cache miss, replace/ uncache a block with counter reaching 3



Strategy 1: Least Recently Used (LRU)

- e.g. for a 4-block/set cache, use a $\log_2 4 = 2$ bit counter for each block
- Reset the counter to 0 whenever the block is accessed
- counters of other blocks in the same set should be incremented
- On cache miss, replace/ uncache a block with counter reaching 3

Strategy 2: Random Replacement

- Choose random block
- 😊 Easier to implement at high speed



- Cache Organizations:
Direct, Associative, Set-Associative
- Cache Replacement Algorithms:
Random, Least Recently Used
- Cache Hit and Miss Penalty