



信息科学与技术学院

School of Information Science and Technology

CS 110

Computer Architecture

Cache II Fully Associative Cache

Instructors:

Chundong Wang, Siting Liu & Yuan Xiao

Course website: <https://toast->

[lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html](https://toast-lab.sist.shanghaitech.edu.cn/courses/CS110@ShanghaiTech/Spring-2025/index.html)

School of Information Science and Technology (SIST)

ShanghaiTech University

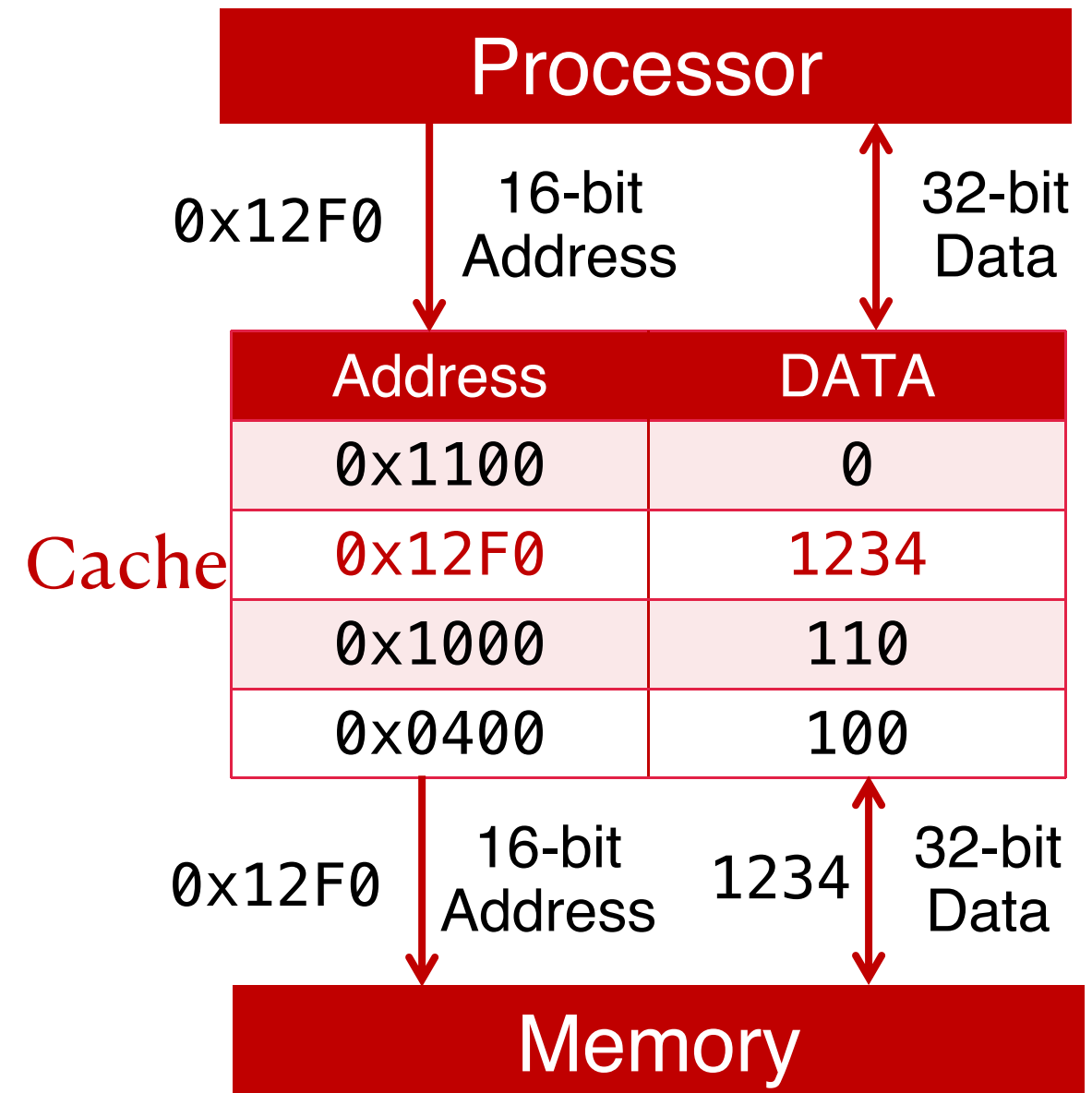
2025/4/17

Administratives

- Mid-term I score published. Please check if there are any questions regarding your marks. Submit regrade request if you have any concerns before this **FRIDAY (Apr. 18th 23:59:59)**! Score change after this date is not possible.
- Project 2.1 released, start early!!!
- HW4 ddl **TODAY**! Submit your answer to gradescope.
- HW5 will be released today, ddl Apr. 30th.
- Project 1.2 ddl **TODAY**!
- Lab 8 checking this week. Lab 9 will be released and checked next week.

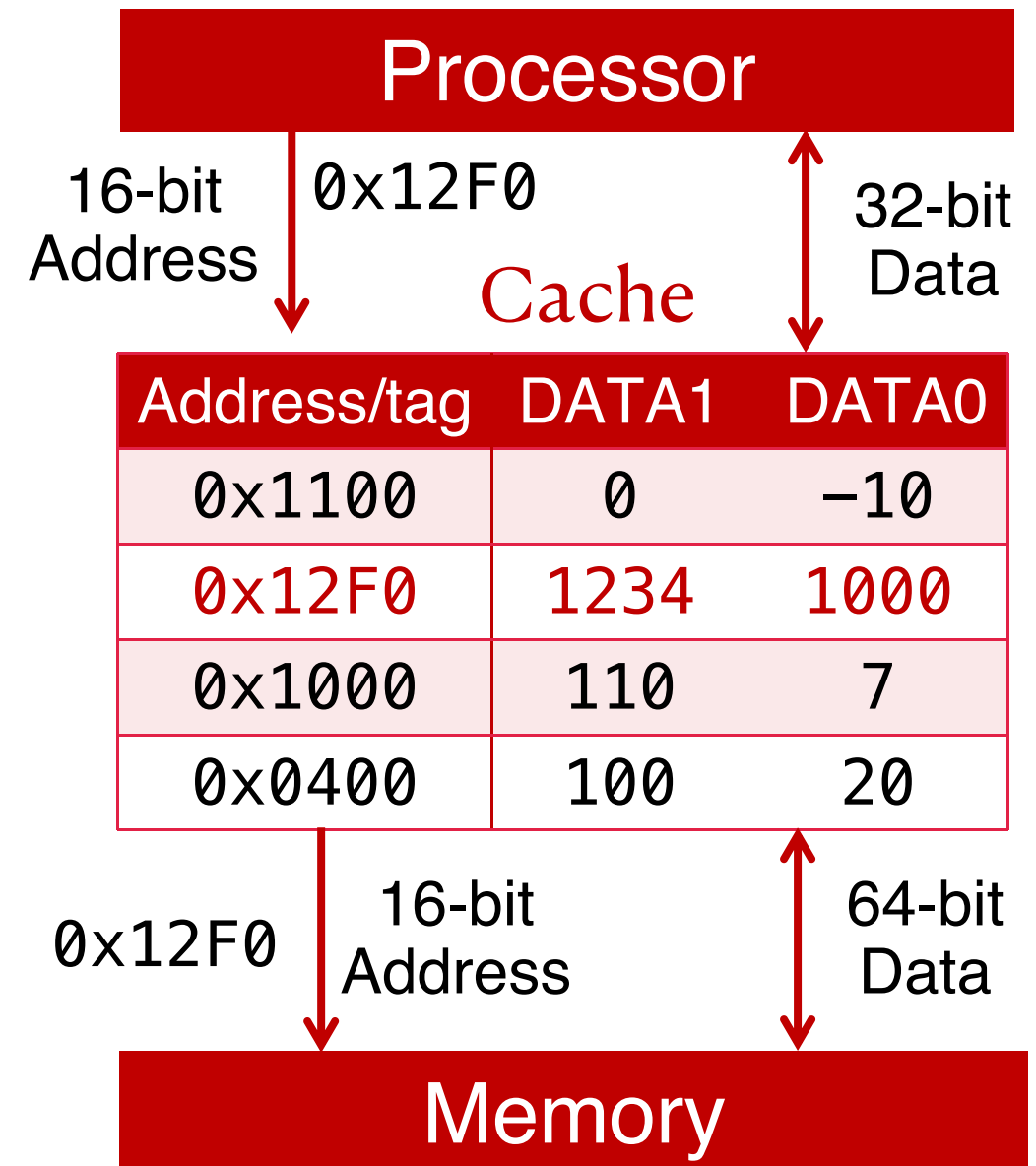
Block Alignment

- From now on, for ease of discussion, we **ASSUME A 16-BIT ADDRESS!!!**
 - Word blocks are aligned, so binary address of all words in cache always ends in 00_{two}
 - How to take advantage of this to save hardware and energy?
 - Don't need to compare last 2 bits of 32-bit byte address (comparator can be narrower)
- ==> Don't need to store last 2 bits of 32-bit byte address in Cache Tag (Tag can be narrower)



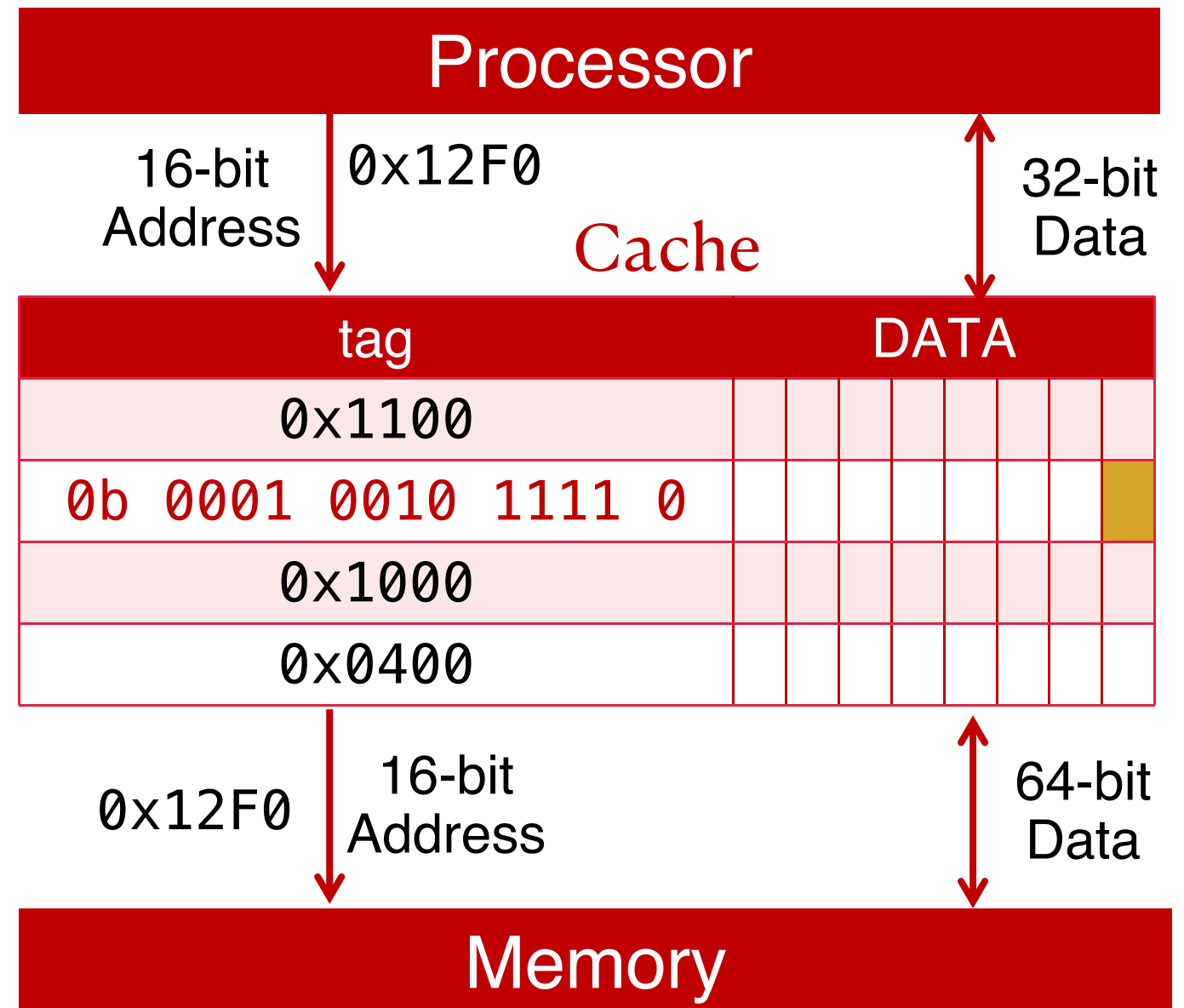
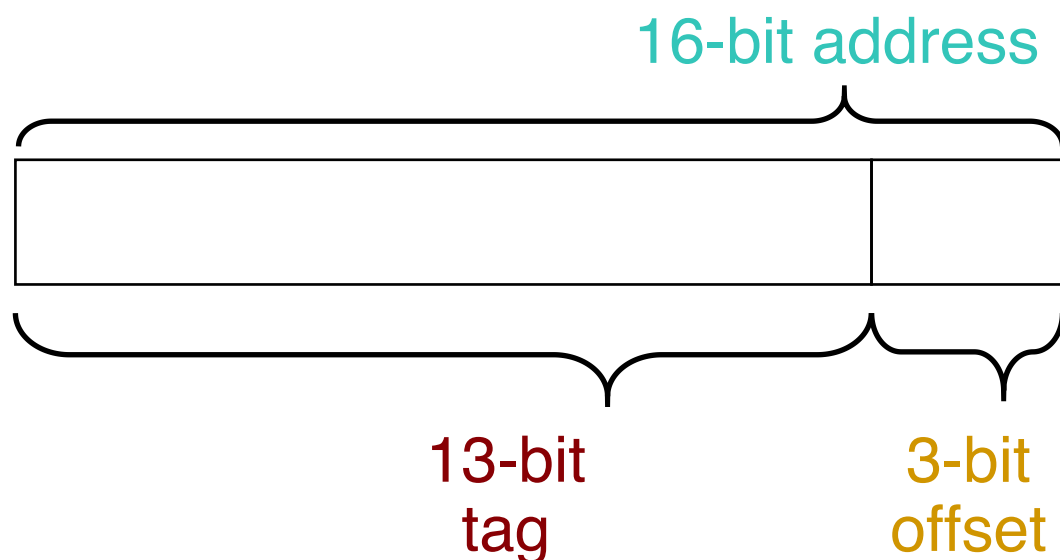
A 32B Cache with 8B Block

- Blocks must be aligned in pairs, otherwise could get same word twice in cache;
- Tags only have even-numbered words;
- Last 3 bits of address always 000_{two} ;
- Tags, comparators can be narrower;
- Can get hit for either word in block.



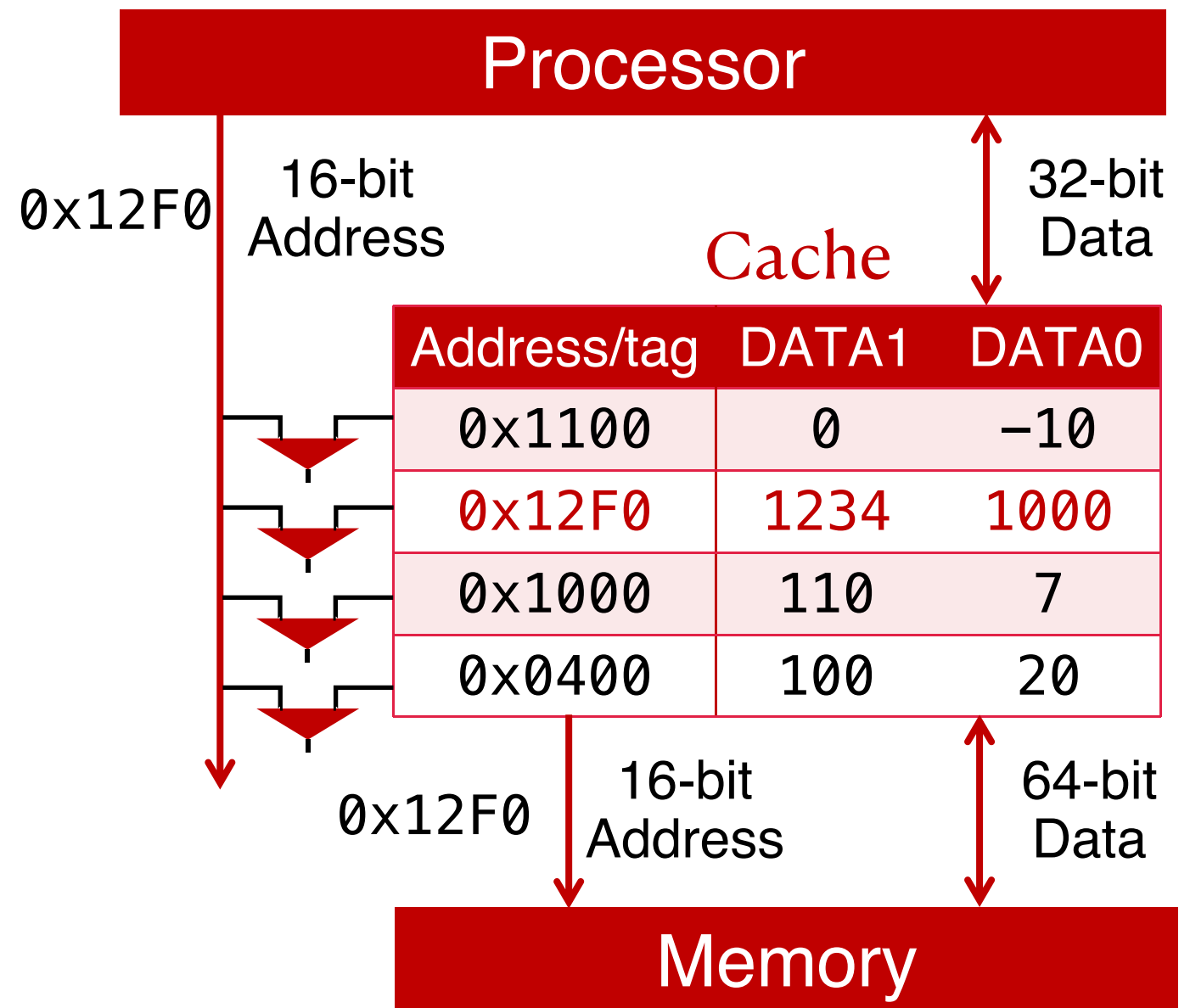
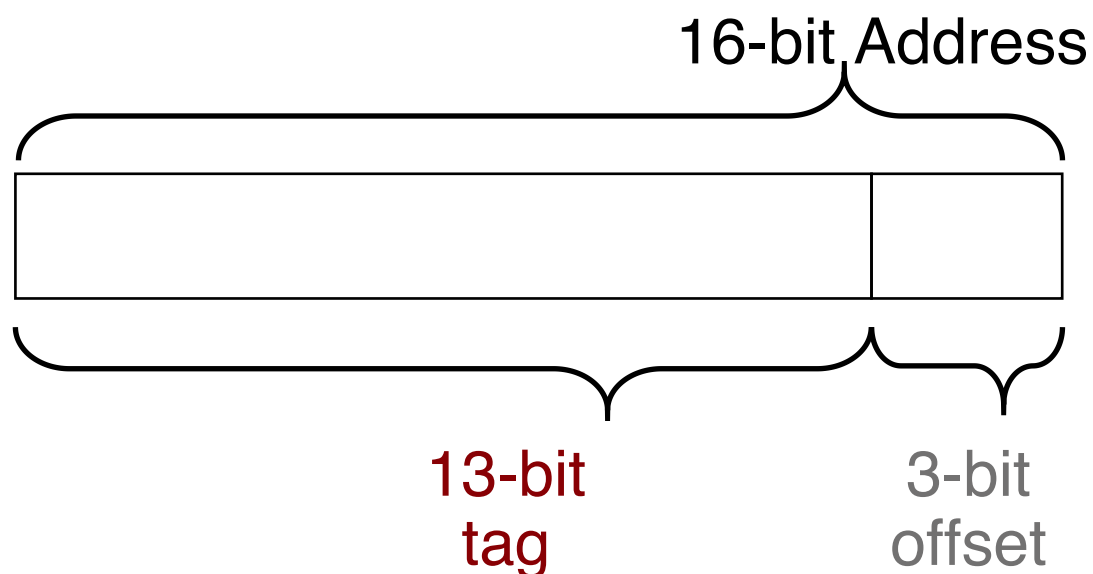
1b Example

- `lb t0, 0(t1)`
- `t1` 0x12F0
- High 13 bits are used to compare to the tags;
 - Hit!!!
 - Fetch data in that cache block
 - Low 3 bits are used as **offset (0b000)**



Hardware Implementation

- Need to compare each tag and thus **#cache block** comparators to decide hit or not;
- Comparators are expensive;
- **Fully associative:**
 - Arbitrary memory address can go to arbitrary cache blocks



Terminology for Fully Associative Cache

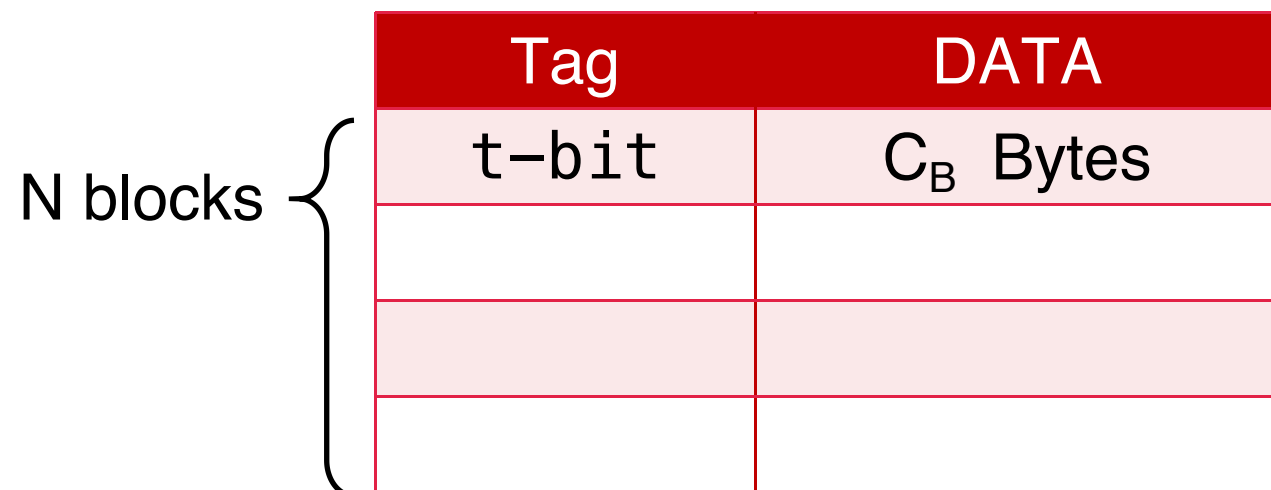
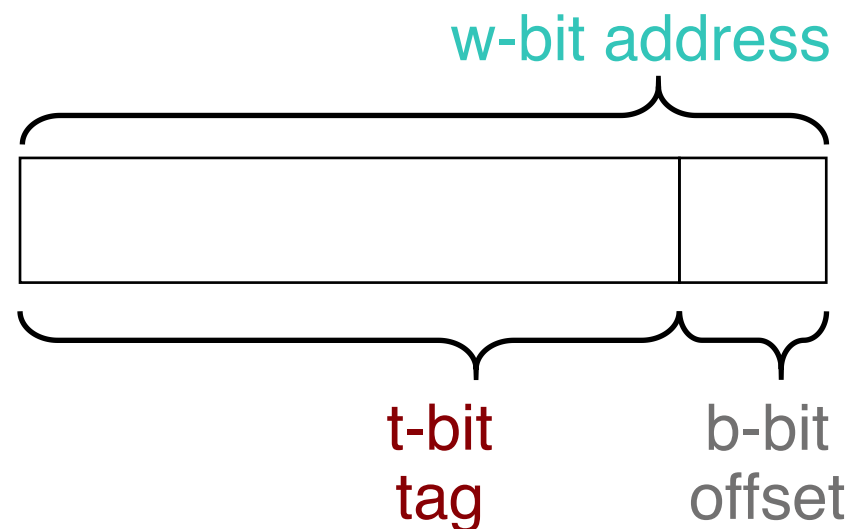
- Cache size: total size of the cache (C)
- Cache block size: $C_B \rightarrow$ decides the number of offset bits (b)

$$2^b = C_B$$

- Number of cache blocks (#cache block, N)

$$N * C_B = C$$

- Bit width of memory address (w): 16-bit in our examples
- Bit width of Tag (t): $t = w - b$
- Hardware implication: comparators? actual storage requirement?



Warm Up a Fully Associative Cache

- **Valid bit**
- We need an indicator (i.e., flag) to tell if each entry is valid for this particular program.
 - Valid bit: indicates if data stored at a cache line is valid;
 - 1 = valid, 0 = invalid
- Cold start: Empty cache with valid bit 0; initial state of the cache

Tag	valid bit	DATA
t-bit	0	C_B Bytes

Run Assembly to Heat the Cache

4B cache block

- 1. Load byte at address 0x043F
 - Tag: 0b 0000 0100 0011 11
 - Offset: 11
 - Check valid bits and compare the tags: cache miss!
 - Cache sends 0x043C to the main memory;

Tag	valid bit	DATA			
	0	11	10	01	00
	0				
	0				
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at address 0x043F
 - Tag: 0b 0000 0100 0011 11
 - Offset: 11
 - Check valid bits and compare the tags: cache miss!
 - Cache sends 0x043C to the main memory;
 - Memory data (e.g. 0x12342587) filled in the first cache block with tag recorded and valid bit set 1;

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
	0				
	0				
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at address 0x043F
 - Tag: 0b 0000 0100 0011 11
 - Offset: 11
 - Check valid bits and compare the tags: cache miss!
 - Cache sends 0x043C to the main memory;
 - Memory data (e.g. 0x12342587) filled in the first cache block with tag recorded and valid bit set 1;
 - Use offset to index the 3rd byte from the first cache block;
 - Cache send 0x12 to the processor.

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
	0				
	0				
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at address 0x043F
- 2. Load byte at address 0x1234
 - Tag: 0b 0001 0010 0011 01
 - Offset: 00
 - Check valid bits and compare the tags: cache miss!
 - Cache sends 0x1234 to the main memory;
 - Memory data (e.g. 0x87654321) filled in the second cache block with tag recorded and valid bit set to 1.
 - Use offset to index ...

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
	0				
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at 0x43F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
 - Tag: 0b0...0 0010 00
 - Offset: 10
 - Check valid bits and compare the tags: cache miss!
 - Cache sends 0x0020 to the main memory;
 - Memory data (e.g. 0x12345678) filled in the second cache block with tag recorded and valid bit set to 1;
 - Use offset to index the 2nd byte;
 -

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
 - Tag: 0b 0000 0100 0011 11
 - Offset: 00
 - Check valid bits and compare the tags: cache **hit**!
 - Use offset to index the 0th byte;
 - Cache sends 0x12342587 to the processor;

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
 - Assume memory[0x0F40] = 0xFFFFFFFF;

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
	0				

Run Assembly to Heat the Cache (Cont'd)

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
 - Assume memory[0x0F40] = 0xFFFFFFFF;

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
0x03D0	1	FF	FF	FF	FF

Block Replacement

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
 - Assume memory[0x0F40] = 0xFFFFFFFF;
- Cache is full
 - If there is another cache miss, a victim will be evicted from the cache;

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
0x03D0	1	FF	FF	FF	FF

Block Replacement (Cont'd)

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
0x03D0	1	FF	FF	FF	FF

Block Replacement (Cont'd)

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120
 - Cache miss and we need to replace one cache block with the data at 0x0120;

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
0x03D0	1	FF	FF	FF	FF

Block Replacement Policy

- Least recently used (LRU)
 - Replace the entry that has not been used for the longest time, i.e., has the oldest previous access.
 - Pro: **Temporal locality!**
 - Recent past use implies likely future use
 - Con: Complicated hardware to keep track of access history

Tag	valid bit	DATA			
0x010F	1	12	34	25	87
0x048D	1	87	65	43	21
0x0008	1	12	34	56	78
0x03D0	1	FF	FF	FF	FF

LRU Implementation

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - 0 indicates the highest priority (the most recently used);
 - 3 indicates the least recently used.

Tag	VB	LRU	DATA			
0x010F	1		12	34	25	87
0x048D	1		87	65	43	21
0x0008	1		12	34	56	78
0x03D0	1		FF	FF	FF	FF

LRU Example

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.
- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

Tag	VB	LRU	DATA			
	0					
	0					
	0					
	0					

LRU Example (Cont'd)

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.
- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

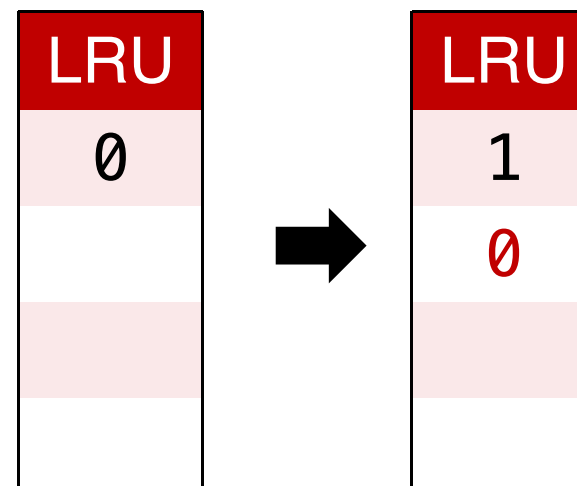
Tag	VB	LRU	DATA			
0x010F	1	0	12	34	25	87
	0					
	0					
	0					

LRU
0

LRU Example (Cont'd)

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.
- 1. Load byte at 0x043F
- **2. Load byte at 0x1234**
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

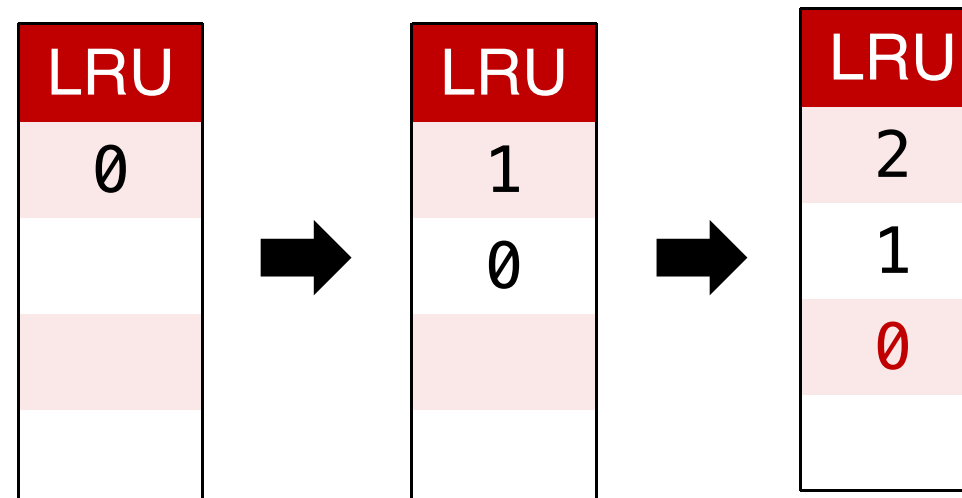
Tag	VB	LRU	DATA			
0x010F	1	1	12	34	25	87
0x048D	1	0	87	65	43	21
	0					
	0					



LRU Example (Cont'd)

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.
- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- **3. Load halfword at 0x0022**
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

Tag	VB	LRU	DATA			
0x010F	1	2	12	34	25	87
0x048D	1	1	87	65	43	21
0x0008	1	0	12	34	56	78
	0					

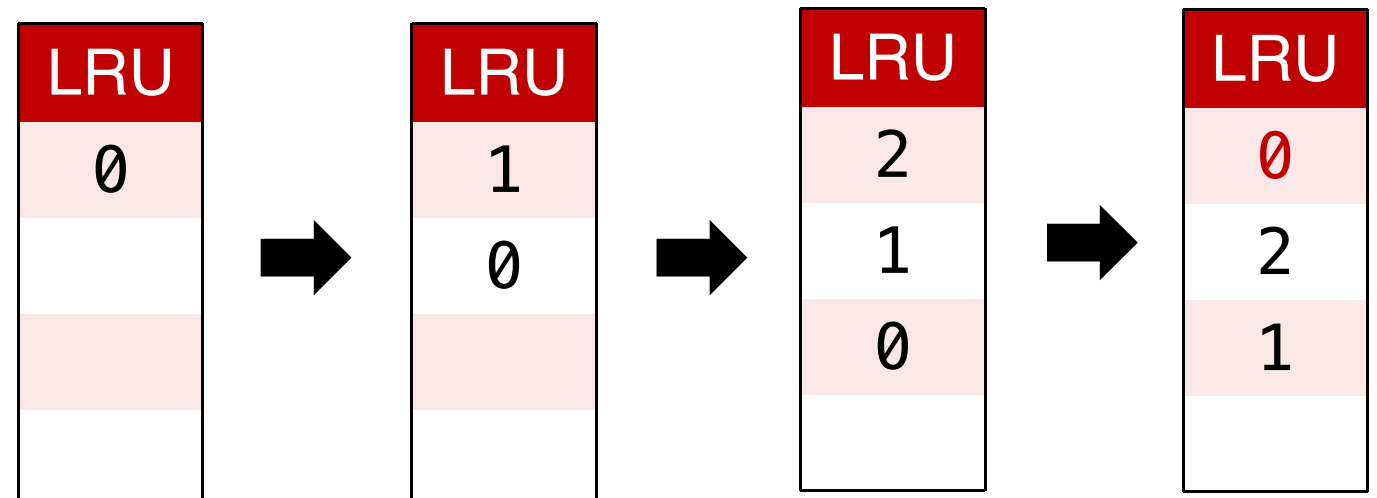


LRU Example (Cont'd)

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.

Tag	VB	LRU	DATA			
0x010F	1	0	12	34	25	87
0x048D	1	2	87	65	43	21
0x008	1	1	12	34	56	78
	0					

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

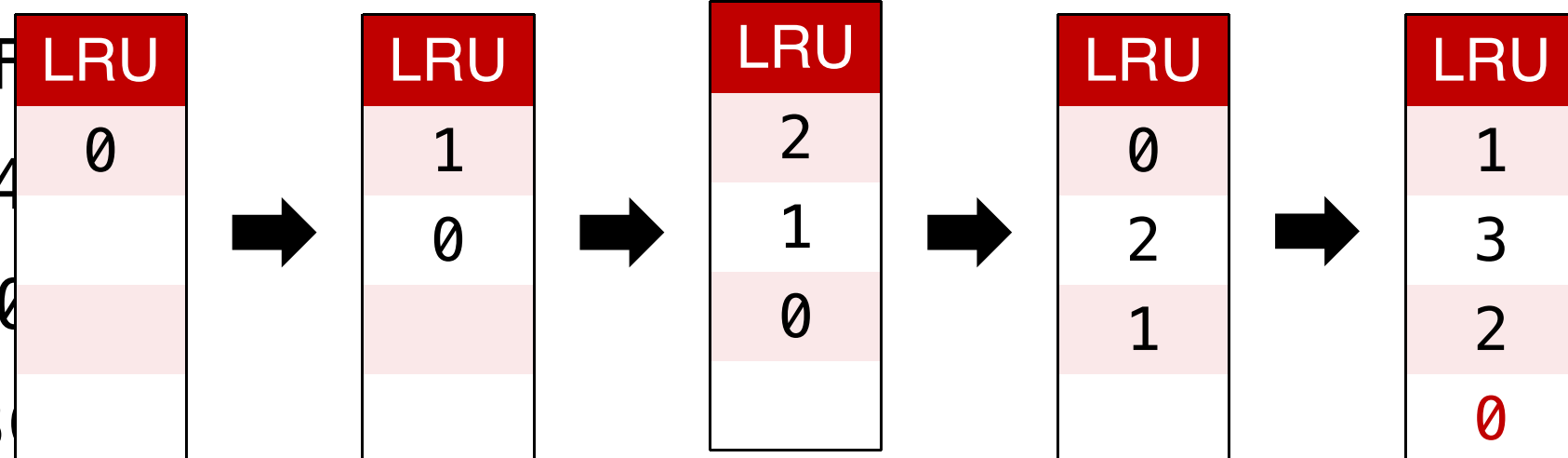


LRU Example (Cont'd)

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.

Tag	VB	LRU	DATA			
0x010F	1	1	12	34	25	87
0x048D	1	3	87	65	43	21
0x0008	1	2	12	34	56	78
0x03D0	1	0	FF	FF	FF	FF

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0008
- 4. Load word at 0x043D
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

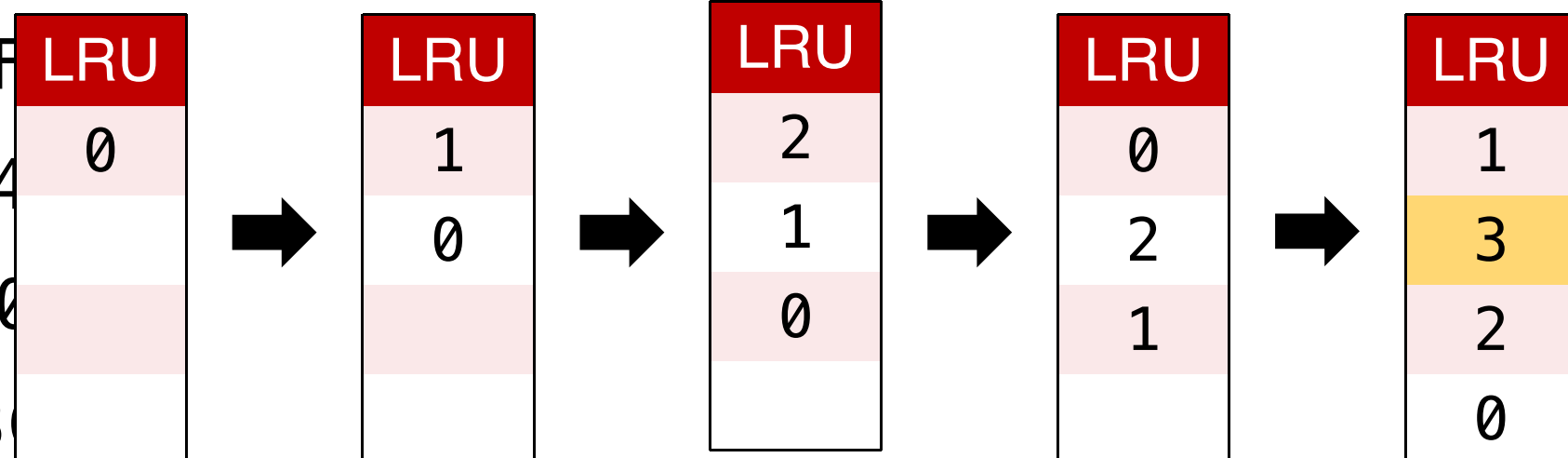


LRU Example (Cont'd)

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.

Tag	VB	LRU	DATA			
0x010F	1	1	12	34	25	87
0x048D	1	3	87	65	43	21
0x0008	1	2	12	34	56	78
0x03D0	1	0	FF	FF	FF	FF

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0000
- 4. Load word at 0x043F
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120



Question: What is in the tag/VB/LRU

- Add extra information to record cache usage;
 - We can use index to indicate its priority;
 - LRU = 0 indicates the highest priority (the most recently used);
 - LRU = 3 indicates the least recently used.
- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120

Tag	VB	LRU	DATA			
0x010F	1		12	34	25	87
0x0008	1		12	34	56	78
0x03D0	1		FF	FF	FF	FF

LRU

Question: What is in the tag/VB/LRU

- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120
- 7. Load halfword at 0x043E

Tag	VB	LRU	DATA			
0x010F	1	2	12	34	25	87
0x0048	1	0	so	me	wo	rd
0x0008	1	3	12	34	56	78
0x03D0	1	1	FF	FF	FF	FF

The Other Replacement Policies

- Least recently used (LRU)
 - Replace the entry that has not been used for the longest time, i.e., has the oldest previous access.
 - Pro: **Temporal locality!**
 - Recent past use implies likely future use
 - Con: Complicated hardware to keep track of access history

LRU is ideal for **temporal locality**.
In practice, FIFO/LIFO/Random are the most common.

- Most Recently Used (MRU)
 - Replace the entry that has the newest previous access.
- First In, First Out (FIFO)
 - Replace the oldest line in the set (queue).
- Last In, First Out (LIFO)
 - Replace the newest line in the set (stack).
- Random

Reasonable approximations to LRU, MRU without add too much hardware.

Works surprisingly okay (when given a low temporal locality workload)

What about Write?

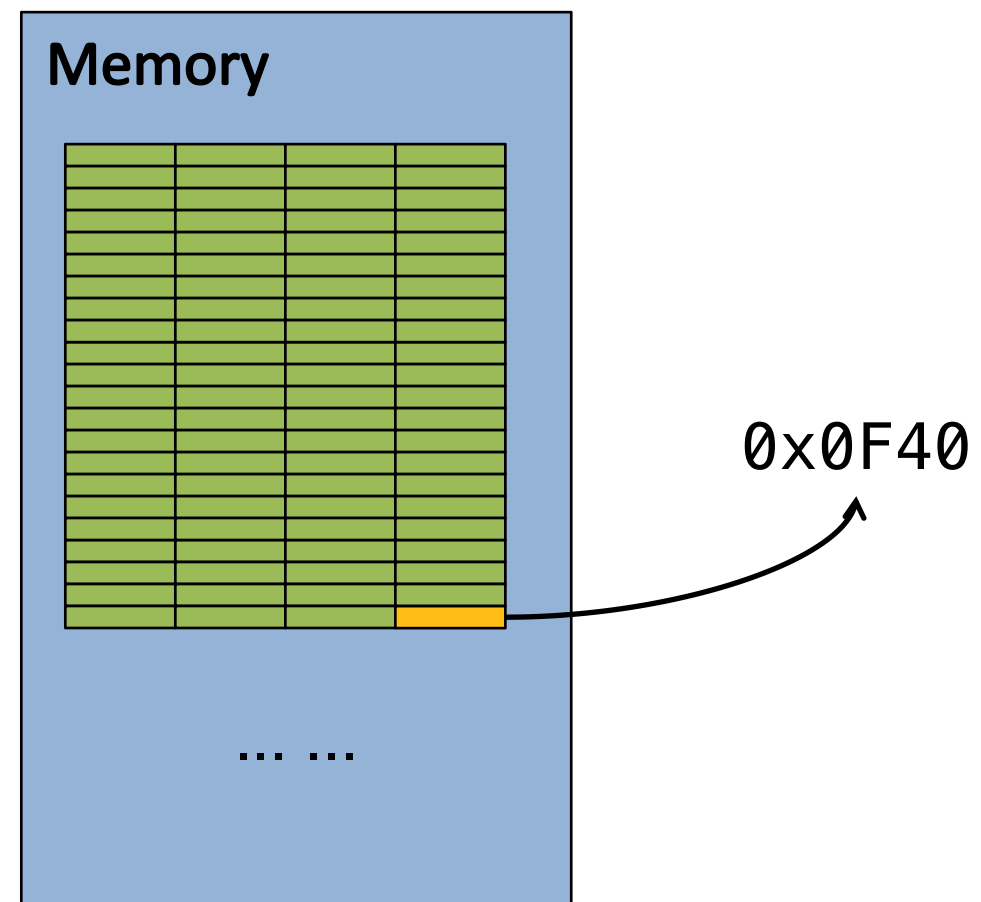
- Write Policies
- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120
- 7. Store byte 0x0 to 0x0F43

Tag	VB	LRU	DATA			
0x010F	1	2				
0x0048	1	0				
0x0008	1	3				
0x03D0	1	1				

What about Write?

- Write Policies
- 1. Load byte at 0x043F
- 2. Load byte at 0x1234
- 3. Load halfword at 0x0022
- 4. Load word at 0x043C
- 5. Load byte at 0x0F43
- 6. Load word at 0x0120
- 7. Store byte 0x0 to 0x0F40
 - Cache **HIT**!

Tag	VB	LRU	DATA			
0x010F	1	2				
0x0048	1	0				
0x0008	1	3				
0x03D0	1	1				



Write-back vs. Write-through

- **Store** instructions write to memory, which **changes values**.
- Hardware needs to ensure that cache and memory have **consistent** information.

Tag	VB	LRU	DATA			
0x010F	1	2				
0x0048	1	0				
0x0008	1	3				
0x03D0	1	1				

- Write-through
 - Write to the cache and memory at the same time.
 - (more writes to memory → longer time)
- Write-back (not the **write back phase** in pipeline)
 - Write data in cache and set a **dirty** bit to 1.
 - When this block gets replaced from the cache (and “back” to memory), write to memory.

Simple to implement

(typically) lower traffic to memory

Write-back in a LRU Fully Associative Cache

- Store byte 0x0 to 0x0F40
 - Cache **hit**!
- Write-back
 - Write data in cache and set a dirty bit to 1.
 - When this block gets evicted (replacement) from the cache (and “back” to memory), write to memory.

Tag	VB	LRU	Dirty	DATA			
0x010F	1	2	0				
0x0048	1	0	0				
0x0008	1	3	0				
0x03D0	1	1	1				00

Cache hit w/write-back:
 update cache line, and wait until this
 block is replaced before writing back
 to memory

Write-back in a LRU Fully Associative Cache

- Store byte 0x0 to 0x0F40
 - Cache **hit**!
- Write-back
 - Write data in cache and set a dirty bit to 1.
 - When this block gets evicted (replacement) from the cache (and “back” to memory), write to memory.
 - Update LRU

Tag	VB	LRU	Dirty	DATA			
0x010F	1	2	0				
0x0048	1	1	0				
0x0008	1	3	0				
0x03D0	1	0	1				00

Cache hit w/write-back:
 update cache line, and wait until this
 block is replaced before writing back
 to memory

Write with Cache Miss—Write-Allocate

- 7. Store byte 0x0 to 0xFF40
 - Cache **miss**!
- Write-allocate
 - Allocate in cache a space to deal with this write (cache block replacement)
 - Update LRU
 - Set dirty bit and implement write-back policy; or write-through

Tag	VB	LRU	Dirty	DATA			
0x010F	1	2	0				
0x0048	1	1	0				
0x0008	1	3	0				
0x03D0	1	0	0				



Tag	VB	LRU	Dirty	DATA			
0x010F	1	2	0				
0x0048	1	3	0				
0x3FD0	1	0	1				0
0x03D0	1	1	0				

No-Write-Allocate (Write Around)

- 7. Store byte 0xFF40, 0x0
 - Cache **miss**!
- Non-write-allocate
 - The data is directly write to the main memory without loading it into the cache;
- Advantage: avoid evicting data that may be used later;
- Disadvantages: access the main memory each time, may increase the latency;

Tag	VB	LRU	Dirty	DATA			
0x010F	1	2	0				
0x0048	1	1	0				
0x0008	1	3	0				
0x03D0	1	0	0				

Summary

- Fully associative cache
 - Data can go to each cache block (placement policy);
 - Address is divided and used as tag and offset;
 - Cache contains tag, valid bit, LRU, Dirty bit (if applicable), data;
 - Address is compared to the tags in parallel to decide cache miss/hit;
 - Cache block replacement policies;
 - Write policies.