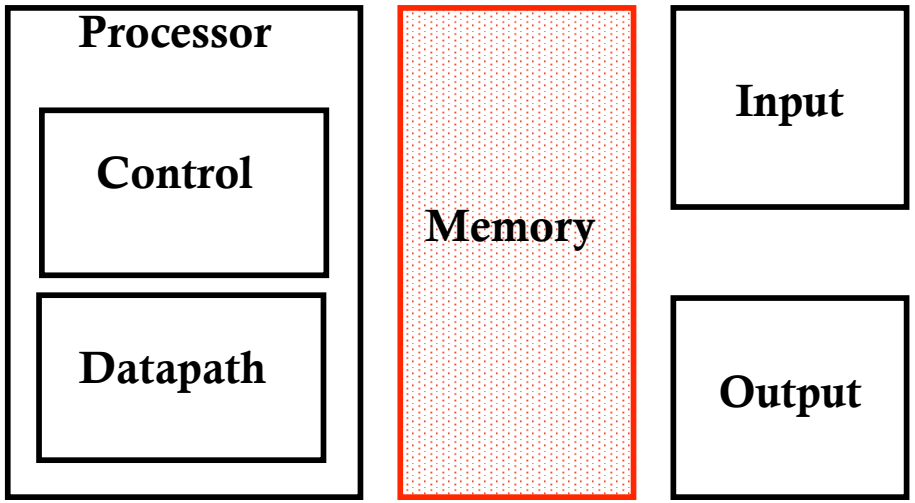
CS465

Lecture 9: Large and Fast: Exploiting Memory Hierarchy

*Slides adapted from Computer Organization and Design by Patterson and Henessey

Big Picture: Where are We Now?

- The five classic components of a computer



- Topics:
  - Locality and memory hierarchy
  - Simple caching techniques
  - Many ways to improve cache performance

Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, $10 – $20 per GB
- Flash
  - 5,000ns – 50,000ns, $0.75 - $1.00 per GB
- Magnetic disk
  - 5ms – 20ms, $0.05 – $0.10 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

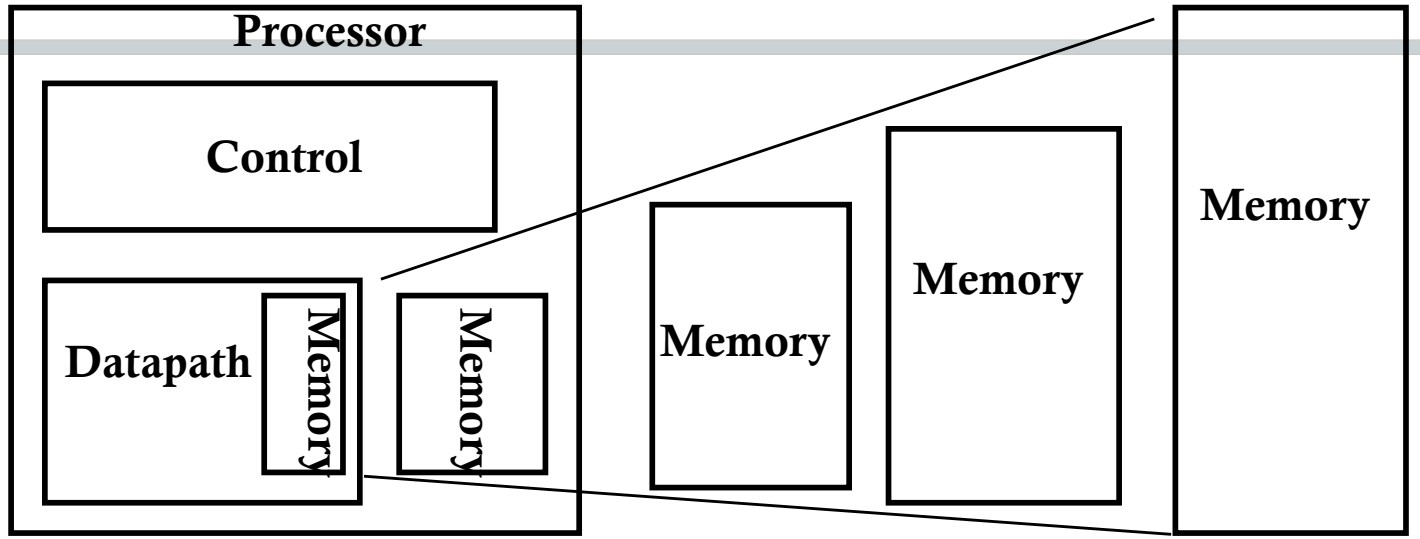
Typical numbers for 2012

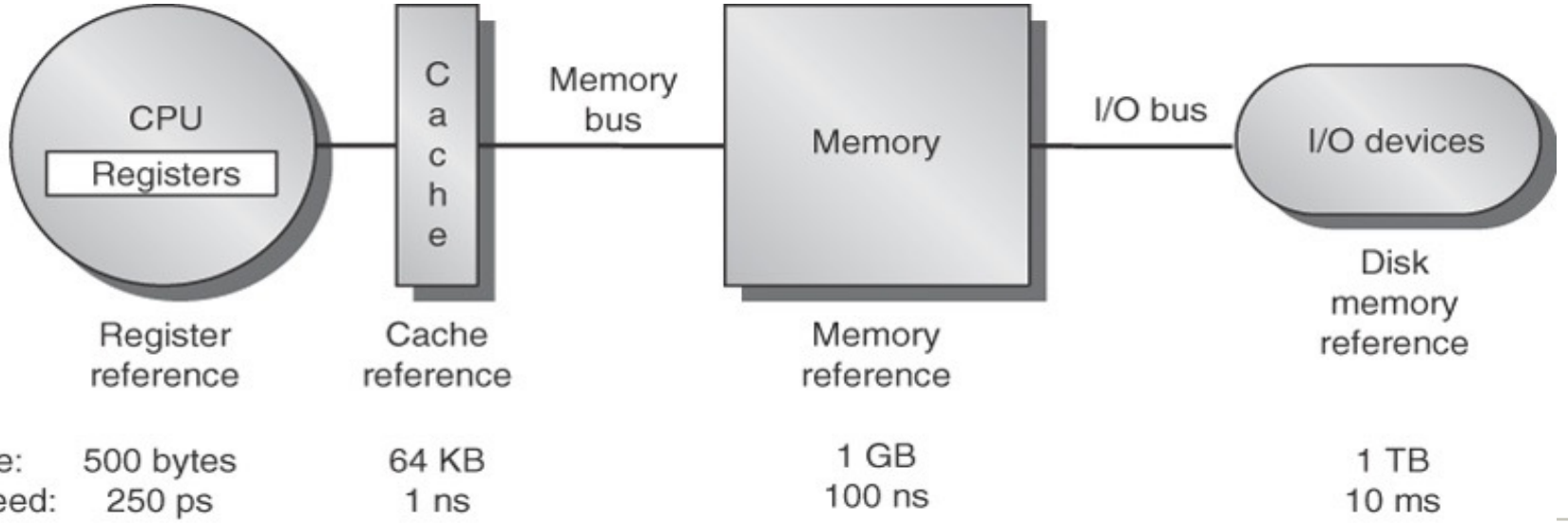
Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, $500 – $1000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, ~~$10 – $20 per GB~~ $3-$6 perGB
- Flash
  - 5,000ns – 50,000ns, ~~$0.75 - $1.00 per GB~~ $0.06 - $0.12 per GB
- Magnetic disk
  - 5ms – 20ms, ~~$0.05 – $0.10 per GB~~ $0.01 – $0.02 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

Typical numbers for 2020

Memory Hierarchy (1/2)



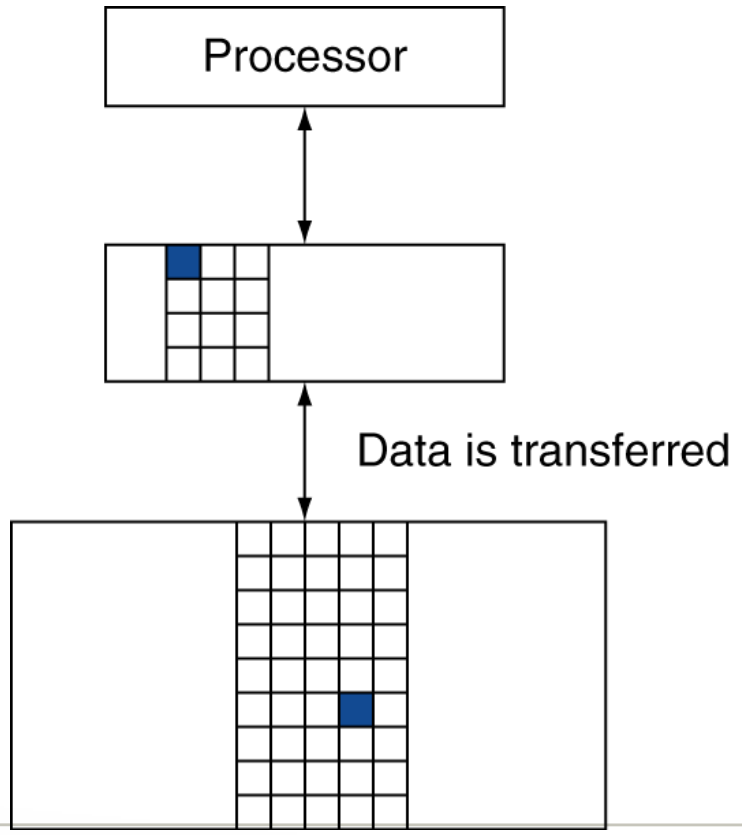


Memory Hierarchy (2/2)

- If level closer to processor, it will be:
  - Faster
  - Smaller
  - More expensive
- Lowest level (usually disk) contains all data and instructions
  - Higher levels have a subset of lower levels (contains most recently used data)
- Goal: illusion of large, fast, cheap memory

Memory Hierarchy Levels

- Disk contains everything while cache only has a subset
  - When processor needs something, first search the highest level



  - If search succeeds (hit), access it w/ the least time
  - If search fails (miss), bring it from the lower levels of memory
    - Copy data into the higher level
    - Unit of copying (block) often has multiple words / bytes
  - Future accesses to the same data are likely to hit in the higher levels

Entire idea is based on the concept of locality

Principle of Locality

- Programs access a small segment of their address space at any time; observations:
  - If we use it now, we will want to use it again soon
  - If we use it now, we will use those nearby soon
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

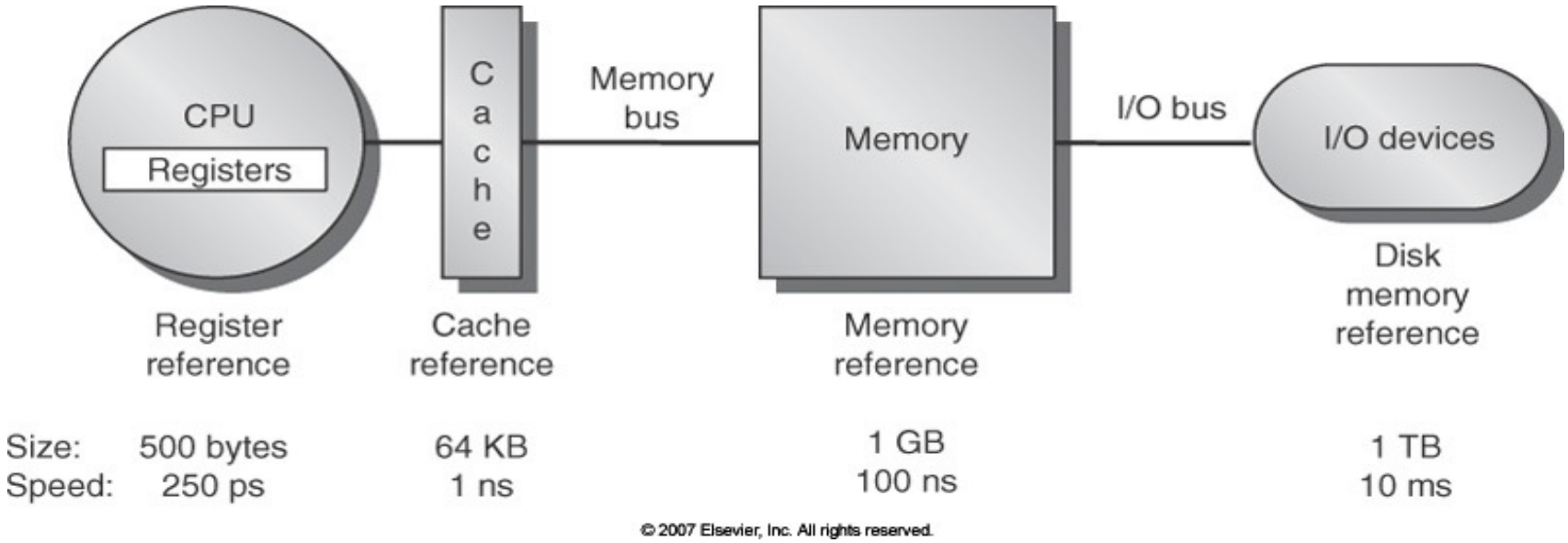
Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level
  - Hit rate: the fraction of memory access found in the upper level = hits/accesses
  - Hit time: time to access the upper level
    - RAM access time + time to determine hit/miss
- Miss: data needs to be retrieved from a block in the lower level
  - Miss rate = 1 - (hit rate) = misses/accesses
  - Miss penalty: time to fetch a block into a level of memory hierarchy from the lower level
  - Access time on a miss = hit time + miss penalty
- Hit time << miss penalty

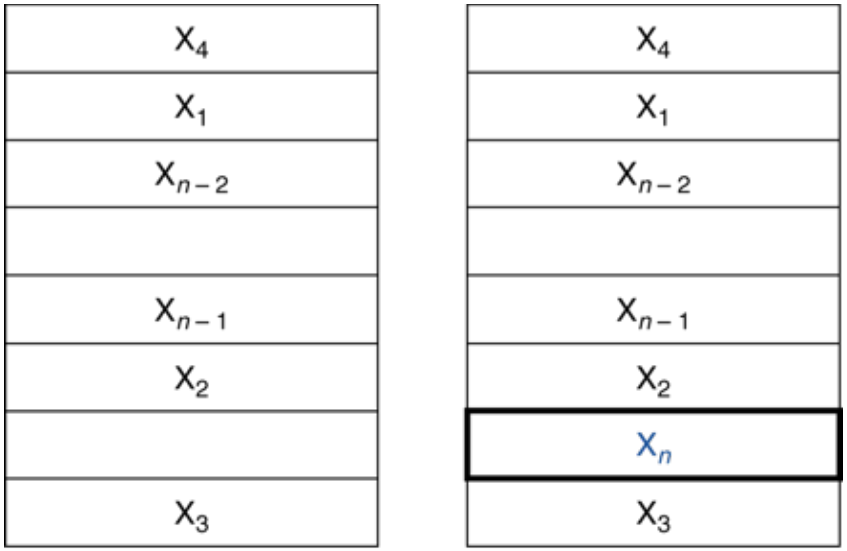
RoadMap



- Cache as a faster buffer of main memory
  - Organization options, evaluation, improvements
- Main memory as a faster buffer of disk

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$

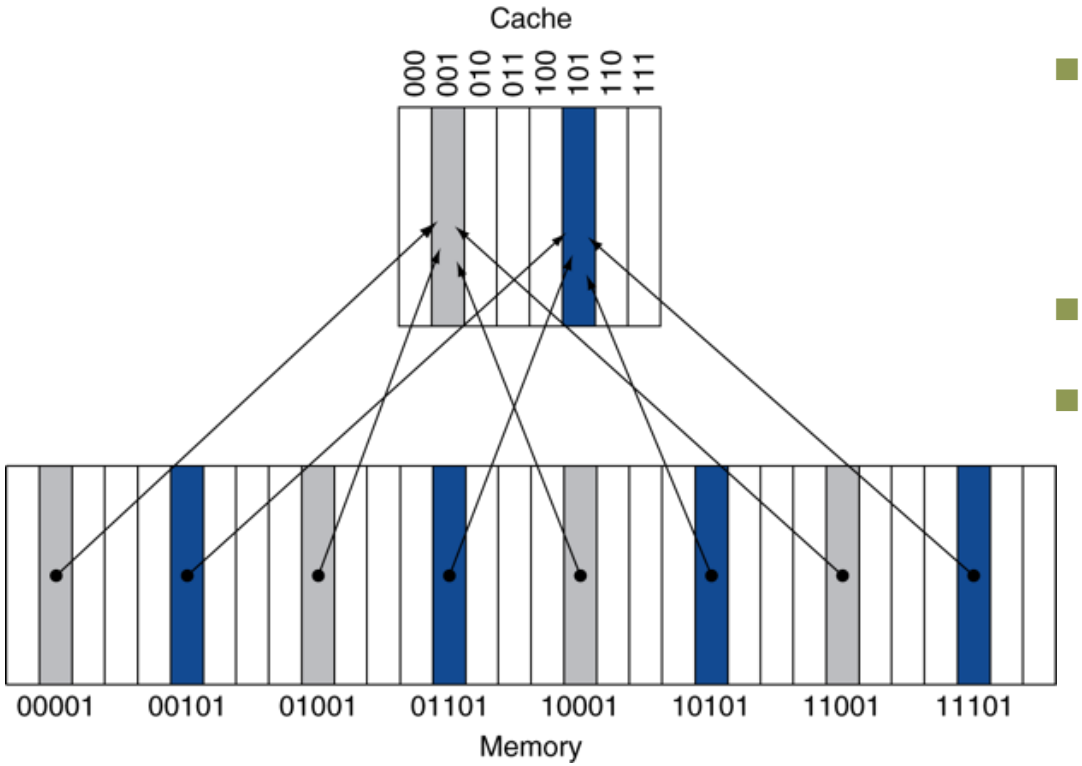


a. Before the reference to X_n b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?
- How does a memory address map into a location in cache?

Direct Mapped Cache

- Each memory address is mapped to one block in cache
  - Location = (Block address) modulo (#Blocks in cache)



- Block is the unit of transfer between cache and memory
- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- Note that multiple memory addresses map to same cache index
- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
  - Index = addr mod 8
- Initial state (empty cache)

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Hit/miss	Cache block
22 (10 110)	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Hit/miss	Cache block
22 (10 110)	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Hit/miss	Cache block
26(11 010)	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Hit/miss	Cache block
22(10 110)	Hit	110
26(11 010)	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Hit/miss	Cache block
16(10 000)	Miss	000
3(00 011)	Miss	011
16(10 000)	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Hit/miss	Cache block
18(10 010)	Miss	010

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Hit/miss	Cache block
18(10 010)	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Access Operations

- Three scenarios for cache reading:
  - Cache hit:
    cache block is valid and contains proper address, so read desired word
  - Cache miss (invalid block):
    nothing in cache at the appropriate block, so fetch from memory
  - Cache miss (valid block but tag mismatch), block replacement:
    wrong data is in cache at the appropriate block, so discard it and fetch desired data from memory

Cache Example

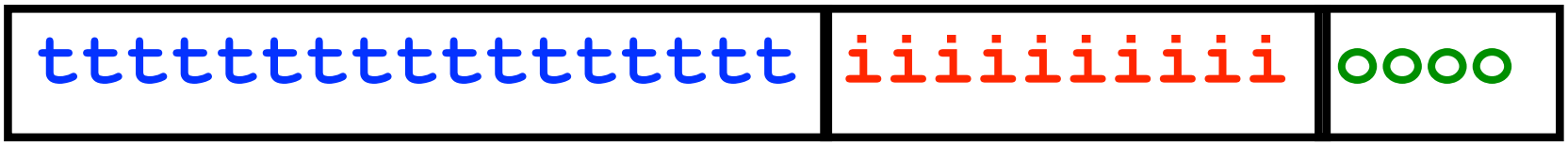
Word addr	Hit/miss	Cache block
18(10 010)	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

- Cache has 8-blocks, 1 word/block, load-word requests by processor

Larger Block Size

- What if we have a block size > 1 word?
  - Need to specify which word to access within the block
  - Similar situations for byte accessing and block of multiple bytes
- Byte address: usually need to divide into three fields



tag	index	byte
to check	to	offset
if have	select	within
correct block	block	block

Directed Mapped Cache Address Subdivision

- All fields are read as unsigned integers
- Index: specifies the cache index (which "row" of the cache we should look in)

  →no. of blocks in cache determines the no. index bits

- Offset: specifies which byte within the mapped block we want to access

  →no. of bytes in one block determines the no. offset bits

- Tag: the remaining bits; used to distinguish between all the memory addresses that map to the same location

  →all remaining bits go to tag

Exercise

- For a direct-mapped cache size of 16KB (1KB = 2^10 byte) and 4-word per block
- Determine the length of the tag, index and offset fields assuming a 32-bit byte address ?

Cache Access Example

- Direct-mapped cache, 16KB of data, 4-word blocks
- Read(lw) 4 byte addresses
  - 0x00000014
  - 0x0000001C
  - 0x00000034
  - 0x00008014
- Memory values on right:
  - Only cache/memory level of hierarchy

Cache Access Example

- Direct-mapped cache, 16KB of data, 4-word blocks
  - 32-bit byte address division: 4-bit offset, 10-bit index, 18-bit tag
  - Calculation from previous example
- 4 addresses divided into Tag, Index, Byte Offset fields

```
00000000000000000 0000000001 0100   0x00000014
00000000000000000 0000000001 1100   0x0000001C
00000000000000000 0000000011 0100   0x00000034
00000000000000010 0000000001 0100   0x00008014
      Tag               Index    Offset
```

Cache Access Example

- Valid bit: determines whether anything is stored in that row (when computer initially turned on, all entries invalid)

Index	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

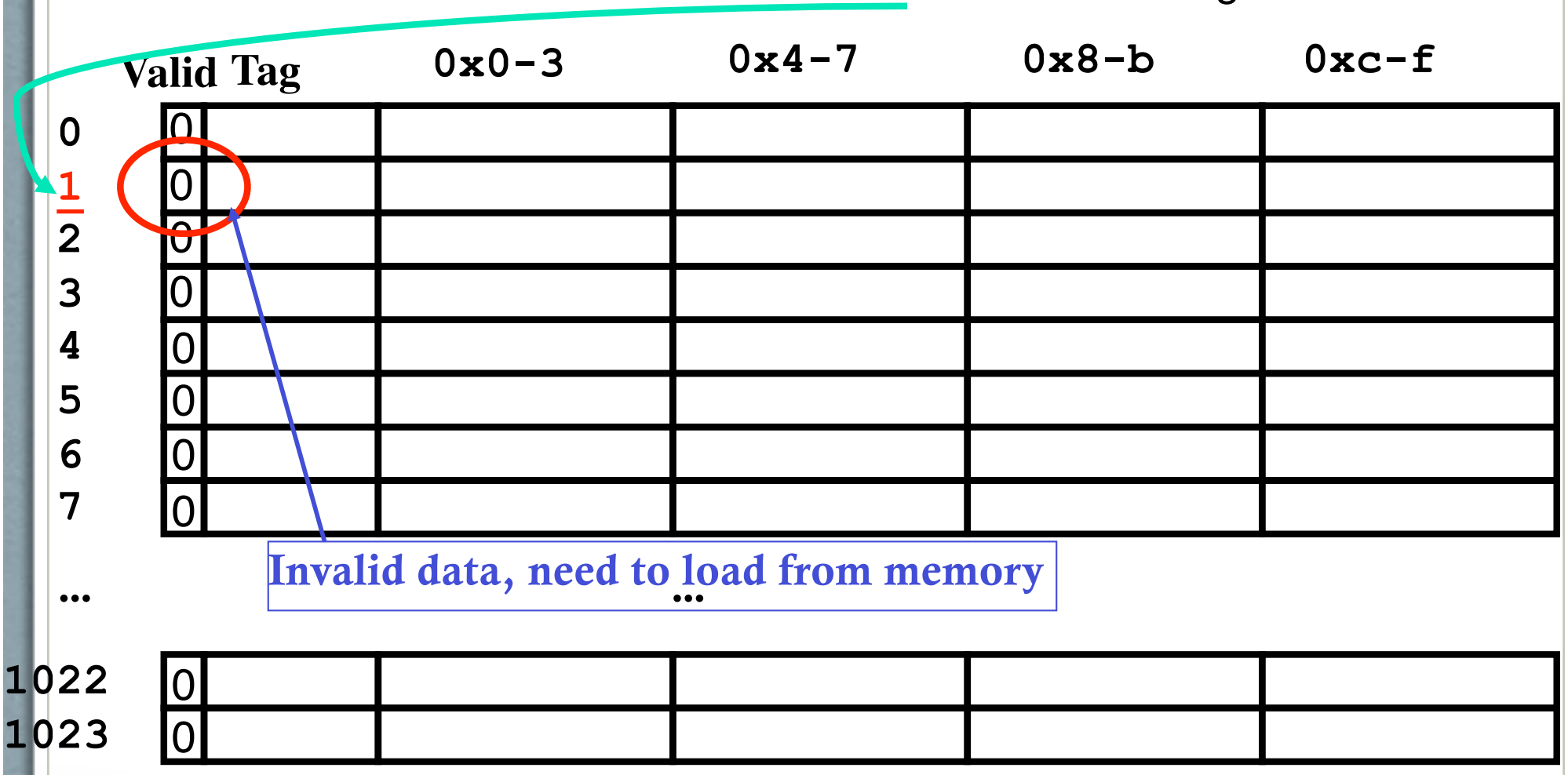
1. lw 0x00000014

- 00000000000000000 0000000001 0100 Tag / Index / Offset

	Valid	Tag	0x0-3	0x4-7	0x8-b	0xc-f
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
…				…		
1022	0					
1023	0					

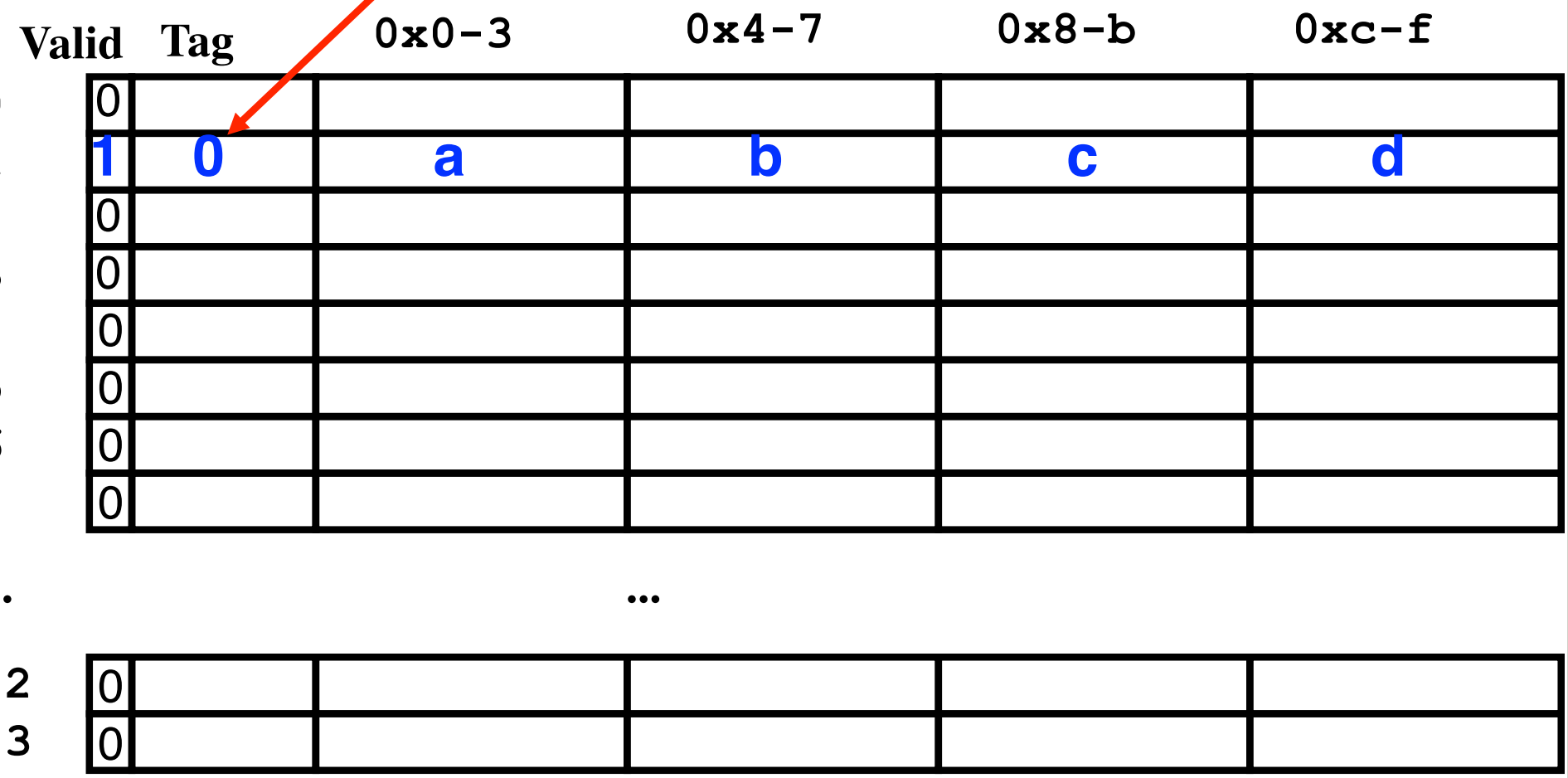
1. lw 0x00000014

- 00000000000000000 0000000001 0100 Tag / Index / Offset



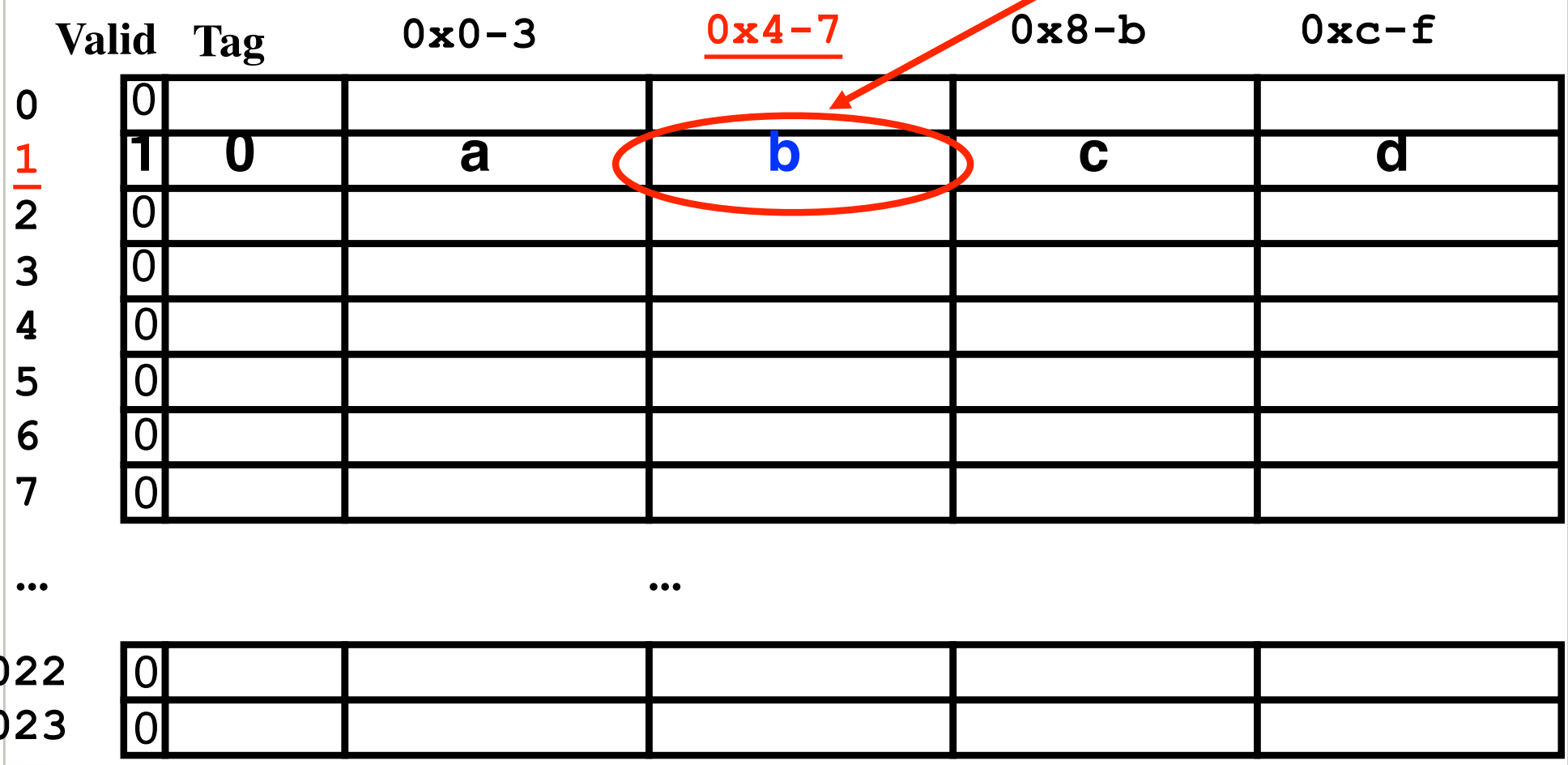
Loading; Setting Tag/Valid Bit

- 0000000000000000 0000000001 0100 Tag / Index / Offset



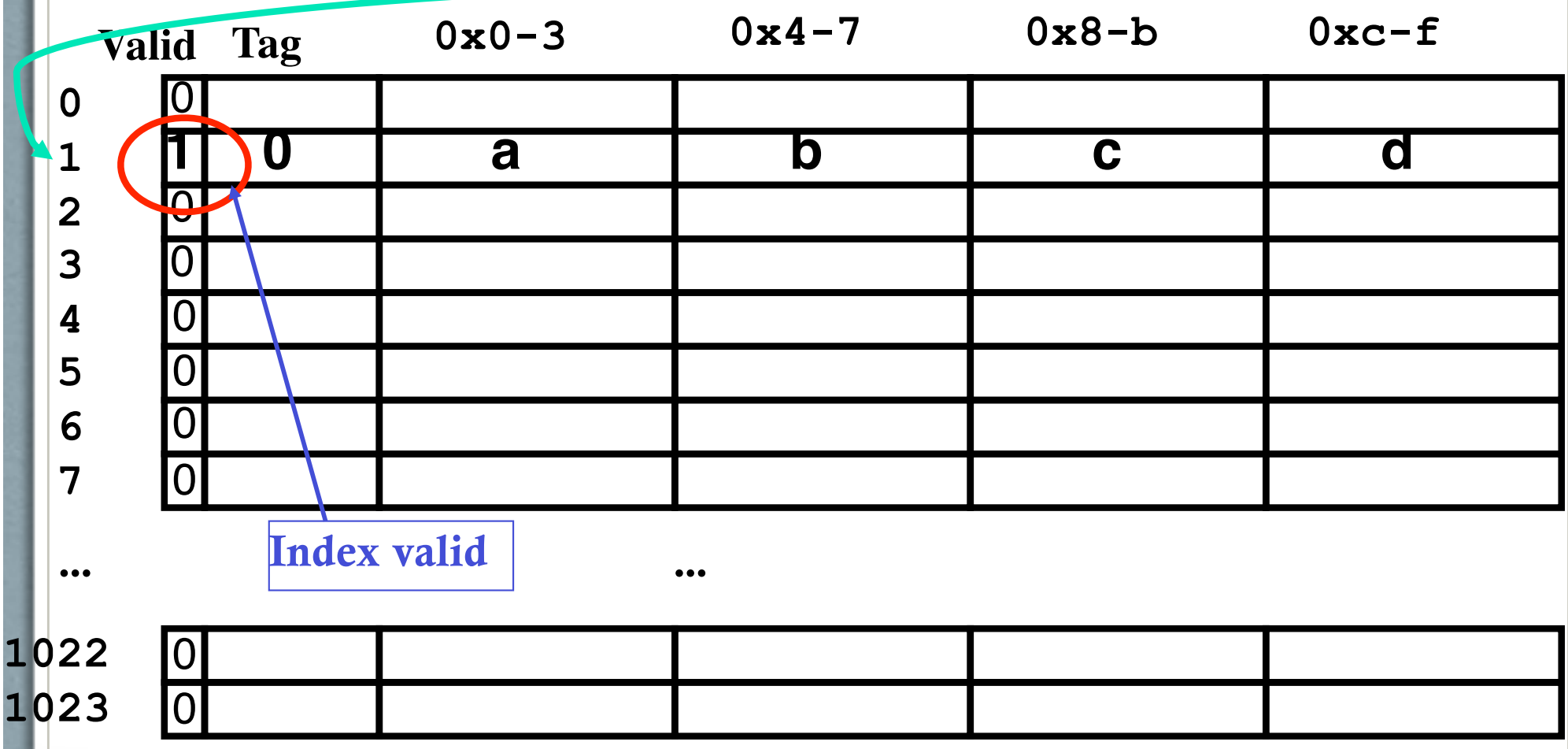
Read from Cache at Offset

- 00000000000000000 000000001 0100 Tag / Index / Offset



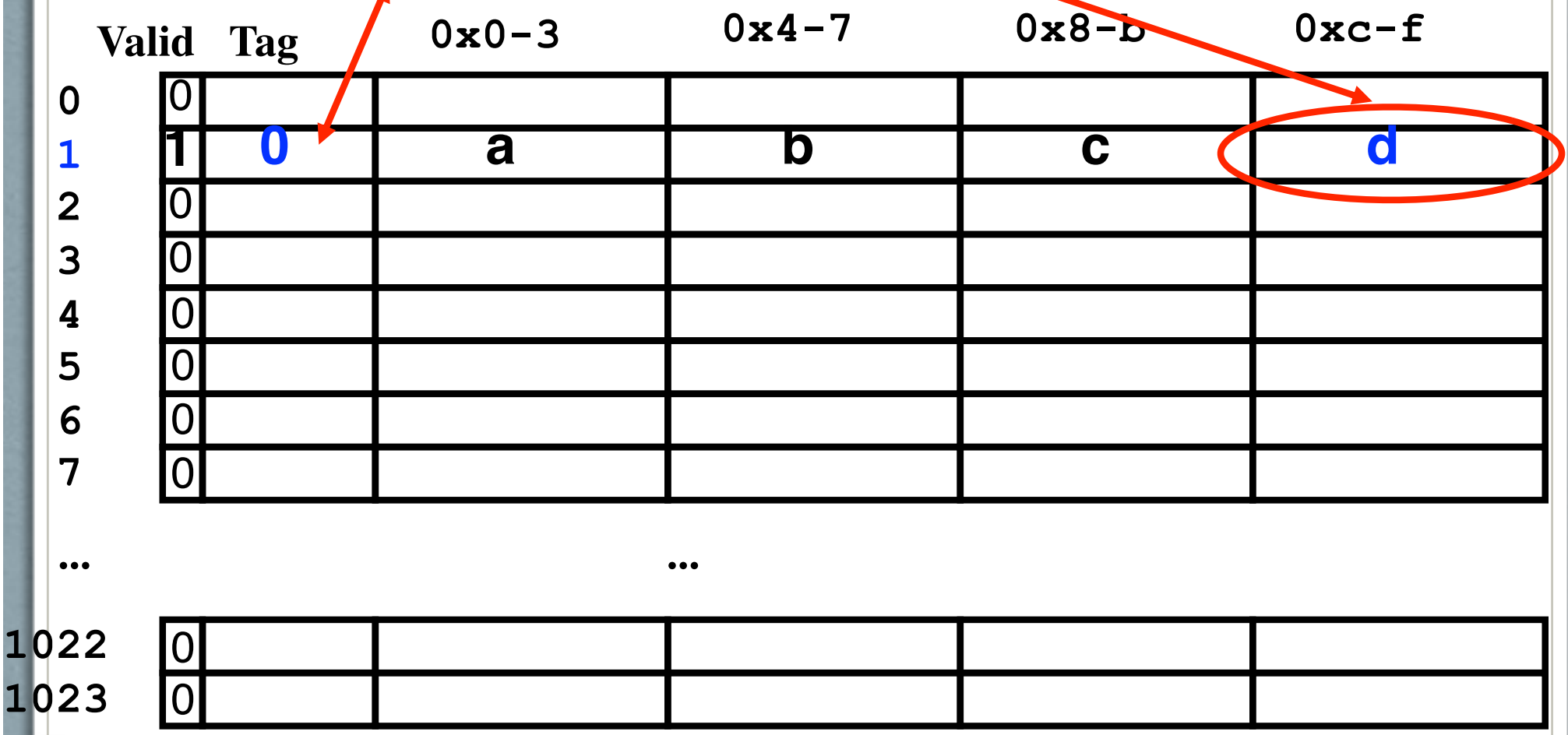
2. lw 0x0000001C

- 00000000000000000 0000000001 1100



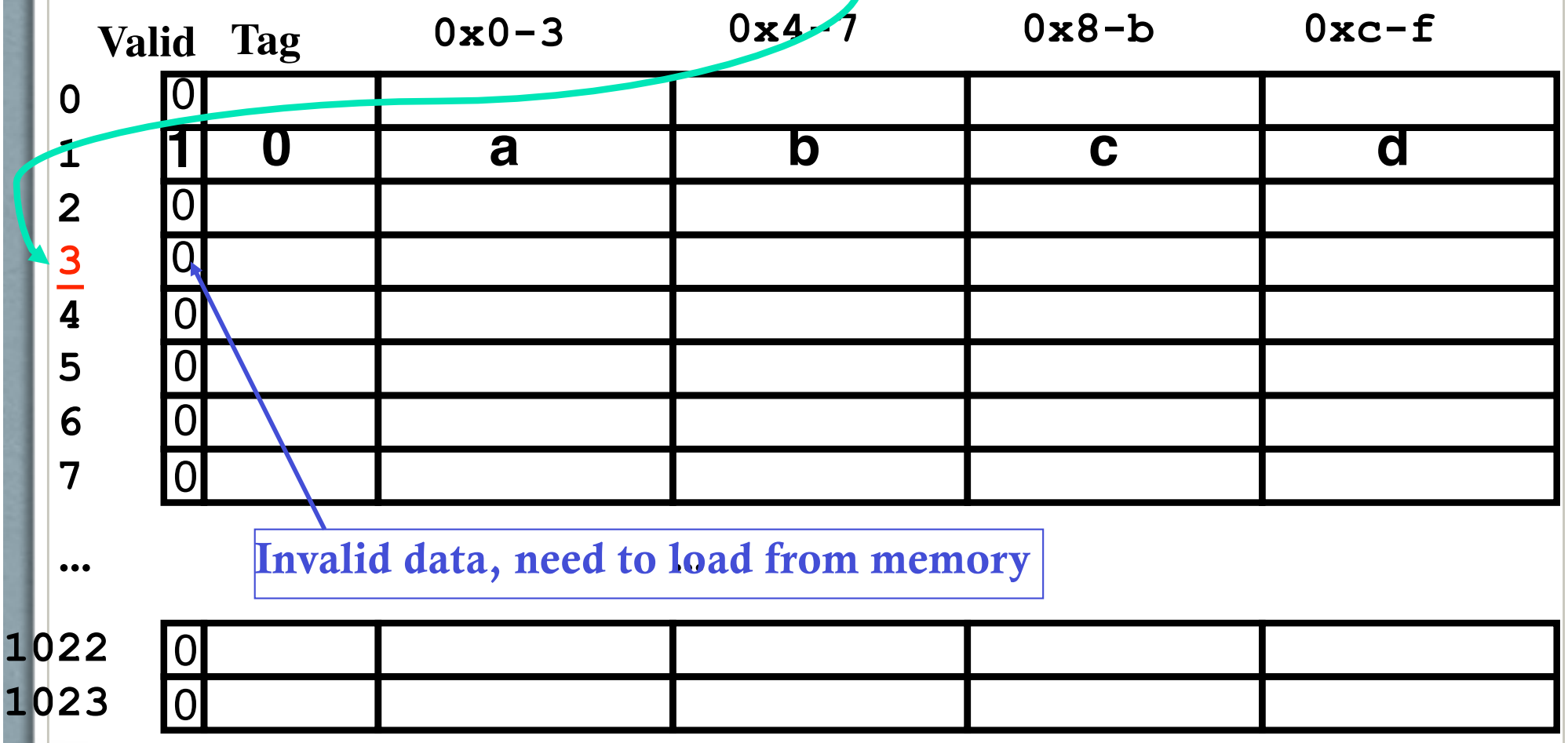
Index Valid, Tag Match, Return d

- 00000000000000000 000000001 1100



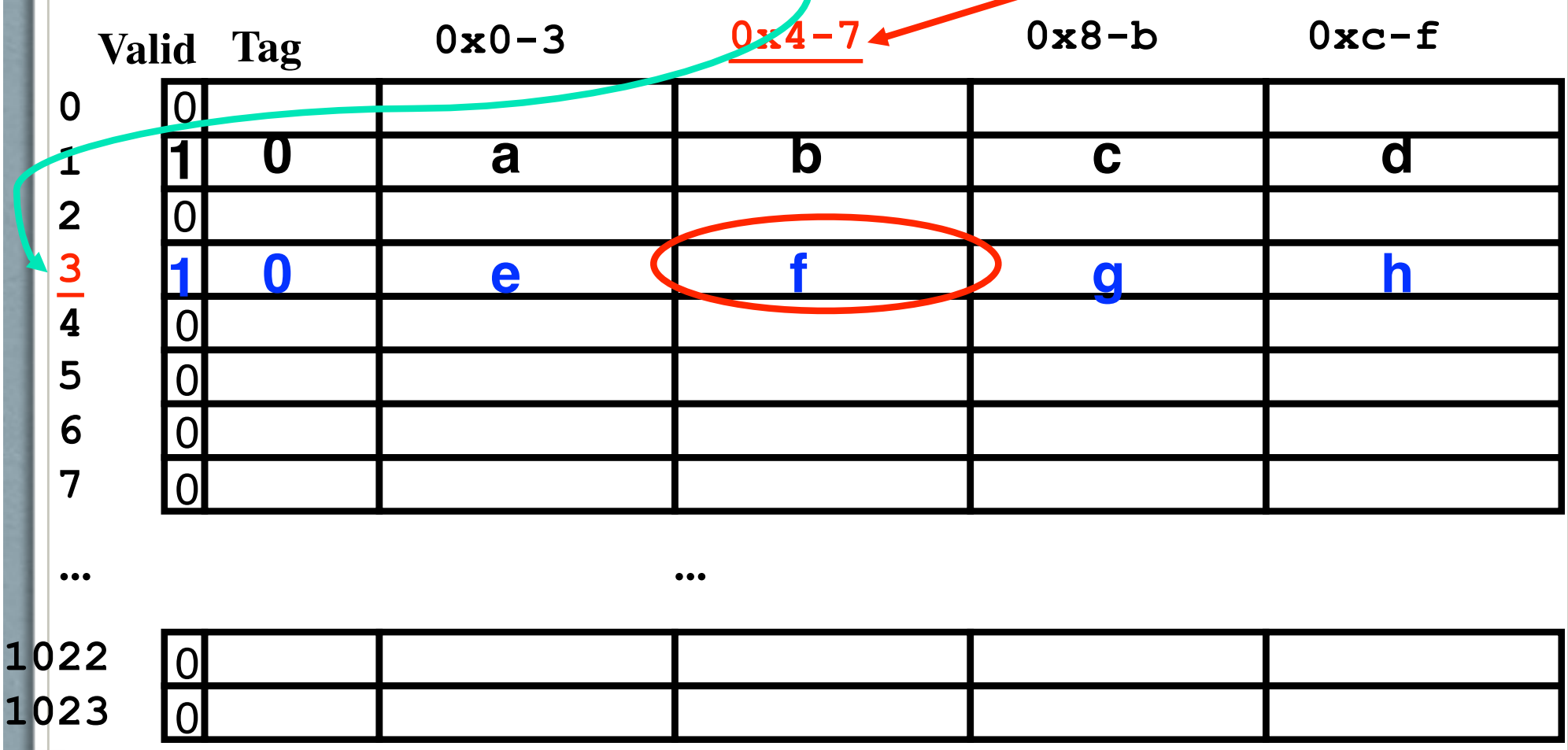
3. lw 0x00000034

- 00000000000000000 000000011 0100



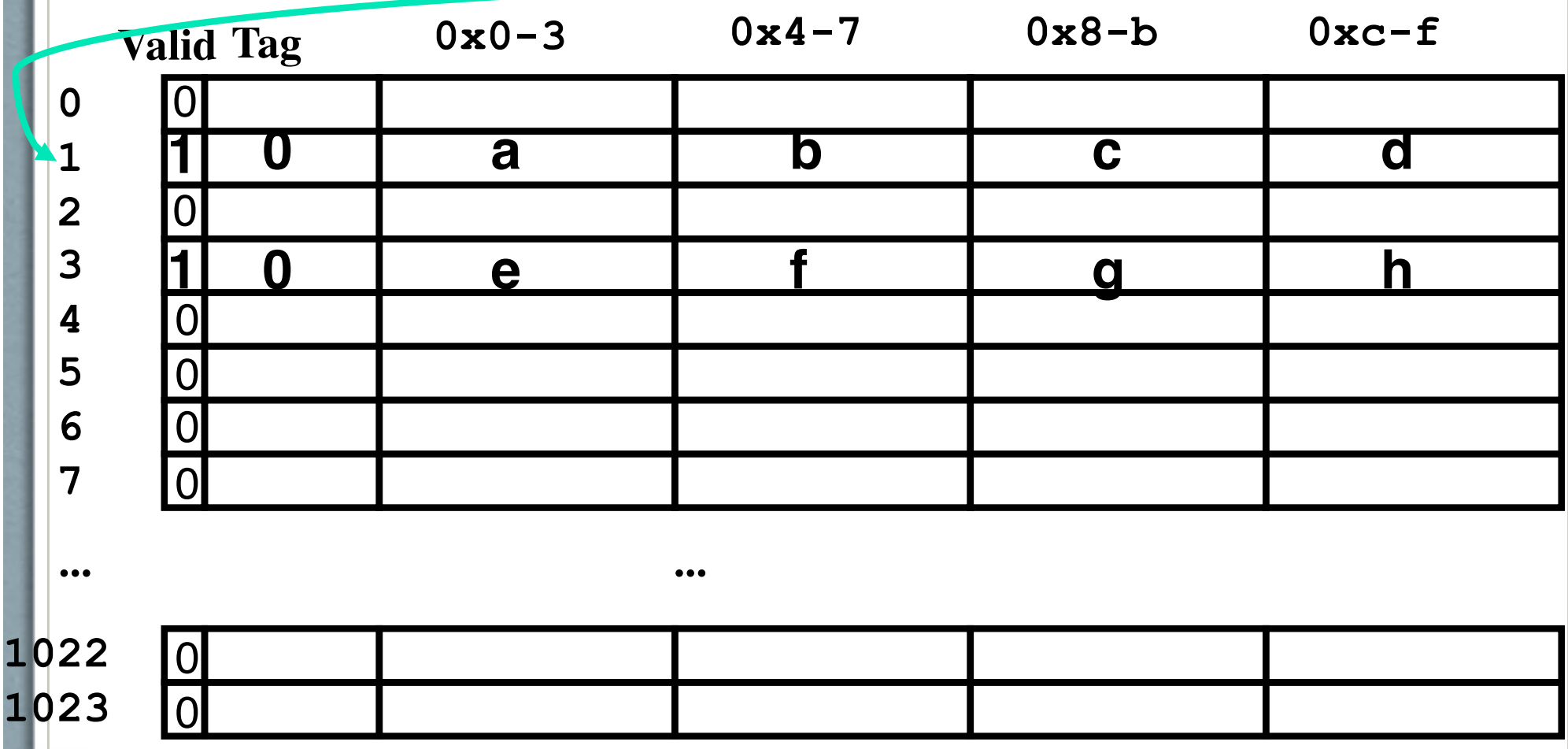
Load Cache block; Return word f

- 00000000000000000 0000000011 0100



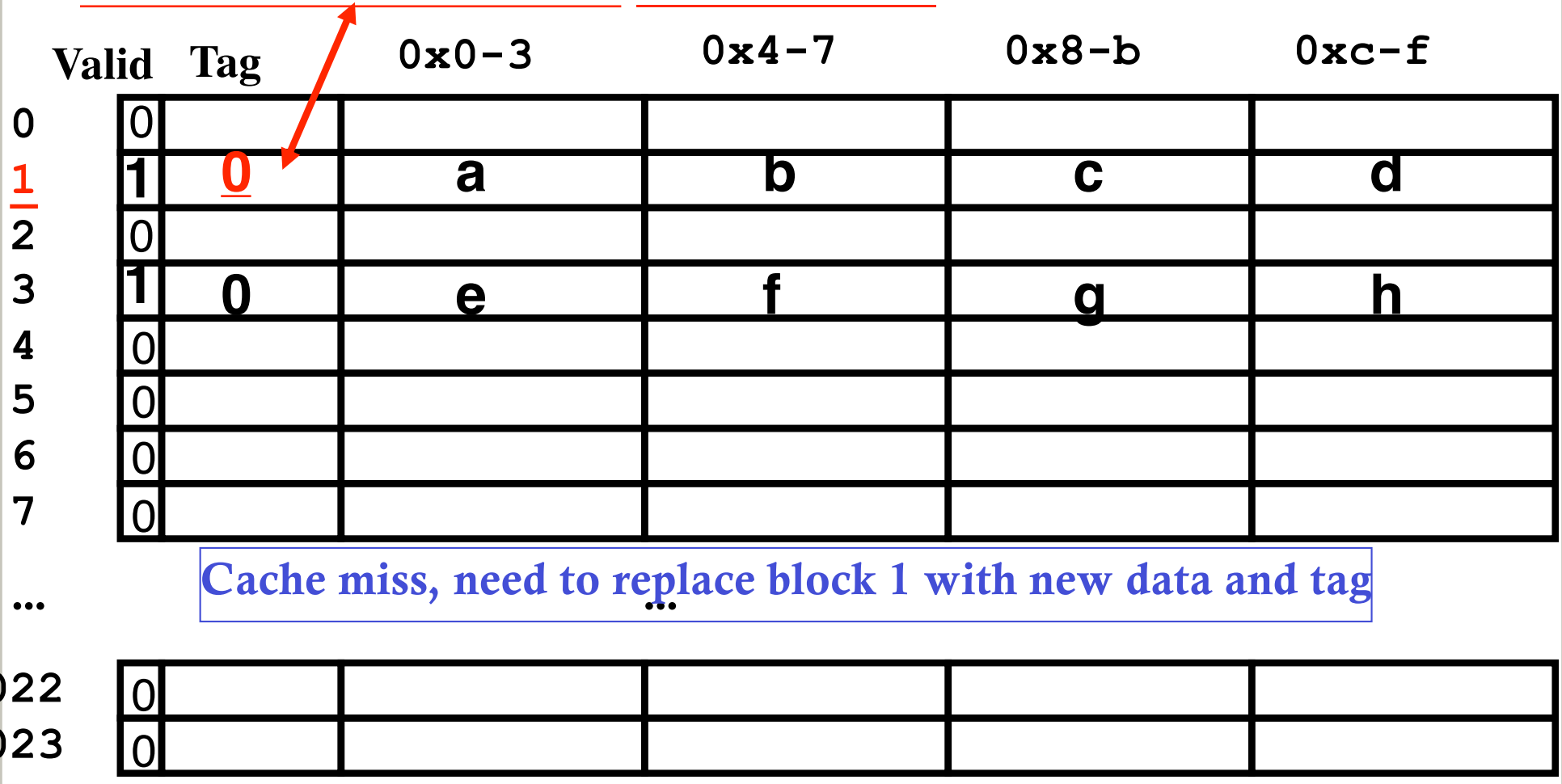
4. lw 0x00008014

- 00000000000000010 000000001 0100



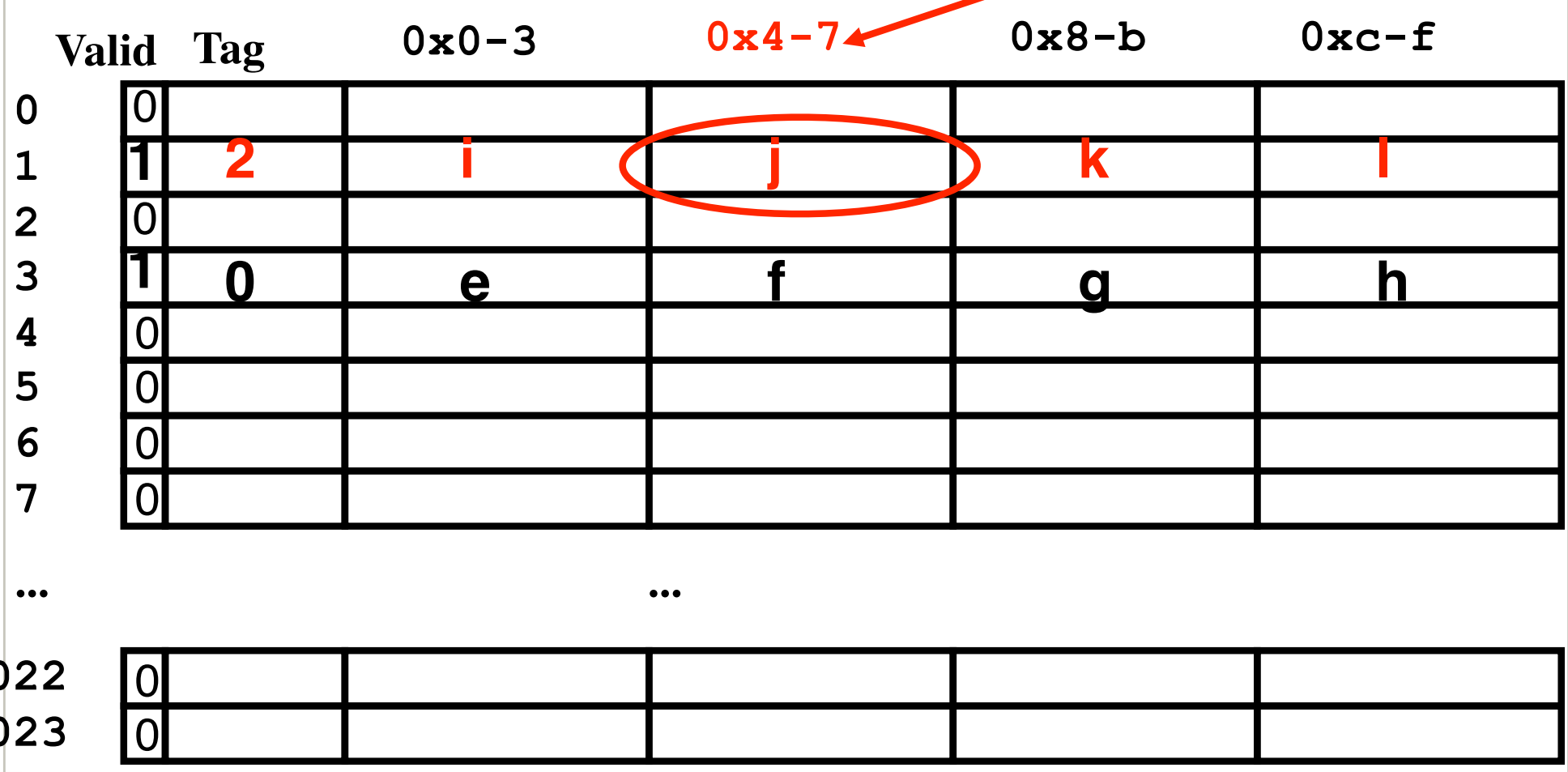
Tag Not Match

- 00000000000000010 0000000001 0100



After Replacement: return word j

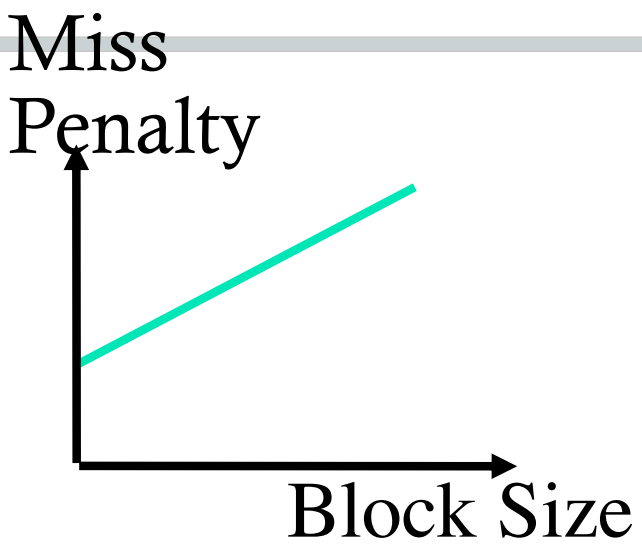
- 00000000000000010 0000000001 0100

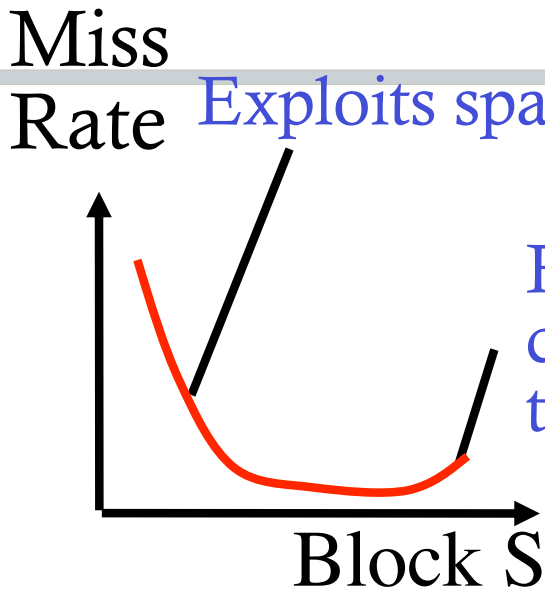


Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks $\Rightarrow$ fewer of them
    - More competition $\Rightarrow$ increased miss rate
- Larger miss penalty
  - Can override benefit of reduced miss rate

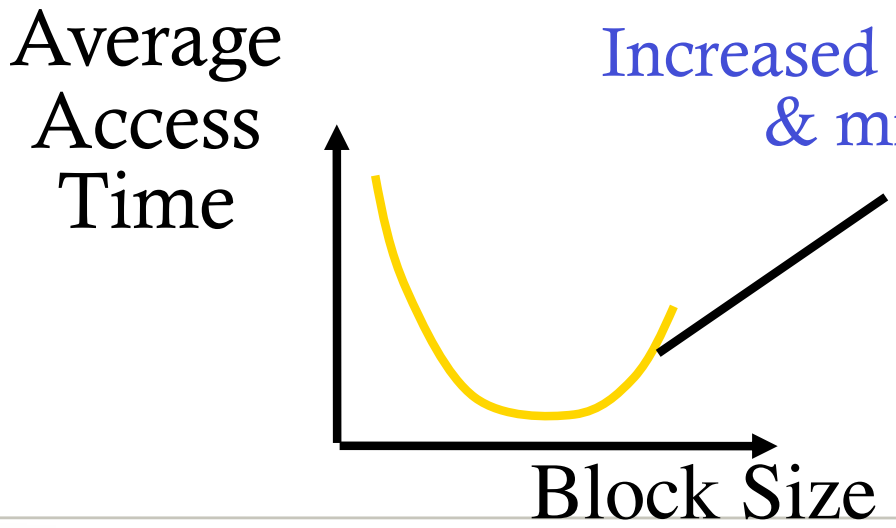
Block Size Tradeoff





Exploits spatial locality

Fewer blocks:
compromises
temporal locality



Increased miss penalty
& miss rate