15-213 Recitation
Caches & C Review

Your TAs

Friday, September 26th

Reminders

- **`attacklab`** was due *yesterday*.
- **`cachelab`** was released yesterday, and is due ***Thursday, October 9th***.
- Part 1 of the Midterm (take-home) is coming out October 1st, and is due ***October 8th***

Agenda

- Intro to `cachelab`
- Review: Cache Concepts
- Review: Programming in C
- Parsing Command-Line Arguments with `getopt()`
- Cache Practice Problems

cachelab

`cachelab`: Overview

- First project-based assignment:
  - You'll write a *cache simulator* in C from scratch!
- Take in parameters defining the cache structure (**`s`**, **`E`**, **`b`**).
- Read a "trace file" of memory accesses and simulate them.
- After simulating those accesses, return the number of hits, misses, evictions, etc.

Using Git

- This is the first lab where we will use Git!
  - Note you will download the assignment through Github
- Try to keep good version control practice!
  - Commit after every “good fix”!
  - Good version control can help you recover from “bad mistakes”!

Review: Cache Concepts

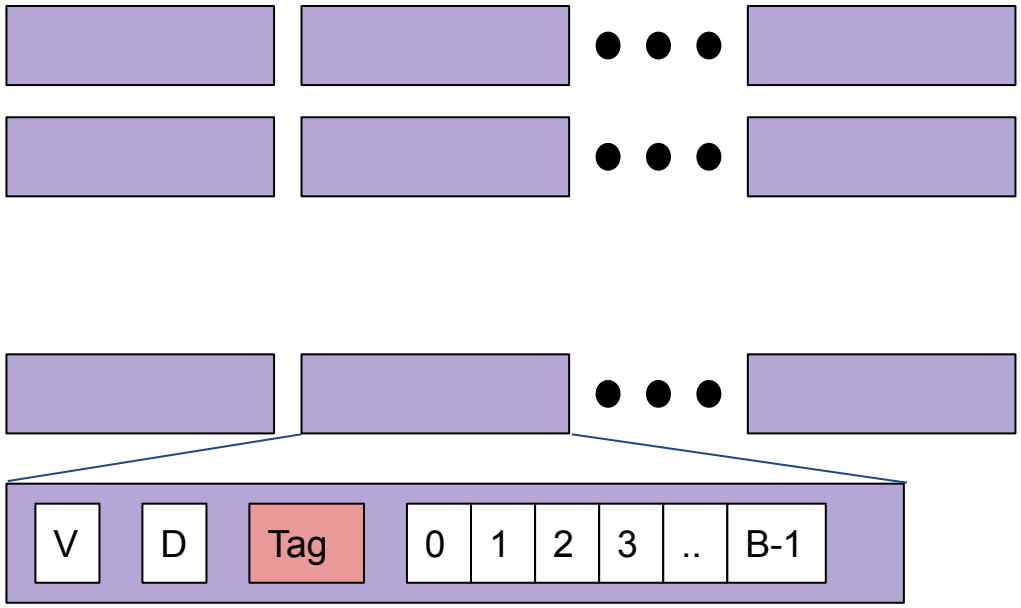
Cache Concepts: Configurations

- Your cache simulators need to support *parameters* (**`s`**, **`E`**, **`b`**) that allow the user to configure the layout of the cache.
- But what do these parameters mean?
- Let's review how a cache is organized!

Cache Organization

Note: don't need to store data for **`cachelab`**.

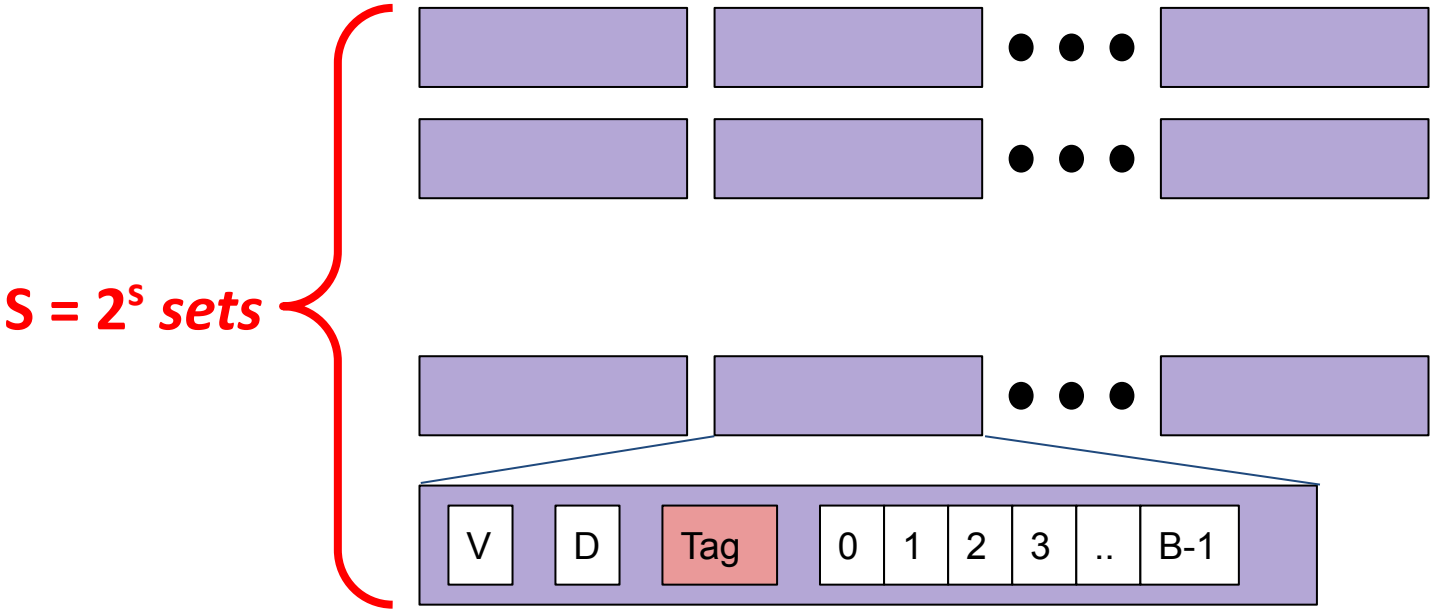
- A cache is composed of *sets*
- Each set is composed of some number of *lines*
- Each line stores the cached data itself, as well as information used by the cache.



Cache Organization

- A cache is composed of *sets*

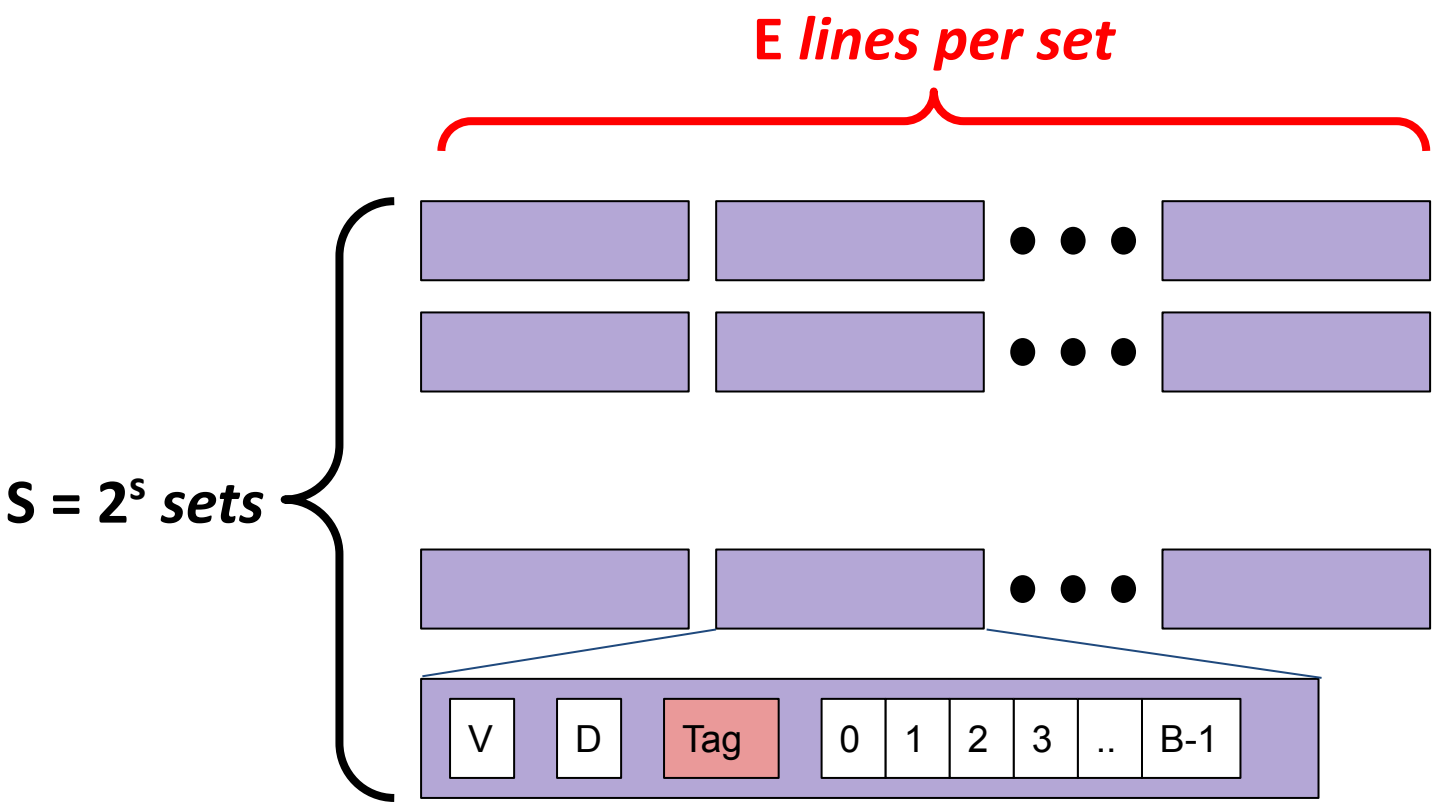
s – Number of set *bits*
$S = 2^s$ – Number of *sets*



Cache Organization

- Each set is composed of *lines*

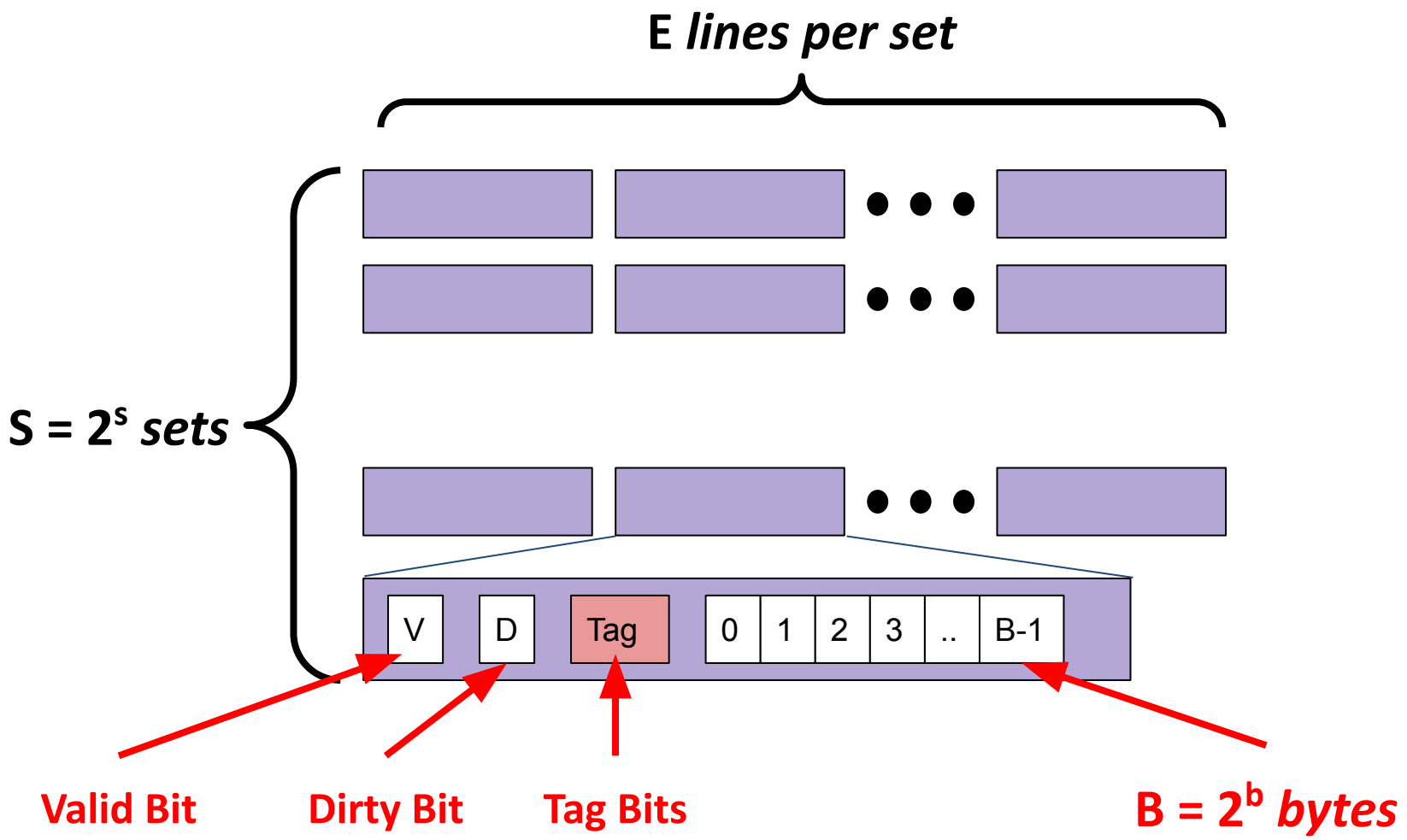
E – Number of *lines per set*



Cache Organization

- Each line stores data

b – Number of block offset *bits*
B = 2^b – Block Size



Cache Concepts: Cache Read

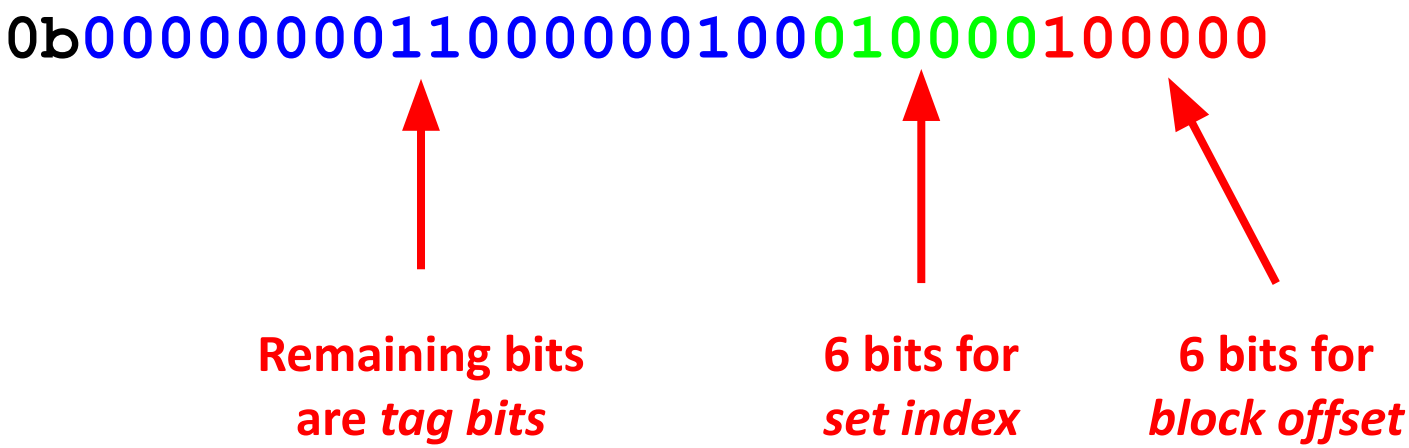
- We have an address that we want to look up in our cache.

`0x00604420`

- How do we search for it? Which set? Which line?
- Our *parameters* (`s` and `b`) determine how we partition the bits of our address.

Cache Concepts: Cache Read

- Our *parameters* (**s** and **b**) determine how we partition the bits of our address.
- Suppose **s** = 6 and **b** = 6



Cache Concepts: Cache Read

Tag: `00000000011000000100`
Set: `010000`
Block Offset: `100000`

- These bits now tell us how to do the lookup in our cache!
- Use set index (`0b010000` = 16) to select the set
- Loop through lines in that set to find a matching tag (`0b00000000011000000100`)
- If found and valid bit is set: ***Hit!***
  - Locate data starting at byte offset (`0b100000`)

Cache Concepts: Cache Miss

- But what happens if the cache doesn't have our data?
- We have a *cache miss*
- If we have a free line in the set, just load data into there
- Otherwise, the set is full!
  - We have to *evict* a line according to a *replacement policy*.
  - **`cachelab`**: LRU (Least Recently Used)
  - Other policies exist!
- Finally, load our new line into the free slot.

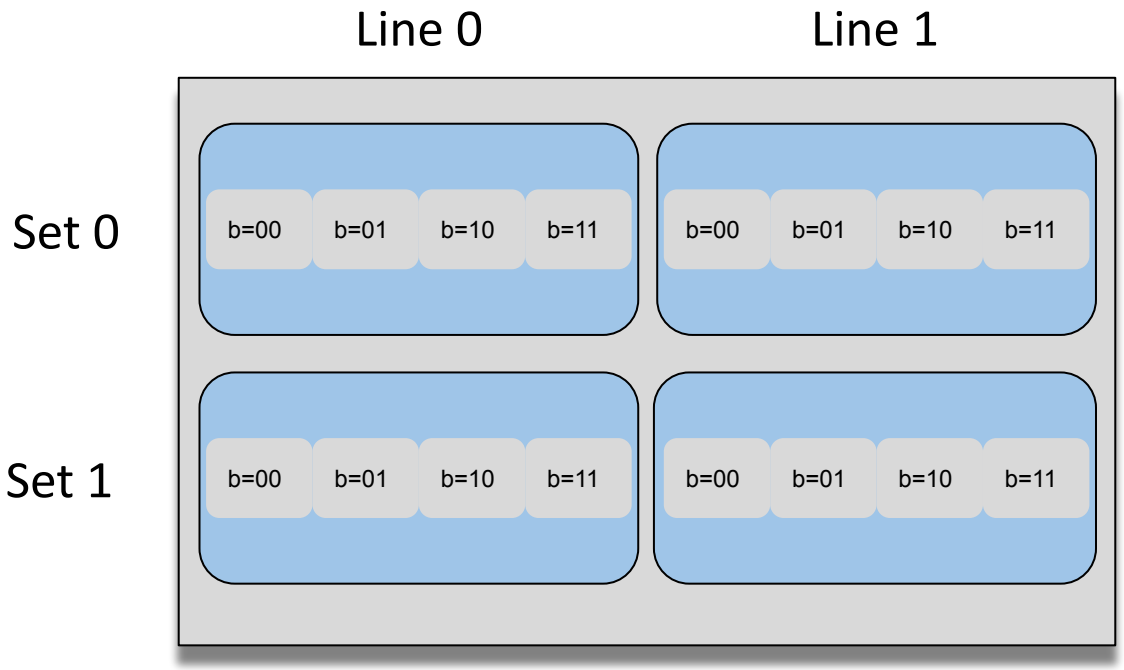
Cache Concepts: Dirty Bit

- You will implement a ***write-back, write-allocate*** policy for **`cachelab`**.
- *Write-Allocate:* Write misses load the line into cache, update it in place.
- *Write-Back*: Defer writing updates to memory until line is evicted.
  - Expensive to flush every evicted line to memory.
  - ***Dirty bit*** indicates whether cache line has been written to, and needs to be flushed to memory.

Example Trace

Example Trace

- We will use the following configuration:
  - **`s`** = 1, **`E`** = 2, **`b`** = 2



Example Trace: Reading a Trace

bpr.trace

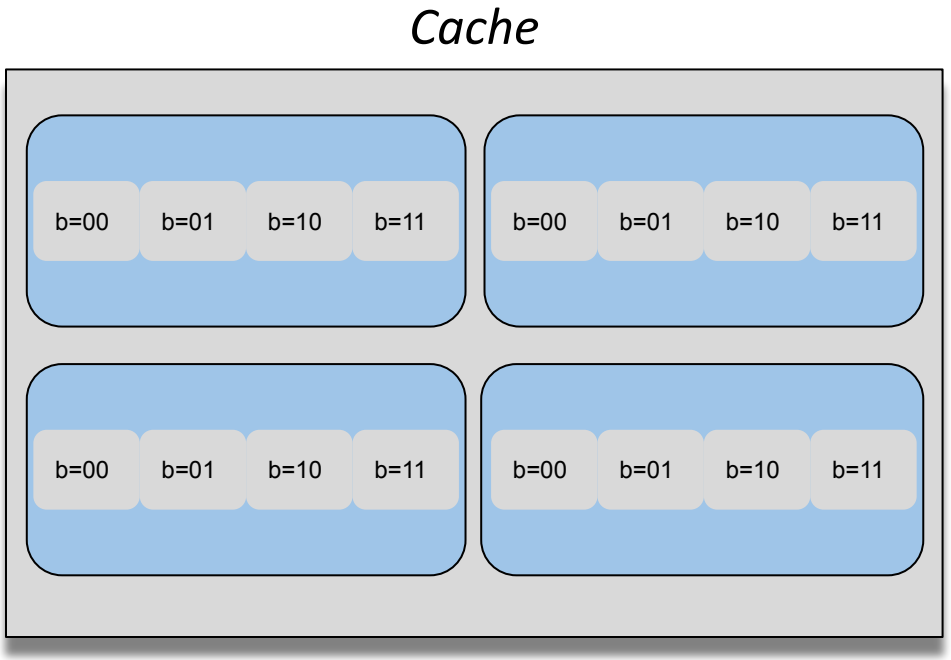
```
L 0,1
L 0,1
L 1,1
S 2,1
L 5,1
L 4,1
L 8,1
L 0,1
L 16,1
L 9,1
L 24,1
L 32,1
L 0,1
```

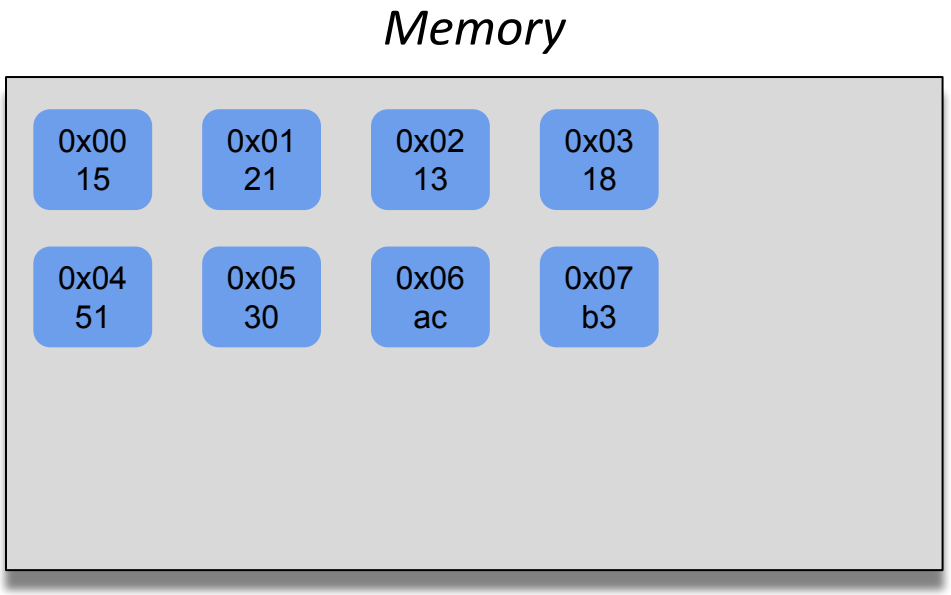
- **op <Addr>, <Size>**
- **op:**
  - **L – Load**
  - **S – Store**
- Note generally, we will **not** concern ourselves with the value being read/written
- For this trace however, we demonstrate write behavior through data values

Example Trace

```
...
L 0,1
...
```

Will this instruction result in a hit or a miss?





Example Trace

```
...
L 0,1
...
```

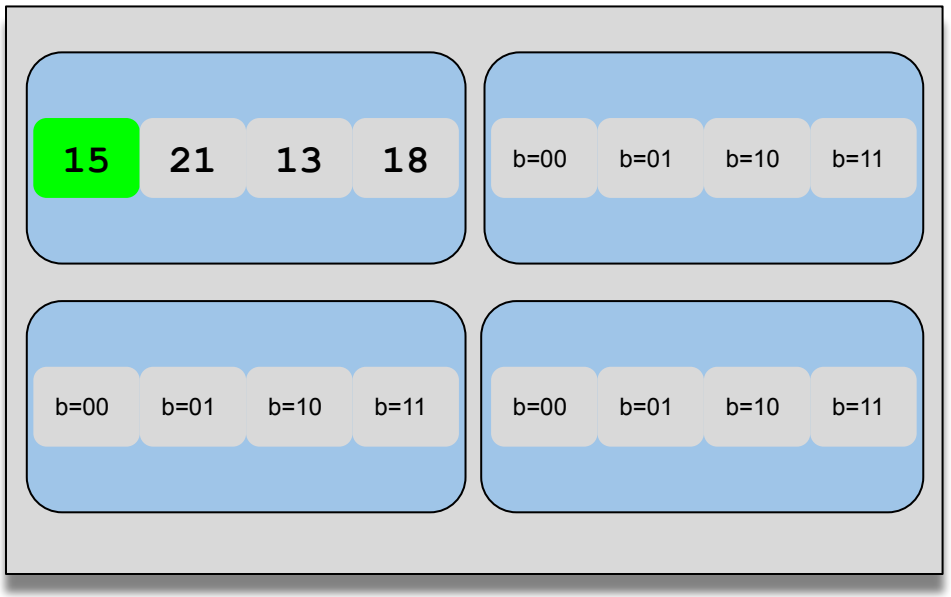
Miss

Why that line?

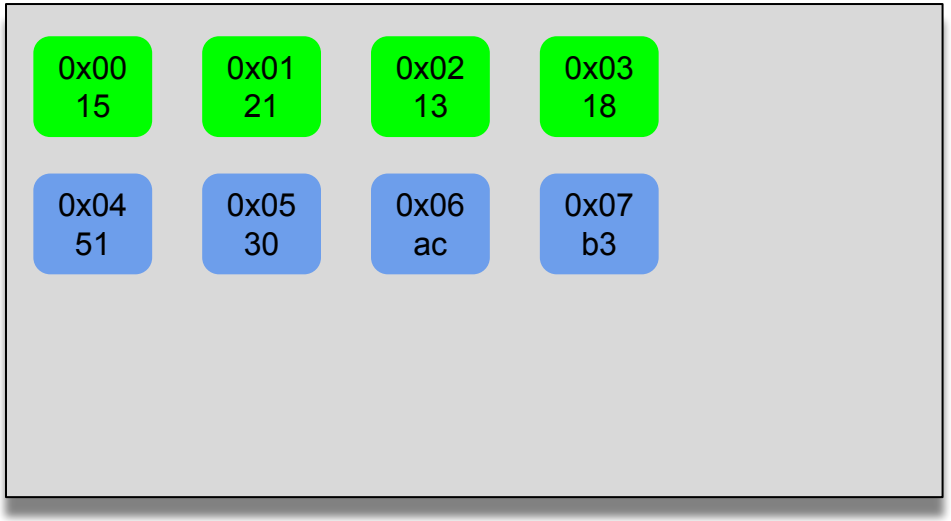
Where are those values from?

What kind of miss is this?

Cache



Memory



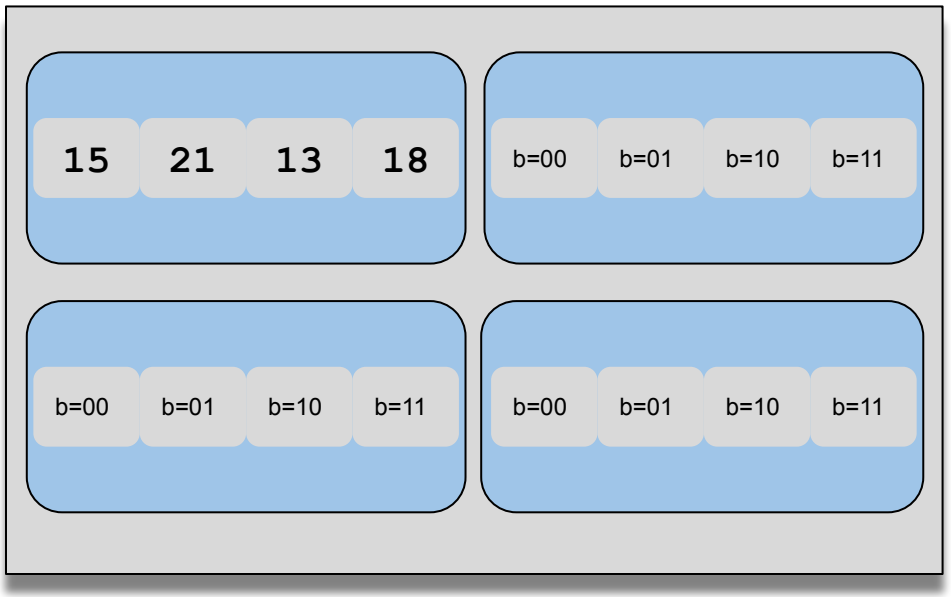
Example Trace

```
...
L 0,1
L 0,1
...
```

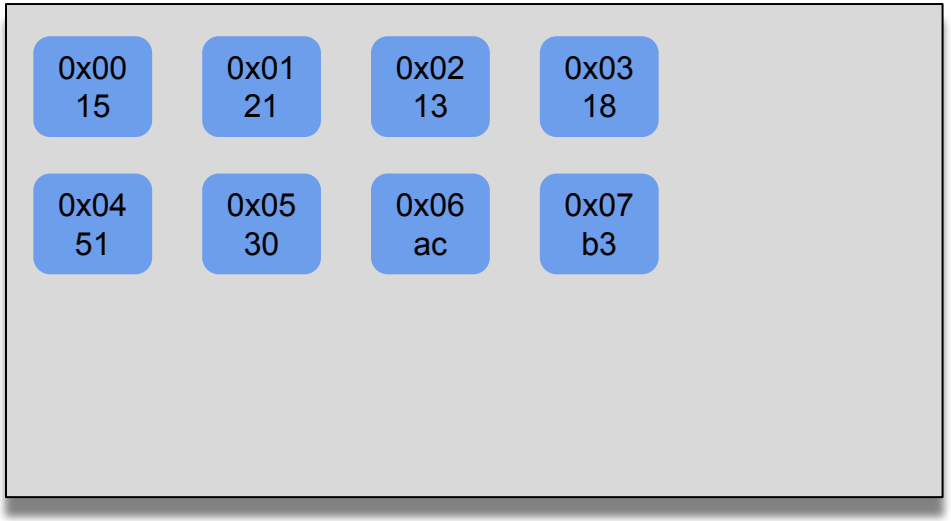
Miss
???

Will this instruction result in a hit or a miss?

Cache



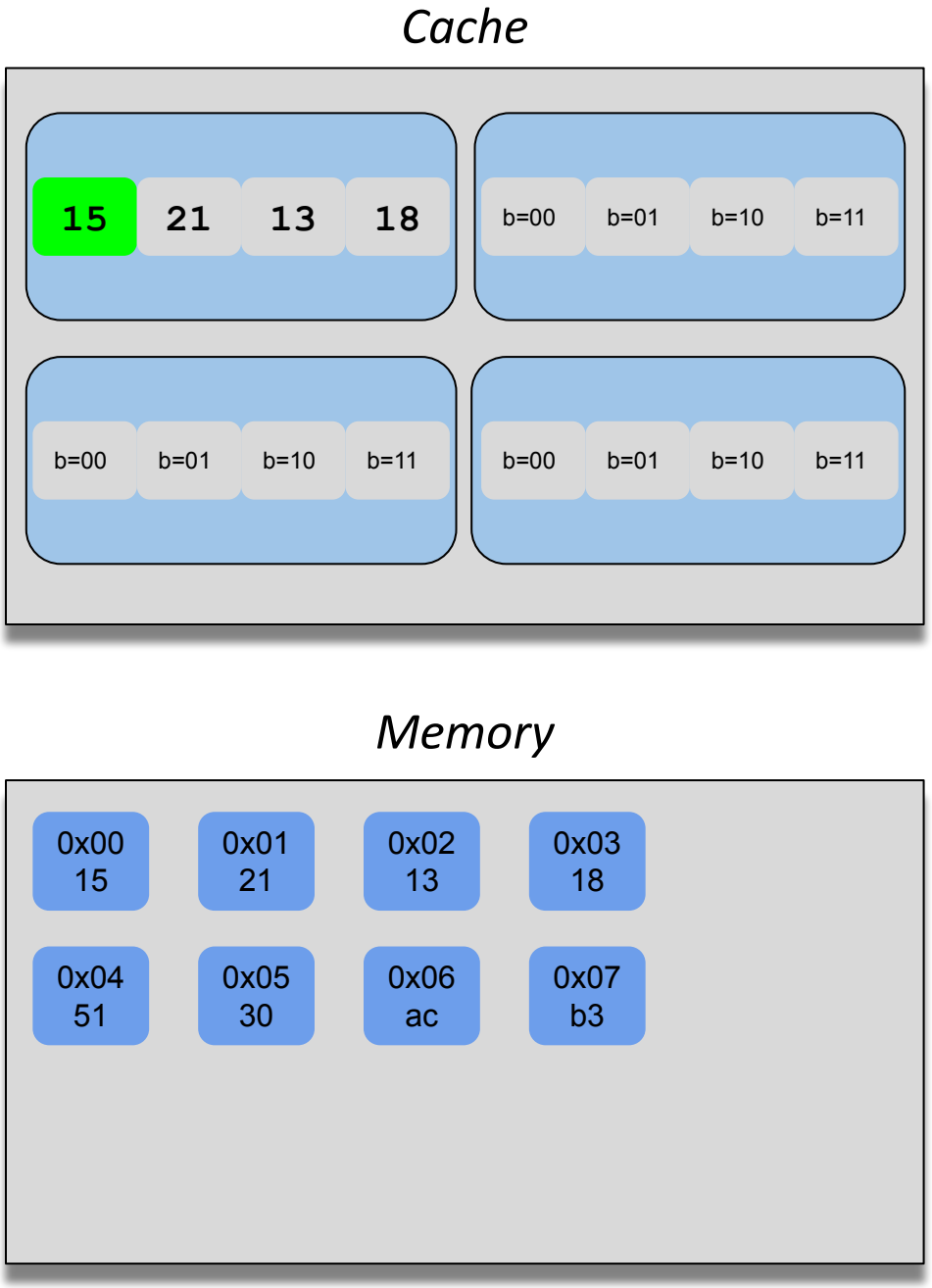
Memory



Example Trace

```
...
L 0,1
L 0,1
...
```

Miss
Hit!



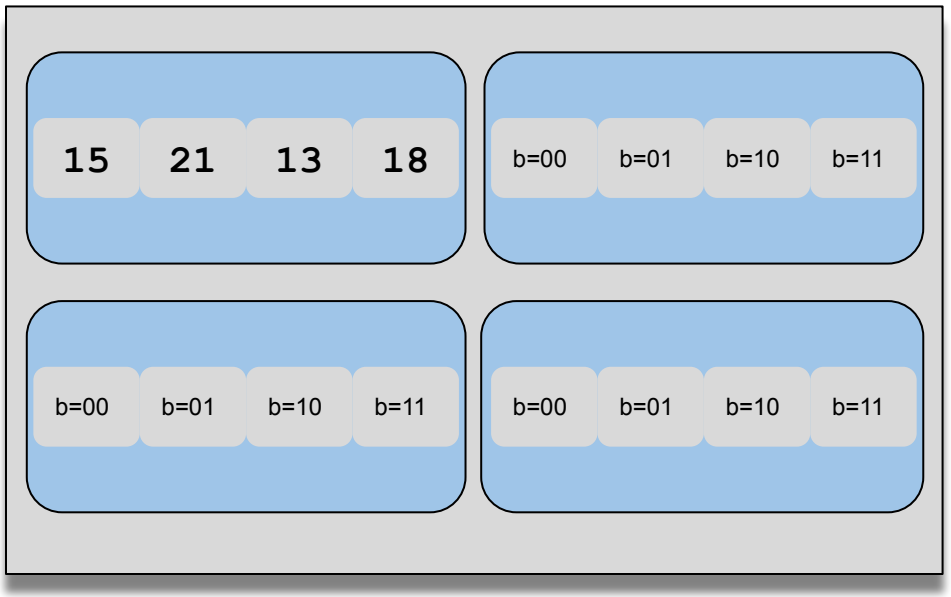
Example Trace

```
...
L 0,1
L 0,1
L 1,1
...
```

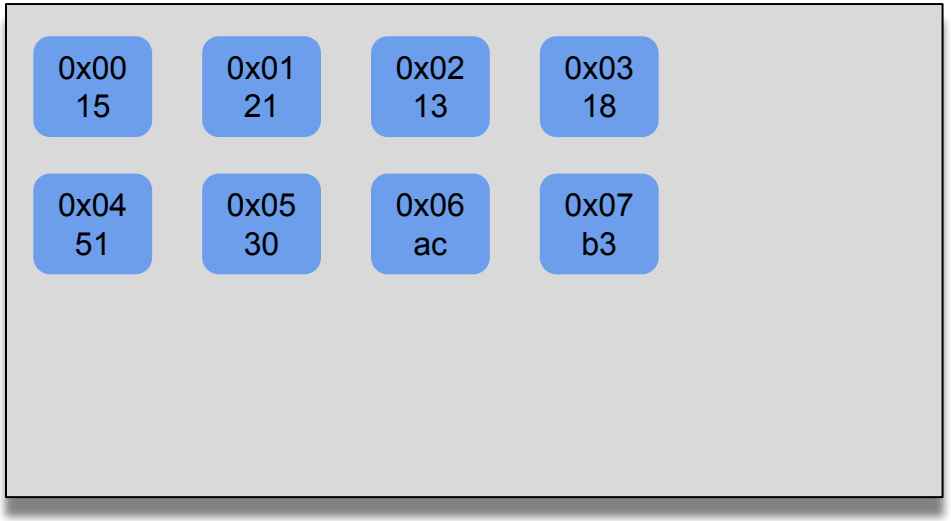
Miss
Hit!
???

Will this instruction result in a hit or a miss?

Cache



Memory



Example Trace

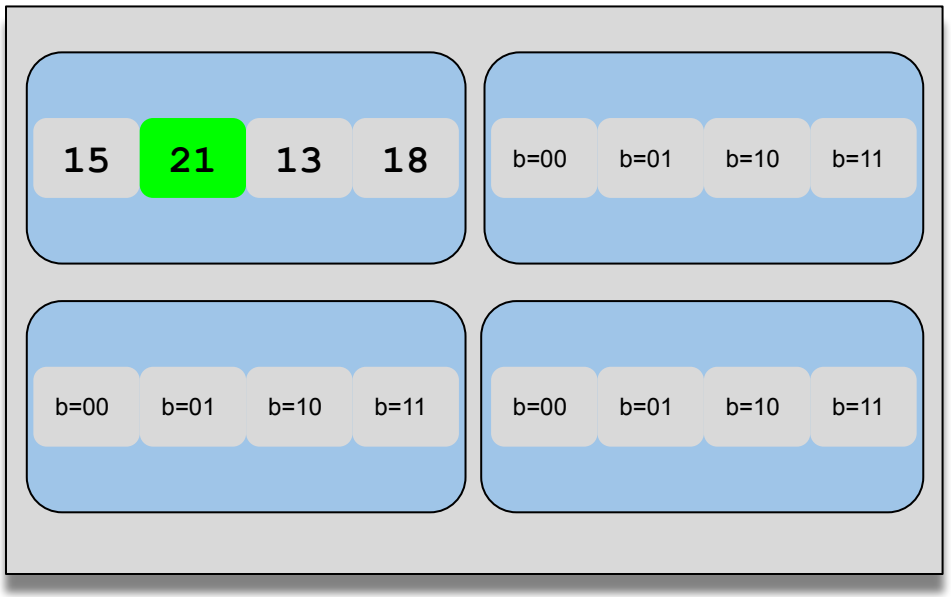
```
...
L 0,1
L 0,1
L 1,1
...
```

Miss
Hit!
Hit!

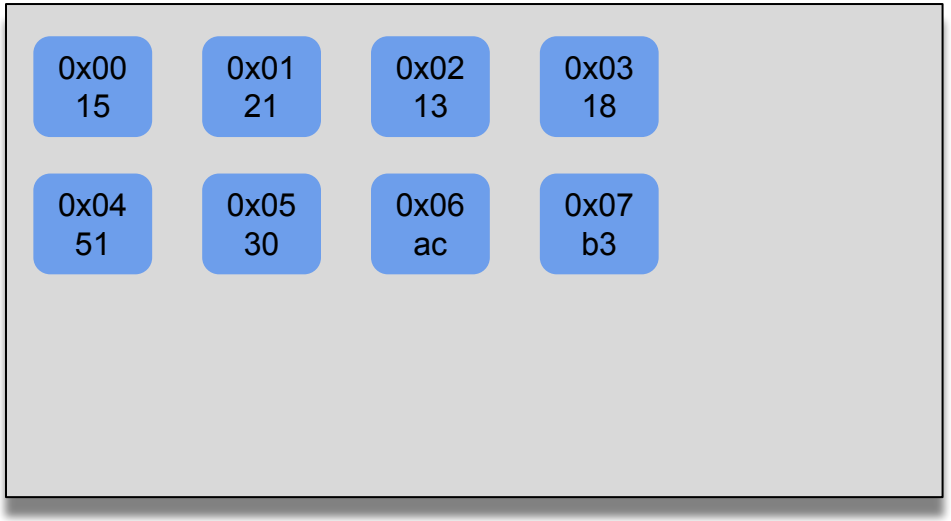
Not a miss!

We had already loaded all four bytes of the line into cache. Why?

Cache



Memory



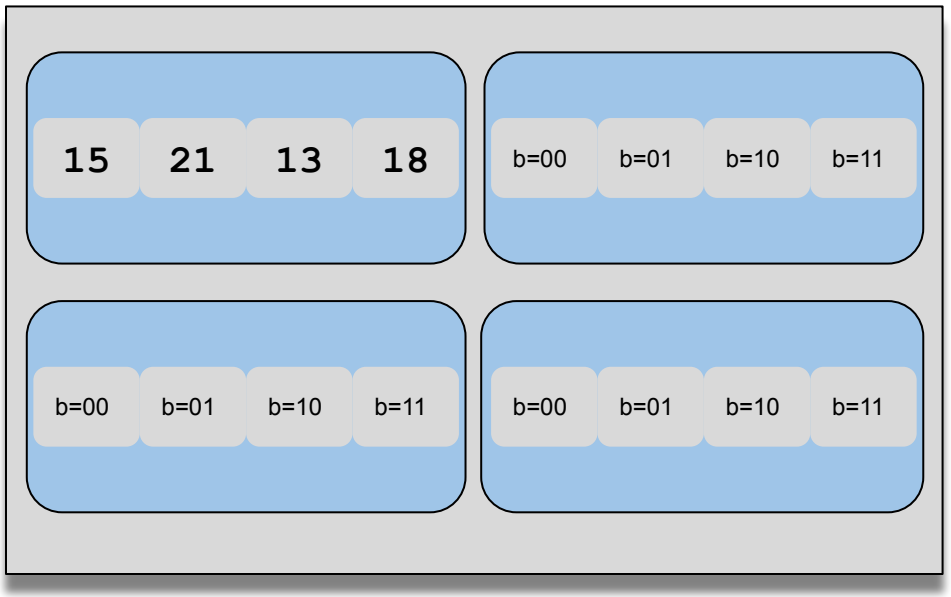
Example Trace

```
...
L 0,1
L 1,1
S 2,1
...
```

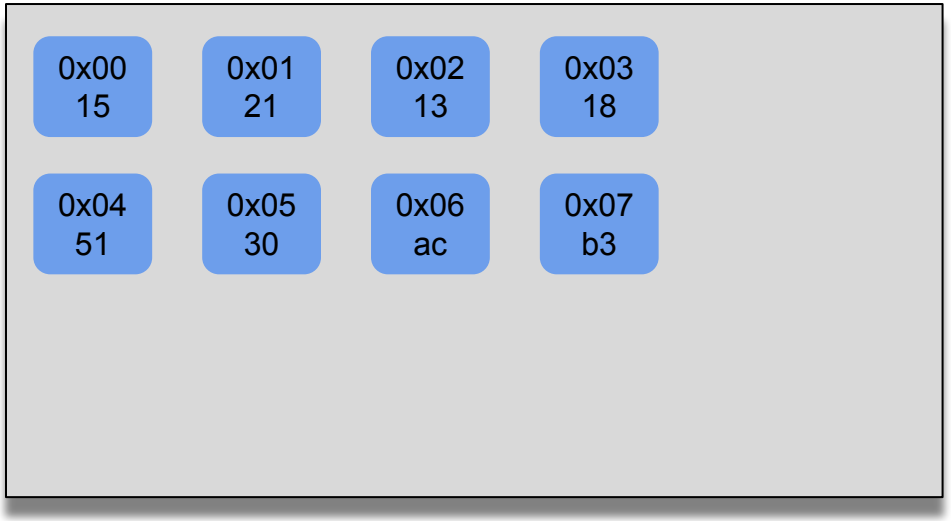
```
Hit!
Hit!
???
```

Will this instruction result in a hit or a miss?

Cache



Memory



Example Trace

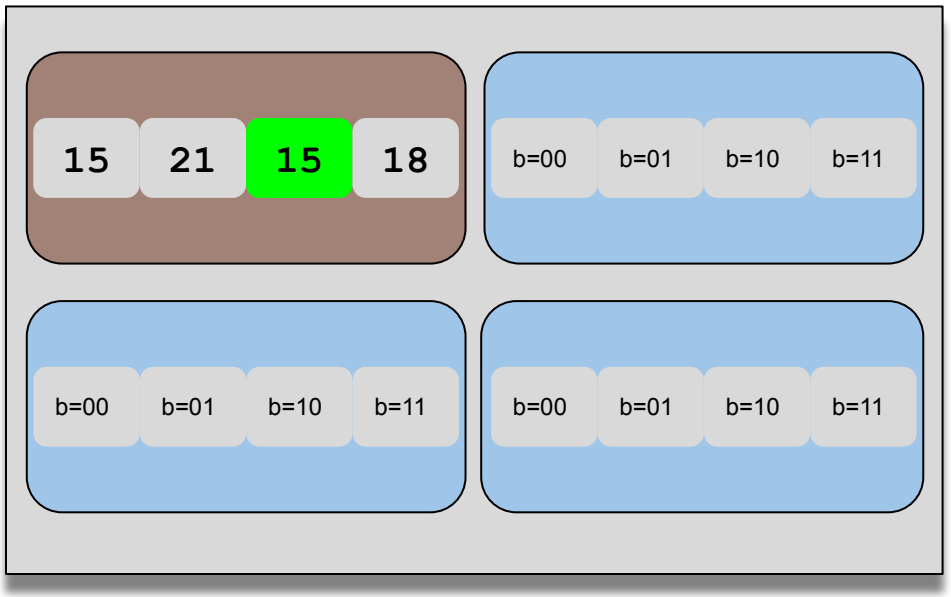
```
...
L 0,1
L 1,1
S 2,1
...
```

Hit!
Hit!
Hit!

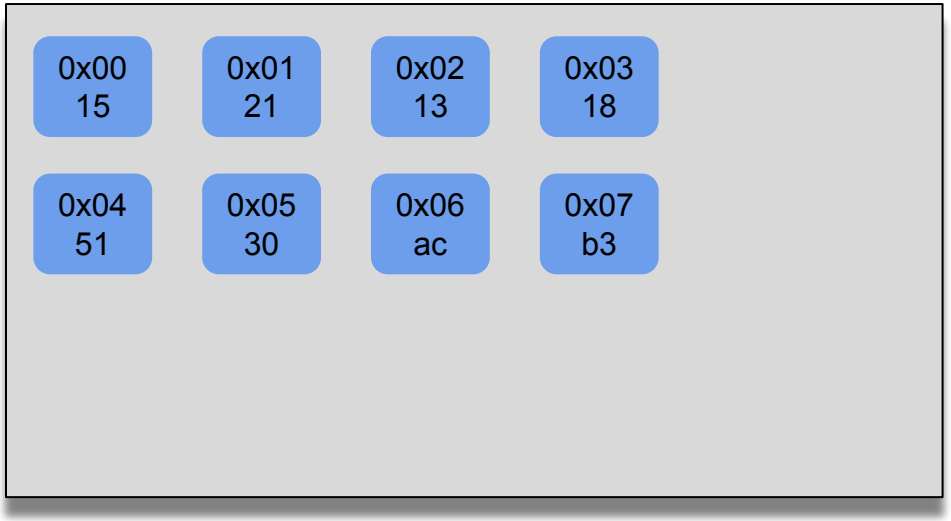
Write hit!

Set dirty bit.

Cache



Memory



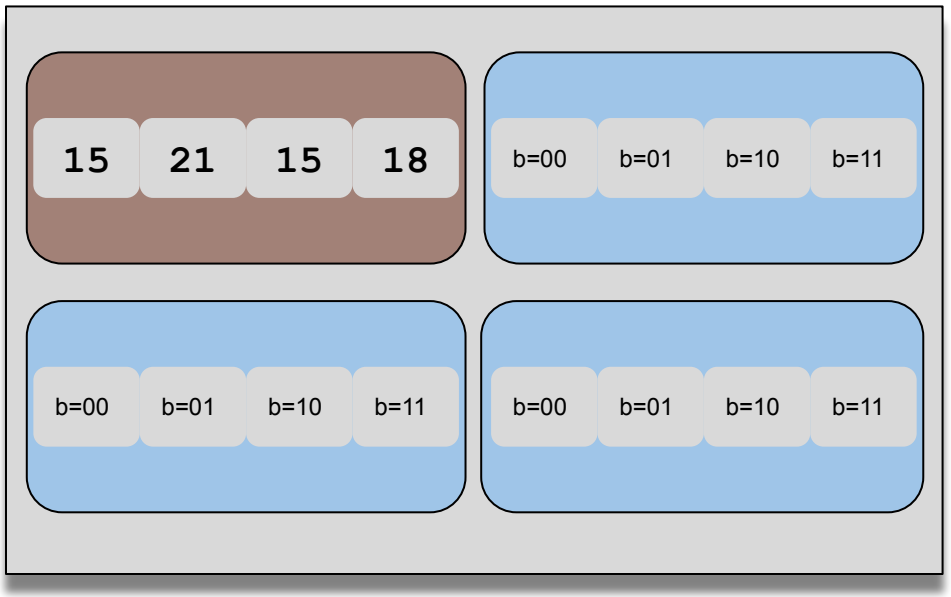
Example Trace

```
...
L 1,1
S 2,1
L 5,1
...
```

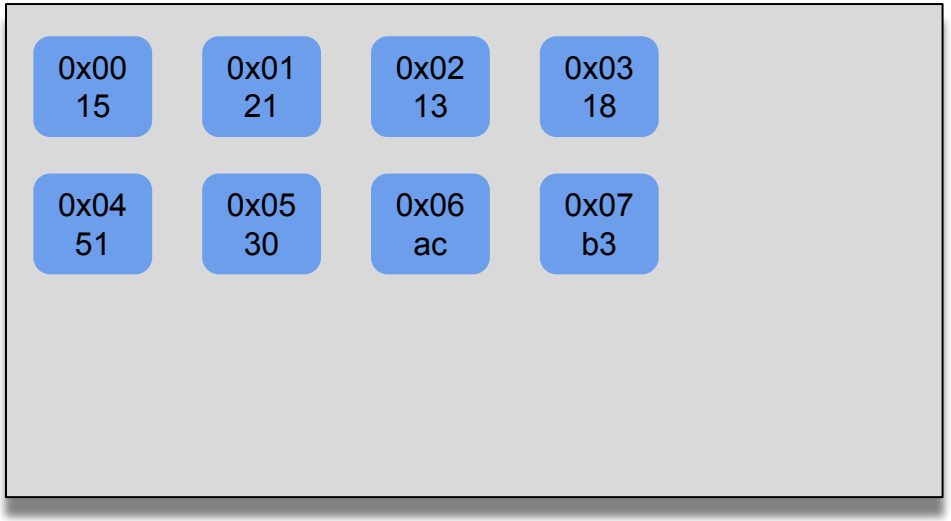
```
Hit!
Hit!
???
```

Will this instruction result in a hit or a miss?

Cache



Memory



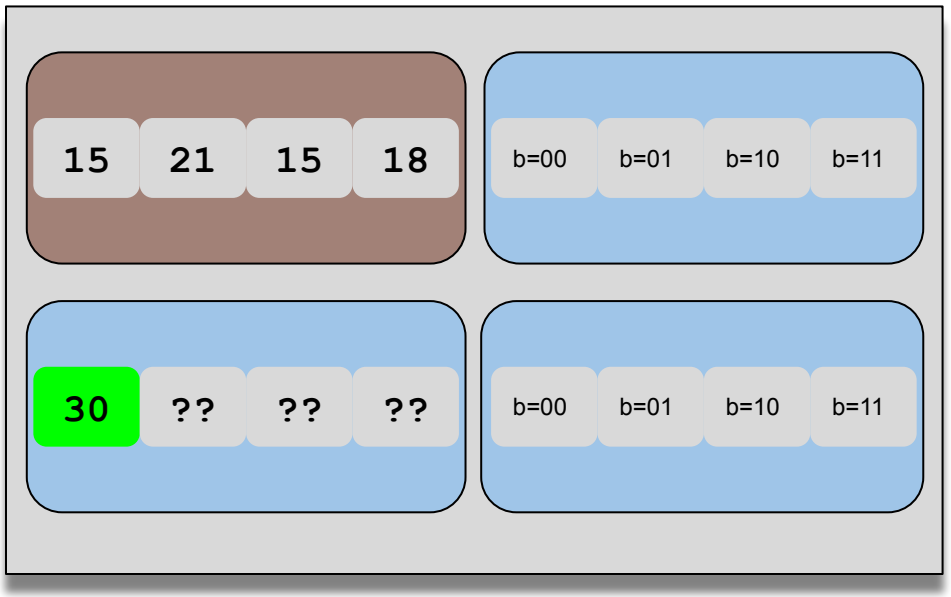
Example Trace

```
...
L 1,1
S 2,1
L 5,1
...
```

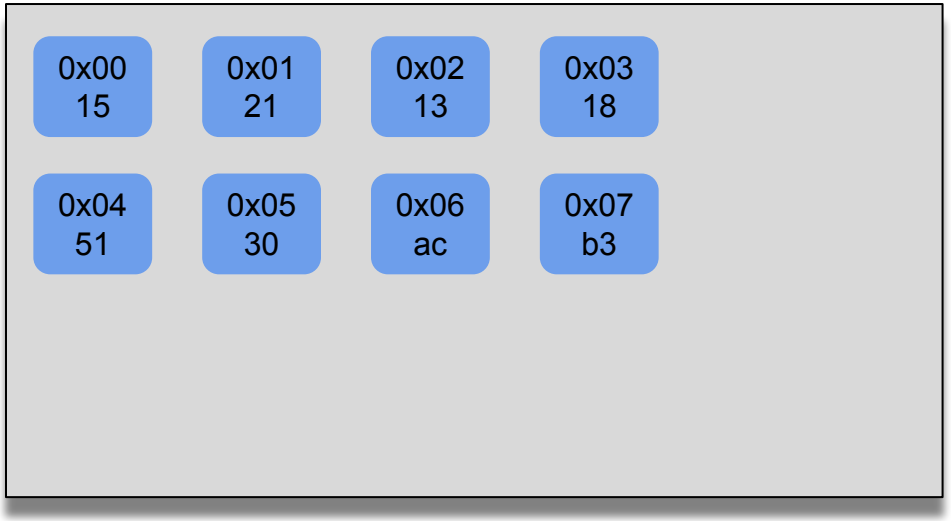
Hit!
Hit!
Miss

Do we load just one byte like this?

Cache



Memory



Example Trace

```
...
L 1,1
S 2,1
L 5,1
...
```

Hit!
Hit!
Miss

Do we load just one byte like this?

No!

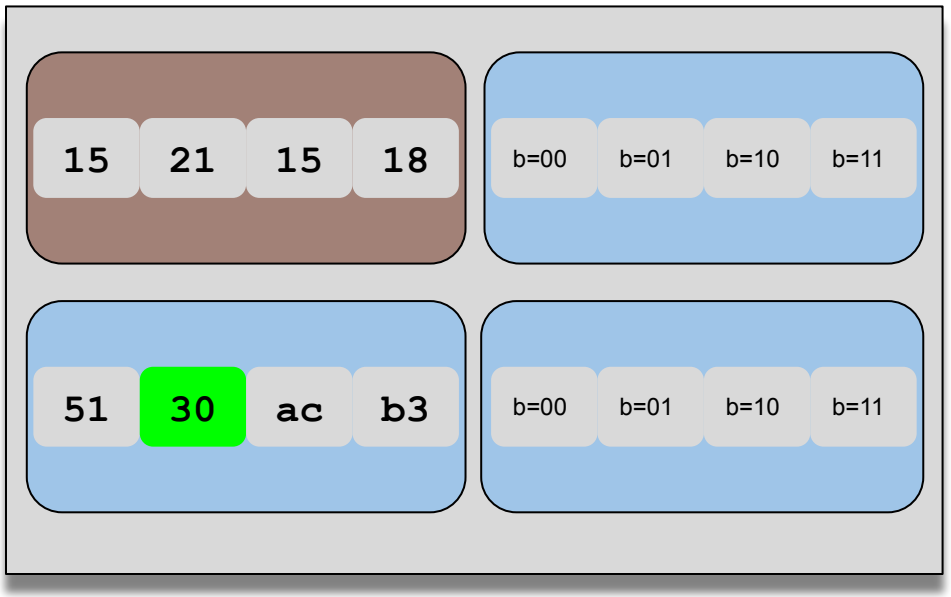
Example Trace

```
...
L 1,1
S 2,1
L 5,1
...
```

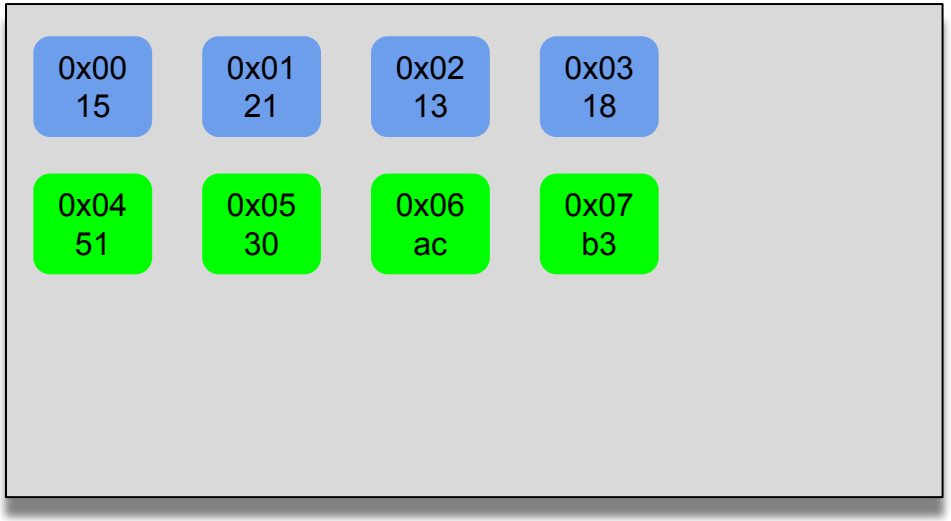
Hit!
Hit!
Miss

Why do we start with a byte from below address 5?

Cache



Memory



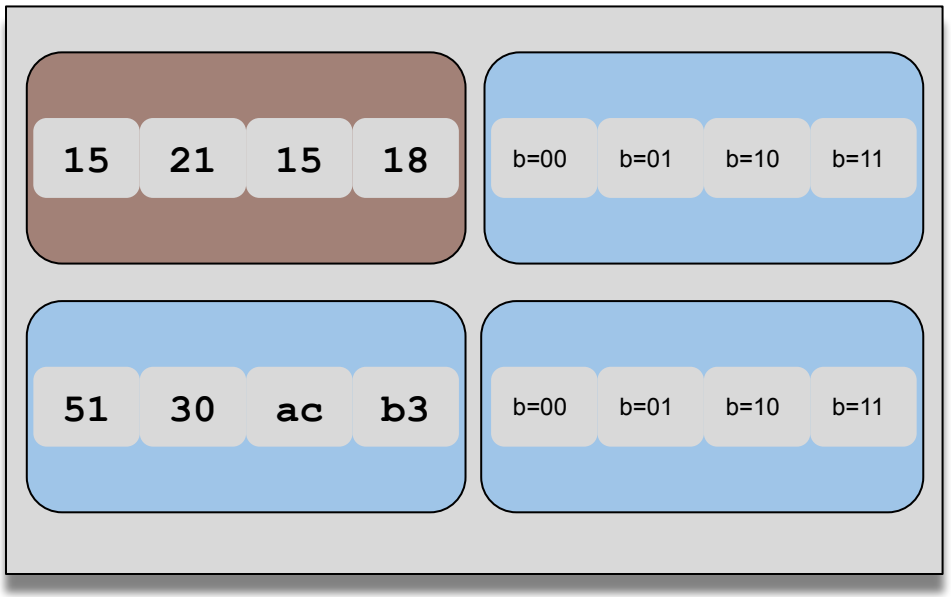
Example Trace

```
...
S 2,1
L 5,1
L 4,1
...
```

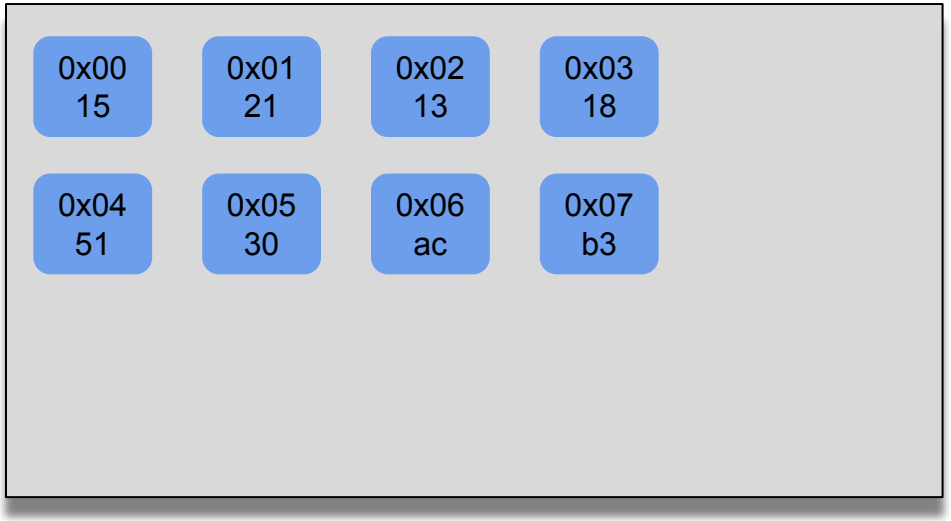
```
Hit!
Miss
???
```

Will this instruction result in a hit or a miss?

Cache



Memory

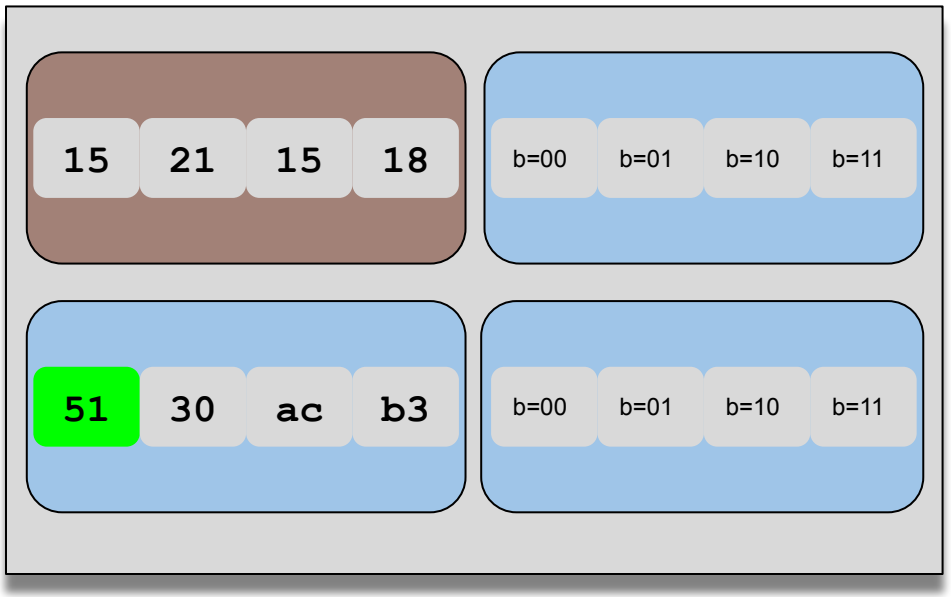


Example Trace

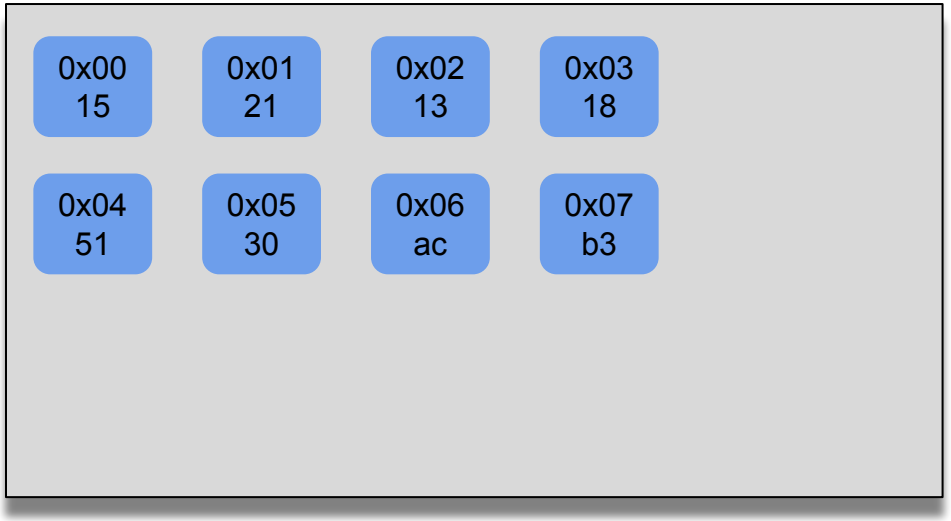
```
...
S 2,1
L 5,1
L 4,1
...
```

Hit!
Miss
Hit!

Cache



Memory



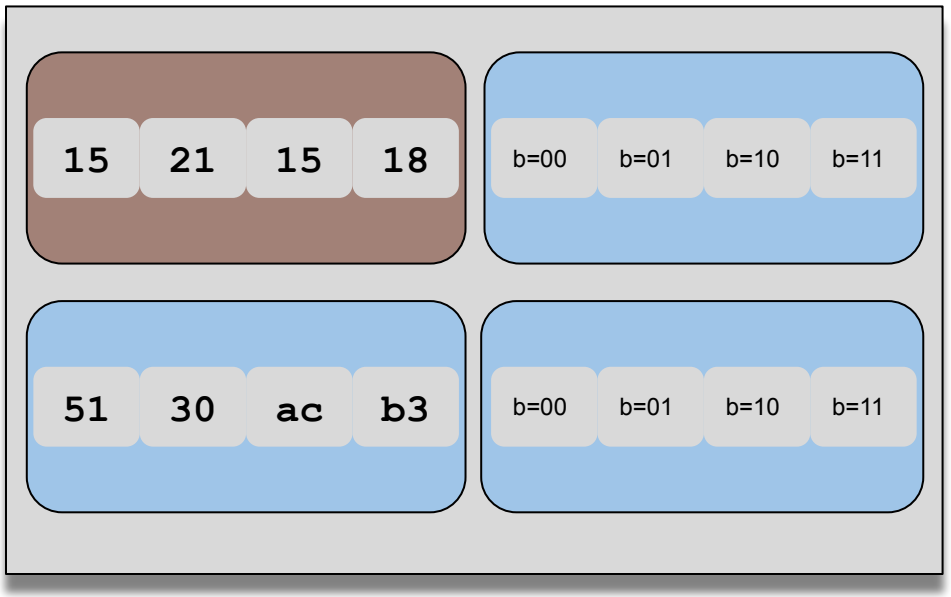
Example Trace

```
...
L 5,1
L 4,1
L 8,1
...
```

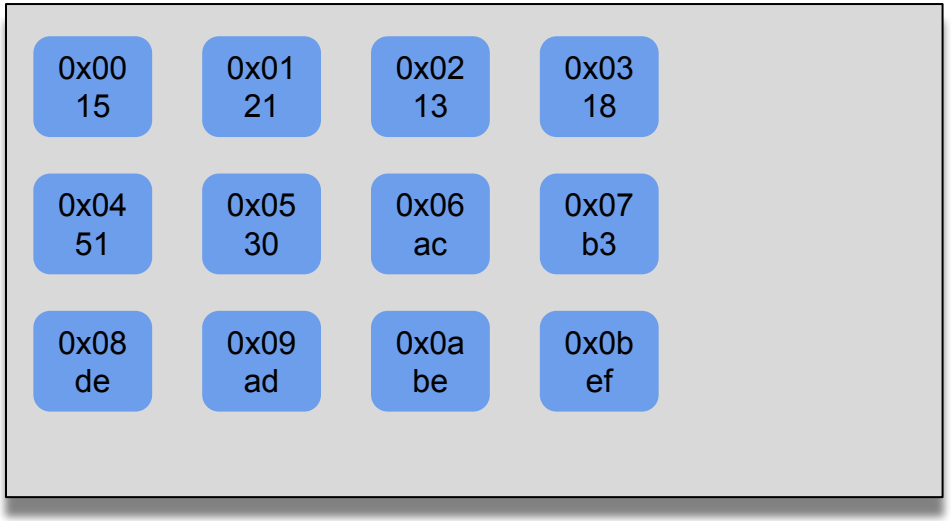
```
Miss
Hit!
???
```

Will this instruction result in a hit or a miss?

Cache



Memory



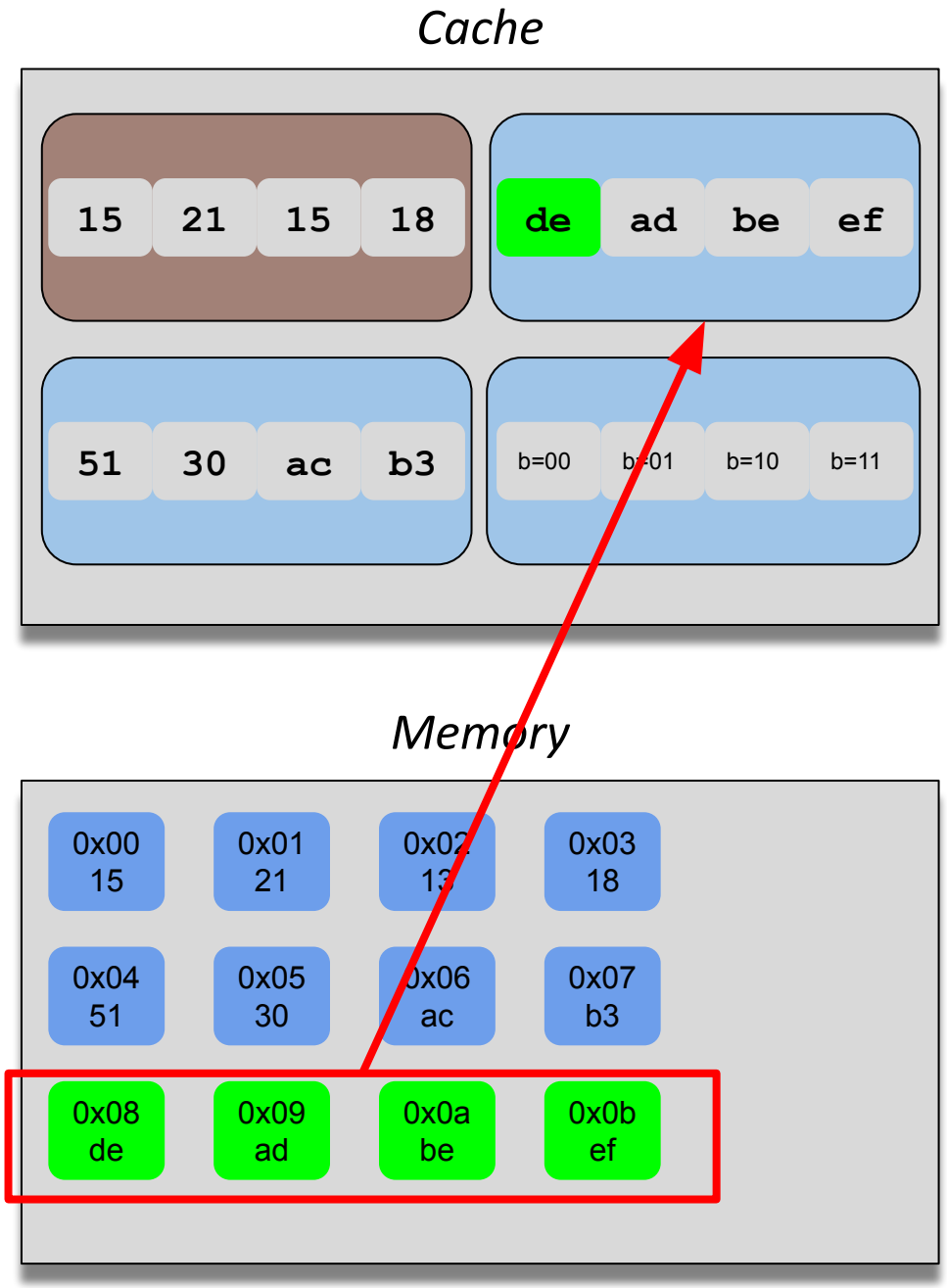
Example Trace

```
...
L 5,1
L 4,1
L 8,1
...
```

Miss
Hit!
Miss

Miss!

We had a free line, so just load the data into there.



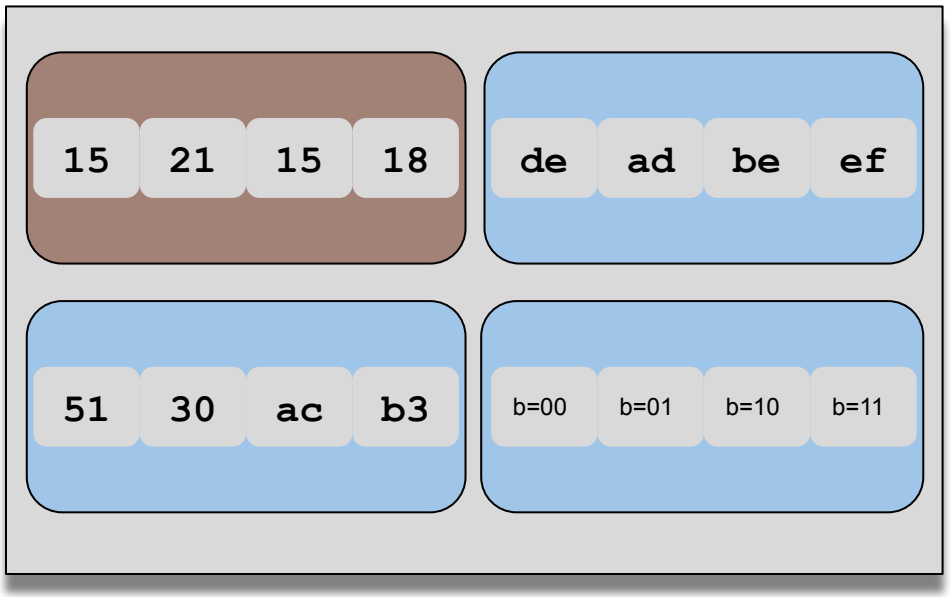
Example Trace

```
...
L 4,1
L 8,1
L 0,1
...
```

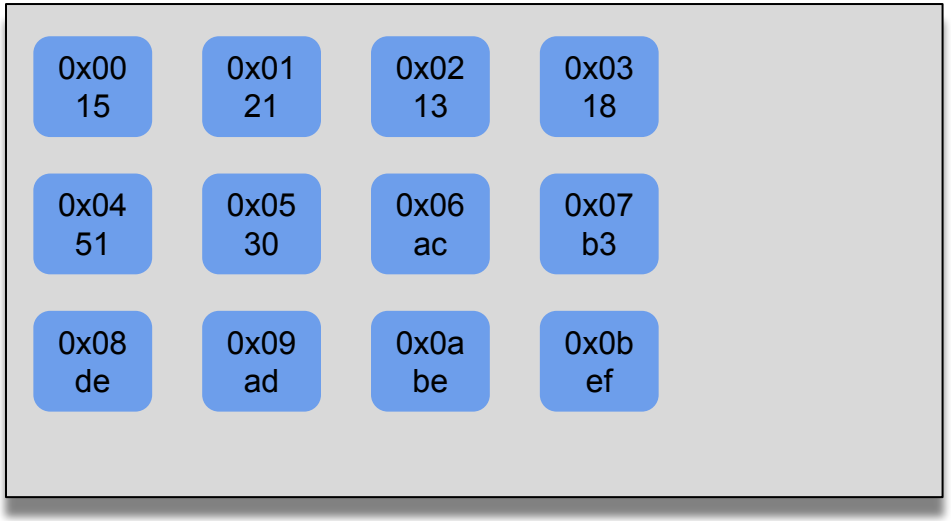
```
Hit!
Miss
???
```

Will this instruction result in a hit or a miss?

Cache



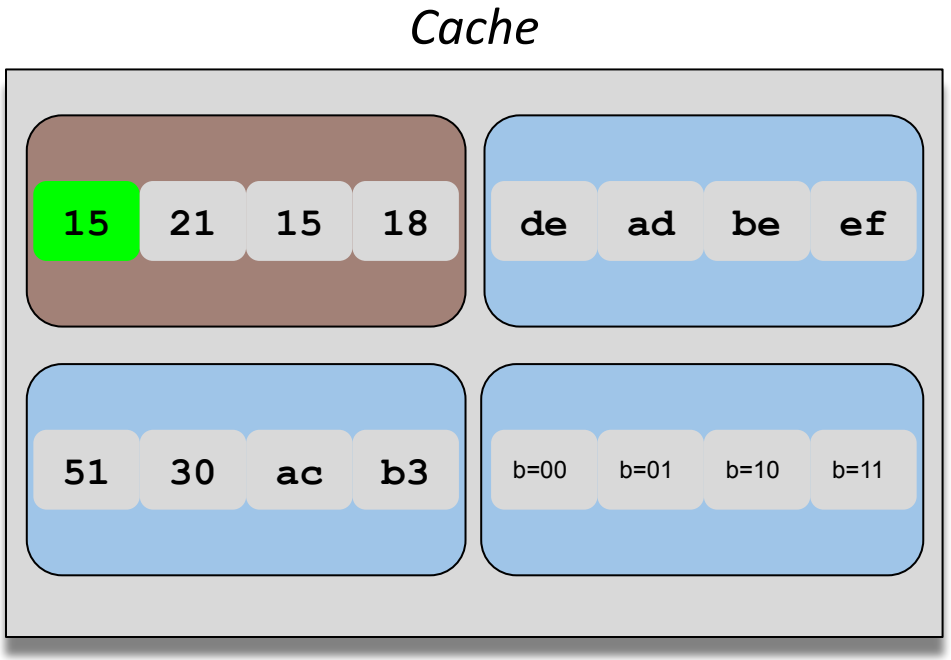
Memory

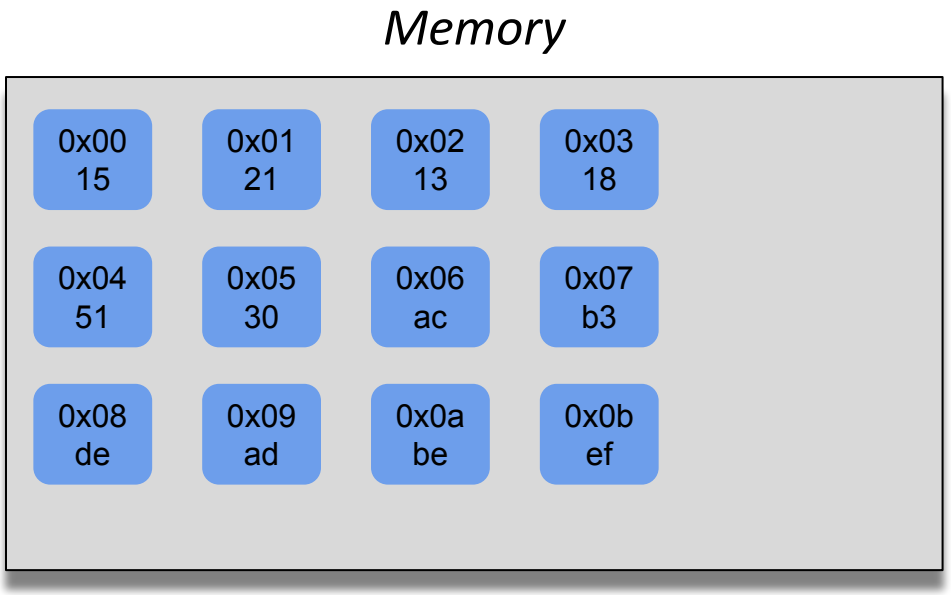


Example Trace

```
...
L 4,1
L 8,1
L 0,1
...
```

Hit!
Miss
Hit!



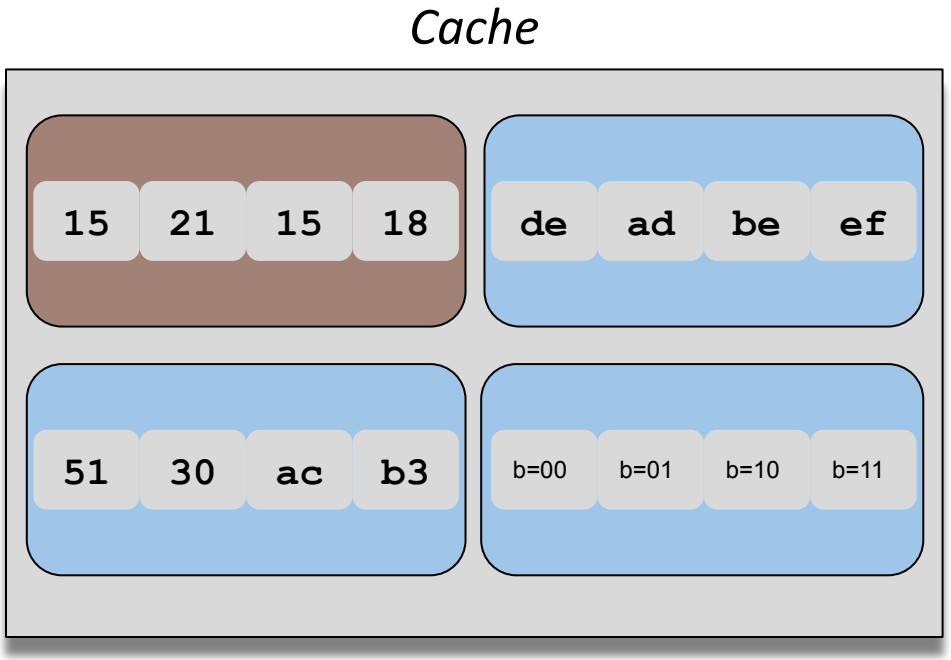


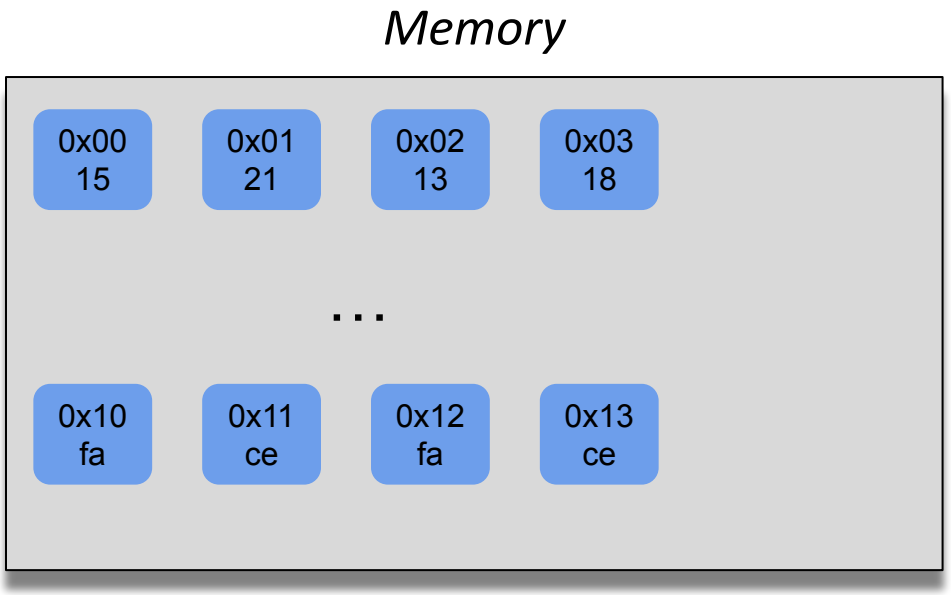
Example Trace

```
...
L 8,1
L 0,1
L 16,1
...
```

```
Miss
Hit!
???
```

Will this instruction result in a hit or a miss?





Example Trace

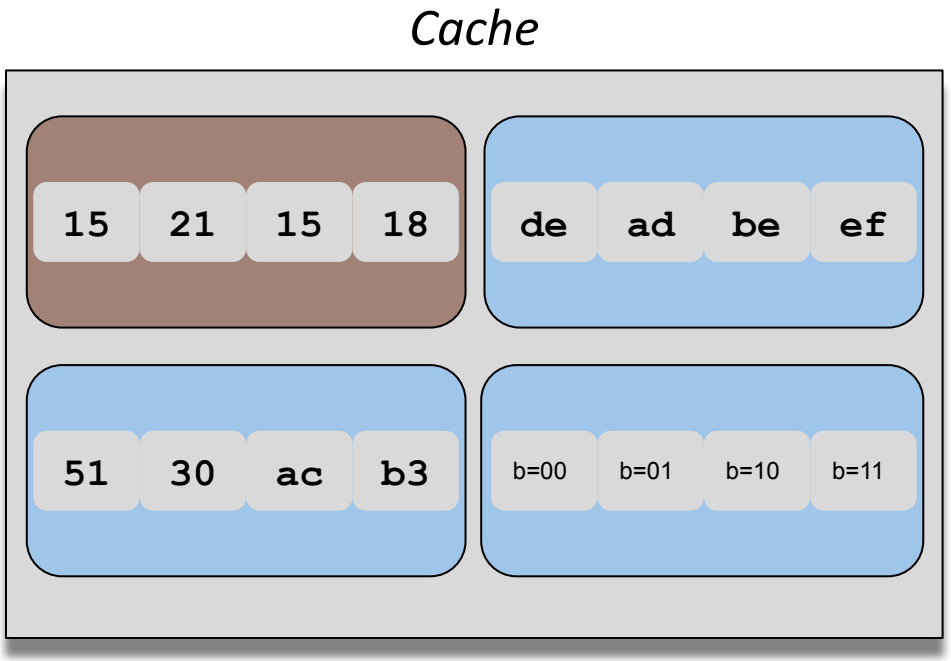
```
...
L 8,1
L 0,1
L 16,1
...
```

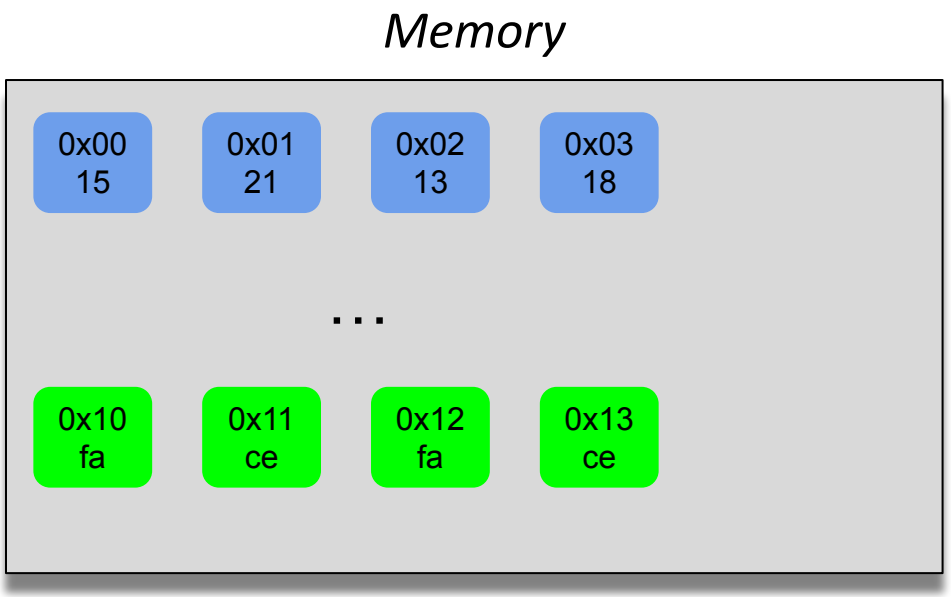
Miss
Hit!
Miss

What kind of miss is this?

16 = 0b10000

=> Set Index 0
=> Have to evict!

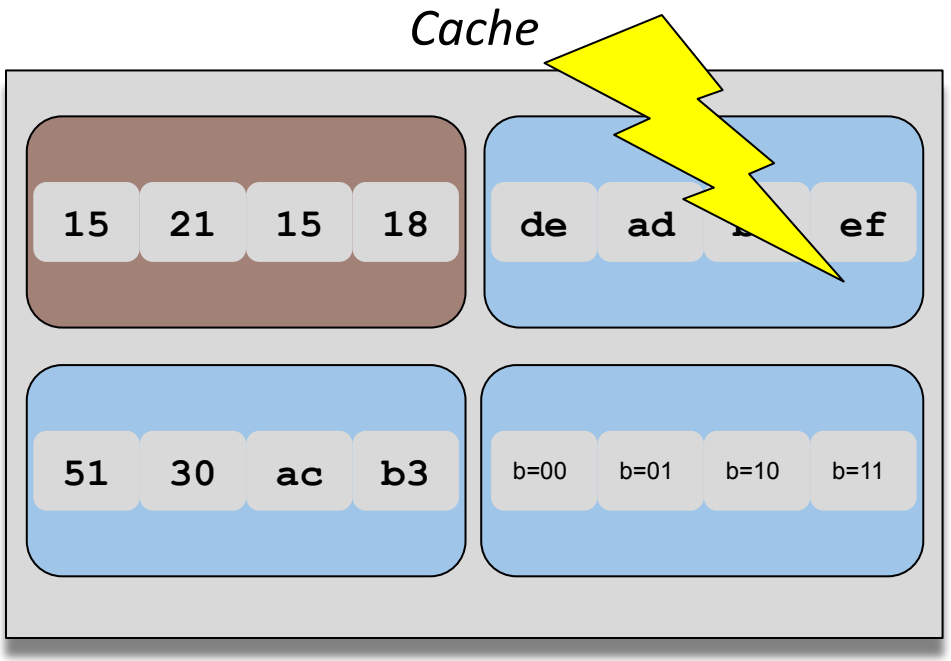




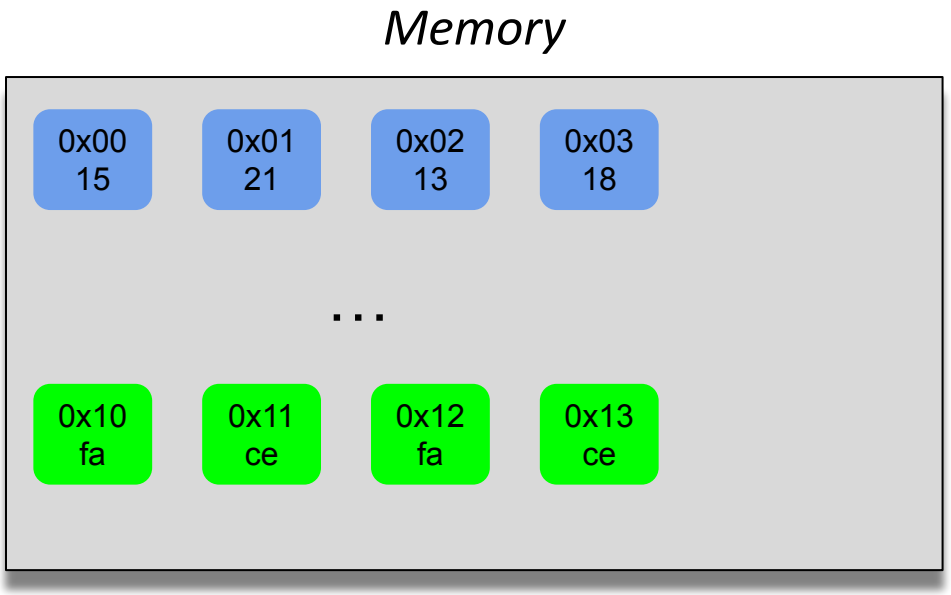
Example Trace

```
...
L 8,1
L 0,1
L 16,1
...
```

```
Miss
Hit!
Miss
```



1. ***Cold Miss*** *(first time seeing this block)*
1. ***Evict LRU*** *(Least Recently Used) line from set 0*

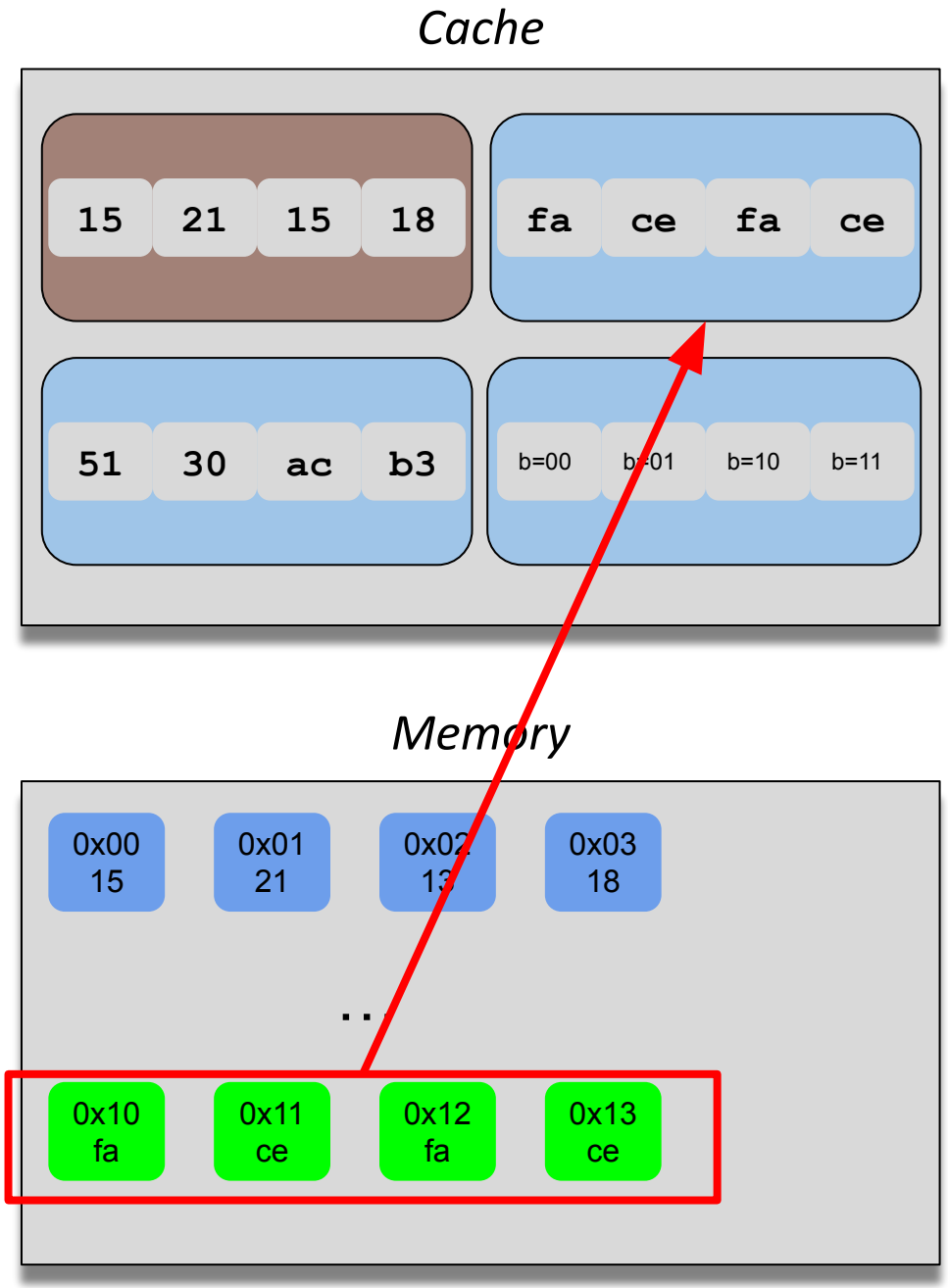


Example Trace

```
...
L 8,1
L 0,1
L 16,1
...
```

Miss
Hit!
Miss

Load new data into line



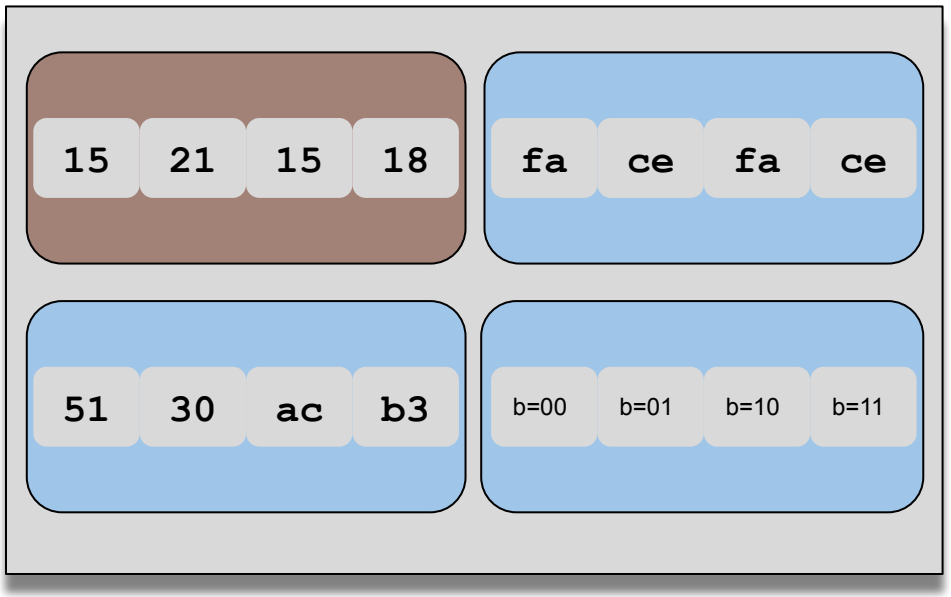
Example Trace

```
...
L 0,1
L 16,1
L 9,1
...
```

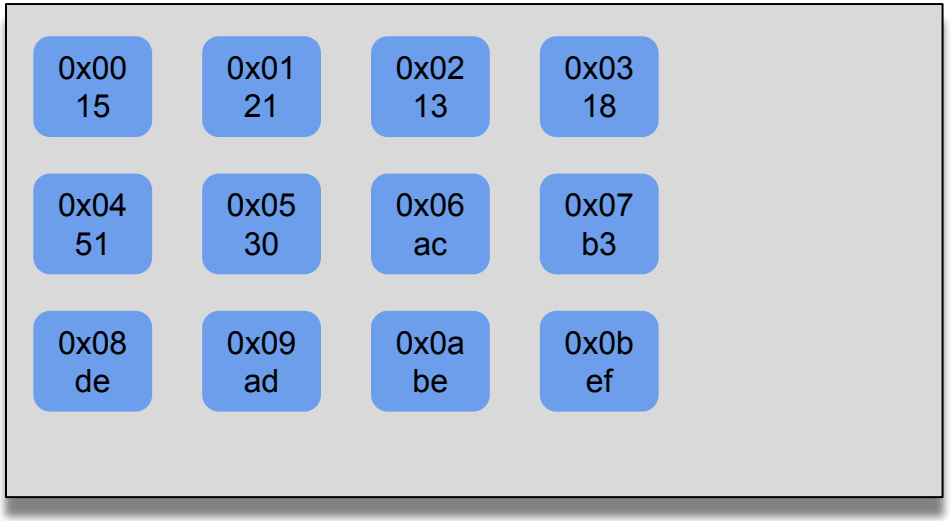
Hit!
Miss
???

Will this instruction result in a hit or a miss?

Cache



Memory



Example Trace

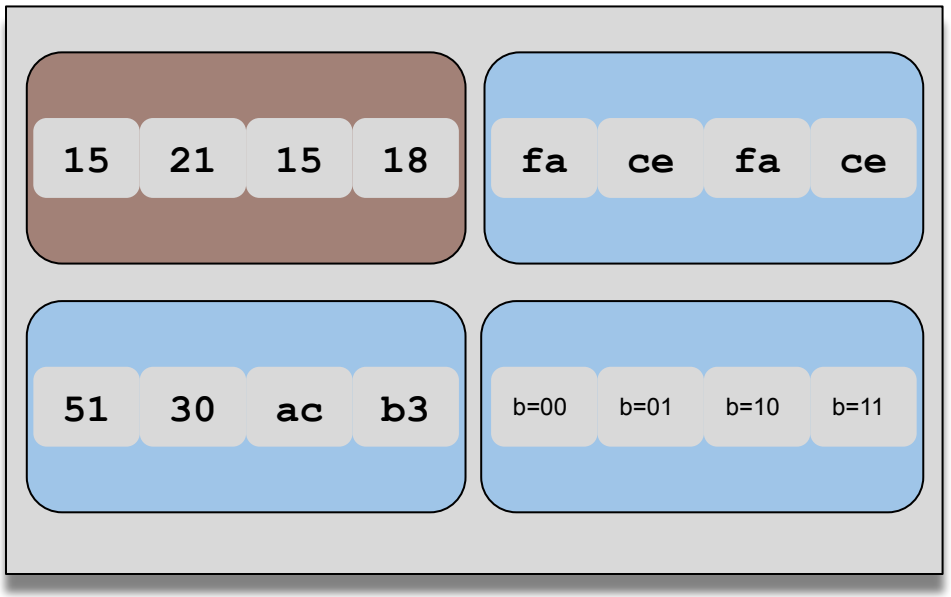
```
...
L 0,1
L 16,1
L 9,1
...
```

Hit!
Miss
Miss

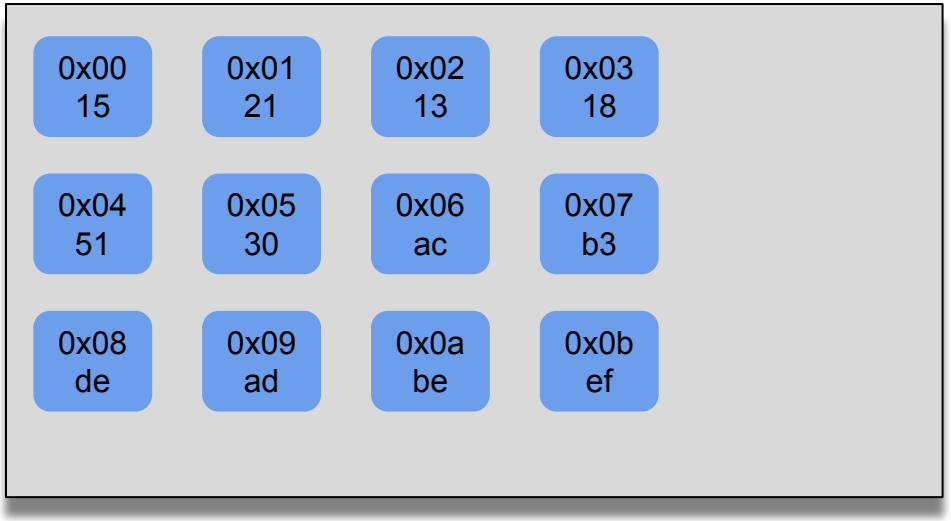
What kind of miss is this?

Has the block been in the cache before?

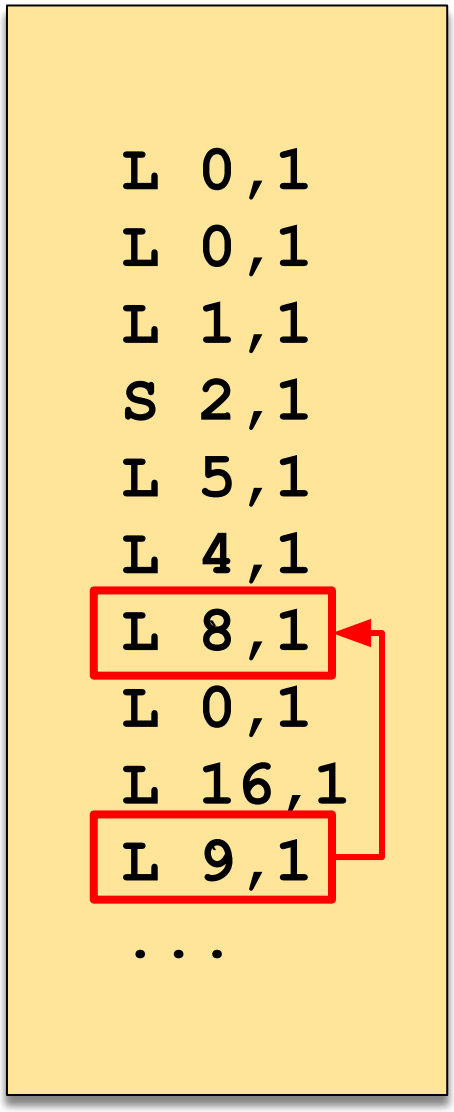
Cache



Memory

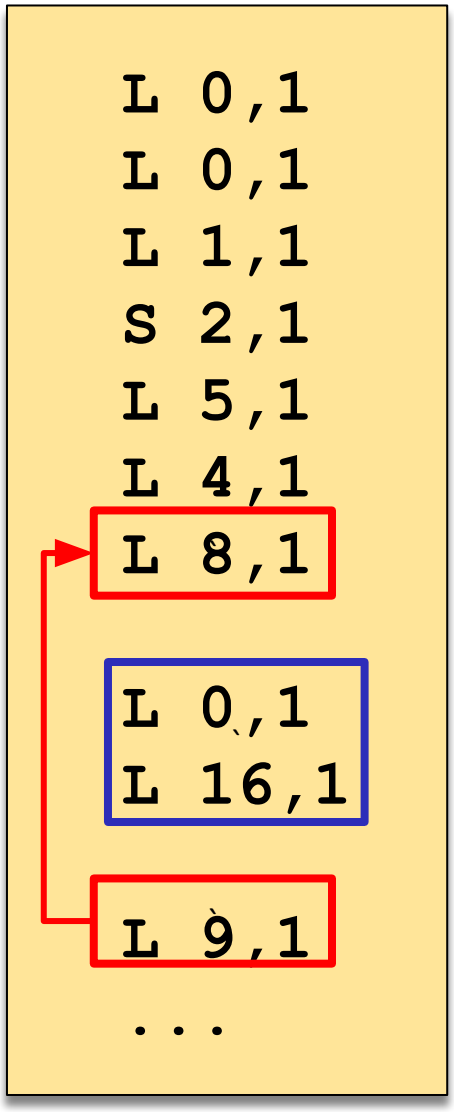


Cache Concepts: Conflict/Capacity Misses



- Has this block been in the cache before?
- Yes!
- If we've seen the block before:
  - Not a cold miss
  - Either a *conflict miss* or a *capacity miss*.

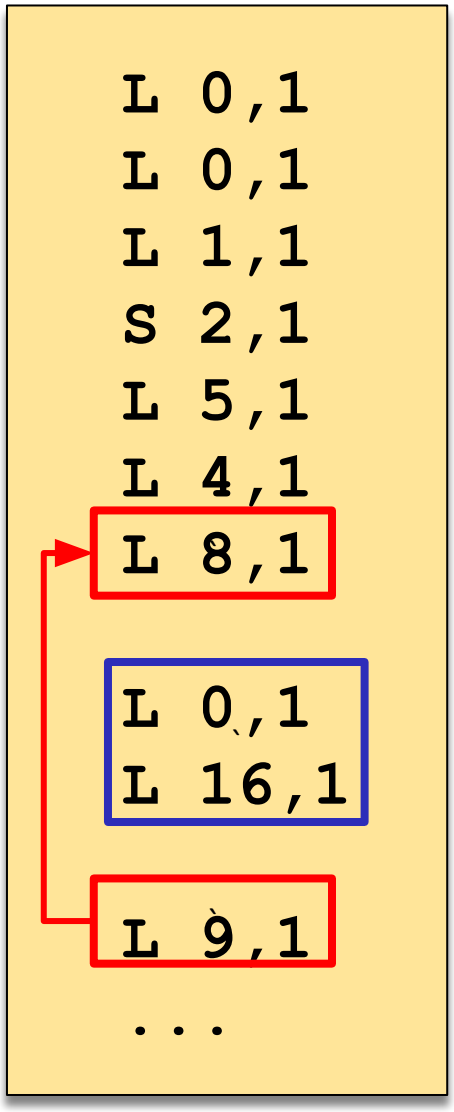
Cache Concepts: Conflict/Capacity Misses



How to distinguish between the two:

1. Find the last reference to that block in the trace.
2. Count the number of *unique* blocks referenced *in-between*:
    a. If the number is greater than or equal to the total number of lines in the cache: ***Capacity Miss***
    b. Otherwise: ***Conflict Miss***

Cache Concepts: Conflict/Capacity Misses



- In this case:
  - *Two* unique blocks in between current reference and last reference.
  - But we have *four* total lines in the cache
  - So we have a ***Conflict Miss***.

Example Trace

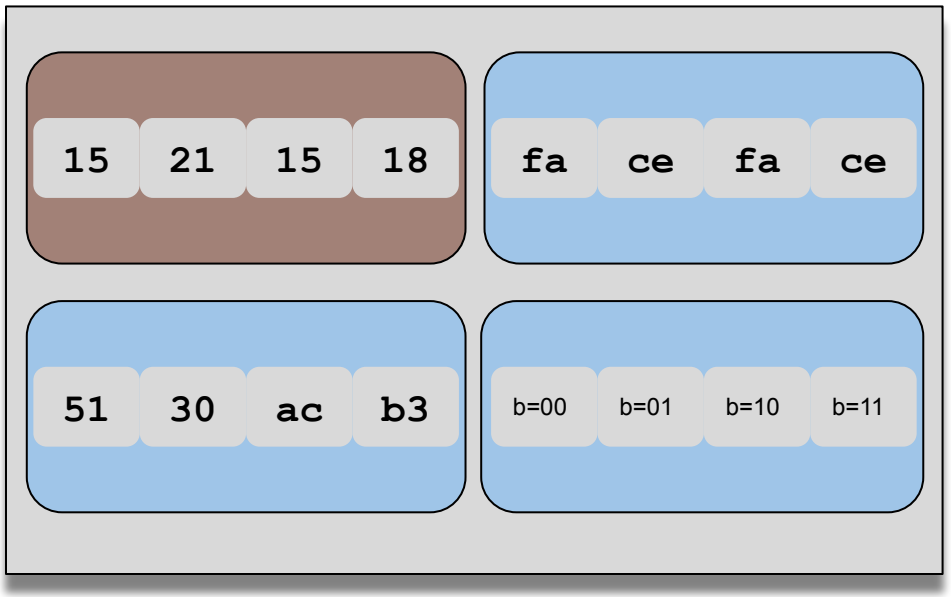
```
...
L 0,1
L 16,1
L 9,1
...
```

Hit!
Miss
Miss

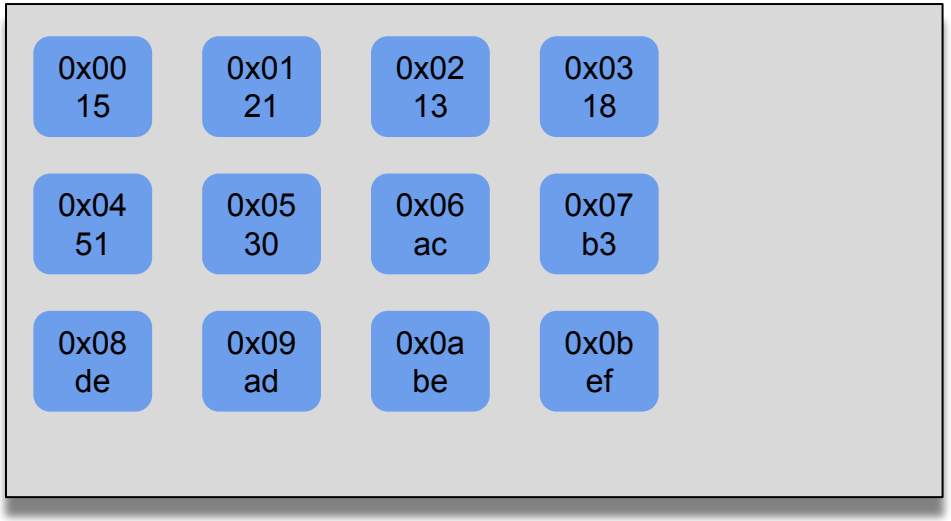
9 = 0b1001

=> Set Index 0
=> Have to evict!

Cache



Memory



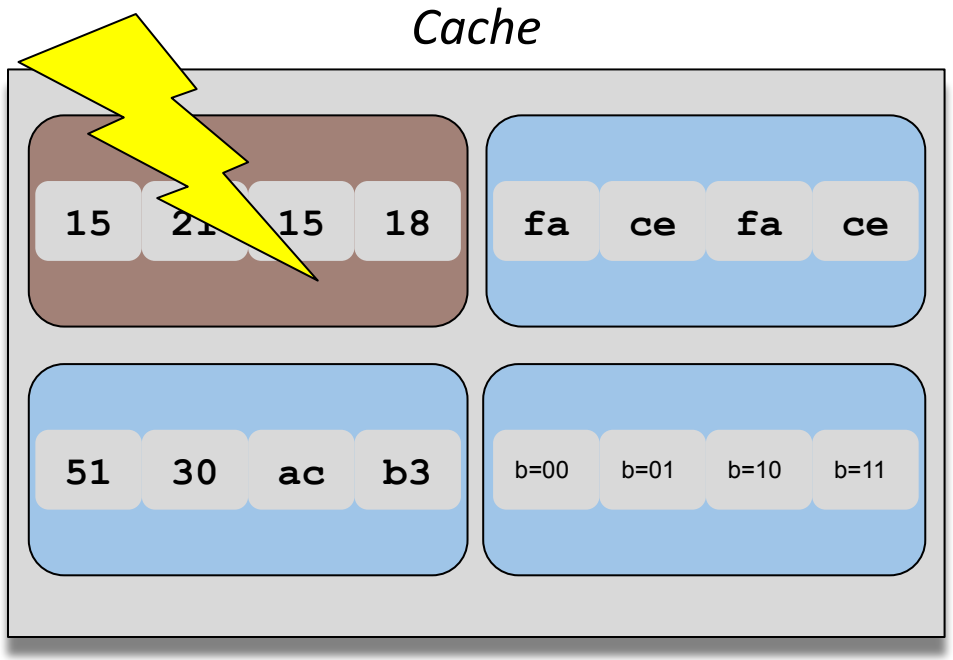
Example Trace

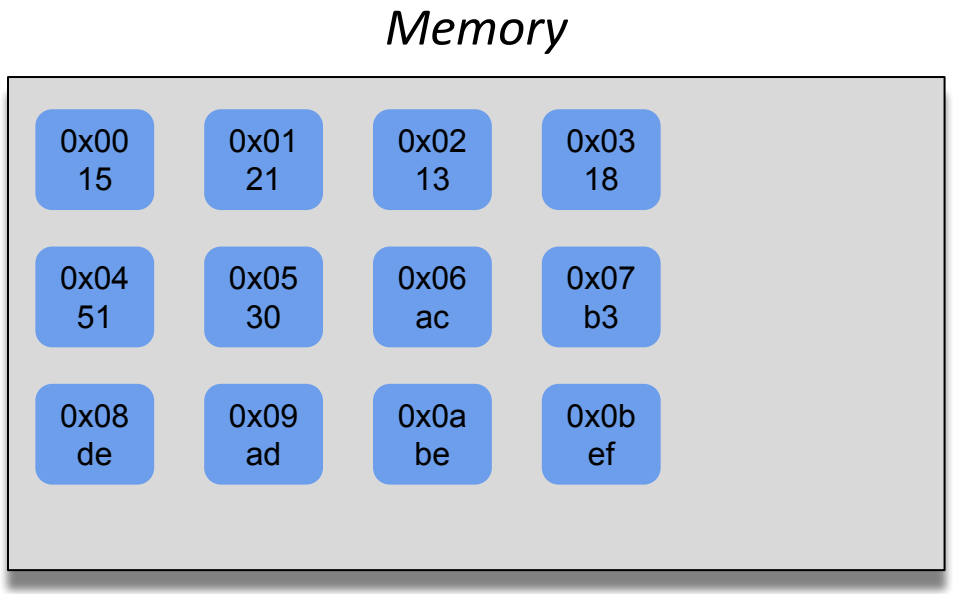
```
...
L 0,1
L 16,1
L 9,1
...
```

Hit!
Miss
Miss

Evict least recently used line

Dirty bit set => Dirty Eviction



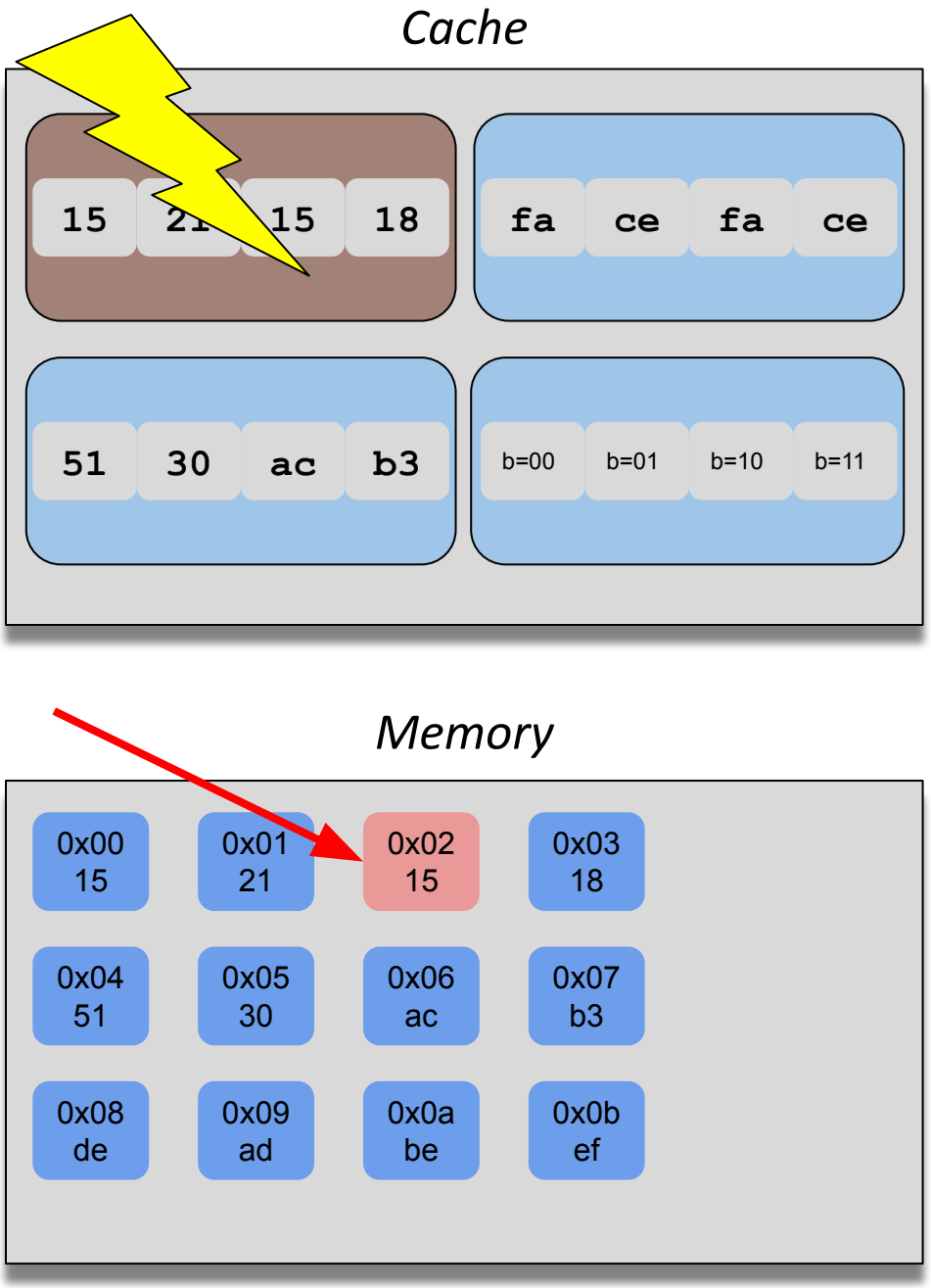


Example Trace

```
...
L 0,1
L 16,1
L 9,1
...
```

Hit!
Miss
Miss

Write-back policy!

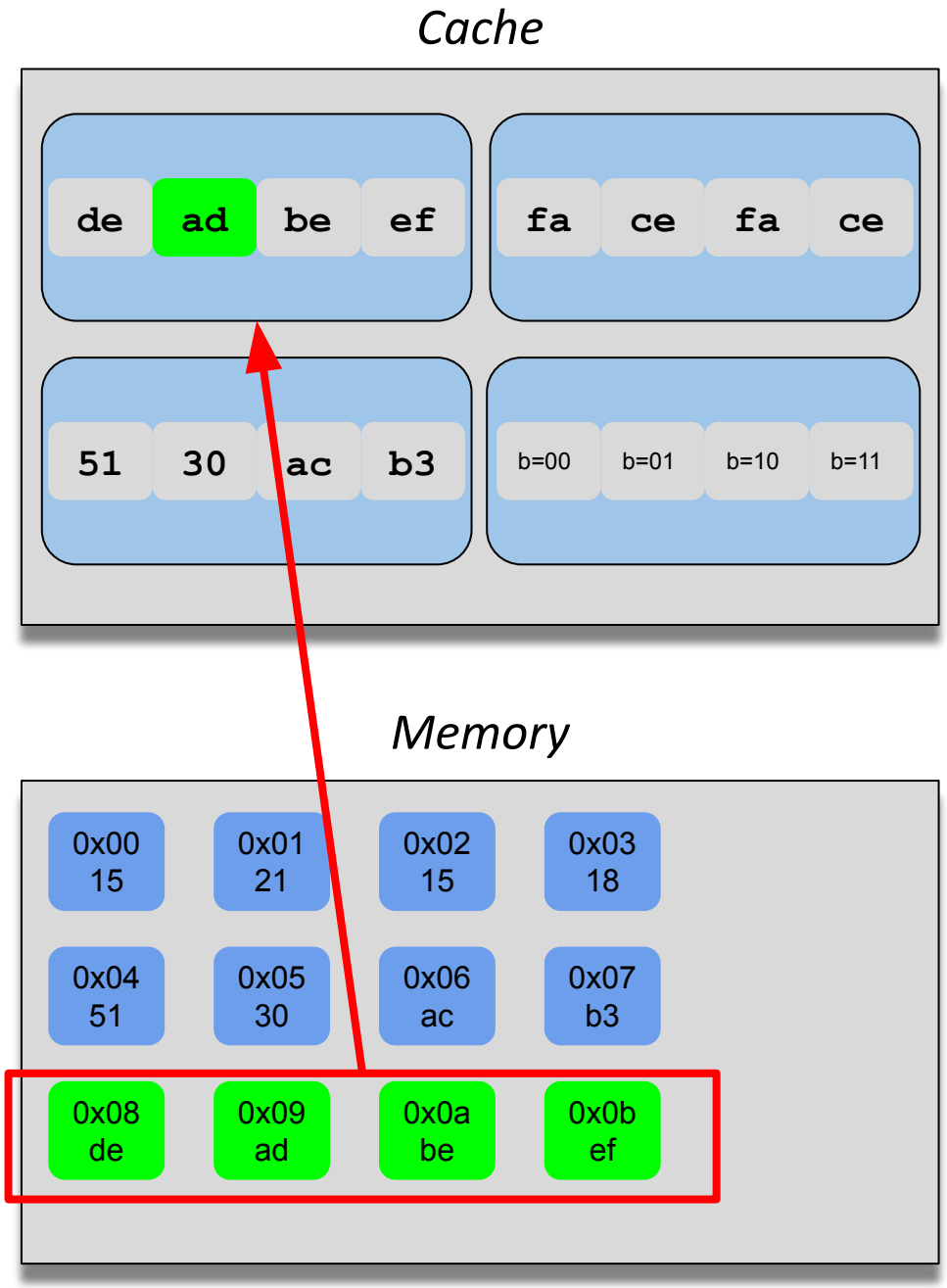


Example Trace

```
...
L 0,1
L 16,1
L 9,1
...
```

Hit!
Miss
Miss

Load new value into line

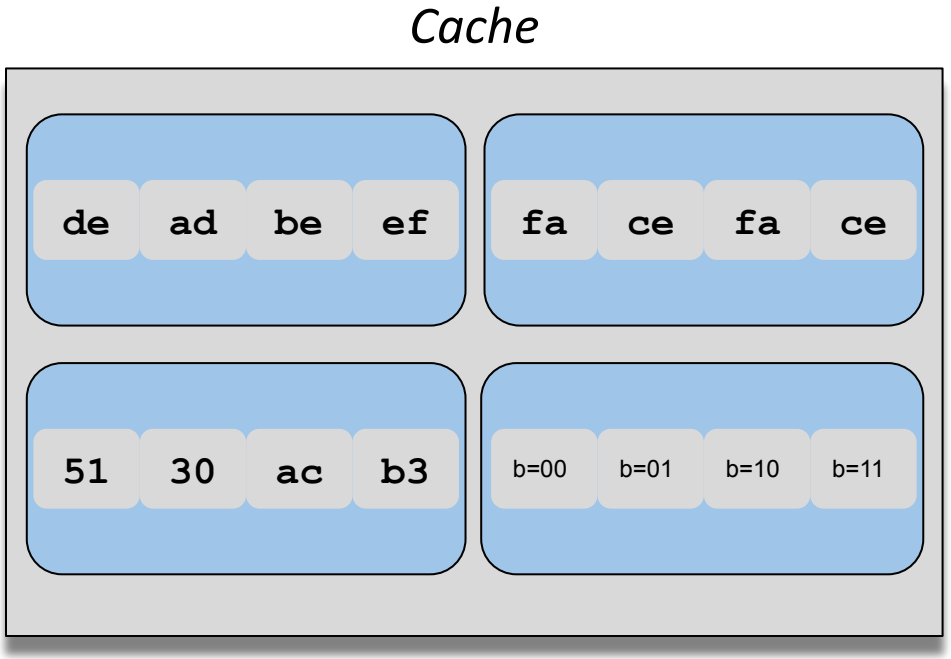


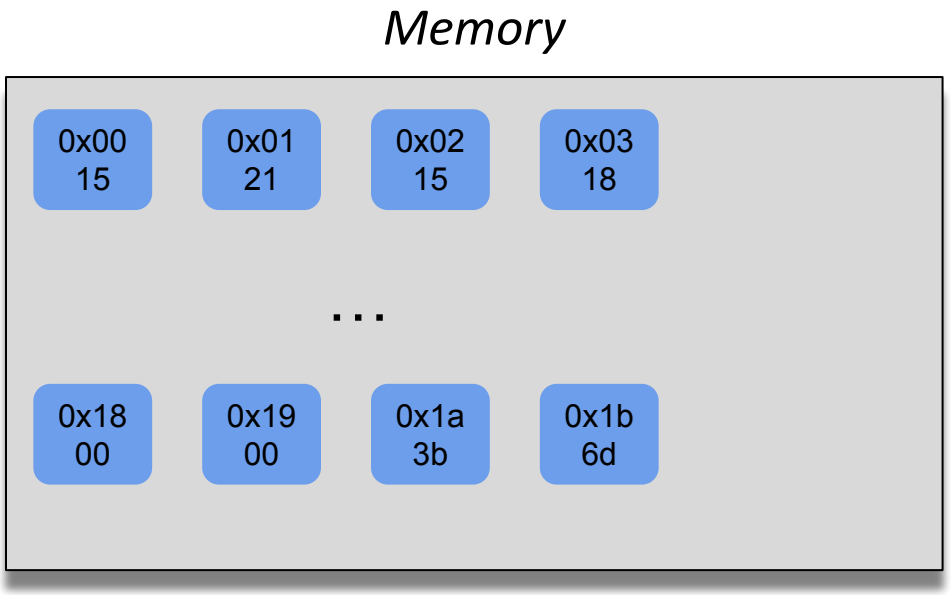
Example Trace

```
...
L 16,1
L 9,1
L 24,1
...
```

Miss
Miss
???

Will this instruction result in a hit or a miss?





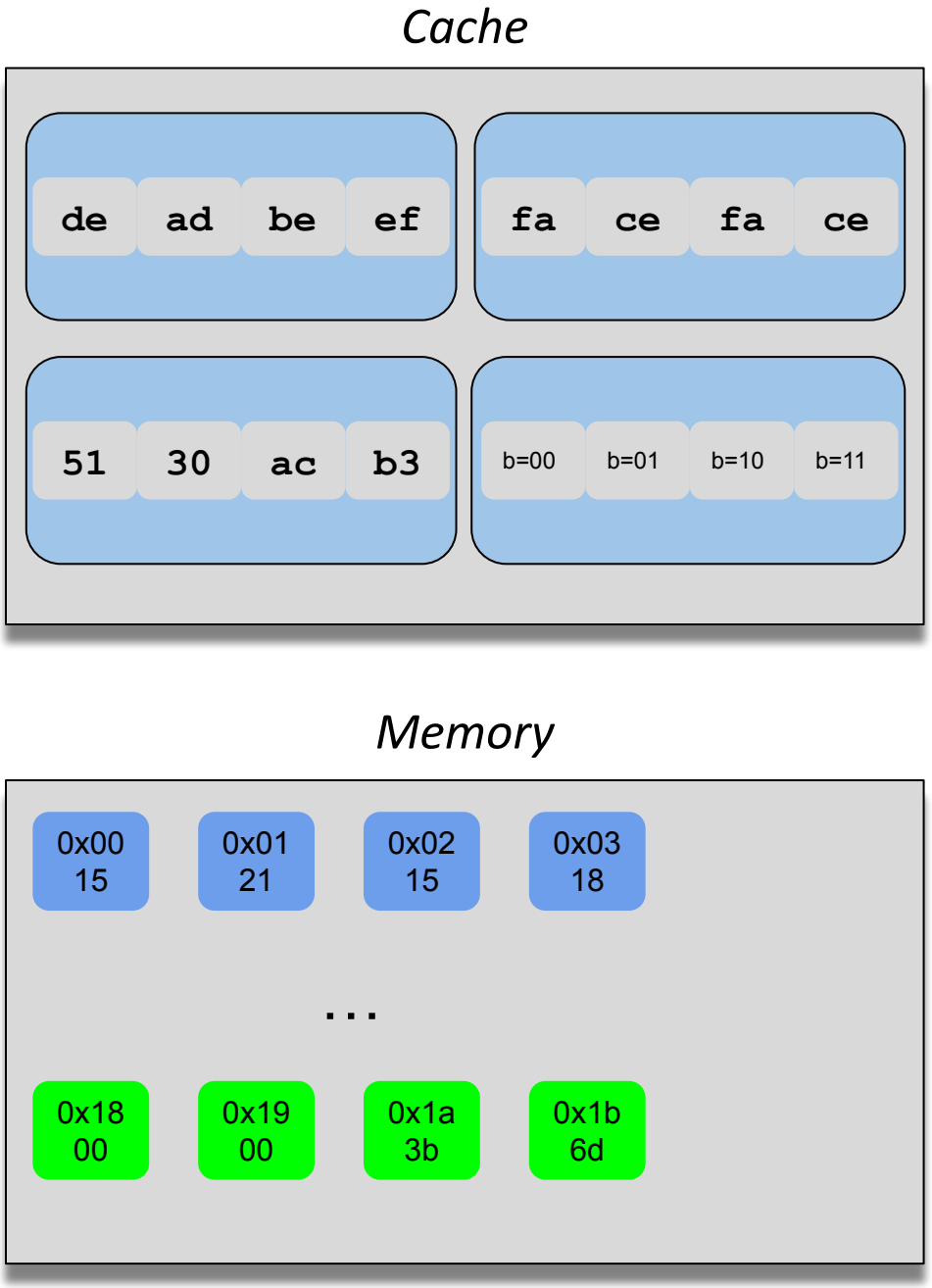
Example Trace

```
...
L 16,1
L 9,1
L 24,1
...
```

Miss
Miss
Miss

What type of miss is this?

Which line will get evicted?

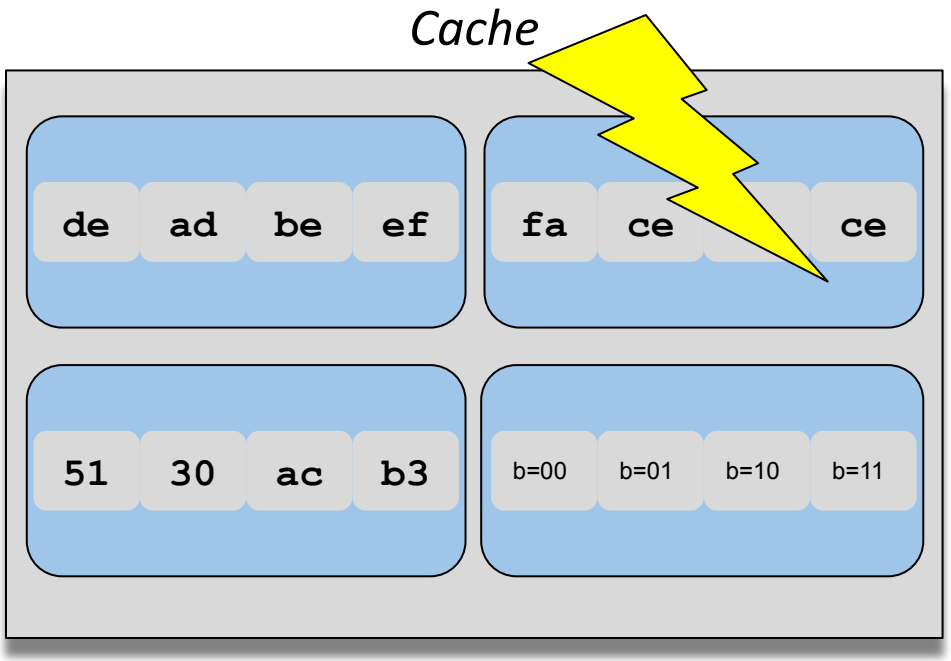


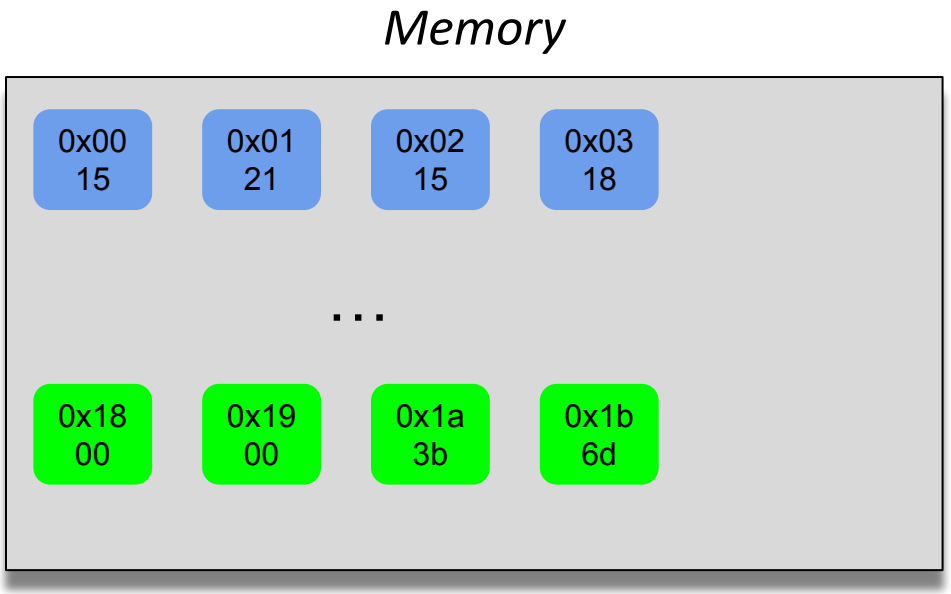
Example Trace

```
...
L 16,1
L 9,1
L 24,1
...
```

Miss
Miss
Miss

Evict least recently used line



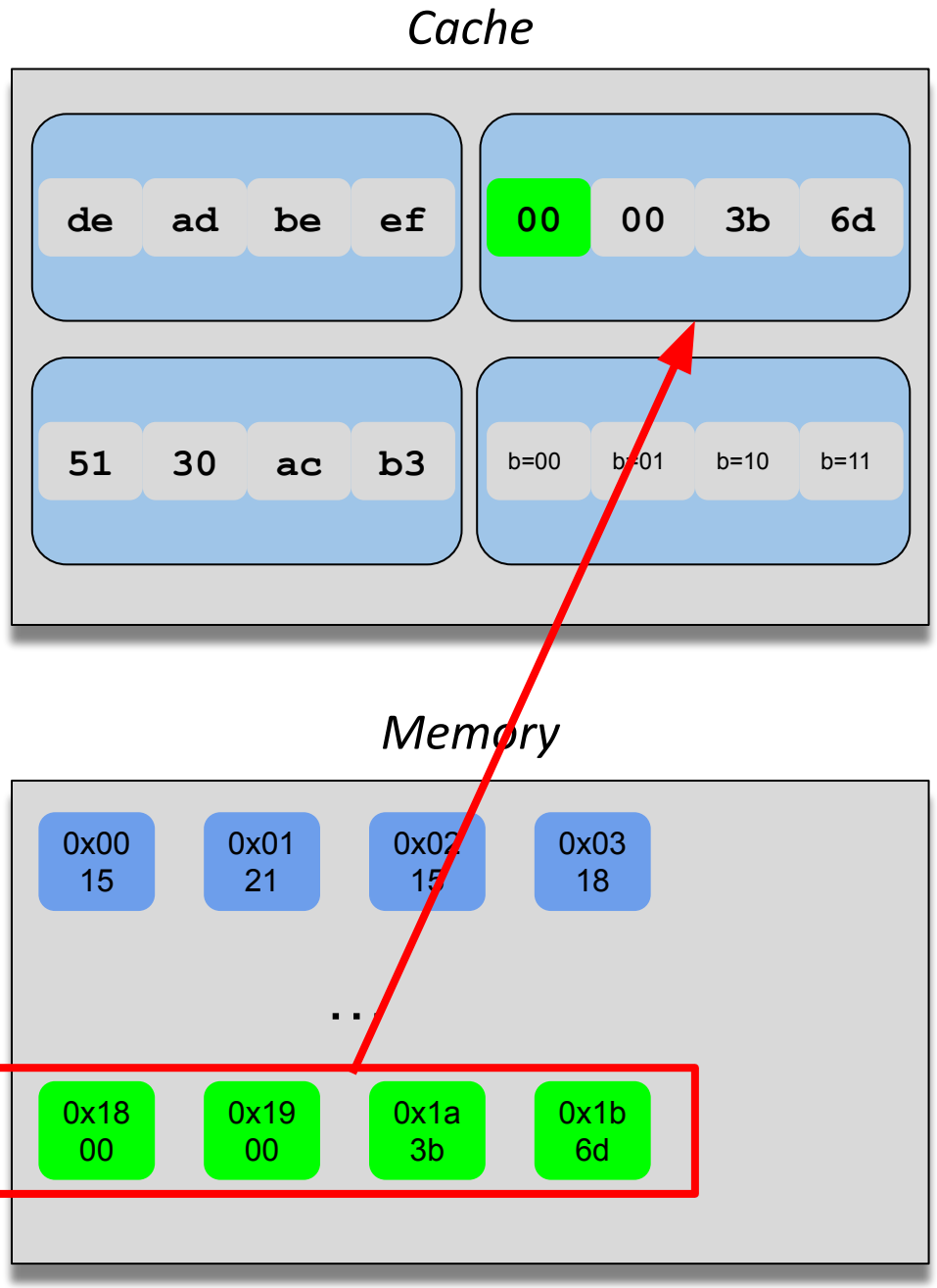


Example Trace

```
...
L 16,1
L 9,1
L 24,1
...
```

Miss
Miss
Miss

Load new value into line

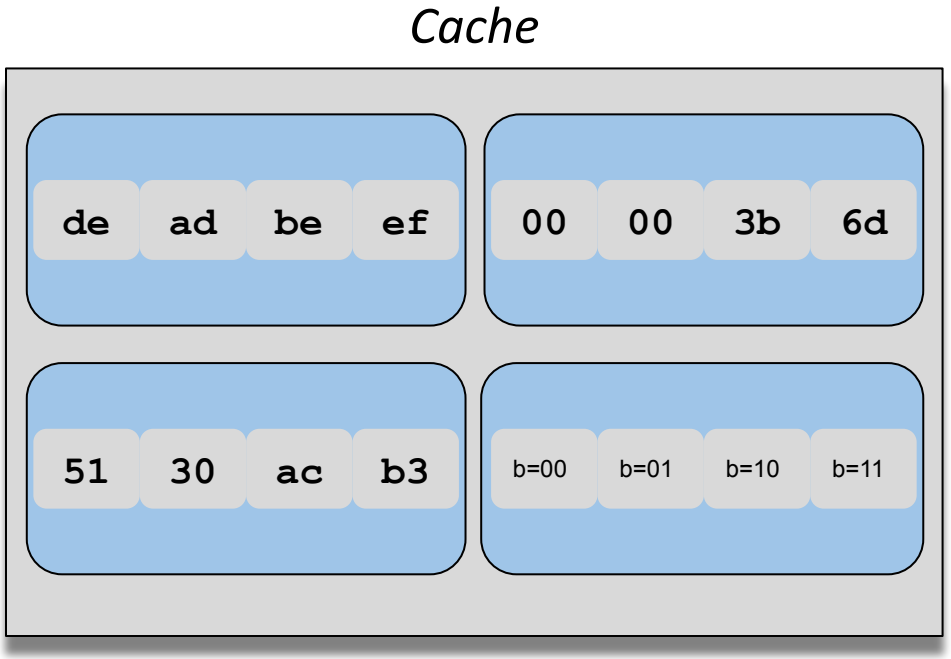


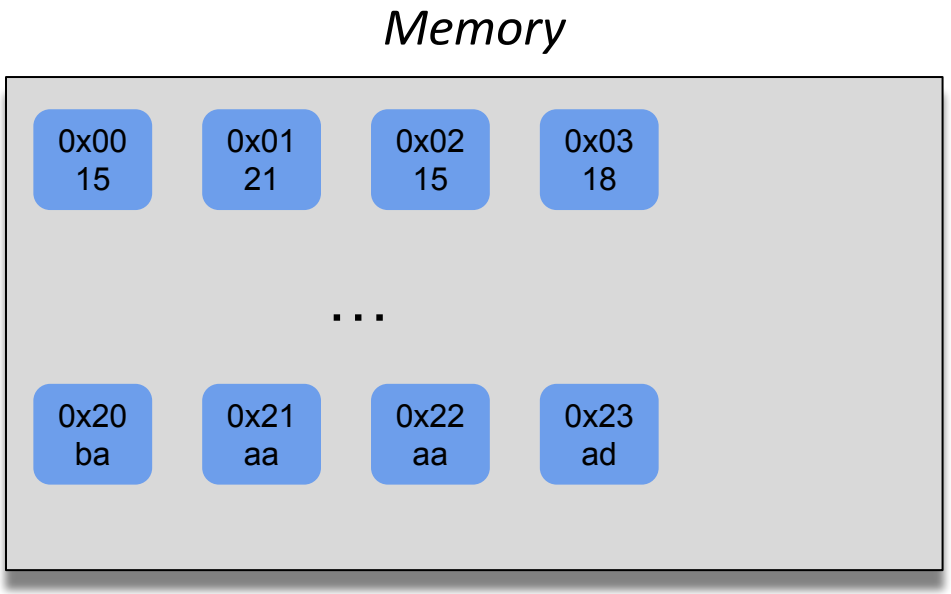
Example Trace

```
...
L 9,1
L 24,1
L 32,1
...
```

Miss
Miss
???

Will this instruction result in a hit or a miss?





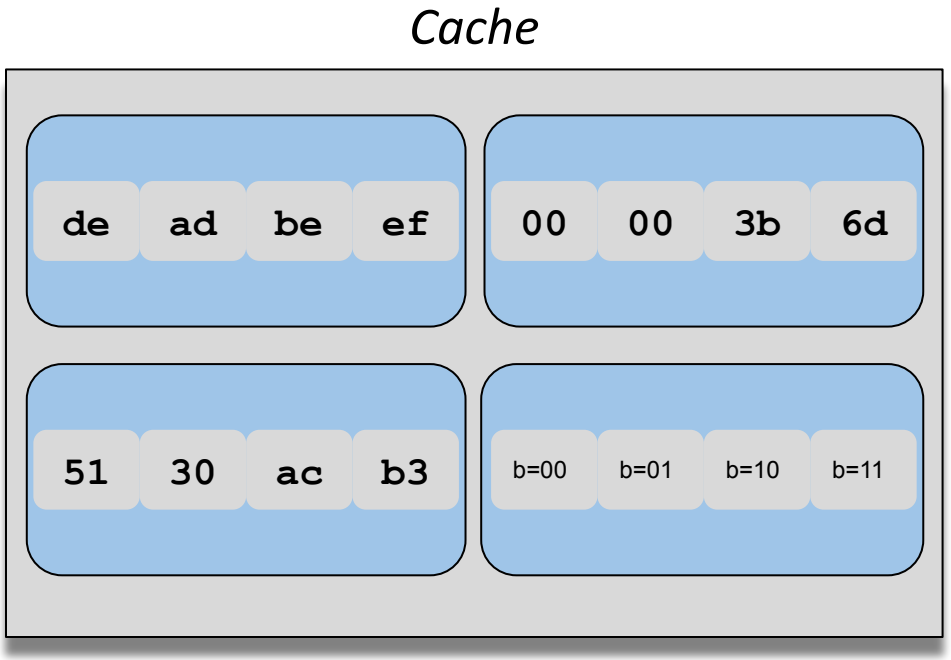
Example Trace

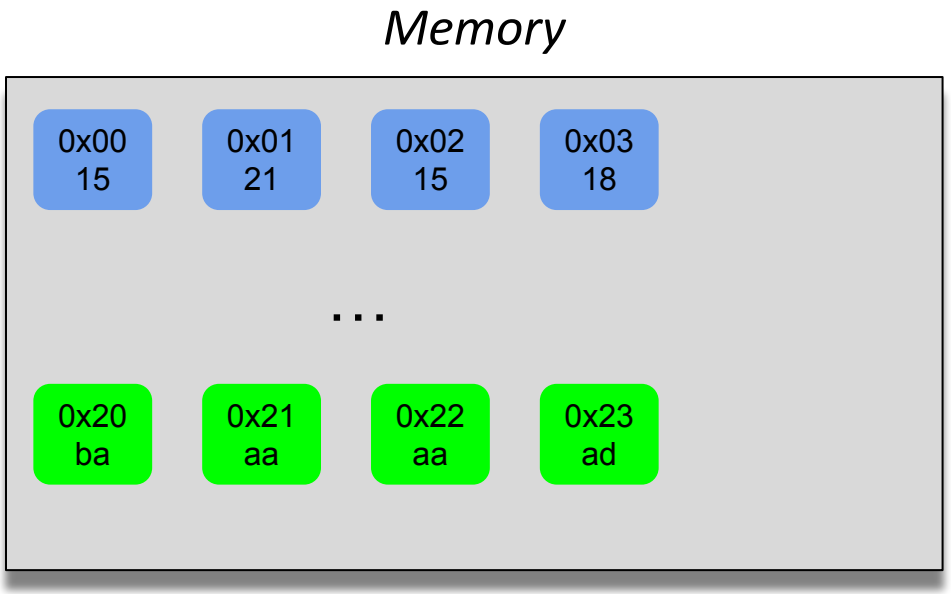
```
...
L 9,1
L 24,1
L 32,1
...
```

Miss
Miss
Miss

What type of miss is this?

Which line gets evicted?



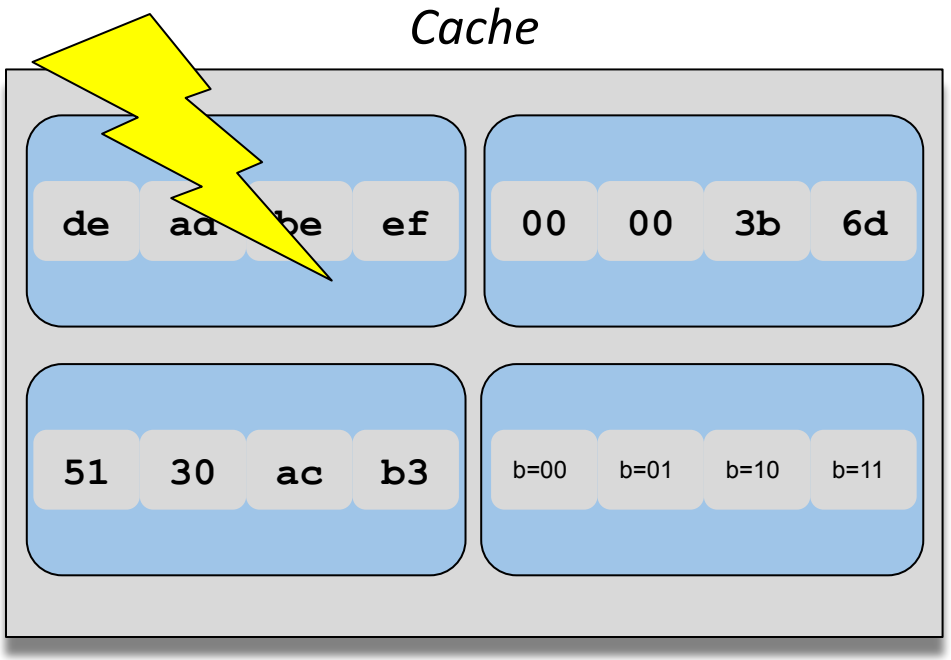


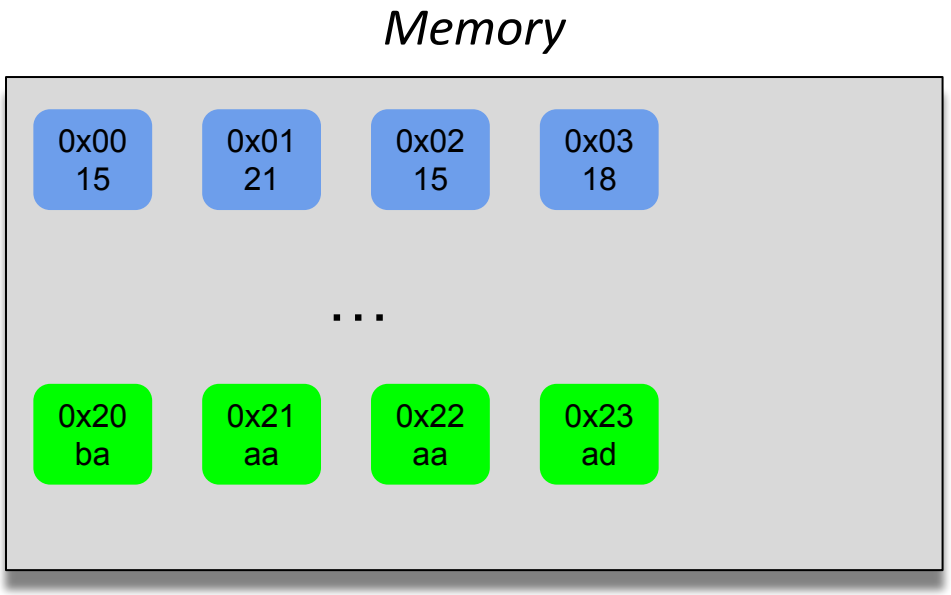
Example Trace

```
...
L 9,1
L 24,1
L 32,1
...
```

Miss
Miss
Miss

Evict least recently used line



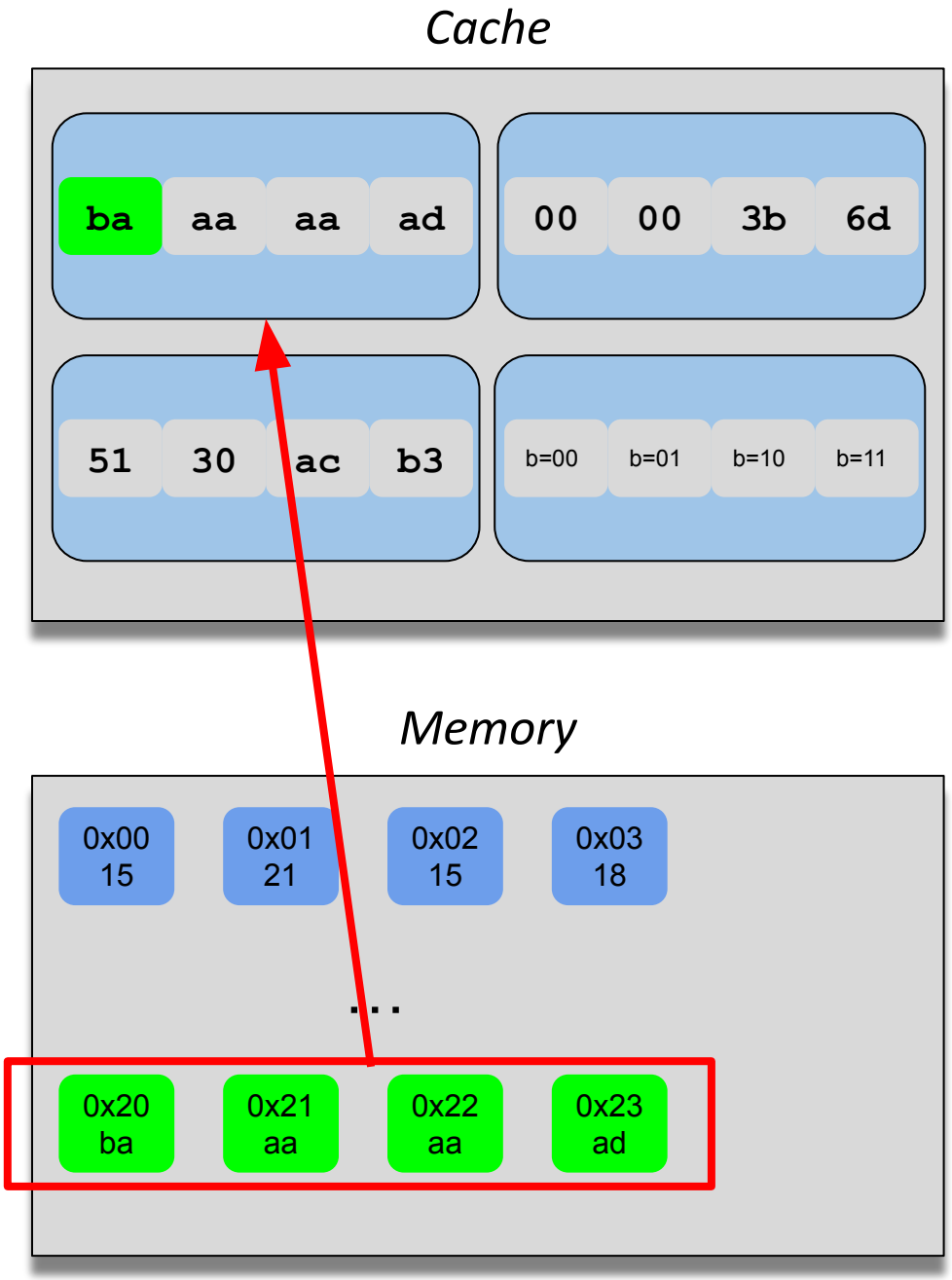


Example Trace

```
...
L 9,1
L 24,1
L 32,1
...
```

Miss
Miss
Miss

Load new value into line

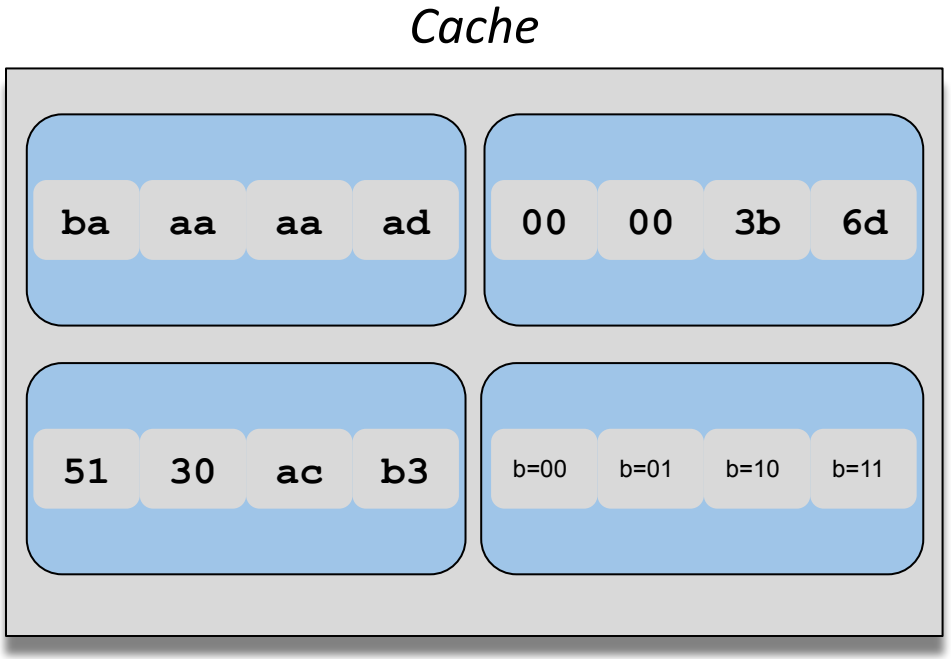


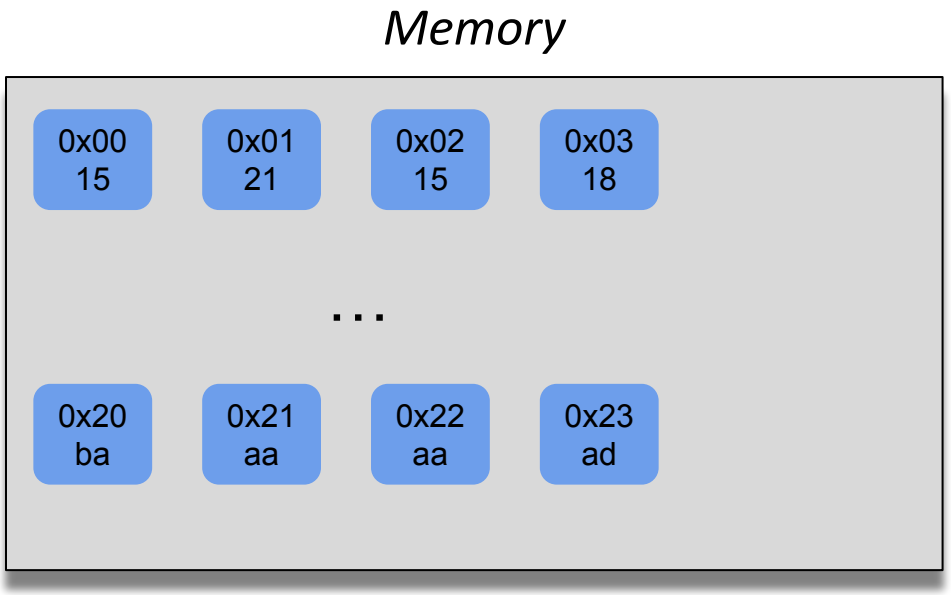
Example Trace

```
...
L 24,1
L 32,1
L 0,1
...
```

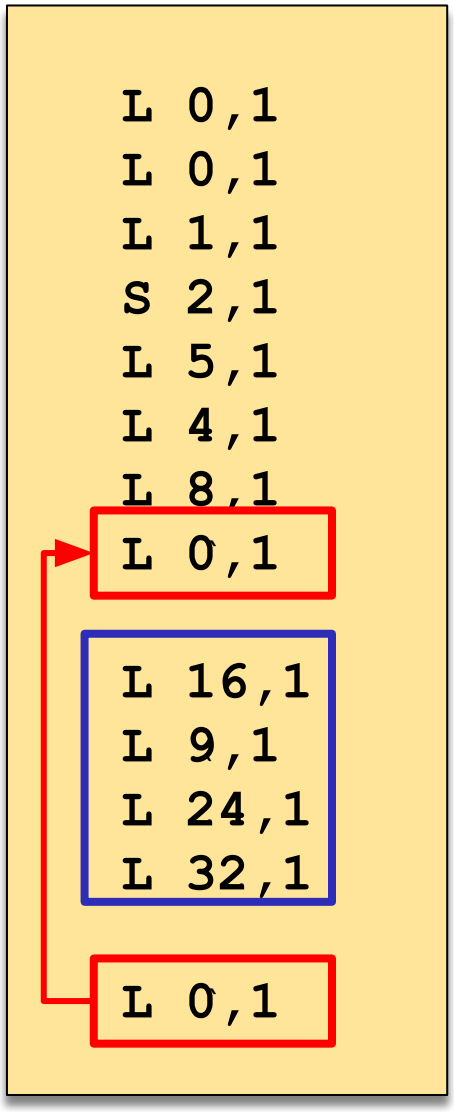
Miss
Miss
???

Will this instruction result in a hit or a miss?





Cache Concepts: Conflict/Capacity Misses

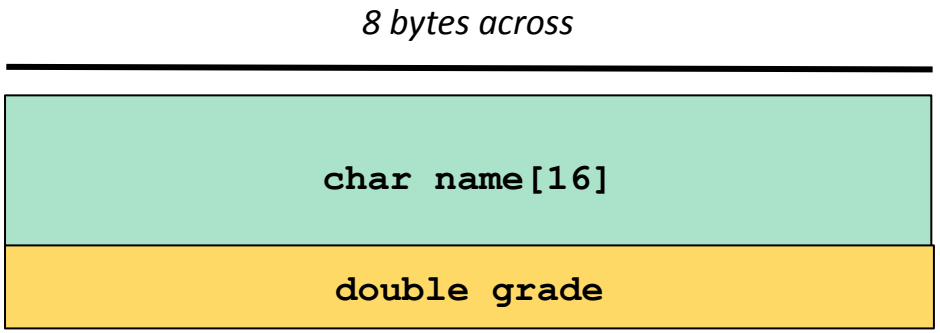


- In this case:
  - Number of unique blocks *in-between* current reference and most recent reference: 4
  - Our cache has 4 total lines
  - So: ***Capacity Miss***
- Note: the cache is **not** full!

Review: Programming in C

Programming in C: Structs

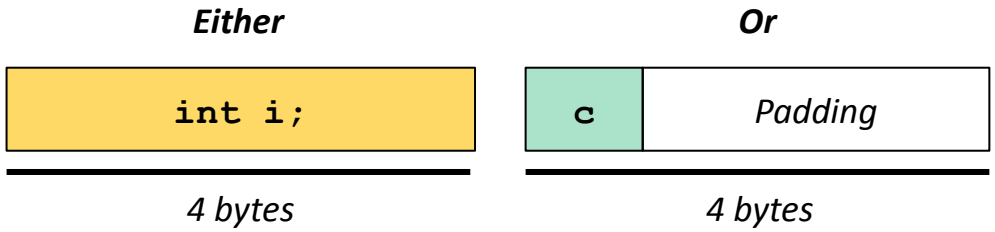
```
struct student {
   char name[16];
   double grade;
};
```



- Group multiple related fields under one block of memory, at one address.
- Will probably be useful for **`cachelab`**!

Programming in C: Unions

```
union temp {
    int i;
    char c;
};
```



- Store potentially different data types in the same region of memory.
- Specifies multiple ways to interpret data at the same memory location.

Programming in C: Style

- ***Code Reviews***: **`cachelab`** will be the first lab graded for style by your TAs.
  - Comments
  - File Header
  - Modularity
  - Correctness:
    - **`malloc()`** can fail! Library functions can fail!
    - Memory leaks, File Descriptor leaks
  - Note: We will only review your final submission on autolab, not your github code

Using `getopt()`

- Let us first learn what **`getopt()`** does.
- Read the man pages!
  - **`man 3 getopt`**
  - https://linux.die.net/man/3/getopt

Activity: `getopt_example.c`

Returns **-1** when done parsing!

```
while ((opt = getopt(argc, argv, "vn:")) != -1) {
    <-- Omitted -->
}
```

- Arguments are **-v** and **-n**
- Colon indicates option **-n** has required argument, which will get parsed into **optarg**.

Activity: `getopt_example.c`

```
while ((opt = getopt(argc, argv, "vn:")) != -1) {
    switch (opt) {
        case 'v':
            verbose = 1;
            break;
        case 'n':
            n = atoi(optarg);
            break;
        default:
            fprintf(stderr, "usage: …");
            exit(1);
    }
}
for (int i = 0; i < n; i++) {
    if (verbose) printf("%d\n", i);
}
printf("Done counting to %d\n", n);
```

Define the optstring

Count up to **-n** argument

If **-v** argument is set, print all numbers before n

Cache Practice Problems

Cache Practice Problems

- We'll work through a series of questions together.
- Write down your answer to each question.
- Discuss with classmates!

Cache Practice Problem: Locality

- The following function exhibits which type of locality?

  Consider ***only array accesses***.

```
void who(int *arr, int size) {
    for (int i = 0; i < size-1; ++i)
        arr[i] = arr[i+1];
}
```

A. Spatial

B. Temporal

C. Both spatial and temporal

D. Neither

Cache Practice Problem: Locality

- The following function exhibits which type of locality?

  Consider ***only array accesses***.

```
void who(int *arr, int size) {
    for (int i = 0; i < size-1; ++i)
        arr[i] = arr[i+1];
}
```

A. Spatial

B. Temporal

C. ***Both spatial and temporal***

D. Neither

- ***Spatial:*** **Items with nearby addresses tend to be referenced close together in time.**
- ***Temporal:*** **Recently accessed addresses tend to be accessed again in the near future.**

Cache Practice Problem: Locality

- The following function exhibits which type of locality?

  Consider ***only array accesses***.

```
void coo(int *arr, int size) {
    for (int i = size-2; i >= 0; --i)
        arr[i] = arr[i+1];
}
```

A. Spatial

B. Temporal

C. Both spatial and temporal

D. Neither

Cache Practice Problem: Locality

- The following function exhibits which type of locality?

  Consider ***only array accesses***.

```
void coo(int *arr, int size) {
    for (int i = size-2; i >= 0; --i)
        arr[i] = arr[i+1];
}
```

A. Spatial

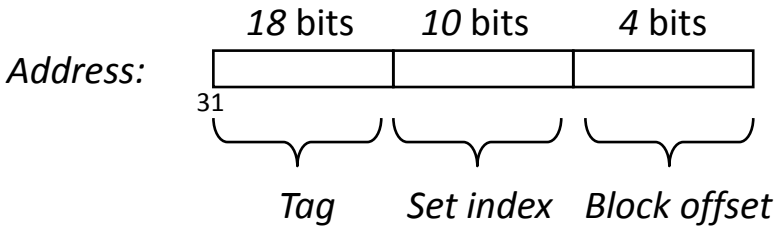
B. Temporal

C. Both spatial and temporal

D. Neither

Cache Practice Problem: Cache Parameters

- Given the following address partition, how many int values fit in each block?



A. 0

B. 1

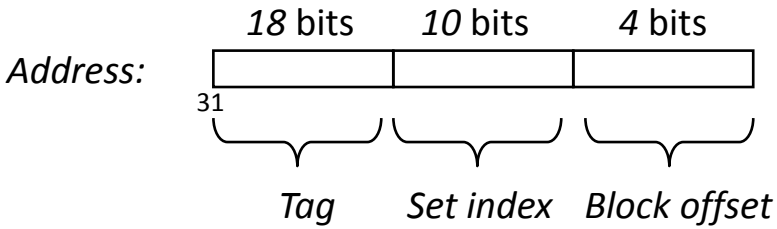
C. 2

D. 4

E. Not enough information to determine

Cache Practice Problem: Cache Parameters

- Given the following address partition, how many int values fit in each block?



A. 0

B. 1

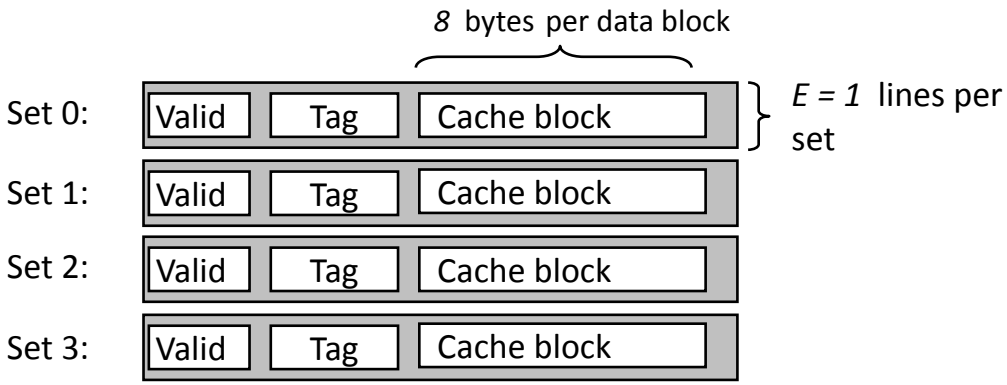
C. 2

D. 4

E. Not enough information to determine

- (**b** = 4) Four Block Offset Bits
- So block size is 2^4 = 16 bytes
- Integers are 4 bytes
- So we can fit four integers in each block.

Cache Practice Problem: Cache Parameters

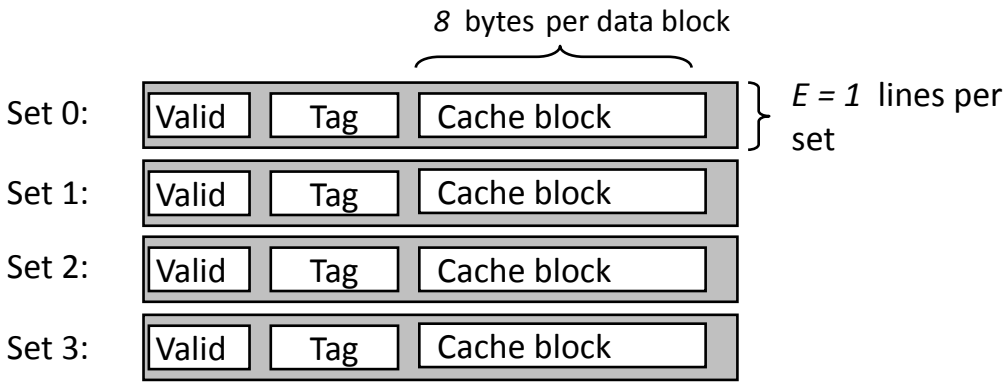


- What are the parameters corresponding to this cache organization?

  (assume a 32 bit address space)

Option	t (# Tag Bits)	s	b
A	1	2	3
B	27	2	3
C	25	4	3
D	1	4	8
E	20	4	8

Cache Practice Problem: Cache Parameters

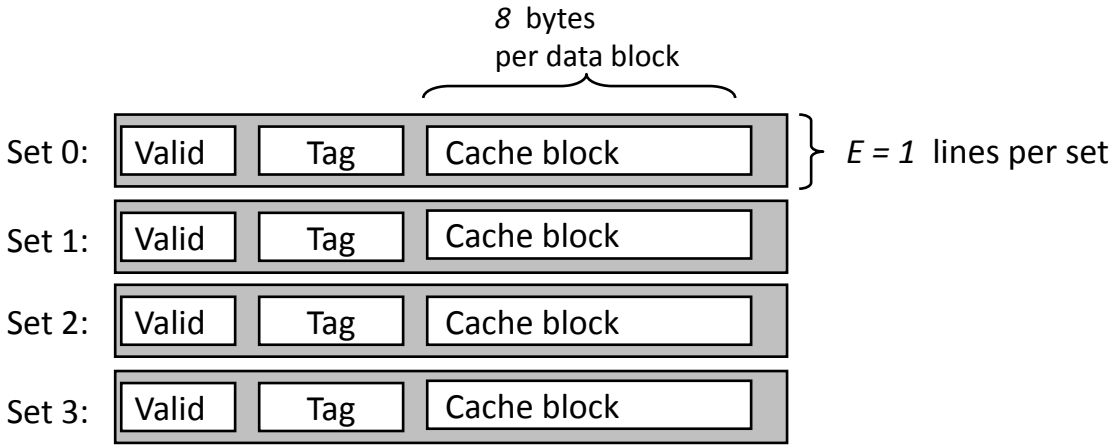


- What are the parameters corresponding to this cache organization?

  (assume a 32 bit address space)

Option	t (# Tag Bits)	s	b
A	1	2	3
B	27	2	3
C	25	4	3
D	1	4	8
E	20	4	8

Cache Practice Problem: Which Set?



Which *set* does the address **`0xfa1c`** map to?

A. 0

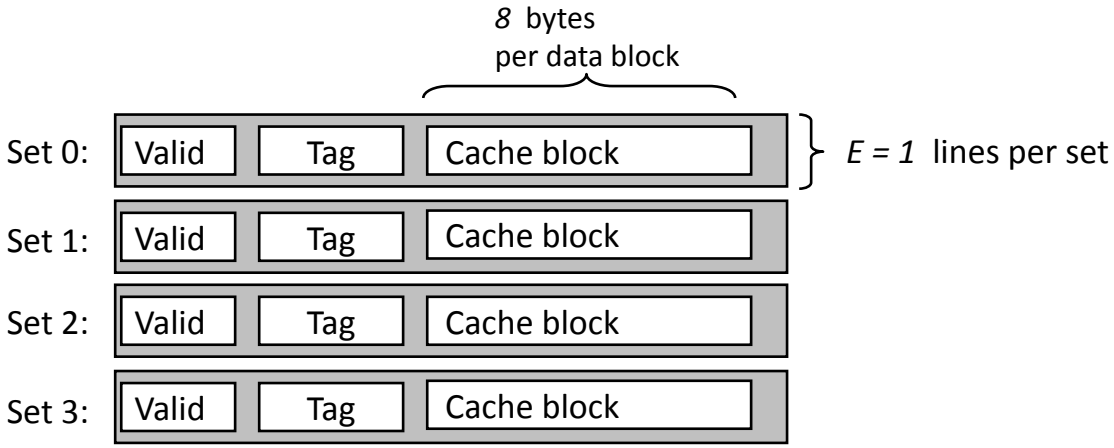
B. 1

C. 2

D. 3

E. None of the above

Cache Practice Problem: Which Set?



Which *set* does the address **`0xfa1c`** map to?

A. 0

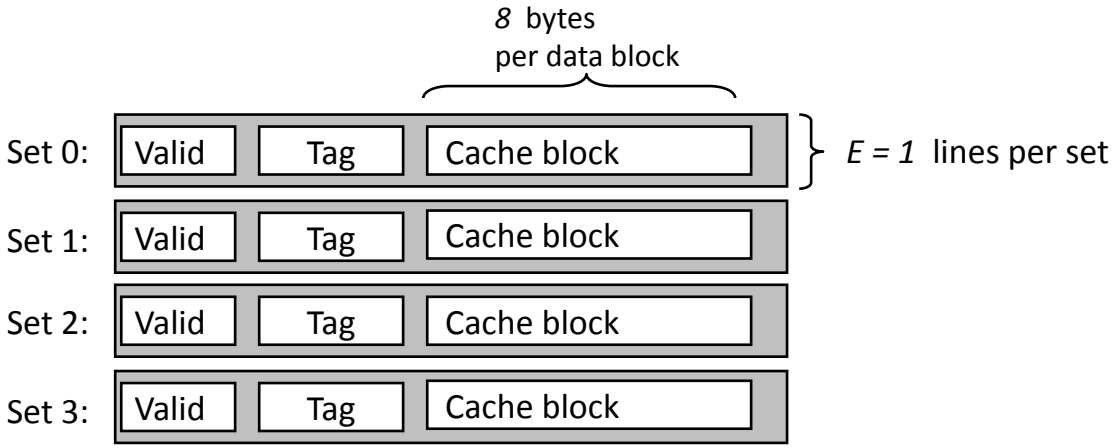
B. 1

C. 2

D. 3

E. None of the above

Cache Practice Problem: Range



Which range of addresses will be in the same block as **`0xfa1c`**?

A. **`0xfa1c`**

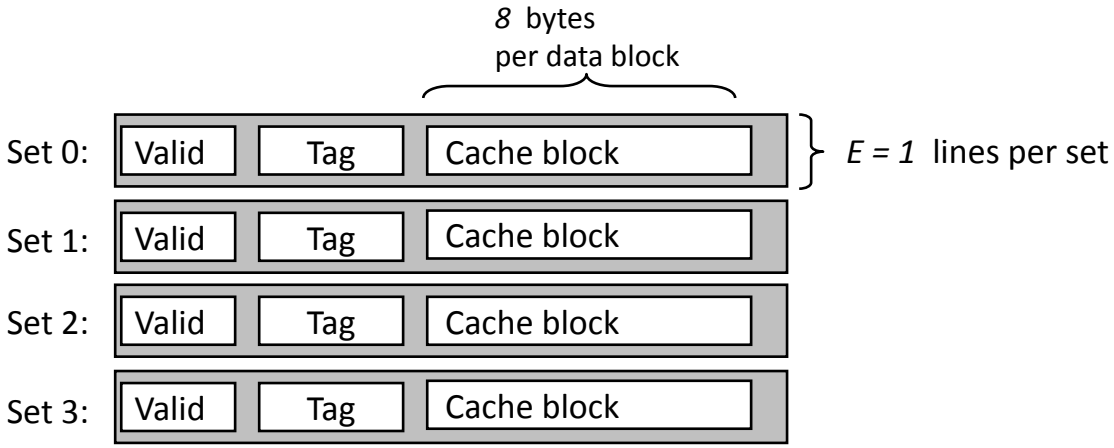
B. **`0xfa1c-0xfa23`**

C. **`0xfa1c-0xfa1f`**

D. **`0xfa18-0xfa1f`**

E. It depends on the access size

Cache Practice Problem: Range



Which range of addresses will be in the same block as **`0xfa1c`**?

A. **`0xfa1c`**

B. **`0xfa1c-0xfa23`**

C. **`0xfa1c-0xfa1f`**

D. ***`0xfa18-0xfa1f`***

E. It depends on the access size

Cache Practice Problem

- If **N** = 16, how many bytes of **a** does the loop access?

```
int foo(int* a, int N)
{
    int i;
    int sum = 0;
    for(i = 0; i < N; i++)
    {
        sum += a[i];
    }
    return sum;
}
```

A. 4

B. 16

C. 64

D. 256

Cache Practice Problem

- If **N** = 16, how many bytes does the loop access of **a**?

```
int foo(int* a, int N)
{
    int i;
    int sum = 0;
    for(i = 0; i < N; i++)
    {
        sum += a[i];
    }
    return sum;
}
```

A. 4

B. 16

C. 64

D. 256

Cache Practice Problem: Cache Misses

```
void muchAccessSoCacheWow(int *bigArr){
    // 48 KB array of ints
    int length = (48*1024)/sizeof(int);
    int access = 0;

    // traverse array with stride 8
    // pass 1
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }

    // pass 2
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }
}
```

- 32 KB cache
- 32 bit address space.
- 8-way associative
- 64 bytes per block.
- LRU

What is the miss rate on “Pass 1”?

A. 0%
B. 25%
C. 33%
D. 50%
E. 66%

Cache Practice Problem: Cache Misses

```
void muchAccessSoCacheWow(int *bigArr){
    // 48 KB array of ints
    int length = (48*1024)/sizeof(int);
    int access = 0;

    // traverse array with stride 8
    // pass 1
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }

    // pass 2
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }
}
```

- 32 KB cache
- 32 bit address space.
- 8-way associative
- 64 bytes per block.
- LRU

What is the miss rate on “Pass 1”?

A. 0%
B. 25%
C. 33%
D. 50%
E. 66%

Cache Practice Problem: Cache Misses

```
void muchAccessSoCacheWow(int *bigArr){
    // 48 KB array of ints
    int length = (48*1024)/sizeof(int);
    int access = 0;

    // traverse array with stride 8
    // pass 1
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }

    // pass 2
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }
}
```

- 32 KB cache
- 32 bit address space.
- 8-way associative
- 64 bytes per block.
- LRU

What is the miss rate on “Pass 2”?

A. 0%
B. 25%
C. 33%
D. 50%
E. 66%

Cache Practice Problem: Cache Misses

```
void muchAccessSoCacheWow(int *bigArr){
    // 48 KB array of ints
    int length = (48*1024)/sizeof(int);
    int access = 0;

    // traverse array with stride 8
    // pass 1
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }

    // pass 2
    for(int i = 0; i < length; i+=8){
        access = bigArr[i];
    }
}
```

- 32 KB cache
- 32 bit address space.
- 8-way associative
- 64 bytes per block.
- LRU

What is the miss rate on “Pass 2”?

A. 0%
B. 25%
C. 33%
D. 50%
E. 66%

Wrapping Up

- **`cachelab`** tips:
  - Review Lectures
  - Start early! This lab can be challenging!
  - Don’t get discouraged!
- C Programming Review materials:
  - Ed: #98
  - Keep an eye Ed for *Bootcamp 4: C Programming*.

The End!