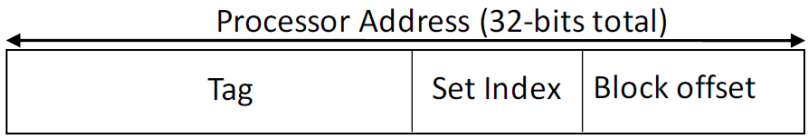
Caches

Relationships between 3 mappings

Direct Mapped

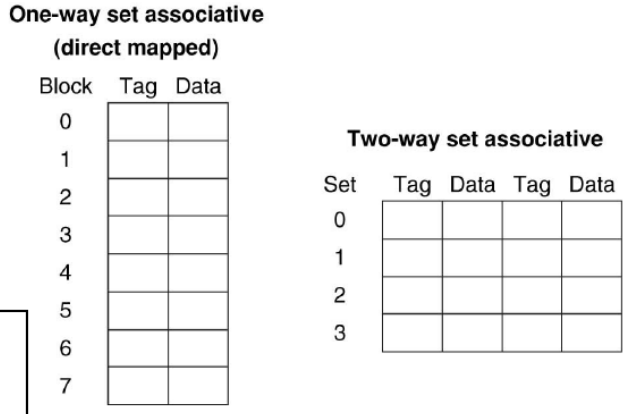
Set Associative

Fully Associative: remove set index

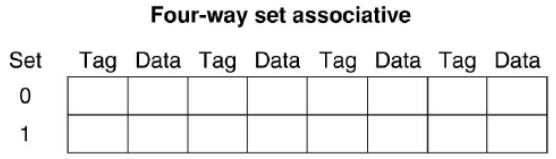


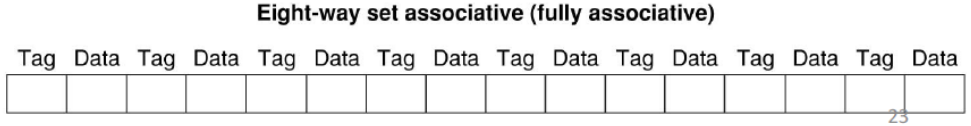
Same format of address:
If each set maps to N numbers, then:
Direct Mapped: a+log(N)+c
Set Associative: a+n_w+(log(N)-n_w)+c
Fully Associative: remove set index

Different Organizations of an Eight-Block Cache



Total size of $ in blocks is equal to *number of sets* × *associativity*. For fixed $ size and fixed block size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative $ is same as a fully associative $.

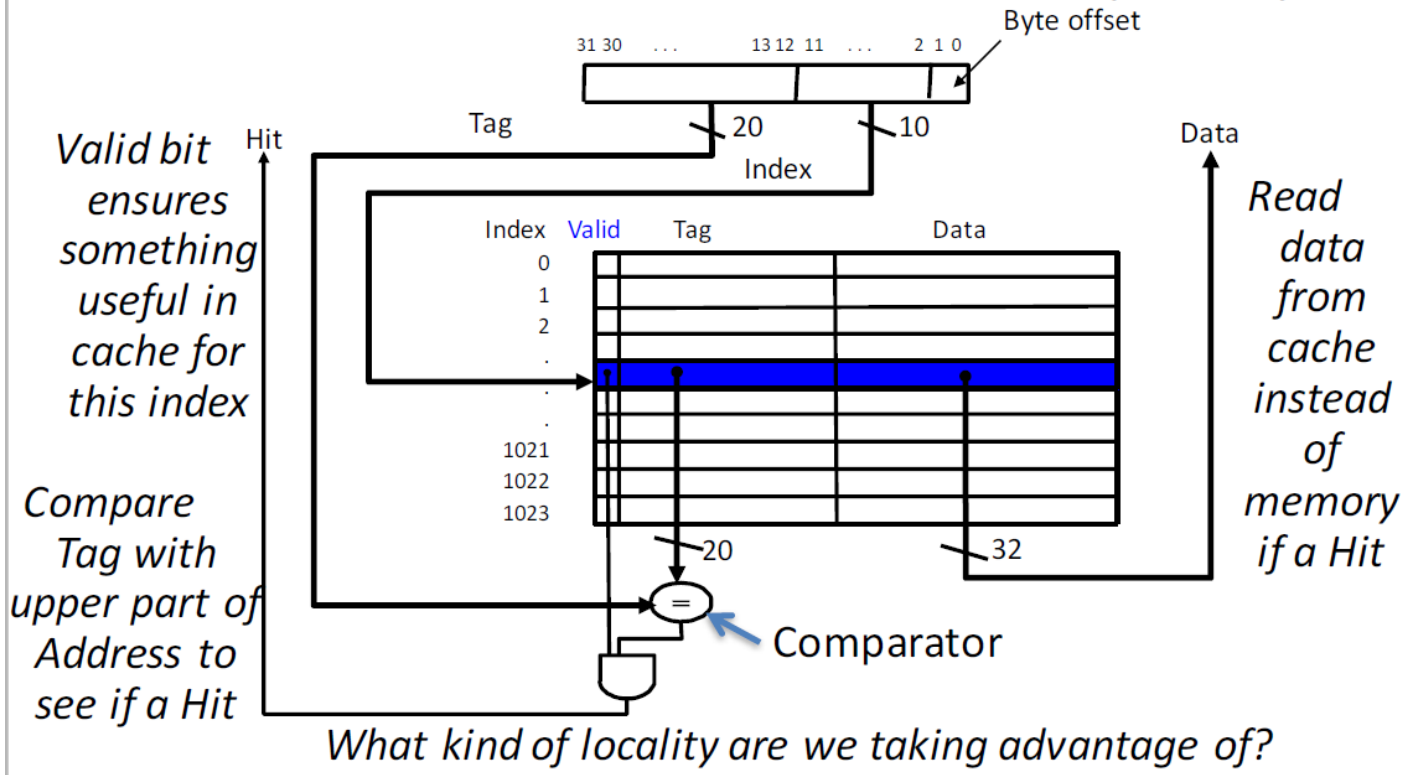




Direct Mapped Cache

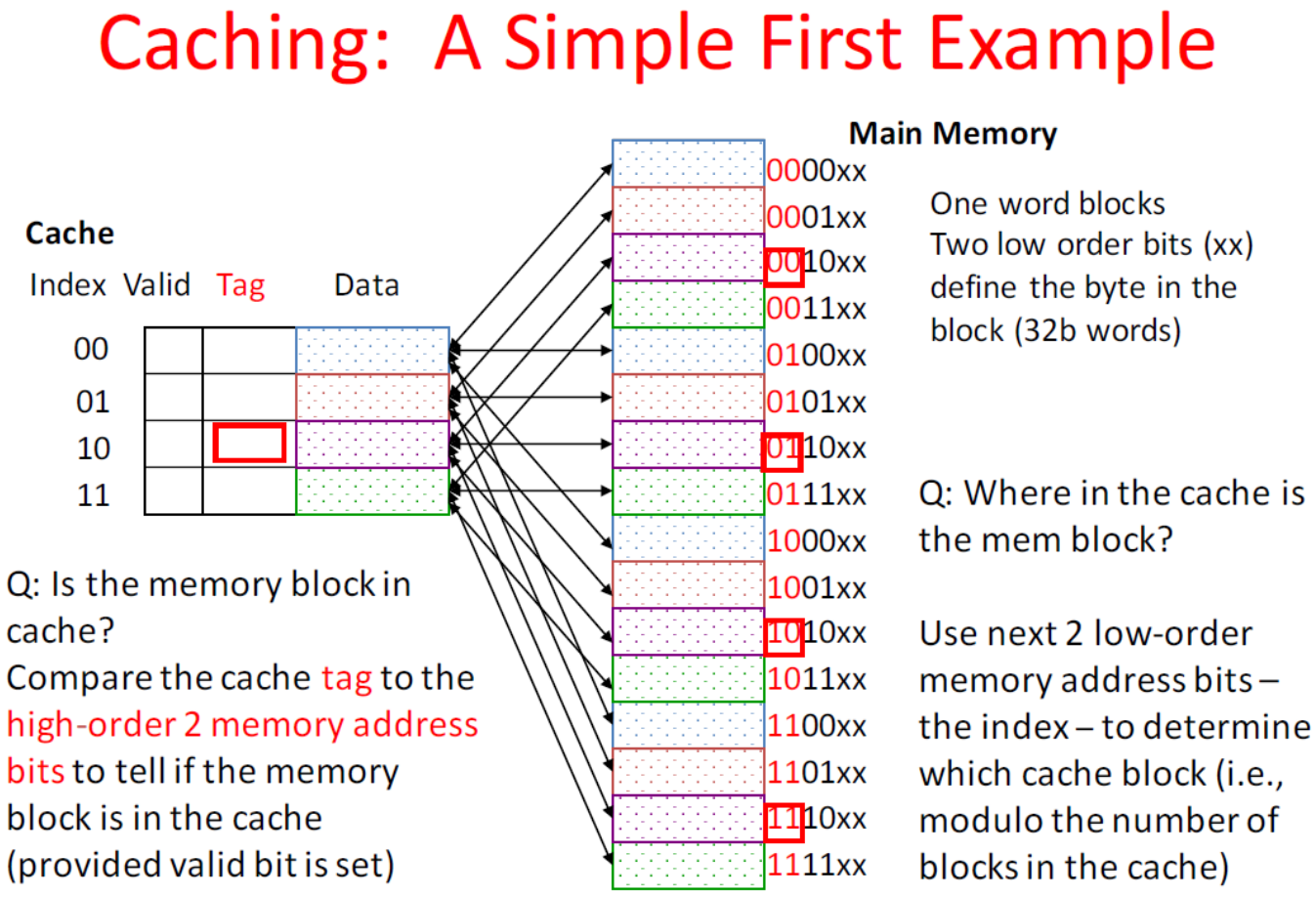
- Only one comparator is enough – each memory block is mapped to only 1 index in cache
- Number of index bits determined by cache size and block size
- Index_num = cache_size / 2^(byte_offset) (in Byte)

- One word blocks, cache size = 1K words (or 4KB)

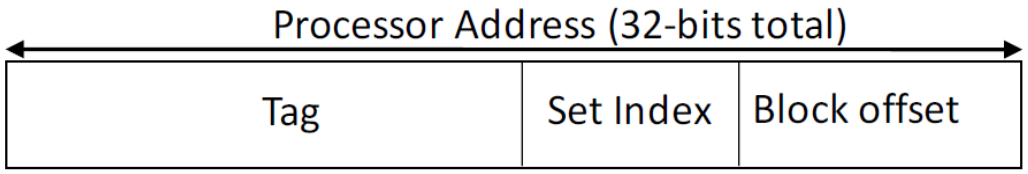


Direct Mapped Cache

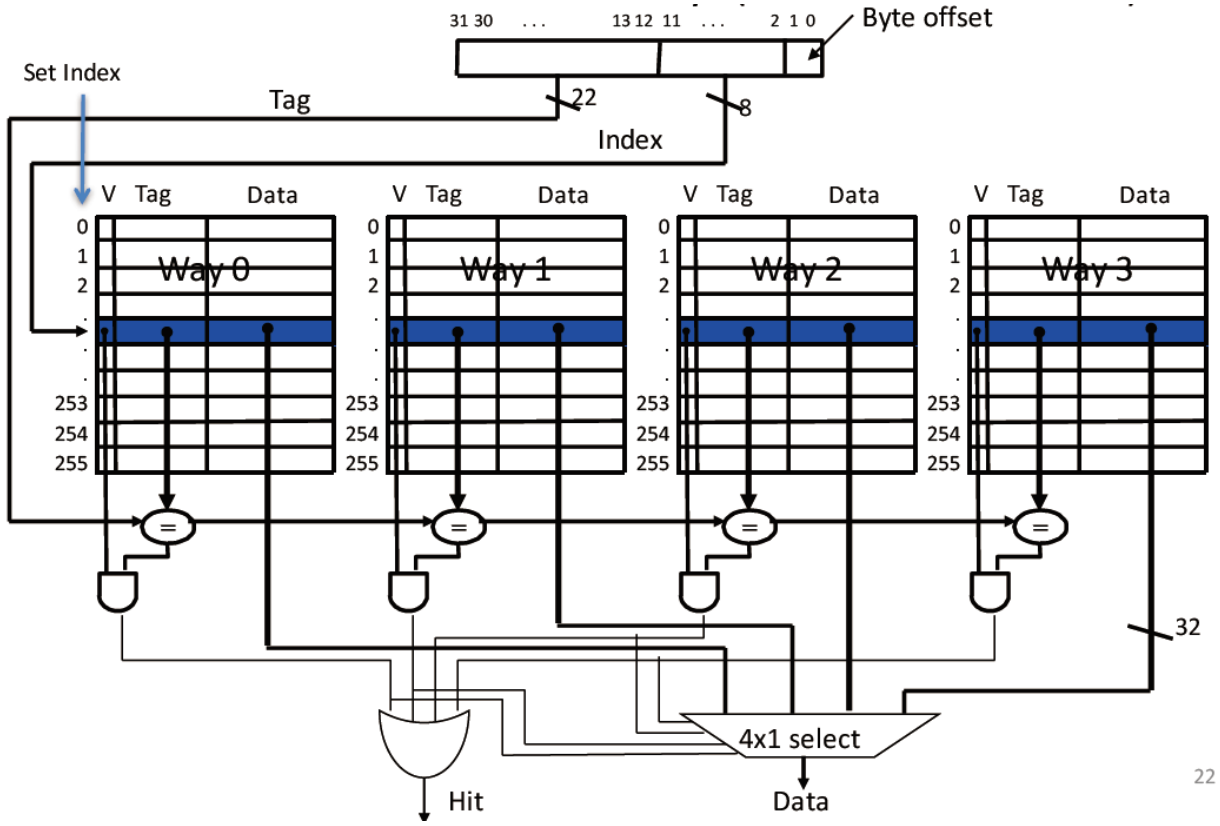
- A 16B cache
- Memory blocks with the save index could be stored in the same data address of a cache
- Compare Tag(the next 2 low-order bits) to judge if the memory block ins in cache
- If in, add byte offset



Set-Associative Caches



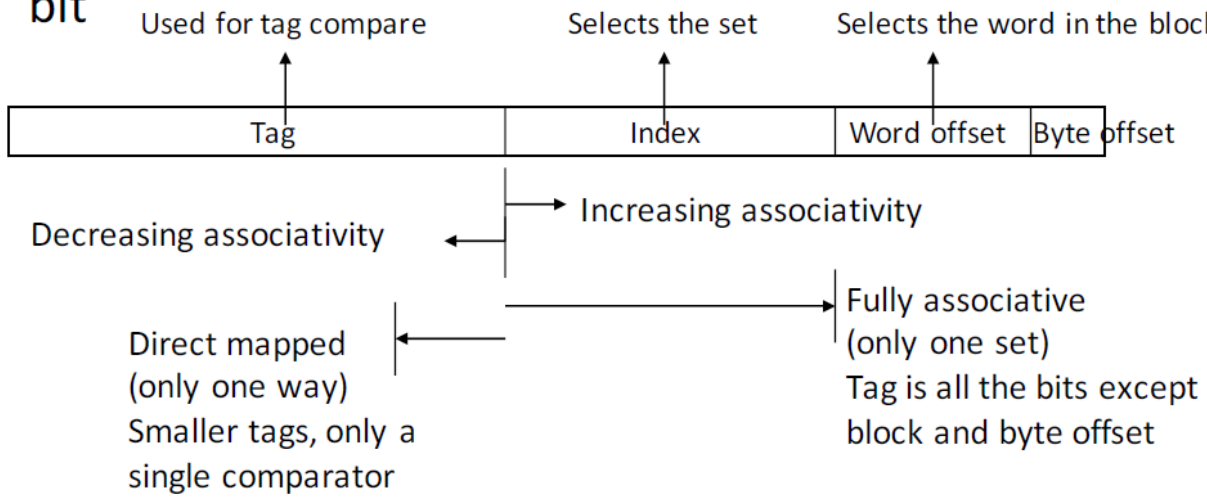
- A mixture of Fully Associative and Direct Mapped
  - FA: looks up every tag
  - DM: compare with only 1 tag
  - SA: looks up N ways
- Tag_width + index_width + offset_width = const
- If one is changed, we can change another to maintain the cache size.



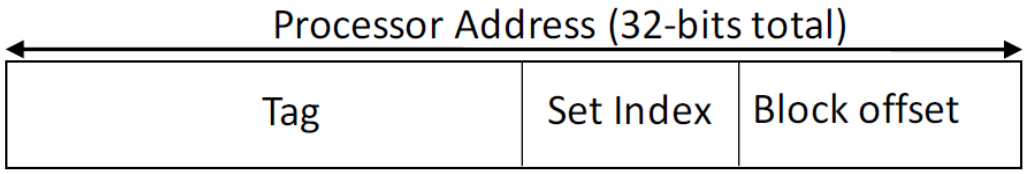
Set-Associative Caches

Range of Set-Associative Caches

- For a *fixed-size* cache and fixed block size, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit



Set-Associative Caches



- For a cache with constant total capacity, if we increase the number of ways by a factor of 2, which statement is false:
- A: The number of sets could be doubled
- B: The tag width could decrease
- C: The block size could stay the same
- D: The block size could be halved
- E: Tag width must increase

$2^i 2^b 2^w$ =const→ $i + b + w$ =const
Tag width must increase by 1.

- 1 more index bit
- A: true if we divide block size by 4
- B: False.
- C: byte offset not changed
- D: b_width-1
- E: Correct

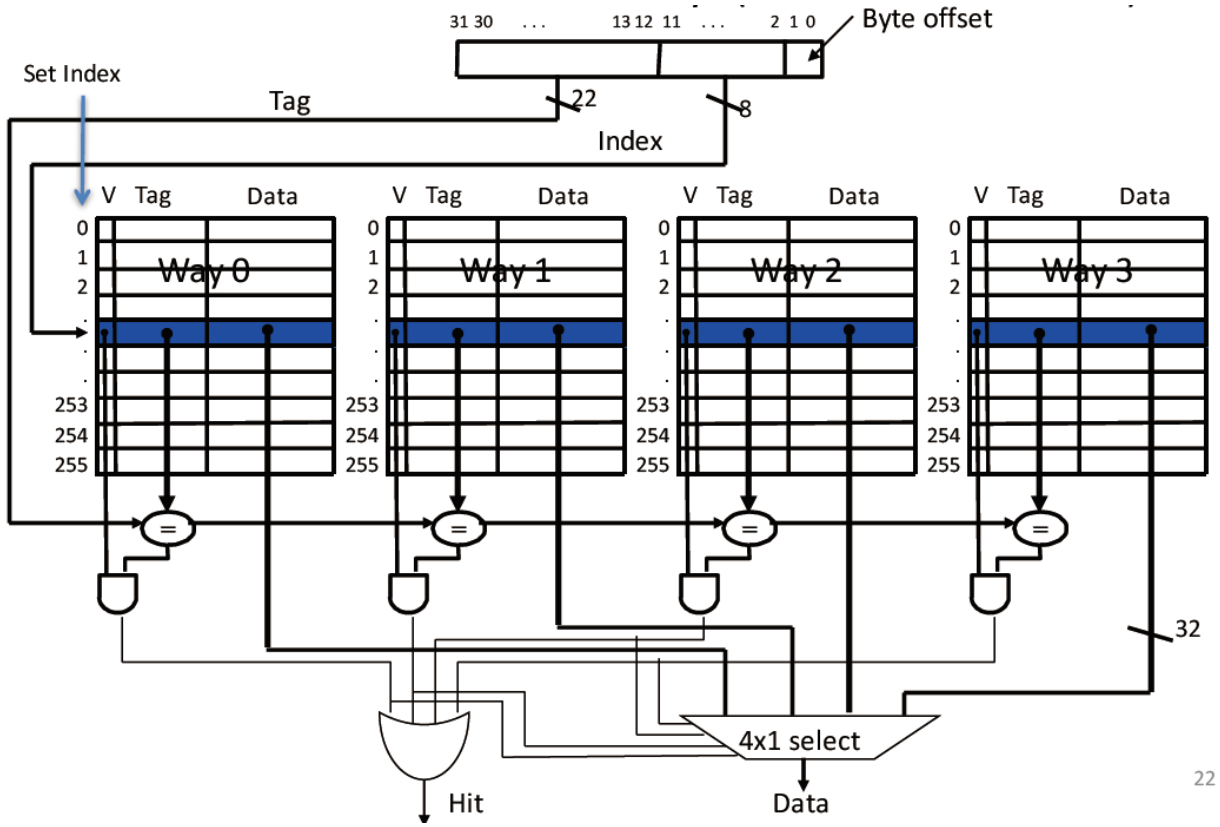
Set-Associative Caches

Associativity * # of sets * block_size

*Bytes = blocks/set * sets * Bytes/block*

*C = N * S * B*

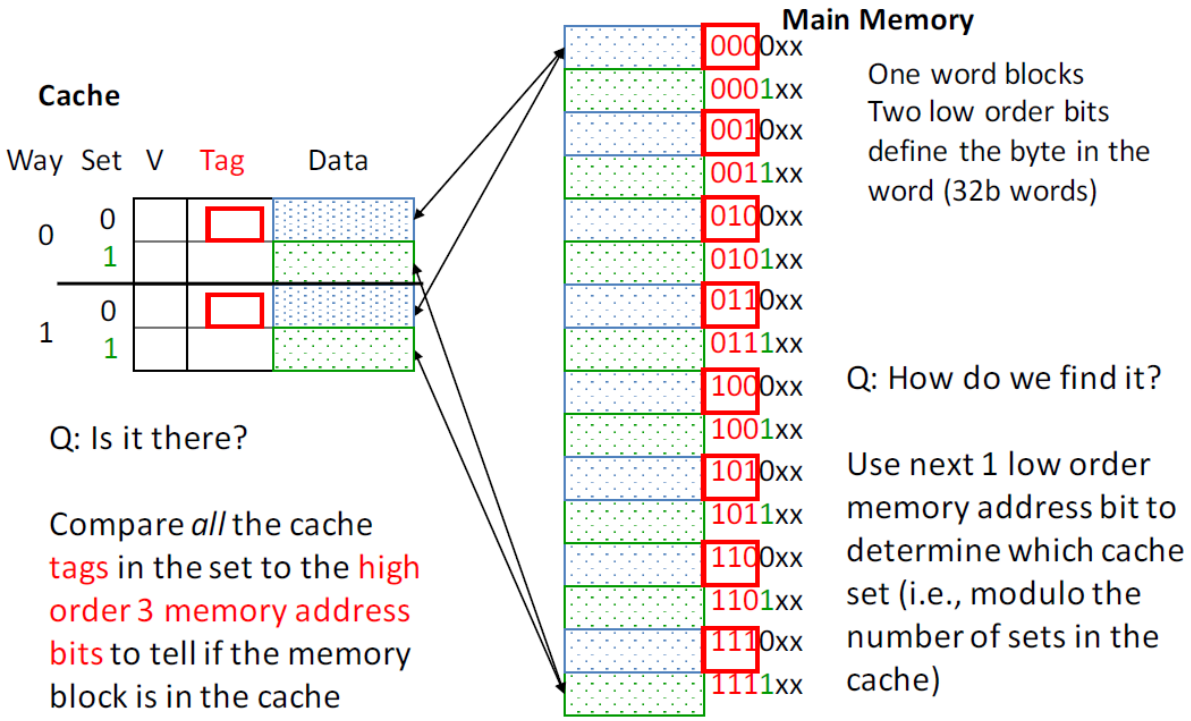
Tag	*Index*	*Byte Offset*



Set-Associative Caches

Same tag can be put anywhere in the set.

Example: 2-Way Set Associative $
(4 words = 2 sets x 2 ways per set)



Average Memory Access Time(AMAT)

AMAT = Time for a hit + Miss rate x Miss penalty

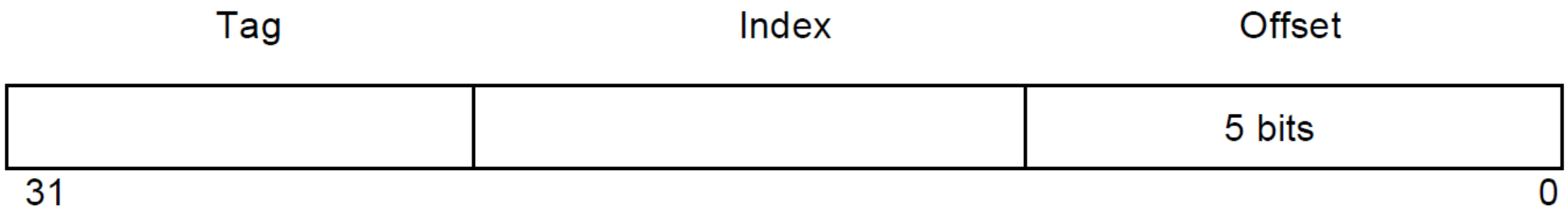
Given a 200 psec clock, a miss penalty of 50 clock cycles, a miss rate of 0.02 misses per instruction and a cache hit time of 1 clock cycle, what is AMAT?

- ☐ A: ≤200 psec
- ☐ B: 400 psec
- ☐ C: 600 psec
- ☐ D: ≥ 800 psec

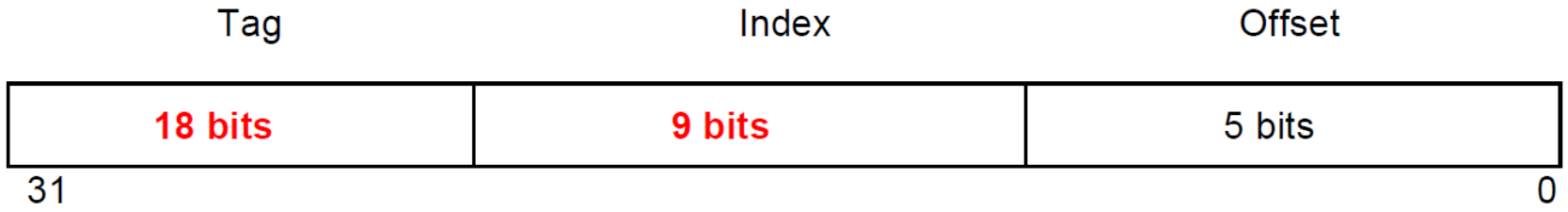
Exercise

- Consider a 32-bit physical memory space and a 32 KiB 2-way associative cache with LRU replacement.

You are told the cache uses 5 bits for the offset field. Write in the number of bits in the tag and index fields in the figure below.



Exercise

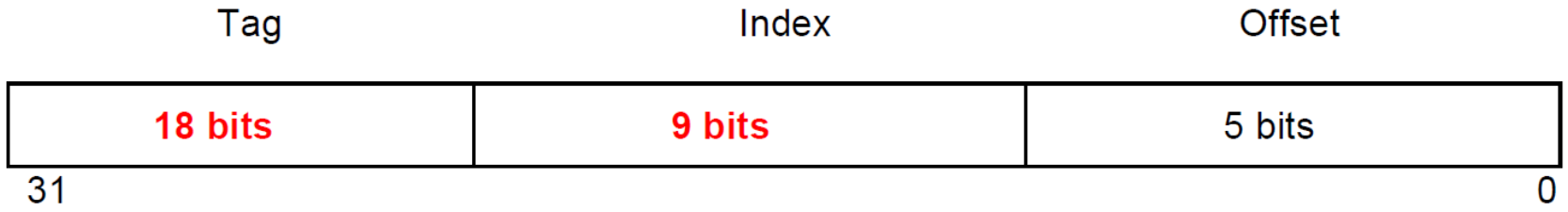


- For the same cache, after the execution of the following code:

```
int ARRAY_SIZE = 64 * 1024;
int arr[ARRAY_SIZE]; // *arr is aligned to a cache block
/* loop 1 */ for (int i = 0; i < ARRAY_SIZE; i += 8) arr[i] = i;
/* loop 2 */ for (int i = ARRAY_SIZE - 8; i >= 0; i -= 8)
                arr[i+1] = arr[i];
```

- 1. What is the hit rate of loop 1? What types of misses (of the 3 Cs), if any, occur as a result of loop 1?
- 2. What is the hit rate of loop 2? What types of misses (of the 3 Cs), if any, occur as a result of loop 2?

Exercise



- For the same cache, after the execution of the following code:

```
int ARRAY_SIZE = 64 * 1024;
int arr[ARRAY_SIZE]; // *arr is aligned to a cache block
/* loop 1 */ for (int i = 0; i < ARRAY_SIZE; i += 8) arr[i] = i;
/* loop 2 */ for (int i = ARRAY_SIZE - 8; i >= 0; i -= 8)
                arr[i+1] = arr[i];
```

- 1. What is the hit rate of loop 1? What types of misses (of the 3 Cs), if any, occur as a result of loop 1? 0, Compulsory Misses
- 2. What is the hit rate of loop 2? What types of misses (of the 3 Cs), if any, occur as a result of loop 2? 9/16, Capacity Misses