Introduction to Caches

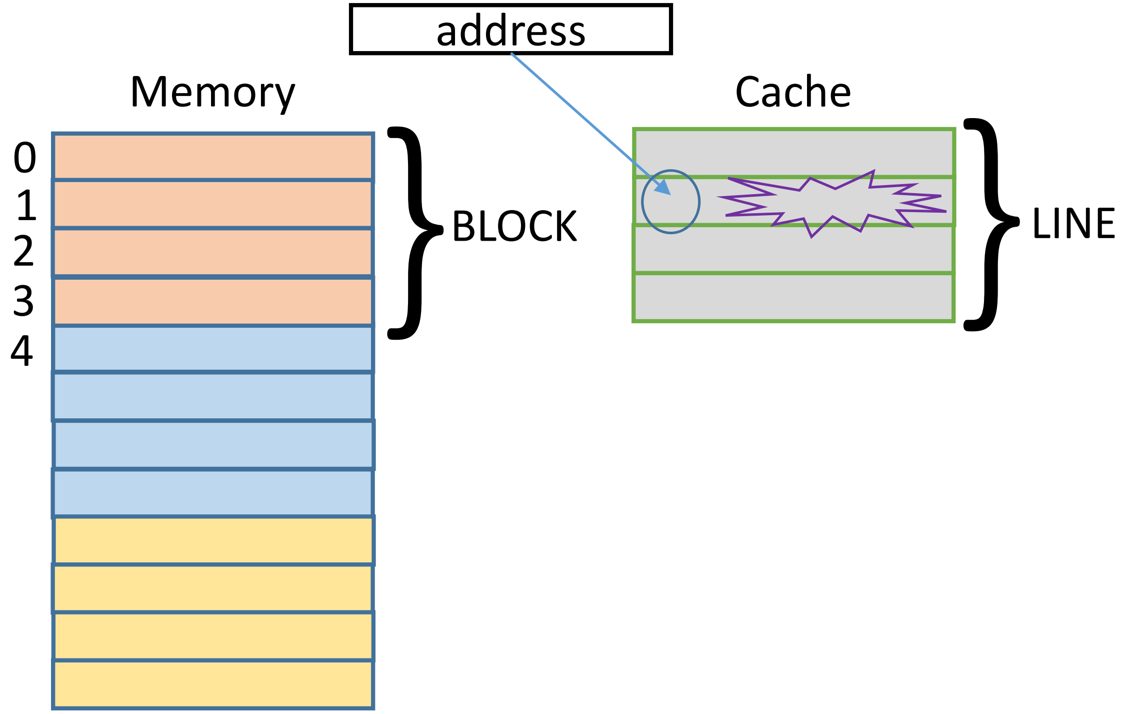
Quincy Flint

Caches

- Locality Principle
  - Memory references

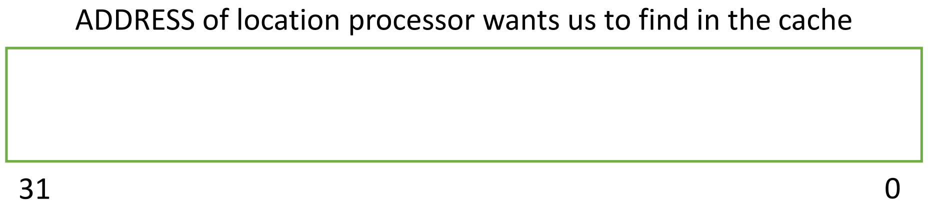
- Cache Organization
  - Hit or Miss
  - Block size
  - Block starting address
    - Aligned by block

```
int sum = 0;
for (int j=0, j<1000, j++)
        sum = sum + arr[j];
```



Block Offset and Block Number

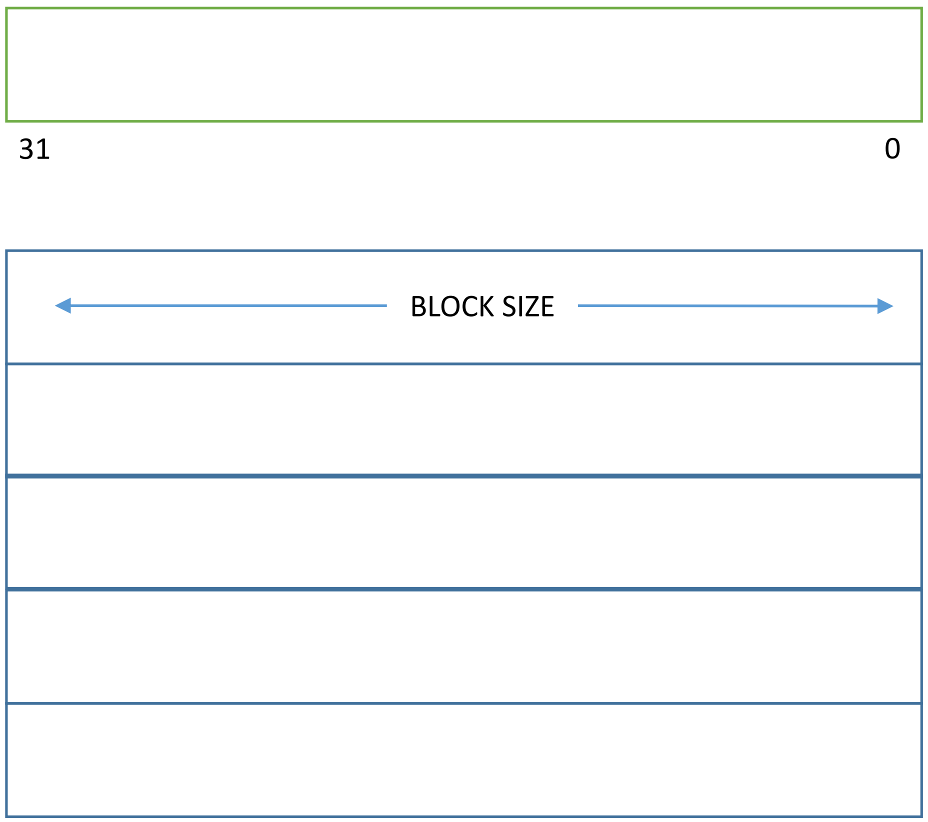
- Processor produces a 32-bit address


ADDRESS of location processor wants us to find in the cache

Block Offset and Block Number

- Processor produces a 32-bit address
- Example:
  - Block Size = 16

ADDRESS of location processor wants us to find in the cache

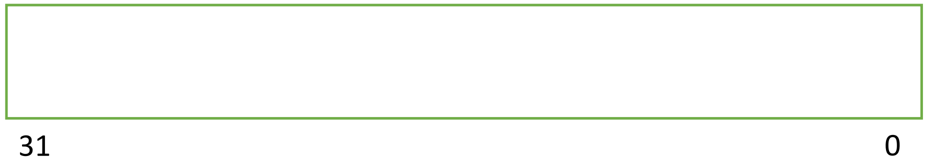


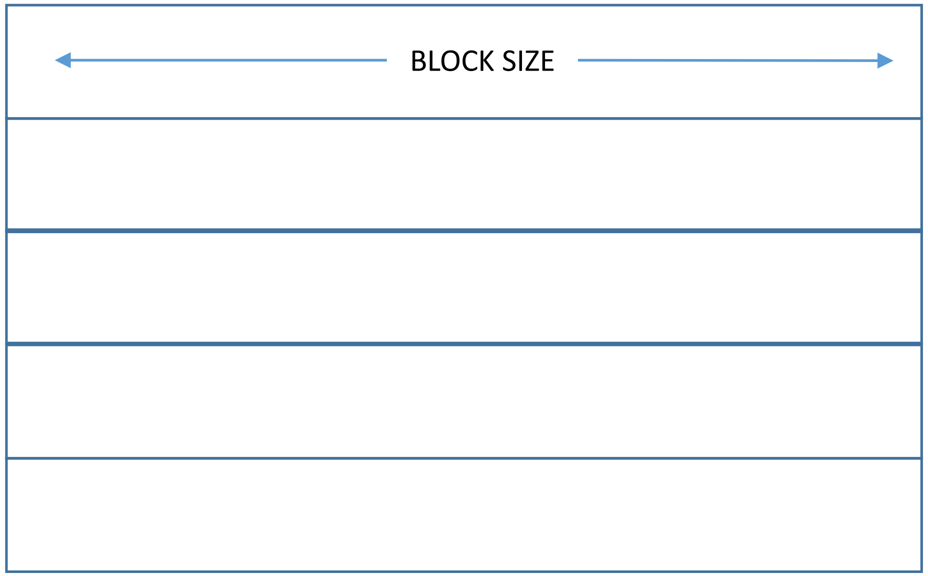
CACHE shown as an array of LINES of size BLOCK SIZE

Block Offset and Block Number

- Processor produces a 32-bit address
- Example:
  - Block Size = 16
  - How many bits of address are used to determine where in the block we are?

ADDRESS of location processor wants us to find in the cache





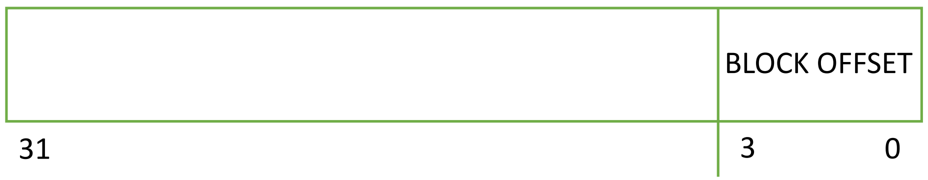
CACHE shown as an array of LINES of size BLOCK SIZE

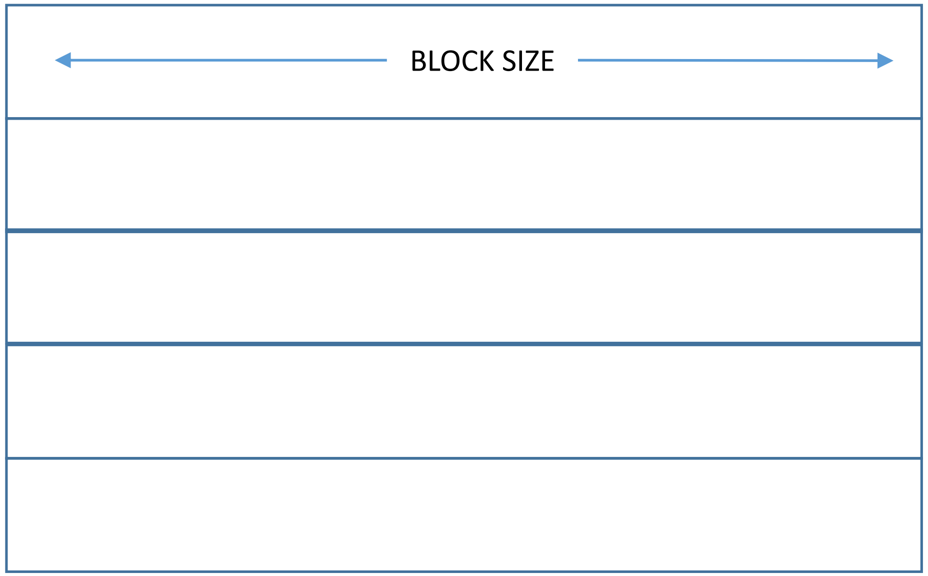
Block Offset and Block Number

- Processor produces a 32-bit address
- Example:
    - Block Size = 16
    - How many bits of address are used to determine where in the block we are?

BLOCK OFFSET

ADDRESS of location processor wants us to find in the cache





CACHE shown as an array of LINES of size BLOCK SIZE

Block Offset and Block Number

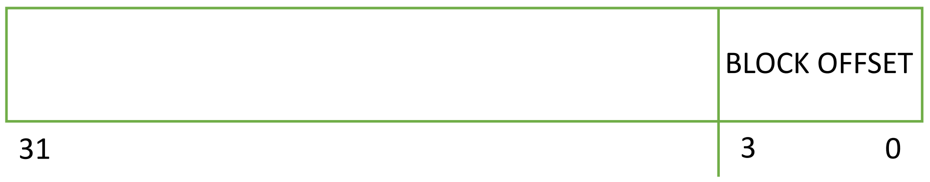
- Processor produces a 32-bit address

- Example:
  - Block Size = 16
  - How many bits of address are used to determine where in the block we are?

BLOCK OFFSET

  - How many bits of the address tell us which block we are looking for?

ADDRESS of location processor wants us to find in the cache

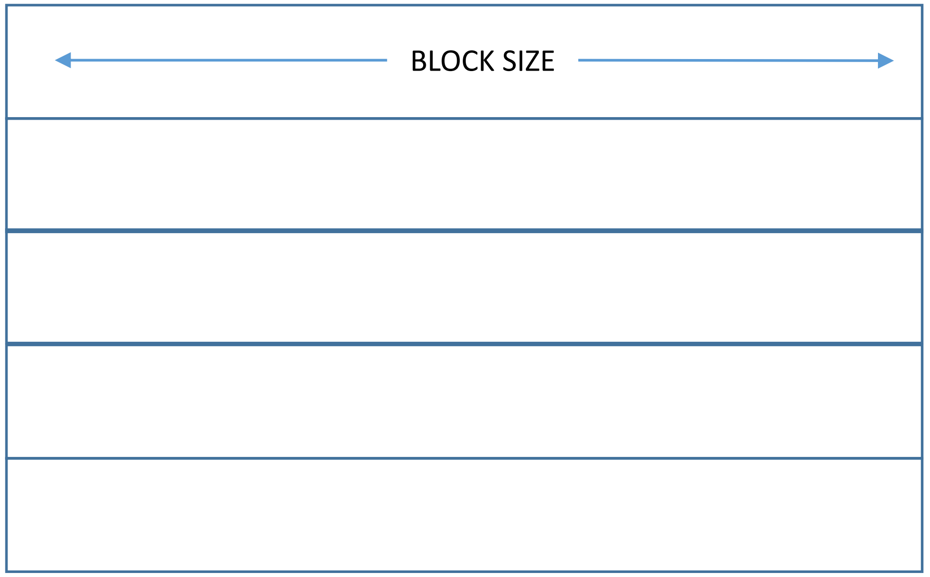


BLOCK OFFSET

31

3

0



BLOCK SIZE

CACHE shown as an array of LINES of size BLOCK SIZE

Block Offset and Block Number

- Processor produces a 32-bit address

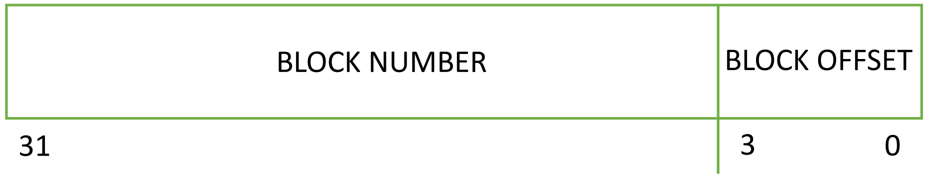
- Example:
  - Block Size = 16
  - How many bits of address are used to determine where in the block we are?

    BLOCK OFFSET

  - How many bits of the address tell us which block we are looking for?

    BLOCK NUMBER

ADDRESS of location processor wants us to find in the cache



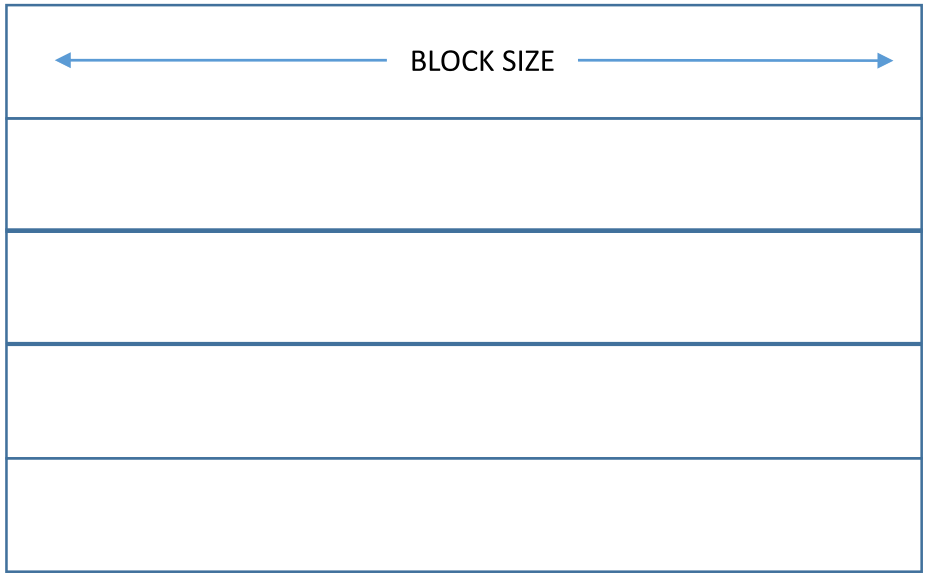
BLOCK NUMBER

BLOCK OFFSET

31

3

0



BLOCK SIZE

CACHE shown as an array of LINES of size BLOCK SIZE

Cache Tags

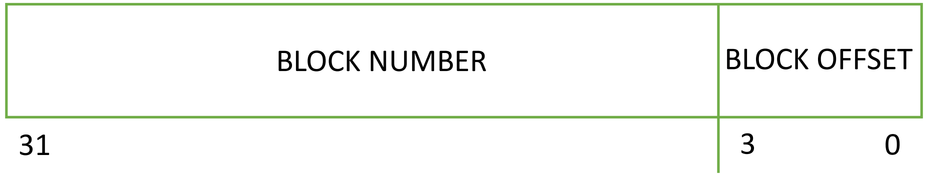
- A **cache tag** is a unique identifier for a block in cache line

Cache Tags

- A **cache tag** is a unique identifier for a block in cache line

- Assume each cache line can contain any block from memory…

ADDRESS of location processor wants us to find in the cache



BLOCK NUMBER

BLOCK OFFSET

31

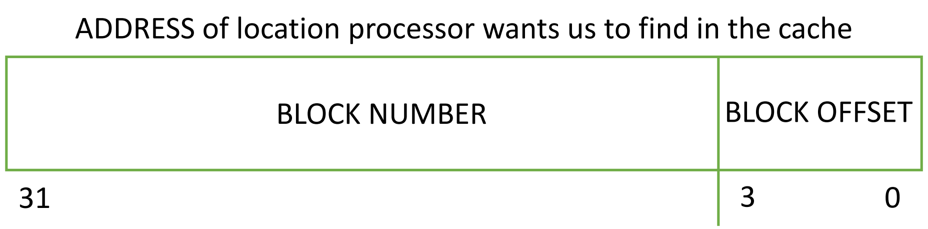
3

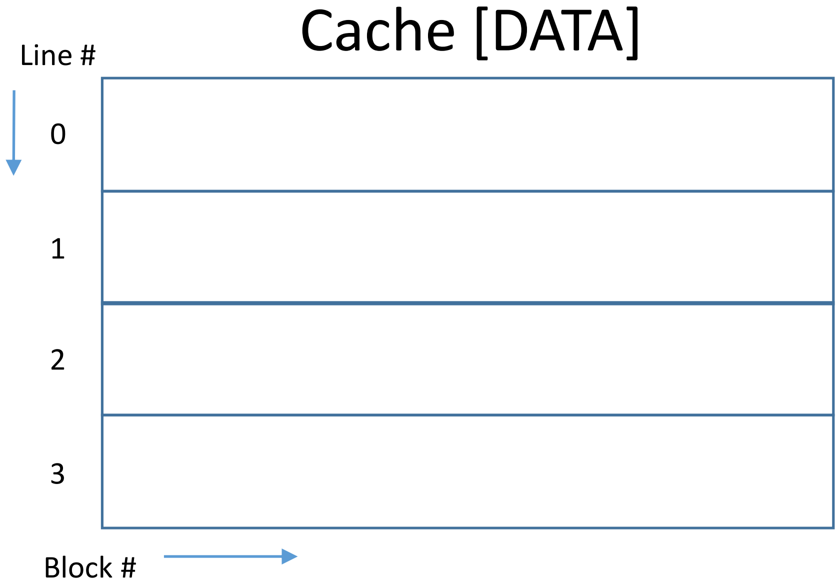
0

Cache Tags

- A **cache tag** is a unique identifier for a block in cache line

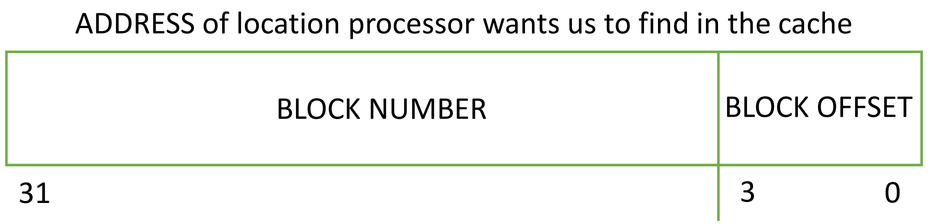
- Assume each cache line can contain any block from memory…





Cache Tags

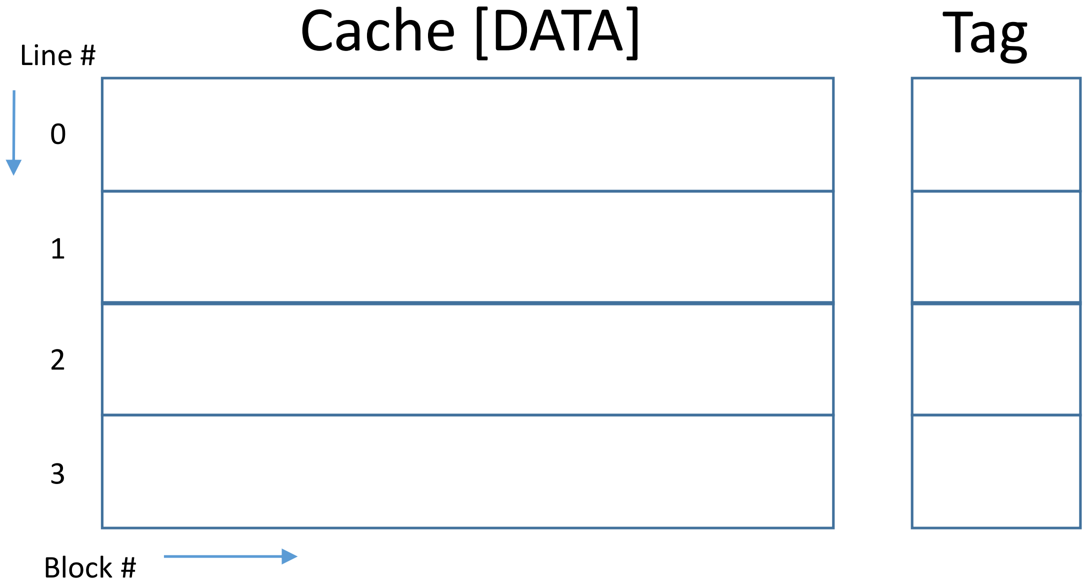
- A **cache tag** is a unique identifier for a block in cache line
- Assume each cache line can contain any block from memory…



ADDRESS of location processor wants us to find in the cache

BLOCK NUMBER | BLOCK OFFSET

31 3 0



Cache [DATA]

Tag

Line #

0

1

2

3

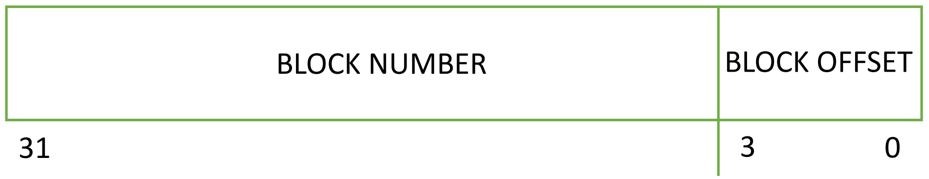
Block #

Cache Tags

- A **cache tag** is a unique identifier for a block in cache line

- Assume each cache line can contain any block from memory…

ADDRESS of location processor wants us to find in the cache



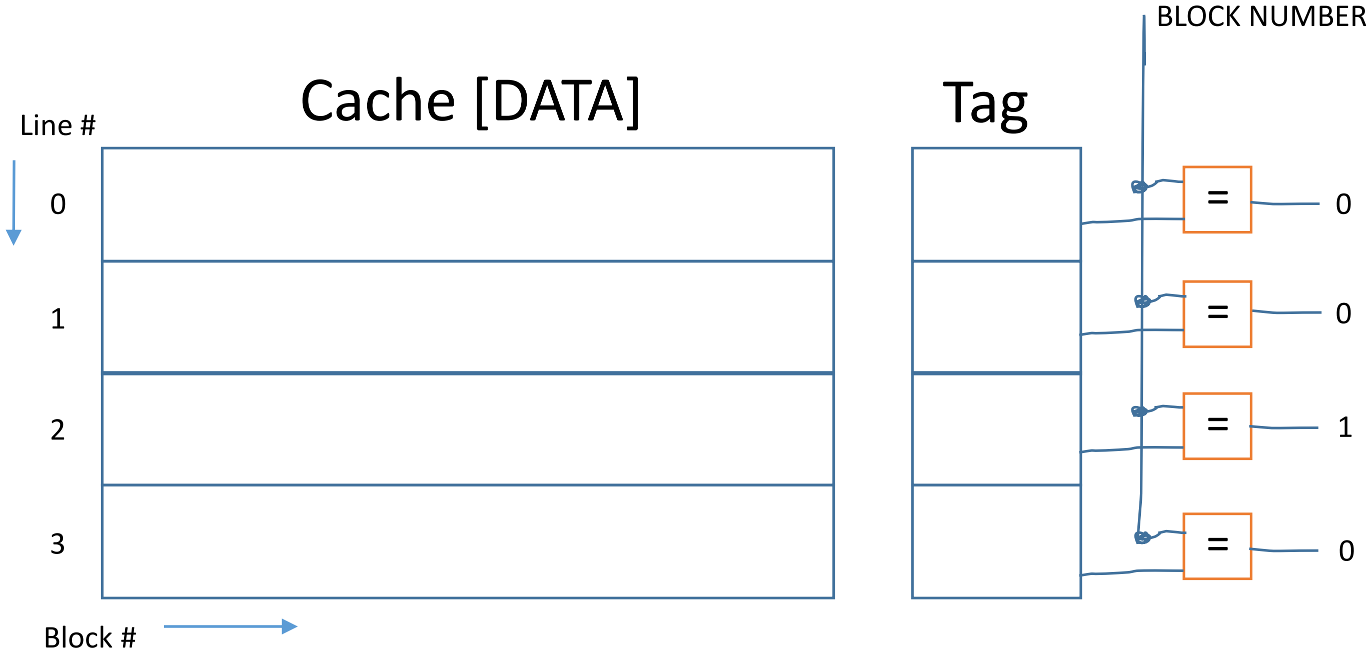
BLOCK NUMBER

BLOCK OFFSET

31

3

0



BLOCK NUMBER

Cache [DATA]

Tag

Line #

0

1

2

3

Block #

=

=

=

=

0

0

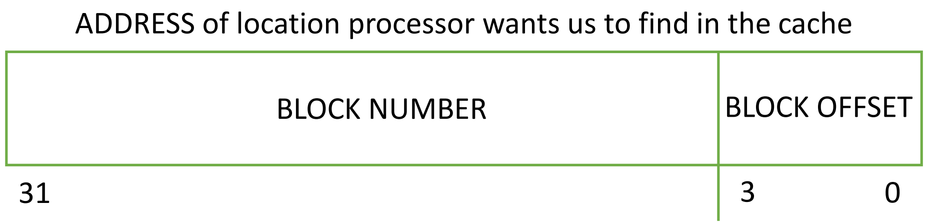
1

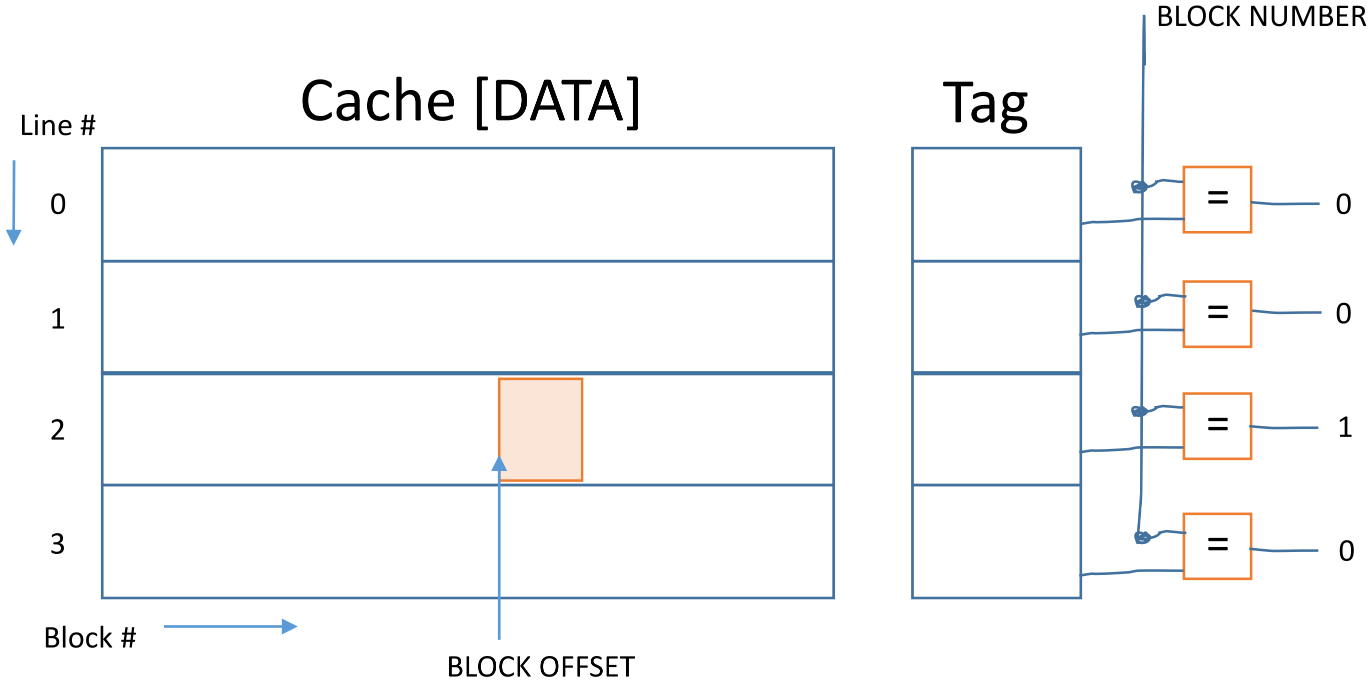
0

Cache Tags

- A **cache tag** is a unique identifier for a block in cache line

- Assume each cache line can contain any block from memory…



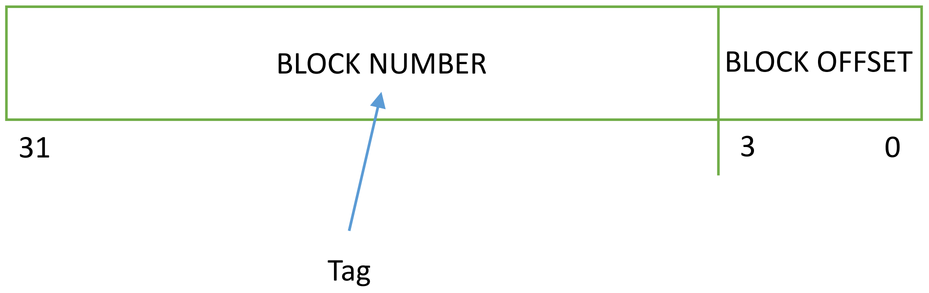


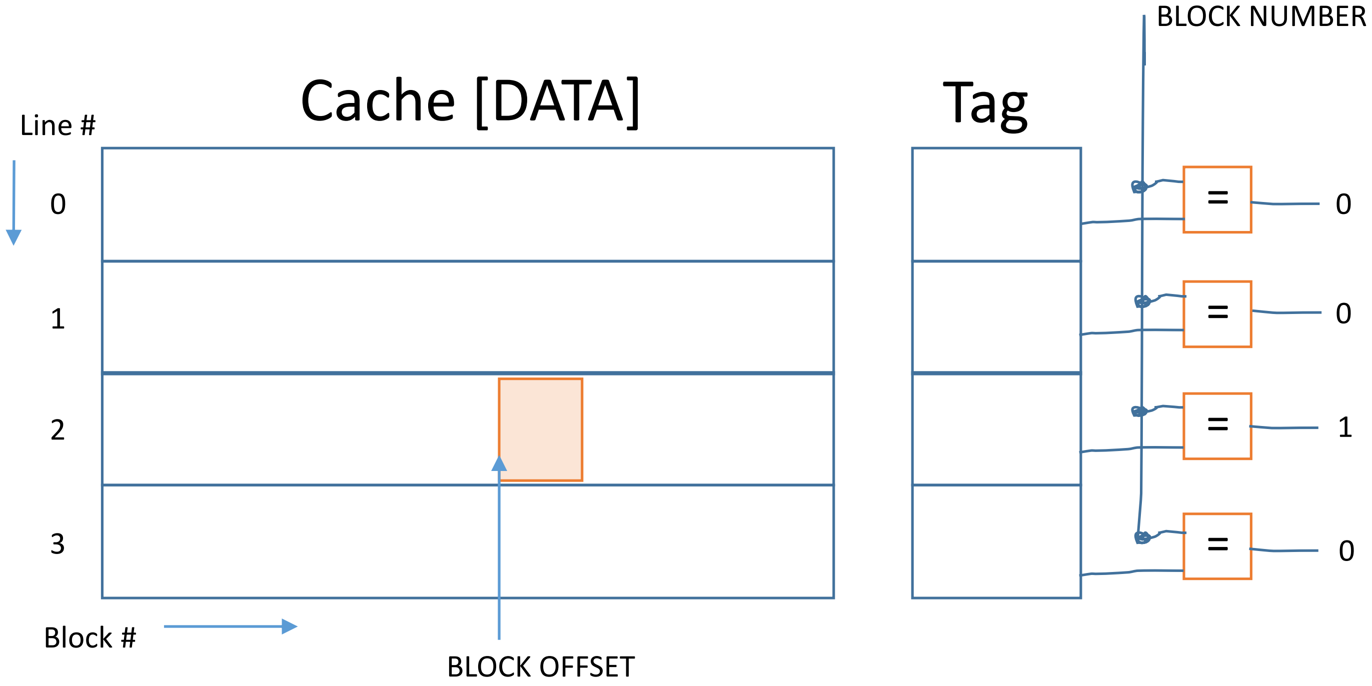
Cache Tags

- A **cache tag** is a unique identifier for a block in cache line

- Assume each cache line can contain any block from memory…

ADDRESS of location processor wants us to find in the cache





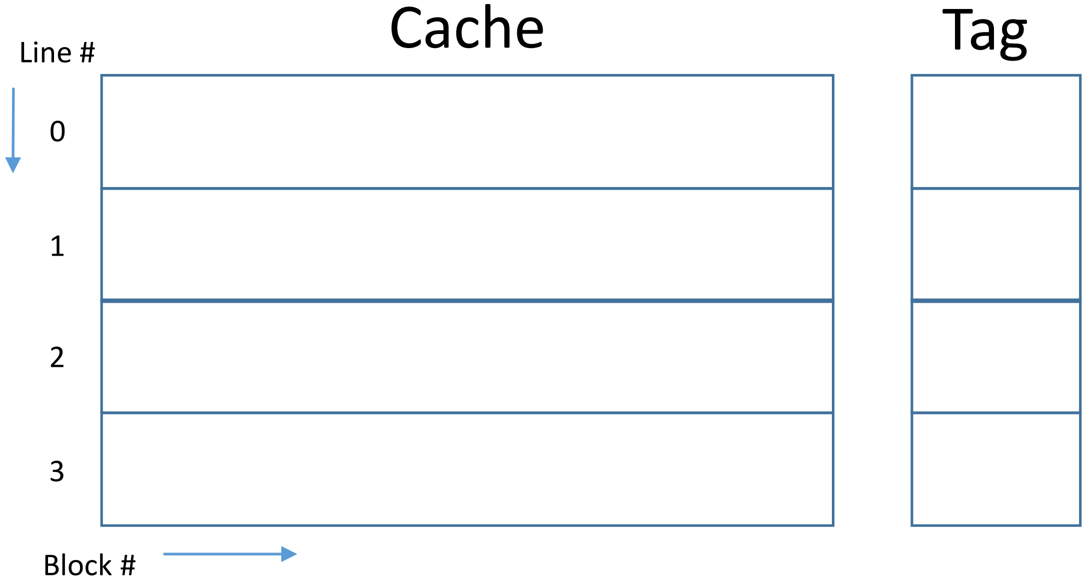
Valid Bit

- The valid bit in a cache entry tells us if the data is real or not.

- Solves the problem of ambiguity of tag and data when initialized.

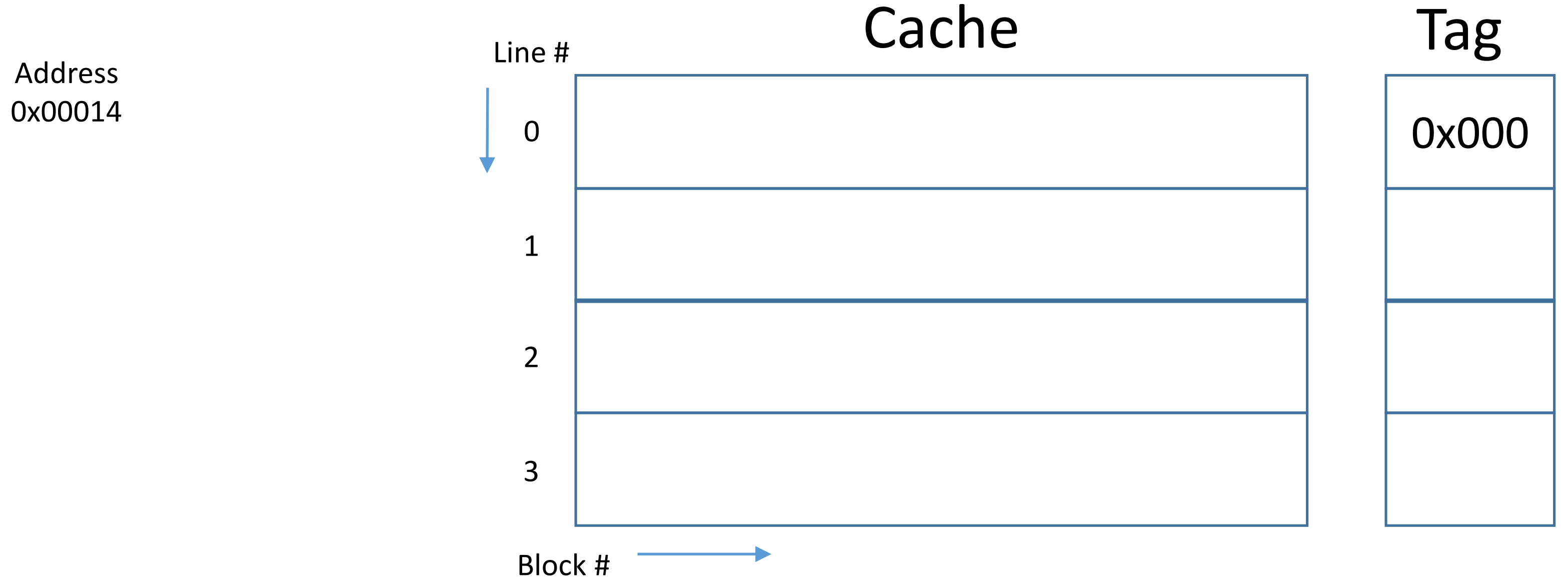
Valid Bit

- The valid bit in a cache entry tells us if the data is real or not.
- Solves the problem of ambiguity of tag and data when initialized.



Valid Bit

- The valid bit in a cache entry tells us if the data is real or not.
- Solves the problem of ambiguity of tag and data when initialized.

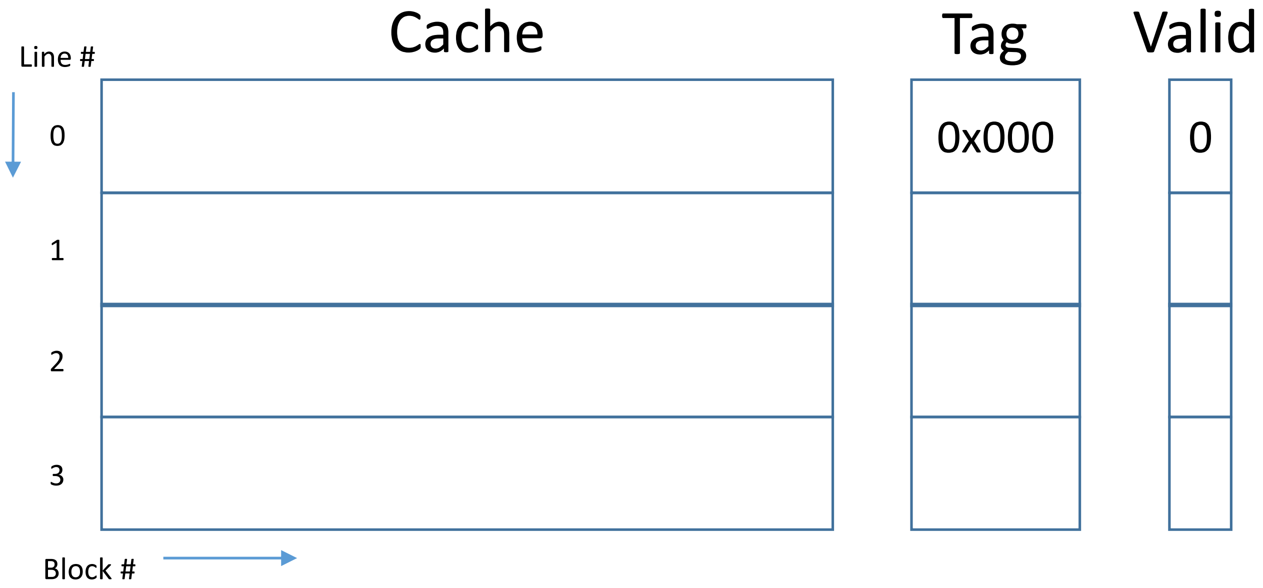


Valid Bit

- The valid bit in a cache entry tells us if the data is real or not.

- Solves the problem of ambiguity of tag and data when initialized.

Address
0x00014



Line #	Cache	Tag	Valid
0		0x000	0
1			
2			
3			

Block #

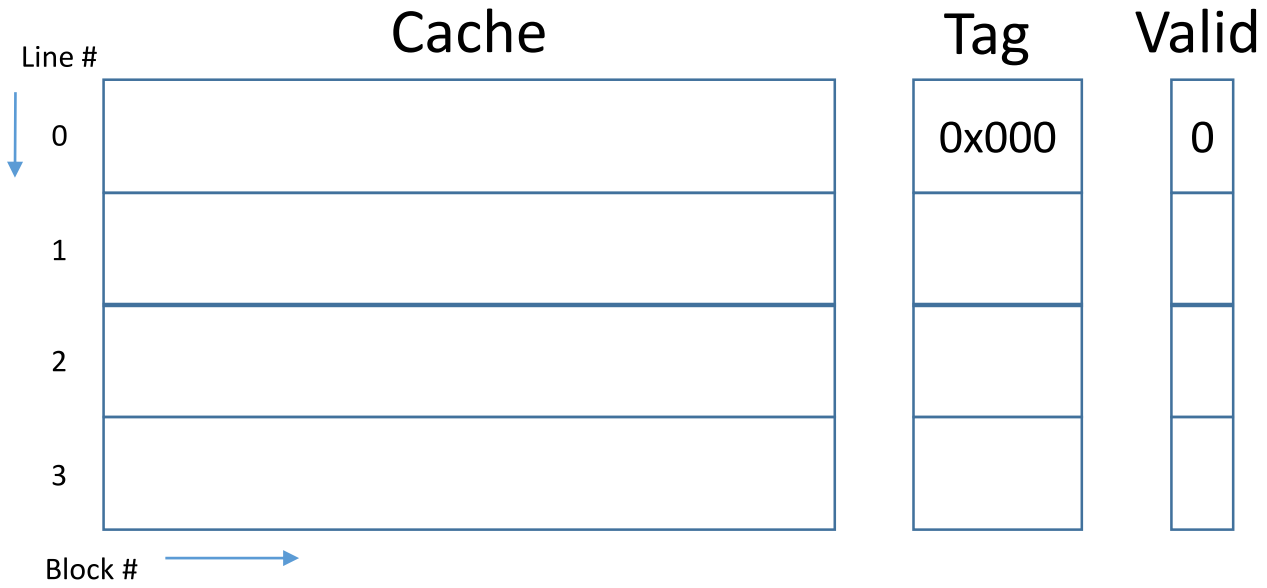
Valid Bit

- The valid bit in a cache entry tells us if the data is real or not.

- Solves the problem of ambiguity of tag and data when initialized.

Address
0x00014

Hit = (Tag == Block Number) AND Valid



Line #	Cache	Tag	Valid
0		0x000	0
1			
2			
3			

Types of Caches

- Fully Associative
  - Any block of memory can be placed in any line of cache

Types of Caches

- Fully Associative
  - Any block of memory can be placed in any line of cache
- Direct Mapped
  - A block of memory can only go in 1 line

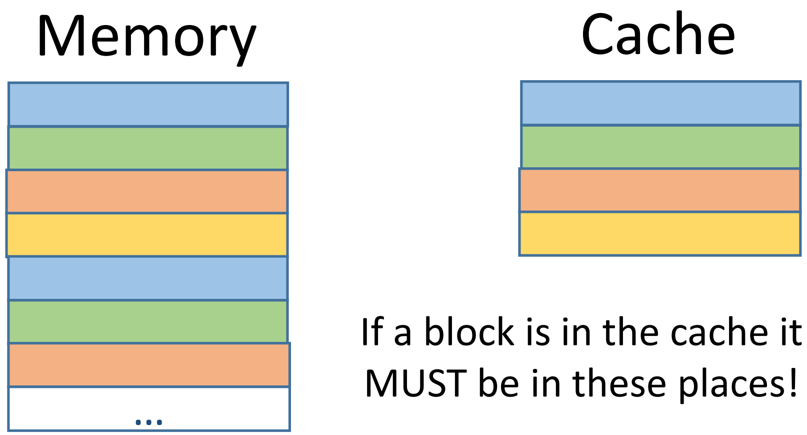
Types of Caches

- Fully Associative
  - Any block of memory can be placed in any line of cache
- Direct Mapped
  - A block of memory can only go in 1 line
- Set Associative
  - N lines where a block can be placed

Types of Caches

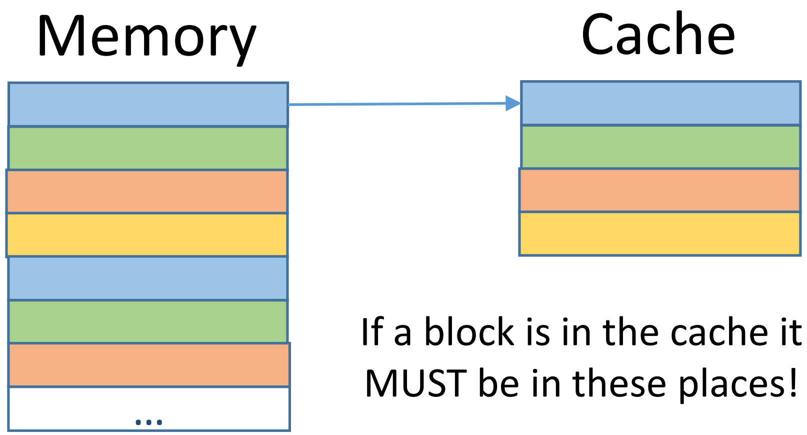
- Fully Associative [Cache Size-Way SA]
  - Any block of memory can be placed in any line of cache
- Direct Mapped [1-Way SA]
  - A block of memory can only go in 1 line
- Set Associative
  - N lines where a block can be placed

Direct Mapped



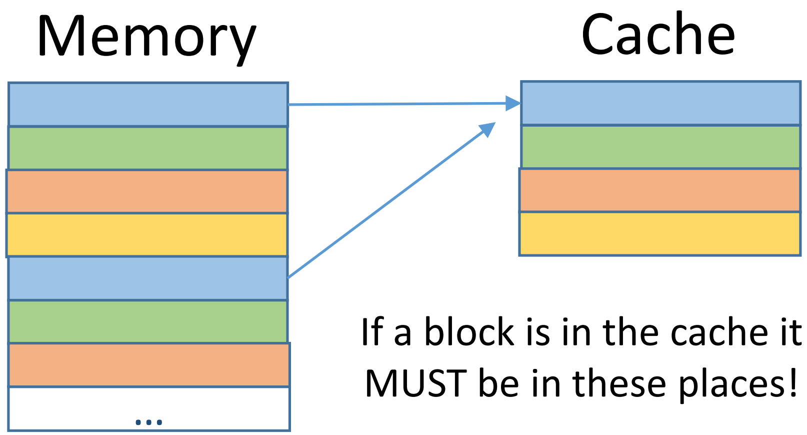
If a block is in the cache it MUST be in these places!

Direct Mapped



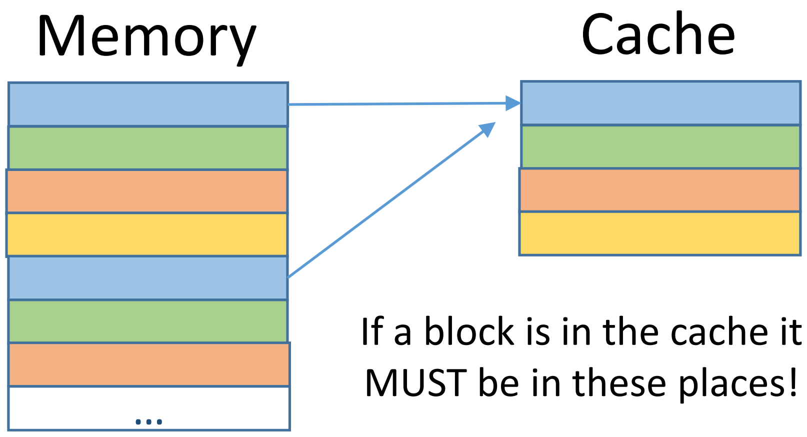
If a block is in the cache it MUST be in these places!

Direct Mapped



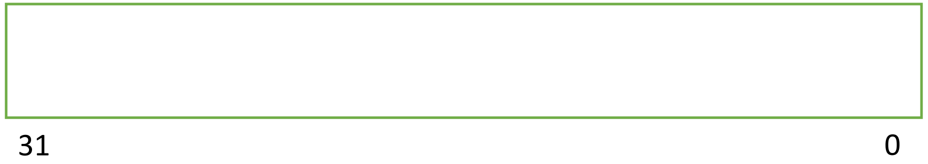
If a block is in the cache it MUST be in these places!

Direct Mapped

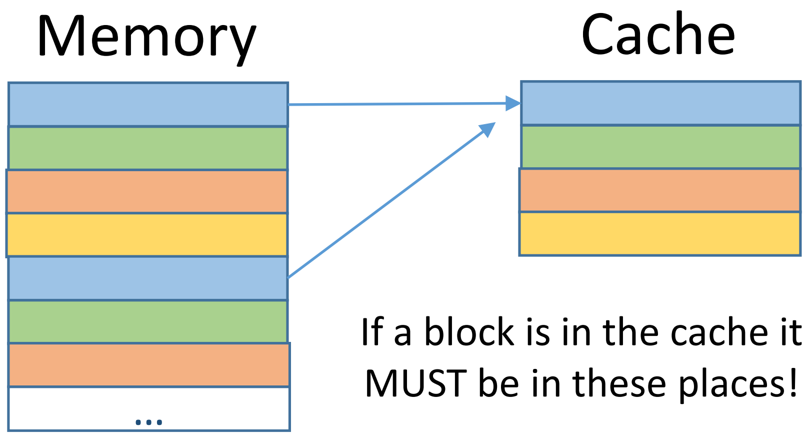


If a block is in the cache it MUST be in these places!

ADDRESS of location processor wants us to find in the cache

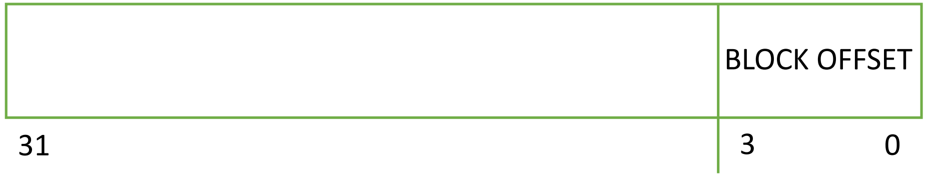


Direct Mapped

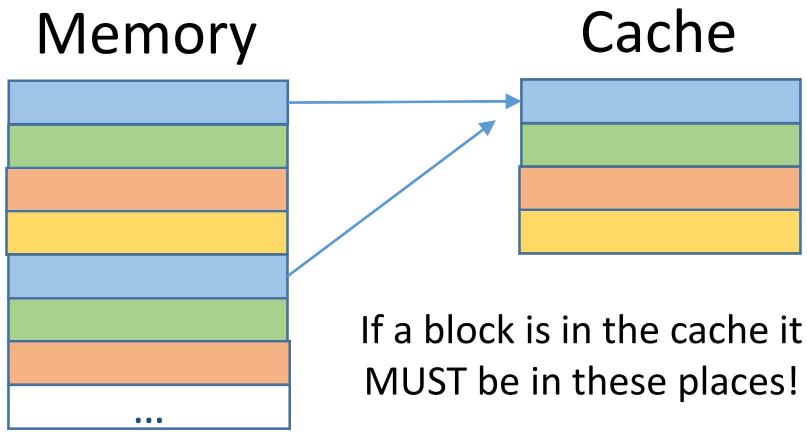


If a block is in the cache it MUST be in these places!

ADDRESS of location processor wants us to find in the cache



Direct Mapped

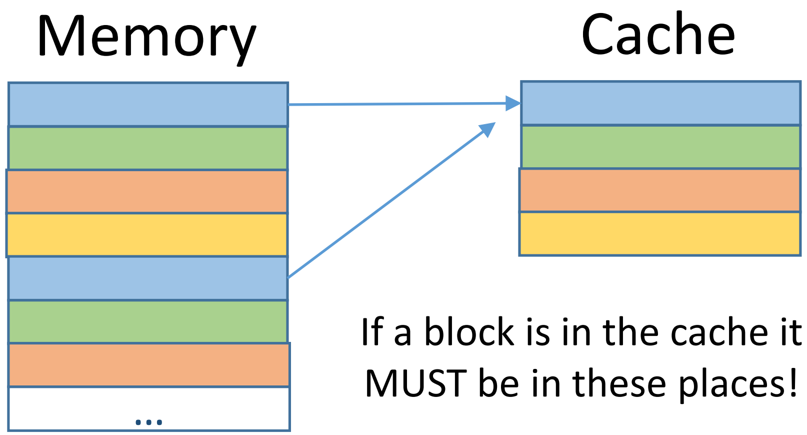


If a block is in the cache it MUST be in these places!

ADDRESS of location processor wants us to find in the cache

	INDEX	BLOCK OFFSET
31		3 0

Direct Mapped

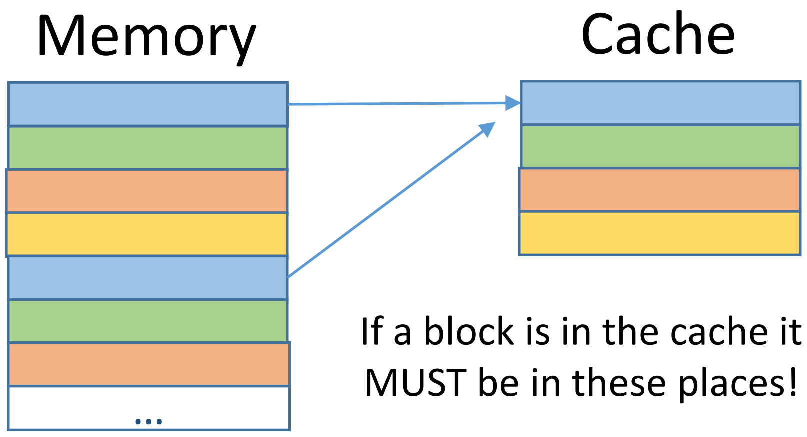


If a block is in the cache it MUST be in these places!

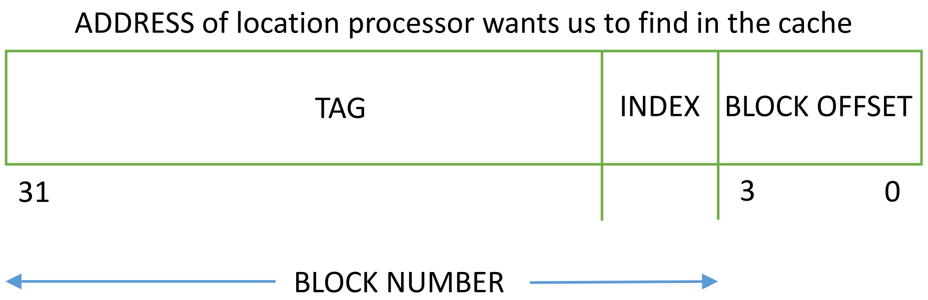
ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET
31		3 0

Direct Mapped



If a block is in the cache it MUST be in these places!

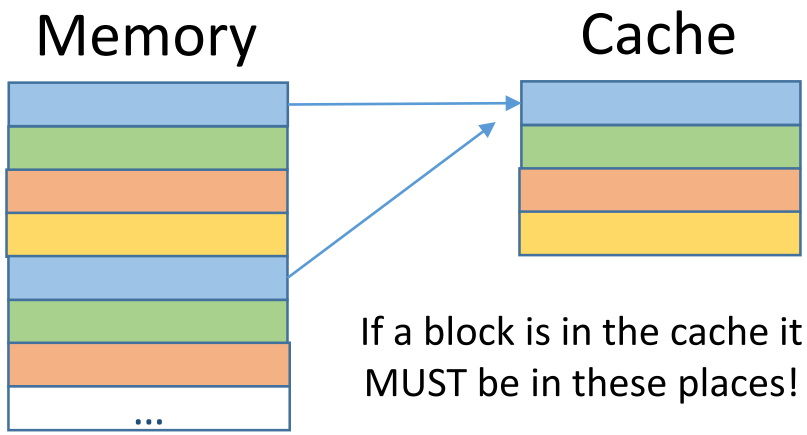

ADDRESS of location processor wants us to find in the cache

TAG | INDEX | BLOCK OFFSET

31 … 3 … 0

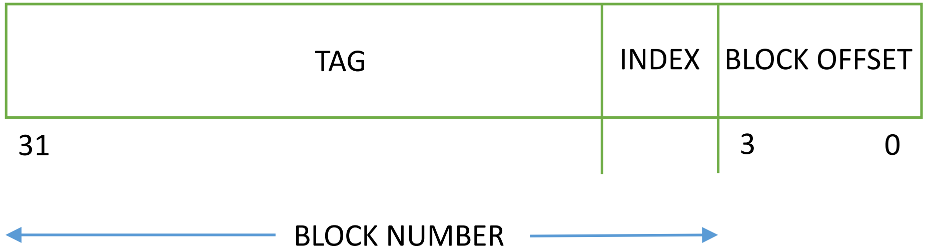
BLOCK NUMBER

Direct Mapped



If a block is in the cache it MUST be in these places!

ADDRESS of location processor wants us to find in the cache



TAG

INDEX

BLOCK OFFSET

31

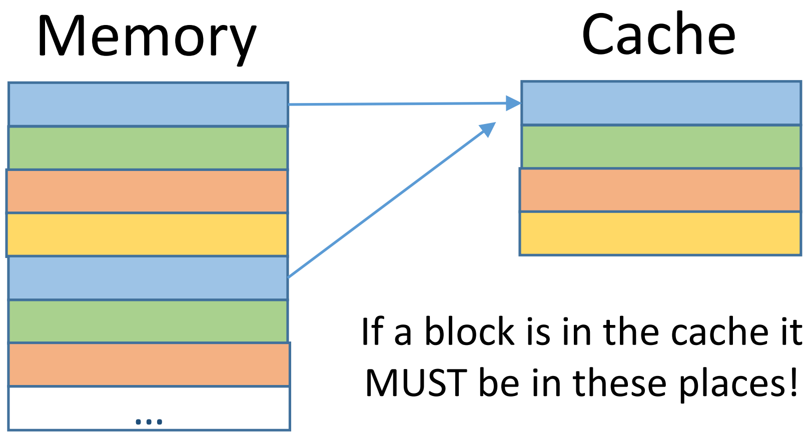
3

0

BLOCK NUMBER

Tag tells us which block (of those possible) is in the cache.
It does not include the index bits as they are redundant!

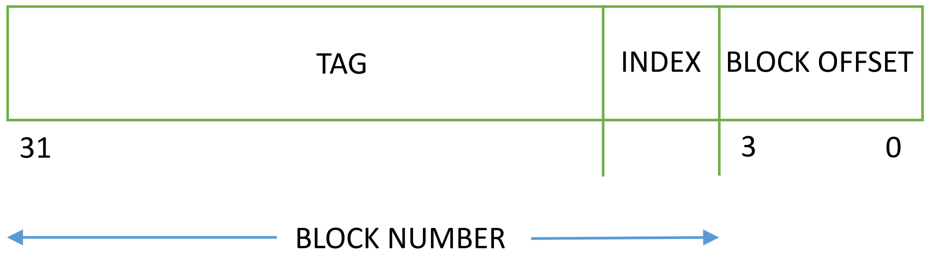
Direct Mapped



If a block is in the cache it MUST be in these places!

- Considerations:
  - Only need to search 1 place!
    - Fast, cheap, efficient
  - A block must go in 1 place!
    - Underutilized, conflicts

ADDRESS of location processor wants us to find in the cache



Tag tells us which block (of those possible) is in the cache. It does not include the index bits as they are redundant!

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

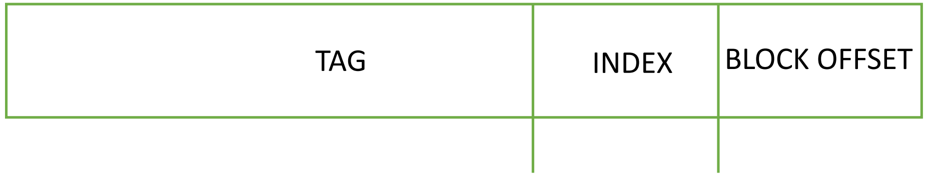
- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

ADDRESS of location processor wants us to find in the cache



TAG

INDEX

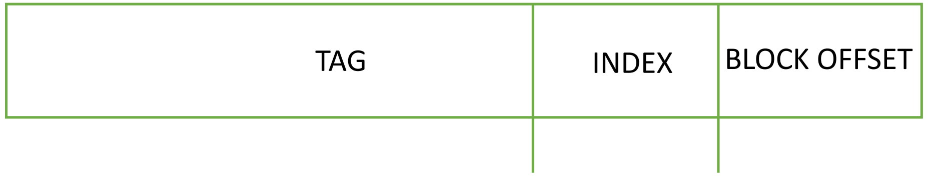
BLOCK OFFSET

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

ADDRESS of location processor wants us to find in the cache



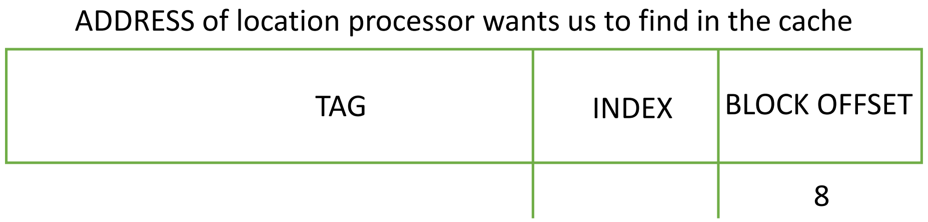
TAG	INDEX	BLOCK OFFSET

Offset: **8 bits** to address 256 Byte blocks

Direct Mapped Cache Quiz 1

- Given:
    - 16 kB direct-mapped cache
    - 256 Byte blocks
    - Address 0x12345678

- Which blocks conflict?
    - 0x12345677
    - 0x11335577
    - 0x11115678
    - 0x12341666



ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET
		8

Offset: **8 bits** to address 256 Byte blocks

Number of Blocks = 16 kB cache/ 256 Bytes per block = 64

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET
		8

Offset: **8 bits** to address 256 Byte blocks

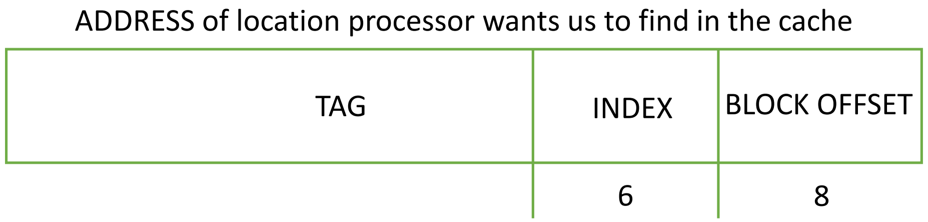
Number of Blocks = 16 kB cache/ 256 Bytes per block = 64

Index bits: **6 bits** to address 64 blocks

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666



ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET
	6	8

Offset: **8 bits** to address 256 Byte blocks

Number of Blocks = 16 kB cache/ 256 Bytes per block = 64

Index bits: **6 bits** to address 64 blocks

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET
	6	8

Offset: **8 bits** to address 256 Byte blocks

Number of Blocks = 16 kB cache/ 256 Bytes per block = 64

Index bits: **6 bits** to address 64 blocks

0x123456 78

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET
	6	8

Offset: **8 bits** to address 256 Byte blocks

Number of Blocks = 16 kB cache/ 256 Bytes per block = 64

Index bits: **6 bits** to address 64 blocks

0x123456 77
0x113355 77
0x111156 78
0x123416 66

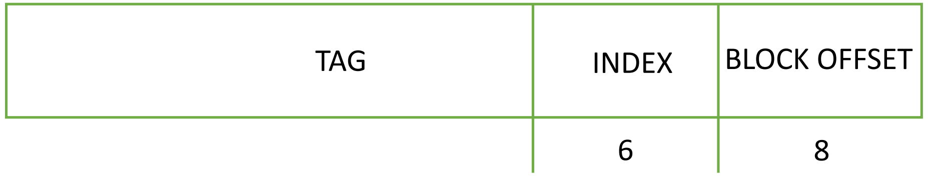
0x123456 78

Direct Mapped Cache Quiz 1

- Given:
  - 16 kB direct-mapped cache
  - 256 Byte blocks
  - Address 0x12345678

- Which blocks conflict?
  - 0x12345677
  - 0x11335577
  - 0x11115678
  - 0x12341666

ADDRESS of location processor wants us to find in the cache



Offset: **8 bits** to address 256 Byte blocks

Number of Blocks = 16 kB cache/ 256 Bytes per block = 64

Index bits: **6 bits** to address 64 blocks

0x123456 77
0x113355 77
0x111156 78
0x123416 66

0x56 = 01 01 0110
0x55 = 01 01 0101
0x16 = 00 01 0110

0x123456 78

Direct Mapped Cache Quiz

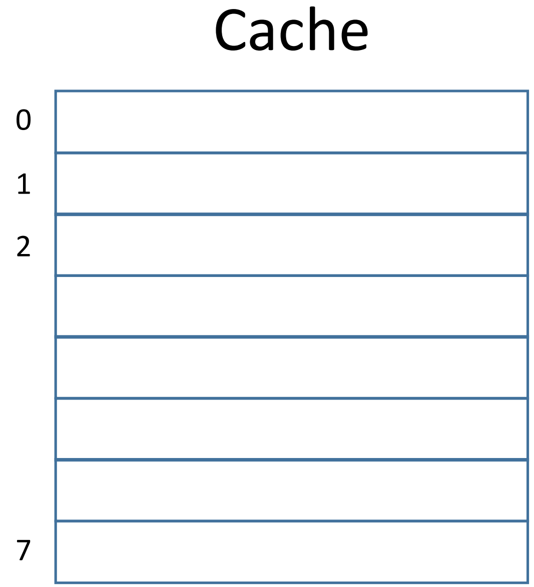
- Given:
  - Byte addressable cache
  - 32 Byte blocks
  - Sequence of addresses
    - [A] 0x3F1F
    - [B] 0x3F2F
    - [C] 0x3F2E
    - [D] 0x3E1F
- What does the cache do?

Cache

Direct Mapped Cache Quiz

- Given:
  - Byte addressable cache
  - 32 Byte blocks
  - Sequence of addresses
    - [A] 0x3F1F
    - [B] 0x3F2F
    - [C] 0x3F2E
    - [D] 0x3E1F

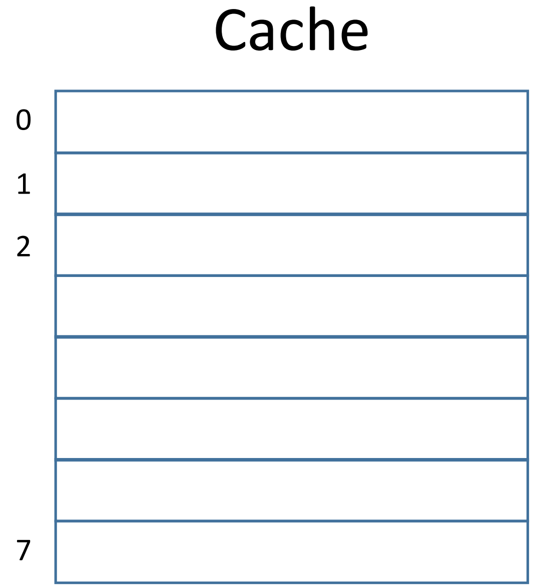
- What does the cache do?



Offset: **5 bits** to address 32 Byte blocks

Direct Mapped Cache Quiz

- Given:
  - Byte addressable cache
  - 32 Byte blocks
  - Sequence of addresses
    - [A] 0x3F1F
    - [B] 0x3F2F
    - [C] 0x3F2E
    - [D] 0x3E1F



Offset: **5 bits** to address 32 Byte blocks

Index bits: **3 bits** to address 7 cache lines

- What does the cache do?

Direct Mapped Cache Quiz

- Given:
  - Byte addressable cache
  - 32 Byte blocks
  - Sequence of addresses
    - [A] 0x3F1F
    - [B] 0x3F2F
    - [C] 0x3F2E
    - [D] 0x3E1F

- What does the cache do?

Cache

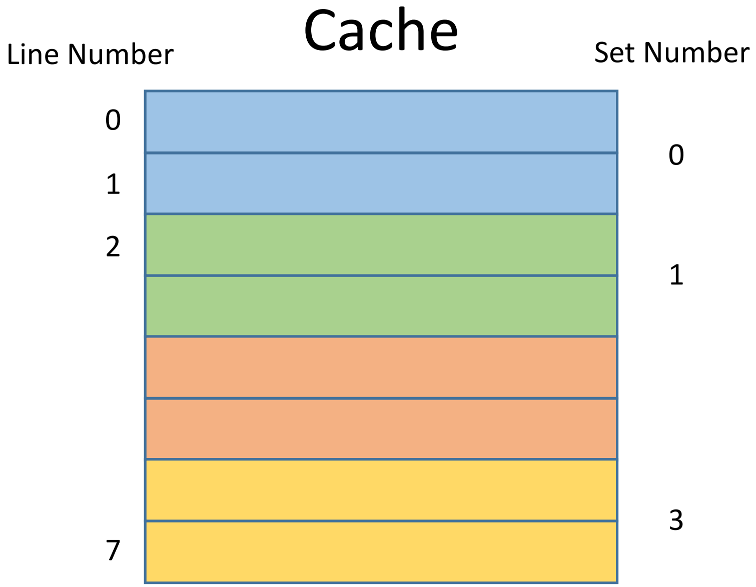
Offset: **5 bits** to address 32 Byte blocks

Index bits: **3 bits** to address 7 cache lines

Tag bits: 8 bits remaining

Set-Associative Cache

- N-Way Set-Associative Cache
  - A block can be placed in 1 of N lines



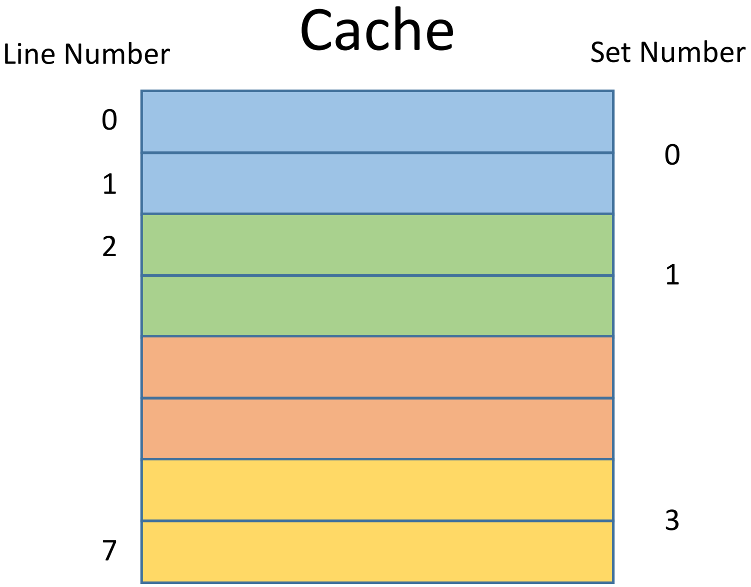
Here we have:

8 cache lines
4 sets
2 ways

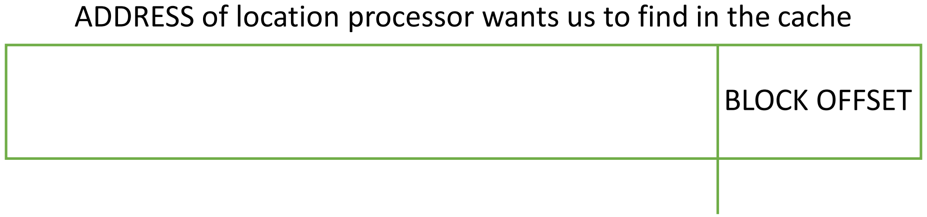
2-Way SA so each set has 2 lines where a block can go

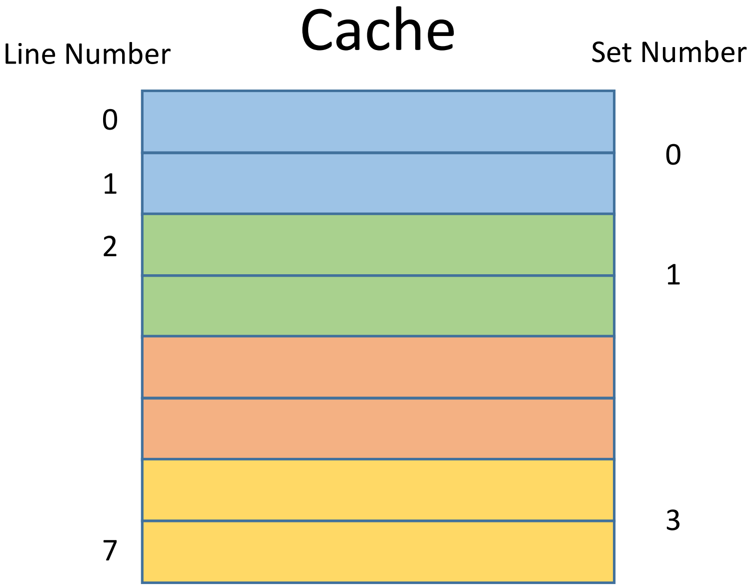
Set-Associative Caches: **tag, index, offset**

ADDRESS of location processor wants us to find in the cache



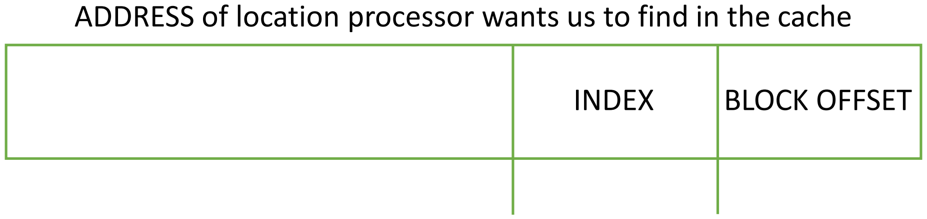
Set-Associative Caches: **tag, index, offset**

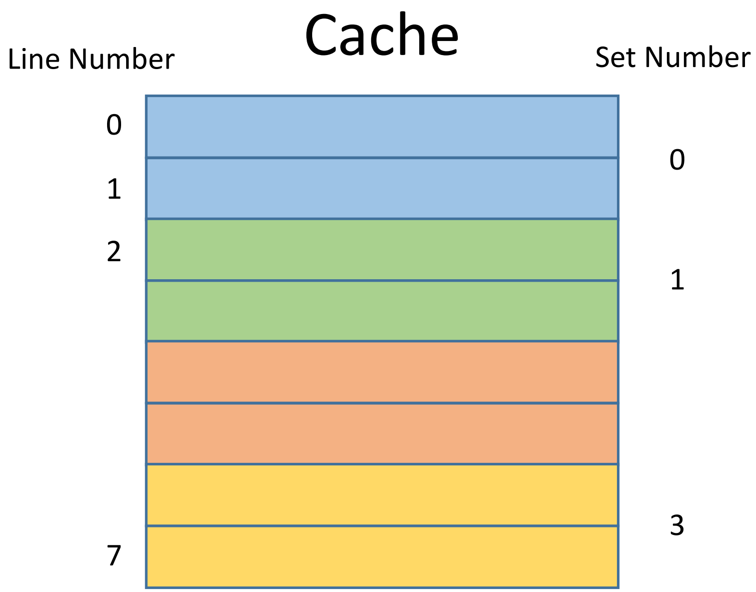




- **Block Offset** is the same as before!

Set-Associative Caches: **tag, index, offset**

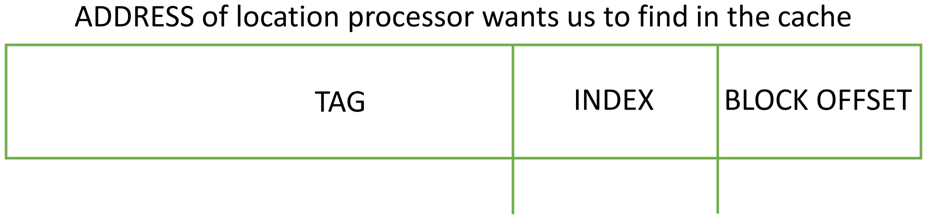


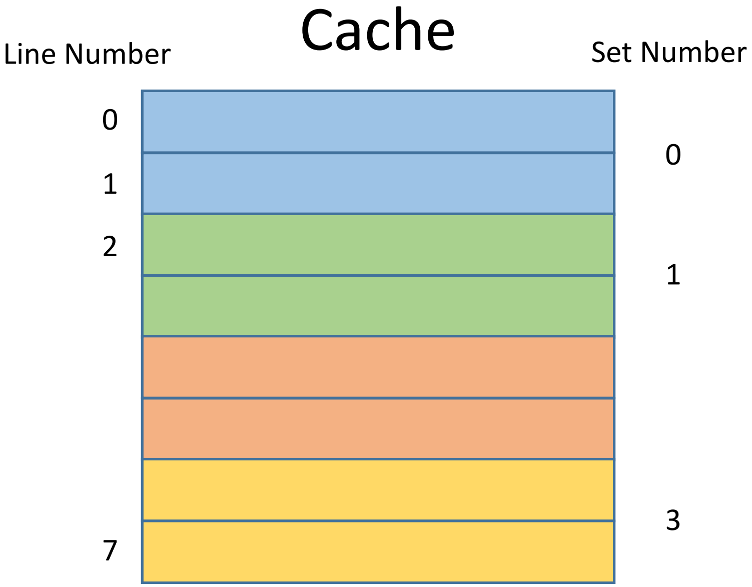


- **Block Offset** is the same as before!

- **Index bits** determine set!
  - With 4 sets we need 2 index bits

Set-Associative Caches: **tag, index, offset**





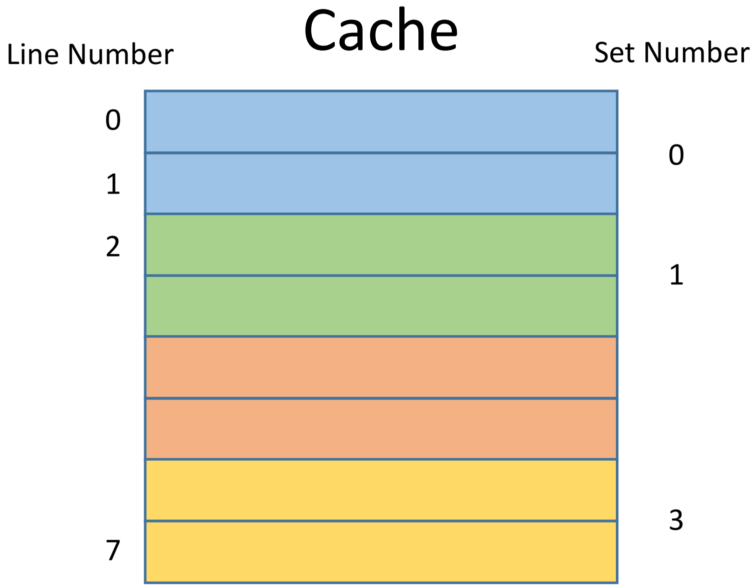
- **Block Offset** is the same as before!
- **Index bits** determine set!
  - With 4 sets we need 2 index bits
- **Tag** is the remaining block offset bits not used by index

Don’t keep index bits in tag! They are repeated

2-Way Set-Associative Cache Quiz

- Given:
  - Byte addressable cache
  - 32 Byte blocks
  - Sequence of addresses
    - [A] 0xF303
    - [B] 0xF503
    - [C] 0xF563
    - [D] 0xEF63

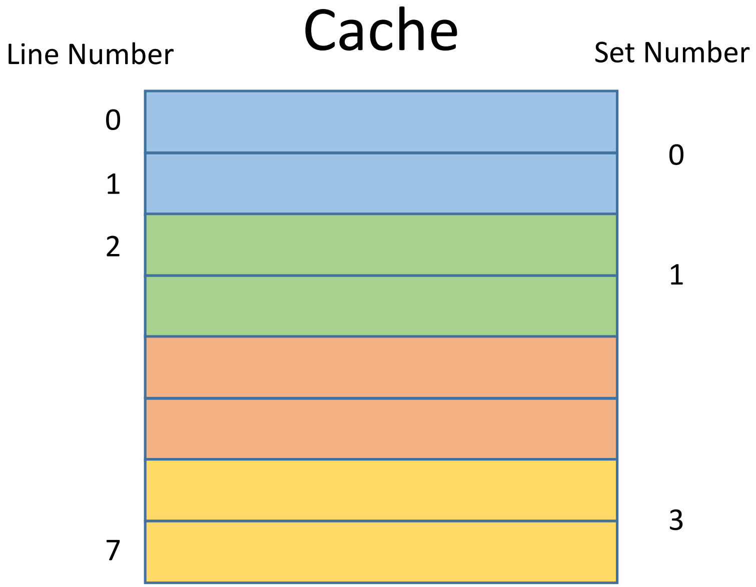
- What does the cache do?



2-Way Set-Associative Cache Quiz

- Given:
  - Byte addressable cache
  - 32 Byte blocks
  - Sequence of addresses
    - [A] 0xF303
    - [B] 0xF503
    - [C] 0xF563
    - [D] 0xEF63

- What does the cache do?



Offset: **5 bits** to address 32 Byte blocks

Index bits: **2 bits** to address 4 sets

Tag bits: 9 bits remaining

Fully-Associative Cache

- **Block Offset** is the same as before!

- We do not need **index bits**!

- The **tag** is the size of the block offset!

Review

- A **block** is composed of Bytes
    - Block size given in problem statements
    - A block is the smallest unit of data we can pull from memory
- **Sets** are composed of (1 or more) blocks
    - Number of sets = number of blocks / number of ways
    - A block of memory goes in a cache line
- A **cache** is composed of sets

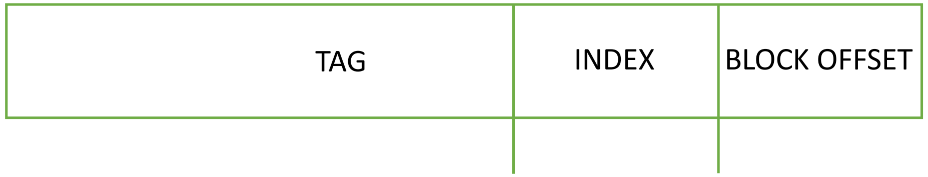
Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA

Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA

ADDRESS of location processor wants us to find in the cache



TAG

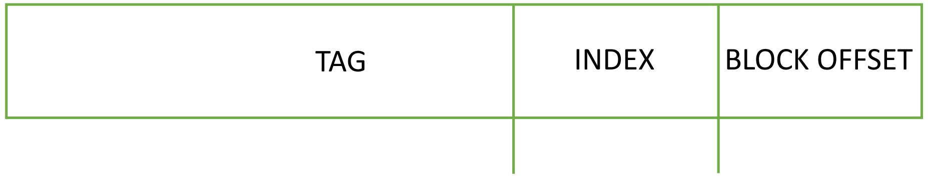
INDEX

BLOCK OFFSET

Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA

ADDRESS of location processor wants us to find in the cache



TAG

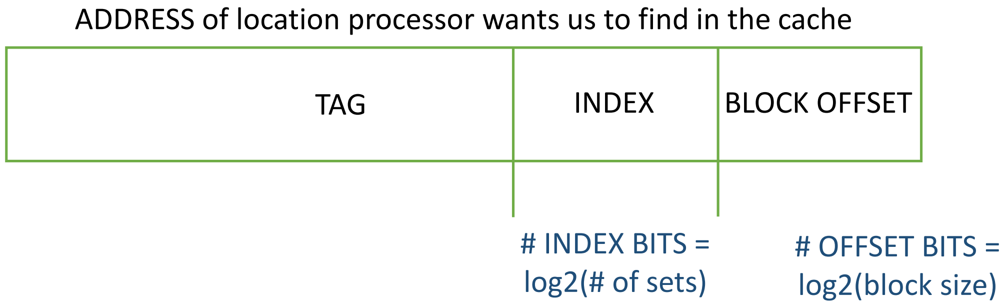
INDEX

BLOCK OFFSET

# OFFSET BITS =
log2(block size)

Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA



ADDRESS of location processor wants us to find in the cache

# INDEX BITS = log2(# of sets)

# OFFSET BITS = log2(block size)

Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA

ADDRESS of location processor wants us to find in the cache

TAG	INDEX	BLOCK OFFSET

# INDEX BITS =
log2(# of sets)

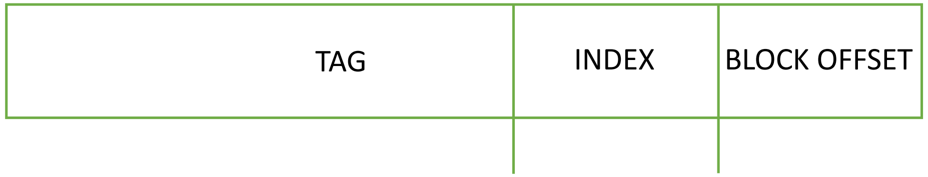
# OFFSET BITS =
log2(block size)

# of sets =
# cache lines / # ways

Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA

ADDRESS of location processor wants us to find in the cache



# INDEX BITS =
log2(# of sets)

# OFFSET BITS =
log2(block size)

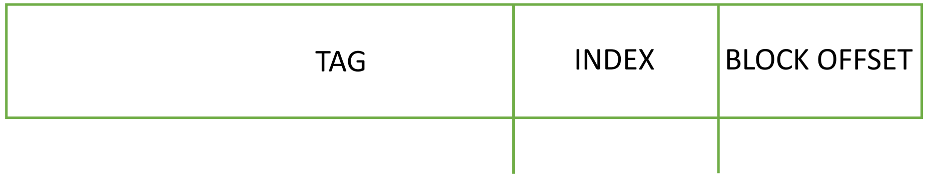
# of sets =
# cache lines / # ways

# cache lines =
cache size / block size

Recap

- Direct Mapped is 1-way SA
- Fully Associative is N-way SA

ADDRESS of location processor wants us to find in the cache



TAG

INDEX

BLOCK OFFSET

# TAG BITS
= # ADDR_BITS – # INDEX BITS - # OFFSET BITS

# INDEX BITS =
log2(# of sets)

# OFFSET BITS =
log2(block size)

# of sets =
# cache lines / # ways

# cache lines =
cache size / block size

References

- Patterson and Hennessy, Computer Organization and Design
- David Black-Schaffer