Lecture 17

- Today:
  - More caches

How big is the cache?

Suppose we have a byte-addressable machine with 16-bit addresses with a cache with the following characteristics:

- It is direct-mapped
- Each block holds one byte
- The cache index is the four least significant bits

Two questions:

- How many blocks does the cache hold?

- How many bits of storage are required to build the cache (*e.g.*, for the data array, tags, etc.)?

More cache organizations

Now we'll explore some alternate cache organizations.

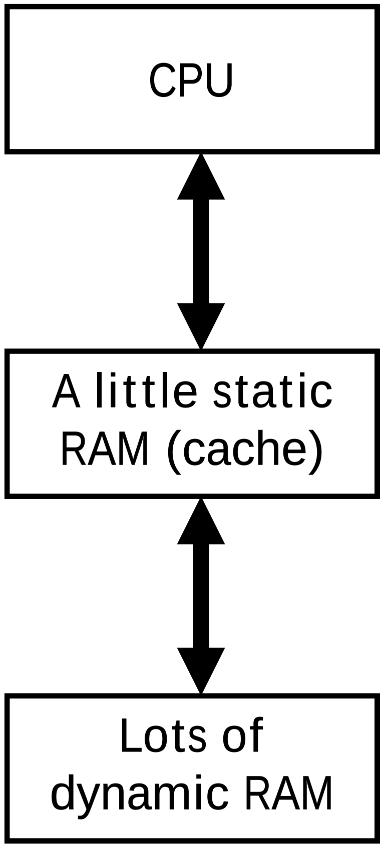
— How can we take advantage of spatial locality too?

— How can we reduce the number of potential conflicts?

- We'll first motivate it with a brief discussion about cache performance.

Memory System Performance

- To examine the performance of a memory system, we need to focus on a couple of important factors.
  - How long does it take to send data from the cache to the CPU?
  - How long does it take to copy data from memory into the cache?
  - How often do we have to access main memory?
- There are names for all of these variables.
  - The hit time is how long it takes data to be sent from the cache to the processor. This is usually fast, on the order of 1-3 clock cycles.
  - The miss penalty is the time to copy data from main memory to the cache. This often requires dozens of clock cycles (at least).
  - The miss rate is the percentage of misses.



Average memory access time

- The average memory access time, or AMAT, can then be computed.

  AMAT = Hit time + (Miss rate x Miss penalty)

  This is just averaging the amount of time for cache hits and the amount of time for cache misses.
- How can we improve the average memory access time of a system?
  - Obviously, a lower AMAT is better.
  - Miss penalties are usually much greater than hit times, so the best way to lower AMAT is to reduce the miss penalty *or* the miss rate.
- However, AMAT should only be used as a general guideline. Remember that execution time is still the best performance metric.

Performance example

- Assume that 33% of the instructions in a program are data accesses. The cache hit ratio is 97% and the hit time is one cycle, but the miss penalty is 20 cycles.

AMAT = Hit time + (Miss rate x Miss penalty)
=
=

- How can we reduce miss rate?

Performance example

- Assume data accesses only. The cache hit ratio is 97% and the hit time is one cycle, but the miss penalty is 20 cycles.

$$\begin{aligned}\text{AMAT} &= \text{Hit time} + (\text{Miss rate} \times \text{Miss penalty}) \\ &= 1 \text{ cycle} + (3\% \times 20 \text{ cycles}) \\ &= 1.6 \text{ cycles}\end{aligned}$$

- If the cache was perfect and never missed, the AMAT would be one cycle. But even with just a 3% miss rate, the AMAT here increases 1.6 times!

- How can we reduce miss rate?

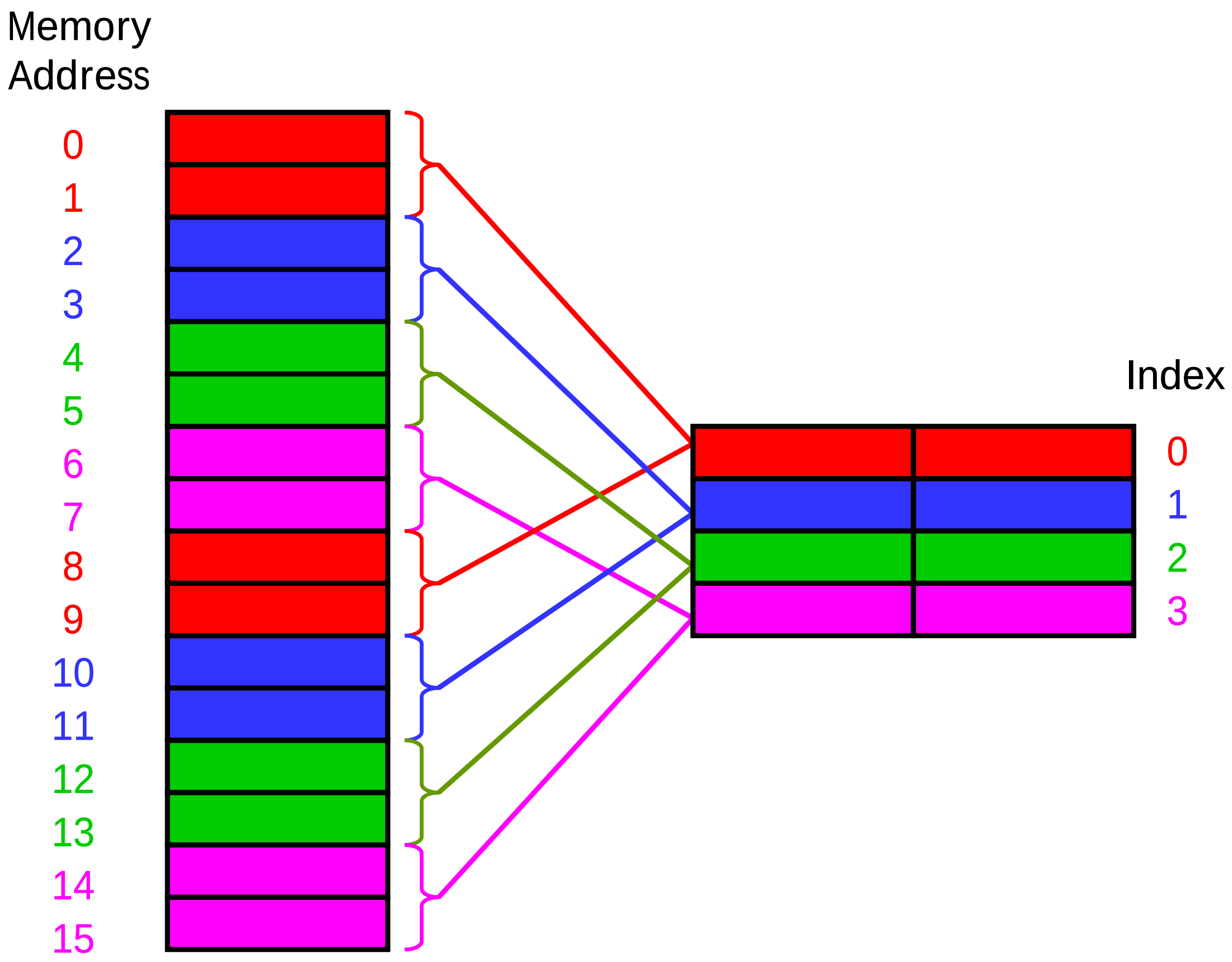
Spatial locality

- One-byte cache blocks don't take advantage of spatial locality, which predicts that an access to one address will be followed by an access to a nearby address.
- What can we do?

Spatial locality

- What we can do is make the cache block size larger than one byte.

- Here we use two-byte blocks, so we can load the cache with two bytes at a time.
- If we read from address 12, the data in addresses 12 *and* 13 would both be copied to cache block 2.

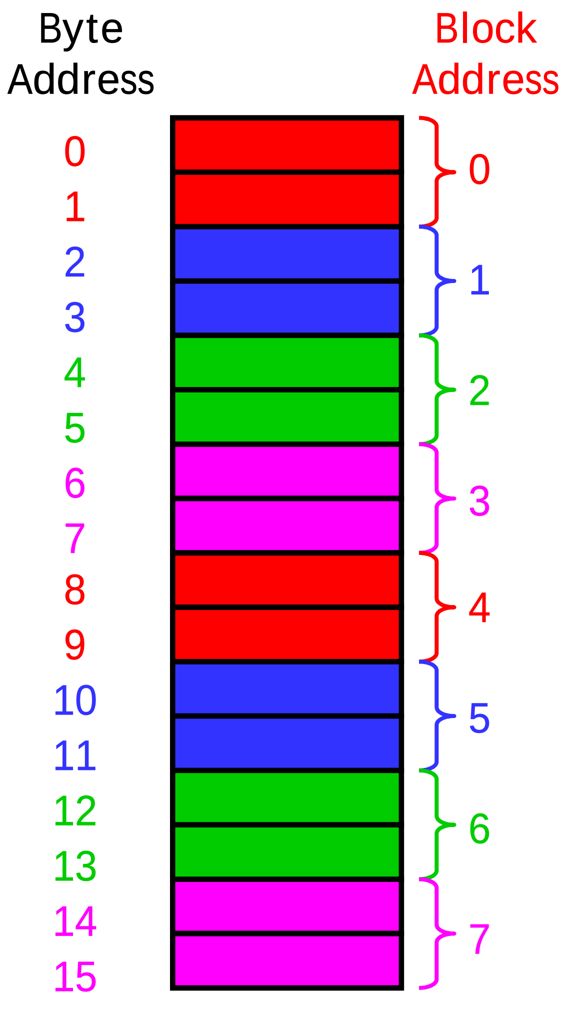


Block addresses

- Now how can we figure out where data should be placed in the cache?
- It's time for block addresses! If the cache block size is 2^n bytes, we can conceptually split the main memory into 2^n-byte chunks too.
- To determine the block address of a byte address *i*, you can do the integer division

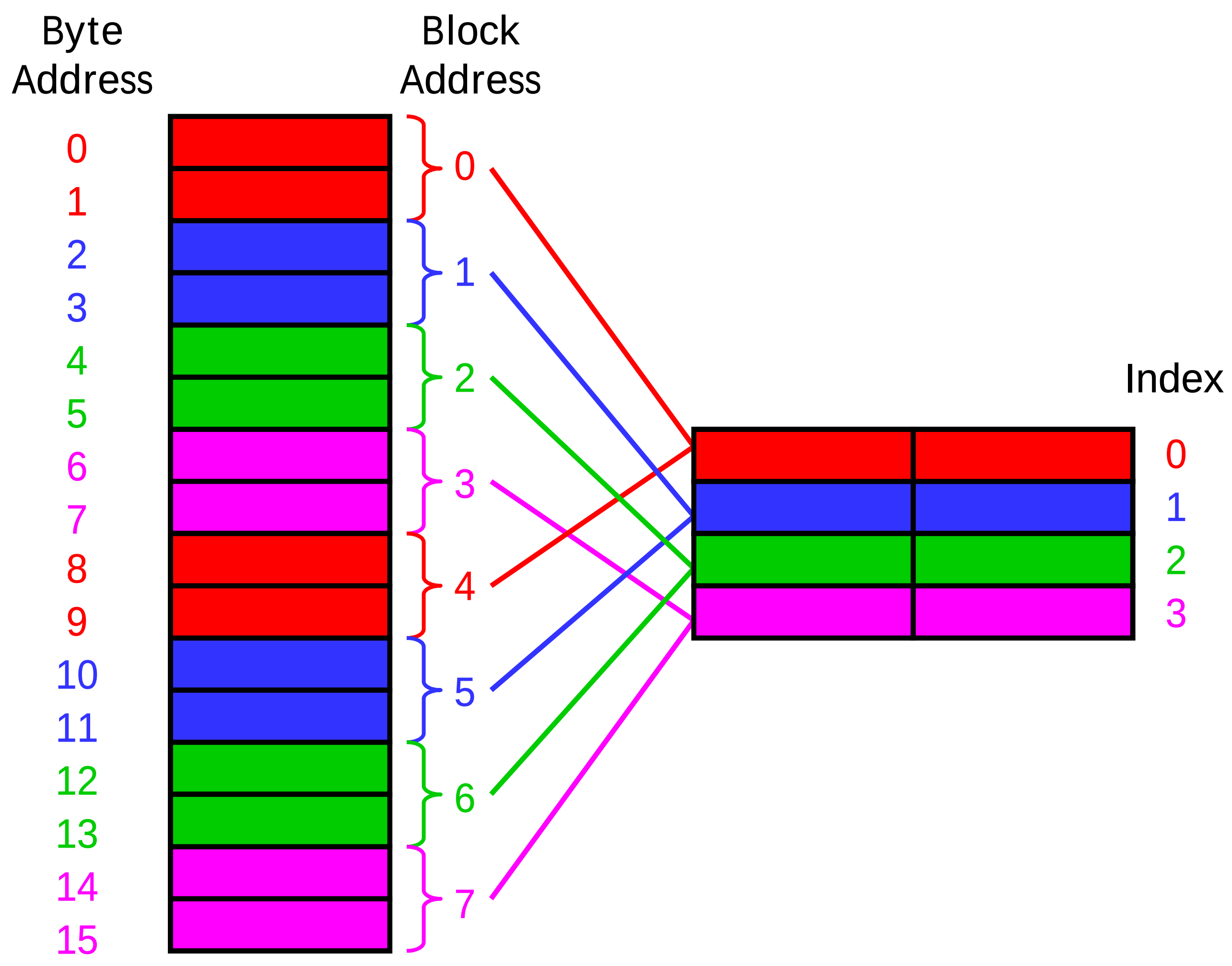
$$i\ /\ 2^n$$

- Our example has two-byte cache blocks, so we can think of a 16-byte main memory as an "8-block" main memory instead.
- For instance, memory addresses 12 and 13 both correspond to block address 6, since 12 / 2 = 6 and 13 / 2 = 6.



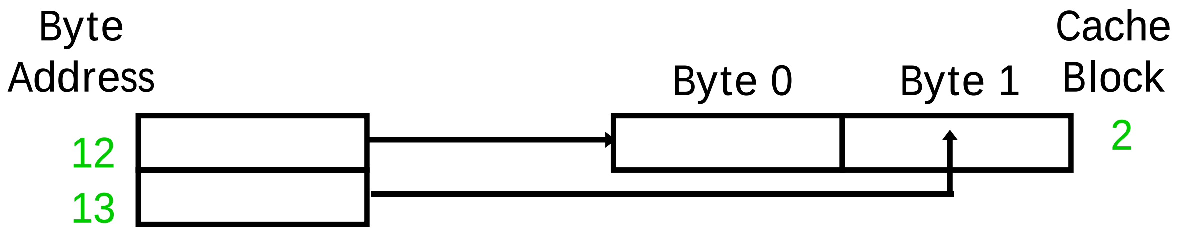
Cache mapping

- Once you know the block address, you can map it to the cache as before: find the remainder when the block address is divided by the number of cache blocks.
- In our example, memory block 6 belongs in cache block 2, since 6 mod 4 = 2.
- This corresponds to placing data from memory *byte* addresses 12 and 13 into cache block 2.



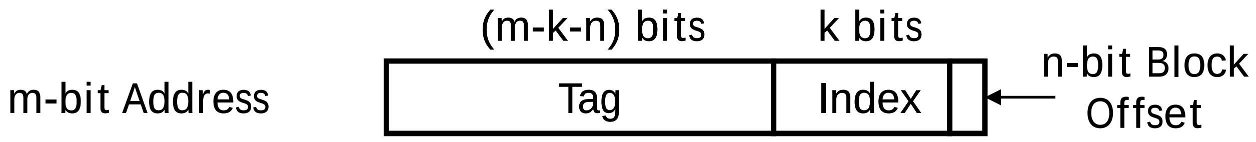
Data placement within a block

- When we access one byte of data in memory, we'll copy its entire *block* into the cache, to hopefully take advantage of spatial locality.
- In our example, if a program reads from byte address 12 we'll load all of memory block 6 (both addresses 12 and 13) into cache block 2.
- Note byte address 13 corresponds to the *same* memory block address! So a read from address 13 will also cause memory block 6 (addresses 12 and 13) to be loaded into cache block 2.
- To make things simpler, byte *i* of a memory block is always stored in byte *i* of the corresponding cache block.

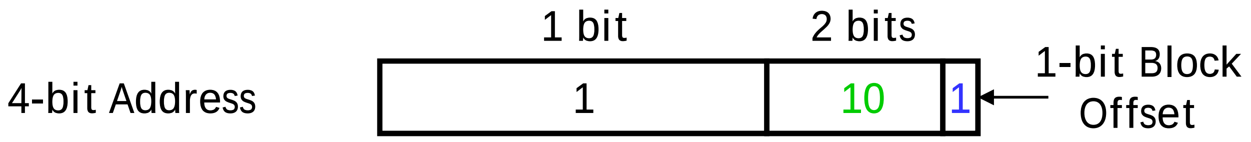


Locating data in the cache

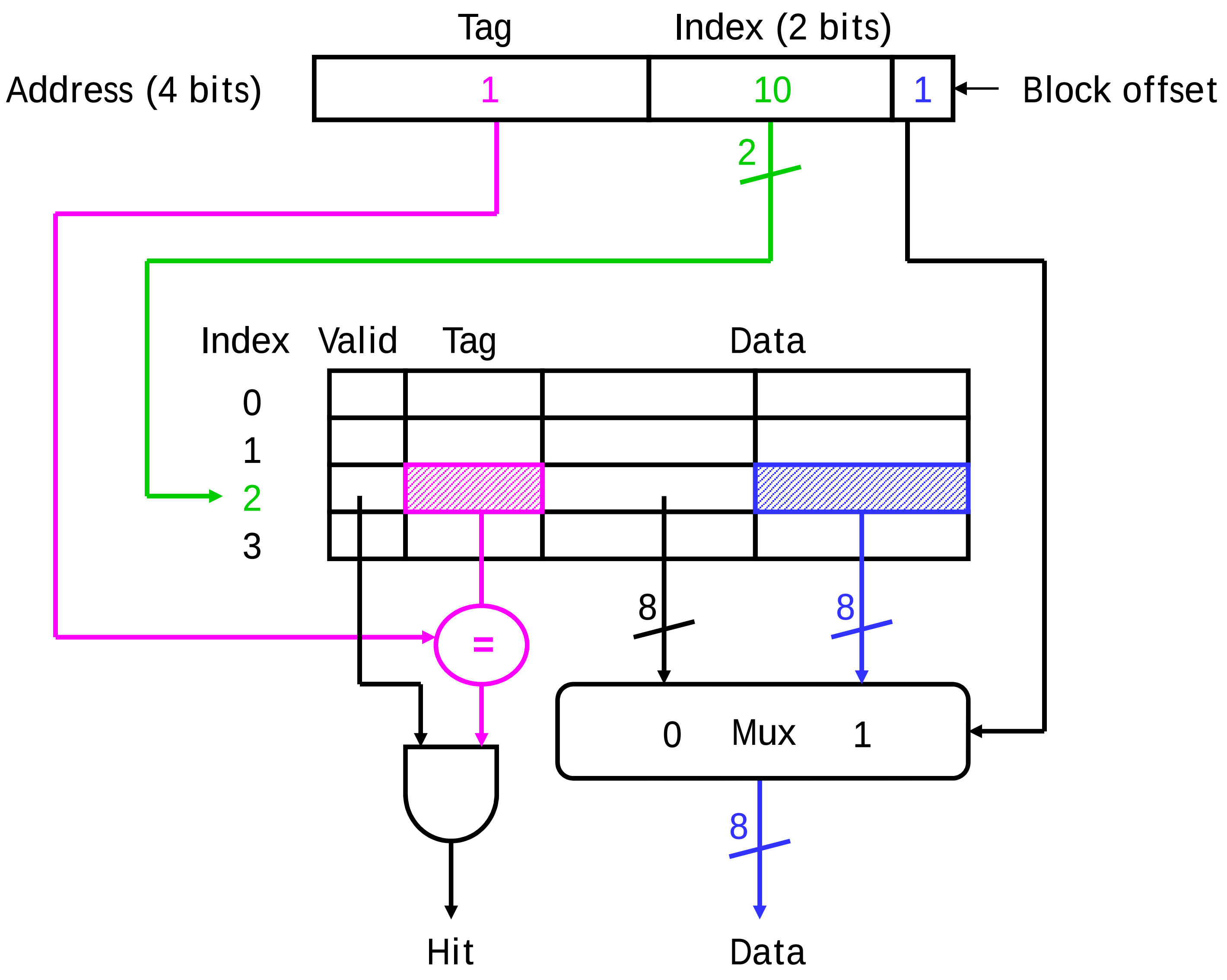
- Let's say we have a cache with 2^k blocks, each containing 2^n bytes.
- We can determine where a byte of data belongs in this cache by looking at its address in main memory.
  - *k* bits of the address will select one of the 2^k cache blocks.
  - The lowest *n* bits are now a block offset that decides which of the 2^n bytes in the cache block will store the data.



- Our example used a 2^2-block cache with 2^1 bytes per block. Thus, memory address 13 (1101) would be stored in byte 1 of cache block 2.

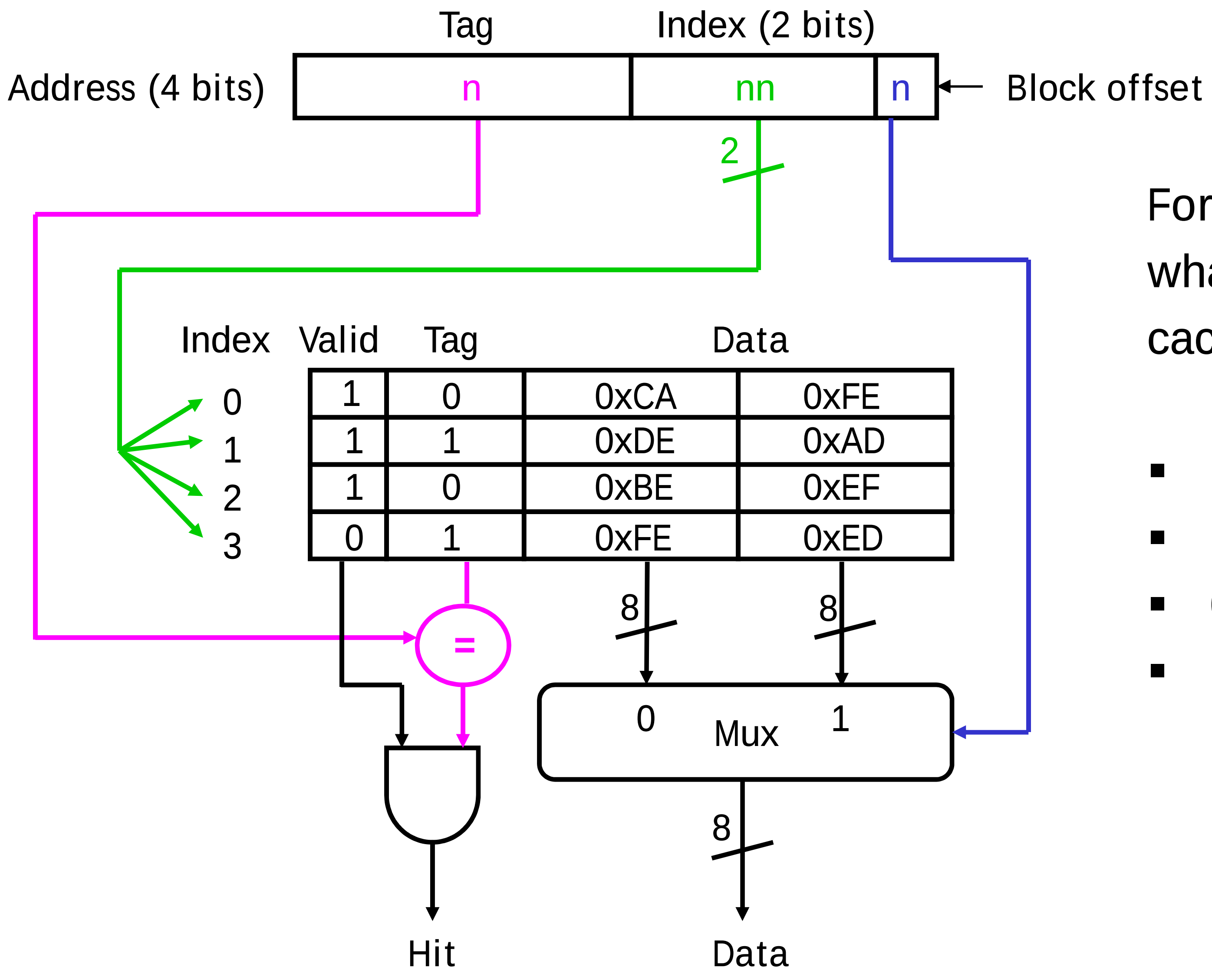


A picture



Tag
Index (2 bits)
Address (4 bits)
1
10
1
Block offset
2
Index
Valid
Tag
Data
0
1
2
3
8
8
=
0
Mux
1
8
Hit
Data

An exercise

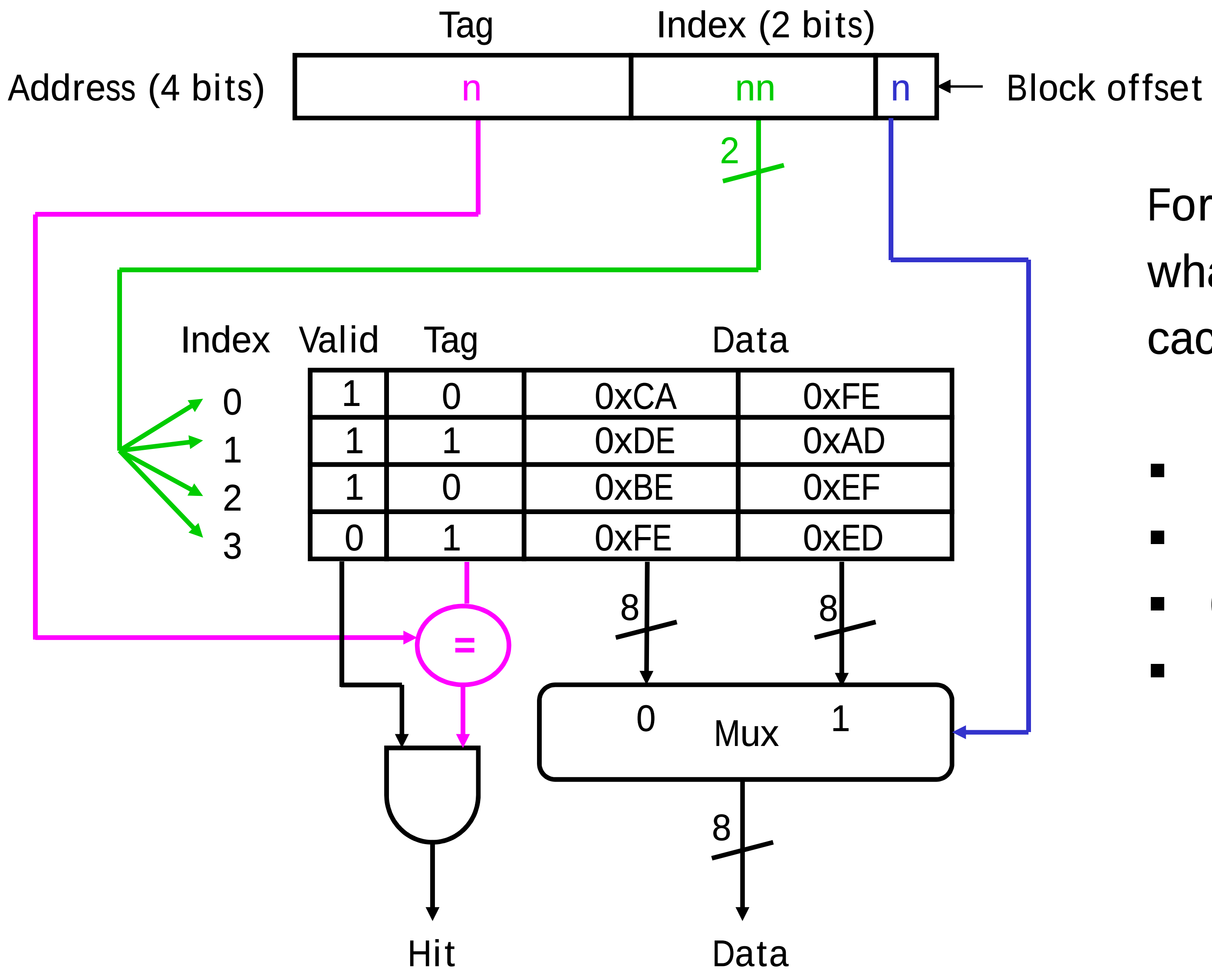


Index	Valid	Tag	Data	
0	1	0	0xCA	0xFE
1	1	1	0xDE	0xAD
2	1	0	0xBE	0xEF
3	0	1	0xFE	0xED

For the addresses below, what byte is read from the cache (or is there a miss)?

- 1010
- 1110
- 0001
- 1101

An exercise



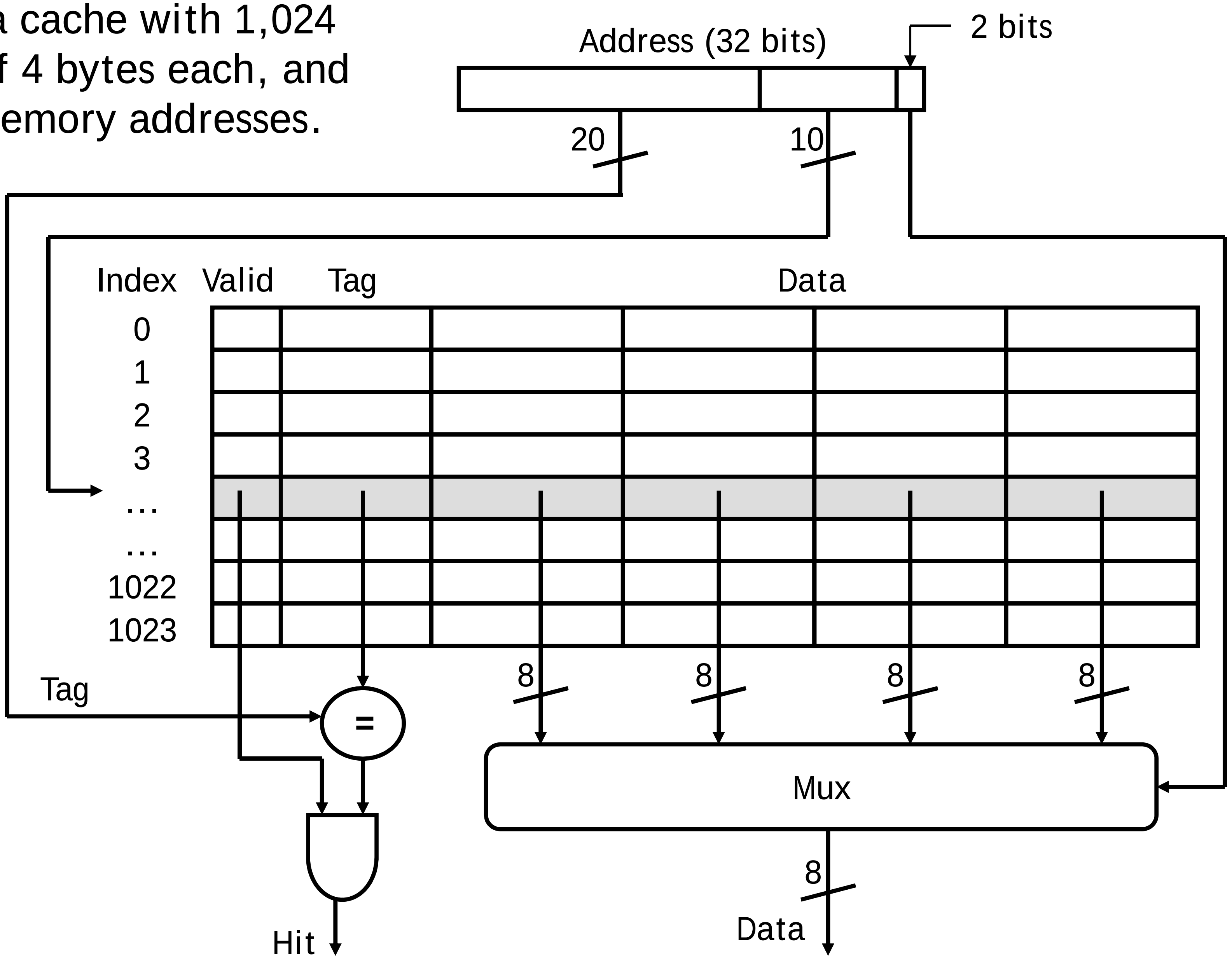
Index	Valid	Tag	Data	
0	1	0	0xCA	0xFE
1	1	1	0xDE	0xAD
2	1	0	0xBE	0xEF
3	0	1	0xFE	0xED

For the addresses below, what byte is read from the cache (or is there a miss)?

- 1010 (0xDE)
- 1110 (miss, invalid)
- 0001 (0xFE)
- 1101 (miss, bad tag)

A diagram of a larger example cache

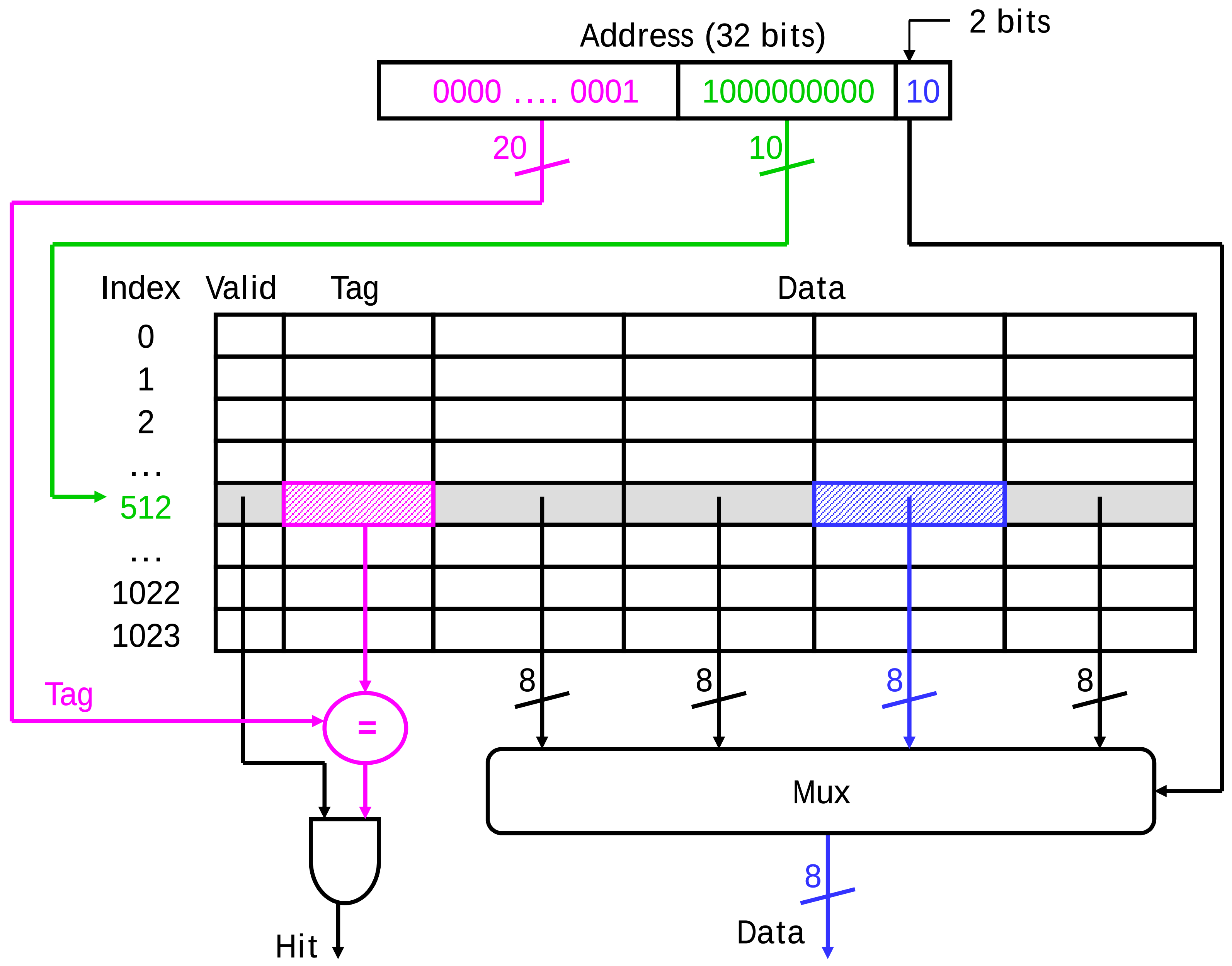
- Here is a cache with 1,024 blocks of 4 bytes each, and 32-bit memory addresses.



A larger example cache mapping

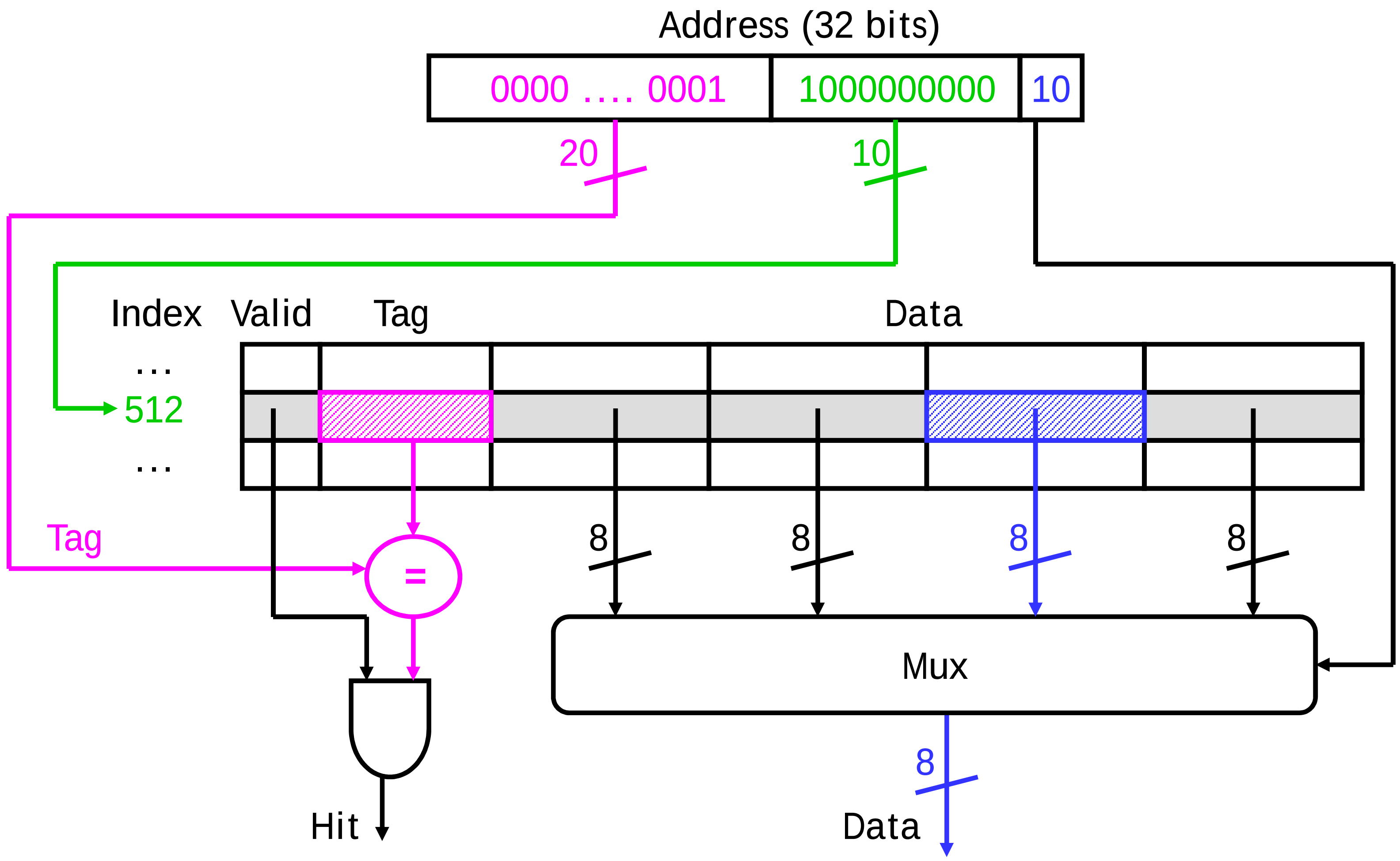
- Where would the byte from memory address 6146 be stored in this direct-mapped 2^{10}-block cache with 2^{2}-byte blocks?
- We can determine this with the binary force.
  - 6146 in binary is 00...01 1000 0000 00 10.
  - The lowest 2 bits, 10, mean this is the second byte in its block.
  - The next 10 bits, 1000000000, are the block number itself (512).
- Equivalently, you could use arithmetic instead.
  - The block offset is 6146 mod 4, which equals 2.
  - The block address is 6146/4 = 1536, so the index is 1536 mod 1024, or 512.

A diagram of a larger example cache mapping



What goes in the rest of that cache block?

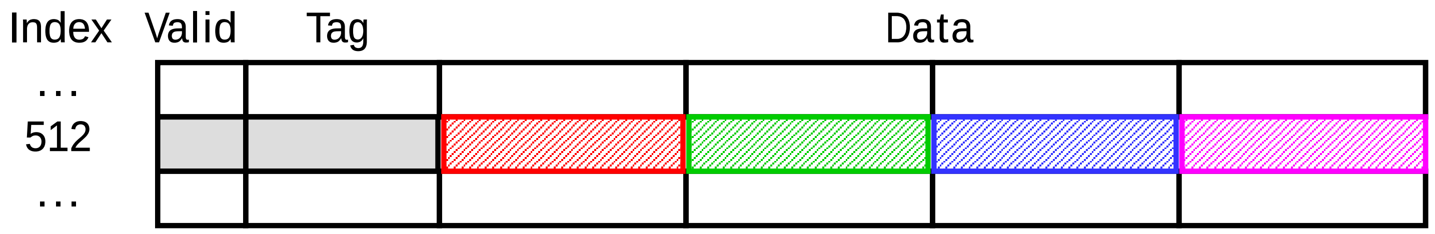
- The other three bytes of that cache block come from the same memory block, whose addresses must all have the same index (1000000000) and the same tag (00...01).



The rest of that cache block

- Again, byte *i* of a memory block is stored into byte *i* of the corresponding cache block.
  - In our example, memory block 1536 consists of byte addresses 6144 to 6147. So bytes 0-3 of the cache block would contain data from address 6144, 6145, 6146 and 6147 respectively.
  - You can also look at the lowest 2 bits of the memory address to find the block offsets.

Block offset	Memory address	Decimal
00	00..01 1000000000 00	6144
01	00..01 1000000000 01	6145
10	00..01 1000000000 10	6146
11	00..01 1000000000 11	6147


Index
Valid
Tag
Data
...
512
...