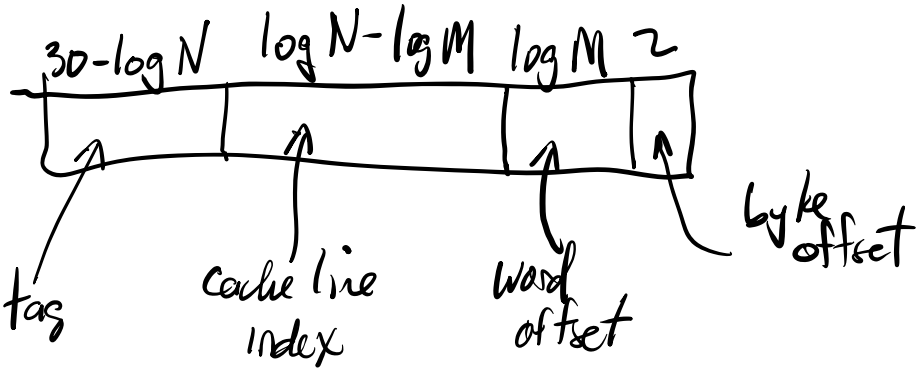
Where we left off:

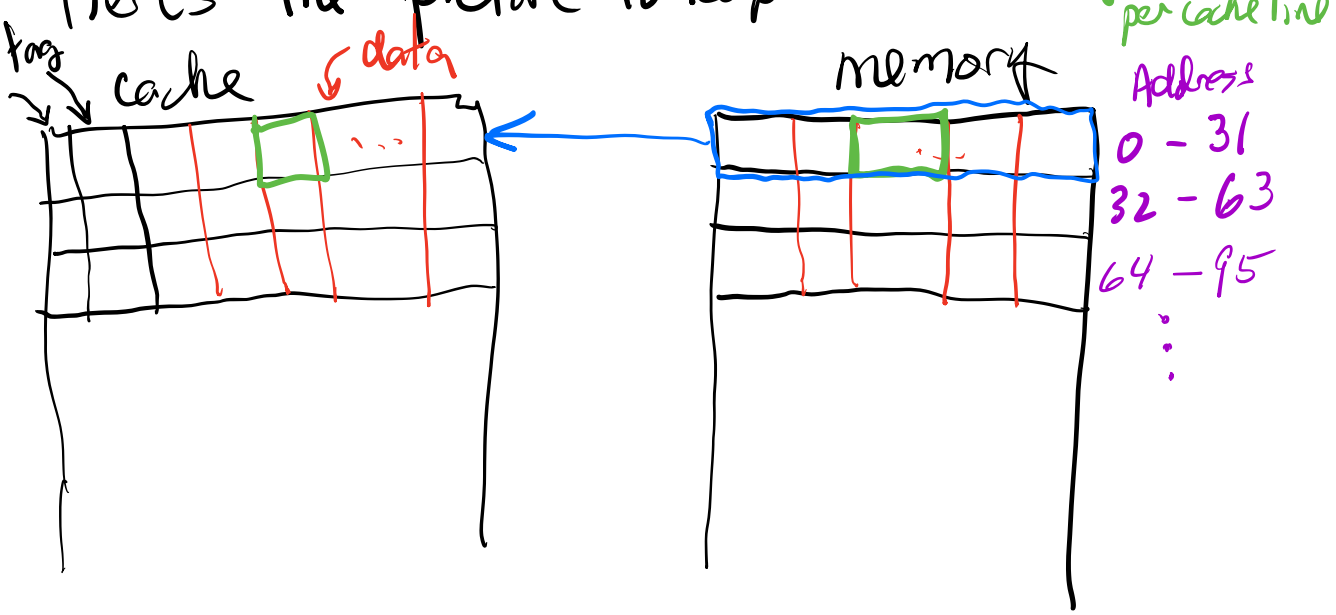
- Multi-word cache line
  - M words/line
  - N total words in the cache
  - N/M cache lines in the cache

Address: (32 bits)



Here's the picture to keep in mind:

Assume 8 words = 32B per cache line



When writing to one word in a cache line, the entire cache line must be brought into the cache.

Cache Performance

Why is caching so important?

Let's look at a simple model of a running program.

- assume a single cache,
- where a cache hit makes the data or instruction available immediately (no delay)
- and a cache miss incurs a 100 cycle penalty
  - processor waits for 100 cycles
  - for both instructions & data

Assume also that:

- a program executes I instructions.

Under an ideal case (impossible), where every memory access is a cache hit, and the processor can execute an instruction every cycle, the program will take I cycles to execute.

"perfect cache"

Under more realistic, but good, cache performance:

- 2% cache miss rate for instructions (98% cache hit rate for instructions)
- 4% cache miss rate for data (96% cache hit rate)

Not all instructions access data in memory

- Assume 35% of instructions access data in memory.

How long does the program take to execute (in cycles)?

$$\text{total cycles} = I + (0.02 \times I \times 100)$$

0.02: instr. cache miss rate; I: # of instructions; 100: cache miss penalty

$$+ (0.04 \times 0.35 \times I \times 100)$$

0.04: data cache miss rate; 0.35: # of instructions that access memory; 100: cache miss penalty

$$= I + 2I + 1.4I$$

$$= 4.4I$$

So, the program is 4.4 times slower than in the ideal case.

What if you double the speed of the processor, without changing the size of the cache and how long a cache miss takes?

- so a cycle in the new processor only takes 1/2 as long as a cycle in the old processor.
- memory hasn't speeded up, so the same delay is now 200 cycles in the new processor.

Total number of cycles in the new processor

$= I + (0.02 \times I \times 200)$
$+ (0.04 \times .35 \times I \times 200)$

$= I + 4I + 2.8I$

$= 7.8I$ cycles in the new processor

$= 3.9I$ in the old processor.

So, the speedup due to doubling the speed of the processor is

$$\frac{4.4I}{3.9I} = 1.13$$

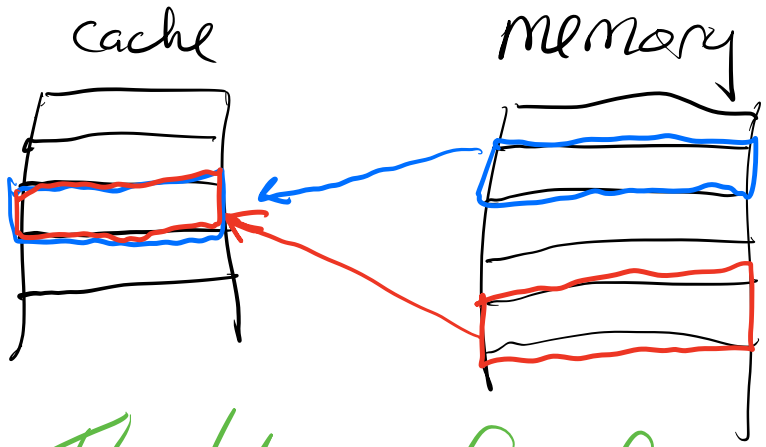
13% speedup!

What causes cache misses?

① "Compulsory" miss
- a cache line won't be in the cache the first time it is accessed.

② "Capacity" miss:
- the cache isn't big enough to hold all the instructions + data the program is currently using.

③ "Conflict" misses:
- multiple cache lines in memory that are currently being used map to the same cache entry.



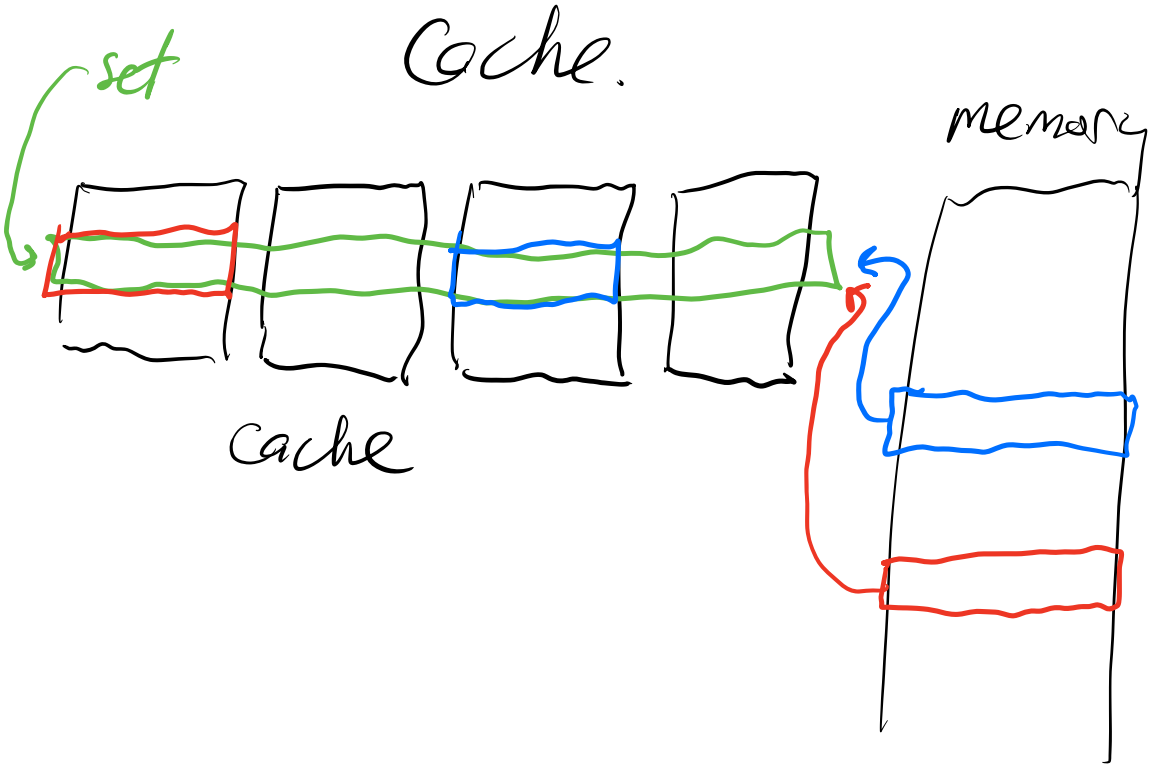
The blue and red cache lines can't both be in the cache at the same time.
- even if there is plenty of room in the cache.

To solve conflict misses, we'd like some flexibility about where to put in the cache a cache line from memory.

Solution: Use a "set associative cache"
- allows a cache line in memory to be brought into one of a set of possible cache entries.

"n-way set associative cache" allows a choice of one of a set of n cache entries in which to put a cache line.

Example: 4-way set associative cache.



How do we use the bits of an address to specify which set a cache line in memory is mapped to?

Assume:

N words in the cache

M words per cache line

so N/M cache entries in the cache

Assume also a n-way set associative cache

- so the number of sets is the number of cache entries divided by the size of each set

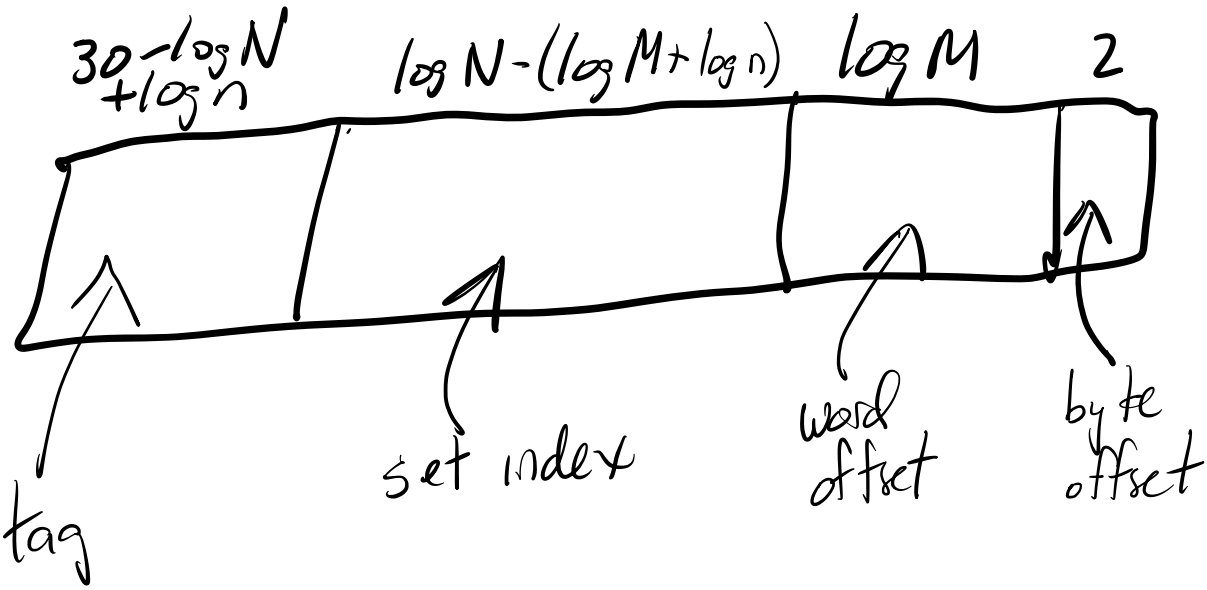
$$\#sets = (N/M) \div n$$

$$= N/(M * n)$$

So, the number of bits in an address needed to select one of $N/(M \times n)$ sets is

$$\log\left(N/(M \times n)\right)$$

$$= \log N - (\log M + \log n)$$



Example:

Assume: $M = 8$ words/line

$N = 256\text{KB} = 64\text{K}$ words

$= 2^{16}$ words

$n = 4$ (4-way cache)