- Effective memory access time - all access times to caches and memory added together
    - Tmm = 50 ns, 10% of accesses are main memory
    - Tcache = 5 ns, 90% of accesses are cache
    - 50ns * .1 + 5ns * .9 = 5ns + 4.5ns = 9.5 ns = effective memory access time
- Direct Mapping Cache
    - Maps each block to a specific line of the cache
    - Example - 32-bit address, block size of 8 words, 1024 line cache
        - Come up with the tag, line, and word ID
        - Because block size is 8, 3 bits are needed for the word ID
        - Line # - 10 bits because 1024 lines are needed
        - Because the address is 32-bits long, there are 19 bits left over for the tag
    - Where does address 0x3534AB92 lie?
        - 00111001001101001010|||0101110010|||010
        - Tag ||| Line # ||| Word ID
    - Because the routing is fixed, the same number of bits in an address are always sent the same way
    - XOR gate - 0 = bits equal, 1 = bits different
    - If different tags share the same line number, a problem of thrashing can happen
- Fully Associative Cache
    - Only tag and word ID exist, line # goes away
    - Allows any block to be stored in any line of the cache
    - Slowest type of cache and the most expensive
    - Thrashing does not exist
- Set Associative Cache
    - Groups lines of the cache into sets, each with K lines (k-way set associative)
    - Each block of memory is assigned or can be found in exactly one set
    - A balance between Direct Mapping and Fully Associative
    - It essentially breaks up one cache into many fully associative caches
    - Cheaper than Fully Associative, but not as fast as Direct Mapping
    - Tag ||| Set # ||| Word ID
    - Example
        - 4-way set associative - each set has 4 lines
        - Still have 3 bits for the word ID
        - # of sets = 1024 = # of ways * # of sets
        - Number of sets = 8 bits
        - That means the tag becomes 21 bits
- Discarding Data from the Cache
    - Least frequently used - throw out block that has been used the least
        - Counter is needed to know which one is least used
        - Problem is something was just loaded and then discarded
    - First in First out - throw out the oldest block
        - Timer is needed to know which one has been in the longest
        - Problem - something frequently used and old is discarded
    - Least Recently Used - throw out block that has been hit in the longest time
        - Timer is required, or a flag is required
        - Likely the best performer and the easiest to implement
    - Random - throw out at random
        - Worst performer but easiest and cheapest to implement