To identify the block currently in a slot of the cache, each cache entry contains a tag value. Thus, if slot zero in the cache contains tag K, the value in slot zero corresponds to block zero from the area of memory that has tag K.

Why use tags? A cache must uniquely identify the entry in a slot. Because a tag identifies a large group of blocks rather than a single byte of memory, using a tag requires fewer bits to identify a section of memory than using a full memory address. Furthermore, as the next section explains, choosing the block size and the size of memory identified by a tag to be powers of two makes cache lookup extremely efficient.

12.22 Using Powers Of Two For Efficiency

Although the direct mapping described above may seem complex, using powers of two simplifies the hardware implementation. In fact, the hardware is elegant and extremely efficient. Instead of modulo arithmetic, both the tag and block number can be computed by extracting groups of bits from a memory address. The high-order bits of the address are used as the tag, the next set of bits forms a block number, and the final set of bits gives a byte offset within the block. Figure 12.9 illustrates the division.

Figure 12.9 Illustration of how using powers of two allows a cache to divide a memory address into three separate fields that correspond to a tag, a block number, and a byte offset within the block.

Once we know that all values can be obtained via bit extraction, the algorithm for lookup in a direct-mapped memory cache is straightforward. Think of the cache as an array. The idea is to extract the block number from the address, and then use the block number as an index into the array. Each entry in the array contains a tag and a value. If the tag in the address matches the tag in the cache slot, the cache returns the value. If the tag does not match, the cache hardware must fetch the block from memory, place a copy in the cache, and then return the value. Algorithm 12.1 summarizes the steps.

The algorithm omits an important detail. Each slot in the cache has a *valid bit* that specifies whether the slot has been used. Initially (i.e., when the computer boots), all valid bits are set to 0 (to indicate that none of the slots contain blocks from memory). When it stores a block in a slot, the cache hardware sets the valid bit to 1. When it examines the tag for a slot, the hardware reports a mismatch if the valid bit is not set, which forces a copy of the block to be loaded from memory.