line is *clean* or *dirty.* When the line is eventually flushed, dirty lines must be written back to main memory in their entirety. Clean lines can be flushed without further action. While a write-back cache cannot absolutely eliminate the longer write latency of main memory, it can reduce the overall system impact of writes, because the microprocessor can perform any number of writes to the same cache line, and only a fixed write-back penalty results upon a flush.

The central problem in designing a cache is how to effectively hold many scattered blocks from a large main memory in a small cache memory. In a standard desktop PC, main memory may consist of 256 MB of DRAM, whereas the microprocessor's cache is 256 kB—a difference of three orders of magnitude! The concept of cache lines provides a starting point with a defined granularity to minimize the problem somewhat. Deciding on a 16-byte line size, for example, indicates that a 32-bit address space needs to be handled only as 2^{28} units rather than 2^{32} units. Of course, 2^{28} is still a very large number! Each cache line must have an associated tag and/or index that identifies the higher-order address bits that its contents represent (28 bits in this example). Different cache architectures handle these tags and indices to balance cache performance with implementation expense. The three standard cache architectures are *fully associative*, *direct mapped*, and *n-way set associative*.

A fully associative cache, shown in Fig. 7.2, breaks the address bus into two sections: the lower bits index into a selected cache line to select a byte within the line, and the upper bits form a tag that is associated with each cache line. Each cache line contains a valid bit to indicate whether it contains real data. Upon reset, the valid bits for each line are cleared to 0. When a cache line is loaded with data, its tag is set to the high-order address bits that are driven by the microprocessor. On subsequent transactions, those address bits are compared in parallel against every tag in the cache. A hit occurs when one tag matches the requested address, resulting in that line's data advancing to a final multiplexer where the addressed bytes are selected by the low-order address bits. A fully associative cache is the most flexible type, because any cache line can hold any portion of main memory. The disadvantage of this scheme is its complexity of implementation. Each line requires address match-

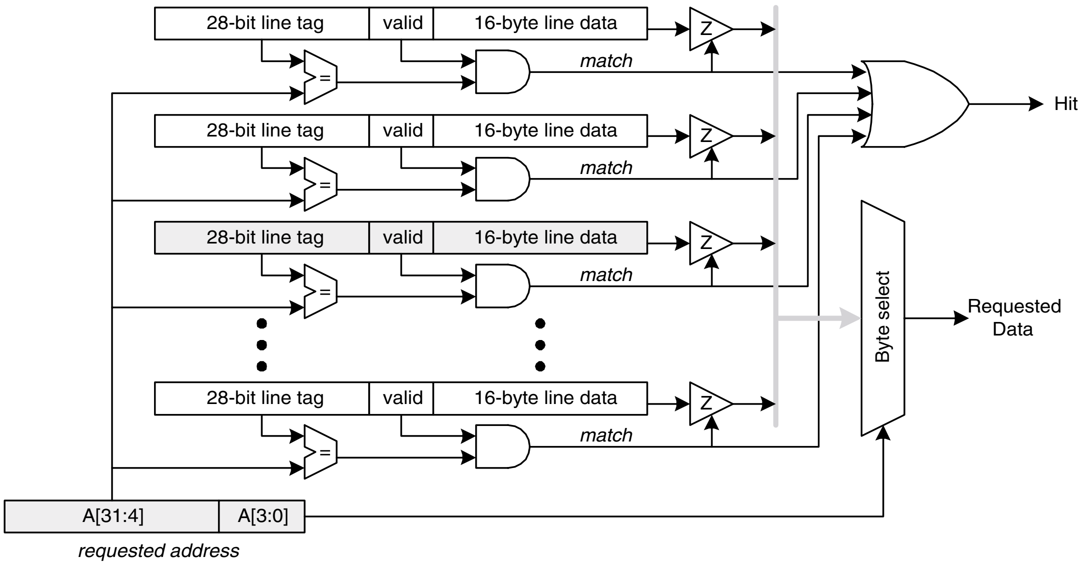

FIGURE 7.2 Fully associative cache.

ing logic, and each match signal must be logically combined in a single location to generate a final hit/miss status flag.

A direct mapped cache, shown in Fig. 7.3, breaks the address bus into three sections; the lower bits retain their index function within a selected line, the middle bits select a single line from an array of uniquely addressable lines, and the upper bits form a tag to match the selected cache line. As before, each cache line contains a valid bit. The difference here is that each block of memory can only be mapped into one cache line—the one indexed by that block's middle address bits, A[15:4] in this example (indicating a 64-kB total cache size). During a cache miss, the controller determines which line is selected by the middle address bits, loads the line, sets the valid bit, and loads the line tag with the upper address bits. On subsequent accesses, the middle address bits select a single line whose tag is compared against the upper address bits. If they match, there is a cache hit. A direct mapped cache is much easier to implement as compared to a fully associative cache, because parallel tag matching is not required. Instead, the cache can be constructed with conventional memory and logic components using off-the-shelf RAM for both the tag and line data. The control logic can index into the RAM, check the selected tag for a match, and then take appropriate action. The disadvantage to a direct mapped cache is that, because of the fixed mapping of memory blocks to cache lines, certain data access patterns can cause rapid *thrashing*. Thrashing results when the microprocessor rapidly accesses alternate memory blocks. If the alternate blocks happen to map to the same cache line, the cache will almost always miss, because each access will result in a flush of the alternate memory block.

Given the simplicity of a direct mapped cache, it would be nice to strike a compromise between an expensive fully associative cache and a thrashing-sensitive direct mapped cache. The *n-way* set associative cache is such a compromise. As shown in Fig. 7.4, a two-way set associative cache is basically two direct mapped cache elements connected in parallel to reduce the probability of thrashing. More than two sets can be implemented to further reduce thrashing potential. Four-way and two-way set associative caches are very common in modern computers. Beyond four elements, the payback of thrashing avoidance to implementation complexity declines. The term *set* refers to the number of entries in each direct mapped element, 4,096 in this example. Here, the

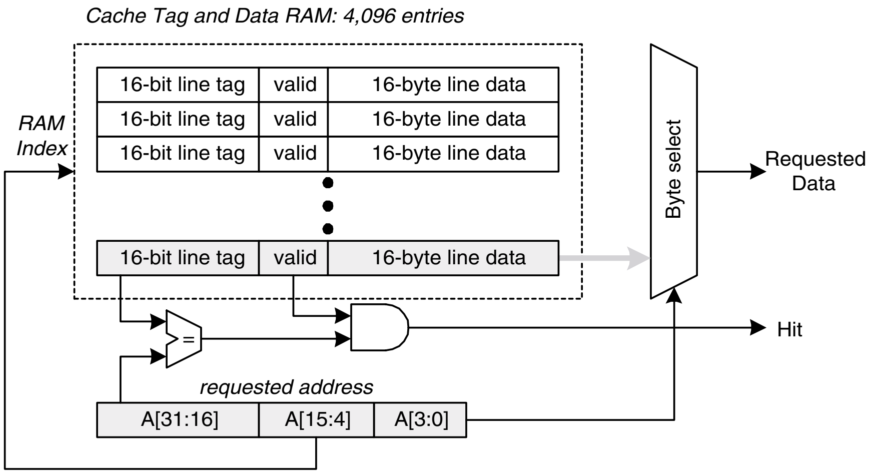

FIGURE 7.3 64-kB direct mapped cache.