associate a pair of tag bits with each actual tag bit. This two-bit structure allows the creation of a third “don’t care” state, X. A ternary CAM is more flexible than a binary CAM, because it can match portions of a tag rather than all bits. In networking applications, this is very useful, because similar operations are often performed on groups of addresses (tags) from common destinations. It is as if the post office wanted to sort out all letters being sent to ZIP codes 11230 through 11239. A ternary CAM would be able to match the pattern 1123X with a single entry. In contrast, a binary CAM would require ten redundant entries to perform the same job.

A ternary CAM is often used to implement rather complex lookup tables with searches prioritized according to the number of X bits in each tag. Using the ZIP code example, it is possible that a post office would want to perform two overlapping searches. It may want to sort all ZIP codes from 11230 through 11239 into a particular bin, except for 11234, which should be sorted into its own bin. A ternary CAM could be setup with two overlapping entries: 11234 and 1123X. To ensure that the 11234 entry always matched ahead of the 1123X entry, it would be necessary to verify proper setup of the specific CAM being used. A ternary CAM may have a rule that the lowest or highest winning entry in the array wins. While this example is simple, the concept can be extended with many levels of overlap and priority.

Managing a ternary CAM with overlapping entries is more complex than managing a binary CAM, because the winning entry priority must be kept in sync with the application’s needs, even as the CAM is updated during operation. A CAM is rarely initialized once and then left alone for the remainder of system operation. Its contents are modified periodically as network traffic changes. Let’s say that the ZIP code CAM was initialized as follows in consecutive entries: 1121X, 11234, 1123X, 112XX. Where would a new special-case entry 11235 be placed? It would have to precede the 1123X entry for it to match before 1123X. Therefore, the system would have to temporarily move CAM entries to insert 11235 into the correct entry. If there is enough free space in the CAM, the system could initialize it and reserve free entries in between valid entries. But, sooner or later, the CAM will likely become congested in a local area, requiring it to be reorganized. How the data is arranged and how the CAM is reorganized will affect system performance, because it is likely that the CAM will have to be temporarily paused in its search function until the reorganization is complete. Solutions to this pause include a multibank CAM architecture whereby the system reorganizes the lookup table in an inactive bank and then quickly swaps inactive and active banks.

A CAM often does not associate general data bits with each entry, because the main purpose of a CAM is to match tags, not to store large quantities of data. It is therefore common to couple a CAM with an external SRAM that actually holds the data of interest and that can be arbitrarily expanded according to application requirements as shown in Fig. 8.19. In this example, the CAM contains

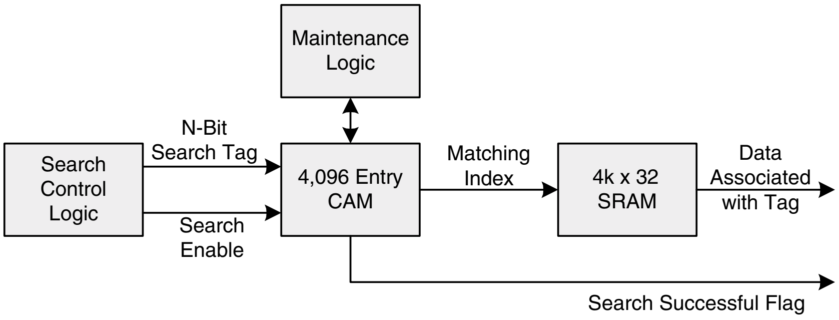

**FIGURE 8.19** CAM augmentation with external SRAM.

4,096 entries and returns a 12-bit index when a tag has been successfully matched. This index serves as the address of an SRAM that has a 32-bit data path as required by the application.

When combined with conventional memory and some control logic, a CAM subsystem is sometimes referred to as a *search engine*. A search engine is differentiated from a stand-alone CAM by being capable of semi-autonomous lookups on behalf of another entity such as data processing logic in either hardware or software. A search engine's control logic can be as simple as accepting a search tag and then returning data along with a success flag. It can get more complex to include specific table maintenance functions so that CAM overhead operations are completely offloaded from the data processing logic. Search engines are especially useful when interfacing with special-purpose *network processor* devices. These processors run software to parse packets and make decisions about how each packet should be handled in the system. The tag lookup function is offloaded to a search engine when there is not enough time for a software algorithm to search a large table.