Input register	1001	0011
Masking register	0011	1100
Work 1 in CAM	0011	1100
Word 2 in CAM	1001	0010
Word 3 in CAM	1101	0000

Figure 2. Example of CAM contents.

CONTENT-ADDRESSABLE STORAGE

Consider a list of student records being stored in a computer memory, each record containing a student's name, a student's ID number, a phone number, and a grade. The instructor is trying to get a list of students' names with A grades. The conventional method of solving this task requires reading all records in the list sequentially and comparing all grades to A. For those records with grades equal to A, the corresponding names are identified. This implementation seems to be straightforward, but could be time-consuming. To speed up the search, one might suggest using additional hardware to perform comparisons simultaneously. The matched records could then be found with a shorter delay. Content-addressable memory (CAM) was, therefore, introduced to speed up many data-processing applications.

In a typical memory unit, an address is first specified and the corresponding content is then read. In CAM, the content can be identified for access by the data themselves, rather than by an address in read-only memory (ROM) or random-access memory (RAM). For the preceding example, if CAM is used to store student records, the search process is replaced by simultaneous comparisons of all students' grades to the grade A. Names of students with an A grade will then be selected and the process finished. To perform the comparisons in a parallel fashion, hardware for memory has to be modified and the cost of the memory is increased. For this reason, CAM is employed only when the search time is crucial, even though CAM can operate as a RAM wherein the address is used to get the corresponding word.

IMPLEMENTATION

A simple block diagram of CAM for m words with n bits per word is shown in Fig. 1. The masking register is used to specify a subfield of the comparand register to be chosen as the key. The contents of each word in CAM is chosen according to the masking register. The selected subfields of all the words in CAM are compared to the key simultaneously. For those words that match the key, the corresponding indicators will be set and the contents placed in the output register.

Consider an example with CAM contents as displayed in Fig. 2. From the bit pattern that appears in the masking register, only the middle 4 bits of the comparand register are selected as a key. The key is then compared to three words in CAM and Word 2 and Word 3 will be chosen.

The performance of CAM relies on its ability to conduct the comparisons simultaneously. Therefore, additional circuits are required to perform the task. Figure 3 shows a simple logic circuit for a 1-bit CAM cell, which includes circuits for reading from and writing into the memory cell as well as a match circuit for comparing the flip-flop contents to a corresponding bit in comparand register. Output for a match circuit is set to 1 if the data in the cell match the data in the corresponding bit in the comparand register or if the bit is not selected as a subfield of the key. The match indicator for a word will be set only if all match circuits within a word are set to 1. Figure 4 shows a bit slice of CAM that is formed by selecting a memory cell at the same position for each memory word in CAM. The fully parallel CAM allows all bit slices to perform comparisons in parallel. The comparison result from each selected bit in a memory word will be available simultaneously to specify the match indicator. But the circuits for each cell and the communication among cells makes CAM more expensive and complicated than conventional storage.

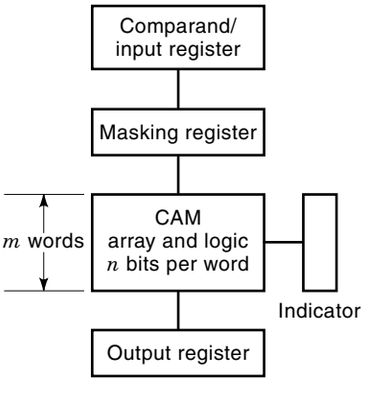

Figure 1. CAM block diagram.

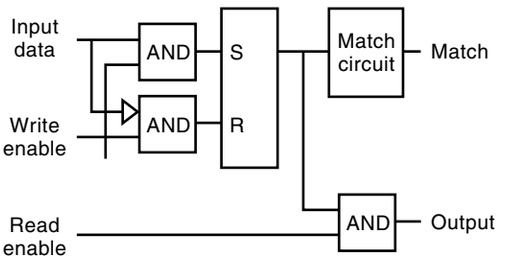

Figure 3. Logic diagram of CAM bit cell.

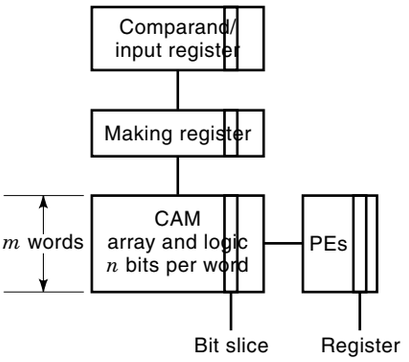

Figure 4. Bit slice CAM with PEs.

When only one bit slice is allowed to perform comparisons at any given time, delays for determining the match indicator can be expected. On the other hand, significant savings on hardware can be achieved. Figure 4 illustrates an implementation of a bit serial CAM in which a set of processing elements (PE) is added to the memory. To perform the comparisons among bits in the same bit slice and key, additional circuits are provided to read bit slices and place each bit in the corresponding PE. All PEs perform comparisons simultaneously, and the bit slices are replaced when the comparisons are finished.

For a bit serial CAM, a PE is assigned to each word in CAM, and all PEs can perform the processing simultaneously. Comparison, as described previously, is one of the functions that can be conducted by PEs. If PEs are enhanced to have registers and arithmetic logic capabilities, the bit serial CAM can be treated as a SIMD (single instruction stream multiple data stream) computer (1).

The discussions of CAM implementations thus far have been at bit level. It is also possible to expand the implementations discussed previously to be character based, word based, or field based. This can be achieved by adding additional hardware for fully parallel CAM or by developing algorithms for bit serial CAM. Numerous algorithms have been proposed in Ref. 2 for performing fast search and ordered retrieval of data in a bit serial CAM.

For applications where speed is a critical factor and the amount of information is relatively small, CAM is usually considered. Recent advances in semiconductor processing and memory design have increased the density and lowered the cost of CAM devices and expanded the applications for CAMs. CAM applications can be found mainly in the areas of database environment (3), signal processing (1,4–6), network routing tables, and computer address mapping for cache memory and translation look-aside buffers (5,7).

BIBLIOGRAPHY

1. K. E. Batcher, Bit-serial parallel processing systems, *IEEE Trans. Comput.,* **C-31**: 377–384, 1982.
2. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing,* New York: McGraw-Hill, 1984.
3. D. K. Hsiao (ed.), *Advanced Database Machine Architecture,* Englewood Cliffs, NJ: Prentice-Hall, 1983.
4. W. D. Hillis, *The Connection Machine,* Cambridge: MIT Press, 1985.
5. M. M. Mano, *Computer System Architecture,* Englewood Cliffs, NJ: Prentice-Hall, 1993.
6. J. L. Potter (ed.), *The Massively Parallel Processor,* Cambridge, MA: MIT Press, 1985.
7. C. C. Lu and Y. H. Shin, Parallel implementations of Huffman coding using associative memory, *Int. J. Model. Simulat.,* **16** (2): 67–72, 1996.

CHENG-CHANG LU
Kent State University

CONTEXT-SENSITIVE GRAMMARS.

See CONTEXT-SENSITIVE LANGUAGES.