

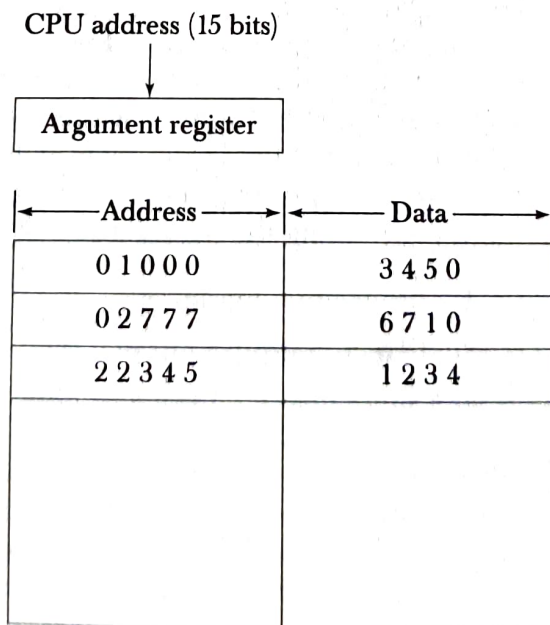
Figure 12-10 Example of cache memory.

a duplicate copy in main memory. The CPU communicates with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

Associative Mapping

The fastest and most flexible cache organization uses an associative memory. This organization is illustrated in Fig. 12-11. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory. The diagram shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

Figure 12-11 Associative mapping cache (all numbers in octal).



tag field

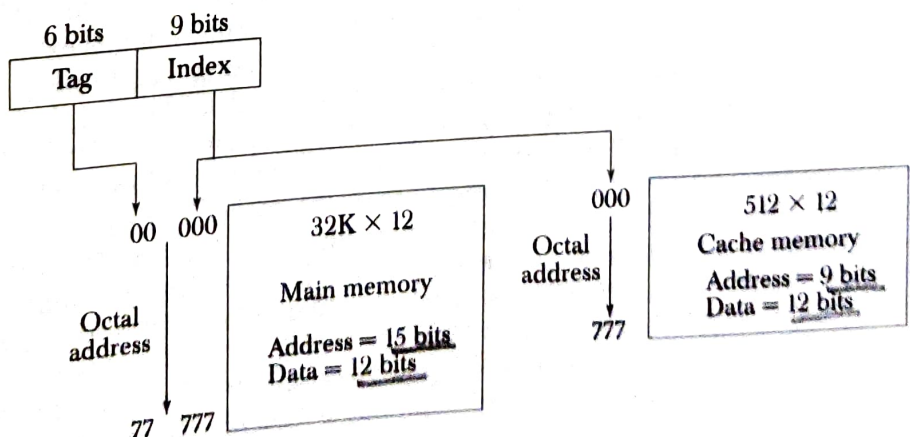
If the address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory. If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache. The decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache. A simple procedure is to replace cells of the cache in round-robin order whenever a new word is requested from main memory. This constitutes a first-in first-out (FIFO) replacement policy.

Direct Mapping

Associative memories are expensive compared to random-access memories because of the added logic associated with each cell. The possibility of using a random-access memory for the cache is investigated in Fig. 12-12. The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the *index* field and the remaining six bits form the *tag* field. The figure shows that main memory needs an address that includes both the tag and the index bits. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

In the general case, there are 2^k words in cache memory and 2^n words in main memory. The n -bit memory address is divided into two fields: k bits for the index field and $n - k$ bits for the tag field. The direct mapping cache organization uses the n -bit address to access the main memory and the k -bit index to access the cache. The internal organization of the words in the cache memory is as shown in Fig. 12-13(b). Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a

Figure 12-12 Addressing relationships between main and cache memories.



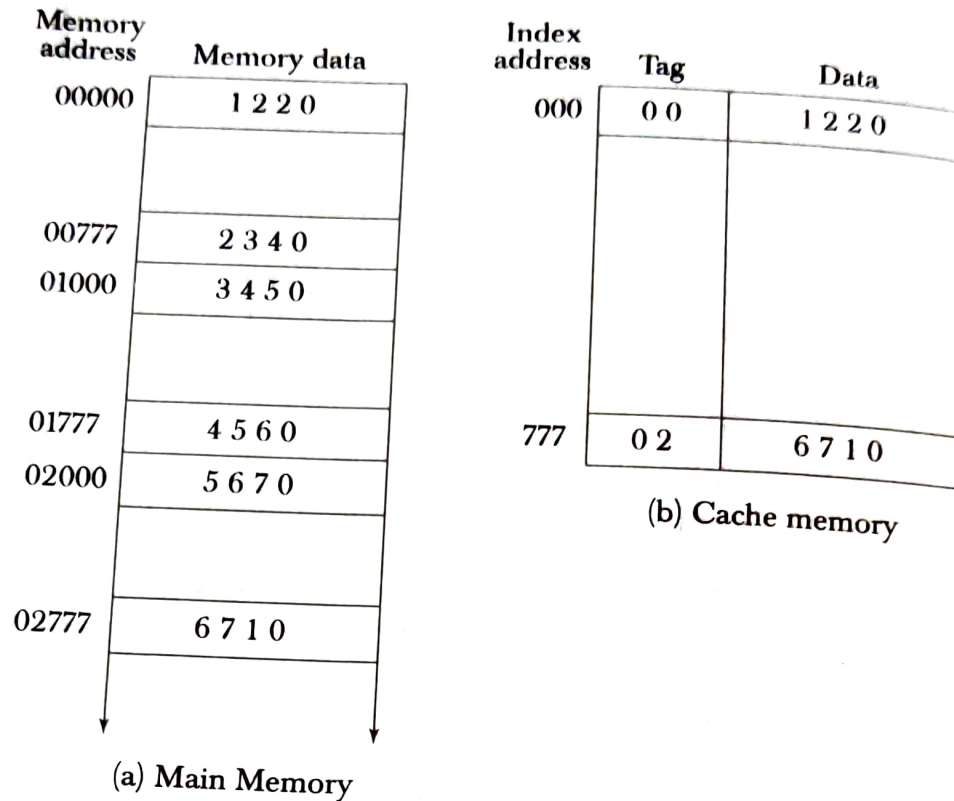


Figure 12-13 Direct mapping cache organization.

memory request, the index field is used for the address to access the cache. The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value. The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly. However, this possibility is minimized by the fact that such words are relatively far apart in the address range (multiples of 512 locations in this example).

To see how the direct-mapping organization operates, consider the numerical example shown in Fig. 12-13. The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220). Suppose that the CPU now wants to access the word at address 02000. The index address is 000, so it is used to access the cache. The two tags are then compared. The cache tag is 00 but the address tag is 02, which does not produce a match. Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU. The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.

The direct-mapping example just described uses a block size of one word. The same organization but using a block size of 8 words is shown in Fig. 12-14.

	Index	Tag	Data
Block 0	000	0 1	3 4 5 0
	007	0 1	6 5 7 8
Block 1	010		
	017		
Block 63	770	0 2	
	777	0 2	6 7 1 0

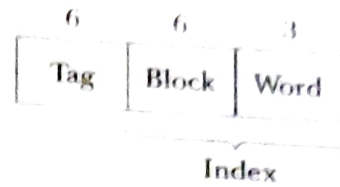


Figure 12-14 Direct mapping cache with block size of 8 words.

The index field is now divided into two parts: the block field and the word field. In a 512-word cache there are 64 blocks of 8 words each, since $64 \times 8 = 512$. The block number is specified with a 6-bit field and the word within the block is specified with a 3-bit field. The tag field stored within the cache is common to all eight words of the same block. Every time a miss occurs, an entire block of eight words must be transferred from main memory to cache memory. Although this takes extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs.

Set-Associative Mapping

It was mentioned previously that the disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time. A third type of cache organization, called set-associative mapping, is an improvement over the direct-mapping organization in that each word of cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set. An example of a set-associative cache organization for a set size of two is shown in Fig. 12-15. Each index address refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is $2(6 + 12) = 36$ bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512×36 . It can accommodate 1024 words of main memory since each word of cache contains two data words. In general, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.

Index	Tag	Data	Tag	Data
000	0 1	3 4 5 0	0 2	5 6 7 0
777	0 2	6 7 1 0	0 0	2 3 4 0

Figure 12-15 Two-way set-associative mapping cache.

The octal numbers listed in Fig. 12-15 are with reference to the main memory contents illustrated in Fig. 12-13(a). The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000. Similarly, the words at addresses 02777 and 00777 are stored in cache at index address 777. When the CPU generates a memory request, the index value of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name “set-associative.” The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in cache. However, an increase in the set size increases the number of bits in words of cache and requires more complex comparison logic.

When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value. The most common replacement algorithms used are: random replacement, first-in, first-out (FIFO), and least recently used (LRU). With the random replacement policy the control chooses one tag-data item for replacement at random. The FIFO procedure selects for replacement the item that has been in the set the longest. The LRU algorithm selects for replacement the item that has been least recently used by the CPU. Both FIFO and LRU can be implemented by adding a few extra bits in each word of cache.

Writing into Cache

An important aspect of cache organization is concerned with memory write requests. When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer. However, if the operation is a write, there are two ways that the system can proceed.

The simplest and most commonly used procedure is to update main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the *write-through* method. This method has the advantage that main memory always contains the same data as the cache. This characteristic is important in systems with direct memory access transfers. It ensures that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

The second procedure is called the *write-back* method. In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory. The reason for the write-back method is that during the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache. It is only when the word is displaced from the cache that an accurate copy need be rewritten into main memory. Analytical results indicate that the number of memory writes in a typical program ranges between 10 and 30 percent of the total references to memory.

Cache Initialization

One more aspect of cache organization that must be taken into consideration is the problem of initialization. The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, but in effect it contains some nonvalid data. It is customary to include with each word in cache a *valid bit* to indicate whether or not the word contains valid data.

The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again. The introduction of the valid bit means that a word in cache is not replaced by another word unless the valid bit is set to 1 and a mismatch of tags occurs. If the valid bit happens to be 0, the new word automatically replaces the invalid data. Thus the initialization condition has the effect of forcing misses from the cache until it fills with valid data.

12-6 Virtual Memory

In a memory hierarchy system, programs and data are first stored in auxiliary memory. Portions of a program or data are brought into main memory as the