Cache Memory

Analysis of large number of programs has shown that a number of instructions are executed repeatedly. This may be in the form of a simple loops, nested loops, or a few procedures that repeatedly call each other. It is observed that many instructions in each of a few localized areas of the program are repeatedly executed, while the remainder of the program is accessed relatively less. This phenomenon is referred to as locality of reference.

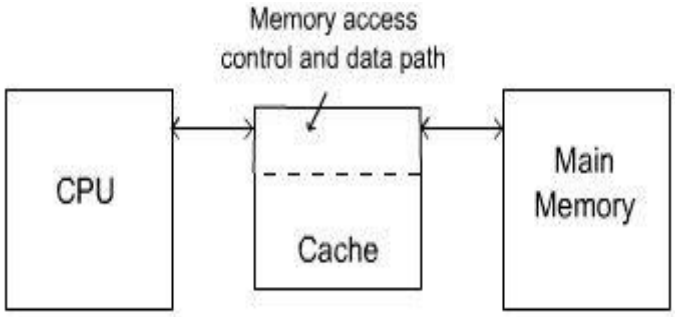

Figure 3.13: Cache memory between CPU and the main memory

Now, if it can be arranged to have the active segments of a program in a fast memory, then the total execution time can be significantly reduced. It is the fact that CPU is a faster device and memory is a relatively slower device. Memory access is the main bottleneck for the performance efficiency. If a faster memory device can be inserted between main memory and CPU, the efficiency can be increased. The faster memory that is inserted between CPU and Main Memory is termed as Cache memory. To make this arrangement effective, the cache must be considerably faster than the main memory, and typically it is 5 to 10 time faster than the main memory. This approach is more economical than the use of fast memory device to implement the entire main memory. This is also a feasible due to the locality of reference that is present in most of the program, which reduces the frequent data transfer between main memory and cache memory. The inclusion of cache memory between CPU and main memory is shown in Figure 3.13.

Operation of Cache Memory

The memory control circuitry is designed to take advantage of the property of locality of reference. Some assumptions are made while designing the memory control circuitry:

1. The CPU does not need to know explicitly about the existence of the cache.
2. The CPU simply makes Read and Write request. The natures of these two operations are same whether cache is present or not.
3. The address generated by the CPU always refers to location of main memory.
4. The memory access control circuitry determines whether or not the requested word currently exists in the cache.

When a Read request is received from the CPU, the contents of a block of memory words containing the location specified are transferred into the cache. When any of the locations in this block is referenced by the program, its contents are read directly from the cache.

Consider the case where the addressed word is not in the cache and the operation is a read. First the block of words is brought to the cache and then the requested word is forwarded to the CPU. But it can be forwarded to the CPU as soon as it is available to the cache, instead of the whole block to be loaded in the cache. This is called load through, and there is some scope to save time while using load through policy.

The cache memory can store a number of such blocks at any given time. The correspondence between the Main Memory Blocks and those in the cache is specified by means of a mapping function.

When the cache is full and a memory word is referenced that is not in the cache, a decision must be made as to which block should be removed from the cache to create space to bring the new block to the cache that contains the referenced word. Replacement algorithms are used to make the proper selection of block that must be replaced by the new one.

When a write request is received from the CPU, there are two ways that the system can proceed. In the first case, the cache location and the main memory location are updated simultaneously. This is called the store through method or writes through method.

The alternative is to update the cache location only. During replacement time, the cache block will be written back to the main memory. This method is called write back method. If there is no new write operation in the cache block, it is not required to write back the cache block in the main memory. This information can be kept with the help of an associated bit. This bit it set while there is a write operation in the cache block. During replacement, it checks this bit, if it is set, and then writes back the cache block in main memory otherwise not. This bit is known as dirty bit. If the bit gets dirty (set to one), writing to main memory is required.

The write through method is simpler, but it results in unnecessary write operations in the main memory when a given cache word is updated a number of times during its cache residency period.

During a write operation, if the address word is not in the cache, the information is written directly into the main memory. A write operation normally refers to the location of data areas and the property of locality of reference is not as pronounced in accessing data when write operation is involved. Therefore, it is not advantageous to bring the data block to the cache when there a write operation and the addressed word are not present in cache.

Mapping Functions

The mapping functions are used to map a particular block of main memory to a particular block of cache. This mapping function is used to transfer the block from main memory to cache memory. Three different mapping functions are available:

- ***Direct mapping:*** A particular block of main memory can be brought to a particular block of cache memory. So, it is not flexible.
- ***Associative mapping:*** In this mapping function, any block of Main memory can potentially reside in any cache block position. This is much more flexible mapping method.
- ***Block-set-associative mapping:*** In this method, blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. From the flexibility point of view, it is in between to the other two methods.

All these three mapping methods are explained with the help of an example.

Consider a cache of 4096 (4K) words with a block size of 32 words. Therefore, the cache is organized as 128 blocks. For 4K words, required address lines are 12 bits. To select one of the blocks out of 128 blocks, we need 7 bits of address lines and to select one word out of 32 words, we need 5 bits of address lines. So the total 12 bits of address is divided for two groups, lower 5 bits are used to select a word within a block, and higher 7 bits of address are used to select any block of cache memory.

Let us consider a main memory system consisting 64K words. The size of address bus is 16 bits. Since the block size of cache is 32 words, so the main memory is also organized as block size of 32 words. Therefore, the total number of blocks in main memory is 2048 (2K x 32 words = 64K words). To identify any one block of 2K blocks, we need 11

address lines. Out of 16 address lines of main memory, lower 5 bits are used to select a word within a block and higher 11 bits are used to select a block out of 2048 blocks.

Number of blocks in cache memory is 128 and number of blocks in main memory is 2048, so at any instant of time only 128 blocks out of 2048 blocks can reside in cache memory. Therefore, we need mapping function to put a particular block of main memory into appropriate block of cache memory.

Direct Mapping Technique:

The simplest way of associating main memory blocks with cache block is the direct mapping technique. In this technique, block *k* of main memory maps into block *k* modulo m of the cache, where *m* is the total number of blocks in cache. In this example, the value of *m* is 128. In direct mapping technique, one particular block of main memory can be transferred to a particular block of cache which is derived by the modulo function.

Since more than one main memory block is mapped onto a given cache block position, contention may arise for that position. This situation may occur even when the cache is not full. Contention is resolved by allowing the new block to overwrite the currently resident block. So the replacement algorithm is trivial.

The detail operation of direct mapping technique is as follows:

The main memory address is divided into three fields. The field size depends on the memory capacity and the block size of cache. In this example, the lower 5 bits of address is used to identify a word within a block. Next 7 bits are used to select a block out of 128 blocks (which is the capacity of the cache). The remaining 4 bits are used as a TAG to identify the proper block of main memory that is mapped to cache.

When a new block is first brought into the cache, the high order 4 bits of the main memory address are stored in four TAG bits associated with its location in the cache. When the CPU generates a memory request, the 7-bit block address determines the corresponding cache block. The TAG field of that block is compared to the TAG field of the address. If they match, the desired word specified by the low-order 5 bits of the address is in that block of the cache.

If there is no match, the required word must be accessed from the main memory, that is, the contents of that block of the cache is replaced by the new block that is specified by the new address generated by the CPU and correspondingly the TAG bit will also be changed by the high order 4 bits of the address. The whole arrangement for direct mapping technique is shown in the figure 3.14.

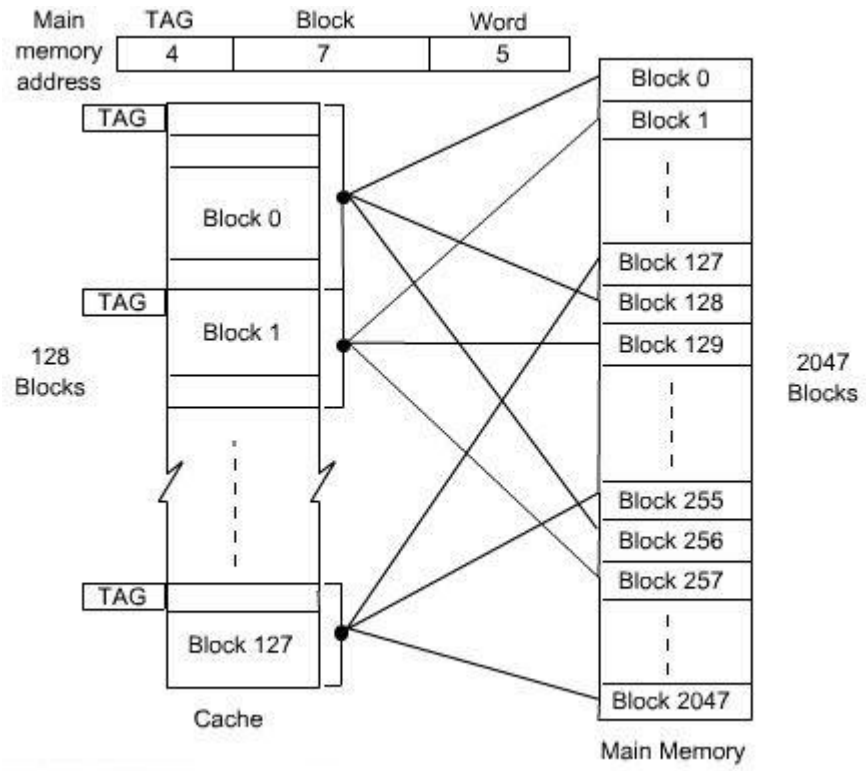

Figure 3.14: Direct-mapping cache

Associated Mapping Technique:

In the associative mapping technique, a main memory block can potentially reside in any cache block position. In this case, the main memory address is divided into two groups, low-order bits identifies the location of a word within a block and high-order bits identifies the block. In the example here, 11 bits are required to identify a main memory block when it is resident in the cache , high-order 11 bits are used as TAG bits and low-order 5 bits are used to identify a word within a block. The TAG bits of an address received from the CPU must be compared to the TAG bits of each block of the cache to see if the desired block is present.

In the associative mapping, any block of main memory can go to any block of cache, so it has got the complete flexibility and we have to use proper replacement policy to replace a block from cache if the currently accessed block of main memory is not present in cache. It might not be practical to use this complete flexibility of associative mapping technique due to searching overhead, because the TAG field of main memory address has to be compared with the TAG field of all the cache block. In this example, there are 128 blocks in cache and the size of TAG is 11 bits. The whole arrangement of Associative Mapping Technique is shown in the figure 3.15.

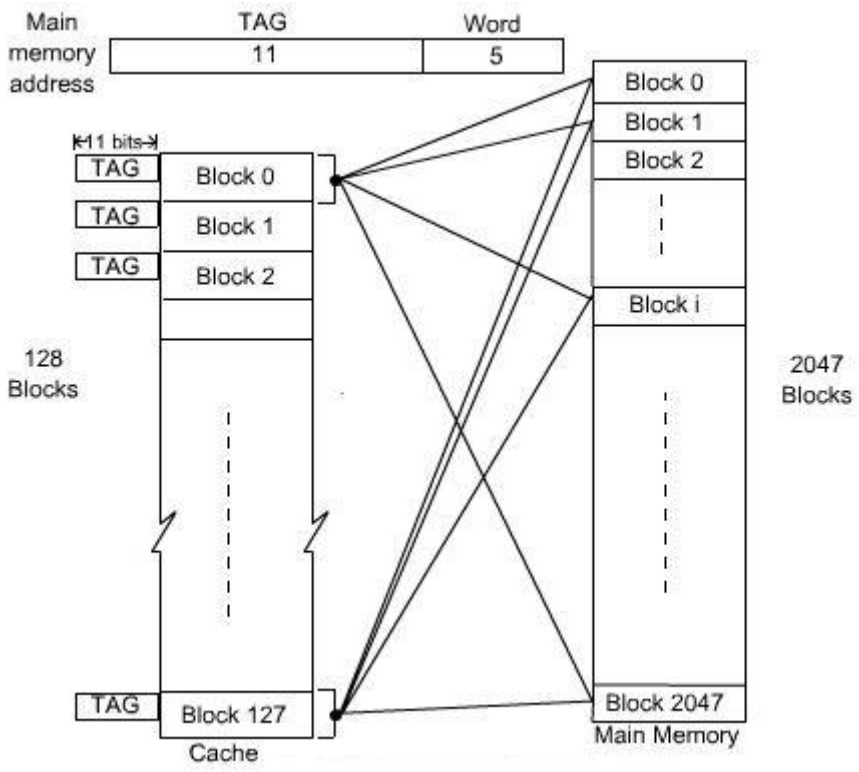

Figure 3.15: Associated Mapping Cache

Block-Set-Associative Mapping Technique:

This mapping technique is intermediate to the previous two techniques. Blocks of the cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. Therefore, the flexibility of associative mapping is reduced from full freedom to a set of specific blocks. This also reduces the searching overhead, because the search is restricted to number of sets, instead of number of blocks. Also the contention problem of the direct mapping is eased by having a few choices for block replacement.

Consider the same cache memory and main memory organization of the previous example. Organize the cache with 4 blocks in each set. The TAG field of associative mapping technique is divided into two groups, one is termed as SET bit and the second one is termed as TAG bit. Each set contains 4 blocks; total number of set is 32. The main memory address is grouped into three parts: low-order 5 bits are used to identifies a word within a block. Since there are total 32 sets present, next 5 bits are used to identify the set. High-order 6 bits are used as TAG bits.

The 5-bit set field of the address determines which set of the cache might contain the desired block. This is similar to direct mapping technique, in case of direct mapping, it looks for block, but in case of block-set-associative mapping, it looks for set. The TAG field of the address must then be compared with the TAGs of the four blocks of that set. If a match occurs, then the block is present in the cache; otherwise the block containing the addressed word must be brought to the cache. This block will potentially come to the corresponding set only. Since, there are four blocks in the set, we have to choose appropriately which block to be replaced if all the blocks are occupied. Since the search is restricted to four block only, so the searching complexity is reduced. The whole arrangement of block-set-associative mapping technique is shown in the figure 3.15.

It is clear that if we increase the number of blocks per set, then the number of bits in SET field is reduced. Due to the increase of blocks per set, complexity of search is also increased. The extreme condition of 128 blocks per set requires no set bits and corresponds to the fully associative mapping technique with 11 TAG bits. The other extreme of one block per set is the direct mapping method.

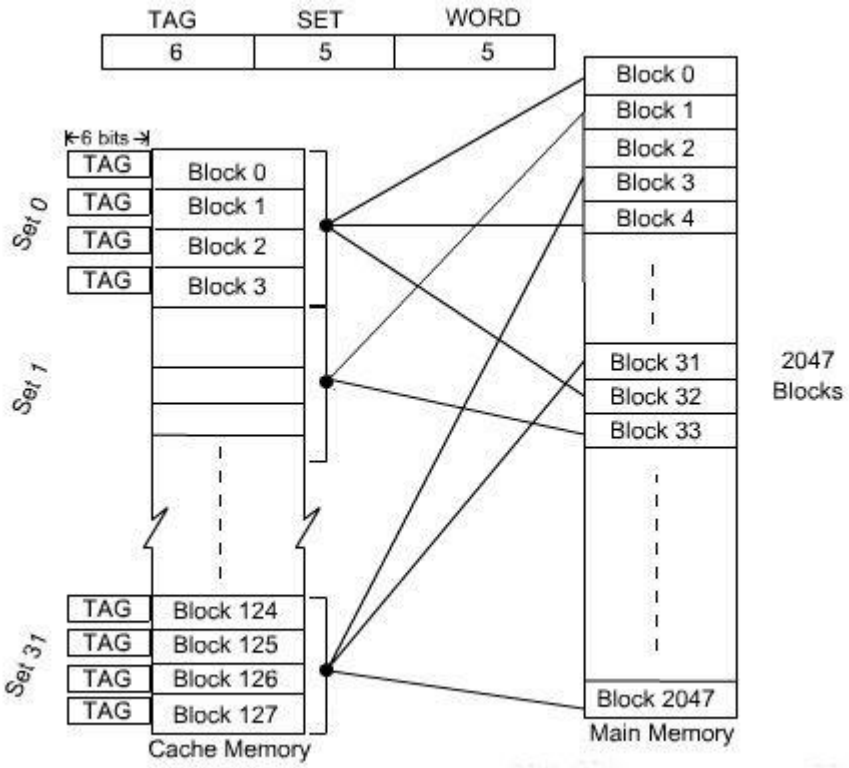

Figure 3.15: Block-set Associated mapping Cache with 4 blocks per set

Replacement Algorithms

When a new block must be brought into the cache and all the positions that it may occupy are full, a decision must be made as to which of the old blocks is to be overwritten. In general, a policy is required to keep the block in cache when they are likely to be referenced in near future. However, it is not easy to determine directly which of the block in the cache are about to be referenced. The property of locality of reference gives some clue to design good replacement policy.

Least Recently Used (LRU) Replacement policy:

Since program usually stay in localized areas for reasonable periods of time, it can be assumed that there is a high probability that blocks which have been referenced recently will also be referenced in the near future. Therefore, when a block is to be overwritten, it is a good decision to overwrite the one that has gone for longest time without being referenced. This is defined as the least recently used (LRU) block. Keeping track of LRU block must be done as computation proceeds.

Consider a specific example of a four-block set. It is required to track the LRU block of this four-block set. A 2-bit counter may be used for each block.

When a hit occurs, that is, when a read request is received for a word that is in the cache, the counter of the block that is referenced is set to 0. All counters which values originally lower than the referenced one are incremented by 1 and all other counters remain unchanged.

When a miss occurs, that is, when a read request is received for a word and the word is not present in the cache, we have to bring the block to cache.

There are two possibilities in case of a miss:

- If the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are incremented by 1.
- If the set is full and a miss occurs, the block with the counter value 3 is removed, and the new block is put in its place. The counter value is set to zero. The other three block counters are incremented by 1.

It is easy to verify that the counter values of occupied blocks are always distinct. Also it is trivial that highest counter value indicates least recently used block.

First In First Out (FIFO) replacement policy:

A reasonable rule may be to remove the oldest from a full set when a new block must be brought in. While using this technique, no updating is required when a hit occurs. When a miss occurs and the set is not full, the new block is put into an empty block and the counter values of the occupied block will be increment by one. When a miss occurs and the set is full, the block with highest counter value is replaced by new block and counter is set to 0, counter value of all other blocks of that set is incremented by 1. The overhead of the policy is less, since no updating is required during hit.

Random replacement policy:

The simplest algorithm is to choose the block to be overwritten at random. Interestingly enough, this simple algorithm has been found to be very effective in practice.