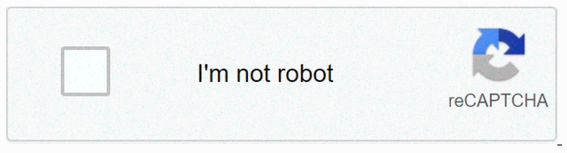

Continue

Cache memory design pdf

Dr. A. P. Shanthi This module aims to discuss the basics of cache memories. We discuss different mapping practices and also reading and writing practices. In principle, four priority questions related to block placement, block recognition, block exchange and writing strategy will be answered. The speed of the main memory is very low compared to the speed of modern processors. Due to good performance, the processor cannot spend much time waiting to access instructions and data in the main memory. It is therefore important to establish a system that shortens the time needed to obtain the necessary information. Since the speed of the main memory unit is limited by electronic and compression constraints, a solution must be sought in a different architectural arrangement. A powerful solution is to use fast cache, which makes the main memory look faster for the processor than it really is. Cache is a smaller and faster memory that stores copies of the most commonly used main memory place. As long as most memory permissions are in cached memory locations, the average latency of memory usages is closer to cache latency than memory latency. The effectiveness of the cache mechanism is based on the property of computer programs, called the reference location. Analysis of the programmes shows that most of their time is spent on routines where many instructions are repeatedly executed. These instructions can form a simple loop, nested loops, or a few procedures that repeatedly call each other. The real detailed model of sequencing instructions is not important - the point is that many instructions in the localized areas of the program are repeatedly executed over a period of time, and the rest of the program is used relatively rarely. This is called the reference locale. It manifests itself in two ways: in time and regional terms. The first means that the recent intrusion is likely to be re-executed very soon. The local perspective means that instructions close to the recently implemented instruction (addresses of the instructions) are also likely to be implemented soon. If active segments of the program can be placed in a fast cache, the total execution time can be significantly reduced. Conceptually, the operation of the cache is very simple. Memory control circuits are designed to take advantage of the locality feature of references. The temporal aspect of the reference point suggests that whenever a data item (help or data) is first needed, this item should be cached, hopefully preserving it until it is needed again. The statusable aspect suggests that instead of retrieving only one item from the main memory to cache, it is useful to retrieve multiple items that are also located in adjacent addresses. Use A term block that refers to a converge audience of any size. Another term that is often used to refer to a cache block is the cache row. Cache in memory hierarchy can be shared or unified/two. A shared cache is one where we have a separate data cache and a separate Help cache. Here, the two caches work side by side, one transfers data and the other transfers instructions. A double or unified cache is where data and instructions are stored in the same cache. A combined cache with a total size equal to the sum of two shared caches usually has a better hit rate. This higher pace is due to the fact that the combined cache does not strictly share the number of entries that the instructions used by the data can use. Nevertheless, many processors use shared help and data cache to increase cache bandwidth. When a read request is received from the processor, the contents of the memory block that contains the specified location are cached. When the program refers to any location in this block, the desired content is read directly from the cache. In general, a reasonable number of blocks can be cached at any time, but this amount is small compared to the total number of blocks in the main memory. The mapping between the main memory blocks and cached cache blocks is determined by a mail merge function. When the cache is full and the cached memory word (Help or data) is referenced, the cache management hardware must decide which block to delete to create space for the new block that contains the referenced word. The collection of rules for the adoption of this Decision constitutes a replacement algorithm. Therefore, the three main problems to be addressed in the cache are · Cache location – where do you cache the block? · Cache credentials – how do you recognize that the requested data is available in the cache or not? · Replacing cache - which block will be replaced in the cache, what will make way for the incoming block? These questions are answered and explained by an example with a main memory size of 1 MB (the main memory address is 20 bits), a cache size of 2 KT, and a block size of 64 bytes. Since the block size is 64 tt, you can immediately recognize that there are 214 blocks in the main memory and 25 blocks cached. That is, the 16,000 blocks of the main memory must be minged into 32 cache blocks. There are three different mapping practices : direct mapping, fully associative mapping, and n-road set associative mapping. They are discussed below. Direct mapping: This is the simplest mapping technique. In this technique, main memory block i is connected to cache block j modulo (number of cache blocks). In our example, it's block j mod 32. That is, the first 32 blocks of the main memory map to the corresponding 32 cache blocks, 0- 1-1, ... and 31-31. Remember, we only have 32 blocks in the stash. So, the following 32 main memory blocks are also connected to the same corresponding cache blocks. So 32 again maps block 0 cached, 33 block 1 cached and so on. That is, the main memory blocks are grouped into groups of 32 blocks, and each of these groups is mapped to the corresponding cache blocks. For example, whenever one of the most important memory blocks 0, 32, 64, ... is cached, it is only stored in cache block 0. So at any time, if any other block is occupying the cache block, it will be deleted and another block will be saved. For example, if we want to import block 64 and block 0 is already available in the cache, block 0 will be deleted and block 64 will be imported. Similarly, blocks 1, 33, 65, ... cache block 1 and so on. You can easily see that 29 main memory blocks connect to the same block in cache. Because more than one block of memory is connected to a location in a cache block, that location can also be disputed when the cache is not full. That is, blocks that have access to the same cache block can compete for the block. For example, if the processor refers to instructions for blocks 0 and 32, conflicts arise even though the cache is not full. The dispute will be resolved by allowing a new block to replace the current tenant block. Therefore, in this case, the replacement algorithm is ne ne ne nerivial. The block cannot be accommodated in any other location. So all it has to do is replace the block that currently lives. The location of the block in the cache depends on the memory address. The memory address can be divided into three fields as shown in Section 26.1. Small-order 6 bits select one block from 64 words. When a new block caches, a 5-bit cache block field specifies the location of the cache where this block must be saved. The 9 bits of the large order of the block's memory address are stored in 9 tag bits associated with its location in the cache. They recognize which of the 29 blocks that can be connected to this cache location are currently cached. When the main memory address is created, first check the block field. It points to a block that you need to check. Now check the tag field. If they match, the block is available in the cache and is a hit. Otherwise, I miss it. After that, the block containing the required word must first be read from the main memory and cached. After the block is identified, use the word field to retrieve one of the 64 words. Note that the word field does not participate in the mapping. Figure 26.1 Direct mapping Consider address 78F28, which is 0111 1000 1111 0010 1000. Now, check if the block is cached or not, divide it into three fields 011110001 to 11100 101000. The block field indicates that you need to check block 28. Now check the nine-bit tag field. If they match, it's a hit. Nniiden the technology is easy to implement. The number of tag transactions to check is only one, and the length of the tag field is also smaller. The replacement algorithm is very simple. However, it is not very flexible. Although the cache is not full, you may have to do a lot of licking between the main memory and the cache due to the rigid mapping policy. Fully associative mapping: This is a much more flexible mapping method where the main memory block can be placed in any cache block location. This means that no block field is required. In this case, 14 tag bits are required to identify the memory block when it is cached. This is shown in Figure 5.8. The address identification brick for addresses received from the processor is compared to the identifier account for each cache block to see if the desired block exists. This is called associative mapping technology. It gives complete freedom to choose the location of the cache where the memory block will be placed. Therefore, cache mode can be used more efficiently. A new block that must be cached must replace (delete) an existing block only if the cache is full. In this case, we need an algorithm that selects the block to replace. Commonly used algorithms are random, FIFO and LRU. Random overwriting makes a random selection of the block you want to delete. FIFO removes the oldest block without considering memory usage models. So it's not very effective. On the other hand, the least recently used technique takes into account the running patterns and removes the block that has not been referred to for the longest time. This is very effective. So the associative mapping is completely flexible. However, the cost of associative cache is higher than the cost of a directly mapped cache, since you need to search all the tag patterns to determine if a block has been entered in the cache. This should be an associative search, as discussed in the previous section. Also note that the length of the tag increases. That is, both the number of tags and the length of the tag increase. Substitution is also complicated. Therefore, this is practically not possible. Figure 26.2 Fully associative mapping Set associative mapping: This is a compromise between the two techniques mentioned above. Cache blocks are grouped into a set of n blocks, and the mapping allows the main memory block to be in any block in a particular group. It is also called n-way set associative mapping. Therefore, the problem of direct method dispute is facilitated by the fact that there are a few options for block placement. At the same time, hardware costs are reduced by reducing the size of the associative search. For example, the main memory address of set-associative mapping technology is shown in Figure 26.3 for a cache with two blocks per set (2-way associative mapping). There are 16 sets in the cache. In this case, memory blocks 0, 16, 32 ... 0, and they can use one of the block locations for this set. blocking. 16 series means that the 4-bit field in the address determines which set of caches might contain the desired block. The 11-bit tag field for the address must then be associatively compared to the identifiers of the two blocks of the set to check if the desired block exists. This double-fending associative search is easy to implement and combine the benefits of both other technologies. This can, in fact, be regarded as a general case; when n is 1, it becomes a direct mapping; when n is the number of blocks cached, it is an associative mapping. Figure 26.3 Associative mapping of associative mapping One more control must be provided for each block, called valid bits. This bit indicates whether the block contains valid data. It should not be confused with the modified or dirty bit mentioned earlier. Dirty bits indicating whether a block has been modified during its cache submission mode are only required on systems that do not use the sign-through method. Valid bits are set to 0 when power is originally enabled on the system, or when the main memory is loaded with new programs and disk data. Transfers from disk to main memory are performed using the DMA mechanism. Normally, they bypass the cache for both cost and performance reasons. The valid bit for a specific cache block is 1 when this block is first loaded from the main memory, when the cache bypass source updates the main memory block, a scan is performed to determine whether the block being loaded is currently cached. If so, its valid bit will be cleared to 0. This ensures that there is no staunch data in the cache. A similar difficulty occurs when the DMA transfer is made from the main memory to the disk and the cache uses the logon protocol. In this case, changes to the data in memory may not match the changes made to the cached copy. One solution to this problem is to clear the cache by forcing the dirty data to be written back into memory before the DMA transfer. This can be done easily by the operating system and does not have a major impact on performance, since such disk transfers do not occur frequently. This must ensure that two different entities (in this case processor and DMA sub-systems) use the same data copies, called cache inequality problem. Read/Write Policies: Lastly, we must also discuss how to follow reading and writing practices. The handler does not need to know explicitly about the existence of the cache. It simply gives read and write requests with addresses that refer to the location of the memory. The cache control circuit determines whether the requested word is currently cached. If so, read/write will run at the appropriate cache site. In this case, it is said that a reading or writing break has occurred. No changes are made to the read operation, so the main memory is not There are two ways for the system to proceed for a write break. Way. the first technique, called the sign-through protocol, cache location and main memory location are updated at the same time. Another technique is to update only the location of the cache and mark it as updated with the associated flag bit, often called a dirty or modified bit. The main memory place of the word will be updated later when the block containing this flagged word is removed from the cache to make room for the new block. This technique is called write-back or copy protocol. The sign-through protocol is simpler, but it leads to unnecessary writing operations in the main memory when the given cache word is updated several times during cache waste disposal. Note that the write-back protocol can also lead to unnecessary write operations, because when the cache block is written back into memory, all words in the block are written back, even if only one word has been changed while the block is cached. This can be avoided if you store more dirty bits per block. During the writing operation, if the addressed word is not cached, an uninserted write occurs. If you use the protocol for the trans-written protocol, the data is written directly to the main memory. In the case of the logon protocol, the block containing the addressed word is first cached, and then the desired word is replaced with new data in the cache. When the writing does not occur, we use the writing targeting policy or not the writing targeting policy. Therefore, if we use the write-back policy, the block will be cached anyway (write booking) and a dirty bit will be set. On the other hand, if it is written through the policy used, the block is not reserved for cache and changes occur immediately in the main memory. Regardless of the writing strategies used, processors usually use a write buffer so that the cache can proceed as soon as the data is placed in the buffer, rather than waiting until the data is actually written in the main memory. In conclusion, we have discussed to increase the need for cache. We have explored a variety of issues related to cache memories, viz., investment practices, replacement policies, and reading and writing practices. Direct mapping is the simplest to carry out. In a direct merged cache, the cache block is available before determining whether it is a hit or miss, since it is possible to assume a hit and continue and recover later if it is over. It also requires only one comparison compared to N-comparisons in n-way associative mapping. In the case of associative mapping, the data has an additional MUX delay, and the data only comes after it is determined whether it hits or not. However, the operation can be accelerated by comparing all set tags side by side and selecting data based on the result of the tag. Associative is more flexible than direct mapping. Complete associative mapping is the most flexible, but also the most complex to implement rarely used. Web Links / Supporting Materials Computer Organization &; Design – Hardware /Software Interface, David A. Patterson and John L. Hennessy, 4. Computer Architecture – Quantitative Approach , John L. Hennessy and David A.Patterson, 5. Computer organization, Carl Hamacher, Zvonko Vranesic and Safwat Zaky, 5. Edition, McGraw-Hill Higher Education, 2011. 2011.