Hide Index Cache

(Especially for ELE408 lab programs)

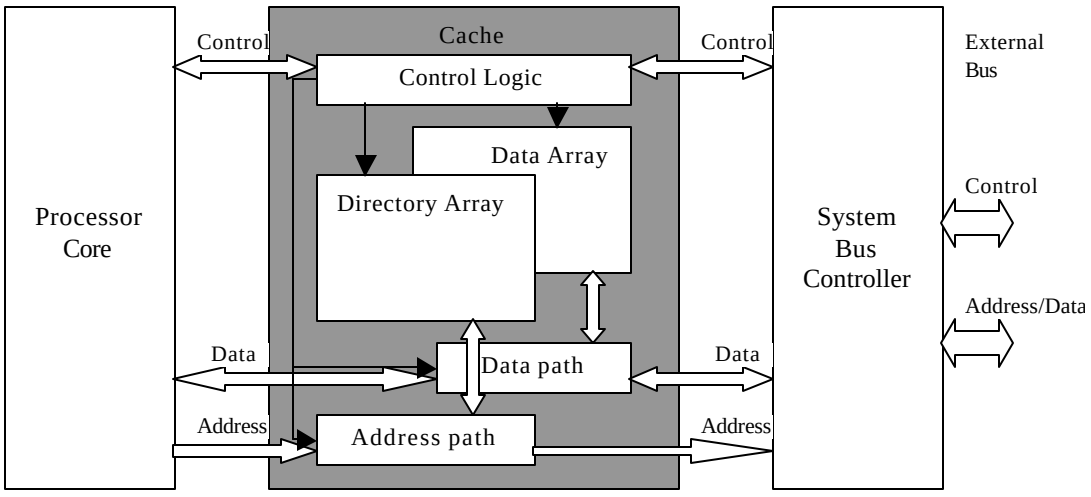
Yinan Liu *yinan@ele.uri.edu*
http://www.ele.uri.edu/~yinan

Introduction

The MCF5307 processor contains a non-blocking, 8-kbyte, 4-way set-associate, unified (instruction and data) cache with a 16-byte line size. The cache improves system performance by providing low-latency data to the MCF5307 instruction and data pipes, which decouples processor performance from system memory performance, resulting in an increase in bus availability for alternate bus masters.

The MCF5307 non-blocking cache services read hits or write hits from the processor while a fill (caused by a cache allocation) is in progress.

As shown in following figure, both instruction and data accesses are performed using a single bus connected to the cache. All addresses from the processor to the cache are physical addresses. If the address matched on of the cache entries, the access hits in the cache. For a read operation, the cache supplies the data to the processor, and for a write operation, the data from the processor updates the cache. If the access does not match one of the cache entries (miss in the cache) or a write access must be written through to memory, the cache performs a bus cycle on the internal bus and correspondingly on the external bus by way of the system bus controller (SIM).

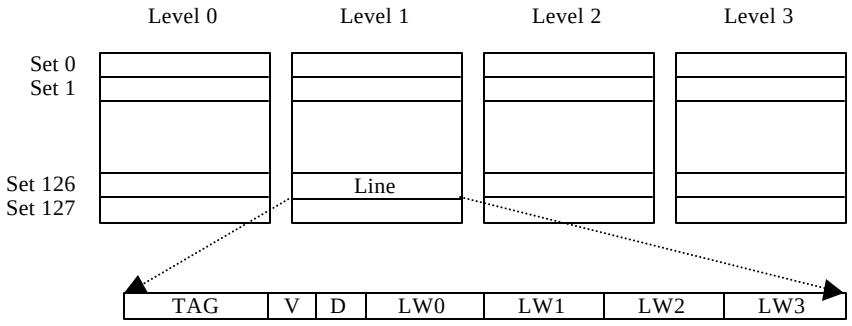


In this paper we will analyze all the application program developed so far in the lab, and with this result change the cache structure of the MCF5307, to get the better hit ratio.

Normal Cache Organization

The 4-way set associative cache is organized as four levels (ways) of 128 sets with each line containing 16 bytes (4 longwords) of storage. The following figure illustrates the cache organization (as well as the terminology used) along with the cache line format.

We have 32bits address bus, we use address bits A[10:4] as a index to select a set. Each line consists of an address tag (upper 21 bits of the address), two status bits, and four longwords of data. The two status bits consist of a valid bit and a dirty bit for the line. Address bits A3 and A2 select the longword within the line.



Analyze data accessing for ELE408 lab Programs

Analyze all the application programs that we have developed so far in the lab. What we can see? A lot of data used repeatedly, such as sub function and loop procedure, especially nested loop. As the following table shows:

This means that in these programs the same data will be used for many times. Keeping such data in the cache, we can get the better performance. In LRU replace algorithm, cache control program only can keep the recently use message, if some data cannot be used during some times, it will be replaced. It is useful to find past usage of data, but it cannot forecast what will be used in the future. For example, the following data array which has 6 cycles: ABCDEFGHIJK ABCDEFGHIJK ABC……. DEFGHIJK. If we want to access this array from a four lines cache, the miss ratio will be 100%, we cannot catch any data through cache. When HIJK in the cache ABCD have already be replaced. If we give the cache controller the ability to guess what will be used in the future. Things will be changed. The cache will keep the data will be used in the future. As the same data array, when HIJK is being read sequentially, if ABCD is kept in the cache. During 6 cycles, we will win 4 hits in each cycle.

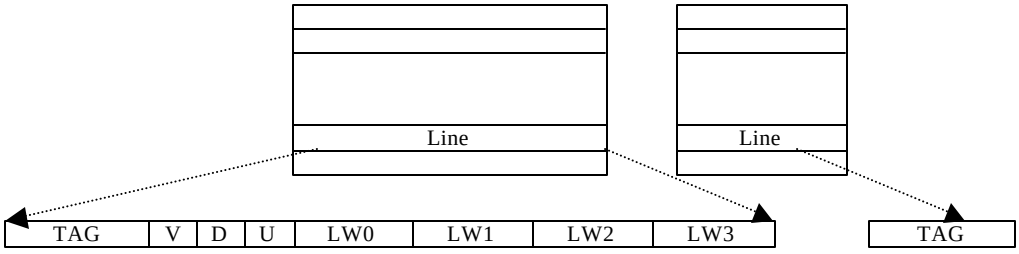
How to make cache guess what kind of data will be used. We can make decision through analyzing the data used before. Sieving out the data used before, we can find out what maybe used for many times. For this purpose we can divide the data in the cache into two groups. One is transferred from memory for the first time, the other is not transferred from memory for the first time, which has been in the cache before. Why we separate it like this? To sequential performing instructions, cache cannot forecast what will be used in the future, so the data that will be used in the future cannot be kept or be transferred into the cache. For loop instructions or nested instructions, it is possible to do some forecast for the cache, and cache can keep the data for the loop or nested instructions using in the future. In this way we can get a better hit ratio.

For normal cache, once the data in cache is replaced, cache cannot keep any record for it, it will be lost absolutely. When it is transferred into cache for second time, there is no different between it and a new data transferred from memory for the first time. The data used before is a important information for cache. Why we say that? Because it is possible to be used for third time, even be used thousands times. Keeping this data in cache for a long time is better than replace it. This is the point of all we talked above: The data be read for the second is likely to be read for the third time or more, it means that it may be accessed by loop instruction or a nested program. Cache should keep this kind of data for longer time than the new data just transferred from memory for the first time, though it is recently used. For this purpose, the data in the cache is divided into two types basically: one can be called the ***transition model***, it should be cleared from cache as quickly as possible, because maybe it won't be used any more; the other one can be called the ***conceal model***, the cache should try it best keep it in the cache, because maybe after inactive time, it will be used again. I think keeping this kind of data in the cache, improving the cache performance, are for actualize the worthiness of adding cache between CPU and memory. What we should do next is rebuilding cache structure and finding a good replacement algorithm for this purpose.

The idea of Hide Index Cache

To know which is used before, the cache must have some mechanism to record the information of the usage of the data. As we talked before, we divide the data in cache into two groups. The new mechanism must can divide two kinds of data auto automatically, it can change data in transition model to the data in conceal model also.

The main idea of this project is add a attach component to the cache, using it to do the work talked above. For this purpose, we can add a hide index to record the data of the conceal model. As the following figure shows, the conventional cache index structure and the data area is at the left side (we suppose the cache has K index). What at the right side is the hide index, it keep the data address of the conceal model data, no data inside. We add a new bit in the conventional cache area, it is for signing the data model (transition model or conceal model). We can manage this kind of cache as 2K indexes cache.



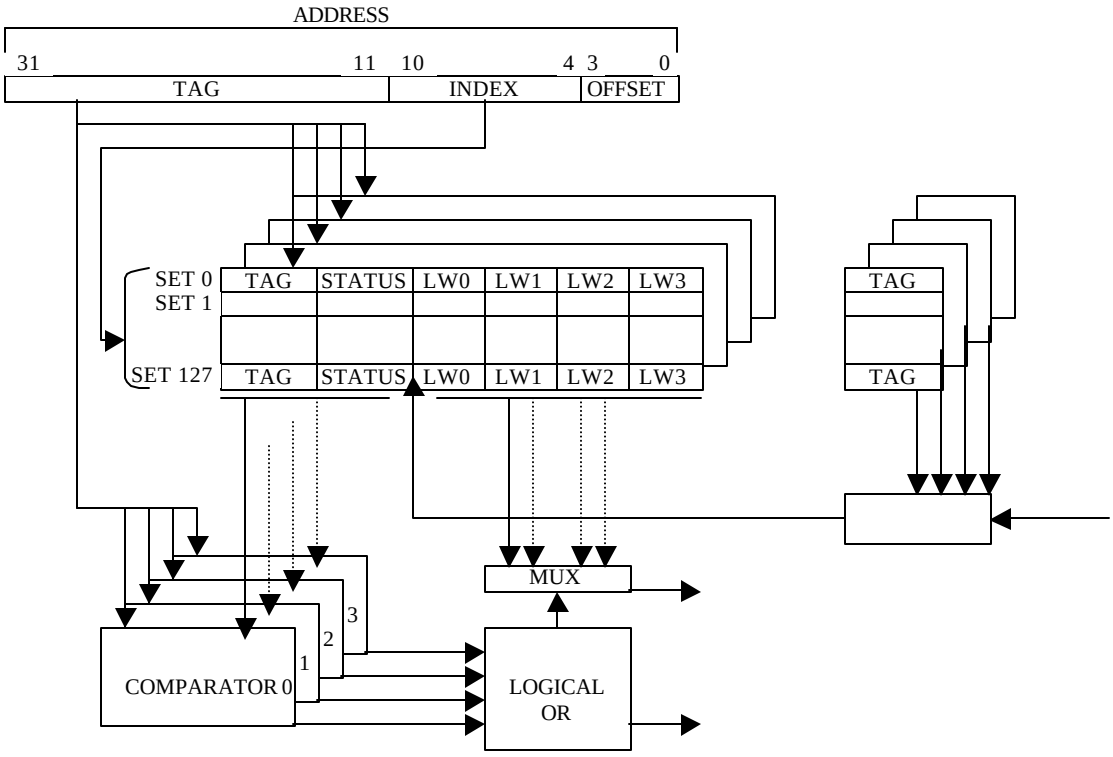
When a new data is transferred into cache, the data of one of K indexes will be replaced in the cache group. The address of the replaced data will be written into the hide index. In a general way, the computer has enough time to modify cache and two different indexes. Because only when the miss happens, the computer needs to modify it, at that time, cache does not do any other work, it is in logjam situation. To manage such cache system, we must control such two cache misses.

1. ***Transition misses***: the data wanted by CPU is not in cache, and not in the hide index too.
2. ***Conceal misses***: the data wanted by CPU is not in cache, but the hide index has it record.

Conceal misses means that the data used before, and now it will be used again, this means that it is possible to be used again in the future. Using it we can forecast the possibility of repeating the same implementation after this two sequential access.

When a new item transferred into the cache, one of bit will be set to 0 or 1, to sign this item to be transition misses or conceal misses. This bit can be used for to control data replacement. When some data should be replaced, cache manage software will look-over all the items in the cache, and hold the item have record in the hide index. In this way, we can control the transition misses.

How does it works



The upper figure shows the structure of the hide index cache. The left area is the same as the normal cache structure, at the right area is the new mechanism added to old cache structure.

1. Initializing the Cache.

   When the cache used for first time (such as the computer just being started), it should be set initial status. For the initial status the hide index will be clear, and the "Used" bit should be set as 0. At this status, all data transferred from memory is new to the cache, and the hit ratio will be 0 before the first hit happens.

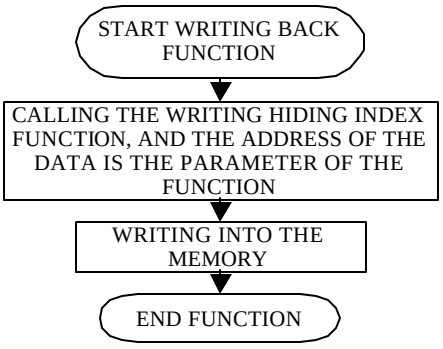
2. Replacing the data in cache

   As data is transferred from memory to cache for many times, the cache will full of data, and then replace will happen. To the data in the same model, the replacement will follow the LRU Algorithm, the recently used will be kept. For the data in the different model (transition model and conceal mode), the cache controller should replace the data which "Used" bit is set. When this data is replaced, it will be wrote back to the memory, and also its address will be write into the hide index cache.

3. Replacing the data in the hide index

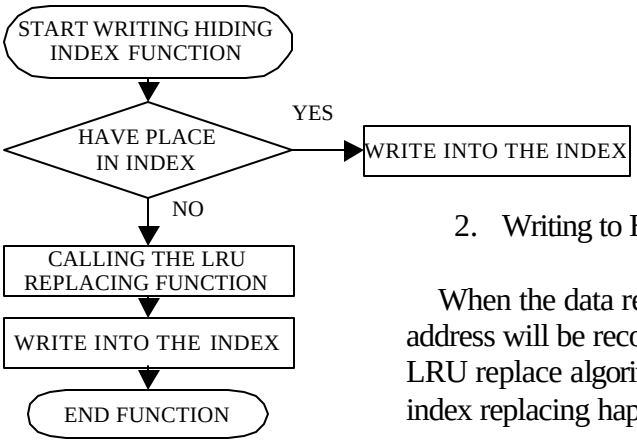
The data in the hide index cache just is the address of used data. For normal replacing, it should follow the LRU Algorthm. When the data that has record in the hide index will be transferred into the cache again, it will clear the record in the hide index first, and set the "Used" bit to be 1, and then write into the cache.

The following flow chart Shows the main function for cache operation.



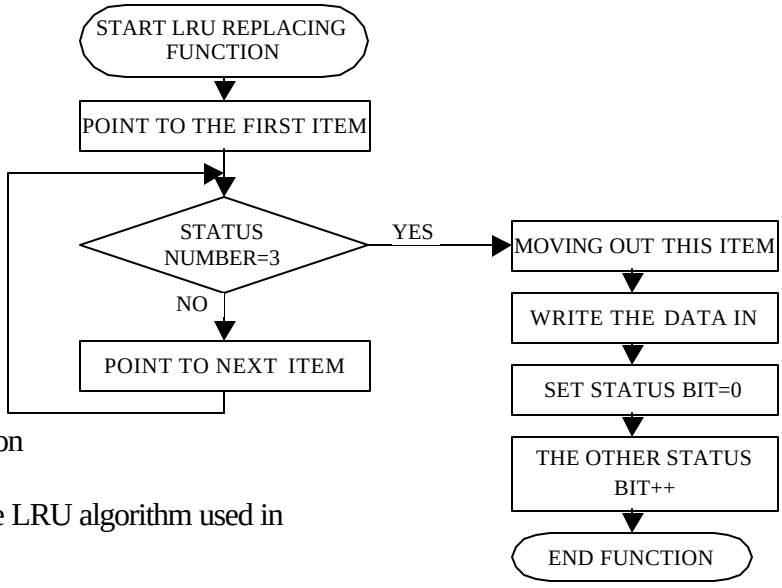
1. Writing back function

When a data in the cache will be replaced, it should be write back to the memory first.



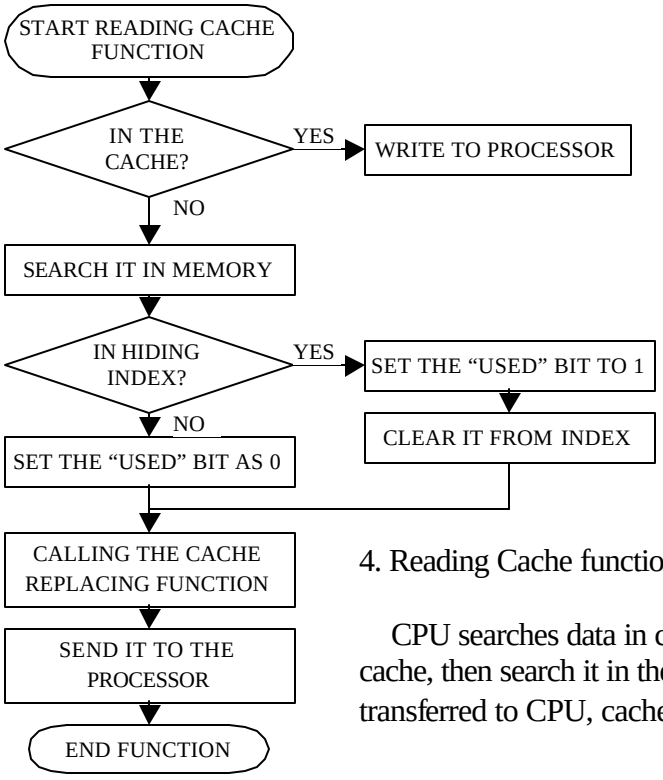
2. Writing to Hiding index function

When the data replacing happens, the data address will be record in the hide index. The LRU replace algorithm is used when the hide index replacing happens.



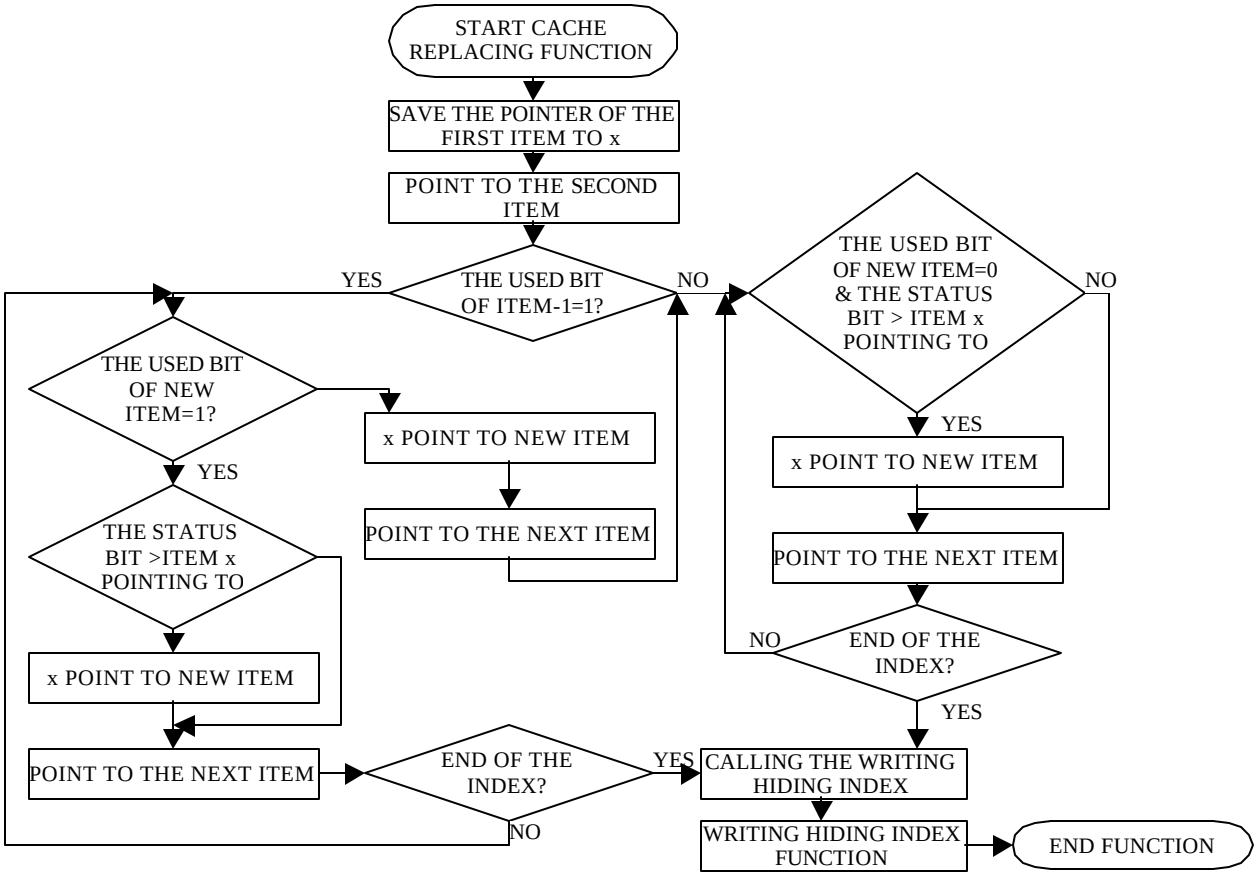
3. LRU replacing function

This function realizes the LRU algorithm used in other functions.



4. Reading Cache function

CPU searches data in cache first. If it is not in the cache, then search it in the memory. During the data is transferred to CPU, cache replacing will happed.



5. Cache replacing function

The basic replacing algorithm is LRU algorithm. Different kinds of data has different priority. The new data should be replaced first.

Estimation of Using Hide Index Cache

The price of hard device is very important. Also it is the reason for make a cache in the computer architecture. Adding hide index is very cheap, the most expensive part of memory and cache system is the data storage, because it needs more hardware. For example, if each line of cache has 16 bytes. For address tag, it only needs 4 bytes. For data storage it needs 12 bytes. When byte number increase to 64 bytes, what hide index need is just lower than 10% of the data storage (4:64). Because the speed is not the most important things to the hide index, we can put the hide index on cache chip or out of the cache chip. The price of hide index will same as memory.