German University in Cairo
Faculty of Media Engineering and Technology
Dr. Nada Sharaf

Concepts of Programming Languages, Spring 2021
CPU Cache
Deadline: 10 June 2021

Project Description

Cache is a component that stores data so that future requests for that data can be served faster. Recently or frequently used data are stored temporarily in the cache to speed up the retrieval of the data by reducing the time needed for accessing cache clients data such as the CPU.

In this project you must present a successful implementation of a simplified CPU cache system that works by retrieving data (from cache if possible, otherwise from memory) given the memory addresses of the data and successfully updating the cache upon the data retrieval.

The idea of a cache is that it introduces hierarchy into memory systems. Thus, instead of having one level for a large memory which will probably be slow because it needs to be cheap, we can have multiple levels.

In the project we will assume we have two levels as shown in Figure 1

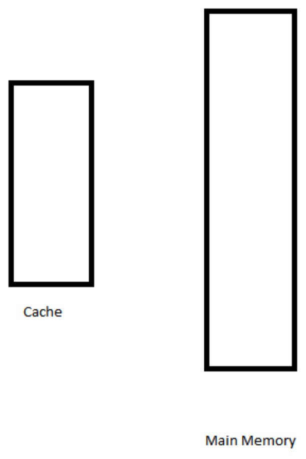


Figure 1: Memory Hierarchy

The operation in this case goes as follows:

1. A specific block of data is requested. The block is identified by its location in the main memory
2. First, the system has to look if the block at the required address was mapped somewhere in the cache.
3. If found in the cache, there is no need to go to the main memory.
4. If the data is not available in the cache then:
   (a) The system has to rad the block from the main memory
   (b) The bread block has to be placed in the cache.
   (c) The place where the block is placed in the cache depends on the type of the cache as shown in the next section
   (d) If the chosen cache location(line) is not empty, then the block already in the cache will be overwritten by the new block read from the main memory

Cache Mapping Types

You will implement 3 types of cache mapping techniques: Direct mapping, fully associative and set associative.

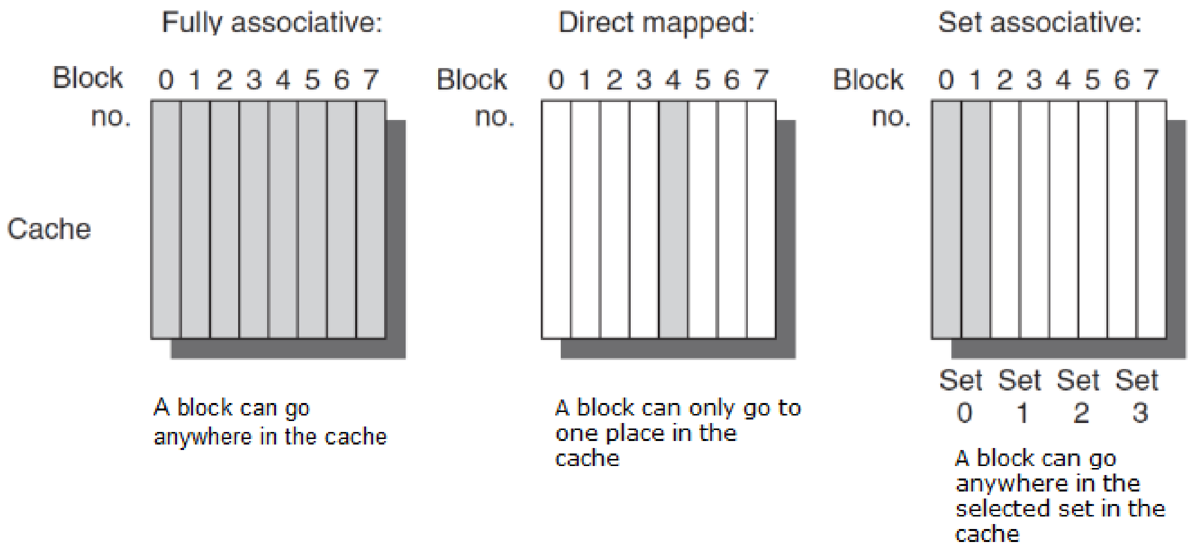


Figure 2: Cache Mapping Types

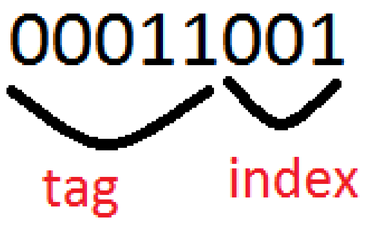
German University in Cairo
Faculty of Media Engineering and Technology
Dr. Nada Sharaf

Address

An address of a block in the main memory is always translated to a binary number. The address is split into two parts index field and a tag field. Each part is assigned a specific number of bits.

- The index always indicates the location inside the cache to which this main memory block will map to
- The tag is used as an identifier for the block

For example if the main memory address is 00011001 such that the index has 3 bits and the tag has 5 bits then the address is divided as below:



Thus the least significant 3 bits in this case are the index bits and the rest of the address is the tag

This means that the memory block with address 00011001 should be mapped to an option in the cache indexed by 001

Cache Options

The number of options inside the cache indicate the number of bits needed for the index part.

- For example, if there are 4 options in the cache then these options will be numbered as follows: 00, 01, 10 and 1. Thus we need **2** bits for the index
- In case we have 8 options in the cache. These options are numbered from 000 to 111. Thus we need **3** bits for the index.
- In general if we have 2^i options in the cache, we need i bits for the index part of the address.

The options inside the cache differ according to the type of the cache. Details on the mapping are shown according to the types in the following sections

Direct Mapping

In the case of direct mapping, any block in the main memory can map to **exactly one location** in the cache. In other words, direct mapping maps each block of data in main memory into only one possible cache line.

In case of direct mapping, the options of mapping are the blocks inside the cache. The index part of the address will specify which line/location in the cache you will go to.

The tag will confirm whether the data stored in this line of cache is of the correct address or not by comparing it with the tag stored in cache.

For example: If the index is 3 bits and the given address is "00000101" then the index is "101" and the tag is "00000". The memory address "00001101" will have the same index "101" but a different tag "00001".

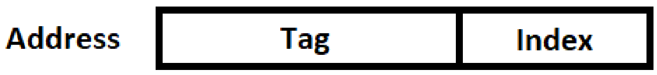


Figure 3: Direct Mapping

Every entry in the cache will have two values:

- Data Value representing the actual data inside this location
- Tag value indicating which specific memory block is inside this cache location. In more words, as shown in the previous example multiple memory blocks can map to the same location inside the cache. How would we know which specific memory block is there? From the tag

Given the below example as part of the cache:

Index	Tag	Data Value
000	00010	-
001	01100	-
010	00001	-
011	01010	-
100	01000	ABC
101	**00001**	-

If we are searching for memory block 00000101, we will do the following:

1. Extract the index. In this case the index is 101
2. Extract the tag. In this case the tag is 00000
3. Extract the tag available in the cache at this location. The tag in this case is 00001

4. Compare both tags together. Since they are not the same then the memory block 00000101 is not available in cache

If we are searching for memory block 00001101, we will do the following:

1. Extract the index. In this case the index is 101
2. Extract the tag. In this case the tag is 00001
3. Extract the tag available in the cache at this location. The tag in this case is 00001
4. Compare both tags together. Since they are equal then the memory block 00001101 is available in cache and the data of this block is ABC

Fully Associative Mapping

In this type of mapping, any block can go into any line of the cache. This means that the tag becomes all of the bits of the address. This enables the placement of any word at any place in the cache memory. It is considered to be the most flexible mapping form. **For example:** If the given address is "0000101" then the tag is "0000101" and can be placed anywhere in the cache so we need to compare all the tags.

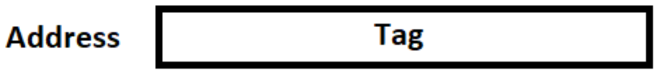


Figure 4: Fully Associative Mapping

Set-associative Mapping

Set-associative caches is a combination of both direct and fully associative caches. In this case the cache is divided into sets. Each set contains a group of blocks.

Whenever you have a block in the4 main memory, this block can map into exactly one set in the cache. However, inside the set it can be placed in any of the blocks.

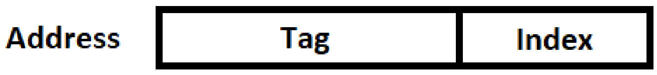


Figure 5: Set-associative Mapping

Set-Index	Block Inside Set	Tag	Data Value
00	0	010001	-
00	1	000011	-
01	0	000010	-
01	1	000001	ABC
10	0	000000	-
10	1	101111	-
11	0	100011	-
11	1	110011	-

In other words, in set-associative, instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set.

As in direct mapping, the address is split into index and tag. But, the index here represent which set to access. The number of options is thus the number of sets available. Then the tag is compared to all the tags in this set to make sure we get the correct address. If we have 4 sets, then the index part is 2 bits, in this case the tag part will be 6 bits in case we are using 8 bit addresses. **For example:** If the index is 2 bits and the given address is "00000101" then the index is "01" and the tag is "000001". that means you should go to the set 101 and we need to compare all the tags in this set.

Given the above example if we are searching for memory block 00000101, we do the following:

1. Extract the index to know which set this block should map to. In this case the index is 01.
2. Extract the tag to start doing the comparisons. In this case the tag is 000001.
3. Go to the set and start comparing block by block. The first block inside set 01 has a tag 000010 which is not equal to 000001. However the second block inside set 01 has a tag 000001 which is equal to 000001. THus, the data value is ABC

Data retrieval

If the processor finds that the memory location is in the cache, a cache hit has occurred and data is read from cache.

If the processor does not find the memory location in the cache, a cache miss has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

Replacement

When a cache miss occurs and no empty slots present, data should be retrieved from the memory to update the cache and get the data. Different replacement strategies are available according to the cache type. In case, of direct mappimg there are no choiced to make since the block maps to exactly one location. However, in fully and set associative caches, a choice of which clock to remove has to be made. Options include:

1. Random: where a random block is chosen to be replaced.
2. First-In First-Out (FIFO) replacement policy. The way it works is that it simply replaces the oldest block that was placed in the cache with the desired one. This is represented by the `Order` which denotes when this item was placed in cache.
3. least recently used, where the block that was least recently used is replaced.

Requirements

In this project, you are required to provide a Prolog and Haskell implementation for the cache system.

The project will be divided into components. There is a general component that every team is required to submit. There are also some specific components. Every team should split the specific components among the members. Thus, every member is only required to work on their specific component.

The specific components are split as follows:

1. Component 1: Direct mapping
   (a) Split address into index and tag part
   (b) Check if a data item is available in cache
   (c) Get data from cache
2. Component 2: Set Associative
   (a) Split address into index and tag part
   (b) Check if a data item is available in cache
   (c) Get data from cache
3. Component 3: Replacing blocks

   (a) Replace a block in case of direct mapping
   (b) Replace a block in case of fully associate cache
   (c) Replace a block in case of set associative cache

4. Component 4 (Optional): Fully Associative
   (a) Split address into index and tag part
   (b) Check if a data item is available in cache
   (c) Get data from cache

In case a team has 3 members, only the first 3 components are to be implemented.

You need to fill in the form:
`https://docs.google.com/forms/d/e/1FAIpQLScJyPH39XJij5RcFK0RSikEYQNdBi_cxdF1zTYZ3HP3vVLf8A/viewform?usp=sf_link`

In case a team does not fill the form, they will be graded for the full project together