Cache Documentation

Introduction

This documentation serves as a guide to use the cache implemented with three different policies – direct mapping, fully associative mapping and n-way set associative mapping. The language used for writing the program is C++. This project is created as part of the end semester assignment of the CSE-112 Computer Organization course offered by Dr. Prabhakar Mateti during Winter 2020 at IIITD.

Basic Assumptions

Following points are assumed while creating the cache -

- A main memory and a cache memory are maintained. Values in each are stored as pairs of (Address => Value)
- Each pair is contained in one word. 1 word is of size 1 byte and can contain one value. One block equals one line. The number of lines is determined by user input. The number of words in one block equals block size.
- For the caches, FIFO cache replacement policy is used (as and when applicable)
- All address values are initialized with 0 in the start
- Block number in memory is same as ‘tag’ number, it is represented in decimal form (and not in binary) for ease of understanding
- All sizes entered/used/operated upon are in bytes.
- Upper Limits – total words – 0xfff, total blocks – 9999 (dec), maximum value – 9999 (dec), maximum words in 1 block – 1024, maximum cache size – 9999*1024 bytes
- Inside the folder – 3 .cpp files (1 for each type), 2 .txt files (for test cases), 1 pdf for documentation

Instructions Supported

The cache supports two instructions – read and write.

The read instruction returns the value at the specified address. In case the address is already present in the cache, a CACHE HIT message is generated and value is returned. Otherwise, the address is fetched from main memory and inserted into cache as per the cache policy and a CACHE MISS is reported along with the value at the address.

The write instruction works in a similar manner, except this time, a value is written at the desired address. Analogous to read, there are two cases – CACHE HIT and CACHE MISS here.

The user can run the program in two modes – either by running a pre-written text file of test cases or by entering custom input.

User Input

Following values will be given as input by the user (when running manually) -

- Main memory size
- Cache memory size
- Block size
- Value of ‘n’ for n-way set associative mapping
- Total number of queries (read/write in correct format)
- Specify the instruction in either of the formats “read <address>” or “write <address> <value>”.
- All addresses are handled in hexadecimal form. Values at an address are taken in decimal form.

Outputs Displayed

At the end of each query, the state of cache is printed (different for different policies). At the end of all queries, the HIT RATIO is printed, defined as the number of hits divided by sum of total hits and misses.

Note: Main memory structure can also be printed if needed, but it commented in the source code.

1. Direct Mapping

Principle – Each block from the main memory is directly mapped to a line in the cache memory, meaning, each block from the main memory can go to one and only one fixed line in the cache memory. There is no cache replacement policy as such here because blocks are directly mapped.

The line number is given by the formula → *block number % number of cache lines*

The output printed is as shown -

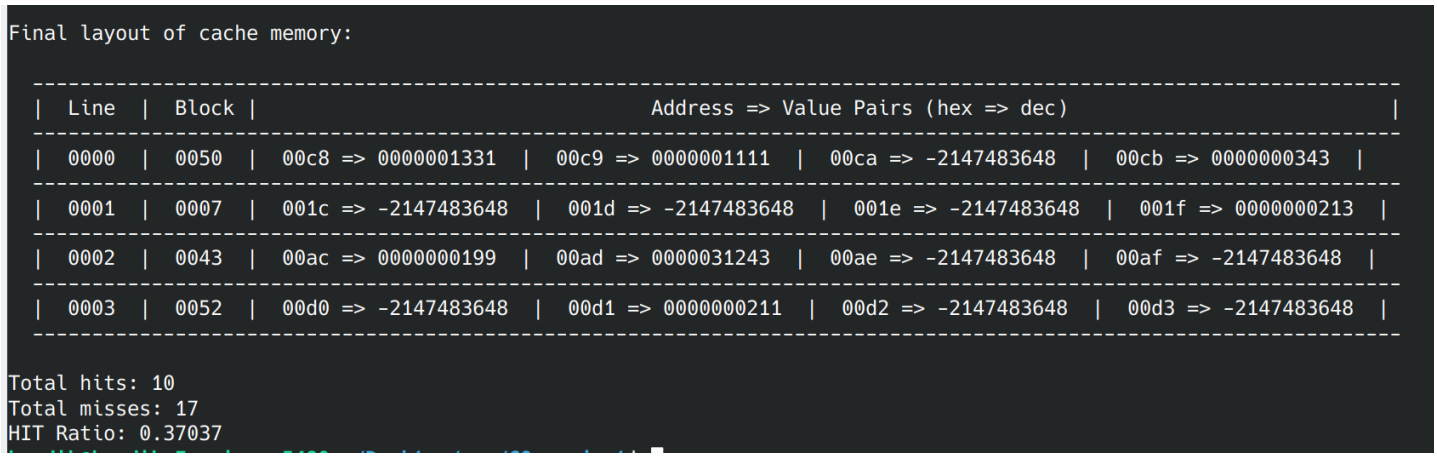
```
Final layout of cache memory:

  ---------------------------------------------------------------------------------------------------------------
  |  Line  |  Block |                                Address => Value Pairs (hex => dec)                        |
  ---------------------------------------------------------------------------------------------------------------
  |  0000  |  0052  |  00d0 => -2147483648  |  00d1 => 0000000211  |  00d2 => -2147483648  |  00d3 => -2147483648  |
  ---------------------------------------------------------------------------------------------------------------
  |  0001  |  0001  |  0004 => -2147483648  |  0005 => -2147483648  |  0006 => -2147483648  |  0007 => -2147483648  |
  ---------------------------------------------------------------------------------------------------------------
  |  0002  |  0050  |  00c8 => 0000001331  |  00c9 => 0000001111  |  00ca => -2147483648  |  00cb => 0000000343  |
  ---------------------------------------------------------------------------------------------------------------
  |  0003  |  0043  |  00ac => 0000000199  |  00ad => 0000031243  |  00ae => -2147483648  |  00af => -2147483648  |
  ---------------------------------------------------------------------------------------------------------------

Total hits: 11
Total misses: 16
HIT Ratio: 0.407407
```

2. Fully Associative Mapping

Principle – In this mapping, any block goes anywhere, meaning, there is no fixed cache line for a block. Therefore, we require a cache replacement policy here. Some examples of such policies are – LRU (least recently used) and FIFO (first in first out).

For the assignment, FIFO policy is used which is implemented using a queue.

The output printed is as shown -

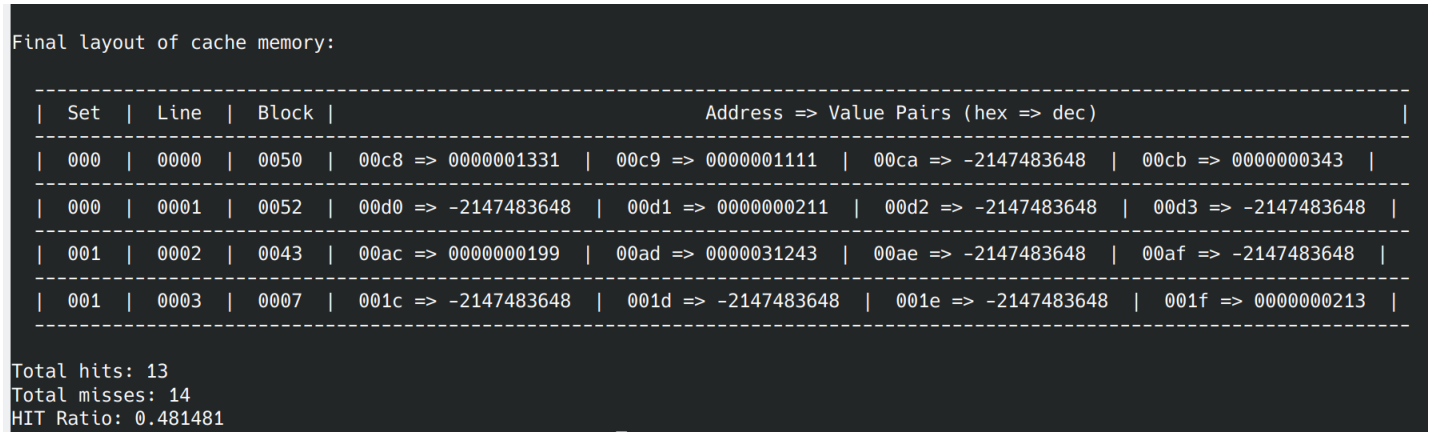


```
Final layout of cache memory:

| Line | Block |                              Address => Value Pairs (hex => dec)                              |
| 0000 | 0050  | 00c8 => 0000001331 | 00c9 => 0000001111 | 00ca => -2147483648 | 00cb => 0000000343 |
| 0001 | 0007  | 001c => -2147483648 | 001d => -2147483648 | 001e => -2147483648 | 001f => 0000000213 |
| 0002 | 0043  | 00ac => 0000000199 | 00ad => 0000031243 | 00ae => -2147483648 | 00af => -2147483648 |
| 0003 | 0052  | 00d0 => -2147483648 | 00d1 => 0000000211 | 00d2 => -2147483648 | 00d3 => -2147483648 |

Total hits: 10
Total misses: 17
HIT Ratio: 0.37037
```

3. n-way Set Associative Mapping

Principle – This mapping is a combination of direct mapping and fully associative mapping. First, the cache lines are divided into sets. Sets are allotted to each block via direct mapping. Then, within every set, fully associative cache mapping is practiced. For this, we again have to specify a cache replacement policy. For this assignment, FIFO policy is used.

Number of sets = number of cache lines / n (n represents lines in 1 set)
Set number for a block = block number % number of sets
Words in 1 block = block size
The output printed is as shown -



```
Final layout of cache memory:

| Set | Line | Block |                                Address => Value Pairs (hex => dec)                              |
| 000 | 0000 | 0050  | 00c8 => 0000001331 | 00c9 => 0000001111 | 00ca => -2147483648 | 00cb => 0000000343 |
| 000 | 0001 | 0052  | 00d0 => -2147483648 | 00d1 => 0000000211 | 00d2 => -2147483648 | 00d3 => -2147483648 |
| 001 | 0002 | 0043  | 00ac => 0000000199 | 00ad => 0000031243 | 00ae => -2147483648 | 00af => -2147483648 |
| 001 | 0003 | 0007  | 001c => -2147483648 | 001d => -2147483648 | 001e => -2147483648 | 001f => 0000000213 |

Total hits: 13
Total misses: 14
HIT Ratio: 0.481481
```

Error Handling

Following basic errors are handled in the assignment -

- Memory address out of range
- Undefined instruction entered by the user

Functions and Structures used in the code

The following features are common with respect to implementation in all the three caches -

- A ‘block’ class to represent a block, containing an array of words, a block id, and set number (for n-way set associative)
- A set class (for n-way set associative) containing a vector of blocks and a queue to maintain FIFO policy for the blocks.
- A search function for the “read” query which searches the block as per the cache policy.
- A write function for the “write” query which searches the block and write the value to it.
- A function to pretty print main memory (commented)
- A function to pretty print cache memory.
- Conditional Clauses to handle incorrect instructions and memory addresses.
- A global variable to calculate hits and misses and thus, the HIT RATIO.

Created by – Hardik Garg, 2019040, section-A, group-2

References – https://youtu.be/eObN3u3eAnU,
https://www.ntu.edu.sg/home/smitha/ParaCache/Paracache/dmc.html