**CS 6354 Graduate Computer Architecture Fall 2024**

**Homework 4 Report**

Student Name: Jing-Ning Su (fzf9mg)

**1. Direct Mapped Cache**

**1.1 Warm-up Exercise**

Below is a sequence of twelve 32-bit memory address references given as word addresses. 1, 18, 2, 3, 4, 20, 5, 21, 33, 34, 1, 4. Note that word and byte addresses are different. The word address appended by offset (two zeros) gives you the byte address. In other words, word addresses are multiples of 4. For example, if the word address is ”1”, the equivalent byte address is ”4” (100).

**Question 1: Assuming a direct-mapped cache with one-word blocks and a total size of 16 blocks, list if each reference is a hit or a miss assuming the cache is initially filled with word address 0, 1, 2, . . . 15 memory data. Show the state of the cache after the last reference. Calculate the hit rate for this reference string. (Correct answer: hit rate = 4/12)**

From the table below, we can see hit rate is 4/12, and cache is 0, 1, 34, 3, 4, 21, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 after the last reference.

(each box in the follows represent a word, each box forms a block)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Addr | 1 | 18 | 2 | 3 | 4 | 20 | 5 | 21 | 33 | 34 | 1 | 4 | Last |
| H/M | H | M | M | H | H | M | H | M | M | M | M | M |
| Cache Status | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 18 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 20 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 20 | | 5 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 2 | | 3 | | 20 | | 21 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 33 | | 2 | | 3 | | 20 | | 21 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 33 | | 34 | | 3 | | 20 | | 21 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 34 | | 3 | | 20 | | 21 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | | |  | | --- | | 0 | | 1 | | 34 | | 3 | | 4 | | 21 | | 6 | | 7 | | 8 | | 9 | | 10 | | 11 | | 12 | | 13 | | 14 | | 15 | |  | |

**Question 2:** **Now, assuming a direct-mapped cache with two-word blocks and a total size of 8 blocks, list if each reference is a hit or a miss assuming the cache is initially empty. Show the state of the cache after**

From the table below, we can see hit rate is 1/12, and cache is [0, 1][34, 35][4, 5][-, -][-, -] [-, -] [-, -] [-, -] after the last reference.

(each box in the follows represent a word, every two boxes form a block)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Addr | 1 | 18 | 2 | 3 | 4 | 20 |
| H/M | M | M | M | H | M | M |
| Cache Status | |  |  | | --- | --- | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 18 | 19 | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 2 | 3 | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 2 | 3 | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 2 | 3 | | 4 | 5 | |  |  | |  |  | |  |  | |  |  | |  |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5 | 21 | 33 | 34 | 1 | 4 | Last |
| M | M | M | M | M | M |
| |  |  | | --- | --- | | 0 | 1 | | 2 | 3 | | 20 | 21 | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 2 | 3 | | 4 | 5 | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 2 | 3 | | 20 | 21 | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 30 | 31 | | 2 | 3 | | 20 | 21 | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 30 | 31 | | 34 | 35 | | 20 | 21 | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 34 | 35 | | 20 | 21 | |  |  | |  |  | |  |  | |  |  | |  |  | | |  |  | | --- | --- | | 0 | 1 | | 34 | 35 | | 4 | 5 | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |

**1.2 Direct Mapped Cache Implementation**

**Code Implementation:**

1. find\_set

|  |
| --- |
| def find\_set(self, address):  return (address >> self.blockBits) & (self.sets - 1) |

Find the cache set identifier corresponding to the address by performing a rightward shift on the address by the number of bits equal to the block size, and then subtracting one from the quotient obtained.

1. find\_tag

|  |
| --- |
| def find\_tag(self, address):  return address >> (self.blockBits + self.setBits) |

The tag field utilized in the cache indexing process is derived by shifting the address to the right, effectively discarding the bits that correspond to the cache set index and the block offset.

1. find

|  |
| --- |
| def find(self, address):  set\_index = self.find\_set(address)  tag = self.find\_tag(address)  if self.metaCache[set\_index][0] == tag:  self.hit += 1  return True  else:  self.miss += 1  return False |

Employ the `find\_set` function to ascertain the cache set index corresponding to the address, and then use the `find\_tag` function to extract the tag. Subsequently, verify if there is a matching tag within the identified cache set to determine if it constitutes a cache hit.

1. load

|  |
| --- |
| def load(self, address):  set\_index = self.find\_set(address)  tag = self.find\_tag(address)  self.metaCache[set\_index][0] = tag  self.cache[set\_index][0] = address |

Upon a cache miss, the `find\_set` function determines the cache set index for the address, the tag is then extracted. Subsequently, the tag in the corresponding cache set is updated to the new address's tag, and the data block associated with the new address is loaded from memory into the cache, implementing the cache replacement policy.

**Simulation and Results:**

This part is just the testing way when I was coding, for the way of using dm\_a.trace, see “2.3 Verification”.

For question 1 in warm-up, since the cache is initially filled, I modified the following code temporarily to simulate the situation. Then, I can use `dm\_b.trace` directly.

|  |
| --- |
| # modified here  def reset(self):  // …  for i in range(self.sets):  for j in range(self.ways):  start\_value = j \* self.sets \* self.blockSize + i \* self.blockSize  self.cache[i][j] = np.arange(start\_value, start\_value + self.blockSize, dtype=int)  // … |

The output of the first question is as follow, corresponds with the result of hit rate = 4/12.

A computer screen shot of a program

Description automatically generated

For question 2, the cache is not initially filled, so I removed the change in `reset` function. The command and result are as follow; it corresponds with the result of hit rate = 1/12.

A computer screen shot of a program

Description automatically generated

**2.3 Verification**

As asked in the guideline, the way of using `dm\_a.trace` and `dm\_b.trace` for testing is as follows.

**Question 1:**

Since question 1 has one-word block and 16 blocks in total, total cache size is 16 \* 4 with block size 4. The command line is `python3 sim\_dm.py dm\_a.trace 64 1 4`.

The output is as follows, however, the middle of it is omitted due to the length.

A screen shot of a computer program

Description automatically generated  
…  
A screenshot of a computer

Description automatically generated

The result is 24 misses, but we need to minus 16 because they were the lines used to initialize the cache. Therefore, the number of true misses is 8, which corresponds with the result of question 1.

**Question 2:**

Since question 2 has two-word block and 8 blocks in total, total cache size is 8 \* 8 with block size 8. The command line is `python3 sim\_dm.py dm\_a.trace 64 1 8`.

A computer screen shot of a program

Description automatically generated

The result is 11 misses, which corresponds with the result of question 2.

**3. Set Associative Cache**

**3.1 Warm-up Exercise**

Below is a sequence of twelve 32-bit memory address references given as word addresses. 1, 18, 2, 3, 4, 20, 5, 21, 33, 34, 1, 4. Note that word and byte addresses are different. The word address appended by offset (two zeros) gives you the byte address. In other words, word addresses are multiples of 4. For example, if the word address is ”1”, the equivalent byte address is ”4” (100).

**Question: Assuming a set-associative cache with two ways, two-word blocks and a total size of 8 blocks, list if each reference is a hit or a miss assuming the cache is initially empty (assume LRU replacement). Show the state of the cache after the last reference. Calculate the hit rate for this reference string. (Correct answer: hit rate = 5 / 12)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Addr | 1 | 18 | 2 | 3 |
| H/M | M | M | M | H |
| Cache Status | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 20 | 5 | 21 | 33 |
| M | M | H | H | M |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |

|  |  |  |  |
| --- | --- | --- | --- |
| 34 | 1 | 4 | Last |
| M | H | H |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  | |

**3.2 Set Associative Cache Implementation**

Code Implementation:

Include the code from cache\_sa.py.

Explain the logic of the code, focusing on the implementation of the find\_set, find\_tag, find, and load functions.

Simulation and Results:

Conduct simulations using sim\_sa.py and present the output.

Interpret the results of the simulation.

**3.3 Verification**

Describe how you used the sa.trace file for verification.

Present the verification results and compare them with the calculations from the warm-up section.

**4. Conclusion**

Summarize the main findings and learning points of the report.

**5. References**

1. To run all test cases with one command, I added files as follows:

1.1 Script for mac:

|  |
| --- |
| #!/bin/bash  python ../code/sim\_dm.py first.trace 16 1 4  python ../code/sim\_dm.py first.trace 16 1 8  python ../code/sim\_dm.py pingpong.trace 16 1 4  python ../code/sim\_sa.py pingpong.trace 16 2 4  python ../code/sim\_sa.py test\_q.trace 64 2 4  python ../code/sim\_sa.py sa.trace 64 2 8  python ../code/sim\_dm.py dm\_a.trace 64 1 4  python ../code/sim\_dm.py dm\_b.trace 64 1 8 |

1.2 Script for windows:

|  |
| --- |
| python ../code/sim\_dm.py first.trace 16 1 4  python ../code/sim\_dm.py first.trace 16 1 8  python ../code/sim\_dm.py pingpong.trace 16 1 4  python ../code/sim\_sa.py pingpong.trace 16 2 4  python ../code/sim\_sa.py test\_q.trace 64 2 4  python ../code/sim\_sa.py sa.trace 64 2 8  python ../code/sim\_dm.py dm\_a.trace 64 1 4  python ../code/sim\_dm.py dm\_b.trace 64 1 8 |

2. Run with the following command:

|  |
| --- |
| # windows  ..\test\test\_all.bat > ..\..\submit\log.txt  # mac  ../test/test\_all.sh > ../../submit/log.txt |

3. If shows permission denied, run the following command:

|  |
| --- |
| chmod +x ../test/test\_all.sh chmod u+w ../../submit/ |