Computer Architecture Project 2019

Cache Simulator

20182705 고주형

1. How to run and what is output

It is same with guideline which is given in eclass

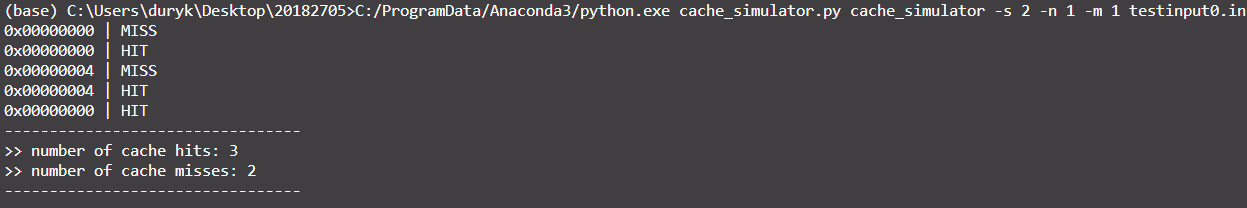
a. Change directory to directory where my project is.

b. Type Command:

>> python.exe cache\_simulator.py cache\_simulator –s <S> -n <N> –m <M> <input\_file\_name>

c. If the command is correct then it prints out the outcome and also make output file with input file’ s name. (if “a.in” is given -> “a.out” is created as an output file)

(Example)



2. Used Language: Python3

3. Test input files and outputs which I used in this project

* For test of input which is given in detailed information e-class site: “testinput0.in”
* For test of a lot of miss pattern: “testinput1.in”
* For test of a lot of hit pattern: “testinput2.in”
* For test of a lot of random pattern: “testinput3.in”

4. Functions

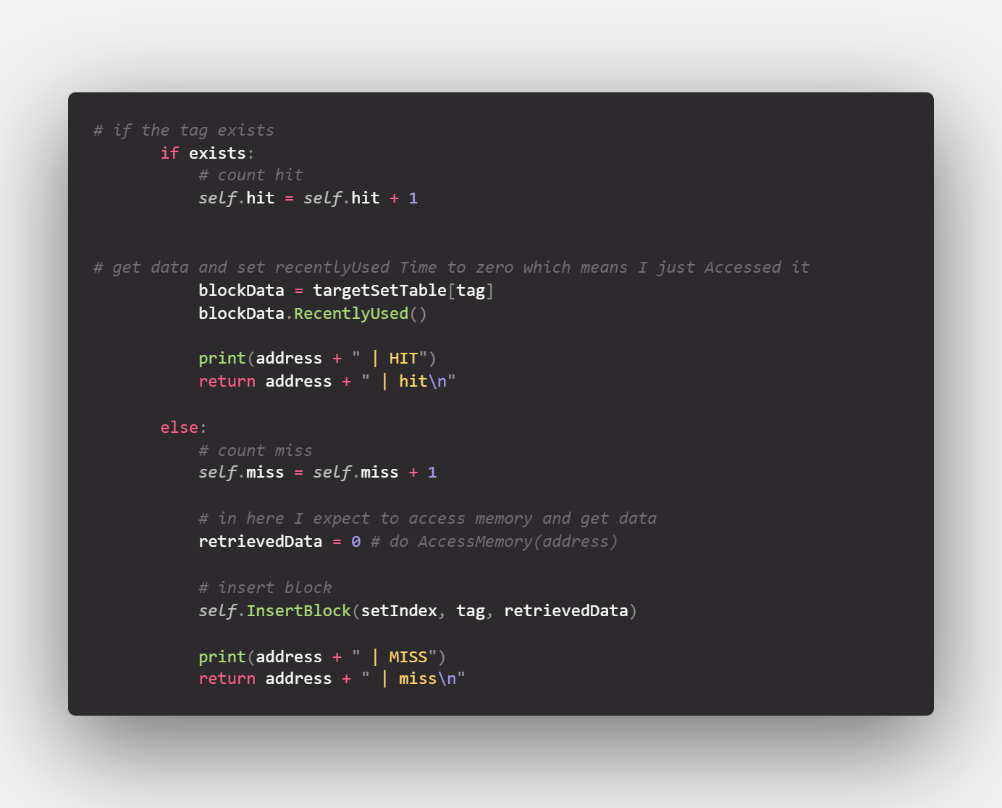
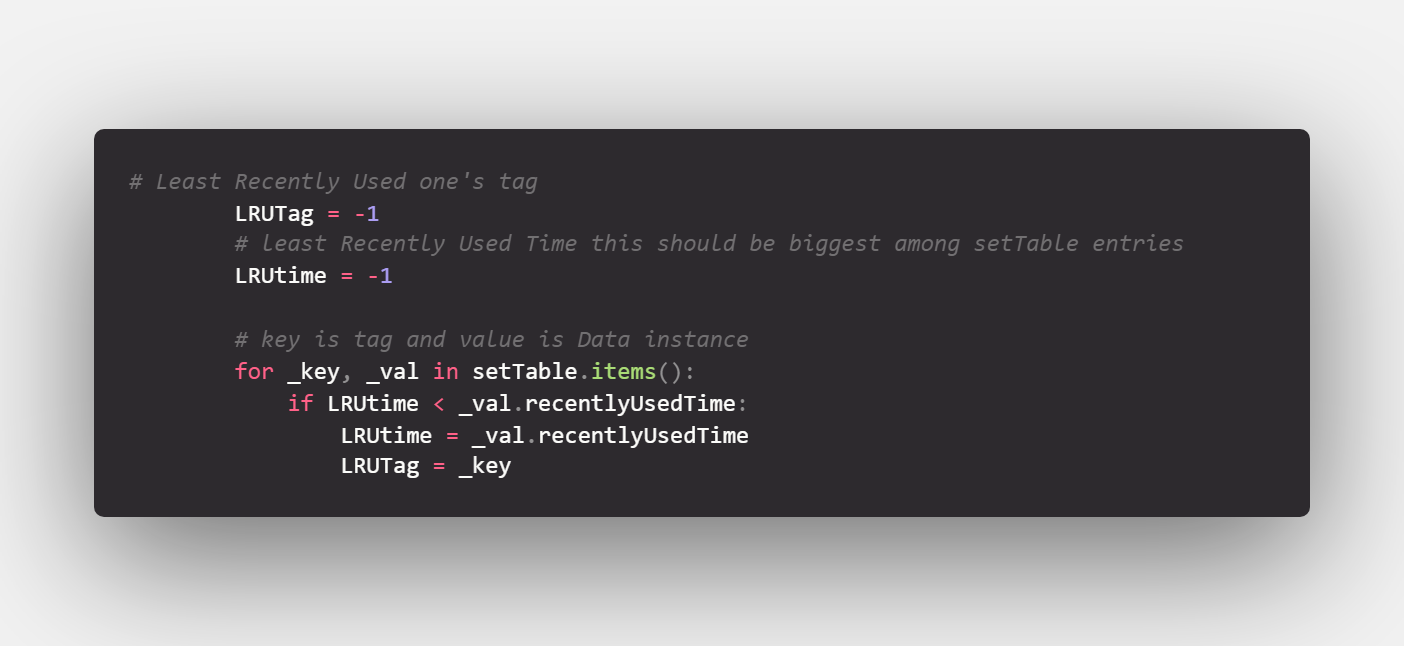
First I used math module for base two log and sys for getting cmd arguments.

import math *# for log*

import sys  *# for getting cmd argument*

1. IsPowerOfTwo(number): function for checking if number is power of 2 (ex) 1, 2, 4, ...  
   if argument is not power of two then prints error message and it exits program
2. ParseInputText(rawText): parse input text that is read from input text file.  
   if int(input\_hex\_string) throws exception it means input is error or unknown value so, I exit program.  
   
3. ParseCommand(inputString, mode=0): parse given inputString and returns number of set, number of blocks in a set, number of word in a block. It also checks if the given argument or command is correct.  
   First It check if the correct number of input is given and correct arguments are given.  
     
   Next, it iterates all input command list and parses everything.  
   
4. Converting Functions (ex) BinaryStringToInt, IntToBinaryString, HexStringToInt, IntToHexString  
   It simply converts to target type.  
   
5. GetFileObject(fileName): Get File Object that points to given filename, It is used for reading input text file.
6. SaveOutputString(filename, outputString): Save outputString to file. It is used for making \*.out file.

5. Class

1. class Data: this is Block's data class
   1. \_\_init\_\_(self, tag=0, memData=0): it is constructor that initializes everything.
   2. CompareTag(self, \_tag): used for comparing tag.
   3. RecentlyUsed(self): it is for LRU. It checks time that it is recently updated. It sets recentlyUsedTime to zero.
   4. ToString(self): convert this object's data to string
2. class Cache: This is class that controls everything about cache. In variable cache dictionary it uses Class Data type as an item of it.
   1. \_\_init\_\_(self, sets, blocks, words): it is constructor that make n-way cache by given input. In this time it also calculates bit mask for ease of extracting bits from block address.
   2. ConstructMasks(self): used when constructor’s is called. It calculates bit mask and shift amount for ease of extraction.
   3. FindBy(self, address): finds block by block address in hex-number-string  
      if the tag exists in the cache table. It updates time variable of that block to check I just accessed to implement LRU.  
      if the tag doesn’t exists. InsertBlock in the Set Table. In InsertTable it determines whether table is full or not. So I will explain LRU implementation in that function part.  
      
   4. InsertBlock(self, setIndex, tag, data=0): inserts Block in the correspoding set table and if the table is full do LRU replacement  
      If it is full, pop LRU block and then insert newly accessed one. If not just inserts block into corresponding setIndex’s set Table  
      
   5. FindLRUBlock(self, setTable): find the least recently used block in the given set Table  
      which is the Block that has largest LRU time value. I updated(plus one) Block Data's LRU time whenever cache has accessed by block address. So I have to find max for LRU block.  
      
   6. ToString(self): it formats string of total outcome.

6. Main Procedures

First, it reads command from CMD

>> inputCommandList = sys.argv

If argument is given by CMD cut first element of argument

Else get Input again.

Next, construct n-way cache based on given command

>> cache = Cache(sets\_InCache, blocks\_InSet, words\_InBlock)

Read input text file

>> fileObject = GetFileObject(inputFileName)

Parse all text string into memoryAccessSequences

>> memoryAccessSequences = ParseInputText(fileObject.read())

Close the opened file that I just used

>> fileObject.close()

Simulate cache with given sequence which is memoryAccessSequences

>> for blockAddress in memoryAccessSequences:

>> # save output whether it is hit or not

>> outputString += cache.FindBy(blockAddress)

Save total output string

>> outputString += cache.ToString()

Print out output

>> print(cache.ToString())

Save output into file

>> SaveOutputString(inputFileName, outputString)

7. More Explain

Cache Implementation:

In Cache Instance, there is table variable.

That variable stores everything about cache. And I thought using Dictionary would do it.

I used setIndex for key and dictionary again for value.

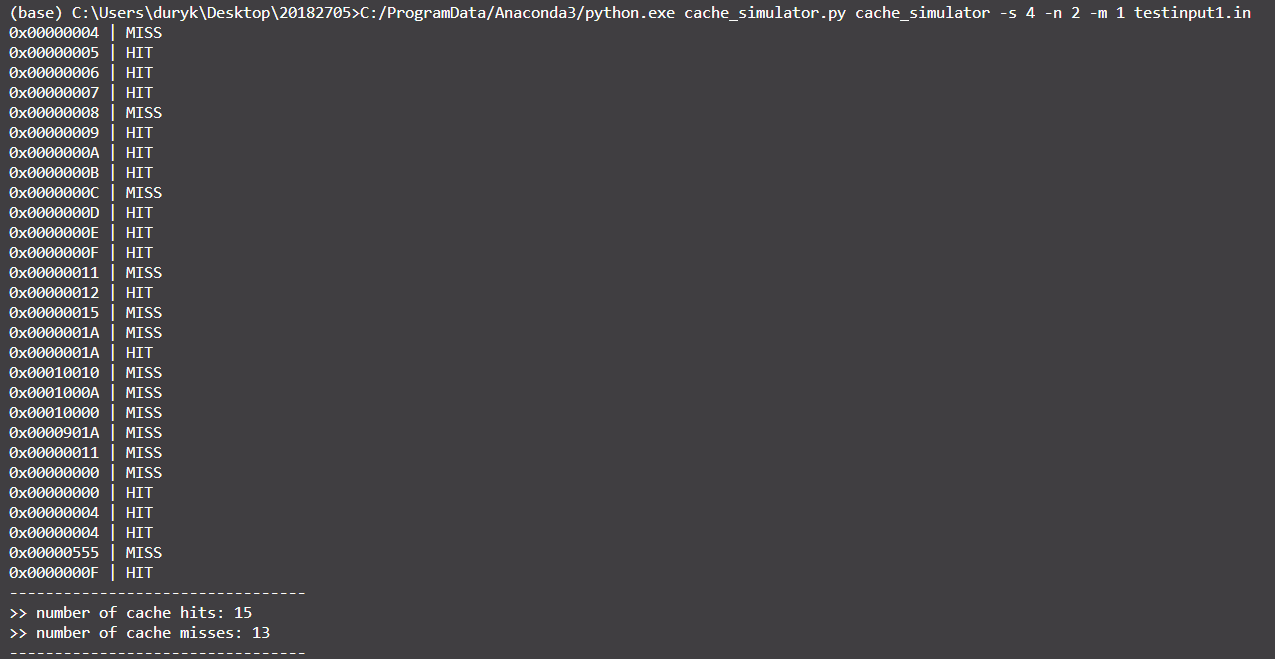
This so table’s dictionary value dictionary. This is because SetTable was also have to store many blocks. I set key which is tag for ease of comparing with input address. Value is Data Class instance. This Data instance is block. It has tag, memoryData and recently accessed data for LRU implementation.

LRU implementation:

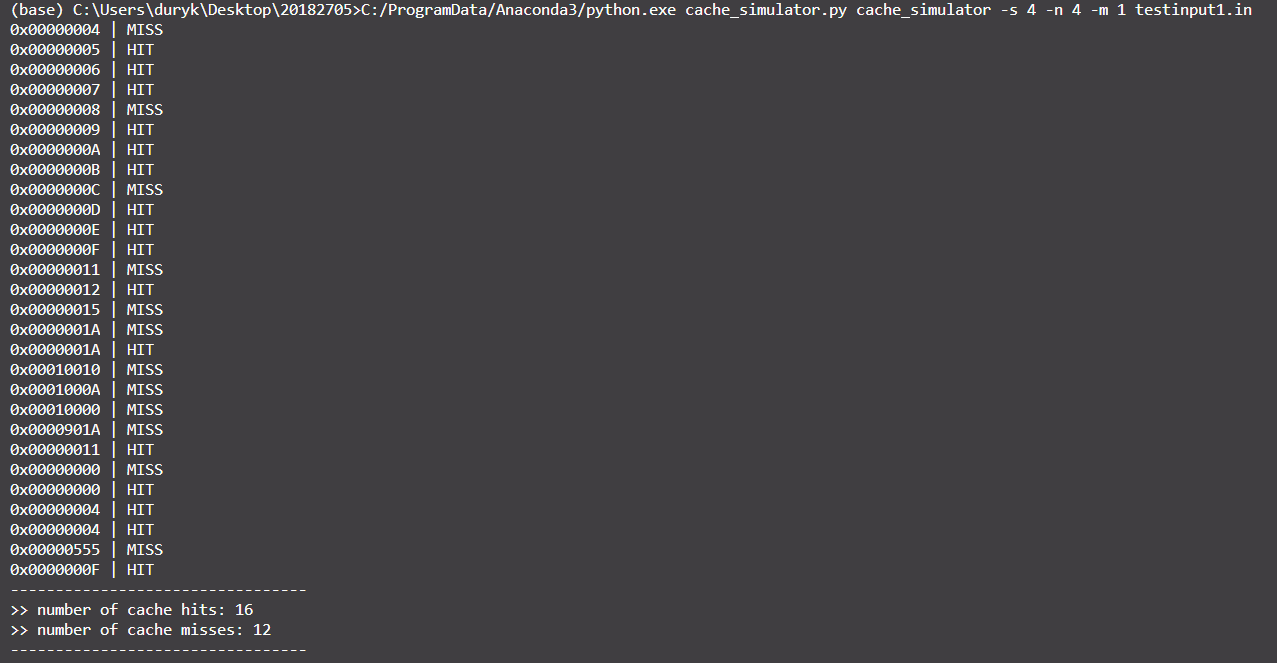
I simply set recentlyUsed variable in data(block) to zero whenever I access that block. And for all another block I plused recentlyUsed one.

8. Output

- 2-way set associative



- 4-way set associative



You can see hit rate increases.