Cache Simulator

Generated by Doxygen 1.9.2

Chapter 1

cache-simulator

A command line cache simulator. Simulates a basic cache, similar to the 351 cache simulator hosted by the University of Washington.

1.1 To launch:

```
python cachesimulator.py "name of ram file"
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Namespace Documentation

3.1 cachesimulator Namespace Reference

Documentation for this program.

Functions

- def flush ()

  The flush function.
- def dump ()

  Cache Dump.
- def mem_dump ()

  Memory Dump.
- def view ()

  Cache View.
- def mem_view ()

  Memory View.
- def init_ram ()

  Initialize Ram.
- def config ()

  Configuration Function.
- def prompt ()

  Prompt.
- def read_write (address)

  Read Writer.
- def read (address)

  Cache Read.
- def write_miss_allocate (address, data)

  Write Miss Allocate.
- def write (address, data)

  Cache Write.
- def activate ()

  Activate.
- def main ()

  Main.

Variables

- list RAM = [ ]
  RAM list.
- list CACHE = [ ]
  CACHE list.
- list RECENTS = [ ]
  RECENCY queue.
- list FREQUENTS = [ ]
  FREQUENCY storage.
- int cache_size = 0
  Cache Size.
- int data_block_size = 0
  Data Block Size.
- int associativity = 0
  Associativity.
- int replacement_policy = 0
  Replacement Policy.
- int write_hit_policy = 0
  Write Hit Policy.
- int write_miss_policy = 0
  Write Miss Policy.
- int cache_hits = 0
  Cache Hits.
- int cache_misses = 0
  Cache Misses.
- string RAM_start = ""
  RAM_start.
- string RAM_end = ""
  RAM_end.

3.1.1 Detailed Description

Documentation for this program.

This is the documentation for my cache simulator program. The cache itself is stored as a 3 dimensional python list. The first one is the set, second is the line, and third is the element in each line.

3.1.2 Function Documentation

3.1.2.1 activate()

```
def cachesimulator.activate ( )
```

Activate.

Final function called by the main. It is responsible for the main flow of the program. It contains an infinite for loop that prompts the user to enter in the command with the prompt function. Then based off of the command that we're given back, it will call one of the functions corresponding to the command.

3.1.2.2 config()

```
def cachesimulator.config ( )
```

Configuration Function.

Function called at the start by default. This is when the CACHE list as well as the global variables are initialized. The cache is set based on the global variables. This function also has slight error checking to make sure that some of the configuration inputs are within the bounds of what we want. If the values are invalid, the user is re prompted.

3.1.2.3 dump()

```
def cachesimulator.dump ( )
```

Cache Dump.

This function is called by the cache dump command. Essentially it writes the data of each line in the cache to an output file called 'cache.txt'.

3.1.2.4 flush()

```
def cachesimulator.flush ( )
```

The flush function.

This function is called when we use the command cache flush. It loops through the three dimensions of the cache list. On each line of a set, it will write the data in the line back to the ram if the dirty bit is one. Then afterwards regardless, it will clear all the values in the cache and set them back to 0 or 00.

3.1.2.5 init_ram()

```
def cachesimulator.init_ram ( )
```

Initialize Ram.

Function called at the start of the program. The program will start with initializing the memory. The memory is always 256 bytes. The function will prompt you for a command to initialize the ram along with the start of the ram that we're copying in and the ending address. The starting address will always be 0x00. Then based off of the ending address it will load in values from the input to the simulated ram in the program.

3.1.2.6 main()

```
def cachesimulator.main ( )
```

Main.

Main function. Is called if this file is the main file that is run. It will call Initialize RAM, then call the Config function, then call the Activate function.

3.1.2.7 mem_dump()

```
def cachesimulator.mem_dump ( )
```

Memory Dump.

Function for dumping the contents of the simulated RAM into a text file named 'ram.txt'

3.1.2.8 mem_view()

```
def cachesimulator.mem_view ( )
```

Memory View.

Function to view the current contents of the memory in the console. Will print it out. Prints out the size as well as the content. Has addresses on the left with the values on the right. Every line contains eight values with the address going by eights.

3.1.2.9 prompt()

```
def cachesimulator.prompt ( )
```

Prompt.

All this function is used for is to print out the prompt and wait for an input. This is called by default by the infinite loop that controls the flow of this program. Once a value has been given, it will then return the value, which will then be used by the main logic.

3.1.2.10 read()

```
def cachesimulator.read (
              address )
```

Cache Read.

This function is the main logic for the cache read function. This is called when we specify the command cache read. First it gets the tag, set index, and block offset from the input address. Then it checks if it's a hit or not. If it's a hit, it will read what we want from the line and output it along with some other information otherwise it will call the read write function if it's a miss.

3.1.2.11 read_write()

```
def cachesimulator.read_write (
              address )
```

Read Writer.

This function is called whenever we have a read miss in the cache. When this happens, we need to take the values from the RAM and throw them into the cache. It checks first if there's an invalid line in that set. If there is, then we write over the invalid. Otherwise we'll choose a block to evict based on policy and then evict that line to make room for the new one.

3.1.2.12 view()

```
def cachesimulator.view ( )
```

Cache View.

Function for printing to the console the current contents of the cache. It will print the basic information of the cache such as sizes or replacement policies in place. Then it will loop through the entire cache and print out the contents.

3.1.2.13 write()

```
def cachesimulator.write (
              address,
              data )
```

Cache Write.

This is the function called when we use the command cache write. What it does is first check if it's a hit or miss. On a hit it will update the dirty bit based off of policy and either write to cache or write to both memory and cache. Otherwise on a miss it will either write directly to the address in the ram or call the write allocate function based off of the write miss policy.

3.1.2.14 write_miss_allocate()

```
def cachesimulator.write_miss_allocate (
              address,
              data )
```

Write Miss Allocate.

Function to do the logic for write miss if we are using the write allocate policy. This is called when we have a write miss. A lot of this logic is similar to having a read miss. We'll first find an invalid line and try to replace that first. Otherwise what it will do is based off of the replacement policy it will evict the line and replace it with the new wanted data.

3.1.3 Variable Documentation

3.1.3.1 associativity

```
int cachesimulator.associativity = 0
```

Associativity.

Global variable for associativity.

3.1.3.2 CACHE

```
list cachesimulator.CACHE = []
```

CACHE list.

A three dimensional list. Used to store the values in the cache. The first dimension is the set. The second is the line in the set. The third is the item in the line. The first two items are the valid and the dirty bit. The third is the tag. Then the remaining elements are the memory bytes stored in the cache.

3.1.3.3 cache_hits

```
int cachesimulator.cache_hits = 0
```

Cache Hits.

Global variable to store the amount of cache hits.

3.1.3.4 cache_misses

```
int cachesimulator.cache_misses = 0
```

Cache Misses.

Global variable to store the amount of cache misses.

3.1.3.5 cache_size

```
int cachesimulator.cache_size = 0
```

Cache Size.

Global variable for the cache size.

3.1.3.6 data_block_size

```
int cachesimulator.data_block_size = 0
```

Data Block Size.

Global variable to store the data block size.

3.1.3.7 FREQUENTS

```
list cachesimulator.FREQUENTS = []
```

FREQUENCY storage.

A list used to store how many times a particular line is used. That is stored as a simple integer. In the LFU policy, the line with the lowest frequency count will be replaced.

3.1.3.8 RAM

```
list cachesimulator.RAM = []
```

RAM list.

A list used to store the values in the 256 byte RAM. These are represented as two character strings.

3.1.3.9 RAM_end

```
string cachesimulator.RAM_end = ""
```

RAM_end.

Used only one to determine the end when initializing RAM.

3.1.3.10 RAM_start

```
string cachesimulator.RAM_start = ""
```

RAM_start.

Used only shortly to determine the start when initializing RAM.

3.1.3.11 RECENTS

```
list cachesimulator.RECENTS = []
```

RECENCY queue.

Queue used for the least recently used replacement policy. Has two dimensions. One is used for the set. In that inner list, it will store a queue of lines. This is in order of which one was accessed least recently. With the most recent one being the last one.

3.1.3.12 replacement_policy

```
int cachesimulator.replacement_policy = 0
```

Replacement Policy.

Global variable for replacement policy.

3.1.3.13 write_hit_policy

```
int cachesimulator.write_hit_policy = 0
```

Write Hit Policy.

Global variable for write hit policy.

3.1.3.14 write_miss_policy

```
int cachesimulator.write_miss_policy = 0
```

Write Miss Policy.

Global variable for write miss policy.

Index