**Documentation**

**Requirements:**

* Python 3.5+
* Built-in modules used: sys, math, collections

**To run the program, run main.py**

**Inputs:**

Main memory size, block size, cache size and set size are taken as inputs. Same values will be used for all the mappings. (Set size is not used in directive and fully associative mapping).

**Assumptions:**

* Single level cache
* Input address is in binary
* No restriction on size or type of data
* Main memory size, block size, cache size and set size are powers of two

**Working of code**

**Main memory is not maintained** i.e. if a block is removed by other, its data is lost. At each step, the mapping provides options to write(W), read(R), print(P) or exit(E). On switching between mappings, the cache **does not** remain the same. Addresses and data stored in a cache are lost once the user switches from one mapping to the other. If an address is not found on reading the cache, it is **not** loaded into the cache.

On replacement, the program prints the data and address deleted. Associative and set-associative mapping uses a **First-In-First-Out** replacement procedure.

**The program terminates if invalid memory size or invalid memory address is provided.**

**Functions and their working:**

**decimalToBinary(n)**

Converts a given decimal number into its binary equivalent. It is used to find line offset, block offset and set value.

**initialize\_cache()**

* Directive: Stores line offset as keys in a dictionary. Tag values are initialised as an empty string and block offsets are stored in another dictionary. Initially, data stored is an empty string.
* Associative: Cache is initialized as an OrderedDict (a dictionary subclass that remembers the order entries were added)
* Set-associative: Cache is initialized as a dictionary with set offset as its keys and an OrderedDict as their value.

**input\_new\_block()**

* Takes input on write(W) instruction.
* Program is terminated if the number of bits in address does not match the word length of the machine.

**cache\_loading()**

* Loads data in the given address.
* If block corresponding to the address is present in the cache, data is written apt position.
* Else if the cache has space for a new block, the corresponding block is loaded.
* Else *replace\_block()* is called.

**replace\_block()**

* Replaces block.
* Prints the tag removed.
* Uses a First-In-First-Out replacement method in associative and set-associative.

**cache\_searching()**

* Searches the given address in the cache
* Prints MISS if the address is not present
* Else, prints HIT Data: <data>
  + If the given address does not contain any data, <data> is empty

**print\_cache():**

* Prints all the addresses and data stored at that instance in the cache.
  + In the case of set-associative mapping, ‘----’ is printed after every set for the convenience of the user.