CACHE IMPLEMENTATION (C++)

Rupanshu Yadav
2019475

BASIC INFO :

- Empty data is considered to be 0 i.e. if no change is made to the memory block then its data is 0.
- Programm is menu-driven, each time you jump back to the main menu all the previous change to a cache is destroyed.
- You can choose any cache you want from the main menu.
- Each address byte can hold data between 0-255(bits) and is indivisible.
- Every element of the count array in each class is incremented during every *read* and *write* operation but not during *print* operation. Also, there are not incremented past 1000000007 to prevent overflow.
- Each cache consist of three main operations:
  - ➢ Read: Reads the value at the given address, outputs 'Read Miss' if the address is not in the cache.
  - ➢ Write: Writes the value given value at the given address. Gives an overflow error if the value is more than 1 byte.
  - ➢ Print: Prints the content of the cache. The format is the entry is empty then output 'empty' else output 'Block <TAG> <{block entries}>' where each element in block entries is 1byte(0-255).
- The rest of the small details are explained using comments in the code.

CLASSES:

- CLASS: Associative, *implements a fully associative cache.*
  Attributes:
  - ➢ *blocksize*: the size of each block
  - ➢ *size*: the size of the cache
  - ➢ *cachelines*: the number of cachelines.
  - ➢ *tag* (vector<long long>) contains the tag of all the data entries.
  - ➢ *data* (vector<vector<long long>>) an array of size *cachelines* containing arrays of size *blocksize* containing all the data of a block.
  - ➢ *count*(vector<int>) counts the number of uses of cache where index *i* is not touched. Useful while evicting using the LRU scheme.

- ➢ *empty*(vector<bool>) stores if the entry in cache at index *i* is empty(true if it is).

Methods:

- ➢ Associative(int cacheline, int siz, int blocksiz) :
  The constructor of the class used to initialize class attribute {*blocksize, cachelines, size*} with function arg {cacheline, siz, blocksiz} respectively. Also, initialize attributes {*tag, data, count, empty*} with sizes {*cachelines, (cachelines, blocksize), cachelines, cachelines*} respectively, initially all the entries in *count* and *empty* are 0 and true respectively.
- ➢ void incre():
  Increments all the entries in the count array
- ➢ long long read(long long add, long long off)
  Reads the value at the tag (add) with offset(off) within the block. It does this by checking if any of the non-empty entries in the cache has its *tag* equal to add.
  Returns :
  - -1, for a Read Miss signal
  - x, where x is the value of the desired address

- ➢ void write(long long add, long long off, long long value)
  - add: tag of the address
  - off : offset in a single block.
  - value: value to be written.

  This function first checks if the address exists in the block, if yes then write the value, else find an empty space and create the address of the block there and write the value, if no block is empty then the block with the maximum *count* is deleted and a new block is inserted according to the LRU scheme.
- ➢ void print()
  Prints the content of the whole cache, in a format mentioned already.

- CLASS: Direct, *implements a direct-mapped cache.*
  Attributes:
  - ➢ *blocksize*: the size of each block
  - ➢ *size*: the size of the cache
  - ➢ *cachelines*: the number of cachelines.
  - ➢ *tag* (vector<long long>) contains the tag of all the data entries.
  - ➢ *data* (vector<vector<long long>>) an array of size *cachelines* containing arrays of size *blocksize* containing all the data of a block.

- ➢ *empty*(vector<bool>) stores if the entry in cache at index *i* is empty(true if it is).

Methods:

- ➢ Direct(int Cachelines, int Size, int Blocksize) :
  The constructor of the class used to initialize class attribute {*blocksize, cachelines, size*} with function arg {Cachelines, Size, Blocksize} respectively. Also, initialize attributes {*tag, data, count, empty*} with sizes {*cachelines, (cachelines, blocksize), cachelines, cachelines*} respectively, initially all the entries in *count* and *empty* are 0 and true respectively.

- ➢ long long read(long long add, long long add2, long long off)
  Reads the value at index add2 in the data array with offset equal to off.
  Returns :
  -1, for a Read Miss signal
  x, where x is the value of the desired address

- ➢ void write(long long add, long long add2, long long off, long long value)
  add: tag of the address
  add2: index in the cache as a result of direct mapping
  off : offset in a single block.
  value: value to be written.
  If the value at add2-th entry in the data is empty of has different tag then add, then the function overwrites that entry otherwise make change the same entry at an offset equal to off.

- ➢ void print()
  Prints the content of the whole cache, in a format mentioned already.

- CLASS: NWay
  Attributes:
  - ➢ *blocksize*: the size of each block
  - ➢ *N*: the size of each set
  - ➢ *size*: the size of the cache
  - ➢ *cachelines*: the number of cachelines.
  - ➢ *tag* (vector<long long>) contains the tag of all the data entries.
  - ➢ *data* (vector<vector<vector<long long>>>) a vector of sets, each set contains data entries, where each data entry have a size *blocksize*.

- ➢ *count*(vector<int>) counts the number of uses of cache where index *i* in the *tag* array is not touched. Useful while evicting using the LRU scheme.
- ➢ *empty*(vector<bool>) stores if the entry in *tag* array at index *i* is empty(true if it is).

Methods:

- ➢ NWay(int Cachelines, int Size, int Blocksize, int n) :
  The constructor of the class used to initialize class attribute {*blocksize, cachelines, size, N*} with function arg {Cachelines, Size, Blocksize, n} respectively.
  Also, initialize attributes {*tag, data, count, empty*} with sizes {*cachelines, (cachelines/N, N, blocksize), cachelines, cachelines*} respectively, initially all the entries in *count* and *empty* are 0 and true respectively.

- ➢ void incre():
  Increments all the entries in the count array

- ➢ long long read(long long add, long long add2, long long off)
  Check each entry is the add2-th set to see if any of them matches the tag-add, if yes return that value else return -1.
  Returns :
  - -1, for a Read Miss signal
  - x, where x is the value of the desired address

- ➢ void write(long long add, long long add2, long long off, long long value)
  - add: tag of the address
  - add2: index of the set as a result of direct mapping performed on sets
  - off : offset in a single block.
  - value: value to be written.

  Check if any entry in the add2-th set matches the tag-add, if yes then perform write operation on that entry using offset-off. If no matches are found the check to see if any block is empty, if yes insert the block there and write the data with offset-off. If no empty blocks are there evict the block with most count and insert the new block the write the data into it.

- ➢ void print()
  Prints the content of the whole cache, in a format mentioned already.

HELPER FUNCTIONS:

- long long toint(string s): Converts a binary string s into an integer, and returns that value
- string tobinary(long long v, long long sz):
  v: the integer to be converted into binary
  sz: the size of the binary string
  Converts the integer v into a binary string of length sz, length is ensured by putting zero before the most significant bit of sz. The value sz is large since the value v is always taken from a string a size less than 32.
- string makesize(string s, long long sz): increases the size of the binary string s to sz by putting ‘0’ before the most significant bit of sz.