Shahriyar Valielahi Roshan
861241935

General Structure:

In this part, I briefly introduce Classes and their functions that I used in my design.

- TDCS.java
    - Create Trace and Config Object
    - Create Simulation Config Object
    - Create Simulation Trace Object
    - Create Simulation Trace Parser Object
    - Create Simulation Config Parser Object
    - Create Cache Unit

  This is the interface of our user and other classes. This file shows the user Introduction Messages, and get the address of Trace and Config from the user. Actually, it instantiates other classes to perform operations like, creating a file object for a trace and a Config file, then it will initialize parser to parse each file and create two simulation Object that contains necessary information for our cache unit to simulate the cache, and finally it calls the writer to write back the result. You can understand the code by just looking at the source which contains required comments.

- fileUnit.java
    - Read file from specified address by user [User should put file in the specified location by the program and write the name of file in the terminal]
    - In the file object we have a database of lines to store each lines in the fileUnit into it. I used this approach to reduce cost of reading file, but it consumes memory as we face large traces, but it's faster.
    - We can also update the fileUnit database [write Object], to write result at once.
    - Please note that each file has a separate fileUnit object
    - Result will be written in "directory+result+traceName.drsvr"
    - Details are not important, because they are just programming stuff

- parserUnit.java
    - anything related to parsing has been implemented here
    - It will detect two trace type. One type is standard, and one is with the format of traces that you proposed to us.
    - It is error tolerant, it means if there exist an error inside a trace file or Config file, the program simply uses the default values, but simulation may be different from your intentions, so the program gives a warning every time that it faces an error to tell the user that it changes the file.
    - It has functions to tokenize, conversion between Sting and numeral values like integer and long, and checking mechanisms to verify the input, and of course, getting data and parsing them so the simulator can understand them.
    - It constructs a simulator unit object for each file to store configurations needed for the program, and to use future functions on those configurations.
    - Again details are just programming tricks! ☺

- simulatorUnit.java
    - Stores values that needed for the Simulator, actually simulatorUnit is an abstraction that has been created for passing large number of parameters to the cache Unit with protection to avoid unnecessary changes.
    - It includes an array of trace Object and Config Object
    - One trace Object for each line stores two parameters, operation type + address;
    - Config object stores the configuration parameters from the trace file, and it will be initialized with the default values to avoid any problem when the input configuration has problems.

- cacheUnit.java
    - Actual simulation will run in this class, it configures the cache organization, reference different blocks, and choose the victim block to replaces on miss, and also update statistics. And numerous functions that make it work properly ☺ I do not have time to write details of code, and you do not have time to read them, so let's just skip details of code, you can find comments inside the code, so you can easily understand what is what.
    - StatObject stores different statistics
    - Replacement unit handle replacement policies
    - ReferenceUnit refer to to cache to detect hit or miss on each operation
    - Decoder unit decodes trace addresses and divide it to tag, index, and offset.

Config, Decode, and Reference:

- Trace Object
- Config Object
- Decoder Unit
- Reference Unit

For each approved trace line, we have a trace object with has two variables, operation and address. As I asked you in the piazza, the operation part of our traces are not important in our project, so they work the same, but I added different type of hit and miss for each of them.

By using the Config Object, the cache unit configures the cache structure into an 2D array which includes "number of sets" Row, and "number of ways" column.

$$Number\ of\ Blocks = \frac{CachSize}{Block\ Size}$$

$$Number\ of\ Sets = \frac{Number\ of\ Blocks}{Number\ of\ Ways}$$

$$offset = \log_2 blockSize \qquad index = \log_2 Number\ of\ Sets,$$
$$Tag = AddressBits - index - offset$$

When we have a fully associative cache, number of ways is equal to the number of blocks, because we have only one set.

After cache configured properly, we should use a decoder to decode each address base on the offset, index and tag bit. For our project, offset bits are used just to detect tag, and they are not used for placement of data. But in real world, each data will write in the byte that is specified in the offset part of address, but in our project, offset value is not important and knowing number of offsets are enough. [I just store tags inside the cache elements]

Fully Associative

- We do not have index, we have only one set
- Search for tag, if found hit
- If not found, get a free block (way), if we could get a free block, cold miss
- If we do not have free block and all blocks are filled, then we should get a victim block, and replace its content. [Capacity Miss]

Direct Mapped

- Each set has only one way [No replacement policy, Miss on tag mismatched]
- Use index to select block [set]
- If "content = -1", Cold Miss
- Else if (tag matched) HIT
- Else Conflict Miss

Set Associative

- Each set has several blocks [Number of Ways]
- Use index to find the set
- If "content = -1", Cold Miss
- Else if (tag matched) HIT
- Else Conflict Miss [Choose a victim to replace]
- The difference between Set Associative and Direct Mapped is that we read conflict miss when we run out of blocks in each set, because we have more block in each set

Replacement:

RANDOM

- I will generate a random number in the range of [0 – Number of ways] and select that number to select the victim

LRU

- I have a linked List which stores ways index for each set with the order that they assigned. When we re-access a block, we use seachTag function to detect its position, then we remove it from the linked list and put it at the front of the list. The front of the list shows the most recently use, and the end of the list shows the least recently used block, so to select the victim, we simply remove the last recently used block.

FIFO

- FIFO uses the similar approach to LRU, but we do not search for a tag on hit, and change the order while we reach hit. List keeps the queue with the same order as they first allocated. Again, we get the last block in the queue as the first element that has been pushed into the queue. [First Come, First Kicked out ☺]

How to run:

- JRE 1.7 is required! [I think it preinstalled on machine]
- Java -jar TDCSim.jar
- Copy Trace and Config in the Directory that the simulator tells you
- Enter trace name file with extension: Example: DRSVR.trace
  - Format1: type address
    - Set the addressline in the Config to 32bits
  - Format2: [14 token] token 9: Address token 13: type [Long traces]
    - Set
- Enter Config name file with extension: Example: DRSVR.config
  - You should follow guide inside the sample Config
- Done! The simulator will run can you will get result at the location that the program will tell inside the program

Evaluation and results

- I ran several small traces to evaluate the function of simulator for small number of sets, because of the nature of my program, when it works properly for the small number of sets, it will work properly for larger number of cells and larger traces too,
  - The only think that I worry about is boundaries for long and integer numbers, I do not face this problem yet.
  - This is a one-man job, I faced two problems:
    - Time
    - You cannot find your own mistakes ☺
  - I tried my best to write a program that is scalable, reconfigurable in future, and also efficient, I hope you take into consideration my efforts when you want to grade my job ☺

For all test cases I used this configuration:

	Direct	4-way	Fully
Cache Size	2048	2048	2048
Block Size	256	256	256
Ways	1	4	8
Blocks	8	8	8
Set	8	2	1
AddressLine	32	32	32
Tag bits	21	23	24
Index bits	3	1	0
Offset bits	8	8	8

Test Case 1:

Direct Map: PASSED

Check direct.trace direct.map

Please check the provided excel file to know about pattern

	Round							
	1	2	3	4	5	6	7	8
0	M	H	M	H	H	M	M	H
1	M	H	M	H	M	M	M	H
2	M	H	M	H	H	M	M	H
3	M	H	M	H	M	M	M	H
4	M	H	M	H	H	M	M	H
5	M	H	M	H	M	M	M	H
6	M	H	M	H	H	M	M	H
7	M	H	M	H	M	M	M	H

Hit:	**28**
Miss	**36**
cold:	**8**
Conflict:	**28**
Total Inst:	**64**
MissRate:	**0.5625**
Average Latency:	**57.25**

Set Associative + Fully Associative: PASSED

Check setLRU.trace setFIFO.map

Please check the provided excel file to know about pattern

Please check the provided result_ setLRU.drsvr for results

I used the trick here, To compare LRU and FIFO, I accesses to blocks in set 0, I HIT Block 0 three times, and then a Miss , and after that again access block1;

LRU: H H H M (Victim = 1) H
FIFO: H H H M (Victim = 0) M

So FIFO should have 1 more victim. For Random, we cannot estimate, but base on my logic that I return a victim way for new free place, there must be no difference from the layer above getVictim function.

FIFO		LRU	
Hit:	**11**	**Hit:**	**12**
Miss	**18**	**Miss**	**17**
cold:	**8**	**cold:**	**8**
Conflict:	**10**	**Conflict:**	**9**
Total Inst:	**29**	**Total Inst:**	**29**
MissRate:	**0.620689655**	**MissRate:**	**0.586206897**
Average Latency:	**63.06896552**	**Average Latency:**	**59.62068966**

For fully associative, the same rule applies except that instead of conflict, we have capacity miss, we have more ways and also we have only one set.

NOTE: I also ran gcc-10M, it will take time sometime between getting trace and Config file, on my system, it took me 14 second before the simulator asked for the Config file, and all simulation took less than a minute for me. DONE ☺