**Computer Organization 2019**

**HOMEWORK 6**

系級: 資訊112 學號: F74082086 姓名: 王維瀚

**問題(Question)**

Q1. How do you know the number of block from input file?

Ans: I first change the input cache size from Kib to Byte (\*2^10), and then divide the cache size by the block size. I can then get the result as number of blocks.

Q2. How do you know how many set in this cache?

Ans: First, I check the input associativity. If it is 0 (direct-mapped), then the number of sets will just be the number of blocks since each set contains one block. If it is 1 (4-way set-associative), then the number of sets is the number of blocks divided by 4 since each set contains 4 blocks. Last, if the associativity is 2 (fully-associative), then there will just be 1 set, and all the blocks are in this set.

Q3. How do you know the bits of the width of the Tag ?

Ans: The address width is 32 bits, and I have to remove the block offset (the last m bits of address for block size = 2^m Bytes) through dividing the address by block size. Then, I can get the tag through dividing the (32-m)-bits offset-removed address by the number of sets. The latter method will help us get rid of the lower n-bits of the offset-removed address, which is the index part.

Q4. Briefly describe your data structure of your cache.

Ans: I use arrays of vector to implement the cache. For number of sets = n, I create n vectors, each representing a set. Every element of vector represents a block in the set, storing the corresponding tag. Roughly the vector resembles a queue, the push\_back and erase are the counterpart of enqueue and dequeue.

Q5. Briefly describe your algorithm of LRU.

Ans: For the FIFO algorithm, I insert new tags at the end of the vector and remove replaced tags at the beginning of the vector. The algorithm of LRU is based on the FIFO algorithm. When a specific tag is traced and it’s hit, I’ll move it to the end of the vector since it’s used most recently. The moving involving erasing the tag from the original position and push it back at the end.

Q6. Briefly describe your algorithm of your policy.

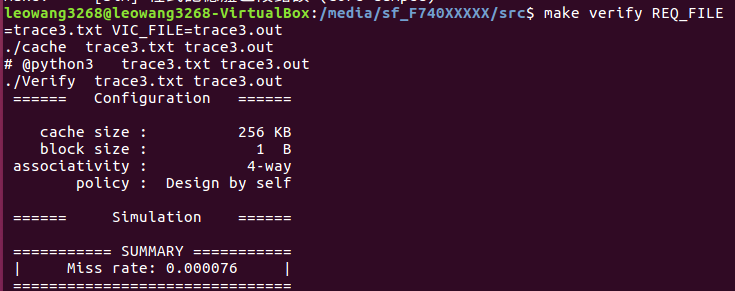
Ans: My policy is LFU (Least Frequent Used), which is intuitive. During each cache hit, I will increase the access time of the according block. Every time I need to replace the tag, I will choose the blocks with the least access time to be replaced with. If there are several blocks with same least-access time, I choose the block that is closest to the beginning of the vector.

Q7. Run trace2.txt, trace3.txt and then makefile to get the miss rate and put it in your report.

trace2.txt:



trace3.txt:



**心得(Report)**

(請寫下完成本次作業的心得、學到哪些東西、困難點的部分。)

(Please write your learned lesson and conclusion, and difficult point.)

1. Vector is a useful data structure, but it may be inefficient when we want to delete element in the middle of the vector and move the following elements forward.

2. If cache hit happens, there must be no victim, and we can continue next trace directly.

3. If cache miss happens and the set we want to insert hasn’t been full, we can also continue next trace after inserting the tag into the set. Size of the set will only be changed under this circumstance.

4. If cache miss happens and the set we want to insert has been full, we must apply the replaced policy. However, if the cache is direct-mapped, all policies are the same since we just replace the only old value with the new value. When applying LRU algorithm, because we have moved our most-recently-used elements to the end of the vector during each cache hit, we can perform the same operation as FIFO, due to the fact that the beginning of the vector is now the least-used element.