Lab 7&8 Cache Lab

20210815 정재현

A. csim.c

Cache lab을 시작하기 앞서 simulation을 하기 위해서는 cache에 대한 깊은 이해가 필요하다. 먼저 cache valid, tag가 존재하고 replacement에서 LRU를 사용해주기 위해 가장 적게 쓰이는 빈도를 나타내는 LRU라는 변수를 선언해주었다. 이 세가지는 Line을 구성하도록 하고, line들을 구성하는 set structure를 만들었다. 이러한 set들이 모여 line의 수와 set의 수를 나타내도록 n\_set, n\_line을 나타내는 변수를 선언하고 set structure를 포함하도록 cache structure를 구성 하였다.

기본적으로 s,b,t를 나타내는 변수를 전역변수로 선언해주었고, 이를 나타내는 S,E도 선언했다. 이어서 simulation에 필요한 hit, miss, eviction의 수를 세주기 위해서 n\_hit, n\_miss, n\_evict를 선언해주었다.

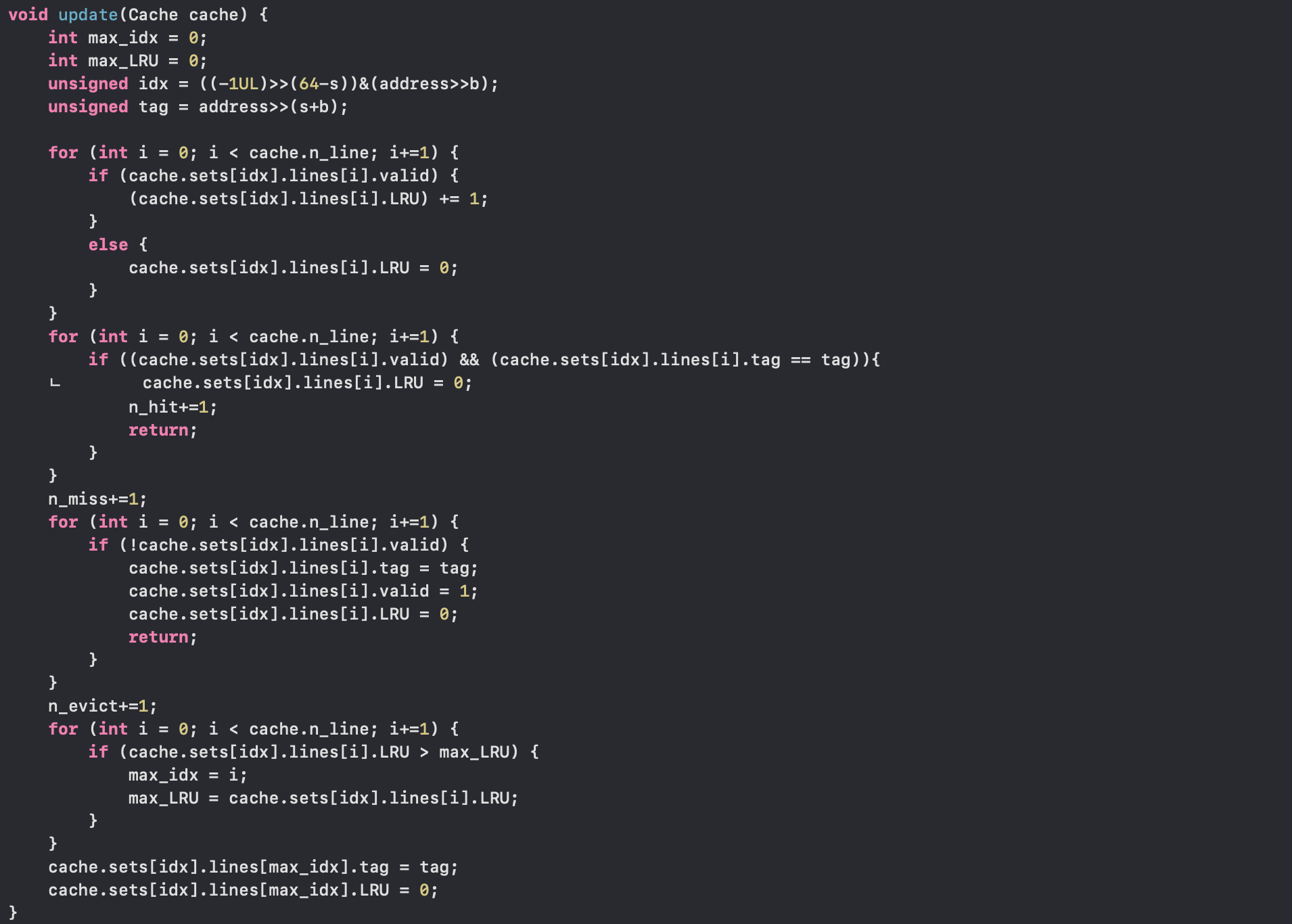
Main 함수로 들어가기 앞서 나는 여러 전역변수와 line, set, cache라는 구조체를 선언해주었다. 이후 update라는 함수를 구성하였다.

먼저 address라는 주소를 받고, 이것은 t s b 비트로 구성되어 있기 때문에 address를 b만큼 shift right를 시켜 index를 구성하도록 하고, s+b만큼 shift right를 시켜 tag를 구성하도록 하였다. 이후 반복문을 사용하여 모든 element들에 대하여 valid\_bit이 1이라면 LRU를 증가시키도록 한다. 아니면 LRU를 0으로 초기화 해준다. 앞의 과정들이 끝나면 setup 과정이 마무리 된 것이다. 이후 update에서는 hit, miss, eviction에 대해 count를 해줘야한다.

반복문을 사용하여 hit 인경우를 count하여 n\_hit를 증가시킬 것이다. 먼저, cache의 valid가 1이고, tag가 같다면 LRU를 0으로 초기화 시켜준다. 그리고 n\_hit를 1 증가시켜주고 return하여 update를 빠져나오도록 한다.

이후의 과정은 miss인데, 이 경우 hit에서 return이 안일어났다는 것을 의미하기 때문에 n\_miss를 1 증가시켜주고 시작한다. 반복문을 사용하고 조건문으로 valid하지 않다면 cache의 tag를 tag로 초기화 시켜주고, valid=1로 만들어준다. 그리고 LRU=0으로 만들어주며 return을 시켜 update를 빠져 나온다.

마지막인 eviction이 있다. 이 과정은 hit과 miss 파트에서 return이 일어나지 않은 경우를 의미하며, 먼저 n\_evict를 1 증가시킨다. 여기서는 replacement를 사용하는데 가장 큰 LRU값과 그에 대응하는 idx값을 찾는 것이 목표이다. 조건문을 이용하여 가장 큰 max\_LRU값보다 LRU가 크다면 그 때의 idx값을 max\_idx로 지정하고, 그 LRU 값을 max\_LRU로 초기화해주게 된다. 그 이후 idx와 max\_idx를 사용하여 접근한 cache의 tag를 tag로 업데이트 해주고, LRU=0으로 해주며 update 함수를 끝내게 된다. 이렇게 되면 hit, miss, evict를 분류할 수 있다.



위의 사진이 setup과 hit, miss, evict부분을 포함한 update 함수를 나타낸다.

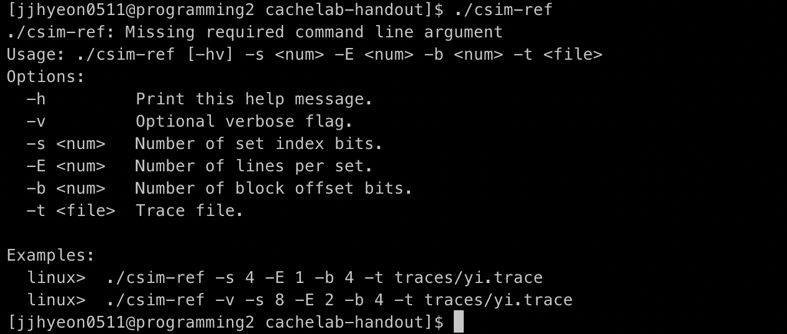
마지막으로 main함수이다. Main 함수에서는 입력받는 값의 명령어를 이용하여 simualtion하도록 update를 호출한다.



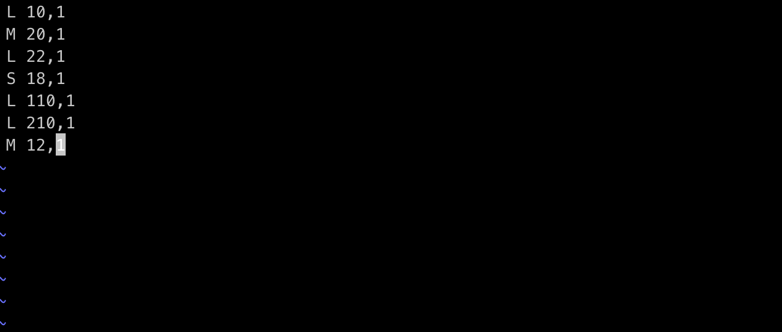
메인 함수는 먼저 위와 같이 처음에 구성할 수 있다.

파일 입출력을 위해 fp를 정의하고 opt를 통해 명령어를 받아오게 된다. 이때,

Cachelab-handout에서 ./csim-ref를 입력하면 다음과 같은 결과를 얻을 수 있다.



따라서 s,E,b,t와 같은 값에 대해 입력을 받음을 알 수 있었고, 추가적으로 trace로 들어가서 vi yi.trace를 입력하면 아래와 같은 결과를 볼 수 있었다.



이를 통해 알 수 있는 것은 우리는 while에서 opt를 사용하여 명령어를 읽어오고 인풋 값이 s,E,b,t에 따라 다른 저장방식을 사용하도록 해야한다. 그 이후 fp가 null인 경우는 파일을 제대로 읽지 못했다는 것을 의미하기 때문에 오류 메세지를 출력하도록 하였다.

해당 반복문을 이용하여 얻은 cache의 n\_line, n\_set을 이용하여 E,S를 구성하도록 하였고, 32라는크기에서 s+b만큼 빼서 t를 구성하도록 하였다. 이때, E,t,s,b 중에 하나라도 0이 존재하게 되면 잘못된 cache 세팅을 나타내기에 따로 조건문을 사용하여 오류 메세지를 출력하도록 했다.

이후 readme 파일을 통해 cache를 사용하면서 동적할당을 하도록 구성을 하였고, Set이라는 structure 에 대해 할 때에는 S만큼 곱해주고, Line에 대해 동적할당 할 때에는 E를 곱해주도록 구성해보았다.

그 다음으로는 반복문을 이용하여 명령어, address, block size를 입력받도록하며 update 함수를 호출하여 hit, miss, evict를 판별하도록 한다. 이때, I 명령어의 경우 cache값의 변화가 없기 때문에 continue를 사용하였고, M의 경우는 update 함수를 호출하도록 하였다. 이때, S의 경우도 update를 호출하는데 M의 경우는 cache에 2번 접근하도록 한다.

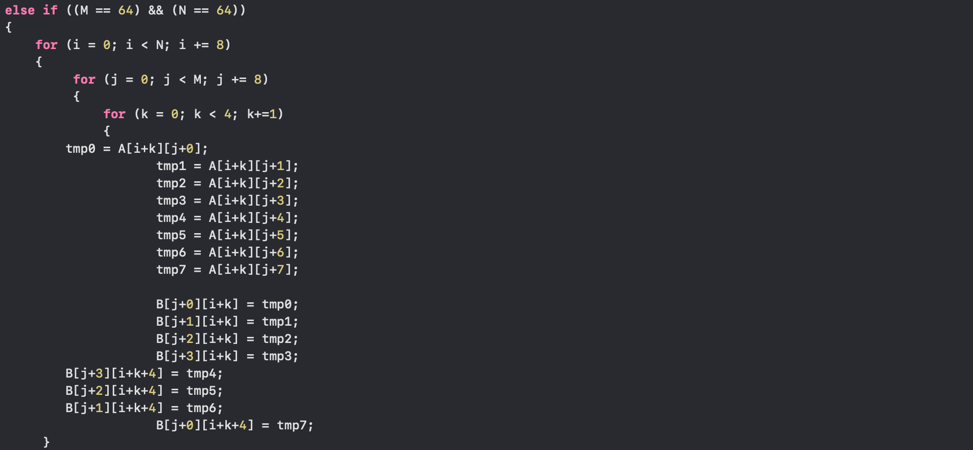
이렇게 해서 명령어와 cache에 접근을 한 이후 동적할당을 했던 cache들을 모두 free를 통해 동적할당 해제를 해주게 된다. 그리고 fp를 닫는다. 마지막으로 printSummary함수에 n\_hit, n\_miss, n\_evict라는 인자를 넣어 호출을 하고 return하여 정상적으로 결과를 도출 시킨다.

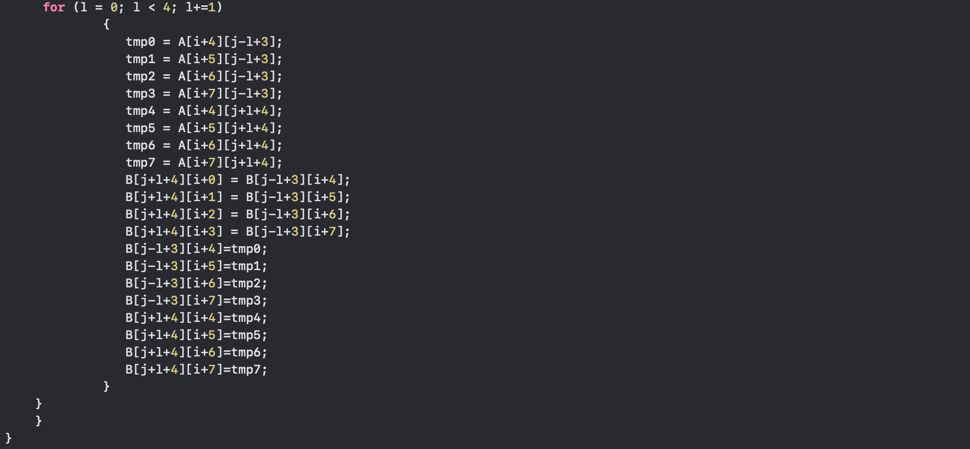
B. trans.c

Trans.c에서는 먼저 cache의 구성요소를 알아야한다. s = 5, E = 1, b = 5을 나타내는데 이를 통해 set은 총 32를 나타내고. E=1이기 때문에 Direct Mapped Cache라는 것을 알 수 있다. 그리고 우리가 다뤄야할 M,N의 종류는 32x32, 64x64, 61x67 지역성을 고려해야한다. 이때, 매트릭스를 transpose 시켜야한다.

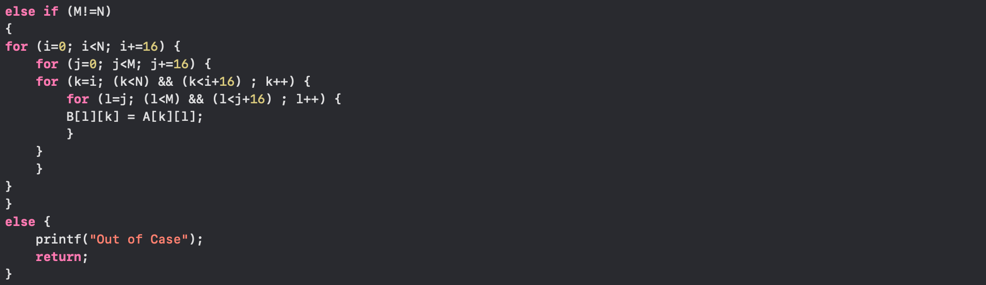


32x32의 코드는 위와 같이 나타낼 수 있다.





64x64의 코드는 위와 같이 나타낼 수 있다. 지역성과 Tmp를 이용하여 A,B를 스왑하는 방식으로 구현하였다.



61x67의 코드는 위와 같이 나타낼 수 있다 마찬가지로 지역성과 tmp를 이용하여 A,B를 스왑하는 방식으로 구현하였다.

C. 마치며

이번 Lab 과제에서는 캐시에 대한 전반적인 이해와 cache set line 등에 대한 개념과 s,e,b bit와 valid, tag , index 등에 대해서 알 수 있었다. 지역성을 활용하고 LRU를 사용하고 hit, miss, evict에 대해 자세하게 알 수 있었고, 앞으로 캐시에 대한 개념을 더 적용해보고 싶다.