**ㅌCache lab report**

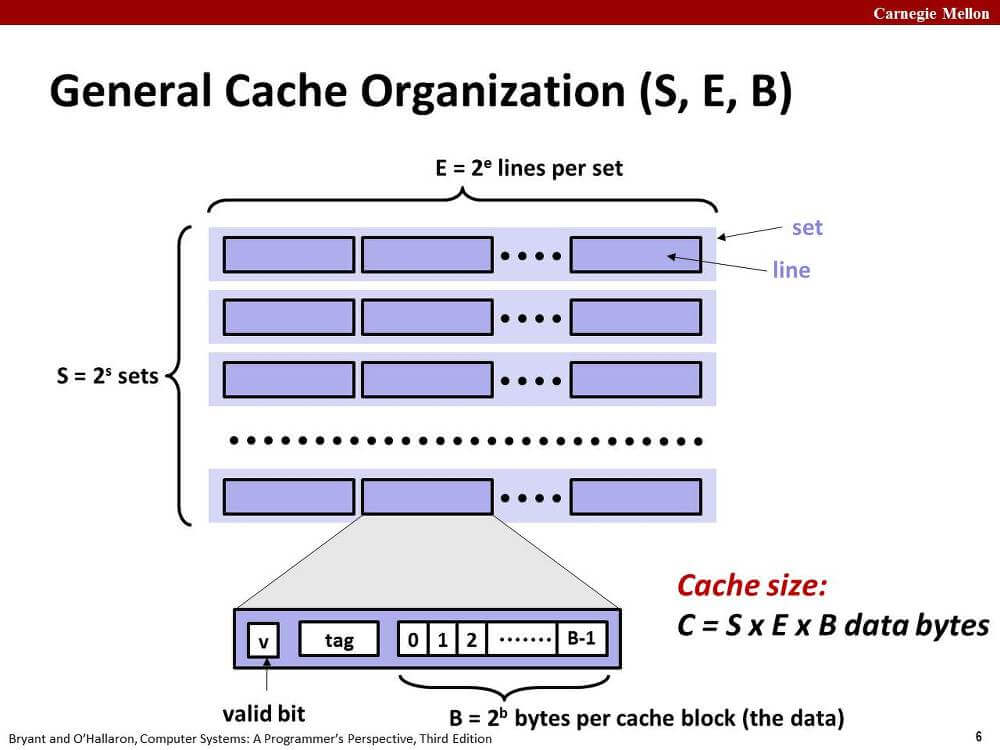
20210774 김주은

<lab 강의 내용 요약>

1) Cache Memory (Lab8)

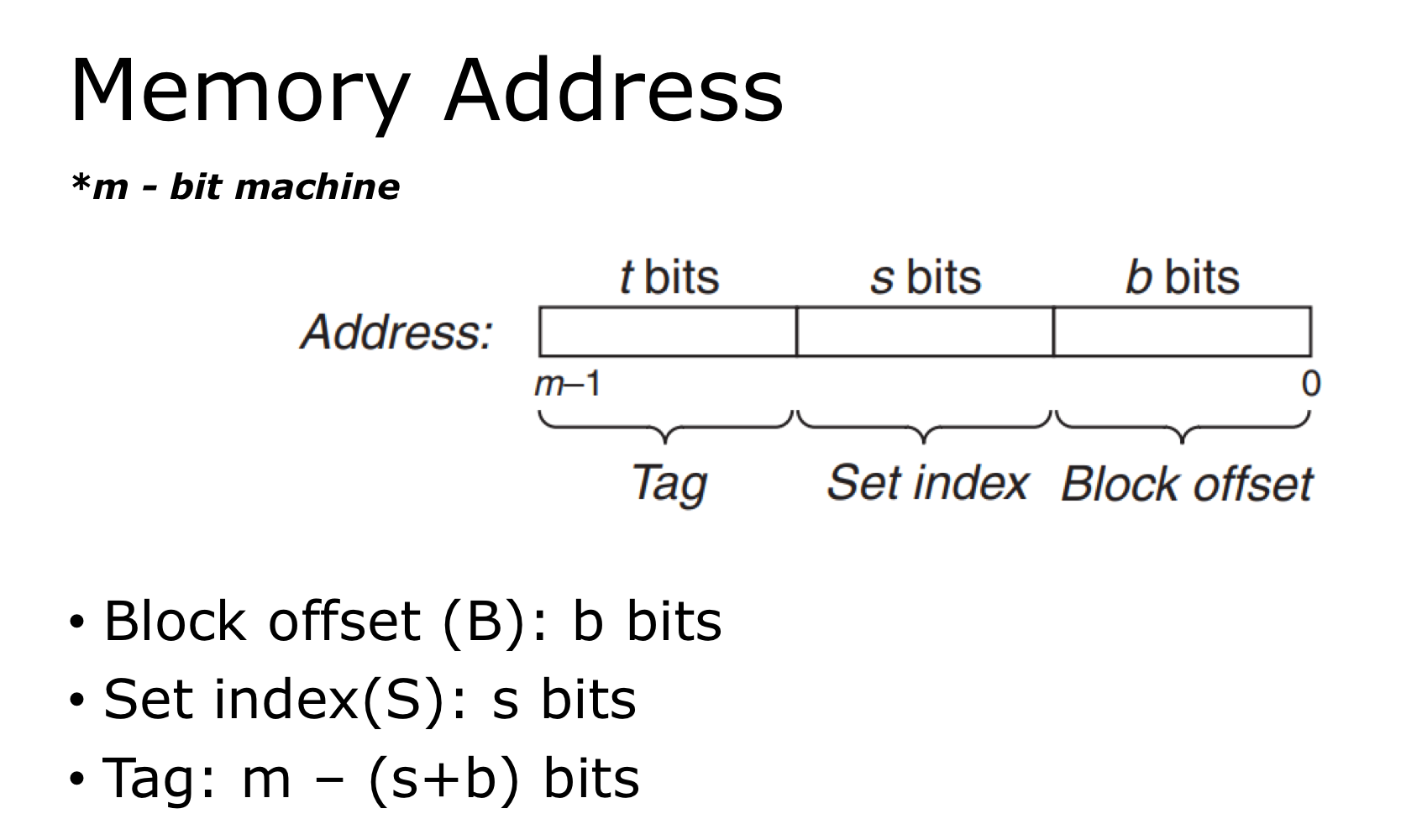
- 고속 기억 장치라고 불리는 메모리이며, 프로그램의 성능을 향상시키기 위해 데이터를 임시로 저장는 역할을 한다. 컴퓨터의 발전에 따라 CPU의 속도가 점점 빨라졌지만 메모리에는 한계가 있기에 캐시가 그 사이의 중재 역할을 하는 것이다.

- 캐시메모리는 locality의 원리를 만족하도록 data와 instruction을 저장한다.



- CPU가 요청한 데이터가 캐시에 있으면 ‘Cache Hit’ , 없어서 메모리에서 가져오면 ‘Cache miss’이며, 캐시에 데이터를 가져오려고 할 때 블록이 이미 차 있으면 ‘Cache eviction’이 발생한다.

- Memory address에서 적절한 bit 계산을 통해 cache의 어떤 블록에 데이터를 저장할 지, 접근할 지를 알 수 있다.



- 가장 아래 Bit에서 b bits 만큼이 block offset, 그 다음 s bit만큼이 set index로 쓰이며, 남은 나머지 bit들이 tag bit으로 쓰인다.

- Cache replacement policy 는 말 그대로 cache에서 블록에 있는 데이터를 교체할 때 어떻게 교체할 지에 대한 원리를 말한다. 이번 lab에서 쓰이는 cache replacement policy는 lru이며, 가장 오래된 line을 새 것으로 교체하는 원리이다.

- valgrind는 memory 에서 생기는 문제를 감지할 수 있는 진단 툴의 개념이다.

- getopt()는 command line에서 받아 오는 입력값에 대해 옵션을 주어 문자열을 자르는 함수라고 할 수 있다.

2) Cache optimization (Lab 9)

- 성능을 판단하기 위해 cache miss가 얼마나 일어나는 지 계산할 수 있다.

- matrix를 다루는 프로그램에서는 blocking이 miss 횟수를 줄일 수 있으므로, optimization 방법 중

<Cache Lab 풀이 내용>

**(1) Part A : Writing a Cache Simulator**

먼저 cache를 하나 만들어둔다.

Cache는 line이 있고, 그 line을 배열로 가지는 set을 배열로 가지는 2차원 배열 형태의 변수라고 할 수 있다.

그래서 아래와 같이 선언하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고, 캐시에서 miss, hit, eviction을 계산할 수 있는 전역 변수를 선언하고 초기화한다. 이는 나중에 miss, hit, eviction이 발생한 것을 감지하고 count해주며 증가해줄 변수들이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 나머지 cache 관련 정보를 담은 전역변수도 선언해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

이제 main함수를 만드는데, 먼저 command line 으로 들어오는 input 값에 대해 각 옵션을 주어 잘라서 각 h,v,s,E,b,t에 대한 정보를 받는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그리고 h 입력값이 들어오면 h\_flag 를 1로 해주고, h\_flag가 1일때는 관련 문구를 출력하도록 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

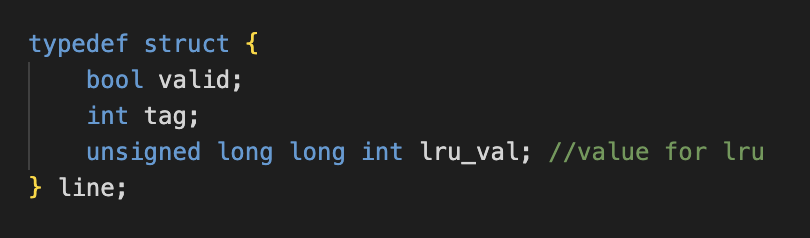
그다음에는 init\_cache()함수를 호출하여 cache값을 초기화해준다.

텍스트, 모니터, 화면, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Cache는 2차원 배열이기 때문에 2차원 배열로 동적할당을 해주고 set은 S 만큼, line는 E만큼 동적할당을 해준다.

여기서 line의 경우에는 나중에 miss, hit, eviction이 일어나는 것을 판단하고 lru를 통해 block을 replace하기 위해



다음과 같이 선언을 해주고, init\_cache() 함수에서 모든 line의 변수들에 대해 다 0으로 초기화를 해준다.

그다음, main에서는 trace\_input(trace)를 호출하여 trace 즉 file을 인자값으로 보내며 tracefile을 받아 처리하는 함수를 호출한다.

먼저, trace\_f로 받은 tracefile을 fopen 함수를 통해 열며 입력값들을 받아오며 While 반복문을 돌린다. V\_flag가 1인 경우에는 입력받은 값에 대한 정보를 같이 출력해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

While 반복문을 돌리며 trace\_file에서 가장 앞 부분은 t\_cmd라는 변수로 받고, 그 다음은 address, 마지막은 size 변수에 받는다.

텍스트이(가) 표시된 사진

자동 생성된 설명

I 명령어가 들어온 경우에는 무시하고, L, S의 경우에는 access\_cache(addr)을 한번 만 호출하며 M의 경우에는 2번 access를 해야 하므로 두 번 호출한다.

Access\_cache(addr) 함수의 경우 set, tag, lru 를 관리할 수 있는 변수들을 선언하고 address 로 부터 각 bit를 계산한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Set\_index bit은 address를 b만큼 right shift 한 후에 sbit 만큼만 추출하고, tag bit은 그 윗 부분을 모두 추출하는 방법이다.

먼저, hit을 체크한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Hit 의 경우, cache의 해당 set의 Line을 찾아가는 과정을 거치며 valid가 1인지 먼저 체크하고 1이면 tag\_bit과 tag가 일치하는지도 확인한다. 이후 일치하면 hit이 된것이므로 lru\_count의 값도 증가시키고 Lru\_val도 업데이트해준다. 그리고 hit\_num도 1만큼 증가시킨다.

그리고 v\_flag가 1인 경우에는 “hit”을 출력하고 종료한다.

Hit이 아닌 경우는 모두 miss 이므로, 먼저 miss\_num을 1만큼 증가시킨다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그다음으로는 eviction이 일어나는 지 체크해준다.

Eviction을 위해 lru 값들을 확인하여 가장 작은 lru값을 가진 Line을 찾는다.

텍스트이(가) 표시된 사진

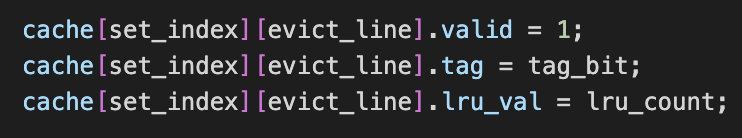
자동 생성된 설명

그 해당 line이 valid하다면 evict\_num을 1만큼 증가시켜주고 v\_flag가 1인 경우 miss\_eviction을 출력하고아닌 경우에는 miss를 출력하게 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Eviction 되었으므로 line의 정보를 다 업데이트 시켜준다.



Trace\_input 함수가 종료된 후에는 printSummary 함수를 호출하여 각 hit,miss, eviction 개수를 출력한다.

이후, 2차원 동적할당된 cache에 대해 모두 동적할당 해제를 해주며 프로그램이 종료된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

해당 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**(2) Part B : Optimizing Matrix Transpose**

1) 32\*32

Block size를 8로 하여 blocking을 통해 최적화를 하였다.

먼저, 32, 32까지 8씩 증가해가며 블록 단위로 반복문을 돌린다.

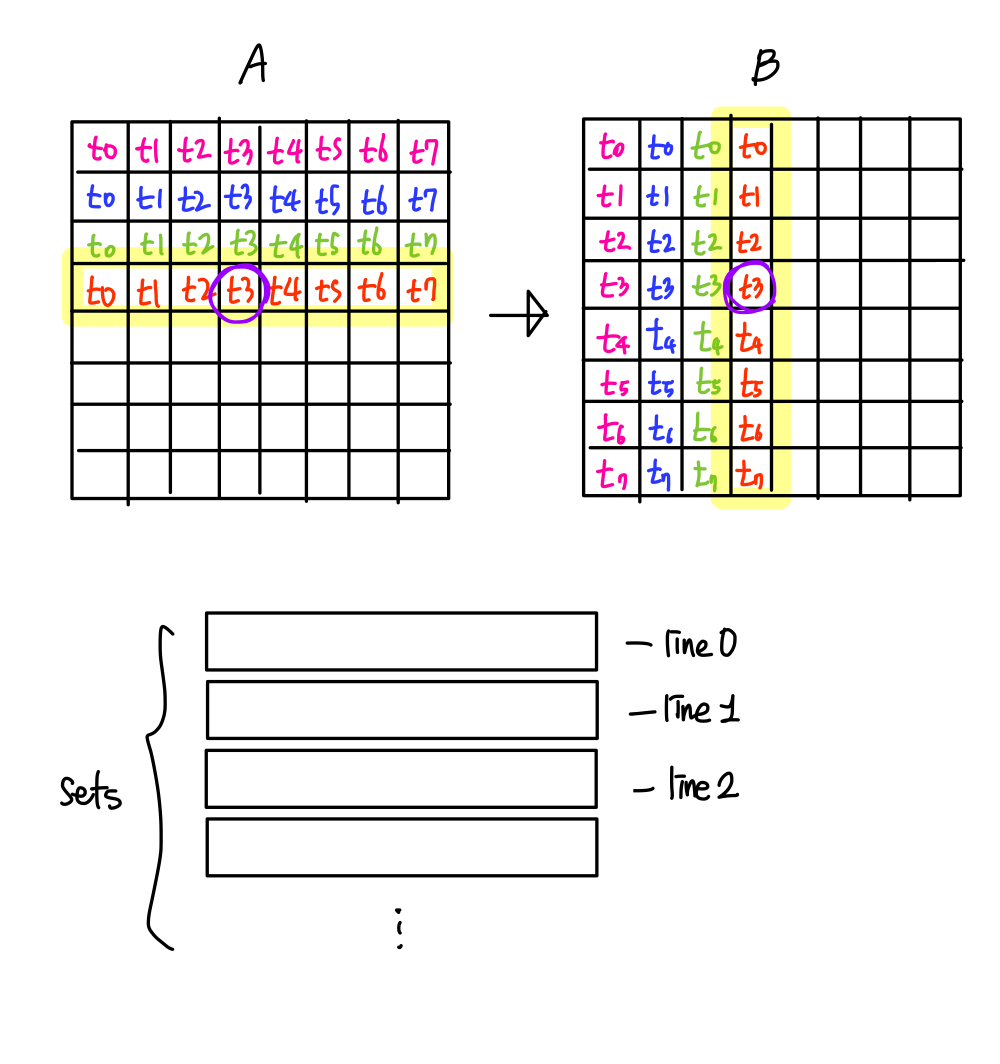
텍스트이(가) 표시된 사진

자동 생성된 설명

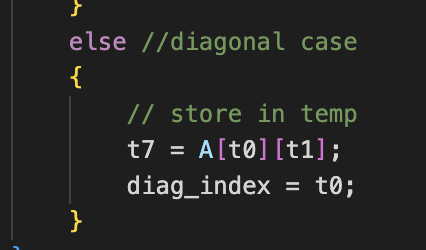
그 다음으로는 블록 안쪽 loop이다.

블록 안쪽 루프는 bl\_i, bl\_j 변수를 활용하여 loop를 돌린다.

이때, Miss 횟수를 줄이기 위해 같은 index를 가지는 즉, diagonal 원소들에 대해서는 temp 변수를 통해 하나의 matrix에만 접근한다. 즉, 동시에 두 matrix의 diagonal 원소들에 접근하는 것을 피한다.



여기서 diagonal 인 경우에는 A,B에서 같은 열을 사용하기 때문에 같은 Line에 들어가게 되며 eviction이 발생하면서 miss 횟수가 더 늘어난다. 그러므로,



Temp 변수에 A의 원소를 저장하고, diag\_flag와 diag\_index를 업데이트 해준다음

텍스트이(가) 표시된 사진

자동 생성된 설명

반복문이 끝나고 이를 감지하여 따로 B 변수에 넣어준다.

결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

2) 64\*64

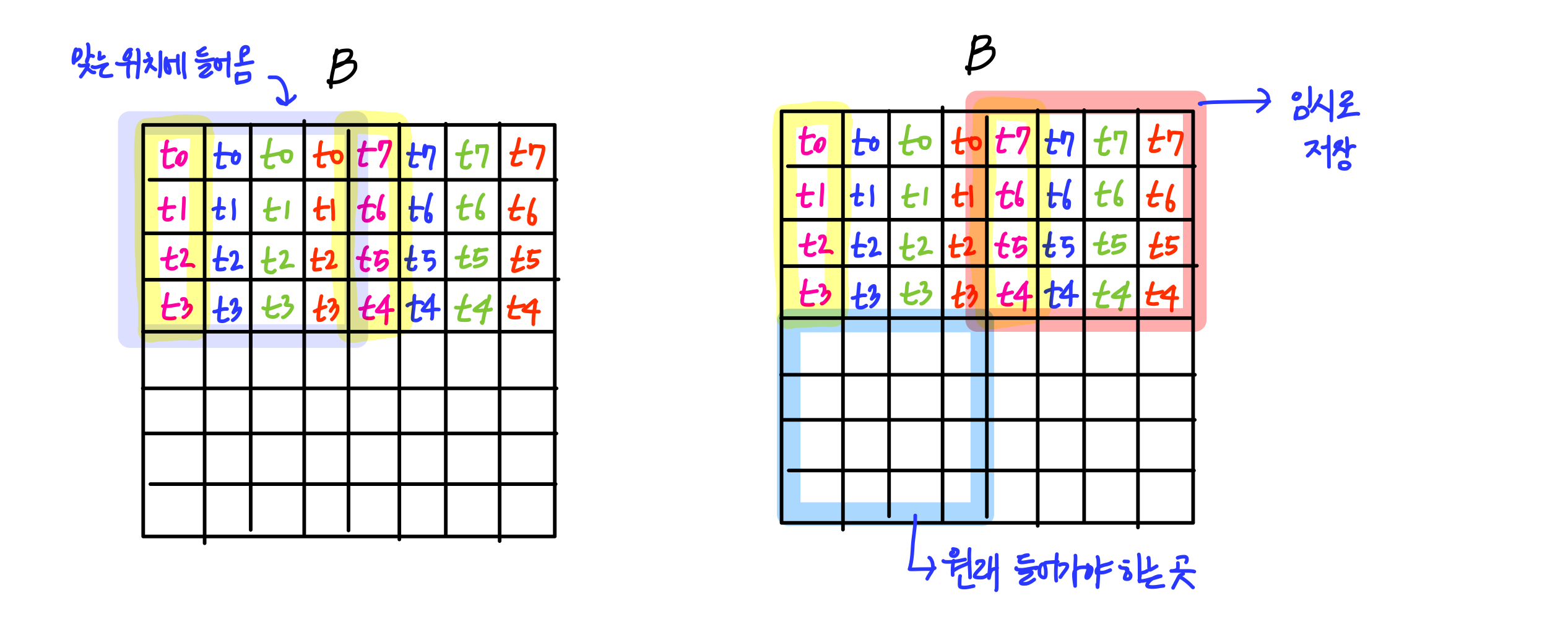
64\*64는 32\*32보다 set index를 공유하는 경우가 더 많아져서 캐시 최적화가 더 어려웠다. 그래서 일단 8size의 block으로 나눈 후에 위쪽 8\*4 와 아래쪽 8\*4로 나뉘어 따로 반복문을 돌려서 처리해주었다.

일단 윗부분 8\*4 부터 하게 되면

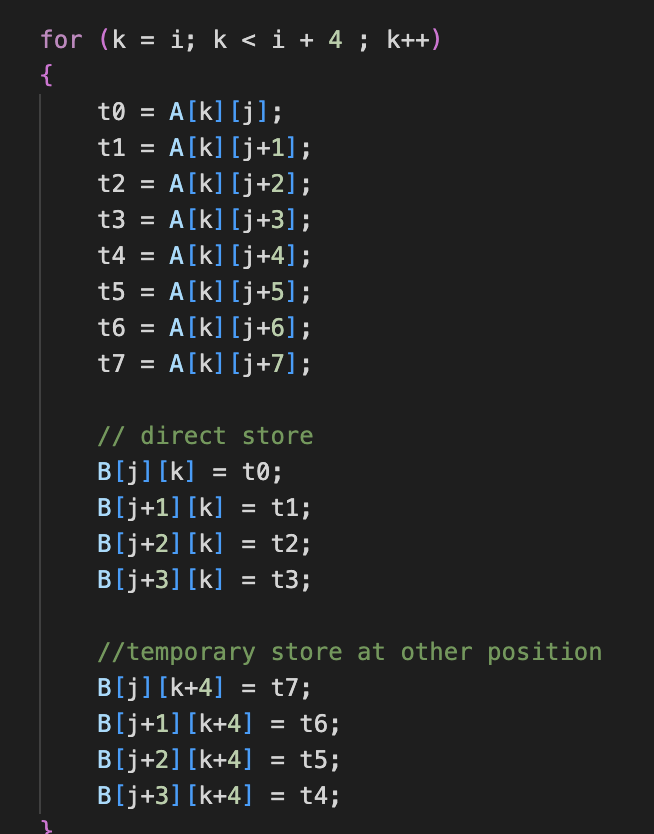
테이블이(가) 표시된 사진

자동 생성된 설명

이렇게 B 행렬에 넣을 때 왼쪽 4\*4부분만 알맞게 넣고 아래쪽 열에는 접근하지 않는다. 그리고 아래쪽 열이 아닌 위쪽 4열만 쓰기 위해 위쪽, 오른쪽 4\*4 부분에 임시적으로 저장한다.



코드는 아래와 같다.



그 다음 반복문에서 이제 B의 아래 4개의 열들에 접근할 것이다.

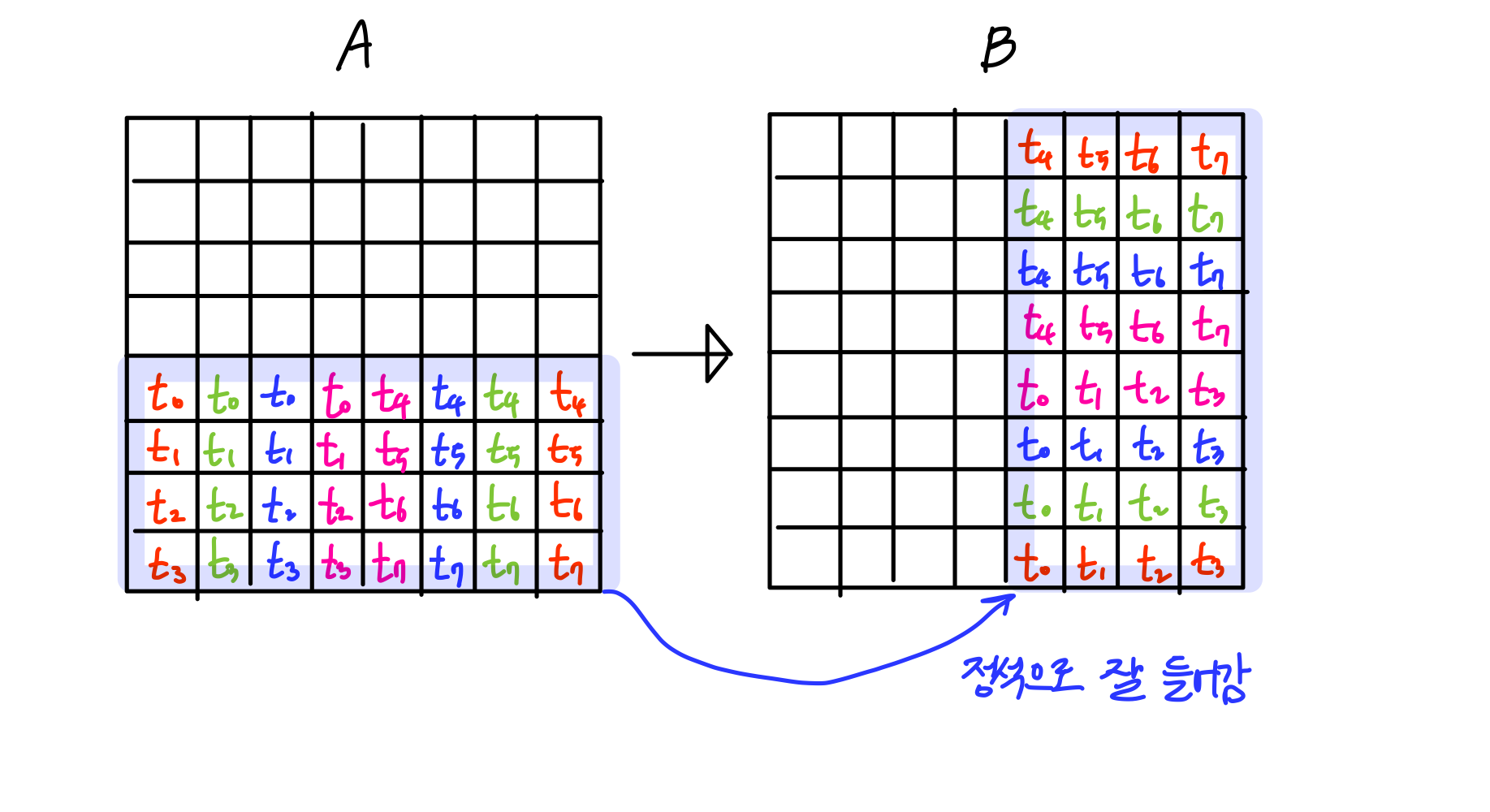
먼저, 임시저장해놨던 B의 위쪽 오른쪽 4\*4 부분에서 데이터들을 가져와

원래 저장해야 하는 곳인 왼쪽 아래쪽 4\*4 부분에 데이터를 이동시킨다.

텍스트, 낱말맞추기게임이(가) 표시된 사진

자동 생성된 설명

그 다음에는 A의 아래쪽 8\*4 데이터들을 B의 오른쪽 4\*8부분에 이동시킨다.



텍스트이(가) 표시된 사진

자동 생성된 설명

코드는 위와 같다.

그리고 해당 최적화의 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

3) 61\*67

1번이었던 32\*32 와 알고리즘은 동일하게 구현하였으며 블록 사이즈만 16으로 늘려서 최적화하였다. 결과는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

최종 결과:

텍스트이(가) 표시된 사진

자동 생성된 설명