<Pa3 – 보고서>

201720736 이균

1. How load_word is handled (explain hit and miss cases separately) & How you implement the least-recently-used replacement policy

캐시에서 메모리로 load하기 위해서는 우선 캐시와 메모리의 hit여부를 판단해야 한다. 따라서 check_cache_data_hit 함수를 이용해서 hit여부를 판단하였다. 이 함수에서 캐시 블록에 valid bit이 있는지 판단한 뒤에 없으면 miss라고 리턴을 하고 만약 valid bit이 있는데 tag까지 같다면 hit인 상황이고, 만약 valid bit이 있지만 tag가 다르다면 miss일 것이다. 이후 hit인 경우에는 timestamp를 최신 클럭사이클로 업데이트 해주어야 한다.
이후 load시에 캐시와 hit여부를 판단 하였다면 만약 miss인 경우에는 메모리에서 캐시로 load를 해야 한다. 따라서 access_memory 함수를 통해서 메모리에서 캐시로 로드 하도록 하였다. 이 함수에서 block_address 를 set개수로 나누면 몇번째 set에 들어갈 지가 나온다(cache_index). 또한 이때 find_entry_index_in_set 함수를 통해서 같은 set에서 어느 블록에 들어갈지 결정해준다.
이때 캐시 블록을 스캔하면서 valid bit이 0이어서 비어있는 곳이 있다면 그곳의 entry를 확정하고 만약 꽉 차있는 경우 LRU_timestamp라는 변수를 활용해서 캐시 블록을 스캔하면서 가장 오래된 timestamp의 index를 찾는다(min_index). 이후 가장 오래된 인덱스를 리턴 하면서 그곳의 timestamp를 최신화한다. 따라서 이후 메모리에서 로드할 때 find_entry_index_in_set로 부터 리턴된 가장 오래된 인덱스의 캐시 블록에 로드 하게 될 것이므로 다음과 같은 방식으로 LRU replacement policy를 구현하였다.
이후 set의 어느 entry_index에 들어갈 지 찾은 다음 dirty bit가 있다면 write back을 한 뒤에 메모리로 load를 해야 한다.
따라서 write back 하는 메모리의 주소는 fully associative cache일 때는 "memory + pEntry->tag * 캐시 한 블록의 크기"인 곳으로 write back 하고, n – way associative cache인 경우는 pEntry->tag * nr_ways * 블록 크기 만큼 떨어진 곳에서 block_offset을 더해주어서 들어가야 하는 메모리블록에 write back 해준 이후에 로드 하려는 주소의 메모리에서 다시 캐시로 로드 해야 한다.

2. How store_word is handled (explain hit and miss cases separately)

Store의 경우에는 역시 check_cache_data_hit 함수를 이용해서 hit여부를 판단하였다. 또한 find_entry_index_in_set을 통해서 역시 set의 어느 index에 들어갈 지 계산하였다. 이후 miss일 때는 캐시에 store해야 하는 것인데 strore할 때에도 역시 dirty bit가 있으면 write back을 먼저 한 뒤에 store역시 블록 단위로 진행되기 때문에 캐시 블록 크기만큼의 메모리 블록을 캐시로 가져온 뒤에 @data를 cache에 store한다. 또한 store시에 dirty bit를 true로 세팅하고, 추후에 있을 load에서 dirty bit를 보고 write back을 할 수 있게 한다. 또한 timestamp역시 최신 클럭으로 세팅

해야 한다. 만약 store시에 hit인 상황이라면 check_cache_data_hit에서 찾은 entry_index의 캐시블록에 이미 load가 되어있는 상황이다. 따라서 @data만 캐시 블록의 알맞은 word_offset에 저장하게 하였다.

3. Lesson learned

이번 과제에서 address를 어떻게 잘라서 무엇을 offset으로 볼 지, 무엇을 index - set index로 볼 지,무엇을 Tag로 볼 지 고민할 수 있었다. 또한 write back에 대한 개념과 fully associative cache, n-way associative cache, LRU의 개념에 대해서 다시 복습하며 공부할 수 있었다. 또한 cache에 load를 할 때에 miss인지 아닌지 보고, 또 miss인데 dirty bit가 있으면 write back을 해야 하고, 이때 또 꽉 차있으면 LRU까지 고려해야 한다는 것을 다시 복기할 수 있었으며 load는 블록단위로 이루어지기 때문에 byte address를 포함하는 block address를 load해야 한다는 것을 알게 되었다. store시에도 블록 단위로 store이 이루어지기 때문에 만약 캐시에 아무것도 없는데 store해야 한다면 먼저 블록 크기만큼 메모리에서 가져다 놓고 store해야 한다는것을 알았으며 역시 load와 마찬가지로 dirty bit를 고려 해서 write back을 해야 한다는 것을 알게 되었다. 과제를 하면서 address개념이 매우 헷갈렸는데 고민을 거듭하고 또 거듭한 끝에 byte address를 캐시 블록의 크기로 나누면 block address가 나오고, 또한 block address를 set의 개수의 modulo는 몇번째 set에 들어갈 지가 나온다. 또한 address에서 index의 앞부분 bit가 tag이므로 block address를 set개수로 나눈 몫이 tag가 된다. 또한 block address를 way개수의 modulo는 같은 set에서 몇 번째 block(block offset)으로 갈 지 나온다는 것을 이제는 이해하게 되었다. 처음 수업을 들을 때 이해가 가지 않던 개념이었는데 직접 과제를 하려고 하다 보니 실전으로 체득하게 된 것 같다. 과제를 하고, 보고서를 쓰면서 캐시에 대해 다시 생각해보면서 캐시의 종류가 direct mapped cache와 n-way associative cache로 나눠지고, 이에따라 set과 그 set의 어느 블록으로 갈 지 결정 해야 하며 load, store시에 dirty bit를 고려해서 write back을 하고, replacement policy로는 LRU방식으로 가장 오래전에 참조된 블록에 로드를 하는 방식을 사용한다는 것과 같은 큰 그림을 그릴 수 있게 된 것 같다.