이번 과제는 cache memory simulator를 구현하는 것이었습니다. 맨 처음 wodrs per block, number of blocks, number of ways를 순서대로 입력받으면, 이 숫자들에 따라 cache memory가 생성됩니다. 이후 lw, sw 명령어와 주소값을 입력하면, lw의 경우 cache memory에 memory data값을 load 하고, sw의 경우 알맞은 cache 주소 값에 data를 store합니다. 이를 제대로 수행할 수 있게 하는 lw function과, sw function을 구현하는 것이 핵심이었습니다. 두 function 모두 tag, index, address값이 중요하기 때문에 두 함수의 제일 위에서 bitwise 연산과 shift 연산을 사용해 index bit 수, offset bit 수, tag값, index 값, 주소값의 맨 처음 값,(faddr). offset값들을 각각 변수들에 저장해 두었습니다.

먼저 load_word function입니다. 매개변수로 unsigned int형 주소값을 받으면, cache memory 상태에 따라 올바른 작동을 하게 해야 합니다.
1. HIT 상태 - 현재 load 하려는 주소의 tag값과 cache memory의 tag값이 같고, cache가 vaild한 상태라면 'HIT'인 상태입니다. 이때는 간단하게 timestamp, vaild 값을 update해 주고, CACHE_HIT을 return 해줍니다.
2. MISS , cache block이 invalid한 상태 – cache block이 아무것도 없이 깨끗하다면, load를 진행하면 됩니다. 바로 입력받은 주소와 같은 memory 주소값에 접근하여, 그 data값을 cache memory에 load합니다. 이때는 입력 받은 주소값 외의 처음 주소값부터 존재하는 data값을 모두 가져와야 하므로 faddr값을 사용합니다. CACHE_MISS를 return해 줍니다.
3. MISS , 이미 cache block이 vaild한 상태 – 이때는 cache memory를 비우고 새로운 data를 load해야 하기 때문에, cache memory를 비울 기준이 필요합니다. 그 기준은 바로 timestamp값이 됩니다. LRU 개념을 이용하여, 가장 timestamp값이 낮은 block의 data를 새로운 data 값으로 load해 줍니다. 이때, 만약 그 block이 dirty하다면 현재 cache에 있는 data를 memory에 update하고, data를 load해 주어야 하고, 아니라면 바로 load를 해 주면 됩니다. 현재 cache에 있는 data를 memory에 update하기 위해서 원래 주소값을 알아야 하는데, 이를 위해 bitwise 연산과 shift 연산을 사용해 구해둔 originaddr값을 사용합니다. CASH_MISS를 return해 줍니다.

두 번째로 store word_function입니다. 매개변수 unsigned int형 주소값과 unsigned int data값을 받으면, cache memory에 따라 올바른 작동을 하게 해야 합니다.
1. HIT 상태 - 현재 store 하려는 주소의 tag값과 cache memory의 tag값이 같고, cache가 vaild한 상태라면 'HIT'인 상태입니다. 이때는 받은 주소값 자리에 넣고자 하는 data를 저장해줍니다. 이때 정확한 주소값을 위해 bitwise연산을 통해 구한 offset값을 이용합니다. 동시에 timestamp, vaild, dirty 값을 update해 줍니다. 또한 CACHE_HIT을 return 해줍니다.
2. MISS , cache block이 invalid한 상태 – cache block이 아무것도 없이 깨끗하다

면, 먼저 입력받은 주소값이 포함된 첫 주소값부터 모든 data를 cache에 저장한 뒤, 저장하고자 하는 data값을 올바른 주소값에 저장해줍니다. 이때도 offset값을 이용합니다. 동시에 timestamp, vaild, dirty 값을 update해 줍니다. 또한 CACHE_MISS을 return 해줍니다.
3. MISS , 이미 cache block이 vaild한 상태 – load word와 비슷합니다. cache memory를 비우고 새로운 data를 store해야 하기 때문에, cache memory를 비울 기준이 필요합니다. 그 기준은 바로 timestamp값이 됩니다. 이때, 만약 그 block이 dirty하다면 현재 cache에 있는 data를 memory에 update하고, data를 load해 저장합니다. 아니라면 바로 store를 해 주면 됩니다. 현재 cache에 있는 data를 memory에 update하기 위해서 원래 주소값을 알아야 하는데, 이를 위해 bitwise 연산과 shift 연산을 사용해 구해둔 originaddr값을 사용합니다. 동시에 timestamp, vaild, dirty 값을 update해 줍니다. 또한 CACHE_MISS을 return 해줍니다. 또한 CASH_MISS를 return해 줍니다.

이번 과제는 저에게는 저번 과제보다 어려웠던 것 같습니다. 먼저 cache의 wirte_back개념을 구현하는 것이 생각보다 어려웠고, 무엇보다 각 상황에 따라 다른 작동을 수행하게 하는 것이 어려웠습니다. 그러나 cache구조를 직접 구현해보면서 memory hierarchy를 이전보다 더 이해할 수 있었던 것 같습니다.