Planning :

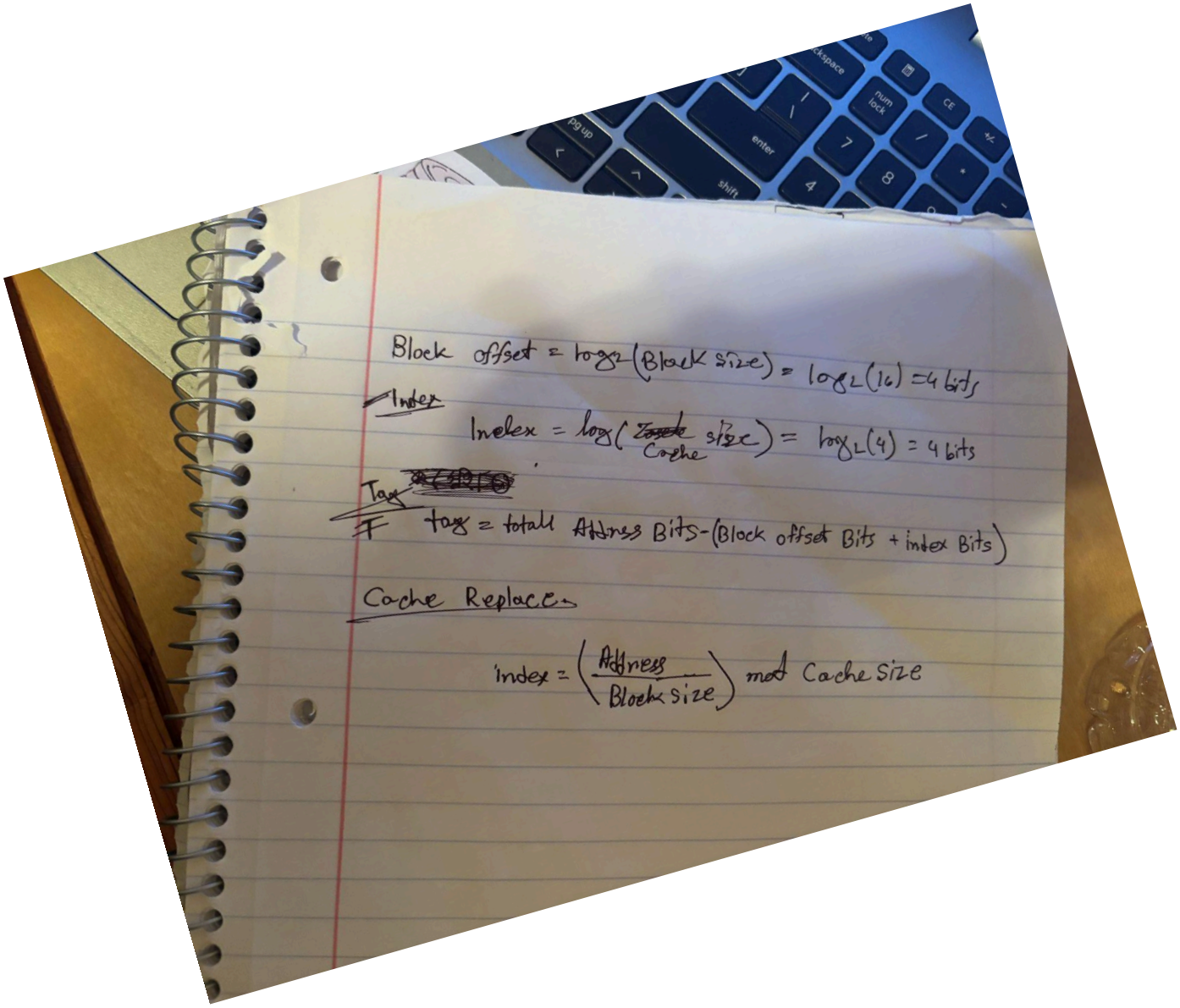
Pseudocode :

```
Main Function:
    Prompt user for the number of memory accesses
    Allocate memory for storing addresses
    If memory allocation fails, print error and exit
    Prompt user for memory addresses in hexadecimal format
    Call direct_mapped_cache with user-provided addresses
    Call fully_associative_fifo with user-provided addresses
    Call fully_associative_lru with user-provided addresses
    Free allocated memory

Direct Mapped Cache Function:
    Initialize cache array with CACHE_SIZE blocks
    Initialize hit, compulsory_misses, and conflict_misses counters
    For each memory access:
        Calculate the index and tag for the current address
        If the cache block at the index is valid and the tag matches:
            Increment hits
        Else:
            If the cache block is invalid, increment compulsory_misses
            Else, increment conflict_misses
            Update cache block with the new tag and set it to valid
    Print hits, compulsory_misses, and conflict_misses

Fully Associative FIFO Function:
    Initialize cache array with CACHE_SIZE blocks
    Initialize hit, compulsory_misses, and capacity_misses counters
    Initialize a FIFO counter
    For each memory access:
        Calculate the tag for the current address
        Check if the address already exists in the cache:
            If found, increment hits and update FIFO order
        Else:
            For each cache block:
                If an invalid block is found, insert the address and update FIFO order
                Increment compulsory_misses
            If all blocks are full:
                Find the oldest cache block (based on FIFO order)
                Replace it with the new address and increment capacity_misses
    Print hits, compulsory_misses, and capacity_misses
```

Fully Associative LRU Function:

```
Initialize cache array with CACHE_SIZE blocks
Initialize hit, compulsory_misses, and capacity_misses counters
Initialize global LRU counter
For each memory access:
    Calculate the tag for the current address
    Check if the address already exists in the cache:
        If found, increment hits and update LRU counter
    Else:
        For each cache block:
            If an invalid block is found, insert the address and update LRU counter
            Increment compulsory_misses
        If all blocks are full:
            Find the least recently used cache block (based on LRU counter)
            Replace it with the new address and increment capacity_misses
Print hits, compulsory_misses, and capacity_misses
```

Math I have done in note book :



Follow Up Questions:

1) More than a day. No partner
2) Different cache replacement policies . Now I have idea how cache systems manage memory access.
3) I think debugging the cache replacement policies
4) I researched the principles of cache memory and learn the three main cache replacement policies. I then make note of the necessary components, including memory addresses, cache structure, and the replacement logic. I start by implementing the Direct-Mapped Cache, as it

was the simplest of the policies. After that, I start doing FIFO and LRU.

5) Yes I do. I was confused how to calculate cache hits and misses. So use AI. Also I saw some tutorials on youtube. I found some solution on stackoverflow that also help me to understand the concept.