Task1: Revise the cache concepts learnt before the midterm.

Task2: Derive the address matrix from the trace file shared along with the project.

Task3: Create Cache memory template for Direct Mapped, Fully Associative cache, Set associative cache types.

Task4: Start working on the direct mapped cache and run the design over different cache and block size combinations.

Task5: Get the simulation results for block size v hit ratio, cache size v hit ratio for the direct mapped cache design

Task4: Start working on the Fully Associative cache and run the design over different cache and block size combinations.

Task5: Get the simulation results for block size v hit ratio, cache size v hit ratio for the fully Associative cache design

Worked on the design of how to implement the Cache Simulator. Assuming the Processor addresses are 32 bits for this entire project. Tasks to be done: Maximum Cache size=64KB =2^16 Bytes= Maximum cache bits=16bits Maximum Block size =256 bytes = Maximum Block bits=9bits Set Associativity 1(Direct mapped),2,4,8,16,32 and 64 (fully associativity) Case 1: Fixed Block size=32B; Fixed Associativity; Cache sizes= 1KB, 2KB, 4KB, 8KB, 16 KB, 32KB, and 64KB of data. Case 2: Fixed cache size 8192; Fixed associativity; Block sizes=1, 2, 4, 8, 16, 32, and 64. Case 3: Fixed Block size=32B; Fixed cache size 8192; Set Associativity = 1, 2, 4, 8, 16, 32, and the fully Associative (64). For implementing the cache simulator, the following two modules have to design. Cache Design: A cache simulator takes following three parameters as an input: A: Associativity (number of ways per set) B: Block size (size of a single block, a block is also referred to as a cache line) C: Cache size (total amount of data in the cache) These three parameters are used to determine tag, index, and offset bits in a cache memory. Num\_offset\_bit = log2 (Block size) Will Implement in task get\_cache\_block\_addr() Num\_index\_bit = log2 (Cache size/ (Block size\* Associativity)) Will Implemented in task get\_cache\_line() Num\_tag\_bit = 32 - Num\_index\_bit - Num\_offset\_bit Will Implemented in task get\_cache\_tag() The cache memory is designed by extracting the bits by implementing the above functions. Here there is no need to store data values in the cache because we are just calculating hit/miss for each memory reference indicated by the trace file.

**Cache Access:**

Here the cache can be represented by a 2-dimensional array of tags. That is Number of rows is “Num\_index\_bit” and Number of columns is “Associativity” (The second dimension only becomes relevant when the associativity of the cache is higher than 1). While accessing the cache we have to check the following two conditions to determine whether the cache Hit or Miss occurs. Case 1: if (tag\_memory !=full) then //Check for memory location is present in cache or not If(Cache == trace\_memory[31: 31-Num\_tag\_bit+1]) Cache\_hit++; else Cache\_miss++; and then check for a free block and load the trace\_memory into the cache. Case 2: if (tag\_memory ==full) then Cache\_miss++; and implement the LRU algorithm for tag\_memory replacement. The Number of LRU Bits for each set is log2(Associativity!) Once Cache design and Cache access is implemented then will check for the different test cases mentioned in the Mini Project 2 document.