Detailed Report on Cache Hit/Miss Simulator

This C program is intended to simulate the functionality of a cache memory system based on information retrieved from input files (cache.access and cache.config). It employs functions like hextoBinary, sliceString, tointeger, twosComplementToDecimal, and binaryToHex to perform cache simulation with FIFO, LRU, or random replacement policies. This report aims to provide an overview and description of each function present within the code.

Code Structure:

Here is an analysis of the coding approach and steps taken for the code to work correctly:

Header Files and Constants :
The program begins with standard C library inclusion and the definition of constants, specifying maximum array dimensions and sizes. These constants help maintain clarity and manage array sizes efficiently.

sliceString(char *str, int start, int end) :
The 'sliceString' function extracts substrings from character arrays. This function is used to extract specific portions of assembly instructions and returns a substring of the original string between these indices. In the function the start index is inclusive but the end index is exclusive.

hextoBinary (char arr[][40], int num, char add[][15]) :
For each line of hexadecimal address, the function converts it into its binary representation. This process involves mapping each hexadecimal digit to its binary equivalent. It is acheived using switch case method.

tointeger(char arr[]) :
The `tointeger` function is responsible for converting character arrays representing decimal numbers into int data types. It iterates through the characters in the array,
And based on their ascii values and converts them to integers.

binaryToHex(char arr[]) :
The 'binaryToHex' function facilitates the conversion of binary strings to their hexadecimal representation. By processing groups of four bits in the binary string, it generates the corresponding hexadecimal digits. This conversion is essential for presenting binary memory addresses or data in a more human-readable format for analysis or output purposes.

twosComplementToDecimal(char arr2[], int yes) :
The `twosComplementToDecimal` function does exactly what the name suggest, converts character arrays representing binary numbers in two's complement format into decimal integers. The input array of characters have the bit values which are then converted to decimal numbers. When the value of "yes" parameter is 1 the input is considered as 2's complement otherwise the input is considered as a binary representations.

Main Function:

The main function serves as the entry point and central coordinator of the cache simulation code. Its primary role encompasses orchestrating the entire cache simulation process, including file handling, data processing, and cache operation simulations based on the provided configurations.

1. File Handling: At the onset, the main function opens and reads data from two input files, namely cache.access and cache.config, using file pointers (file and file2, respectively). These files contain crucial information necessary for cache simulation, such as memory access patterns (cache.access) and configuration settings (cache.config).

2. Reading Access Data: The function interacts with the readAccessData subroutine to parse and retrieve access-related information from the cache.access file. This includes reading memory addresses and access types (read or write) into arrays (access_address and access_type, respectively), essential for cache simulation operations.

3. Reading Configuration Data: Similarly, the main function interacts with the readConfigData subroutine to extract and store configuration settings from the cache.config file. This involves retrieving parameters such as total cache size, block size, associativity, replacement policy, and write type into arrays (config_data), which influence cache behavior.

4. Processing Cache: Once the necessary data is obtained, the main function proceeds to execute the processCache subroutine. This section performs the actual cache simulation based on the acquired access and configuration data. It involves interpreting memory addresses, implementing cache replacement policies (FIFO, LRU, or random), handling cache hits and misses, and outputting simulation results.

5. Closing Files and Program Termination: Finally, after completing the cache simulation process, the main function appropriately closes the file pointers (file and file2) to release system resources. The function then terminates, concluding the execution of the cache simulation code.

In conclusion, analysing this code offers a deeper understanding of cache operations, aiding in optimizing memory access and system performance in practical computing environments. Mastery of these functions allows for efficient cache simulation and contributes to the broader knowledge of memory management in computer systems.