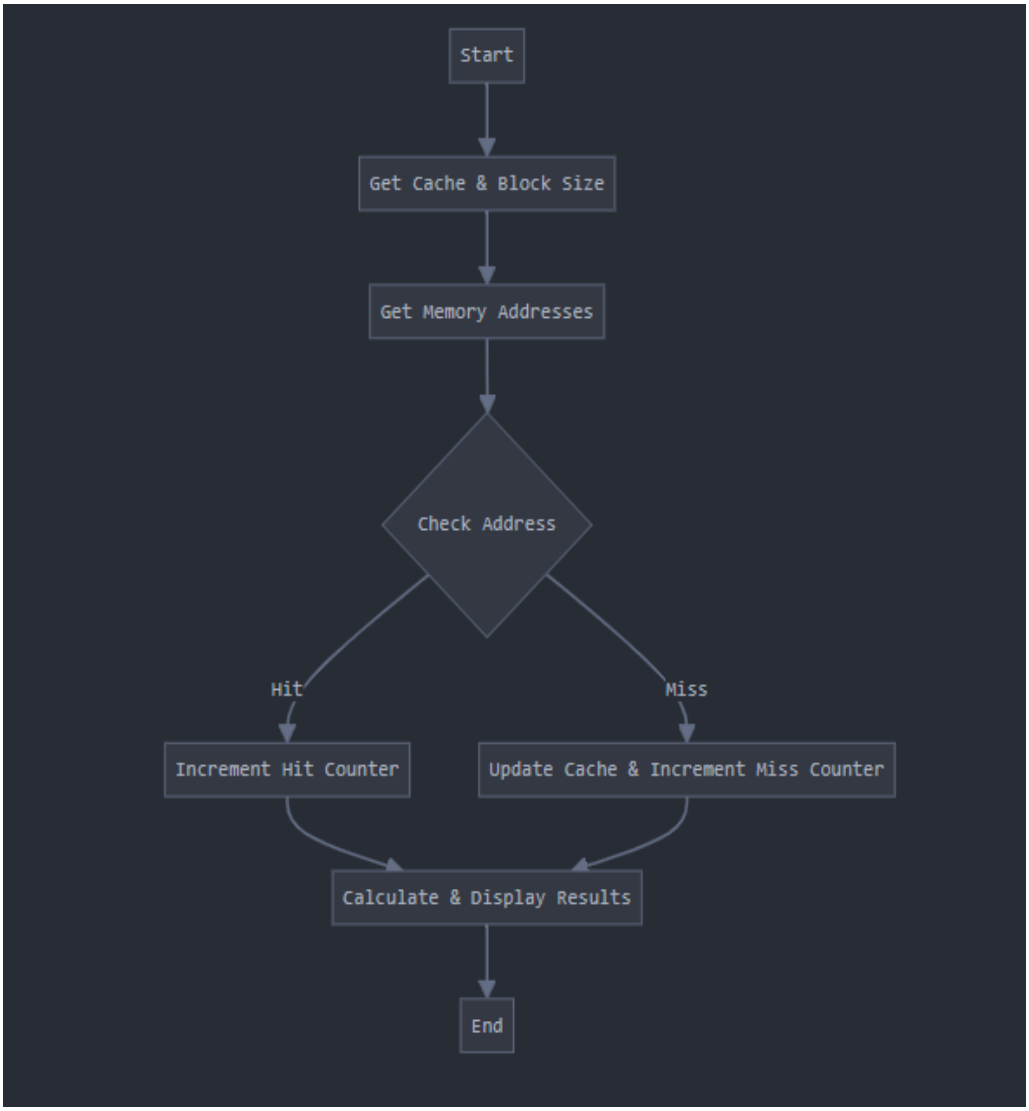
BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	**Sujal Sandeep Dingankar**
UID no.	**2024301005**
Experiment No.	**10**

AIM:	**Implementation of Mapping techniques (Direct Mapping) of Cache memory by using any programming language.**
Program 1	
LANGUAGE:	C
THEORY:	**Mapping** in Cache Memory refers to the technique used to determine where a memory block from main memory will be placed in the cache. It defines the relationship between main memory addresses and cache lines. Effective mapping helps improve cache performance by reducing access time. There are mainly 3 types of mapping : Direct Mapping , Associative Mapping, Set Associative Mapping. **Direct Mapping**: Direct mapping is a simple and widely used cache mapping technique in computer architecture. In this approach, each block of main memory maps to exactly one specific line in the cache. Direct Mapping is very easy to implement ad requires less hardware cost . However, Direct mapping does not handle situations well where multiple active memory blocks map to the same cache line. **Hit Ratio:** The hit ratio is the percentage of memory access requests that result in a "hit," meaning the data being accessed is found in the cache. **Miss Ratio:** The miss ratio is the percentage of memory access requests that result in a "miss," meaning the data being accessed is not found in the cache, and the system has to fetch it from the main memory.
ALGORITHM:	Algorithm for implementing direct mapped cache memory system is: **1. Taking input from the user:** • Read the Cache_size (total cache size) and blockSize (size of each block).

	• Calculate the number of cache lines as cache_lines = Cache_size / BlockSize. **2. Initialize Cache:** • Create two vectors: o cache to store the addresses for each cache line, initialized with -1. o valid to indicate if a cache line is valid, initialized with 0. **3. Input Memory Accesses:** • Read the total number of memory addresses (num_accesses). • Store the memory addresses in a vector memory_addresses. **4. Simulate Cache Access:** • For each memory address, Calculate the cache line index as line = address % cache_lines. • Check if the current address is already present in the cache line: 1. **Hit:** If valid[line] == 1 and cache[line] == address, increment the hits counter. 2. **Miss:** Otherwise, increment the misses counter, update the cache at line with the address, and set valid[line] = 1. **5. Compute Hit and Miss Ratios:** • Calculate hit_ratio = hits / num_accesses. • Calculate miss_ratio = misses / num_accesses. **6. Output the result:** • Print the total number of accesses, cache hits, cache misses, hit ratio, and miss ratio with two decimal places.
FLOWCHART:	**Flowchart for Direct Mapping cache technique is:**



PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int Cache_size;
    int BlockSize;

    // Input cache size and block
size
    printf("Enter the cache size:
");
    scanf("%d", &Cache_size);

    printf("Enter the block size:
```

```
");
    scanf("%d", &BlockSize);

    int cache_lines = Cache_size
/ BlockSize;
    int *cache = (int
*)malloc(cache_lines *
sizeof(int));
    int *valid = (int
*)malloc(cache_lines *
sizeof(int));

    // Initialize cache and valid
arrays
    for (int i = 0; i <
cache_lines; i++) {
        cache[i] = -1;
        valid[i] = 0;
    }

    int hits = 0, misses = 0;
    int num_accesses;

    // Input number of memory
accesses
    printf("Enter the number of
memory addresses: ");
    scanf("%d",
&num_accesses);

    int *memory_addresses =
(int *)malloc(num_accesses *
sizeof(int));
    printf("Enter the memory
addresses:\n");
    for (int i = 0; i <
num_accesses; i++) {
        scanf("%d",
&memory_addresses[i]);
    }

    // Cache simulation
    for (int i = 0; i <
num_accesses; i++) {
```

```
        int address =
memory_addresses[i];
        int line = address %
cache_lines;

        if (valid[line] &&
cache[line] == address) {
            hits++;
        } else {
            misses++;
            cache[line] = address;
            valid[line] = 1;
        }
    }

    // Output results
    double hit_ratio =
(double)hits / num_accesses;
    double miss_ratio =
(double)misses /
num_accesses;

    printf("Total accesses:
%d\n", num_accesses);
    printf("Cache hits: %d\n",
hits);
    printf("Cache misses: %d\n",
misses);
    printf("Hit ratio: %.2f\n",
hit_ratio);
    printf("Miss ratio: %.2f\n",
miss_ratio);

    // Free allocated memory
    free(cache);
    free(valid);
    free(memory_addresses);

    return 0;
}
```

RESULT:

```
Enter the cache size: 16
Enter the block size: 4
Enter the number of memory addresses: 15
Enter the memory addresses:
1 2 3 1 4 6 7 4 5 5 8 7 9 6 7
Total accesses: 15
Cache hits: 6
Cache misses: 9
Hit ratio: 0.40
Miss ratio: 0.60
```

CONCLUSION:	In this experiment, I was able to understand the theory of direct mapping in cache memory and was successfully able to perform the experiment showing cache hits and hit ratio.