RANDOM ACCESS MEMORY

#include <stdio.h>

#define SIZE 10 // Size of the array

int main() {

int arr[SIZE]; // Array to store values

int i, index, value, choice;

// Initialize array with zeros

for (i = 0; i < SIZE; i++) {

arr[i] = 0;

}

// Menu for user interaction

do {

printf("\nRandom Access Memory (RAM) Operations\n");

printf("1. Write to RAM\n");

printf("2. Read from RAM\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter index (0 to %d): ", SIZE - 1);

scanf("%d", &index);

if (index >= 0 && index < SIZE) {

printf("Enter value to write: ");

scanf("%d", &value);

arr[index] = value;

printf("Value written to RAM.\n");

} else {

printf("Invalid index. Please enter a valid index.\n");

}

break;

case 2:

printf("Enter index (0 to %d): ", SIZE - 1);

scanf("%d", &index);

if (index >= 0 && index < SIZE) {

printf("Value at index %d: %d\n", index, arr[index]);

} else {

printf("Invalid index. Please enter a valid index.\n");

}

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Please enter a valid choice.\n");

}

} while (choice != 3);

return 0;

}

OUTPUT:

Random Access Memory (RAM) Operations

1. Write to RAM

2. Read from RAM

3. Exit

Enter your choice: 1

Enter index (0 to 9): 5

Enter value to write: 2

Value written to RAM.

Random Access Memory (RAM) Operations

1. Write to RAM

2. Read from RAM

3. Exit

Enter your choice: 2

Enter index (0 to 9): 5

Value at index 5: 2

Random Access Memory (RAM) Operations

1. Write to RAM

2. Read from RAM

3. Exit

Enter your choice: 3

Exiting...

READ ONLY MEMORY

#include <stdio.h>

// Define the size of the ROM

#define ROM\_SIZE 10

// Define the ROM data (example data)

int rom[ROM\_SIZE] = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

int main() {

int address;

printf("Enter the address (0 to %d) to read from the ROM: ", ROM\_SIZE - 1);

scanf("%d", &address);

// Check if the entered address is valid

if (address >= 0 && address < ROM\_SIZE) {

printf("Data at address %d: %d\n", address, rom[address]);

} else {

printf("Invalid address! Please enter a valid address between 0 and %d\n", ROM\_SIZE - 1);

}

return 0;

}

OUTPUT

Enter the address (0 to 9) to read from the ROM: 6

Data at address 6: 13

VIRTUAL MEMORY

#include <stdio.h>

#include <stdlib.h>

#define MEMORY\_SIZE 100

int virtual\_memory[MEMORY\_SIZE];

void write\_to\_memory(int address, int value) {

if (address >= 0 && address < MEMORY\_SIZE) {

virtual\_memory[address] = value;

printf("Value %d written to address %d\n", value, address);

} else {

printf("Error: Address out of bounds!\n");

}

}

int read\_from\_memory(int address) {

if (address >= 0 && address < MEMORY\_SIZE) {

printf("Value at address %d: %d\n", address, virtual\_memory[address]);

return virtual\_memory[address];

} else {

printf("Error: Address out of bounds!\n");

return -1; // Return an error value

}

}

int main() {

int choice, address, value;

printf("Welcome to Virtual Memory System\n");

do {

printf("\n1. Read from memory\n");

printf("2. Write to memory\n");

printf("3. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter address to read from: ");

scanf("%d", &address);

read\_from\_memory(address);

break;

case 2:

printf("Enter address to write to: ");

scanf("%d", &address);

printf("Enter value to write: ");

scanf("%d", &value);

write\_to\_memory(address, value);

break;

case 3:

printf("Exiting...\n");

break;

default:

printf("Invalid choice! Please try again.\n");

}

} while (choice != 3);

return 0;

}

OUTPUT

Welcome to Virtual Memory System

1. Read from memory

2. Write to memory

3. Exit

Enter your choice: 1

Enter address to read from: 50

Value at address 50: 0

1. Read from memory

2. Write to memory

3. Exit

Enter your choice: 1

Enter address to read from: 52

Value at address 52: 0

1. Read from memory

2. Write to memory

3. Exit

Enter your choice: 2

Enter address to write to: 52

Enter value to write: 5

Value 5 written to address 52

1. Read from memory

2. Write to memory

3. Exit

Enter your choice: 3

Exiting...

MEMORY ALLOCATION

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*array; // Pointer to store dynamically allocated memory

int size, i;

// Asking user for the size of the array

printf("Enter the size of the array: ");

scanf("%d", &size);

// Dynamically allocating memory for the array

array = (int \*)malloc(size \* sizeof(int));

// Checking if memory allocation was successful

if (array == NULL) {

printf("Memory allocation failed!\n");

return 1; // Exiting the program with error code

}

// Getting input for the array elements from the user

printf("Enter %d elements:\n", size);

for (i = 0; i < size; i++) {

scanf("%d", &array[i]);

}

// Displaying the array elements

printf("The elements in the array are: ");

for (i = 0; i < size; i++) {

printf("%d ", array[i]);

}

printf("\n");

// Freeing dynamically allocated memory

free(array);

return 0; // Exiting the program successfully

}

OUTPUT

Enter the size of the array: 5

Enter 5 elements:

2 3 4 5 6

The elements in the array are: 2 3 4 5 6

CACHE MEMORY

#include <stdio.h>

#include <stdlib.h>

#define CACHE\_SIZE 256 // Define cache size in bytes

#define MAIN\_MEMORY\_SIZE 1024 // Define main memory size in bytes

// Cache structure

typedef struct {

int valid; // Valid bit to indicate whether the cache block contains valid data

int tag; // Tag bits for identifying the memory block

int data; // Data stored in the cache block

} CacheBlock;

// Function to simulate cache access

void accessCache(int address, CacheBlock \*cache) {

int cacheIndex = address % CACHE\_SIZE; // Calculate cache index

int tag = address / CACHE\_SIZE; // Calculate tag bits

// Check if cache block is valid and contains the required data

if (cache[cacheIndex].valid && cache[cacheIndex].tag == tag) {

printf("Cache hit! Data found at address %d\n", address);

} else {

// Cache miss scenario

printf("Cache miss! Data not found at address %d\n", address);

// Load data from main memory to cache

cache[cacheIndex].valid = 1;

cache[cacheIndex].tag = tag;

cache[cacheIndex].data = address; // For simplicity, storing the address as data

}

}

int main() {

CacheBlock cache[CACHE\_SIZE]; // Cache memory

int memory[MAIN\_MEMORY\_SIZE]; // Main memory

// Initialize cache and main memory

for (int i = 0; i < CACHE\_SIZE; i++) {

cache[i].valid = 0;

cache[i].tag = 0;

cache[i].data = 0;

}

for (int i = 0; i < MAIN\_MEMORY\_SIZE; i++) {

memory[i] = i; // Initialize main memory with sequential data

}

int address;

// Taking input for memory address

printf("Enter the memory address to access (0 - %d): ", MAIN\_MEMORY\_SIZE - 1);

scanf("%d", &address);

// Accessing cache with the given memory address

accessCache(address, cache);

return 0; // Exiting the program successfully

}

OUTPUT

Enter the memory address to access (0 - 1023): 1009

Cache miss! Data not found at address 1009