

# Hierarchical Memory & Cache Simulator

Architectural Design Document

*System Implementation Report*

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>System Architecture &amp; Assumptions</b>	<b>1</b>
2.1	Memory Layout . . . . .	1
2.2	Operational Assumptions . . . . .	1
<b>3</b>	<b>Component Implementation</b>	<b>1</b>
3.1	Physical Memory Allocator . . . . .	1
3.1.1	Free List Allocator . . . . .	1
3.1.2	Buddy System Allocator . . . . .	2
3.2	Virtual Memory Unit (MMU) . . . . .	2
3.3	Cache Hierarchy . . . . .	2
<b>4</b>	<b>Integration &amp; Data Flow</b>	<b>2</b>
<b>5</b>	<b>Limitations &amp; Constraints</b>	<b>3</b>

# 1 Introduction

This document delineates the architectural specifications and design rationale for a systems-level memory simulator. The project models the critical data path of a computer architecture, encompassing the Central Processing Unit (CPU), Memory Management Unit (MMU), a three-level Cache Hierarchy, and Physical Main Memory.

The primary objective is to facilitate runtime analysis of memory management strategies, providing granular metrics on internal/external fragmentation, page fault latency, and cache hit/miss propagation.

## 2 System Architecture & Assumptions

### 2.1 Memory Layout

The simulation operates within a deterministic environment defined by the following constraints:

- **Address Space:** A 32-bit addressing scheme is employed, utilizing `uint32_t` for all virtual and physical addresses.
- **Physical Memory (RAM):** Modeled as a contiguous byte array. The default configuration allocates 4096 Bytes, modifiable at runtime.
- **Page Size:** Fixed at 64 Bytes, establishing the granularity for frames and page table entries.
- **Granularity:** Memory access occurs at the byte level, though the Buddy System allocator enforces power-of-2 alignment.

### 2.2 Operational Assumptions

To maintain simulation feasibility, the design incorporates the following abstractions:

- **Single Execution Context:** The system models a distinct, single-threaded process. Context switching and thread concurrency are outside the current scope.
- **Symbolic I/O:** Disk storage is abstracted. Page faults trigger a symbolic latency penalty (e.g., 10,000 cycles) rather than performing physical I/O operations.
- **Unified Cache Model:** The cache hierarchy treats instruction fetches and data operands identically, utilizing a unified instruction/data cache structure.

## 3 Component Implementation

### 3.1 Physical Memory Allocator

Memory management is handled through two interchangeable allocation paradigms, utilizing the Strategy Design Pattern to permit runtime switching.

#### 3.1.1 Free List Allocator

This implementation utilizes a Doubly Linked List to track free memory segments. Each block contains a metadata header defining its size and allocation status. Supported placement strategies include:

- **First Fit:** Traverses the list linearly and selects the first block sufficient for the request.

- **Best Fit:** Exhaustively searches for the block yielding the smallest residual hole, minimizing internal fragmentation.
- **Worst Fit:** Selects the largest available block, intended to maximize the size of remaining free chunks.

### 3.1.2 Buddy System Allocator

This allocator manages memory via an implicit binary tree of power-of-2 sized blocks.

- **Splitting:** Allocation requests are rounded up to the nearest power of 2. If a block is too large, it is recursively bisected until the optimal size is reached.
- **Coalescing:** Upon deallocation, the system identifies the block's "buddy" (calculated via bitwise XOR). If the buddy is free, the two blocks merge recursively into a higher-order block.

## 3.2 Virtual Memory Unit (MMU)

The MMU decouples the virtual address space from physical memory.

- **Page Table:** Implemented via a hashed map structure ( $\text{VPN} \rightarrow \text{PageEntry}$ ) to simulate sparse address spaces efficiently.
- **Replacement Policies:**
  - **FIFO:** Utilizes a queue to evict the oldest loaded page.
  - **LRU:** Maintains a list where successful accesses promote pages to the Most Recently Used (MRU) position, ensuring commonly accessed data remains resident.

## 3.3 Cache Hierarchy

The system employs a 3-level inclusive cache hierarchy to simulate memory latency.

Level	Size / Assoc	Latency	Role
L1 Cache	128 B / Direct Mapped	1 Cycle	High-speed lookup
L2 Cache	256 B / 2-Way Set	10 Cycles	Intermediate buffer
L3 Cache	512 B / 4-Way Set	100 Cycles	Last Level Cache (LLC)

Table 1: Cache Hierarchy Configuration

Addresses are decomposed into **Tag**, **Index**, and **Offset** fields. A miss at Level  $N$  propagates a request to Level  $N + 1$ , accumulating latency penalties.

## 4 Integration & Data Flow

The architecture strictly enforces a *Virtual*  $\rightarrow$  *Physical*  $\rightarrow$  *Cache* translation pipeline. The following diagram illustrates the lifecycle of a CPU memory request.

### Execution Sequence:

1. **Translation:** The Virtual Address is split into a Virtual Page Number (VPN) and Offset. The Page Table translates the VPN to a Physical Frame Number.
2. **Fault Handling:** If translation fails, a Page Fault is raised. The policy engine evicts a victim page (if necessary), loads the required data, and updates the Page Table.

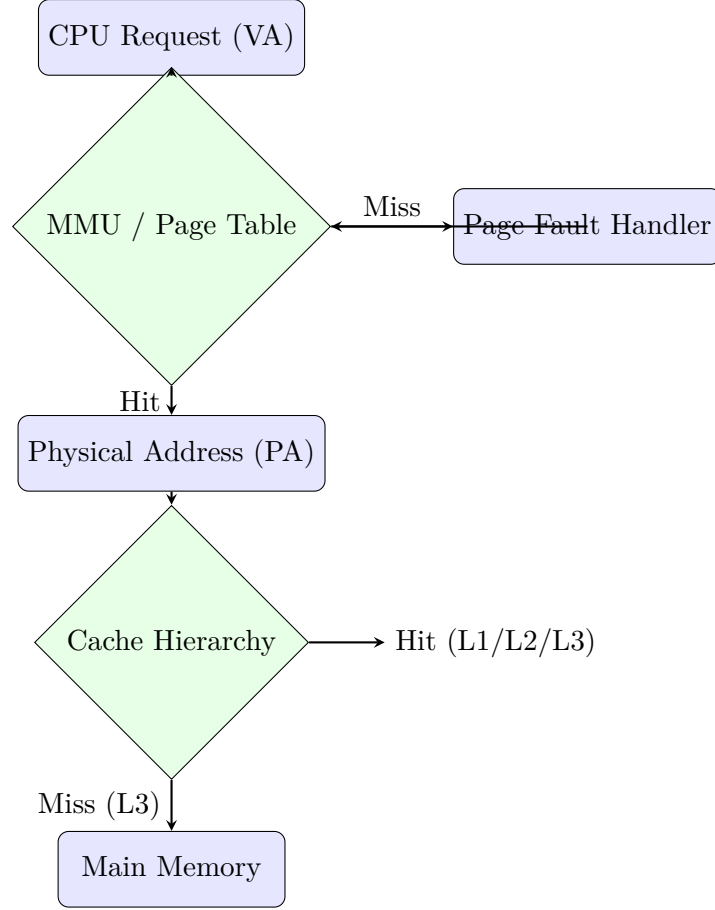


Figure 1: Memory Access Data Path

3. **Cache Access:** The resulting Physical Address is passed to the cache controller.

4. **Latency Accumulation:** Total cycle count is aggregated:

$$T_{total} = T_{translation} + T_{L1} + (Miss?T_{L2} : 0) + \dots + (Miss?T_{RAM} : 0).$$

## 5 Limitations & Constraints

- **Data Persistence:** The cache hierarchy stores metadata (Tags/Valid Bits) to simulate hit/miss behavior but does not replicate the actual data payload.
- **Direct Translation:** A Translation Lookaside Buffer (TLB) is not modeled; all translations incur the full Page Table lookup cost.
- **Thread Safety:** The simulator is designed for sequential execution and is not thread-safe.