Memory Hierarchy

The memory hierarchy is a fundamental design concept in embedded systems used to manage the trade-off between access speed, capacity, and cost. The goal is to provide the illusion of a large, fast, and inexpensive memory to the Central Processing Unit (CPU). The hierarchy is effective because it exploits the Principle of Locality, which states that program access tends to be localized in time and space:

- *Temporal Locality*: If an item is referenced, it is likely to be referenced again soon.
- *Spatial Locality*: If an item is referenced, items whose addresses are close to it are likely to be referenced soon.

Data Access Performance. When the CPU attempts to access data, one of two outcomes occurs:

(1) *Hit*: The accessed data is present in the upper level. The access is satisfied quickly by this upper, faster level. The Hit Ratio is calculated as the ratio of hits to total accesses.

$$\text{Hit Ratio} = \frac{\text{Hits}}{\text{Accesses}} \tag{1}$$

(2) *Miss*: The accessed data is absent from the upper level. A penalty is incurred as the block containing the required data must be copied from the lower, slower level to the upper level before the access can be satisfied.

1.1 Levels of the Hierarchy

The hierarchy is structured based on speed and proximity to the CPU:

(1) *SRAM (Static RAM)*: This is the Cache Memory level, located closest to the CPU. It is small, fast, and expensive. Recently accessed data and spatially nearby items are copied here from DRAM.

(2) *DRAM (Dynamic RAM)*: This is the Main Memory level. It is a larger, slower, and less expensive than SRAM. Recently accessed data and spatially nearby items are copied here from magnetic disk.

(3) *Magnetic Disk/Solid State Drive (SSD)*: This constitutes the slowest and largest level, often referred to as Secondary Storage. All system data and programs are originally stored here.

Memory Type	Access Time	Cost per GB	Role
SRAM	0.5 ns - 2.5 ns	\$2,000 - \$5,000	Cache (Level 1, 2, 3)
DRAM	50 ns - 70 ns	\$20 - \$75	Main Memory
Magnetic Disk	5 ms - 20 ms	\$0,20 - \$2	Secondary Storage

Table 2 / Comparison of memory technologies by access time, cost, and hierarchical role.

DRAM Technology. DRAM stores each data bit as an electrical charge in a small capacitor, with a single transistor acting as a switch to control access to that charge. The primary challenge of DRAM is its volatility leakage: the charge stored in the capacitor dissipates over time. To prevent data loss, DRAM must be periodically refreshed. This process involves reading the contents of a capacitor and immediately writing the charge back, restoring it to its original level. Refreshing is performed on an entire DRAM row at once, rather than on individual cells. This row-based operation minimized the overhead of refreshing, but means that during the refresh cycle, the entire row is unavailable for CPU access.

Bits within a DRAM chip are organized into a two-dimensional rectangular array of row and columns.

- *Row Buffer*: When a row is addressed, the entire row is accessed and copied into a temporary high-speed register called the row buffer (or cache). This allows multiple consecutive words from the same row to be supplied with very low latency, known as burst mode access. This reuse is critical for exploiting spatial locality.
- *Synchronous DRAM (SDRAM)*: SDRAM synchronizes its operation with the CPU clock, enabling control signals to be sent in anticipation of data needs. This allows for consecutive accesses to occur in bursts without having to send a full address for every word, significantly improving bandwidth.

Performance in advanced embedded systems is heavily reliant on memory bandwidth. Several techniques are used to increase the rate at which data can be transferred:

(1) *Data Transfer Rates*: Double Data Rate (DDR) DRAM is a technology that increases the transfer rate by performing data transfers on both the rising and falling edge of the clock signal, effectively doubling the memory clock speed. Quad Data Rate (QDR) DRAM is an enhancement, often used in high-speed applications like network switches, where the DDR principles are applied to separate input and output ports, further maximizing concurrent data movement.

(2) *Architectural Parallelism*: DRAM performance can be scaled up trough physical organization. The memory bus width determined how many words (or bytes) can be transferred in a single access cycle. A 4-word wide bus, for example, transfers 4 words simultaneously, which dramatically reduces the effective latency per byte for contiguous data transfers and improves peak bandwidth. A DRAM chip is divided into independent sections called banks. DRAM banking allows for simultaneous access to multiple independent banks. Interleaving is the process of mapping sequential memory addresses across different banks. While accessing a new bank may introduce a delay (e.g., a 20-cycle bank switching penalty), interleaving allows the system to overlap the row access delays, dramatically improving sustained throughput under random or scattered access patterns.

Flash Storage. Flash storage is a type of non-volatile semiconductor memory. It retains data without power and provides significant performance advantages over traditional disk storage, being typically 100x to 1,000x faster. It also offers advantages in physical footprint, power consumption, and robustness (due to no moving parts), though it remains more expensive per gigabyte than magnetic disk. Flash is broadly divided into two architectures, defined by the arrangement of the memory cells:

(1) *NOR Flash*: The memory cells are connected in parallel, similar to a NOR logic gate. It permits random read and write access at the byte level. It is primarily used for instruction memory in embedded systems (e.g., boot code) due to its rapid access and execution-in-place (XIP) capability.

(2) *NAND Flash*: The memory cells are connected in series, resembling a NAND logic gate. It is denser and cheaper per GB, but requires access in large blocks at a time (block-at-a-time access). It is used for high-capacity storage devices like USB keys, Solid State Drives (SSDs), and media storage.

A critical limitation of flash storage is cell wear-out: individual flash memory blocks can only sustain a finite number of Program/Erase cycles before they become unreliable. This makes flash unsuitable as a direct, unlimited replacement for DRAM or traditional magnetic disk. To mitigate this, wear leveling algorithms are implemented by the flash controller. These algorithms dynamically remap logical data addresses to different physical blocks, ensuring that write operations are spread evenly across all available blocks in the device, thus maximizing the overall lifespan.

Magnetic Disk Storage. Magnetic Disk Storage is a traditional form of non-volatile rotating magnetic storage. Although slower than flash, it offers the lowest cost per gigabyte for mass storage. A disk is composed of:

- *Platters*: Circular magnetic surfaces where data is stored.
- *Tracks*: Concentric rings on the platter surfaces.
- *Cylinders*: The set of tracks at the same radial distance on all platters.
- *Sectors*: The smallest unit of physical data storage, typically holding 512 bytes (or 4096 bytes in newer formats).

Each sector includes additional overhead fields, such as the sector ID, the actual payload, Error-Correcting Code (ECC) and synchronization fields and gaps.

Accessing a sector on a magnetic disk is a mechanical process involving multiple latency components:

$$T_{\text{access}} = T_{\text{queue}} + T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}} + T_{\text{controller}} \tag{3}$$

(1) *Queuing Delay*: The time spent waiting if other I/O requests are pending.

(2) *Seek Time*: The time required to move the read/write head assembly to the correct cylinder. Manufacturers quote an average seek time, but actual seek times are often smaller due to locality and operating system scheduling algorithms (like SCAN or C-SCAN) that order requests to minimize head movement.

(3) *Rotational Latency*: The time required for the desired sector to rotate under the head.

(4) *Data Transfer Time*: The time needed to physically read the data bits once the head is in position.

(5) *Controller Overhead*: Time for the disk controller to manage the request.

Modern disk drives include a smart controller that manages the internal complexities of the physical drive, presenting a simple logical sector interface to the host system (via standards like SCSI, ATA or SATA). These controllers often include a disk cache to prefetch sectors in anticipation of future accesses, which helps mask rotational latency and seek delay for sequential access patterns.

1.2 Cache Memory

Cache memory is the level of the memory hierarchy closest to the CPU, typically implemented using fast Static RAM (SRAM). Its primary role is to act as a temporary staging area for data that the CPU is likely to access soon, thereby reducing memory access latency. In a Direct-Mapped Cache, each memory block has only one possible location where it can be stored in the cache. This location is determined by a simple, deterministic mapping function derived from the memory address.

Address Mapping. The memory address is divided into three fields:

$$\text{Address} = [\text{Tag}]\ [\text{Index}]\ [\text{Offset}] \tag{4}$$

(1) *Block Offset*: These are the low-order bits of the address, which specify the byte offset within the block. The number of offset bits is $\log_2(\text{Block Size})$.

(2) *Cache Index*: These bits determine the specific line in the cache where the block will be stored. Since the number of blocks in the cache (N) is typically a power of 2, the index is found by taking the block's address modulo N. These are the bits immediately to the left of the Block Offset.

(3) *Tag*: These are the high-order bits of the address. Since many main memory blocks can map to the same cache index, the Tag is necessary to uniquely identify which particular memory block is currently stored in that cache location.

When the CPU issues an address, the cache controller extracts the Index to find the corresponding cache line. To ensure correctness, two checks are performed:

- *Tag Match*: The stored tag in the cache line must match the Tag field of the requested address.
- *Valid Bit*: A Valid Bit is stored with each cache line. It is set to '1' if the line currently holds valid data from main memory and '0' otherwise. All valid bits are initialized to '0'. A hit occurs only if both the Tag matches and the Valid Bit is '1'.

A crucial design parameter is the Block Size. Increasing the block size generally reduces the miss rate because it takes greater advantage of spatial locality: when a miss occurs, more neighboring data is brought into the cache. However, for a cache of fixed total capacity, increasing the block size means the cache holds fewer blocks overall. This leads to greater competition for the available cache lines. If the larger block brings in data that is not used, it occupied space needed by other, more frequently accessed data. This unused data is referred to as cache pollution and can ultimately lead to an increased miss rate, overriding the initial benefit. To mitigate the performance impact of a larger miss penalty, two techniques can be employed:

- *Early Restart*: The CPU is allowed to continue execution as soon as the specific word that caused the miss is retrieved from the next memory level, even if the rest of the block has not yet arrived.
- *Critical-Word-First*: A variation where the requested word is transferred first, and the rest of the block follows in a non-sequential order.

Cache Write Policies. In the case of a Cache Miss, the CPU pipeline stalls. The entire block containing the requested data is fetched from the next lower level of the memory hierarchy (e.g., main memory/DRAM). In the case of Instruction Cache Miss, the instruction fetch step of the pipeline must be restarted once the instruction block arrives. On the other hand, for a Data Cache Miss, the data access step of the pipeline must wait to complete the data access once the data block arrives.

Cache write policies dictate how and when a write operation from the CPU updates the data in the cache and the corresponding block in the next level of the memory hierarchy. The primary goal is to maintain data coherence.

(1) *Write-Through*: In the Write-Through policy, on a data write hit, the block in the cache is updated, and the write is immediately forwarded to the next level of hierarchy. Cache and memory are consistently synchronized on every write. However, writes become significantly slower because the CPU must wait for the main memory access to complete, which is much slower than the cache. If the baseline Cycles Per Instruction (CPI) is 1, and 10% of instructions are write operations, and the write latency to memory is 100 cycles, the effective CPI is dramatically increased:

Write Policy	**Miss Alternatives**	**Description**
Write-Through	Write Allocate (Fetch on Miss)	The block is fetched from memory, the data is written on the cache block, and the write is passed through to memory.
	No-Write Allocate (Write Around)	The block is not fetched. The write operation bypasses the cache and is performed only on the next level of the hierarchy. This is often used because many programs write a full block of data sequentially before reading it, making the intermediate fetch unnecessary ("write around").
Write-Back	Write Allocate (Fetch on Miss)	The missing block is usually fetched from memory first. The data is then written into the cache block, and the dirty bit is set. This ensures that the entire block is coherent before modification.

Table 5 / How cache write policies (write-through/write-back) handle write misses.

$$\text{Effective CPI} = 1 + (0.10 \times 100) = 11 \tag{6}$$

(2) *Write-Back*: In the Write-Back policy, on a data write hit, only the block in the cache is updated. The corresponding block in main memory is not immediately updated. A dirty bit (or modified bit) is associated with each cache block. It is set to '1' when the block is modified by a write. The block is written back to the next level of the hierarchy only when that dirty block is selected for replacement by a new block. This policy leads to much lower write traffic to memory and much faster write operations for the CPU, but it introduces complexity in maintaining coherence. A write buffer is often used here too, but to store the evicted dirty block temporarily while the replacement block is being read into the cache.

Performance. Cache performance is measured by decomposing CPU execution time into two key components: program execution cycles, which include the time spent on cache hits, and memory stall cycles, primarily incurred due to cache misses. Under simplifying assumptions, memory stall cycles can be estimated as the product of memory accesses per program, the miss rate, and the miss penalty. Since memory accesses are often proportional to instructions executed, this can also be expressed as:

$$\text{Instruction per Program} \times \text{Misses per Instruction} \times \text{Miss Penalty} \tag{7}$$

offering a practical way to quantify the performance cost of cache misses.

Average Memory Access Time (AMAT) is a critical metric for evaluating cache performance, as it combines both hit time and the cost of misses. While miss and penalty are often emphasized, hit time itself significantly impacts overall speed, especially in high-frequency CPUs where every cycle counts. The formula is straightforward:

$$\text{AMAT} = \text{Hit Time} + (\text{Miss Rate} \times \text{Miss Penalty}) \tag{8}$$

Cache Organization and Design

To overcome the high conflict misses inherent in a direct-mapped cache, where a memory block has only one possible location, Associative Caches are used. Associativity allows a given memory block to be placed in multiple possible cache entries:

- *Fully Associative Cache*: In a Fully Associative Cache, a block can be placed in any available cache entry. There is no index; the cache is searched using only the Tag. To check for a hit, all cache entries must be searched simultaneously. This requires a comparator circuit for every single entry, making it complex and expensive to implement.
- *N-Way Set Associative Cache*: The N-Way Set Associative Cache offers a compromise between the speed of direct mapping and the flexibility of fully associative mapping. The cache is divided into a number of sets, and each set contains N entries. A block is mapped to a specific set determined by the calculation: Block Number mod # Sets in cache. Once the set is determined, only the N entries within that specific set are searched simultaneously. This requires only N comparators, making it much less expensive than a fully associative cache.

When a set is full and a new block needs to be brought in, a replacement policy is needed to decide which existing block to evict. This choice is only necessary for set-associative and fully-associative caches, as a direct-mapped cache has no choice.

(1) *Prefer Non-Valid Entry*: If an entry within the set has an invalid bit (meaning it contains no useful data), that entry is always chosen for replacement.

(2) *Least-Recently Used (LRU)*: If all entries are valid, the policy chooses the entry that has been unused for the longest period of time. LRU is highly effective but becomes computationally complex for associativity beyond 4-way.

(3) *Random*: This policy randomly selects an entry to replace. It provides approximately the same performance as LRU for caches with high associativity.

Multilevel Caches. Modern systems use multiple levels of caches to reduce the average memory access time.

- *Primary Cache (L1)*: Attached directly to the CPU. It is small and very fast (designed for minimal hit time). It services the vast majority of CPU requests.
- *Level-2 Cache (L2)*: Services the misses that occur in the primary cache. It is larger and slower than L1, but still significantly faster than main memory (designed for a low miss rate).
- *Main Memory (DRAM)*: Services the misses that occur in the L2 cache. Some high-end systems may also include a Level-3 (L3) cache.