Joshua Emralino

Professor Khan

CISC 211

20 March 2025

Memory's Role in Program Execution

The primary function of a computer's memory is to store machine-readable data that the CPU uses to execute operations. To do this efficiently, a memory device must balance speed, capacity, efficiency, and affordability. However, no single memory technology can simultaneously optimize all these factors. Instead, modern computer systems use a hierarchical memory structure, where different types of memory serve distinct roles based on their performance and storage trade-offs. At the highest level, extremely fast—but also expensive—memory devices ensure that frequently accessed data is immediately available, while slower, larger storage devices provide long-term data retention.

To illustrate the necessity of this hierarchy, I analyzed my Gen 12 ThinkPad X1 Carbon using the lscpu command. The output indicated that my Intel Core Ultra 7 165U CPU can theoretically execute nearly 200 billion instructions per second. However, my laptop's 1TB SSD—the main device responsible for storing programs and files—has a maximum transfer speed of about 7 GB/s. Assuming an average instruction size of 4 bytes, that equates to approximately 1.75 billion instructions per second. If the CPU had to fetch every instruction directly from the SSD, its efficiency would drop by more than a factor of 100. Moreover, this does not even account for the added latency of accessing SSD storage via data bus, which further slows retrieval. These facts demonstrate the necessity of faster intermediary memory devices that bridge the gap between the CPU and long-term storage.

The primary type of intermediary memory is Random Access Memory (RAM). RAM temporarily stores active program data so that the CPU can access it quickly. Unlike long-term storage devices such as SSDs and HDDs, RAM is volatile, meaning it loses all data when power is removed. This is because RAM stores data using transistors and capacitors that require regular refreshing. This design allows for high-speed operation, making RAM ideal for storing actively used application and system data. However, not all types of RAM are the same. Two primary categories are Static RAM (SRAM) and Dynamic RAM (DRAM).

Cache memory is implemented using SRAM, which does not require periodic refresh cycles and can therefore operate at very high speeds with minimal overhead. However, each SRAM cell requires more transistors than a DRAM cell, making SRAM both more expensive and physically larger per bit of storage. Because of its speed and direct integration into the CPU die, SRAM is used for L1, L2, and L3 cache memory. Modern processors implement multiple levels of cache to balance speed and capacity. L1 cache is the smallest and fastest, typically dedicated to each processor core for immediate data retrieval. L2 cache is larger yet slightly slower and can be either shared by a single core or across a small cluster of cores. Finally, L3 cache is the largest and slowest on-chip cache level, usually shared among all cores so that any core can quickly access data that may be needed by multiple cores simultaneously.

Using the same lscpu command, I inspected my laptop's cache specifications: the Intel Core Ultra 7 165U CPU has 12 MiB of L3 cache, 10 MiB of L2 cache, 640 KiB of L1 instruction cache, and 352 KiB of L1 data cache, totaling 23 MiB of cache memory. Compared to my 1TB SSD, 23 MiB may seem minuscule. However, cache memory is designed for speed and low latency, allowing the CPU to operate near its maximum potential without waiting on slower memory systems.

Together, cache memory and RAM serve as intermediaries between the CPU and permanent storage. Cache memory ensures that the most frequently accessed instructions and data are quickly accessible. Meanwhile, system RAM offers a larger, high-speed memory location where actively running programs can load and retrieve data efficiently. This hierarchy helps the CPU operate near peak performance by reducing the need for frequent accesses to significantly slower SSD or HDD storage. Without these memory layers, every data request would require fetching information from long-term storage, drastically slowing processing, increasing power consumption, and creating severe performance bottlenecks.