Unit-04/Lecture-01

MEMORY ORGANISATION

Memory map

In computer science, a **memory map** is a structure of data (which usually resides in memory itself) that indicates how memory is laid out. Memory maps can have a different meaning in different parts of the operating system. It is the fastest and most flexible cache organization uses an associative memory. The associative memory stores both the address and content of the memory word.

In the boot process, a memory map is passed on from the firmware in order to instruct an operating system kernel about memory layout. It contains the information regarding the size of total memory, any reserved regions and may also provide other details specific to the architecture.

In virtual memory implementations and memory management units, a memory map refers to page tables, which store the mapping between a certain process's virtual memory layout and how that space relates to physical memory addresses.

In native debugger programs, a memory map refers to the mapping between loaded executable/library files and memory regions. These memory maps are used to resolve memory addresses (such as function pointers) to actual symbols.

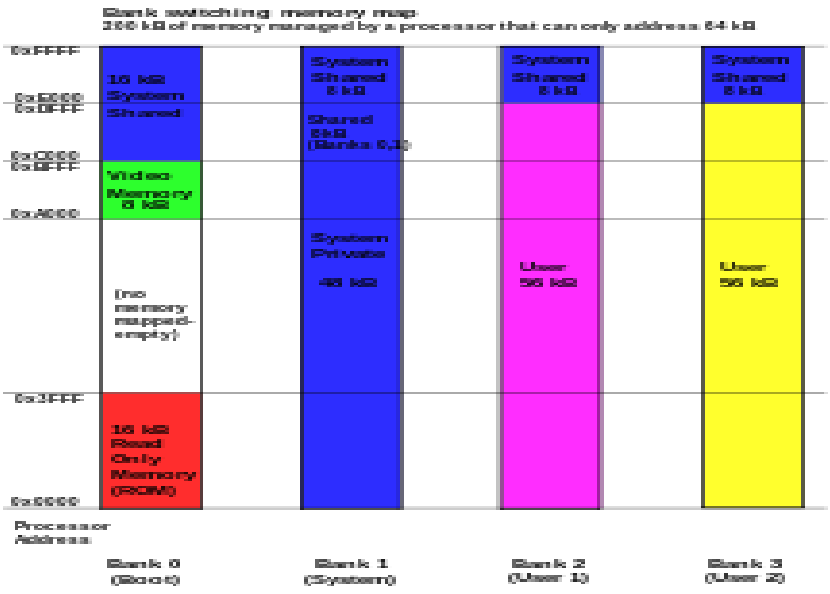


Fig.4.1

A memory map is a massive table, in effect a database, that comprises complete information about how the memory is structured in a computer system. A memory map works something like a gigantic office organizer. In the map, each computer file has a unique memory address reserved especially for it, so that no other data can inadvertently overwrite or corrupt it

Unit-04/Lecture-02

Memory hierarchy

The term **memory hierarchy** is used in computer architecture when discussing performance issues in computer architectural design, algorithm predictions, and the lower level programming constructs such as involving locality of reference. A "memory hierarchy" in computer storage distinguishes each level in the "hierarchy" by response time. Since response time, complexity, and capacity are related,[1] the levels may also be distinguished by the controlling technology.

The many trade-offs in designing for high performance will include the structure of the memory hierarchy, i.e. the size and technology of each component. So the various components can be viewed as forming a hierarchy of memories (m_1,m_2,...,m_n) in which each member m_i is in a sense subordinate to the next highest member m_{i+1} of the hierarchy. To limit waiting by higher levels, a lower level will respond by filling a buffer and then signaling to activate the transfer.

There are four major storage levels.[1]

1. *Internal* – Processor registers and cache.
2. Main – the system RAM and controller cards.
3. On-line mass storage – Secondary storage.
4. Off-line bulk storage – Tertiary and Off-line storage.

This is a general memory hierarchy structuring. Many other structures are useful. For example, a paging algorithm may be considered as a level for virtual memory when designing a computer architecture.

Fig. 4.2

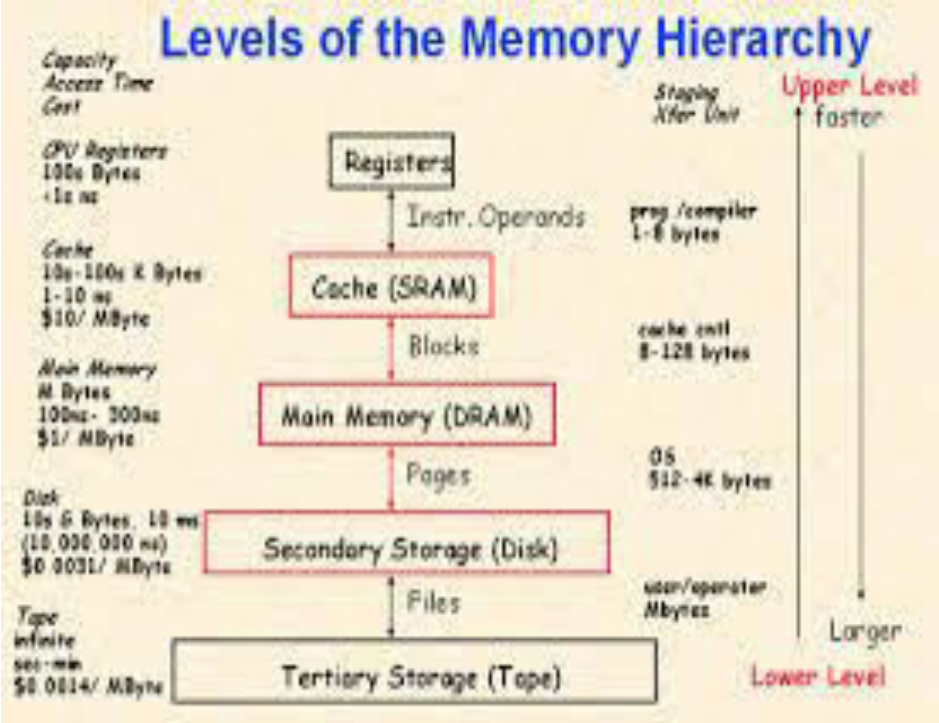


Fig 4.3

	RGPV QUESTIONS	Year	Marks
Q.1	Draw and explain the memory hierarchy in a digital computer. What are the advantages of cache memory over main memory	June 2011	10
Q.2	What is meant by memory hierarchy in a computer system also explain what is Meant by associative memory and virtual memory?	June 2010	10

Unit-04/Lecture-03

Cache Memory

Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

The basic purpose of cache memory is to store program instructions that are frequently re-referenced by software during operation. Fast access to these instructions increases the overall speed of the software program.

As the microprocessor processes data, it looks first in the cache memory; if it finds the instructions there (from a previous reading of data), it does not have to do a more time-consuming reading of data from larger memory or other data storage devices.

Most programs use very few resources once they have been opened and operated for a time, mainly because frequently re-referenced instructions tend to be cached. This explains why measurements of system performance in computers with slower processors but larger caches tend to be faster than measurements of system performance in computers with faster processors but more limited cache space.
Multi-tier or multilevel caching has become popular in server and desktop architectures, with different levels providing greater efficiency through managed tiering. Simply put, the less frequently access is made to certain data or instructions, the lower down the cache level the data or instructions are written.

Cache memory levels

Cache memory is fast and expensive. Traditionally, it is categorized as "levels" that describe its closeness and accessibility to the microprocessor:
Level 1 (L1) cache is extremely fast but relatively small, and is usually embedded in the processor chip (CPU).
Level 2 (L2) cache is often more capacious than L1; it may be located on the CPU or on a separate chip or coprocessor with a high-speed alternative system bus interconnecting the cache to the CPU, so as not to be slowed by traffic on the main system bus.
Level 3 (L3) cache is typically specialized memory that works to improve the performance of L1 and L2. It can be significantly slower than L1 or L2, but is usually double the speed of RAM. In the case of multicore processors,

each core may have its own dedicated L1 and L2 cache, but share a common L3 cache. When an instruction is referenced in the L3 cache, it is typically elevated to a higher tier cache.

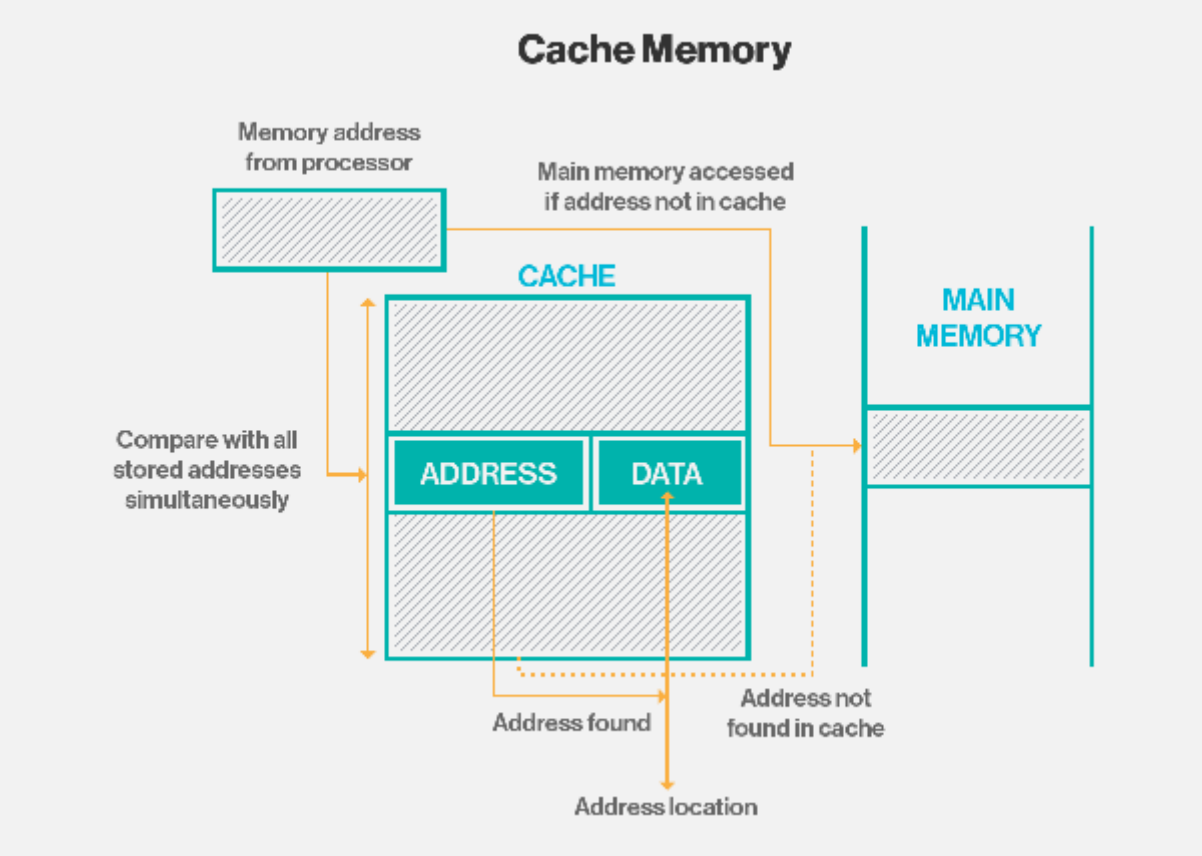


Fig 4.4

	RGPV QUESTIONS	Year	Marks
Q.1	With the help of a diagram explain how cache is used in cache organization .Explain mapping techniques.	June 2014	7
Q.2	Explain cache memory organization. Which mapping techniques are used in cache memory	June 2010	10

Unit-04/Lecture-04

Memory cache configurations

Caching configurations continue to evolve, but memory cache traditionally works under three different configurations:

- **Direct mapping**, in which each block is mapped to exactly one cache location. Conceptually, this is like rows in a table with three columns: the data block or cache line that contains the actual data fetched and stored, a tag that contains all or part of the address of the fetched data, and a flag bit that connotes the presence of a valid bit of data in the row entry.
- **Fully associative mapping** is similar to direct mapping in structure, but allows a block to be mapped to any cache location rather than to a pre-specified cache location (as is the case with direct mapping).
- **Set associative mapping** can be viewed as a compromise between direct mapping and fully associative mapping in which each block is mapped to a subset of cache locations. It is sometimes called *N-way set associative mapping*, which provides for a location in main memory to be cached to any of "N" locations in the L1 cache.

Specialized caches

In addition to instruction and data caches, there are other caches designed to provide specialized functions in a system. By some definitions, the L3 cache is a specialized cache because of its shared design. Other definitions separate instruction caching from data caching, referring to each as a specialized cache.

Other specialized memory caches include the translation lookaside buffer (TLB) whose function is to record virtual address to physical address translations.

Still other caches are not, technically speaking, memory caches at all. Disk caches, for example, may

leverage RAM or flash memory to provide much the same kind of data caching as memory caches do with CPU instructions. If data is frequently accessed from disk, it is cached into DRAM or flash-based silicon storage technology for faster access and response.

In the video below, Dennis Martin, founder and president of Demartek LLC, explains the pros and cons of using solid-state drives as cache and as primary storage.

Specialized caches also exist for such applications as Web browsers, databases, network address binding and client-side Network File System protocol support. These types of caches might be distributed across multiple networked hosts to provide greater scalability or performance to an application that uses them.

Increasing cache size

L1, L2 and L3 caches have been implemented in the past using a combination of processor and motherboard components. Recently, the trend has been toward consolidating all three levels of memory caching on the CPU itself. For this reason, the primary means for increasing cache size has begun to shift from the acquisition of a specific motherboard with different chipsets and bus architectures to buying the right CPU with the right amount of integrated L1, L2 and L3 cache

Unit-04/Lecture-05

A **CPU cache** is a cache used by the central processing unit (CPU) of a computer to reduce the average time to access data from the main memory. The cache is a smaller, faster memory which stores copies of the data from frequently used main memory locations. Most CPUs have different independent caches, including instruction and data caches, where the data cache is usually organized as a hierarchy of more cache levels (L1, L2 etc.)

When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory.

Most modern desktop and server CPUs have at least three independent caches: an **instruction cache** to speed up executable instruction fetch, a **data cache** to speed up data fetch and store, and a translation lookaside buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data. The data cache is usually organized as a hierarchy of more cache levels (L1, L2, etc.; see also multi-level caches below).

Cache entries

Data is transferred between memory and cache in blocks of fixed size, called cache lines. When a cache line is copied from memory into the cache, a cache entry is created. The cache entry will include the copied data as well as the requested memory location (now called a tag).

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. The cache checks for the contents of the requested memory location in any cache lines that might contain that address. If the processor finds that the memory location is in the cache, a cache hit has occurred. However, if the processor does not find the memory location in the cache, a cache miss has occurred. In the case of:

- a cache hit, the processor immediately reads or writes the data in the cache line
- a cache miss, the cache allocates a new entry and copies in data from main memory, then the

request is fulfilled from the contents of the cache.

Cache performance

The proportion of accesses that result in a cache hit is known as the hit rate, and can be a measure of the effectiveness of the cache for a given program or algorithm.

Read misses delay execution because they require data to be transferred from memory, which is much slower than reading from the cache. Write misses may occur without such penalty, since the processor can continue execution while data is copied to main memory in the background.

	RGPV QUESTIONS	Year	Marks
Q.1	Explain hit ratio in cache organization	June 2014	2

Unit-04/Lecture-06

Replacement policies

In order to make room for the new entry on a cache miss, the cache may have to evict one of the existing entries. The heuristic that it uses to choose the entry to evict is called the replacement policy. The fundamental problem with any replacement policy is that it must predict which existing cache entry is least likely to be used in the future. Predicting the future is difficult, so there is no perfect way to choose among the variety of replacement policies available.

One popular replacement policy, least-recently used (LRU), replaces the least recently accessed entry.

Marking some memory ranges as non-cacheable can improve performance, by avoiding caching of memory regions that are rarely re-accessed. This avoids the overhead of loading something into the cache without having any reuse.

- Cache entries may also be disabled or locked depending on the context.

Write policies

If data is written to the cache, at some point it must also be written to main memory. The timing of this write is known as the write policy.

In a write-through cache, every write to the cache causes a write to main memory.

Alternatively, in a write-back or copy-back cache, writes are not immediately mirrored to the main memory. Instead, the cache tracks which locations have been written over (these locations are marked dirty). The data in these locations are written back to the main memory only when that data is evicted from the cache. For this reason, a read miss in a write-back cache may sometimes require two memory accesses to service: one to first write the dirty location to memory and then another to read the new location from memory.

There are intermediate policies as well. The cache may be write-through, but the writes may be held in a store data queue temporarily, usually so that multiple stores can be processed together (which can

reduce bus turnarounds and improve bus utilization).

The data in main memory being cached may be changed by other entities (e.g. peripherals using direct memory access or multi-core processor), in which case the copy in the cache may become out-of-date or stale. Alternatively, when a CPU in a multiprocessor system updates data in the cache, copies of data in caches associated with other CPUs will become stale. Communication protocols between the cache managers which keep the data consistent are known as cache coherence protocols.

CPU stalls

The time taken to fetch one cache line from memory (read latency) matters because the CPU will run out of things to do while waiting for the cache line. When a CPU reaches this state, it is called a stall.

As CPUs become faster, stalls due to cache misses displace more potential computation; modern CPUs can execute hundreds of instructions in the time taken to fetch a single cache line from main memory. Various techniques have been employed to keep the CPU busy during this time.

- Out-of-order CPUs (the Pentium Pro and later Intel designs, for example) attempt to execute independent instructions after the instruction that is waiting for the cache miss data.
- Another technology, used by many processors, is simultaneous multithreading (SMT), or — in Intel's terminology — hyper-threading (HT), which allows an alternate thread to use the CPU core while a first thread waits for data to come from main memory.

Unit-04/Lecture-07

Associative memory

Associative memory in computer organization is when memory is accessed through content rather thanthrough a specific address. Associative memory is also known as associative storage, associative array or content-addressable memory, or CAM.

Associative memory is found on a computer hard drive and used only in specific high-speed searching applications. Most computer memory known as random access memory, or RAM, works through the computer user providing a memory address and then the RAM will return whatever data is stored at that memory address. However, CAM works through the computer user providing a data word and then searching throughout the entire computer memory to see if the word is there. If the computer finds the data word then it offers a list of all of the storage addresses where the word was found for the user.

CAM is faster than RAM in almost every search application, but many people stick with RAM for their computers because a computer with CAM is more expensive than RAM. The reason for the price increase for CAM computers is because with CAM computers, each cell has to have the full storage capability and logic circuits that can match content with external argument. Associative memory computers are best for users that require searches to take place quickly and whose searches are critical for job performance on the machine.

or

A storage unit of digital computers in which selection (entry) is performed not according to concrete address but rather according to a preset combination (association) of attributes characteristic of the desired information. Such attributes can be part of a word (number), attached to it for detection among other words, certain features of the word itself (for example, the presence of specific codes in

its digits), the absolute value of a word, its presence in a preset range, and so on.

The operation of an associative memory is based on the representation of all information in the form of a sequence of zones according to properties and characteristic attributes. In this case the retrieval of information is reduced to the determination of the zone according to the preset attributes by means of scanning and comparison of those attributes with the attributes that are stored in the associative memory. There are two basic methods of realizing the associative memory. The first is the construction of a memory with storage cells that have the capability of performing simultaneously the functions of storage, nondestructive reading, and comparison. Such a method of realizing an associative memory is called network parallel-associative—that is, the required sets of attributes are preserved in all the memory cells, and the information that possesses a given set of attributes is searched for simultaneously and independently over the entire storage capacity. Card indexes for edge-punched cards are prototypes of such an associative memory. Thin-film kryotrons, transfluxors, biaxes, magnetic thin films, and so on are used as storage elements of network-realized associative memories.

The second method of realizing an associative memory is the programmed organization (modeling) of the memory. It consists of the establishment of associative connections between the information contained in the memory by means of ordered arrangement of the information in the form of sequential chains or groups (lists) connected by linkage addresses whose codes are stored in the same memory cells. This procedure is the more suitable for practical realization in dealing with large volumes of information because it provides for the use of conventional accumulators with address reference.

The use of an associative memory considerably facilitates the programming and solution of

informational-logical problems and accelerates by hundreds (thousands) of times the speed of retrieval, analysis, classification, and processing of data.

	RGPV QUESTIONS	Year	Marks
Q.1	Explain a typical associative memory organization Describe the various steps involved in accessing the content of the associative memory	June 2012	10
Q.2	What is Associative Memory ? Explain the concept of address space and memory space in virtual memory	June 2010 June2011	10

Unit-04/Lecture-08

Virtual memory

In computing, **virtual memory** is a memory management technique that is implemented using both hardware and software. It maps memory addresses used by a program, called *virtual addresses*, into *physical addresses* in computer memory. Main storage as seen by a process or task appears as a contiguous address space or collection of contiguous segments. The operating system manages virtual address spaces and the assignment of real memory to virtual memory. Address translation hardware in the CPU, often referred to as a memory management unit or *MMU*, automatically translates virtual addresses to physical addresses. Software within the operating system may extend these capabilities to provide a virtual address space that can exceed the capacity of real memory and thus reference more memory than is physically present in the computer.

The primary benefits of virtual memory include freeing applications from having to manage a shared memory space, increased security due to memory isolation, and being able to conceptually use more memory than might be physically available, using the technique of paging.

Properties: Virtual memory makes application programming easier by hiding fragmentation of physical memory; by delegating to the kernel the burden of managing the memory hierarchy (eliminating the need for the program to handle overlays explicitly); and, when each process is run in its own dedicated address space, by obviating the need to relocate program code or to access memory with relative addressing.

Memory virtualization can be considered a generalization of the concept of virtual memory.

or

Virtual memory is a feature of an operating system (OS) that allows a computer to compensate for shortages of physical memory by temporarily transferring pages of data from random access memory (RAM) to disk storage.

Eventually, the OS will need to retrieve the data that was moved to temporarily to disk storage -- but remember, the only reason the OS moved pages of data from RAM to disk storage to begin with was

because it was running out of RAM. To solve the problem, the operating system will need to move other pages to hard disk so it has room to bring back the pages it needs right away from temporary disk storage. This process is known as paging or swapping and the temporary storage space on the hard disk is called a pagefile or a swap file.

Swapping, which happens so quickly that the end user doesn't know it's happening, is carried out by the computer's memory manager unit (MMU). The memory manager unit may use one of several algorithms to choose which page should be swapped out, including Least Recently Used (LRU), Least Frequently Used (LFU) or Most Recently Used (MRU).

	RGPV QUESTIONS	Year	Marks
Q.1	Give a short note on virtual memory organization .what is Paging?	June 2014	7

Unit-04/Lecture-09

A hardware device or circuit that supports virtual memory and paging by translating virtual addresses into physical addresses.

The virtual address space (the range of addresses used by the processor) is divided into pages, whose size is 2^N, usually a few kilobytes. The bottom N bits of the address (the offset within a page) are left unchanged. The upper address bits are the (virtual) page number. The MMU contains a page table which is indexed (possibly associatively) by the page number. Each page table entry (PTE) gives the physical page number corresponding to the virtual one. This is combined with the page offset to give the complete physical address.

A PTE may also include information about whether the page has been written to, when it was last used (for a least recently used replacement algorithm), what kind of processes (user mode, supervisor mode) may read and write it, and whether it should be cached.

It is possible that no physical memory (RAM) has been allocated to a given virtual page, in which case the MMU will signal a "page fault" to the CPU. The operating system will then try to find a spare page of RAM and set up a new PTE to map it to the requested virtual address. If no RAM is free it may be necessary to choose an existing page, using some replacement algorithm, and save it to disk (this is known as "paging"). There may also be a shortage of PTEs, in which case the OS will have to free one for the new mapping.

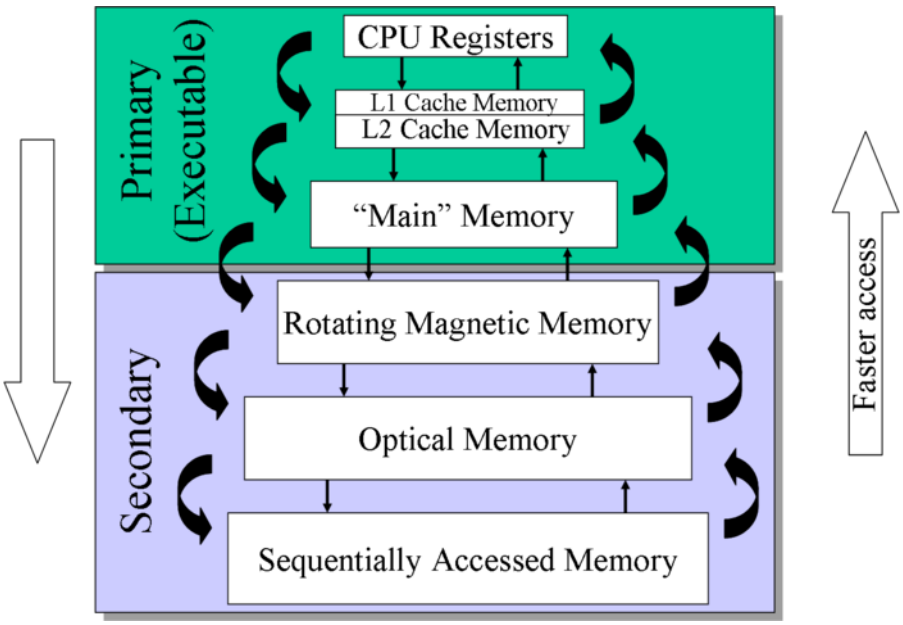
In a multitasking system all processes compete for the use of memory and of the MMU. Some memory management architectures allow each process to have its own area or configuration of the page table, with a mechanism to switch between different mappings on a process switch. This means that all processes can have the same virtual address space rather than require load-time relocation.

An MMU also solves the problem of fragmentation of memory. After blocks of memory have been allocated and freed, the free memory may become fragmented (discontinuous) so that the largest contiguous block of free memory may be much smaller than the total amount. With virtual memory, a contiguous range of virtual addresses can be mapped to several non-contiguous blocks of physical memory.

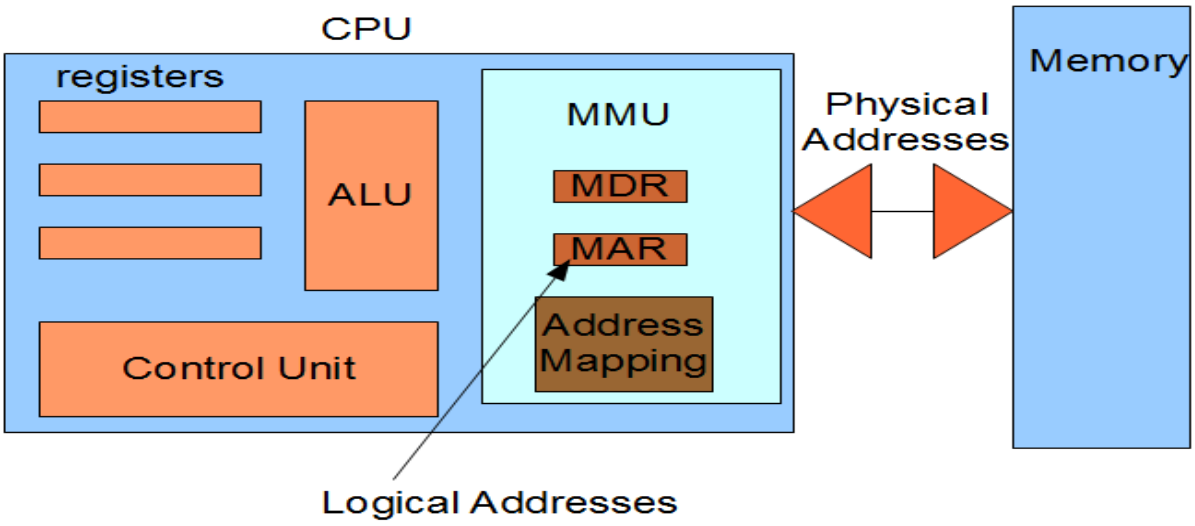
In early designs memory management was performed by a separate integrated circuit such as the MC 68851

used with the Motorola 68020 CPU in the Macintosh II or the Z8015 used with the Zilog Z80 family of processors. Later CPUs such as the Motorola 68030 and the ZILOG Z280 have MMUs on the same IC as the CPU.

Computers have several different types of memory. This memory is often viewed as a hierarchy as shown below



The memory addresses that it uses to reference its data is the *logical address*. The real time translation to the *physical address* is performed in hardware by the CPU's *Memory Management Unit* (MMU). The MMU has two special registers that are accessed by the CPU's control unit. A data to be sent to main memory or retrieved from memory is stored in the *Memory Data Register* (MDR). The desired logical memory address is stored in the *Memory Address Register* (MAR). The address translation is also called address binding and uses a memory map that is programmed by the operating system.



Before memory addresses are loaded on to the system bus, they are translated to physical addresses by the MMU