Compiled by: Dr. Roshan Koju,
Department of Graduate Studies NCIT,
Pokhara University
2024/25

Unit II: Memory Technology

Computer memory is a physical device or system within a computer that stores data, instructions, and information, either temporarily or permanently, for use by the processor and other components. Memory is essential for a computer's operation as it allows the system to store and quickly access the data required for running applications and processing tasks.

The key characteristics of memory devices or memory system are as follows:

a) Location
b) Capacity
c) Unit of Transfer
d) Access Method
e) Performance
f) Physical type
g) Physical characteristics
h) Organization

a. **Location:**

It deals with the location of the memory device in the computer system. There are three possible locations:

- CPU : This is often in the form of CPU registers and small amount of cache
- Internal or main: This is the main memory like RAM or ROM. The CPU can directly access the main memory.
- External or secondary: It comprises of secondary storage devices like hard disks, magnetic tapes. The CPU doesn't access these devices directly. It uses device controllers to access secondary storage devices.

b) **Capacity:**

The capacity of any memory device is expressed in terms of: i)word size ii)Number of words

- **Word size:** Words are expressed in bytes (8 bits). A word can however mean any number of bytes. Commonly used word sizes are 1 byte (8 bits), 2bytes (16 bits) and 4 bytes (32 bits).
- **Number of words:** This specifies the number of words available in the particular memory device. For example, if a memory device is given as 4K x 16.This means the device has a word size of 16 bits and a total of 4096(4K) words in memory.

c) **Unit of Transfer:**

It is the maximum number of bits that can be read or written into the memory at a time. In case of main memory, it is mostly equal to word size. In case of external memory, unit of transfer is not limited to the word size; it is often larger and is referred to as blocks.

d) **Access Methods:**

It is a fundamental characteristic of memory devices. It is the sequence or order in which memory can be accessed. There are three types of access methods:

- **Random Access:** If storage locations in a particular memory device can be accessed in any order and access time is independent of the memory location being accessed. Such memory devices are said to have a random access mechanism. RAM (Random Access Memory) IC's use this access method.

- **Serial Access:** If memory locations can be accessed only in a certain predetermined sequence, this access method is called serial access. Magnetic Tapes, CD-ROMs employ serial access methods.
- **Semi random Access:** Memory devices such as Magnetic Hard disks use this access method. Here each track has a read/write head thus each track can be accessed randomly but access within each track is restricted to a serial access.

e) **Performance:** The performance of the memory system is determined using three parameters

- **Access Time:** In random access memories, it is the time taken by memory to complete the read/write operation from the instant that an address is sent to the memory. For non-random access memories, it is the time taken to position the read write head at the desired location. Access time is widely used to measure performance of memory devices.
- **Memory cycle time:** It is defined only for Random Access Memories and is the sum of the access time and the additional time required before the second access can commence.
- **Transfer rate:** It is defined as the rate at which data can be transferred into or out of a memory unit.

f) **Physical type**: Memory devices can be either semiconductor memory (like RAM) or magnetic surface memory (like Hard disks).

g) **Physical Characteristics:**

  - **Volatile/Non- Volatile:** If a memory devices continues hold data even if power is turned off. The memory device is non-volatile else it is volatile.

h) **Organization:**

- **Erasable/Non-erasable:** The memories in which data once programmed cannot be erased are called Non-erasable memories. Memory devices in which data in the memory can be erased is called erasable memory.
  E.g. RAM(erasable), ROM(non-erasable).

2.1 Hierarchical Memory Structures

The memory hierarchy is an organization of memory storage in computer systems, arranged to balance performance and cost. It is designed to ensure efficient access to data by categorizing memory types based on speed, size, and cost. This structure allows frequently used data to be stored in faster, smaller, and more expensive memory, while less frequently used data resides in slower, larger, and cheaper memory. The purpose of using hierarchical memory is to achieve:

- **Optimize Performance:** Minimize the time required to access data.
- **Cost Efficiency:** Balance the trade-offs between fast, expensive memory and slow, affordable storage.
- **Scalability:** Enable systems to handle varying workloads efficiently.

The idea centers on a fundamental property of computer programs known as locality. Programs with good locality tend to access the same set of data items over and over again, or they tend to access sets of nearby data items. Programs with good locality tend to access more data items from the upper levels of the memory hierarchy than programs with poor locality, and thus run faster.

2.1.1 Memory Levels

The memory hierarchy consists of multiple levels, each with specific characteristics:

1. **Register Memory:**
    - **Location:** Inside the CPU.
    - **Speed:** Fastest memory.
    - **Size:** Smallest capacity (typically a few kilobytes).
    - **Purpose:** Store data being immediately processed by the CPU.
    - **Example:** Program counters, instruction registers.
2. **Cache Memory:**
    - **Location:** Close to the CPU.
    - **Speed:** Faster than RAM but slower than registers.
    - **Size:** Larger than registers (megabytes range).
    - **Purpose:** Store frequently accessed data and instructions.
    - **Levels:** L1 (closest to the CPU), L2, L3 (further away but larger).

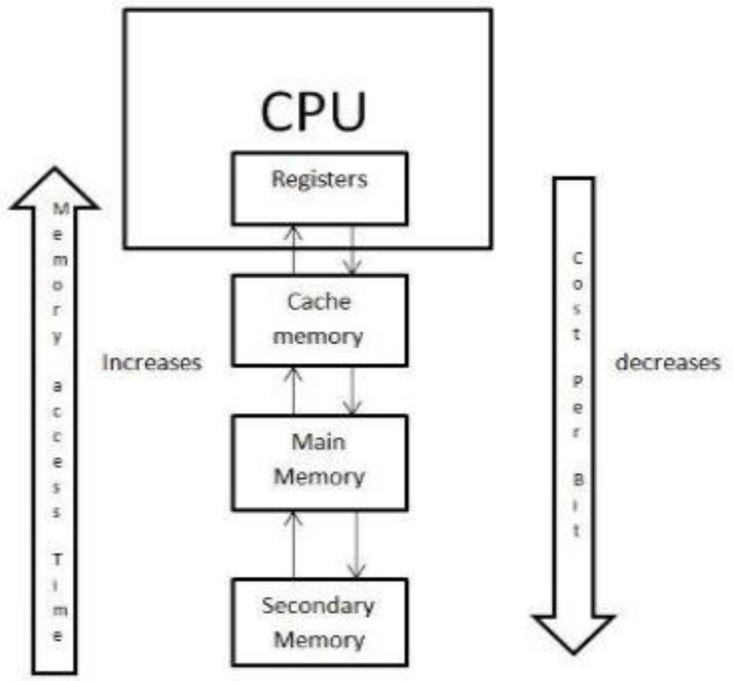


Figure 1

3. **Main Memory (RAM):**
    - **Location:** Separate from the CPU, accessed via a memory controller.
    - **Speed:** Slower than cache memory.
    - **Size:** Larger than cache (gigabytes range).
    - **Purpose:** Store active programs and data being processed.
    - **Types:** DRAM (dynamic RAM) and SRAM (static RAM).
4. **Secondary Storage:**
    - **Location:** External to the main system memory.
    - **Speed:** Slowest among the hierarchy.
    - **Size:** Largest capacity (terabytes or more).
    - **Purpose:** Long-term data storage.
    - **Examples:** Hard disk drives (HDD), solid-state drives (SSD), optical disks.

2.1.2 Performance Factors

1. **Latency:**
    - The delay between the request for data and its delivery.
    - **Lower latency** indicates faster access times.
2. **Bandwidth:**
    - The amount of data that can be transferred in a given time period.
    - Higher bandwidth allows for more data transfer.
3. **Access Time:**
    - The total time required to retrieve data from memory.
    - Includes latency and transfer time.

2.1.3 Comparison of Performance Across Levels:

Memory Level	Latency (ns)	Bandwidth (GB/s)	Access Time
Registers	~1	100+	Fastest
Cache (L1)	~1-3	50-100	Very Fast
Main Memory	~50-100	10-25	Moderate
Secondary Storage	~10,000-100,000	0.5-5	Slowest

2.1.4 Trade-offs: Cost vs. Speed vs. Capacity

- **Registers:** Fastest but very costly and limited in capacity.
- **Cache Memory:** High speed and moderate cost; provides a balance for frequently accessed data.
- **Main Memory (RAM):** Larger capacity but slower and less expensive than cache.
- **Secondary Storage:** Largest and cheapest but significantly slower.

Key Trade-offs:

- **Cost:** Increases as memory speed increases.
- **Capacity:** Decreases as memory speed increases.

- **Speed:** Higher levels in the hierarchy are faster.

2.1.4 Practical Examples

1. **Intel Core i9 Processors:**
   - Multi-level cache (L1, L2, L3) to enhance CPU performance.
   - DDR4/DDR5 RAM for main memory, ensuring high-speed data access.
   - NVMe SSDs as secondary storage for faster boot times and file access.
2. **AMD Ryzen Processors:**
   - Incorporate large L3 caches ("Game Cache") to optimize gaming and workload performance.
   - Support for high-speed RAM (e.g., DDR5) and compatibility with fast SSDs.
3. **Data Centers:**
   - Use tiered storage solutions with DRAM for active workloads and SSD/HDD for archival data.
   - Example: Google and Amazon optimize data access with memory hierarchies to improve cloud service performance.

2.2 Memory Capacity Planning

Memory capacity planning is the process of forecasting, analyzing, and allocating adequate memory resources to ensure optimal performance, reliability, and scalability of a system or application. It involves evaluating current workloads, anticipating future needs, and implementing cost-effective solutions. Following can be enlisted as importance of memory capacity planning:

- **Performance Optimization**: Prevents memory bottlenecks and enhances overall system efficiency.
- **Cost Management**: Balances resource allocation to avoid over-provisioning or underutilization.
- **Scalability**: Prepares the system for growth and ensures it can handle increased workloads.
- **Reliability**: Reduces risks of crashes, slowdowns, and service disruptions.

2.2.1 Factors Affecting Capacity can be as follows:

a. Workload Requirements

- **Application Type**: Memory demands vary for databases, web servers, AI/ML models, etc.
- **Concurrency Levels**: Number of simultaneous processes or users impacts memory needs.
- **Usage Patterns**: Peak vs. average usage trends influence memory allocation decisions.

b. System Architecture

- **Hardware Configuration**: CPU-memory bandwidth, storage hierarchy, and network constraints.
- **Software Optimization**: Inefficient code or memory leaks can inflate memory requirements.
- **Operating System**: Memory management techniques (e.g., virtual memory, paging).

c. Future Scalability

- **Growth Projections**: Anticipating increases in user base or data volume.
- **New Features**: Integration of new functionalities may demand more memory.
- **Elasticity**: Systems should adapt to dynamic workload variations.

2.2.2 Memory Planning Methods:

a. Advanced Workload Profiling and Analysis

Workload profiling involves a deep analysis of current and historical usage patterns to understand how memory is utilized by applications and processes. By employing tools like Task Manager, htop, or Prometheus, system administrators can capture real-time data on memory usage. Analyzing peak load times and identifying critical processes helps pinpoint bottlenecks and optimize resource allocation. Profiling also includes simulating worst-case scenarios to ensure that systems remain stable under extreme conditions.

b. Predictive Modeling

Predictive modeling applies data analytics and machine learning to forecast future memory demands based on historical performance data. By training predictive algorithms on workload metrics, systems can anticipate spikes and plan accordingly. For example, a model might predict seasonal increases in resource requirements for e-commerce platforms. Predictive modeling reduces reactive adjustments and enables a proactive approach to capacity planning, leading to smoother operations.

c. Resource Pooling and Dynamic Allocation

Resource pooling leverages shared memory resources in virtualized or cloud-based environments to handle fluctuating workloads efficiently. Dynamic allocation tools, such as Kubernetes for containerized applications, enable systems to adjust memory resources automatically. For instance, during high-demand periods, auto-scaling can allocate additional memory to maintain performance, while during off-peak hours, unused resources can be reclaimed to minimize costs. This approach ensures optimal utilization and elasticity.

d. Benchmarking and Stress Testing

Benchmarking establishes performance baselines by simulating typical and peak workloads using tools like SPEC Benchmarks or Stress-ng. Stress testing further evaluates system stability by applying extreme memory loads, identifying thresholds where performance degradation occurs. These methods are critical for understanding the limits of existing systems and ensuring they can handle anticipated demands. Benchmarking also provides data for comparing hardware and software configurations to optimize resource usage.

e. Capacity Simulation and Planning Tools

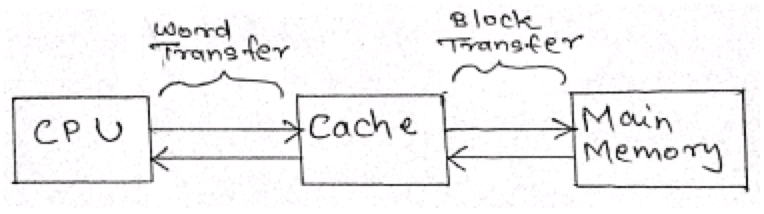
Capacity planning tools simulate different workload scenarios to evaluate system performance and forecast resource needs. Popular solutions, such as VMware vRealize Operations, AWS Trusted Advisor, and Grafana, provide insights into memory utilization trends and recommend optimizations. These tools enable organizations to model future workloads, test various configurations, and plan memory capacity with precision. By integrating simulation results into decision-making, businesses can achieve cost-effective scalability and reliability.

2.3 Cache Memories and Management

Cache memory is a small, high-speed storage layer situated between the CPU and main memory, designed to store frequently accessed data and instructions. It serves as a buffer to reduce the time needed to fetch data from slower memory levels. The purpose of cache memory can be enlisted as follows:

- Minimize latency in data access.
- Enhance CPU efficiency by reducing memory bottlenecks.
- Improve overall system performance.

When CPU attempts to read a word from main memory, check is made to determine if the word is in cache. It so, then word is delivered form cache. If word is not there in cache then a block of main memory consisting some word along with that word, is read into cache and the required word is delivered to CPU. This is called "Principle of Locality of Reference". During a miss if there are no empty blocks in the cache, then some replacement policies such as FIFO, LRU, LFU, etc. are used.



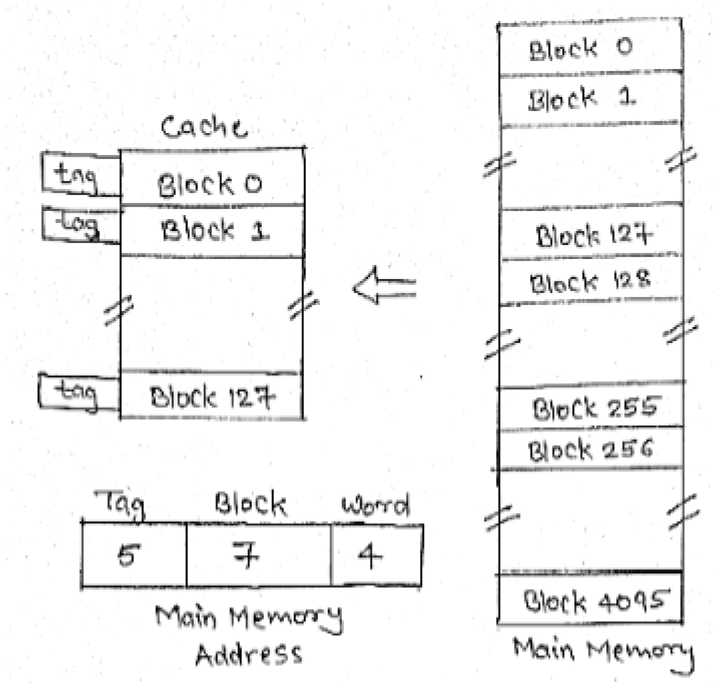
When a block of data is retrieved from main memory and put into the cache, the desired word and a number of adjacent words are retrieved. As the block size increases from a very small size, the hit ratio will at first increase due to the principle of locality of reference, which says that words in the vicinity of a referenced word are more likely to be referenced in the near future. As the block size increases, however, the hit ratio will decrease as the probability of reusing the new information becomes less than that of using the information that has been replaced.

2.3.1 Cache organization techniques :

The different Cache mapping technique are as follows:-
1) Direct Mapping
2) Associative Mapping
3) Set Associative Mapping

Consider a cache consisting of 128 blocks of 16 words each, for total of 2048(2K) words and assume that the main memory is addressable by 16-bit address. Main memory is 64K which will be viewed as 4K blocks of 16 words each.

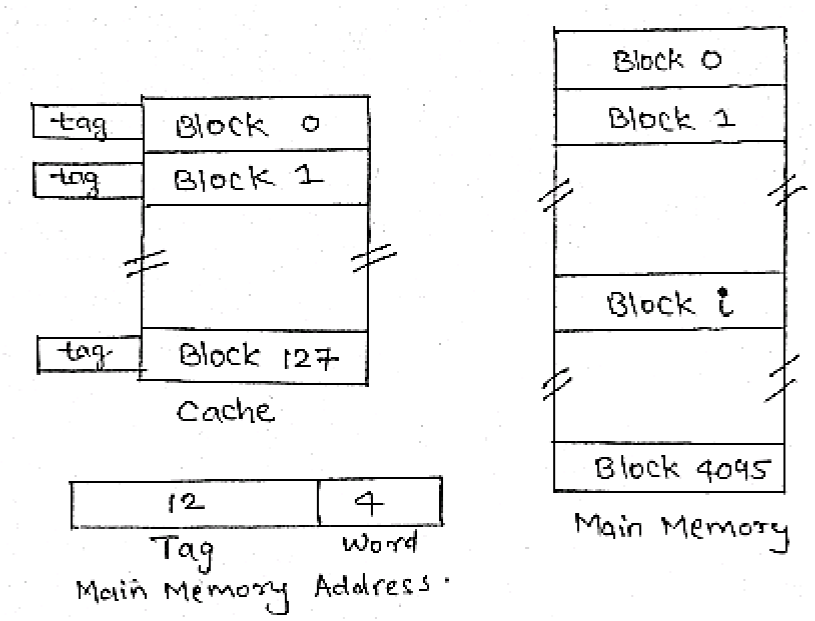
a. Direct Mapping**: -**



1) The simplest way to determine cache locations in which store Memory blocks is direct Mapping technique.
2) In this block J of the main memory maps on to block J modulo 128 of the cache. Thus main memory blocks 0,128,256,….is loaded into cache is stored at block 0. Block 1,129,257,….are stored at block 1 and so on.
3) Placement of a block in the cache is determined from memory address. Memory address is divided into 3 fields, the lower 4-bits selects one of the 16 words in a block.
4) When new block enters the cache, the 7-bit cache block field determines the cache positions in which this block must be stored.
5) The higher order 5-bits of the memory address of the block are stored in 5 tag bits associated with its location in cache. They identify which of the 32 blocks that are mapped into this cache position are currently resident in the cache.
6) It is easy to implement, but not Flexible

- **Advantages**:
  - Simple and easy to implement.
  - Low hardware cost.
- **Disadvantages**:
  - High conflict misses when multiple blocks map to the same line (e.g., blocks 4, 12, and 20).

b. **Associative Mapping: -**



1) This is more flexible mapping method, in which main memory block can be placed into any cache block position.
2) In this, 12 tag bits are required to identify a memory block when it is resident in the cache.
3) The tag bits of an address recevied from the processor are compared to the tag bits of each block of the cache to see, if the desired block is present. This is known as Associative Mapping technique.

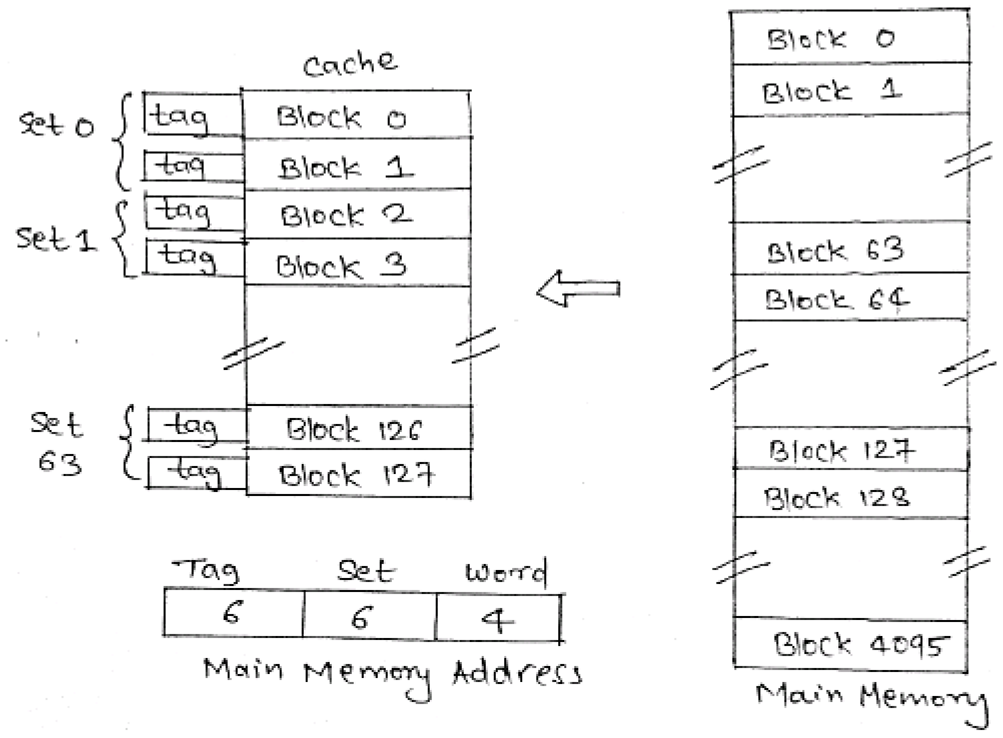
4) Cost of an associated mapped cache is higher than the cost of direct-mapped because of the need to search all 128 tag patterns to determine whether a block is in cache. This is known as associative search.

- **Advantages**:
  - Reduces conflict misses, as there are no restrictions on mapping.
- **Disadvantages**:
  - Higher hardware complexity due to the need for a fully associative search.
  - Slower lookup times compared to direct mapping.
- **Implementation**:
  - Uses content-addressable memory (CAM) to search for memory block tags.

c. **Set-Associated Mapping: -**

1) It is the combination of direct and associative mapping technique.
2) Cache blocks are grouped into sets and mapping allow block of main memory reside into any block of a specific set. Hence contention problem of direct mapping is eased , at the same time , hardware cost is reduced by decreasing the size of associative search.
3) For a cache with two blocks per set. In this case, memory block 0, 64, 128,…..,4032 map into cache set 0 and they can occupy any two block within this set.
4) Having 64 sets means that the 6 bit set field of the address determines which set of the cache might contain the desired block. The tag bits of address must be associatively compared to the tags of the two blocks of the set to check if desired block is present. This is two way associative search.

- **Advantages**:
  - Balances simplicity and performance.
  - Reduces conflict misses compared to direct mapping.
- **Disadvantages**:
  - Moderate hardware complexity.
  - Conflict misses can still occur within a set.



2.3.2 Performance Metrics

Performance of cache is largely measured by how effective the cache is at reducing the number of slow memory accesses.

Cache Hit Ratio

The **cache hit ratio** is a measure of how often requested data is found in the cache. It is calculated using the formula:

$$\text{Cache Hit Ratio} = \frac{\text{Number of Cache Hits}}{\text{Total Number of Memory Accesses}}$$

- **Cache Hit**: Occurs when the requested data is found in the cache.
- **Cache Miss**: Occurs when the requested data is not found in the cache, and must be fetched from slower storage.

A higher cache hit ratio is desirable as it reduces the latency in accessing data.

Miss Ratio

The **miss ratio** is the complement of the cache hit ratio and represents how often requested data is **not** found in the cache. It is calculated as:

$$\text{Miss Ratio} = 1 - \text{Cache Hit Ratio}$$

The miss ratio directly affects performance because every cache miss leads to accessing slower memory or storage, which increases latency.

Impact on System Performance

- **High Hit Ratio**: A high cache hit ratio leads to faster data retrieval, improving system performance and responsiveness.
- **High Miss Ratio**: A high miss ratio increases the need to fetch data from slower sources, causing higher latency and poor performance.

To optimize system performance, caching strategies focus on increasing the hit ratio and reducing the miss ratio.

2.3.3 Management Techniques in Caching Systems

a. Prefetching

Prefetching is a technique used to load data into the cache before it is actually requested by the system. This can improve cache hit ratios by anticipating which data will be needed next.

- **Types of Prefetching**:
    - **Hardware Prefetching**: Done by the hardware (CPU or memory controller), which predicts memory access patterns and loads data into cache before it's requested.
    - **Software Prefetching**: Done by software applications or the operating system, where the programmer explicitly requests the prefetch of certain data.

While prefetching can reduce cache misses, excessive prefetching can lead to inefficient use of cache space if incorrect predictions are made.

b. Replacement Policies

Cache replacement policies determine how the cache replaces data when it is full and a new piece of data needs to be loaded. Common replacement policies include:

1. **Least Recently Used (LRU)**:
    - Evicts the least recently accessed data. This assumes that data that hasn't been used recently is less likely to be used in the near future.
2. **First-In-First-Out (FIFO)**:
    - Evicts the oldest data in the cache. This method is simple but may not always be optimal, as it does not account for how frequently or recently data was accessed.
3. **Least Frequently Used (LFU)**:

- Evicts the data that has been accessed the least number of times. This is useful if the assumption is that data with lower frequency of access is less likely to be needed soon.

4. **Random Replacement**:
   - Randomly selects data to evict when new data needs to be cached. This is often used in simple systems when other policies are too complex to implement.

Each policy has trade-offs in terms of simplicity, efficiency, and performance.

c. Write Strategies: Write-Through vs. Write-Back

The way data is written to the cache and the main memory can also affect system performance. The two primary write strategies are **write-through** and **write-back**.

1. **Write-Through**:
   - In a write-through cache, every write to the cache is immediately written to the main memory as well.
   - **Pros**: Ensures that the main memory is always consistent with the cache, making it easier to maintain data integrity.
   - **Cons**: Can be slower because every write operation requires both a cache update and a memory update, leading to higher latency.
2. **Write-Back**:
   - In a write-back cache, data is only written to the main memory when it is evicted from the cache, not immediately after each write.
   - **Pros**: Can improve performance because fewer writes are made to the main memory. This reduces latency and minimizes the load on the memory system.
   - **Cons**: The main memory may become inconsistent with the cache, creating potential data integrity issues if the system crashes before the write-back occurs.

When to Use Write-Through vs. Write-Back:

- **Write-Through** is typically used in systems where data integrity is critical, and the penalty of writing to memory on every cache update is acceptable (e.g., in systems with fast memory or low write intensity).
- **Write-Back** is used in performance-sensitive applications where reducing memory write operations is a priority (e.g., in systems with slower memory or where frequent writes occur).

2.4 Virtual Memory Technology

Virtual memory, simply put is when the hard disk is used to provide an extension to main memory. In essence, virtual memory is a facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available.

When virtual memory is used, the address fields of machine instructions contain virtual addresses. For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory. The advantage of using such a scheme is that it is possible for a process to be larger than all of main memory. One of the most fundamental restrictions in programming has been lifted.

The process of address translation is shown in Figure 2 which shows the Virtual Memory organization as well.

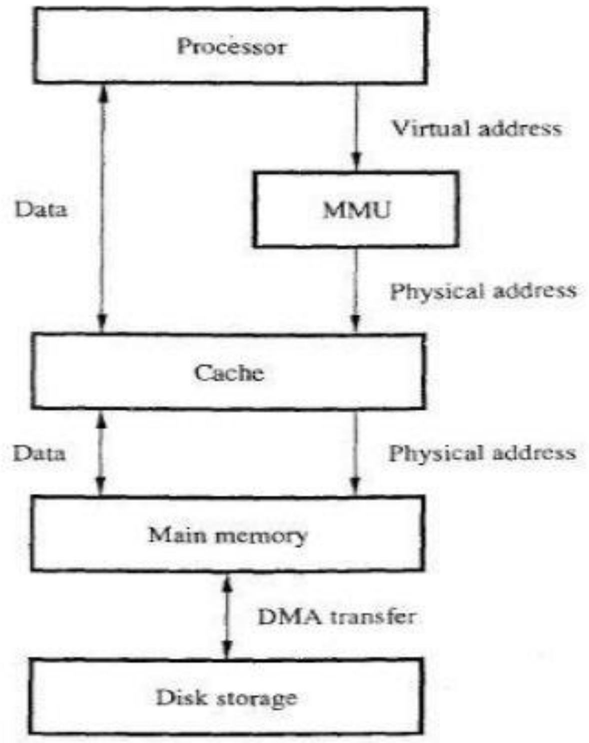


Figure 2

Without demand paging (this is the practice of getting pages from the secondary memory into the main memory as and when required. Programs are considered to be divided into **fixed length/fixed size** pages) and virtual memory, a programmer must be acutely aware of how much memory is available.

If the program being written is too large, the programmer must devise ways to structure the program into pieces that can be loaded one at a time. With virtual memory scheme and demand paging as specified above, that job is left to the OS and the hardware. As far as the programmer is concerned, he or she is dealing with a huge memory, the size associated with disk storage. Because a process executes only in main memory, that memory is referred to as real memory.

But a programmer or user perceives a much larger memory—that which is allocated on the disk. This latter is therefore referred to as **virtual memory**.

Virtual memory allows for very effective multiprogramming and relieves the user of the unnecessarily tight constraints of main memory.

1. When a program does not fit into the main memory, the parts of the program that are currently not being executed are stored on the hard disk and are transferred to main memory as and when required. Thus, simply put, Virtual memory is just the extension of the main memory in the disk where parts of very big programs that are currently being executed but cannot be stored completely in the main memory itself are stored.
2. This shows that the CPU no longer generates the physical address directly. This is translated to the physical address by the MMU which is then used to address the main memory as shown in Figure 6:

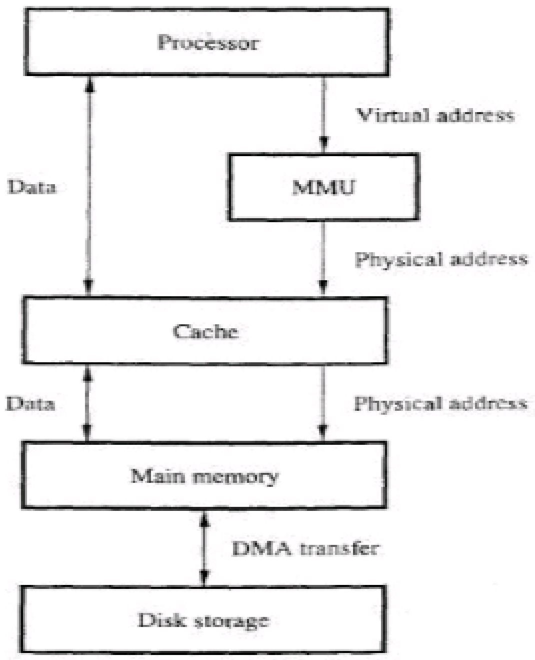


Figure 6

a. Paging:

1. Programs getting divided into fixed size partitions or pieces named pages. This led to the development of the important concept of virtual memory. With the use of paging, truly

effective multiprogramming systems came into being. Virtual Memory uses **demand paging**, which simply means that each page of a process is brought in only when it is needed, that is, on demand. It is clear that we can make better use of memory by loading in just a few pages.

2. Then, if the program branches to an instruction on a page not in main memory, or if the program references data on a page not in memory, a **page fault** is triggered. This tells the OS to bring in the desired page.

3. The basic mechanism for reading a word from memory involves the translation of a virtual, or logical, address, consisting of page number and offset, into a physical address, consisting of frame number and offset, using a page table. Because the page table is of variable length, depending on the size of the process, we cannot expect to hold it in registers. Instead, it must be in main memory to be accessed.

4. When a particular process is running, a register (Page Table Register) holds the starting address of the page table for that process. The page number of a virtual address is used to index that table and look up the corresponding frame number. This is combined with the offset portion of the virtual address to produce the desired real address. Physical address generation from virtual addresses is shown in Figure 7.

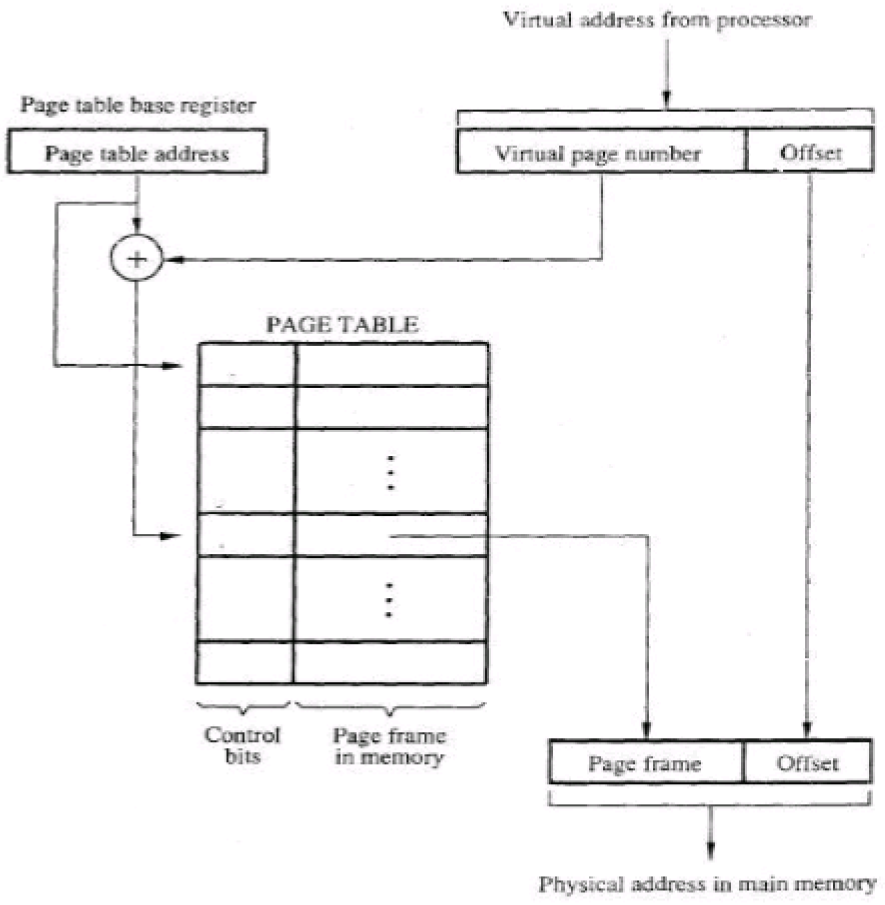


Figure 7

b. Segmentation:

- Paging was invisible to the programmer and serves the purpose of providing the programmer with a larger address space, segmentation is usually visible to the programmer and is provided as a convenience for organizing programs and data and as a means for associating privilege and protection attributes with instructions and data. Segments basically contain many pages.

- Thus Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments. Segments are of variable, indeed dynamic, size while pages were fixed size fragments. OS will assign programs and data to different segments. There may be a number of program segments for various types of programs as well as a number of data segments. Each segment may be assigned access and usage rights. Memory references consist of a (segment number, offset) form of address.

- The address translation mechanism when segmentation and paging is used with paging is shown in Figure 8.

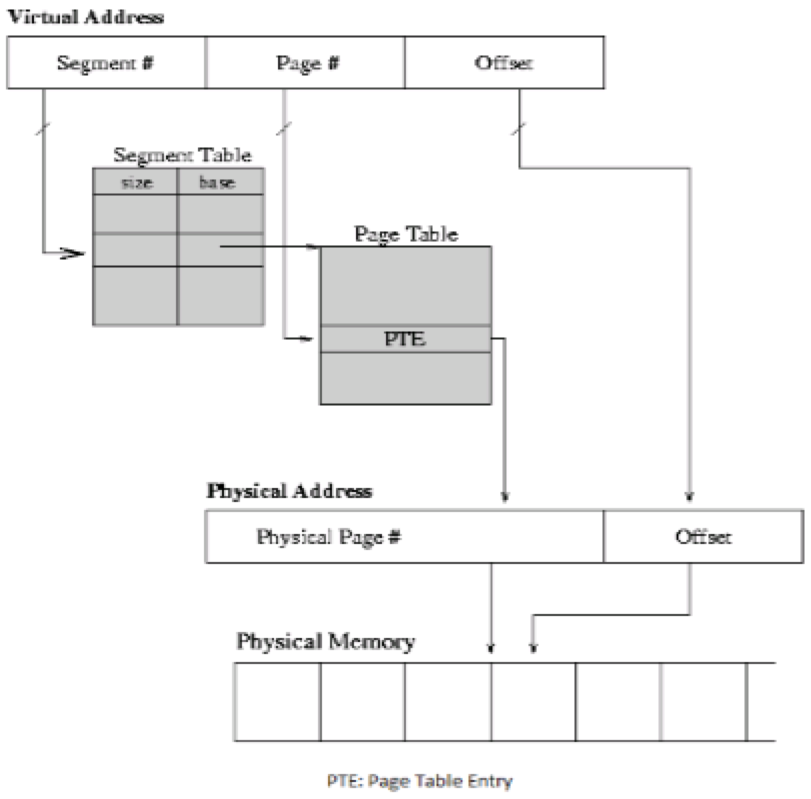


Figure 8

- The virtual address is now composed of three parts: The higher order bits give the segment number, the bits after that give the page number and the bits after that give the offset.
- Thus an extra Segment table is used. This along with the page number is used to index the page table.
- The process after that is similar to the address translation mechanism when paging was used.
- Segmentation is optional and can be disabled.

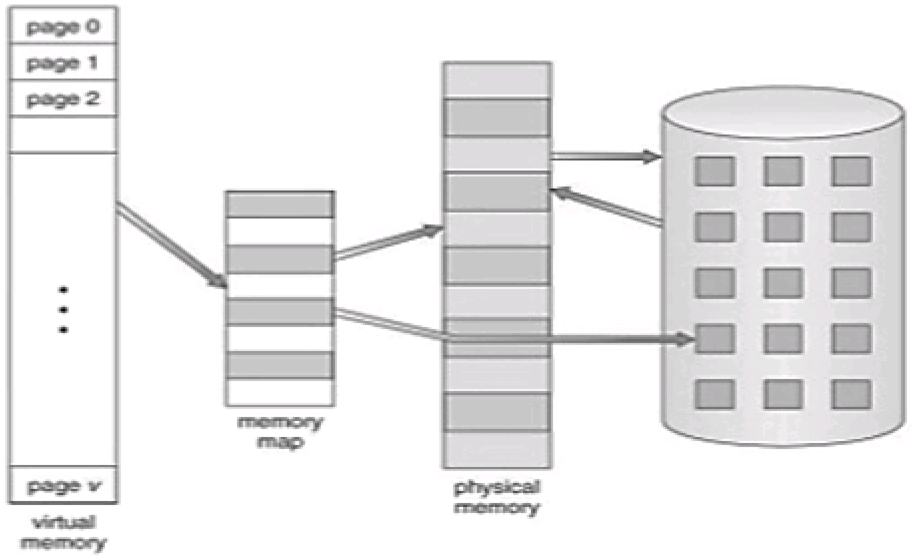
More on paging:

- Paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory.

- In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages.
- Paging allows the physical address space of the process to be non contiguous
- Paging is to deal with external fragmentation problem. This is to allow the logical address space of a process to be non-contiguous, which makes the process to be allocated physical memory.
- Logical address space of a process can be non-contiguous; process is allocated physical memory.

Whenever the latter is available.

- Divide physical memory into fixed-sized blocks called frames (size is power of 2, between 512 bytes and 8192 bytes).
- Divide logical memory into blocks of same size called pages.
- Keep track of all free frames.
- To run a program of size n pages, need to find n free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation-allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- The logical memory of the process is contiguous, but pages need not be allocated contiguously in memory.
- By dividing memory into fixed size pages, we can eliminate external fragmentation.
- Paging does not eliminate internal fragmentation



Address generated by CPU is divided into:

1. Page number (p) – used as an index into a page table which contains base address of each page in physical memory.

2. Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit.

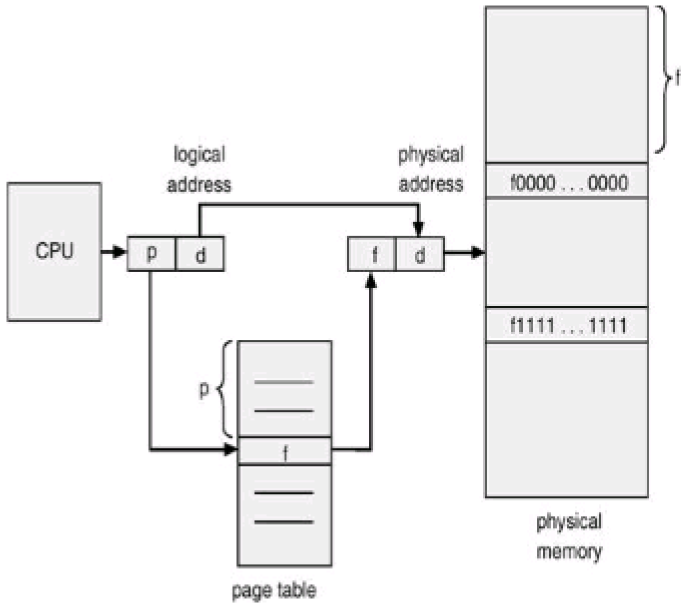
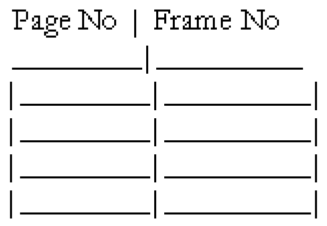


Fig. Address translation scheme

Implementation of page table:

- Page table is kept in main memory.
- Page-table base register (PTBR) points to the page table.
- Page-table length register (PTLR) indicates size of the page table.
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative registers or translation look-aside buffers (TLBs).
- Associative registers - parallel search



Page No	Frame No

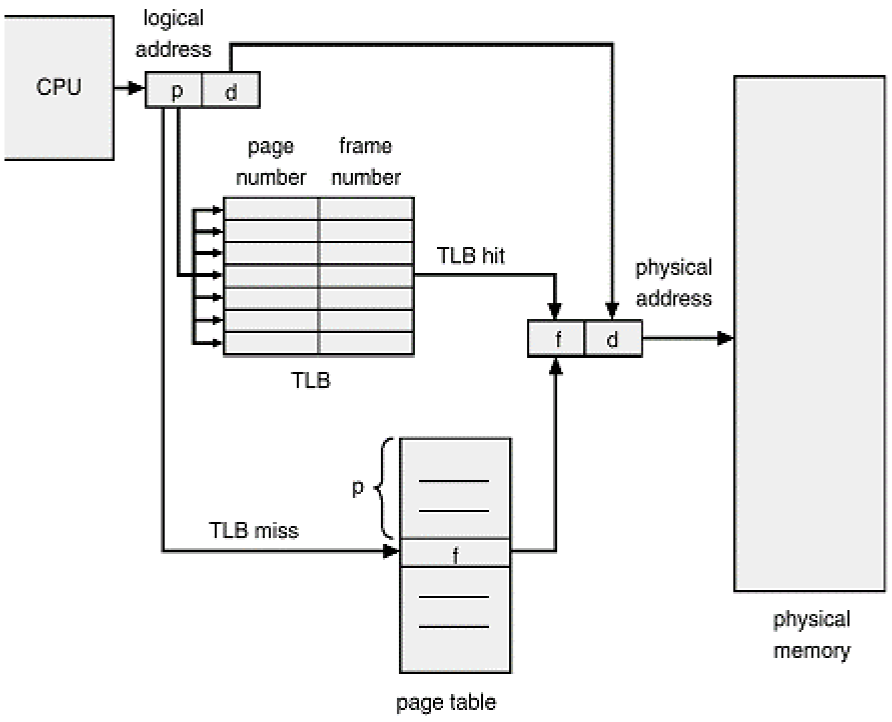
Address translation (A', A")

- If A' in associative register, get frame number out.
- Otherwise get frame number from page table in memory.
    - Hit ratio - percentage of times that a page number is found in the associative registers; ratio related to number of associative registers.
    - Effective Access Time (EAT)
    - associative lookup = e time units
    - memory cycle time = m time units
    - hit ratio = a

EAT = $(m + e) a + (2m + e) (1 - a) = 2m + e - ma$

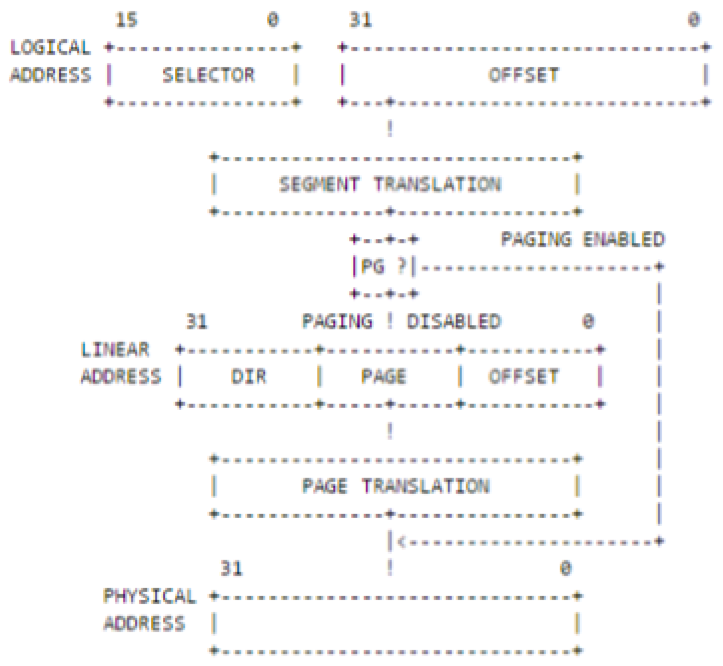
- Memory protection implemented by associating protection bits with each frame.
- Valid-invalid bit attached to each entry in the page table:

  - ``valid'' indicates that the associated page is in the process' logical address space, and is thus a legal page.
  - ``invalid'' indicates that the page is not in the process' logical address space.
- Write bit attached to each entry in the page table.
  - Pages which have not been written may be shared between processes
  - Do not need to be swapped - can be reloaded.



Conversion of logical address to physical address

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
  - Logical address - generated by the CPU; also referred to as virtual address.
  - Physical address - address seen by the memory unit.
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

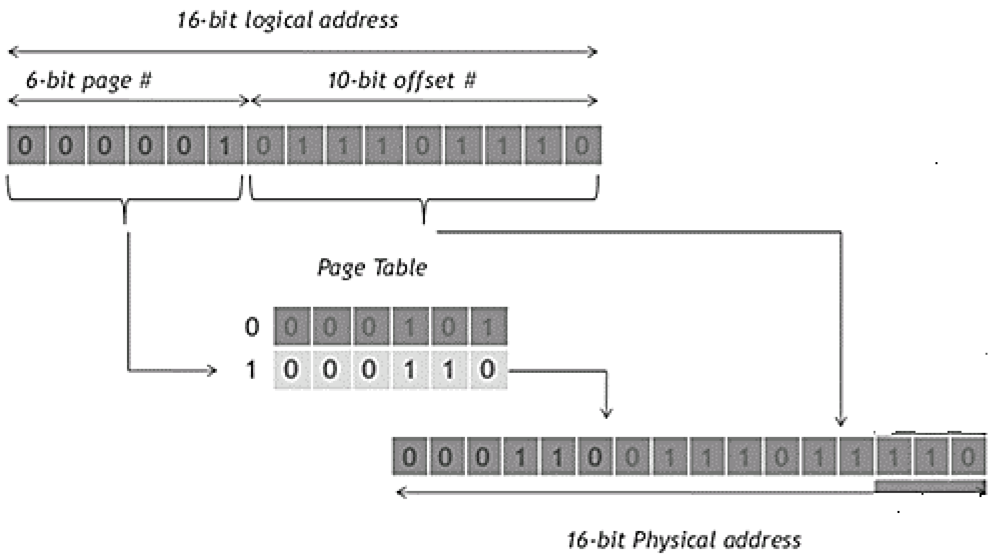


There are three steps to translate logical address to physical address

Step1: Find the index field from the segment selector and use the index field to locate the segment descriptor for the segment in global descriptor table (GDT)

Step 2: Test the access and limit the field of descriptor to make sure that the segment is accessible and the offset is within the limit of the segment

Step3: The base address of the segment will be obtained from the segment descriptor. Then the base address of the segment will be added to the offset to determine a linear address



2.5 Page Replacement in Virtual Memory Systems

Virtual memory allows an operating system to extend the available memory beyond the physical limits of RAM by using secondary storage (like a hard disk or SSD) to store parts of programs and data that are not currently in use.

The page replacement is done by swapping. The required pages from backup storage to main memory and vice-versa. This swapping is done by checking the contents of physical memory. If there is a free frame in the memory then swap-in the required page into the frame which is free. In case, if there is no frame in physical memory then first find the frame which is not currently in use. The content of this frame is swapped-out from memory to backup storage. Then bring the required page in the frame which is now free.

Only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in.

Following steps can be taken to deal with this problem:

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free Frames.

This technique is called Page replacement and is most commonly used.

Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults

Importance of Page Replacement:

- **Minimizes page faults**: By making intelligent decisions about which pages to replace, the system can reduce the number of expensive page faults.
- **Optimizes memory utilization**: Efficient page replacement ensures that frequently used data remains in memory, improving overall system performance.

To achieve these goals, various page replacement algorithms have been developed, each with its own approach and trade-offs.

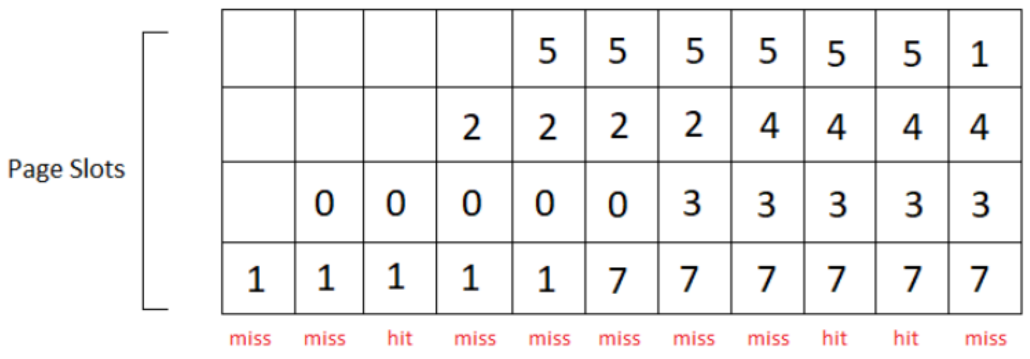
Algorithm Analysis

2.5.1 FIFO (First-In, First-Out) Algorithm

- The **FIFO** algorithm is one of the simplest page replacement strategies. The basic idea is to replace the page that has been in memory the longest, regardless of how frequently or recently it has been used.
- It uses a **queue** to keep track of the order in which pages were loaded into memory. When a page fault occurs, the algorithm evicts the page at the front of the queue (i.e., the oldest page), and inserts the new page at the back.

Example

Let the page reference string be {1, 0, 1, 2, 5, 7, 3, 4, 3, 4, 1} and there are total 4 page slots. Then,



					5	5	5	5	5	5	1
Page Slots				2	2	2	2	4	4	4	4
		0	0	0	0	0	3	3	3	3	3
	1	1	1	1	1	7	7	7	7	7	7
	miss	miss	hit	miss	miss	miss	miss	miss	hit	hit	miss

Page Faults = 8

Advantages:

- **Simple to implement**: FIFO is easy to understand and implement, making it a popular choice for simple systems.
- **Efficient for small systems**: In scenarios where memory demands are not high, FIFO can work reasonably well.

Limitations:

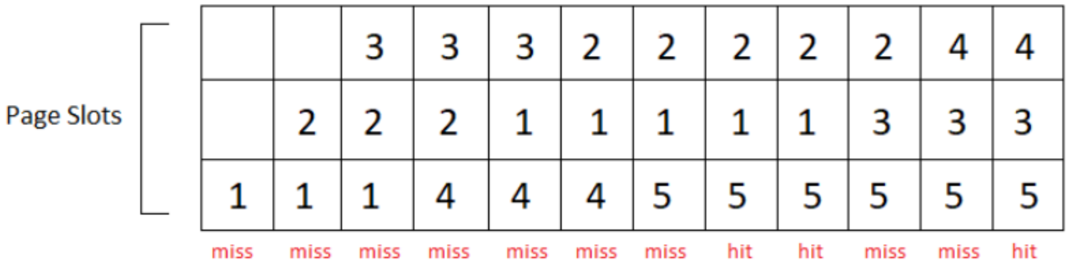
- **Belady's Anomaly**: One of the main disadvantages of FIFO is that it can sometimes perform worse as the number of page frames increases. This phenomenon, called **Belady's Anomaly**, shows that increasing the number of frames can actually increase the number of page faults in FIFO.
- **Suboptimal Performance**: FIFO does not take into account the frequency or recency of page accesses, which can lead to suboptimal performance in many scenarios. Pages that are frequently used or will be used soon might be evicted in favor of less useful pages.

2.5.2 Least Recently Used (LRU) Algorithm

- The **LRU** algorithm aims to keep the most recently used pages in memory. When a page fault occurs, LRU replaces the page that has not been accessed for the longest period of time.
- LRU assumes that pages that were used recently will likely be used again soon, so it prioritizes keeping these pages in memory.

Example

Let the page reference string be {1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4} and there are total 3 page slots. Then,



Page Slots			3	3	3	2	2	2	2	2	4	4
		2	2	2	1	1	1	1	1	3	3	3
	1	1	1	4	4	4	5	5	5	5	5	5
	miss	miss	miss	miss	miss	miss	miss	hit	hit	miss	miss	hit

Page Faults = 9

Strengths:

- **Better than FIFO**: LRU generally performs better than FIFO in many scenarios, as it takes recency into account. It ensures that pages that have been recently used are less likely to be replaced.
- **More Efficient in Real-World Usage**: It is often closer to optimal compared to FIFO, especially when memory access patterns are localized (i.e., processes tend to reuse pages over time).

Limitations:

- **Overhead**: Maintaining the stack or counters can be resource-intensive, especially with a large number of pages or high-frequency accesses.
- **Implementation Complexity**: Although more efficient than FIFO, LRU can be harder to implement effectively in hardware and software, especially in systems with limited resources.

2.5.3 Most Frequently Used (MFU) Algorithm

- The **MFU** algorithm takes a different approach compared to FIFO and LRU. It replaces the page that has been **most frequently accessed** in the past. If more than one page is accessed the same number of times, then the page which occupied the frame first will be replaced.
- The reasoning behind this approach is that a frequently accessed page might be less useful in the future because it has already been used extensively.

Example:

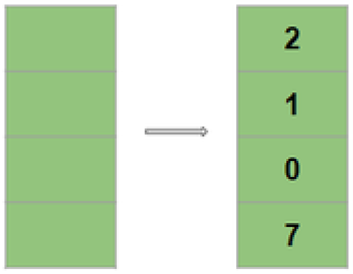
Consider a Main Memory with a number of frames = 4 and the following are the data block access requests made by the CPU.

CPU Requests - 7, 0, 1 , 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1

Given, the number of frames = 4. Initially, all are empty

(i) The 4 frames are initially empty, the first 4 blocks occupy them.

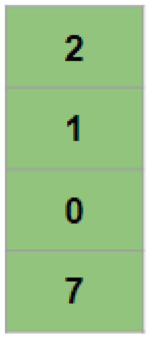
~~7~~, ~~0~~, ~~1~~, ~~2~~, 0 , 3 , 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1



First four blocks occupy the 4 frames

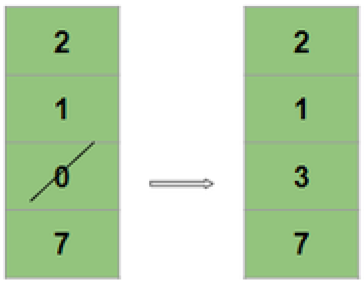
(ii) Block 0 has already occupied the frame.

~~7~~, ~~0~~, ~~1~~, ~~2~~, <u>0</u> , 3 , 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1



(iii) Block 0 is accessed maximum times, so replaced with incoming Block 3.

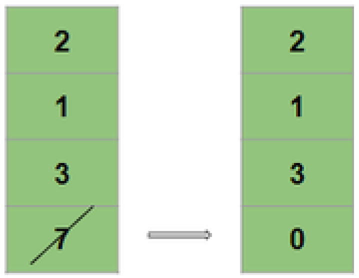
~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, 3 , 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1



Block 0 replaced with Block 3

(iv) Blocks 2,1,3,7 are accessed once and Block 7 occupied the frame first, so, Block 7 replaced with incoming Block 0.

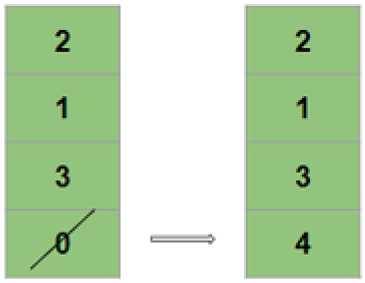
~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, 0 , 4 , 2 , 3 , 0 , 3 , 2 , 1



Block 7 replaced with Block 0

(v) Block 0 is accessed maximum times so replaced with incoming Block 4.

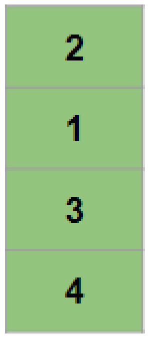
~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, ~~0~~, 4 , 2 , 3 , 0 , 3 , 2 , 1



Block 0 replaced with Block 4

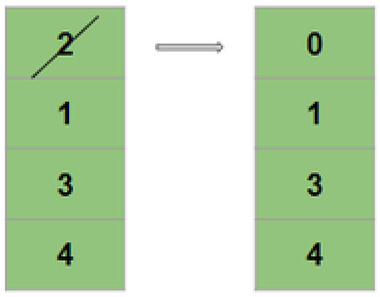
(vi) Blocks 2,3 have already occupied the frame.

~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, ~~0~~, ~~4~~, 2 , 3 , 0 , 3 , 2 , 1



(vii) Block 2 and Block 3 are accessed maximum times and Block 2 occupied the frame first, so, Block 2 replaced with incoming Block 0.

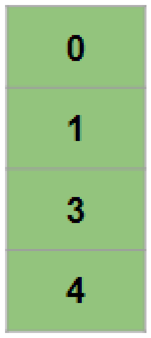
~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, ~~0~~, ~~4~~, ~~2~~, ~~3~~, 0 , 3 , 2 , 1



Block 2 replaced with Block 0

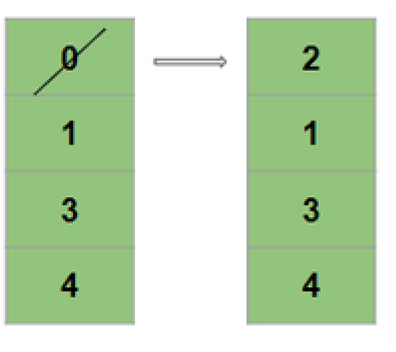
(viii) Block 3 has already occupied the frame.

~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, ~~0~~, ~~4~~, ~~2~~, ~~3~~, ~~0~~, 3 , 2 , 1



Block 0 is accessed maximum times so replaced with incoming Block 2.

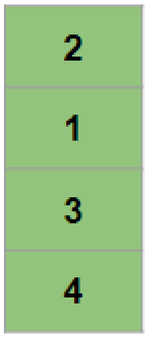
~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, ~~0~~, ~~4~~, ~~2~~, ~~3~~, ~~0~~, ~~3~~, 2 , 1



Block 0 replaced with Block 2

(i) Block 1 has already occupied the frame.

~~7~~, ~~0~~, ~~1~~, ~~2~~, ~~0~~, ~~3~~, ~~0~~, ~~4~~, ~~2~~, ~~3~~, ~~0~~, ~~3~~, ~~2~~, 1



Final Image

Comparison with LRU and FIFO:

- **FIFO**: FIFO doesn't consider how often or how recently pages are accessed, which can lead to inefficient use of memory. MFU, by contrast, prioritizes replacing frequently used pages.
- **LRU**: LRU focuses on recency of access, while MFU focuses on frequency. LRU is generally more effective in scenarios where localized access patterns are present, while MFU may be more effective in systems where certain pages are highly repetitive.

Limitations:

- **Counterproductive in Some Scenarios**: The assumption that frequently used pages will not be needed again soon may not hold in all cases, making MFU suboptimal in some situations.
- **Complexity**: Like LRU, MFU requires tracking the frequency of accesses for all pages, which can be resource-intensive.

Hands on Numerical

1) What is the maximum size of the address space generated in a processor with has 32 bit address?

2) a) How many 128 x 8 RAM chips are needed to provide a memory capacity of 2048 bytes?
b) How many lines of the address bus must be used to access 2048 bytes of memory?
c) How many lines must be decoded for chip select? Specify the size of the decoders.
d.1) if a computer uses RAM chips of 1024 x 1 capacity, how many chips are needed to provide a memory capacity of 1024 bytes?
d.2) How many chips are needed to provide a memory capacity of 16K bytes?

3) An application program in a computer with cache uses 1400 instruction fetch from cache and 100 from main memory. What is hit ratio?

4) Consider a two level cache with access time 5 nsec and 80 nsec respectively. If the hit ratio are 95% and 75% respectively in the two caches and main memory access time is 250 nsec. What is the effective access time? (effective access time =H1T1+H2(1-H1)T2+…….

5) Let's say that we have two levels of cache, backed by DRAM:
- L1 cache costs 1 cycle to access and has miss rate of 10%
- L2 cache costs 10 cycles to access and has miss rate of 2%
- DRAM costs 80 cycles to access (and has miss rate of 0%)
Then what would be average memory access time (AMAT)?

6. A computer has a single cache (off-chip) with a 2 ns hit time and a 98% hit rate. Main memory has a 40 ns access time. What is the computer's effective access time? If we add an on-chip cache with a .5 ns hit time and a 94% hit rate, what is the computer's effective access time? How much of a speedup does the on-chip cache give the computer?

7. Assume a computer has on-chip and off-chip caches, main memory and virtual memory. Assume the following hit rates and access times: on-chip cache 95%, 1 ns, off-chip cache 99%, 10 ns, main memory: X%, 50 ns, virtual memory: 100%, 2,500,000 ns. Notice that the on-chip access time is 1 ns. We do now want our effective access time to increase much beyond 1 ns. Assume that an acceptance effective access time is 1.6 ns. What should X be (the percentage of page faults) to ensure that EAT is no worse than 1.6 ns?

8. A computer has an 8 GByte memory with 64 bit word sizes. Each block of memory stores 16 words. The computer has a direct-mapped cache of 128 blocks. The computer uses word level addressing. What is the address format? If we change the cache to a 4- way set associative cache, what is the new address format?

9. Assume a computer has 32 bit addresses. Each block stores 16 words. A direct-mapped cache has 256 blocks. In which block (line) of the cache would we look for each of the following addresses? Addresses are given in hexadecimal for convenience.

   a. 1A2BC012
   b. FFFF00FF
   c. 12345678
   d. C109D532

Answers: Of the 32 bit address, the last four bits denote the word on the line. Since four bits is used for one hex digit, the last digit of the address is the word on the line. With 256 blocks in the cache, we need 8 bits to denote the block number. This would be the third to last and second to last hex digit.

a. this would be block 01, which is block 1
b. this would be 0F which is block 15

c. this would be 67 which is block 103 (remember, 67 is a hex value)

d. this would be 53 which is block 83.

10. Assume a program consists of 8 pages and a computer has 16 frames of memory. A page consists of 4096 words and memory is word addressable. Currently, page 0 is in frame 2, page 4 is in frame 15, page 6 is in frame 5 and page 7 is in frame 9. No other pages are in memory. Translate the memory addresses below.

    a. 111000011110000

    b. 000000000000000

11. You have been asked to design a cache with the following properties:
- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The cache is direct mapped
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache
- Addresses are to the word

12. Now, let's consider what happens if we make our cache 2-way set-associative instead of direct mapped. However, as before, the following still applies:
- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache
- Addresses are to the word

14. Now, let's consider what happens if we make our cache 4-way set-associative instead of direct mapped. However, as before, the following still applies:
- Data words are 32 bits each
- A cache block will contain 2048 bits of data
- The address supplied from the CPU is 32 bits long
- There are 2048 blocks in the cache
- Addresses are to the word

15. Find the average memory access time for a processor given the following:
- The clock rate is 1 ns
- The miss penalty is 25 clock cycles
- 1% of instructions are not found in the cache.

- 5% of data references are not found in the cache
- 15% of memory accesses are for data.
- The memory system has a cache access time (including hit detection) of 1 clock cycle.
- Assume that the read and write miss penalties are the same and ignore other write stalls.

16. A computer system uses 16-bit memory addresses. It has a 2K-byte cache organized in a direct-mapped manner with 64 bytes per cache block. Assume that the size of each memory word is 1 byte.

(a) Calculate the number of bits in each of the Tag, Block, and Word fields of the memory address.
(b) When a program is executed, the processor reads data sequentially from the following word addresses:
128, 144, 2176, 2180, 128, 2176
All the above addresses are shown in decimal values. Assume that the cache is initially empty. For each of the above addresses, indicate whether the cache access will result in a hit or a miss.

17. Repeat Problem # 1, if the cache is organized as a 2-way set-associative cache that uses the LRU replacement algorithm.

18. Given page reference string: {1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6 } with total 4 page slots. Compare the number of page faults for LRU, FIFO and MFU page replacement algorithm

19. Consider the reference string 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 2, 0 for a memory with three frames and calculate number of page faults by using Least Recently Used (LRU), FIFO,MFU Page replacement algorithms.

20. Repeat above problem 19 with 5 frames.

Discussion Questions:

1. Why do we need a hierarchical memory structure in computer systems?
2. How does latency impact overall system performance?
3. Compare the cost-to-performance ratio of cache memory and main memory.
4. Analyze the memory hierarchy in your personal computer or smartphone. Identify the components and discuss how they align with the concepts discussed.
5. How can predictive modeling improve memory capacity planning?
6. Compare the benefits of resource pooling versus static allocation.
7. What are the challenges of memory capacity planning in cloud environments?
8. Compare and contrast direct, associative, and set-associative cache mapping techniques.
9. How does the MESI protocol maintain cache coherence in a multi-core system?
10. What are the trade-offs between write-through and write-back strategies?
11. What is the purpose of organizing memory into a hierarchical structure?
12. How does the cost-speed-capacity trade-off influence the design of memory hierarchies?
13. What are the primary differences between register memory, cache memory, main memory, and secondary storage?
14. Can you explain the differences between L1, L2, and L3 caches in terms of performance and capacity?
15. How does the principle of locality of reference improve cache performance?
16. What are the trade-offs between write-through and write-back caching strategies?
17. In what scenarios is prefetching beneficial, and when can it become counterproductive?
18. What is the significance of cache hit and miss ratios in system performance?
19. How do FIFO, LRU, and LFU cache replacement policies differ in terms of their approaches and effectiveness?
20. What challenges do virtual memory systems address in modern computing?
21. How does demand paging enable efficient use of virtual memory?
22. What is the role of the memory management unit (MMU) in virtual memory systems?
23. Can you explain the process of address translation in paging systems?
24. What is the purpose of segmentation in memory management, and how does it complement paging?
25. What are the implications of internal and external fragmentation in memory management?
26. How do predictive modeling and workload profiling contribute to memory capacity planning?
27. What is the impact of poorly managed memory resources on system performance and reliability?
28. Why might Belady's Anomaly occur in FIFO page replacement strategies?

Analytical questions:

1. Discuss the implications of memory latency, bandwidth, and access time on the design of high-performance computing systems.

2. Evaluate the effectiveness of virtual memory in enabling multiprogramming and multitasking in modern operating systems.
3. Compare and contrast the benefits and limitations of direct, associative, and set-associative cache mapping techniques.
4. How would you design a predictive memory capacity planning model for a large-scale e-commerce platform like Amazon?
5. Critically evaluate the role of page replacement algorithms in minimizing page faults in virtual memory systems.
6. Discuss how advancements in NVMe SSD technology have reshaped the secondary storage layer in the memory hierarchy.
7. How do memory management schemes like segmentation and paging address the challenges of large, complex processes in modern applications?
8. How do modern processors integrate memory hierarchy designs (registers, caches, RAM) to achieve optimal performance for both sequential and parallel processing workloads?
9. How does the choice of cache block size and replacement policies affect the hit ratio and overall system efficiency in different types of workloads, such as gaming versus scientific computing?
10. Analyze the trade-offs between memory size, paging overhead, and application performance in virtual memory systems. How can these trade-offs influence system design decisions?
11. With emerging memory technologies such as HBM (High Bandwidth Memory) and persistent memory (e.g., Intel Optane), how might traditional memory hierarchies evolve to handle increasing demands for speed and capacity?
12. In cloud environments, how can advanced memory management techniques like resource pooling, dynamic allocation, and predictive scaling be combined to minimize latency and maximize cost-effectiveness for variable workloads?
13. With the emergence of persistent memory technologies like Intel Optane, how is the boundary between volatile and non-volatile memory shifting, and what are the implications for system architecture?
14. Evaluate the role of memory compression and deduplication technologies in optimizing the performance of virtualized environments and cloud-based systems. What trade-offs are involved?
15. Analyze the impact of 3D NAND technology on the cost, capacity, and reliability of secondary storage. How does it influence the design of modern memory hierarchies?