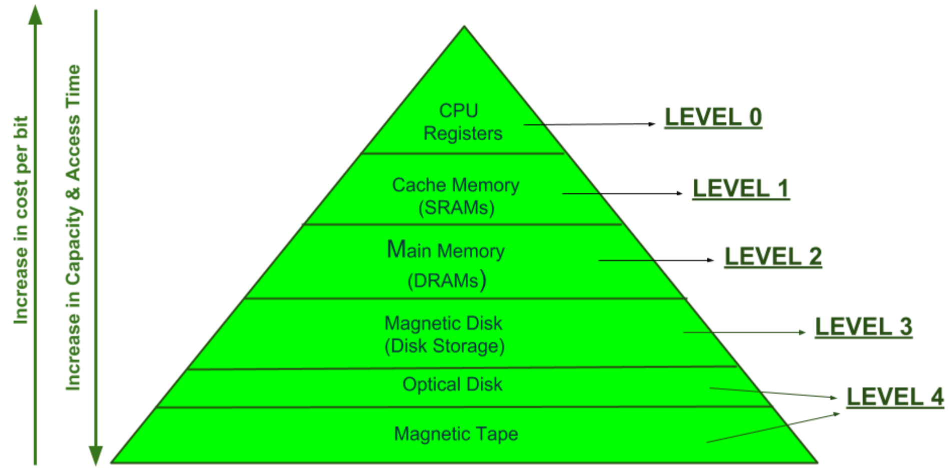
UNIT-2

Memory Hierarchy:

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behaviour known as locality of references. The figure below clearly demonstrates the different levels of memory hierarchy:



MEMORY HIERARCHY DESIGN

This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory –**
   Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
2. **Internal Memory or Primary Memory –**
   Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics of Memory Hierarchy Design from above figure:

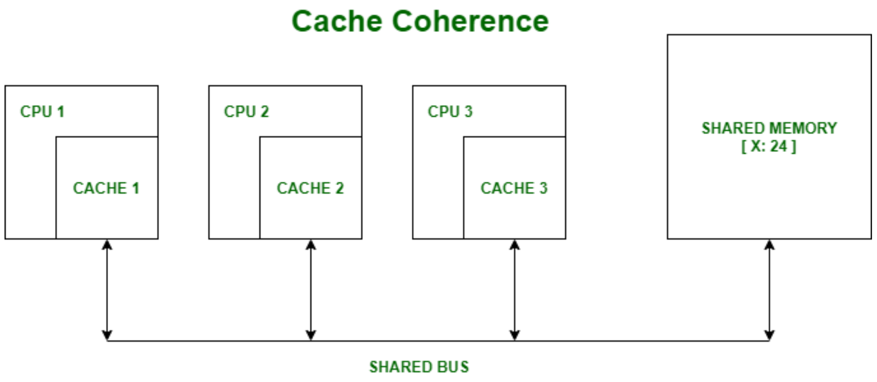
1. **Capacity:**
   It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
2. **Access Time:**
   It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
3. **Performance:**
   Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.
4. **Cost per bit:**
   As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

(Coherence and Locality Properties)

Coherence:

In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy. In a shared memory multiprocessor with a separate cache memory for each processor, it is possible to have many copies of any one instruction operand: one copy in the main memory and one in each cache memory. When one copy of an operand is changed, the other copies of the operand must be changed also.

Example: Cache and the main memory may have inconsistent copies of the same object.



Suppose there are three processors, each having cache. Suppose the following scenario:-

- **Processor 1 read X:** obtains 24 from the memory and caches it.
- **Processor 2 read X:** obtains 24 from memory and caches it.
- **Again, processor 1 writes as X:** 64, Its locally cached copy is updated. Now, processor 3 reads X, what value should it get?
- Memory and processor 2 think it is 24 and processor 1 thinks it is 64.

As multiple processors operate in parallel, and independently multiple caches may possess different copies of the same memory block, this creates a cache coherence problem. Cache coherence is the discipline that ensures that changes in the values of shared operands are propagated throughout the system in a timely fashion.

There are three distinct level of cache coherence: -

1. Every write operation appears to occur instantaneously.
2. All processors see the same sequence of changes of values for each separate operand.
3. Different processors may see an operation and assume different sequences of values; this is known as non-coherent behaviour.

There are various Cache Coherence Protocols in multiprocessor system. These are:-

1. MSI protocol (Modified, Shared, Invalid)
2. MOSI protocol (Modified, Owned, Shared, Invalid)
3. MESI protocol (Modified, Exclusive, Shared, Invalid)
4. MOESI protocol (Modified, Owned, Exclusive, Shared, Invalid)

These important terms are discussed as follows:

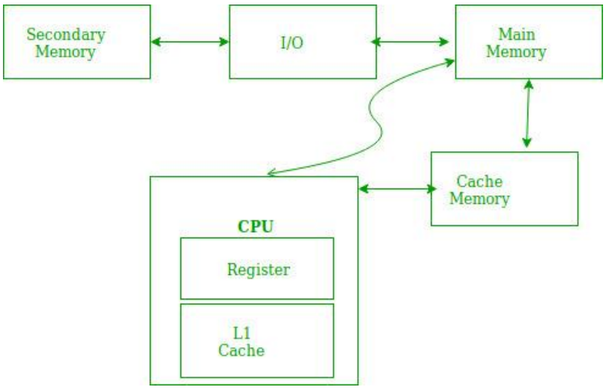
- **Modified –** It means that the value in the cache is dirty, that is the value in current cache is different from the main memory.
- **Exclusive –** It means that the value present in the cache is same as that present in the main memory, that is the value is clean.
- **Shared –** It means that the cache value holds the most recent data copy and that is what shared among all the cache and main memory as well.
- **Owned –** It means that the current cache holds the block and is now the owner of that block, that is having all rights on that particular block.
- **Invalid –** This states that the current cache block itself is invalid and is required to be fetched from other cache or main memory.

Coherency mechanisms: There are three types of coherence:

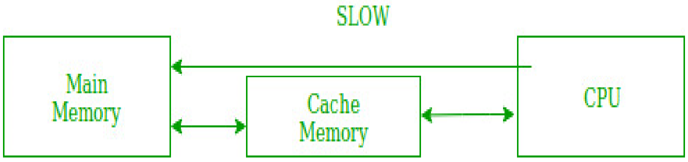
1. **Directory-based –** In a directory-based system, the data being shared is placed in a common directory that maintains the coherence between caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed, the directory either updates or invalidates the other caches with that entry.
2. **Snooping –** First introduced in 1983, snooping is a process where the individual caches monitor address lines for accesses to memory locations that they have cached. It is called a write invalidate protocol. When a write operation is observed to a location that a cache has a copy of and the cache controller invalidates its own copy of the snooped memory location.
3. **Snarfing –** It is a mechanism where a cache controller watches both address and data in an attempt to update its own copy of a memory location when a second master modifies a location in main memory. When a write operation is observed to a location that a cache has a copy of the cache controller updates its own copy of the snarfed memory location with the new data.

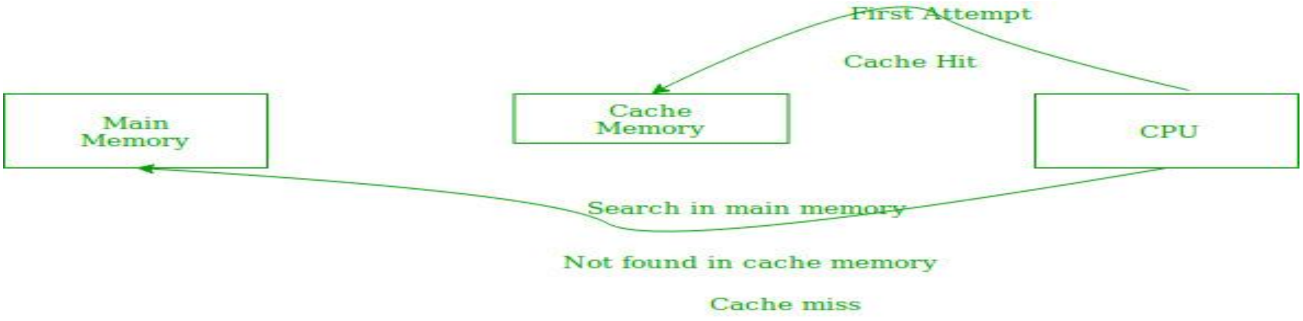
Locality of reference:

Locality of reference refers to a phenomenon in which a computer program tends to access same set of memory locations for a particular time period. In other words, **Locality of Reference** refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by loops and subroutine calls in a program.



1. In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.
2. In case of subroutine calls, every time the set of instructions are fetched from memory.
3. References to data items also get localized that means same data item is referenced again and again.





In the above figure, you can see that the CPU wants to read or fetch the data or instruction. First, it will access the cache memory as it is near to it and provides very fast access. If the required data or instruction is found, it will be fetched. This situation is known as a **cache hit.**

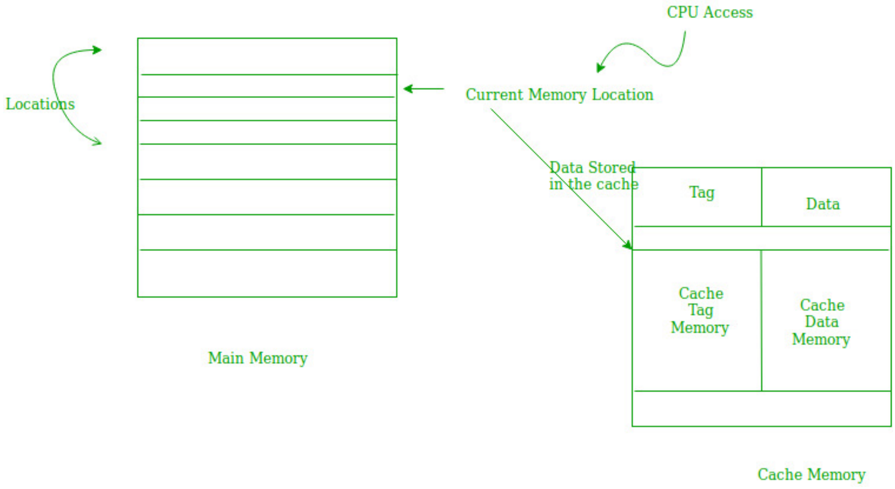
But if the required data or instruction is not found in the cache memory then this situation is known as a **cache miss.**

The main memory will be searched for the required data or instruction that was being searched and if found will go through one of the two ways:

1. First way is that the CPU should fetch the required data or instruction and use it and that's it but what, when the same data or instruction is required again. CPU again must access the same main memory location for it and we already know that main memory is the slowest to access.
2. The second way is to store the data or instruction in the cache memory so that if it is needed soon again in the near future it could be fetched in a much faster way.

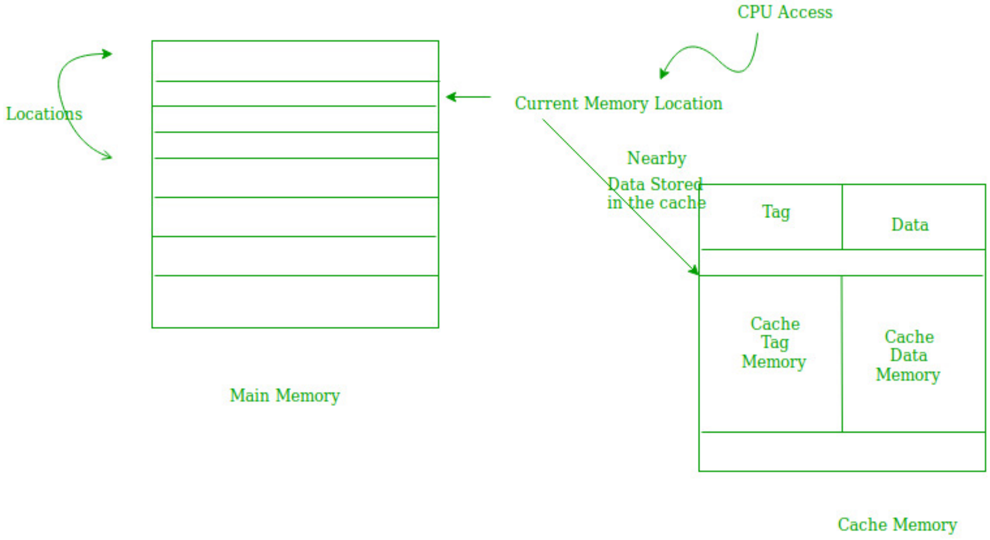
Cache Operation:
It is based on the principle of locality of reference. There are two ways with which data or instruction is fetched from main memory and get stored in cache memory. These two ways are the following:

1. **Temporal Locality –**
   Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.



   When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based on the fact that same data or instruction may be needed in near future. This is known as temporal locality. If some data is referenced, then there is a high probability that it will be referenced again in the near future.

2. **Spatial Locality –**
   Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.



Cache Performance:

The performance of the cache is measured in terms of hit ratio. When CPU refers to memory and find the data or instruction within the cache memory, it is known as cache hit. If the desired data or instruction is not found in the cache memory and CPU refers to the main memory to find that data or instruction, it is known as a cache miss.

Hit + Miss = Total CPU Reference

Hit Ratio(h) = Hit / (Hit + Miss)

Average access time of any memory system consists of two levels: Cache and Main Memory. If *Tc* is time to access cache memory and *Tm* is the time to access main memory then we can write:

Tavg = Average time to access memory

Tavg = h * Tc + (1-h)*(Tm + Tc)

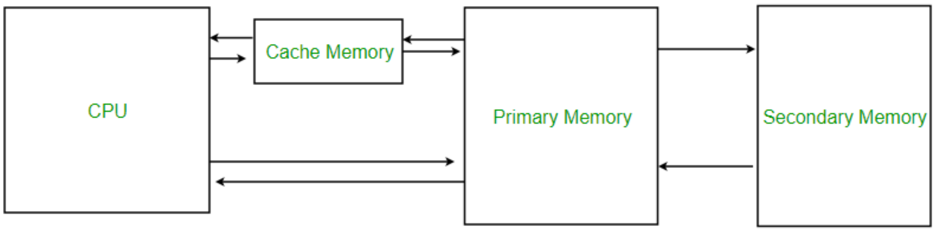
Difference between Spatial Locality and Temporal Locality:

S.No.	Spatial Locality	Temporal Locality
1.	In Spatial Locality, nearby instructions to recently executed instruction are likely to be executed soon.	In Temporal Locality, a recently executed instruction is likely to be executed again very soon.
2.	It refers to the tendency of execution which involve a number of memory locations .	It refers to the tendency of execution where memory location that have been used recently have a access.
3.	It is also known as locality in space.	It is also known as locality in time.
4.	It only refers to data item which are closed together in memory.	It repeatedly refers to same data in short time span.
5.	Each time new data comes into execution.	Each time same useful data comes into execution.
6.	Example : Data elements accessed in array (where each time different (or just next) element is being accessing).	Example : Data elements accessed in loops (where same data elements are accessed multiple times).

Cache Memory Organizations:

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register –**
  It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory –**
  It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory –**
  It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory –**
  It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:
When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For 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.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Mapping:

There are three different types of mapping used for the purpose of cache memory which are as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

1. **Direct Mapping –**
   The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line.
   In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

   i = j modulo m

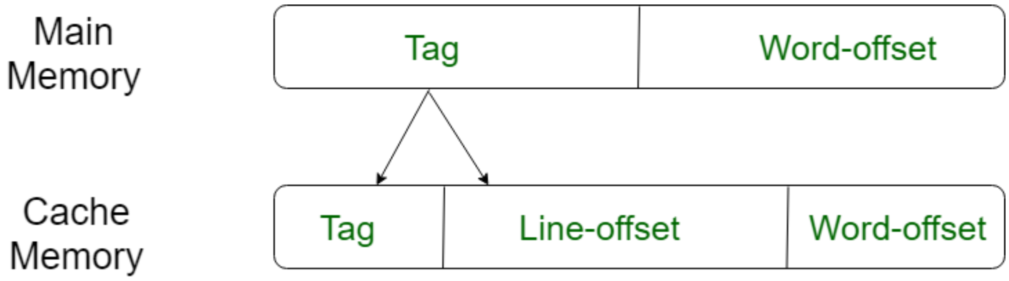
   where

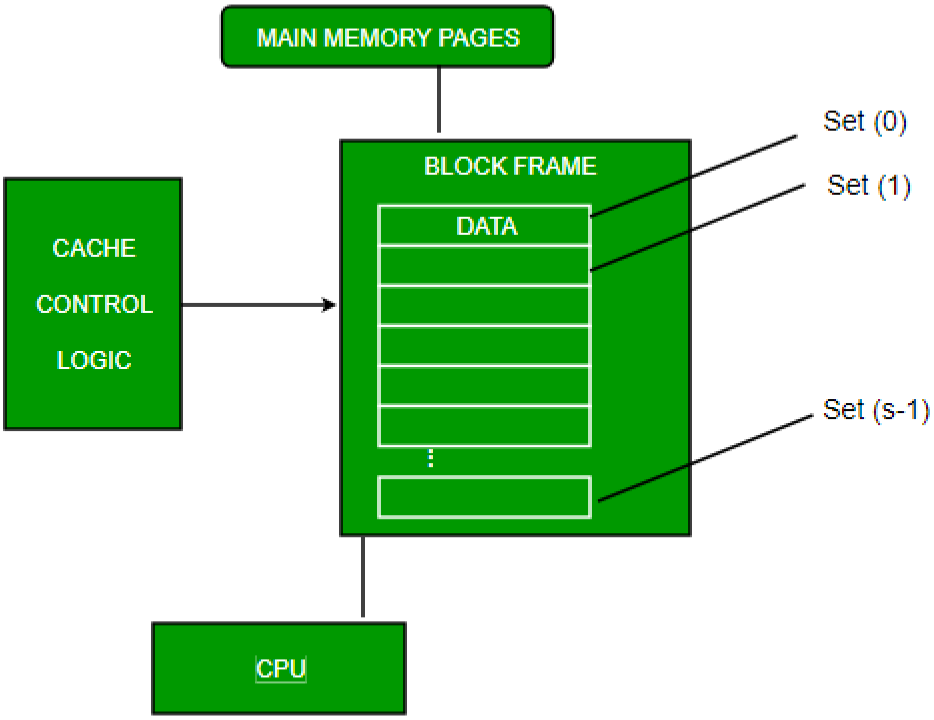
   i=cache line number

   j= main memory block number

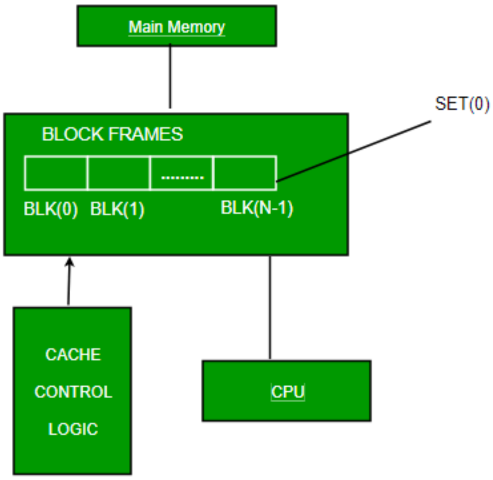
   m=number of lines in the cache

   For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache.





2. **Associative Mapping –**
   In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.



3. **Set-associative Mapping –**
   This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a ***set***. Then a block in memory can map to any one of the lines of a specific set..Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques.

In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are

m = v * k

i= j mod v

where

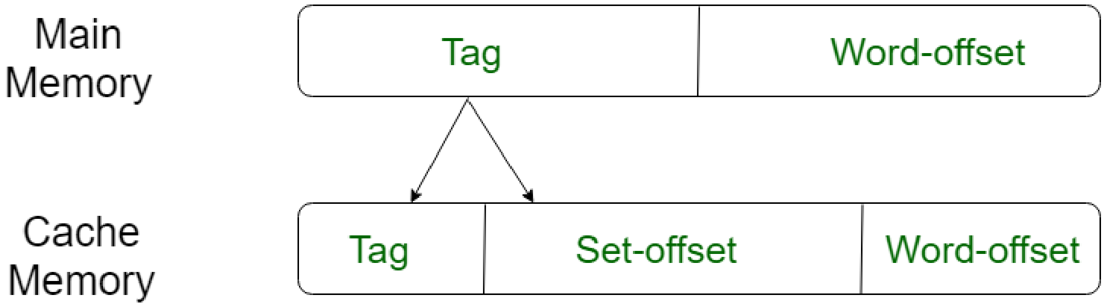
i=cache set number

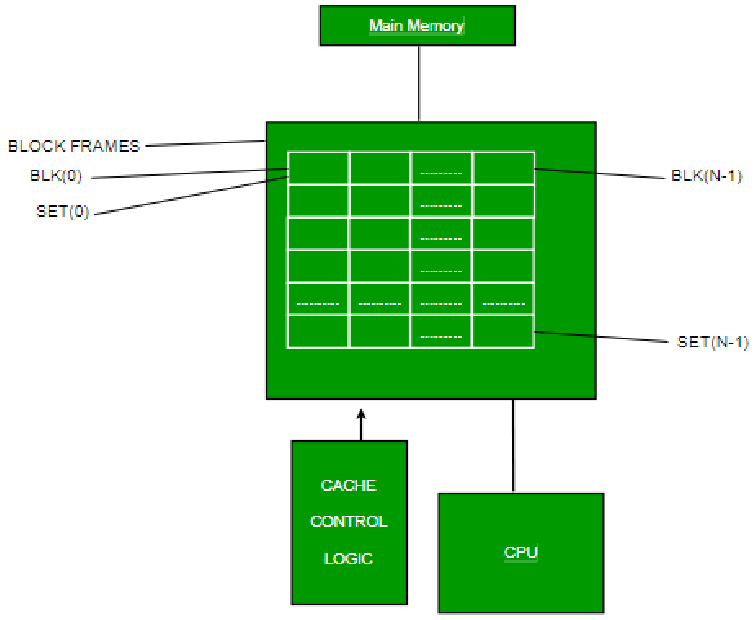
j=main memory block number

v=number of sets

m=number of lines in the cache number of sets

k=number of lines in each set





Application of Cache Memory –

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache –

1. **Primary Cache –**
   A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
2. **Secondary Cache –**
   Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Cache optimization Techniques:

Optimization of cache performance ensures that it is utilized in a very efficient manner.

Average Memory Access Time (AMAT):

AMAT helps in analyzing the cache memory and its performance. The lesser the **AMAT**, the better the performance is. AMAT can be calculated as,

AMAT = Hit Ratio * Cache access time + Miss Ratio * Main memory access time
= $\mathbf{(h * t_{c}) + (1-h) * (t_c + t_m)}$

Example 1: What is the average memory access time for a machine with a cache hit rate of 75% and cache access time of 3 ns and main memory access time of 110 ns.

Solution:

Average Memory Access Time(AMAT) = = $\mathbf{(h * t_{c}) + (1-h) * (t_c + t_m)}$
Given,
Hit Ratio(h) = 75/100 = 3/4 = 0.75
Miss Ratio (1-h) = 1-0.75 = 0.25
Cache access time(t_c) = 3ns

Main memory access time(effectively) = tc + tm = 3 + 110 = 113 ns
Average Memory Access Time(AMAT) = (0.75 * 3) + (0.25 * (3+110))
= 2.25 + 28.25
= 30.5 ns

Example 2: Calculate AMAT when Hit Time is 0.9 ns, Miss Rate is 0.04, and Miss Penalty is 80 ns.

Solution :
Average Memory Access Time(AMAT) = **Hit Time + (Miss Rate * Miss Penalty)**
Here, Given,
Hit time = 0.9 ns
Miss Rate = 0.04
Miss Penalty = 80 ns
Average Memory Access Time(AMAT) = 0.9 + (0.04*80)
= 0.9 + 3.2
= 4.1 ns

Methods for reducing Hit Time, Miss Rate, and Miss Penalty:

Methods to reduce Hit Time:

1. Small and simple caches: If lesser hardware is required for the implementation of caches, then it decreases the Hit time because of the shorter critical path through the Hardware.

2. Avoid Address translation during indexing: Caches that use physical addresses for indexing are known as a physical cache. Caches that use the virtual addresses for indexing are known as virtual cache. Address translation can be avoided by using virtual caches. Hence, they help in reducing the hit time.

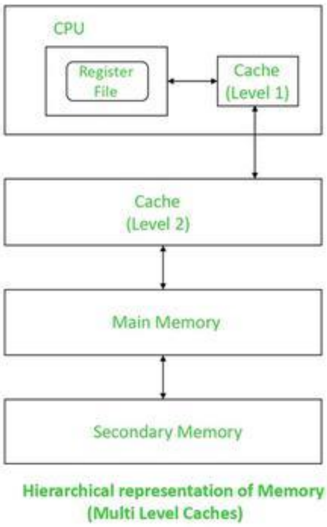
Methods to reduce Miss Rate:

1. Larger block size: If the block size is increased, spatial locality can be exploited in an efficient way which results in a reduction of miss rates. But it may result in an increase in miss penalties. The size can't be extended beyond a certain point since it affects negatively the point of increasing miss rate. Because larger block size implies a lesser number of blocks which results in increased conflict misses.
2. Larger cache size: Increasing the cache size results in a decrease of capacity misses, thereby decreasing the miss rate. But, they increase the hit time and power consumption.
3. Higher associativity: Higher associativity results in a decrease in conflict misses. Thereby, it helps in reducing the miss rate.

Methods to reduce Miss Penalty:

1. Multi-Level Caches: If there is only one level of cache, then we need to decide between keeping the cache size small in order to reduce the hit time or making it larger so that the miss rate can be reduced. Both of them can be achieved simultaneously by introducing cache at the next levels.
Suppose, if a two-level cache is considered:

- The first level cache is smaller in size and has faster clock cycles comparable to that of the CPU.
- Second-level cache is larger than the first-level cache but has faster clock cycles compared to that of main memory. This large size helps in avoiding much access going to the main memory. Thereby, it also helps in reducing the miss penalty.



Hierarchical representation of Memory

2. Critical word first and Early Restart: Generally, the processor requires one word of the block at a time. So, there is no need of waiting until the full block is loaded before sending the requested word. This is achieved using:

- **The critical word first:** It is also called a requested word first. In this method, the exact word required is requested from the memory and as soon as it arrives, it is sent to the processor. In this way, two things are achieved, the processor continues execution, and the other words in the block are read at the same time.
- **Early Restart:** In this method, the words are fetched in the normal order. When the requested word arrives, it is immediately sent to the processor which continues execution with the requested word.

These are the basic methods through which the performance of cache can be optimized.

Virtual Memory:

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.

The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory is available not by the actual number of the main storage locations.

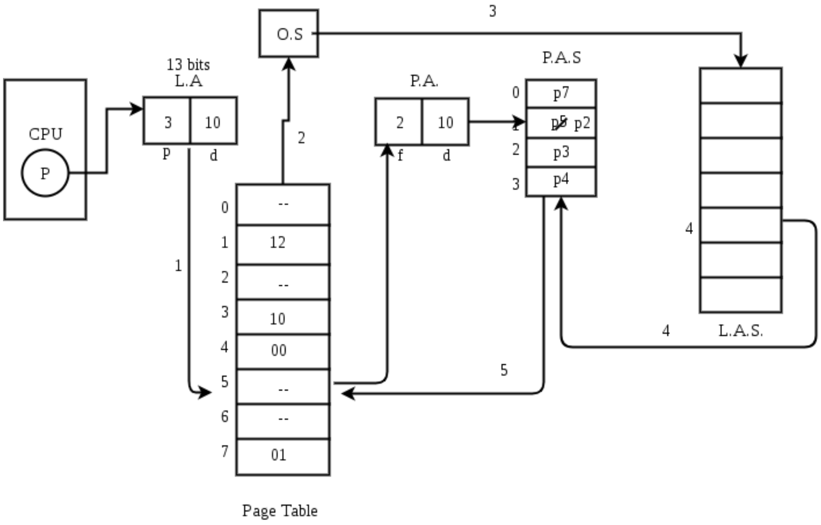
It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
2. A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using **Demand Paging or Demand Segmentation**.

Demand Paging :

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.
The process includes the following steps :



1. If the CPU tries to refer to a page that is currently not available in the main memory, it generates an interrupt indicating a memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision-making of replacing the page in physical address space.

5. The page table will be updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into the ready state.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

Advantages :

- More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
- A process may be larger than all of the main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.
- It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

Page Fault Service Time :

The time taken to service the page fault is called page fault service time. The page fault service time includes the time taken to perform all the above six steps.

Let Main memory access time is: m

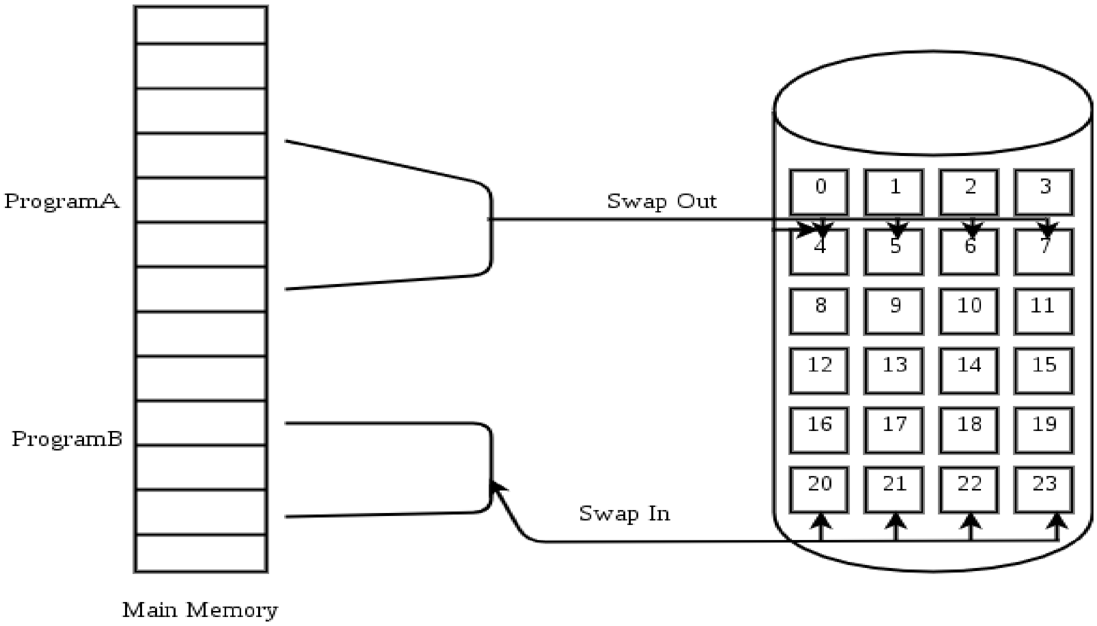
Page fault service time is: s

Page fault rate is : p

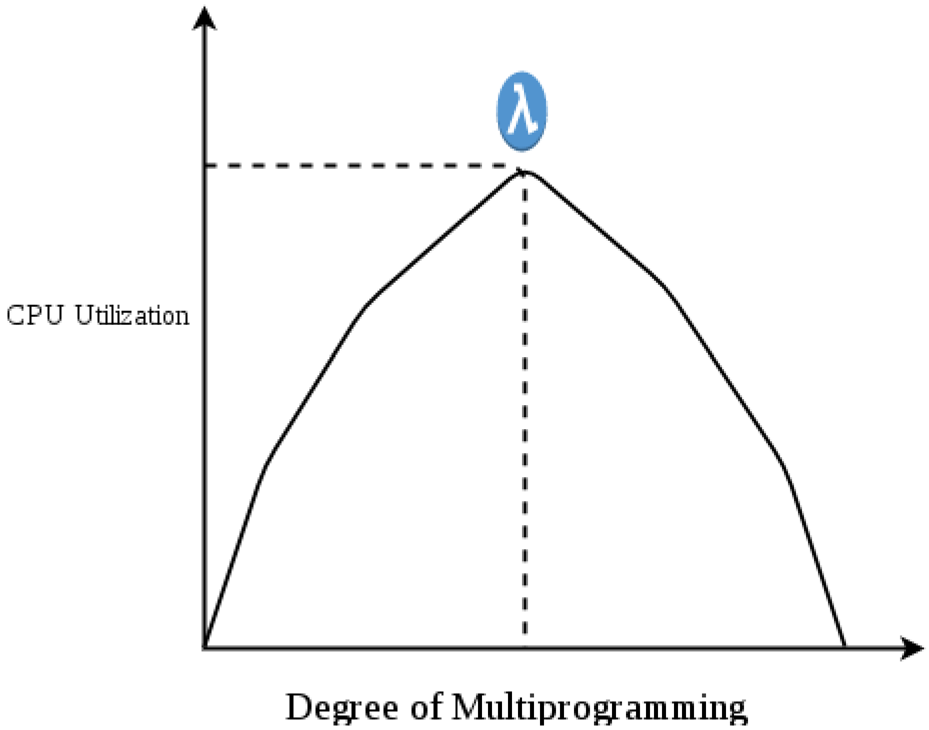
Then, Effective memory access time = (p*s) + (1-p)*m

Swapping:

Swapping a process out means removing all of its pages from memory, or marking them so that they will be removed by the normal page replacement process. Suspending a process ensures that it is not runnable while it is swapped out. At some later time, the system swaps back the process from the secondary storage to the main memory. When a process is busy swapping pages in and out then this situation is called thrashing.



Thrashing :



At any given time, only a few pages of any process are in the main memory and therefore more processes can be maintained in memory. Furthermore, time is saved because unused pages are not swapped in and out of memory. However, the OS must be clever about how it manages this scheme. In the steady-state practically, all of the main memory will be occupied with process pages, so that the processor and OS have direct access to as many processes as possible. Thus when the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing. The system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.

In the given diagram, the initial degree of multiprogramming up to some extent of point(lambda), the CPU utilization is very high and the system resources are utilized 100%. But if we further increase the degree of multiprogramming the CPU utilization will drastically fall down and the system will spend more time only on the page replacement and the time is taken to complete the execution of the process will increase. This situation in the system is called thrashing.

Causes of Thrashing:

1. **High degree of multiprogramming:** If the number of processes keeps on increasing in the memory then the number of frames allocated to each process will be decreased. So, fewer frames will be available for each process. Due to this, a page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.
   For example:
   Let free frames = 400
   Case 1: Number of processes = 100
   Then, each process will get 4 frames.
   Case 2: Number of processes = 400
   Each process will get 1 frame.
   Case 2 is a condition of thrashing, as the number of processes is increased, frames per process are decreased. Hence CPU time will be consumed in just swapping pages.

2. **Lacks Frames**: If a process has fewer frames, then fewer pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

Recovery of Thrashing:

- Do not allow the system to go into thrashing by instructing the long-term scheduler not to bring the processes into memory after the threshold.
- If the system is already thrashing, then instruct the mid-term scheduler to suspend some of the processes so that we can recover the system from thrashing.

Techniques for Fast Address Translation:

In Non Contiguous Memory allocation, processes can be allocated anywhere in available space. The address translation in non-contiguous memory allocation is difficult.
There are several techniques used for address translation in non contiguous memory allocation like Paging, Multilevel paging, Inverted paging, Segmentation, Segmented paging. Different data structures and hardware support like TLB are required in these techniques.