Memory Management

Memory management is primarily concerned with allocation of physical memory of finite capacity to requesting processes.

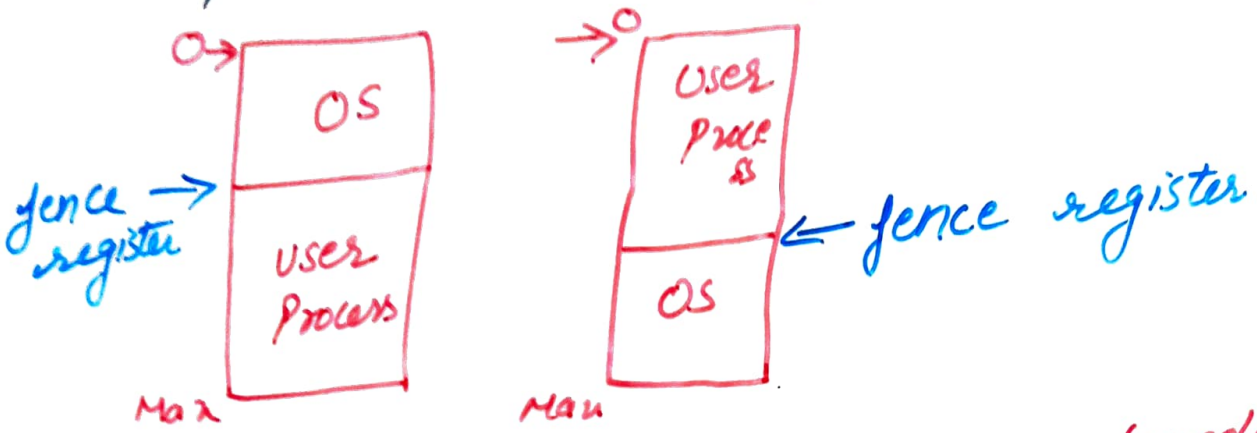
Memory is one of the central resources required for a computer operation. Memory is organised in words, Each word may consist of one byte, two bytes or more. Each memory location is addressable & has unique finite address.

SINGLE PROCESS MONITOR :- The way memory is managed in case of single user system is considerably different from that of the multi-user systems. In a multi user systems the resources are to be allocated among more users than one. This makes memory management in a multi user system is extremely complex.

Even in a single user system more than one process may exist at a time and resources are to be allocated among them.

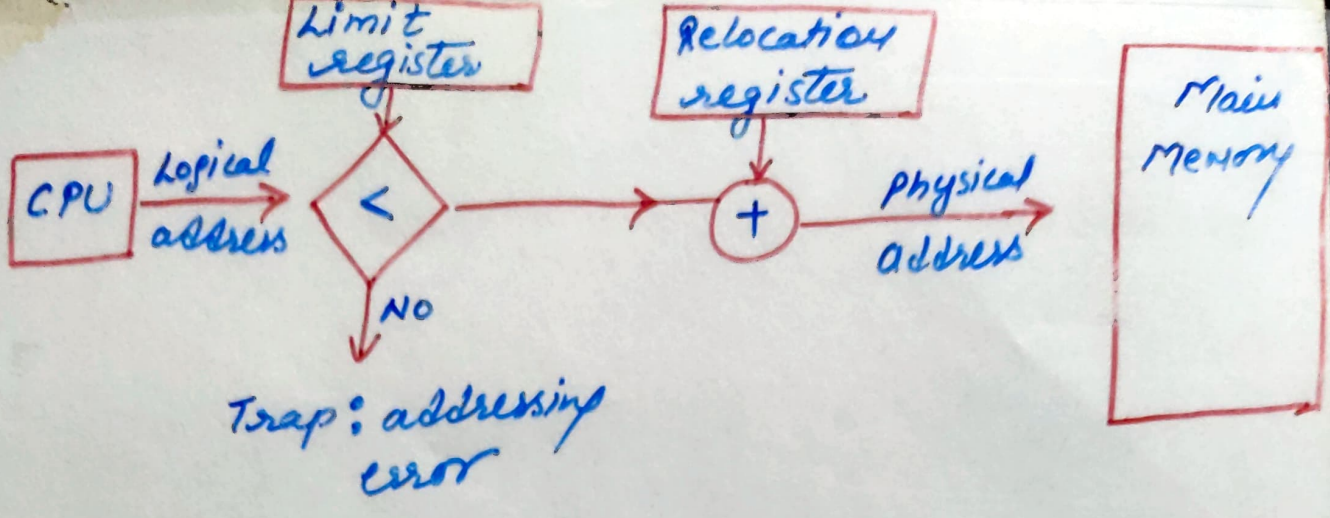
Memory Partition :- Memory is simply divided into two contiguous areas. One of them is usually permanently

allocates to the resident portion of the operating system and the other for the user processes. The OS may be either low Memory or into the high memory



Fence register is used to draw a boundary between the OS and the user Process area.

Both the OS & user processes reside in the memory, some mechanism needs to be enforced to protect the memory allocated to the operating system from being accesses by the user process. This protection may be enforced with the help of two registers - relocation register and limit register. The relocation register contains the value of the smallest physical address while limit register contains the range of the allowable logical address. The h/w support for this type of partitioning is :



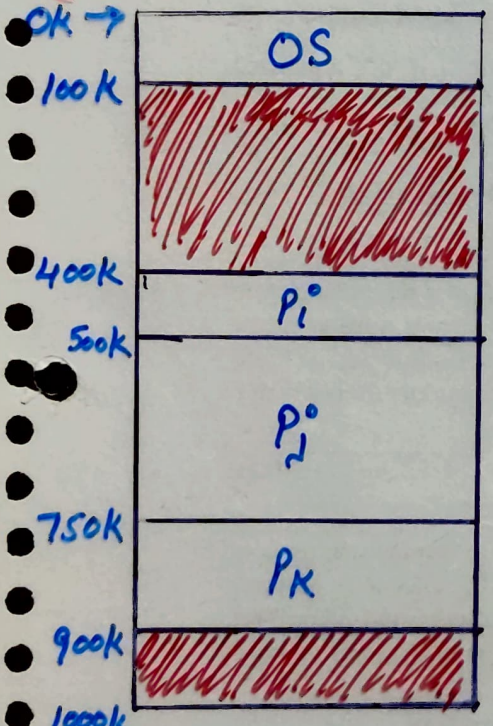
H/W support for relocation & limit registers

In a multiprogramming environment, it is desirable that more than one user processes may be reside into the memory. For this purpose, memory may be divided into multiple partitions of fixed size/ variant size. Depending on when and how partitions are created and modified, memory partitioning may be static or dynamic.

STATIC PARTITION :- static partitioning generally implies that the division of memory is made at some time prior to the execution of user programs and the partitions remain fixed thereafter. The number and sizes of individual partitions are usually determined during the system generation process, taking into consi-

deration the capacity of the available physical memory, the desired degree of multiprogramming and the typical sizes of processes most frequently run on a given installation.

Once partitions are defined, an operating system needs to keep track of their status, such as free or in use, for allocation purposes. Current partition status and attributes are often collected in a data structure called the Partition Description Table (PDT).



Static Partitions

Partition No.	Partition Base	Partition Size	Partition Status
0	0K	100K	Allocated
1	100K	300K	Free
2	400K	100K	Allocated
3	500K	250K	Allocated
4	750K	150K	Allocated
5	900K	100K	Free

Partition Description Table

when the process terminates or needs to be swapped out, this information may be used to locate the related partition and to update its status to FREE. For these basic ideas to be implemented, two important facets must be further elaborated and resolved. The first is the partition allocation strategy i.e. how to select a specific partition for a given process.

The second problem is presented by the situation when no suitable partition is available for allocation.

First fit approach basically consists of allocating the first free partition large enough to accomodate the process being created.

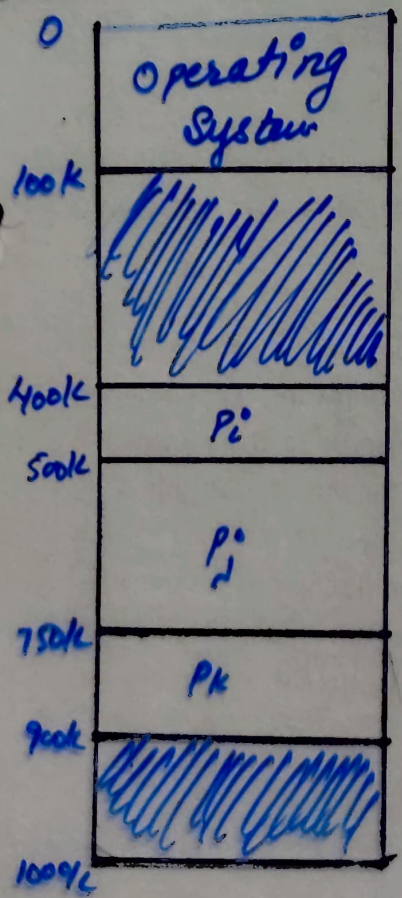
Best fit approach on the other hand, requires that the OS allocate the smallest free partition that meets the requirements of the process under consideration.

Partitioned Memory Allocation - Dynamic :-

The memory manager may continue to create and to allocate partitions to requesting

processes until all physical memory is exhausted, or the maximum allowable degree of multiprogramming is reached.

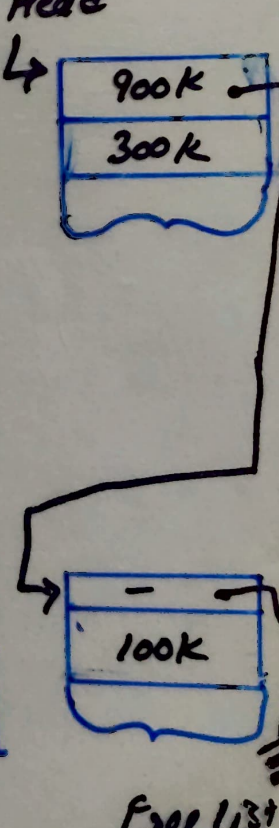
The first step in this activity is to locate a contiguous free area of memory, if any, that is equal to or larger than the process size declared with the submitted image. If a suitable free area is found, the operating system "carves out" a partition from it to provide an exact fit to the process's need. if no suitable free area can be allocated for some reason, the operating system returns an error indication.



Memory

0	OK	100K	Allocated
1	–	–	–
2	400K	100K	Allocated
3	500K	250K	Allocated
4	750K	150K	Allocated
5	–	–	–
6	–	–	–
7	–	–	–

PDT



Free List

(v) Consider a swapping system in which memory consists of the following hole size in memory order

10K, 4K, 20K, 18K, 7K, 9K, 12K, 15K. which hole is taken for successive segment request of 12K, 10K, 9K for worst fit.

(vi) Suppose that a total of 64MB memory is available in a system. This memory space is partitioned into 8 fixed size slots of 8MB each. Assume 8 processes are currently requesting memory usage with size indicated as below:-

2MB, 4MB, 3MB, 7MB, 2MB, 6MB, 1MB, & 8MB

Calculate the size of memory wasted due to external and internal fragmentation. Derive the memory utilization ratio by dividing the total allocated memory by total requested memory.

Logical versus Physical - Address Space :- An address generated by the CPU is commonly referred to as a logical address, whereas an address seen by the memory unit referred to as a physical address.

The compile-time and load-time address binding methods generate identical logical and physical addresses.

The execution-time address binding scheme results in differing logical and physical addresses refer to the logical address as a virtual address.

The set of all logical addresses generated by a program is a logical address space; the set of all physical addresses corresponding to these logical addresses is a physical address space.

The run time mapping from virtual to physical addresses is done by a hardware device called the memory-management unit (MMU).

Fragmentation :- Fragmentation refers to the inability of the operating system to allocate portions of unused memory

Memory fragmentation can be internal as well as external.

Internal fragmentation :- Physical memory divided into fixed size blocks and allocate memory in units based on block size. With this approach, the memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is is internal fragmentation - memory that is internal to a partition but is not being used. The extent to which internal fragmentation causes memory to be wasted in a given system varies depending on several factors, such as the number and size of partitions, frequency of execution of processes of a given size, and the average process size and variance. Wasted memory from internal fragmentation also tends to increase when large but infrequently run processes must be accomodated, because a partition large enough for the purpose may be otherwise poorly utilized.

External fragmentation :- In the dynamic Partition memory is

divided into the little pieces according to the requirement. External fragmentation exists when there is enough total memory space to satisfy a request, but the available spaces are not contiguous; storage is fragmented into a large number of small holes. This fragmentation problem can be severe. In the worst case, we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

How to overcome the problem of Internal fragmentation & External fragmentation.

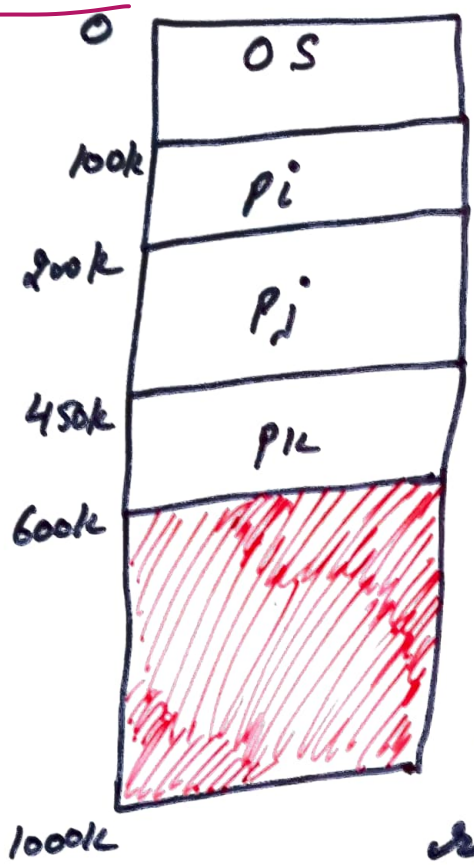
→ To overcome the problem of Internal fragmentation we use Segmentation Technique.

→ To overcome the problem of External fragmentation we use a Technique of

(i) Paging (ii) Compaction

Compaction :- Relocate some or all partitions into one end of memory and thus combine the holes into one

end of memory and thus combine the holes into one large free area. Since affected processes must be suspended and actually copied from one area of memory into another.



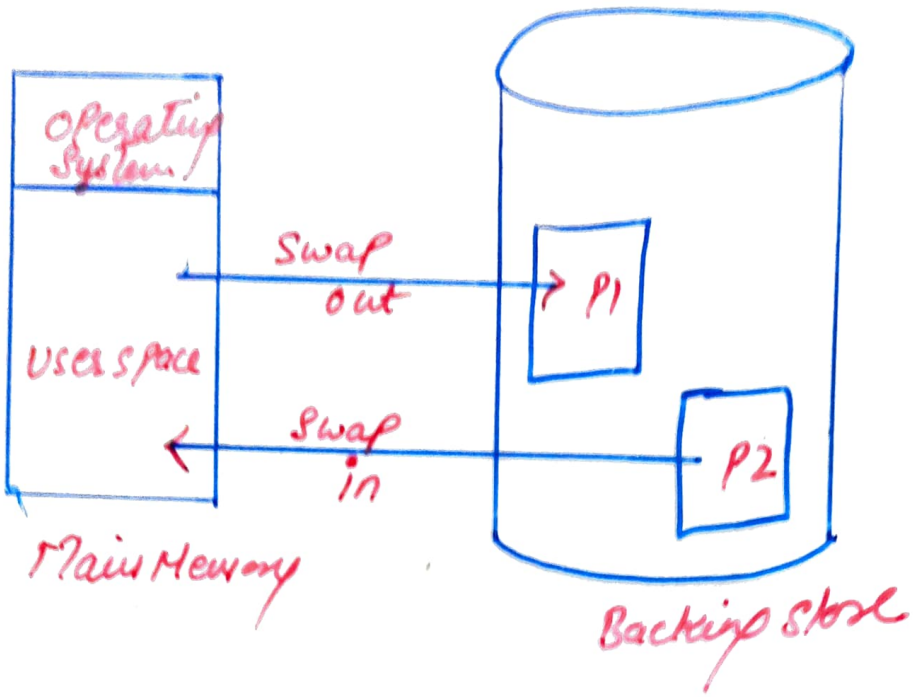
Compaction is avoided:-

→ Changing the position of running process.

→ whenever compaction has to be performed in the system at that time execution of all processes has been suspended since the no. of processes have to be reallocated within a main memory. This will result in a wastage of CPU cycles or CPU time. Hence, a large amount of cost of time will be involved in performing a compaction.

Memory Swapping:- A process needs to be in the memory to be executes. A process, however, can be swapped temporarily out of memory to a backing store, and then brought back into

memory for continued execution.



Swapping requires a backing store. The backing store is commonly a fast disk large enough to accomodate copies of all memory images for all users, and must provide direct access to these memory images. The system maintains a ready queue consisting of all Processes whose memory images are on the backing store or in memory and are ready to run. whenever the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If the process is not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers as normal and transfers control

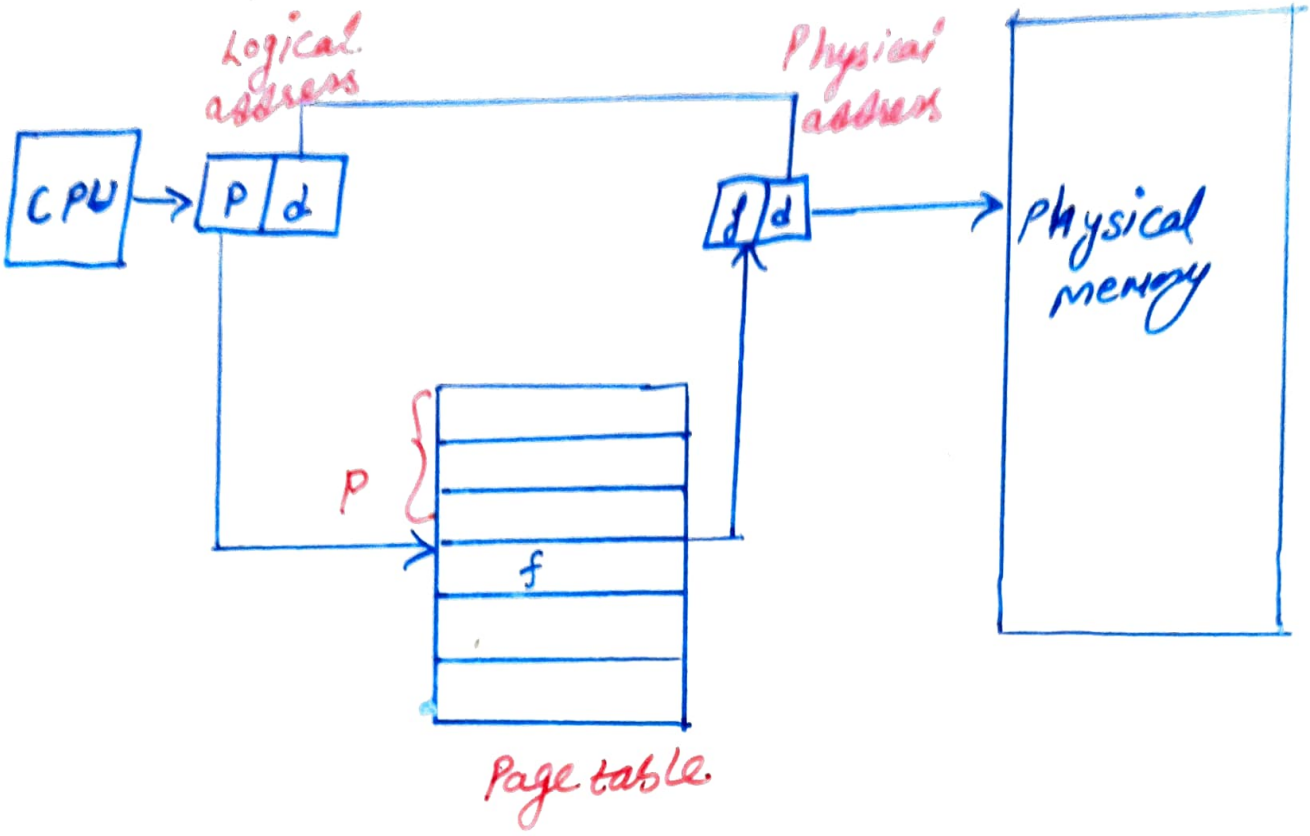
It should be clear that the content switch time in such a swapping system is fairly high. To get an idea of the content switch time, let us assume that the user process is of size 100k & the backing store is Standard hard disk with a transfer rate of 1 Megabyte per second. So the actual rate transfer is

100k / 1000k per second = 1/10 second
= 100 millisecond

Memory Paging :- A possible solution to the external fragmentation problem is to permit the logical address space of a process to be non contiguous, thus allowing a process to be allocated physical memory wherever the latter is available. One way of implementing this solution is through the use of a paging scheme.

Physical memory is broken into fixed size blocks called frames.
Logical memory is broken into blocks of the same size called Pages.

When a process is to be executed, its pages are loaded into any available memory frames from the backing store.



Hardware support of Paging

Every address generated by the CPU is divided into two parts:

- → a page number (p)
- → a page offset (d)

The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address combined with the page offset to define the physical memory address that is sent to the memory unit.

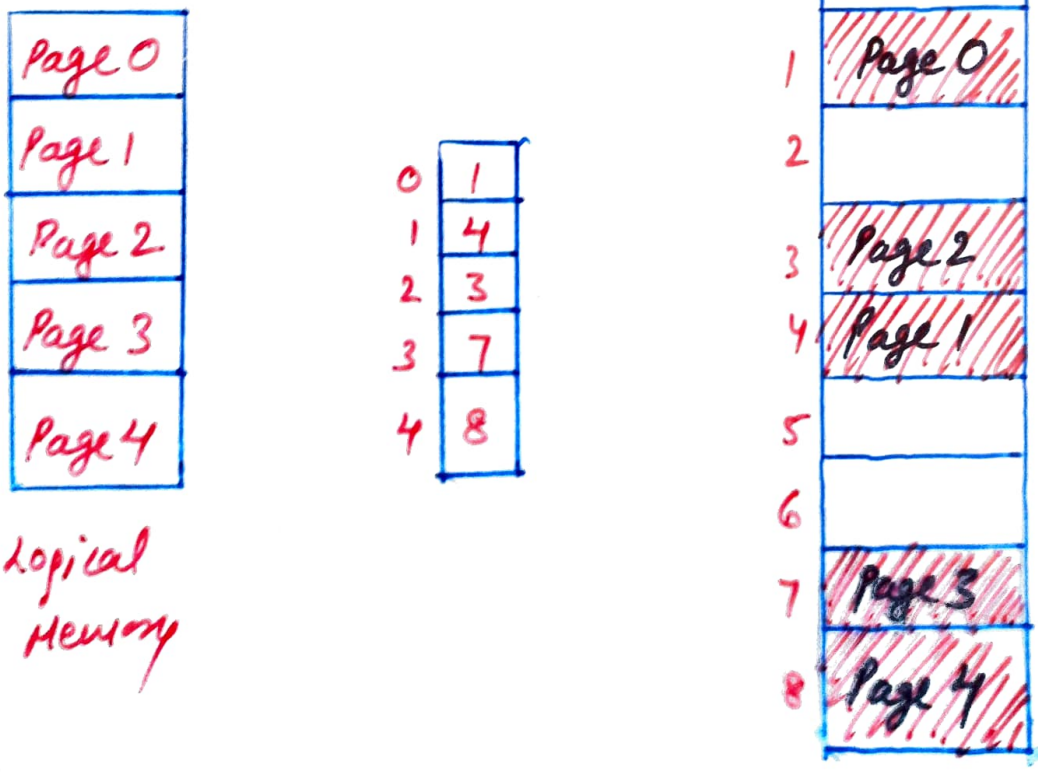
⇒ The page size (like the frame size) is defined by the hardware.

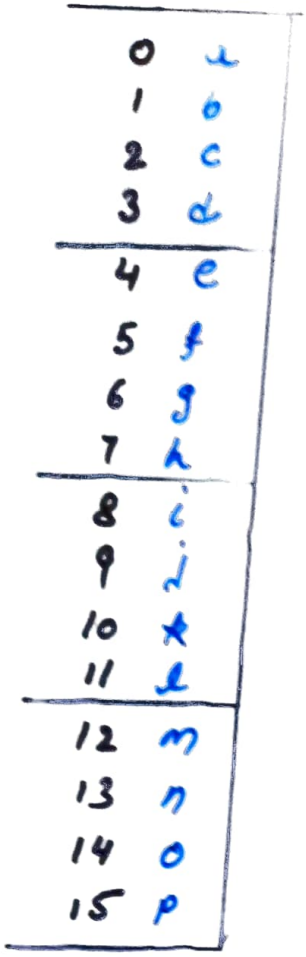
⇒ The size of page is typically a power of 2, depending on the computer architecture.

⇒ If the size of logical address space is 2^m, and a page size is 2^n addressing units (bytes or words), then the high order $m-n$ its of a logical address designate the page number, and the n low-order bits designate the page offset. Thus, the logical address is as follows:

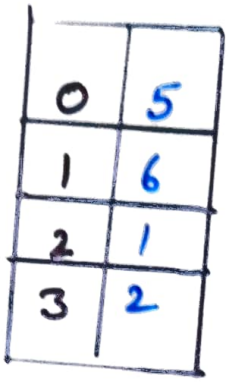
Page number	Page offset
p	d
m-n	n

where p is an index into the page table and d is the displacement within the page.

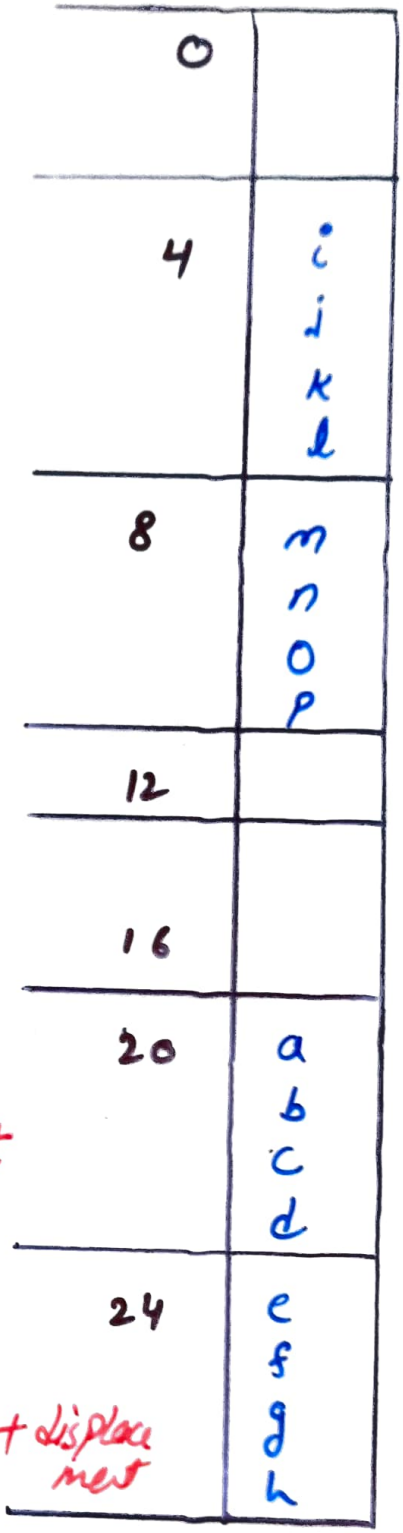




Logical Memory



Page table



Physical Memory

Formulae :- frame × word size + offset

or

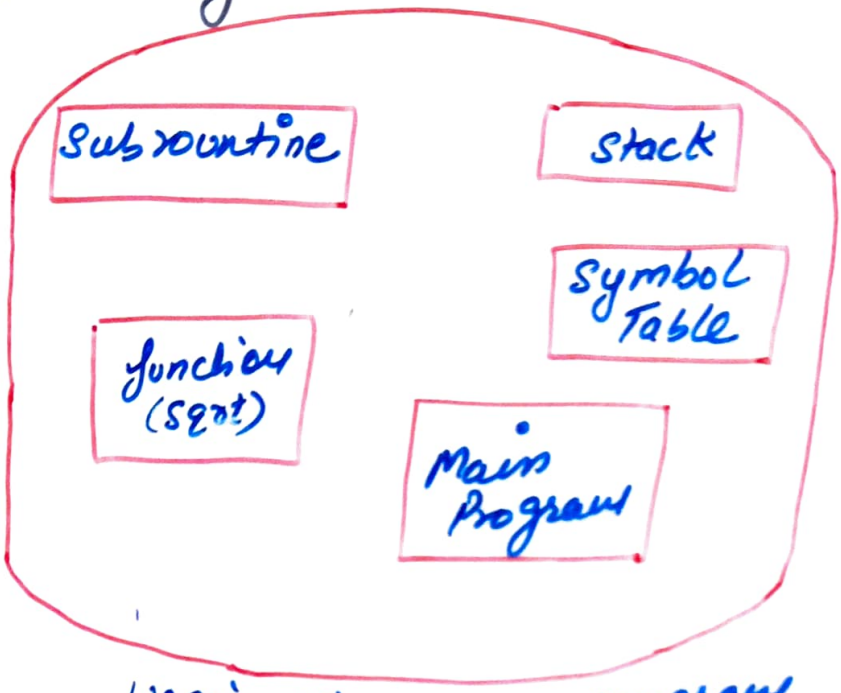
Mapping To Physical

Address = frame no × Size of frame + displacement

SEGMENTATION :- what is the user's view of memory ? Does the user think of memory as a linear array of bytes, some containing instructions and other containing data, or is there some other preference

memory view?

User prefers to view memory as a collection of variable size segments, with no necessary ordering among segments.



User's view of a program

Segmentation is a memory management scheme that supports this user view of memory. A logical-address space is a collection of segments. Each segment has a name and a length. The address specify both the segment name and the offset within the segment. The user therefore specifies both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset.

For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus a logical address consists of a two tuple:

<segment-number, offset>

Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program. A C compiler might create the separate segments for:

1) the global variables;
2) the function call stack, to store parameters and return addresses.
3) the code portion of each function
4) the heap from which memory is allocated at runtime

Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers