Operating System: Memory Management

Prepared By:

Dr. Sanjeev Gangwar

Assistant Professor,

Department of Computer Applications,

VBS Purvanchal University, Jaunpur

Memory Management

Memory is large array of words or bytes, each having its unique address. CPU fetches instructions from memory according to value of program counter. The instructions undergo instruction execution cycle. To increase both CPU utilization and speed of its response to users, computers must keep several processes in memory. Specifically, the memory management modules are concerned with following four functions:

1. Keeping track of whether each location is allocated or unallocated, to which process and how much.

2. Deciding to whom the memory is allocated, how much, when and where. If memory is to be shared by more than one process concurrently, it must be determined which process' request should be satisfied.

3. Once it is decided to allocate memory, the specific locations must be selected and allocated. Memory status information is updated.

4. Handling the deallocation/reallocation of memory. After the process holding memory is finished, memory locations held by it are declared free by changing the status information.

There are varieties of memory management systems. They are:

- **Contiguous Memory Management:** In this approach, each program occupies a single contiguous block of storage locations.
- **Non-Contiguous Memory Management:** In these, a program is divided into several blocks or segments that may be placed throughout main storage in pieces or chunks not necessarily adjacent to one another. It is the function of Operating System to manage these different chunks in such a way that they appear to be contiguous to the user.

Binding of Instructions and Data to Memory: the binding of data and program to memory address can be done at any of the following steps:

- **Compile Time:** Binding at compile time generates absolute addresses where a prior knowledge is required that where a process resides in the memory. After sometime if the starting location of a process in memory is changed, then the entire process must be recompiled to generate the absolute address again.
- **Load Time:** at compile time if it is not known that where a process will reside in memory then the compiler must generate relocatable address. In this case, final binding is delayed until load time. If the starting address is changed then we need only to reload the user code to incorporate this changed value.
- **Execution Time:** this method permits moving a process from one memory segment to another during rum time. In this case, final binding is delayed until run time.

Most systems allow a user process to reside in any part of the physical memory. In most cases, a user program will go through several steps- some of which are optional-before being executed as shown in figure 1. Addresses may be represented in different ways during these steps. Addresses in the source programs are generally symbolic. A compiler will bind these addresses to relocatable addresses. The linkage editor or loader will bind these addresses to absolute addresses. Each binding is a mapping from one address space to another.

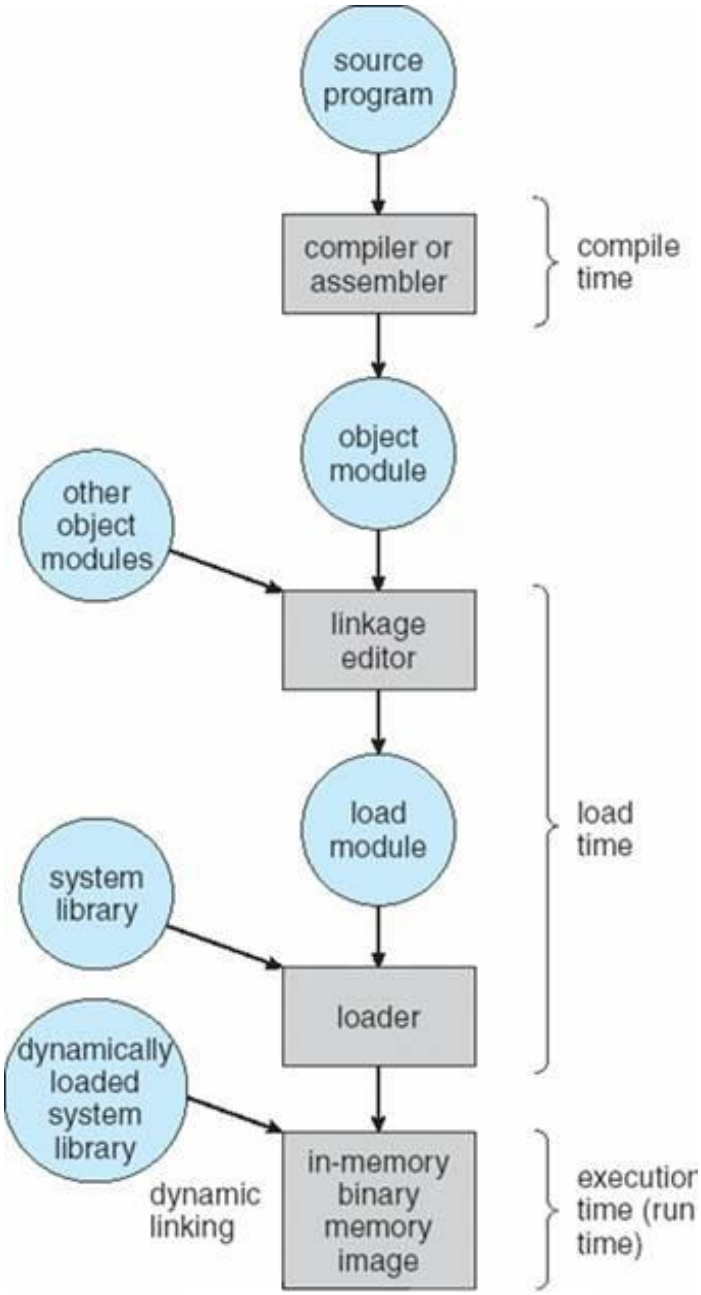

Fig 1: Multistep processing of a user program

Logical And Physical Addresses: An address generated by the CPU is commonly refereed as **Logical Address**, whereas the address seen by the memory unit, that is one loaded into the memory address register of the memory is commonly refereed as the **Physical Address**. The compile time and load time address binding generates the identical **logical and physical addresses.** However, the execution times address binding scheme results in **differing logical and physical addresses.**

The set of all **logical addresses** generated by a program is known as **Logical Address Space**, whereas the set of all **physical addresses** corresponding to these logical addresses is **Physical Address Space.** Now, the run time mapping from virtual address to physical address is done by a hardware device known as **Memory Management Unit (MMU)**.

Here in the case of mapping the base register is known as relocation register. The value in the relocation register is added to the address generated by a user process at the time it is sent to memory. Let's understand this situation with the help of example: If the base register contains the value 1000, then an attempt by the user to address location 0 is dynamically relocated to location 1000, an access to location 346 is mapped to location 1346 as illustrated in Figure 2

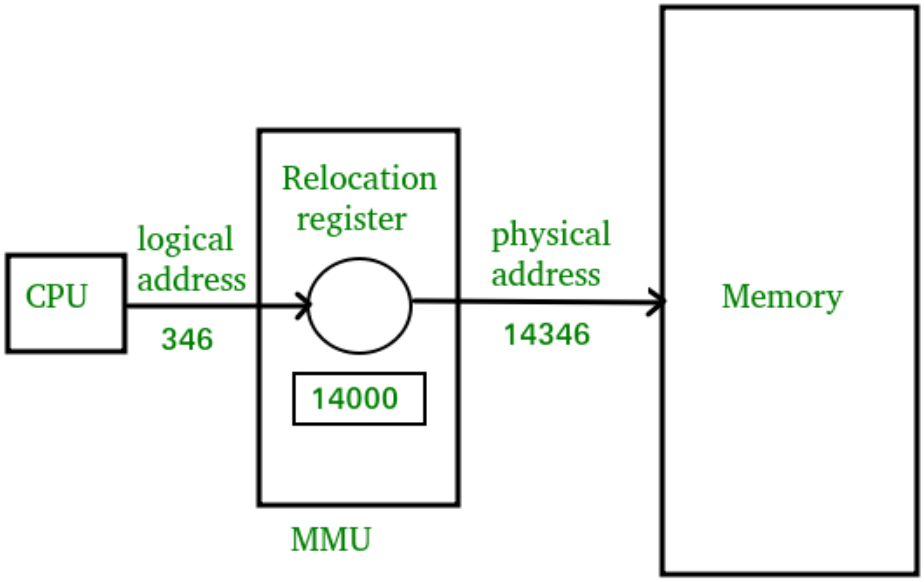

Fig 2: Dynamic relocation using a relocation register

Dynamic Loading

- Routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other the desired

Routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.

- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the operating system is required.
- Implemented through program design.

Dynamic Linking

- Linking is postponed until execution time.
- Small piece of code, stub, is used to locate the appropriate memory-resident library routine, or to load the library if the routine is not already present.
- When this stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory.
- Stub replaces itself with the address of the routine, and executes the routine.
- Thus the next time that code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.
- Operating system is needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

Swapping: Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve main memory utilization. In secondary memory, the place where the swapped-out process is stored is called swap space.

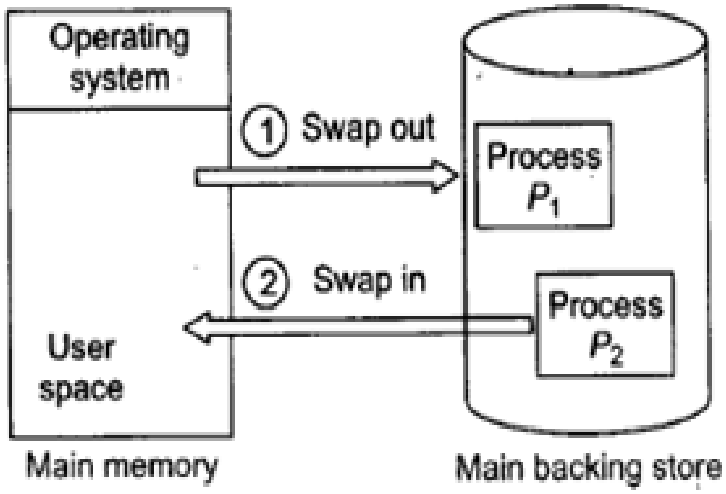

Fig 3: Swapping of two processes using a disk as a backing store

A variant of this swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process so that it can load and execute higher-priority process. When the higher-priority process

Finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is sometimes called roll out, roll in.

Contiguous Allocation: In contiguous memory allocation, all the available memory space remain together in one place. It means freely available memory partitions are not scattered here and there across the whole memory space. The main memory must accommodate both the operating system and the various user processes. The memory is usually divided into two partitions, one for the resident operating system, and one for the user processes.

In the contiguous memory allocation when any user process request for the memory a single section of the contiguous memory block is given to that process according to its need. We can achieve contiguous memory allocation by dividing memory into the fixed-sized partition.

A single process is allocated in that fixed sized single partition. But this will increase the degree of multiprogramming means more than one process in the main memory that bounds the number of fixed partition done in memory. Internal fragmentation increases because of the contiguous memory allocation.

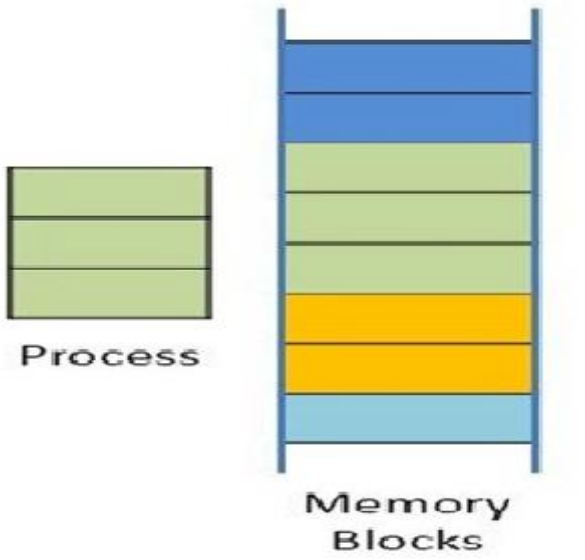

Fig 4: Contiguous Memory Allocation

Fixed sized partition: In the fixed sized partition the system divides memory into fixed size partition (may or may not be of the same size) here entire partition is allowed to a process and if there is some wastage inside the partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

Variable size partition: In the variable size partition, the memory is treated as one unit and space allocated to a process is exactly the same as required and the leftover space can be reused again.

This procedure is a particular instance of the general **dynamic storage-allocation problem**, which is how to satisfy a request of size n from a list of free holes. There are many solutions to this problem. The set of holes is searched to determine which hole is best to allocate, **first-fit, best-fit, and worst-fit** are the most common strategies used to select a free hole from the set of available holes.

- **First-fit:** Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
- **Best-fit:** Allocate the smallest hole that is big enough. We must search the entire list, unless the list is kept ordered by size. This strategy-produces the smallest leftover hole.
- **Worst-fit:** Allocate the largest hole. Again, we must search the entire list unless it is sorted by size. This strategy produces the largest leftover hole which may be more useful than the smaller leftover hole from a best approach.

The disadvantage of contiguous memory allocation is **fragmentation**. There are two types of fragmentation, namely, internal fragmentation and External fragmentation.

Internal fragmentation: When memory is free internally, that is inside a process but it cannot be used, we call that fragment as internal fragment. For example say a hole of size 1252 bytes is available. Let the size of the process be 1258. If the hole is allocated to this process, then six bytes are left which is not used. These six bytes which cannot be used forms the internal fragmentation.

External fragmentation: All the three dynamic storage allocation methods discussed above suffer external fragmentation. When the total memory space that is got by adding the scattered holes is sufficient to satisfy a request but it is not available contiguously, then this type of fragmentation is called external fragmentation.

The solution to this kind of external fragmentation is compaction. **Compaction** is a method by which all free memory that are scattered are placed together in one large memory block. It is to be noted that compaction cannot be done if relocation is done at compile time or assembly time. It is possible only if dynamic relocation is done, that is relocation at execution time.

One more solution to external fragmentation is to have the logical address space and physical address space to be noncontiguous. Paging and Segmentation are popular noncontiguous allocation methods.

Non-contiguous memory allocation: In the non-contiguous memory allocation the available free memory space are scattered here and there and all the free memory space is not at one place. So this is time-consuming. In the **non-**contiguous memory allocation, a process will acquire the memory space but it is not at one place it is at the different locations according to the

Process requirement. This technique of **non-**contiguous memory allocation reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.

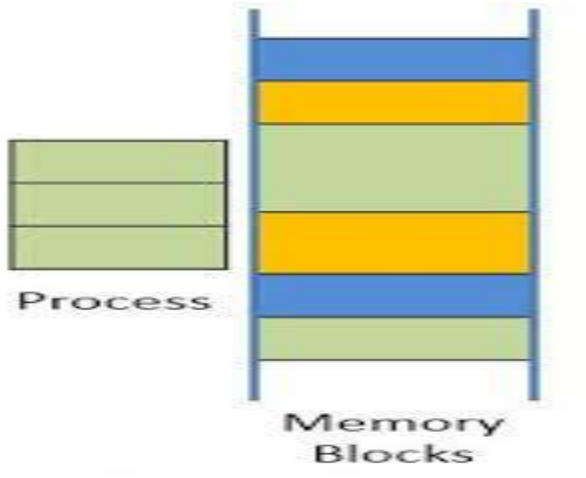

Fig 5: Non-contiguous Memory Allocation

Non-contiguous memory allocation is of different types,

- Paging
- Segmentation

Paging: Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non- contiguous.
The physical memory is divided into a number of fixed size blocks, called frames and the logical address space is also divided into fixed size blocks called pages. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is also divided into fixed size blocks that are of the same size of the frames i.e. the size of a frame is same as the size of a page for a particular hardware.

Every address generated by the CPU is divided into two parts: a page number (p) and 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 is combined with the page offset to define the physical memory address that is sent to the memory unit. The paging model of memory is shown in figure 6.
The page size like the frame size is defined by the hardware. The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical

address into a page number and page offset particularly easy. If the size of logical address is 2m , and a page size is 2n addressing units, then the high order m-n bits of a logical address designate the page number, and the n low order bits designate the page offset.

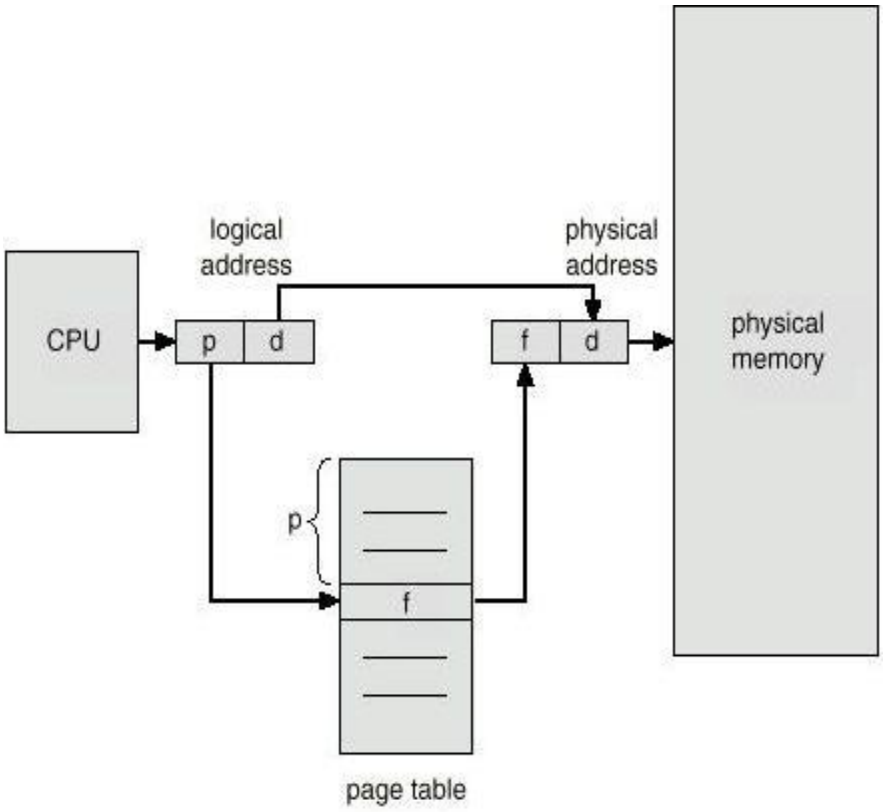

Fig 6: Paging System

Paging Example: consider the memory in figure 7. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the user's view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that the page 0 is in frame 5. Thus logical address 0 maps to physical address 20(=(5x4)+0). Logical address 3 (page 0, offset 3) maps to physical address 23(=(5x4)+3). Logical address 4 is page 1, offset 0, according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24(=(6x4)+0). Logical address 13 maps to physical address 9.

When we use a paging scheme, we have no external fragmentation. Any free frame can be allocated to a process that needs it. However, we may have some internal fragmentation.

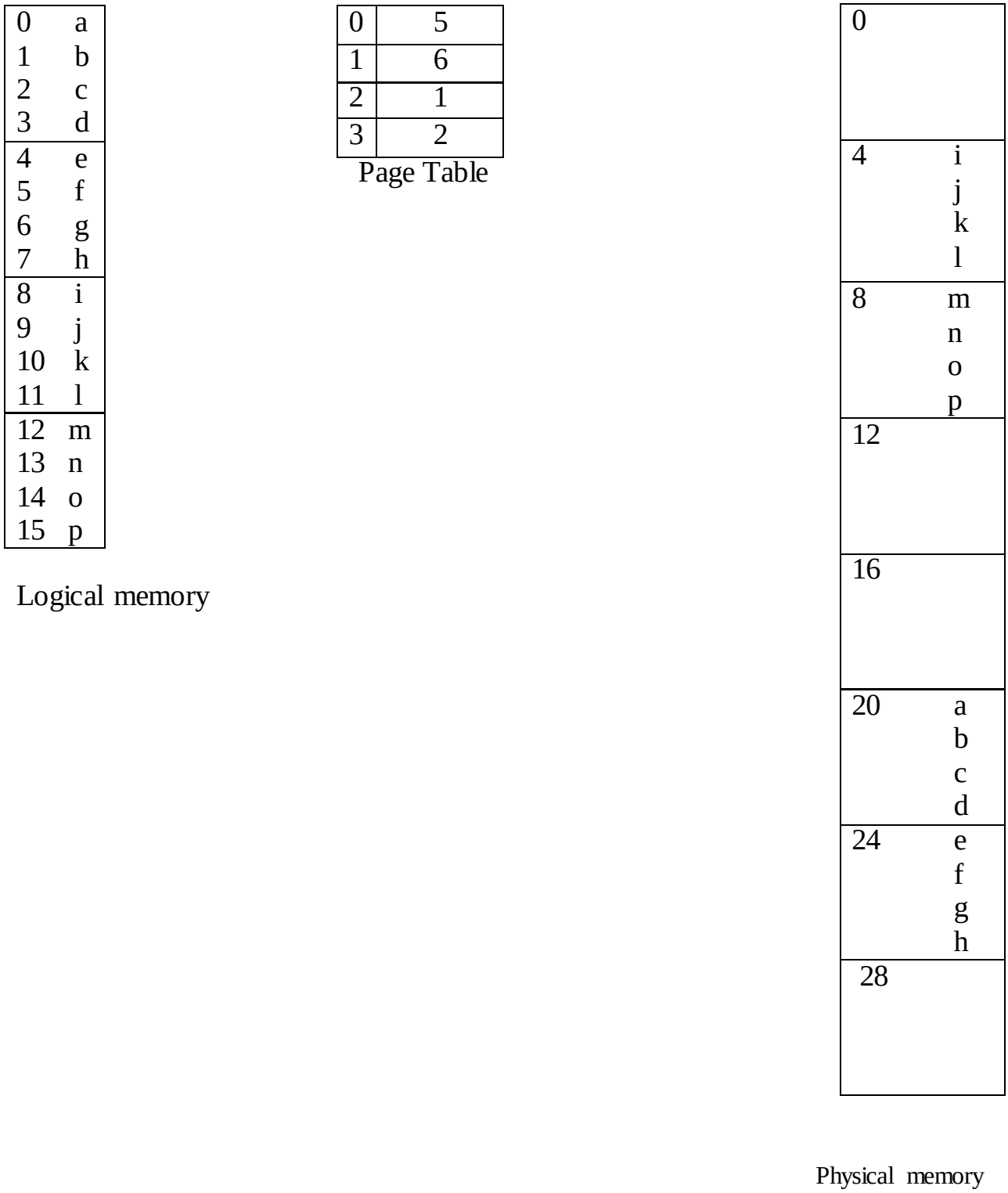

Fig 7: Paging example for a 32-byte memory with 4 byte pages

Segmentation: In Operating Systems, Segmentation is a memory management technique in which, the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process. The details about each segment are stored in a table called as segment table. Segment table is stored in one (or many) of the segments.

Segment table contains mainly two information about segment:

- Base: It is the base address of the segment
- Limit: It is the length of the segment.

Segment base contains the starting physical address where the segment resides in memory, whereas segment limit specifies the size of segment. The use of segment table is illustrated in the figure 8.

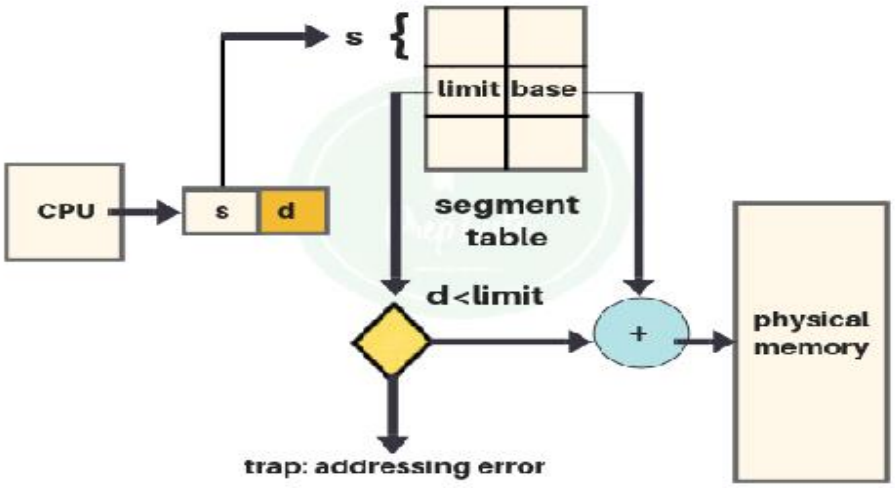

Fig 8: Segmentation Hardware

Logical address consists of two parts 's' and 'd'. The value of offset 'd' must be between 0 and the segment limit. If it is not so, then we trap to the operating system with an error message in addressing which indicates that the logical address attempt beyond the size of segment. If the offset value is legal then it is added to the segment base to produce the address in the physical memory of the desired byte.

Example: consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?

(a) 0430
(b) 110
(c) 2500
(d) 3400
(e) 4112

Sol. For the given logical address the first digit refers to the segment number (s) while remaining digits refers to the offset value (d).

(a) 0430 the first digit 0 refers to segment 0 and 430 refers to offset value for the logical address (0430). Also the size of segment 0 is 600 as shown in given table

So Physical address for logical address 0430 would be

=base + offset (430) = 219+430= 649

(b) Physical address for 110 = 2300 + 10 = 2310
(c) Physical address for 2500 = 90 + 500 = 590

But it is impossible since the segment size is 100 so it is illegal address.

(d) Physical address for 3400 = 1327 + 400 = 1727
(e) Physical address for 4112 = illegal address as the size of segment 4 (96<offset value 112)

References:

(1) Abraham Silberschatz, Galvin & Gagne, Operating System Concepts, John Wiley & Sons, INC.

(2) Harvay M.Deital, Introduction to Operating System, Addition Wesley Publication Company.

(3) Andrew S.Tanenbaum, Operating System Design and Implementation, PHI

(4) Vijay Shukla, Operating System, S.K. Kataria & Sons