Paper Name : Operating System

Prepared by : Debanjali Jana

Memory management :

Paging:

- Physical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available
    - Avoids external fragmentation
    - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
    - Size is power of 2, between 512 bytes and 16 Mbytes
- 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
- Backing store likewise split into pages
- Still have Internal fragmentation

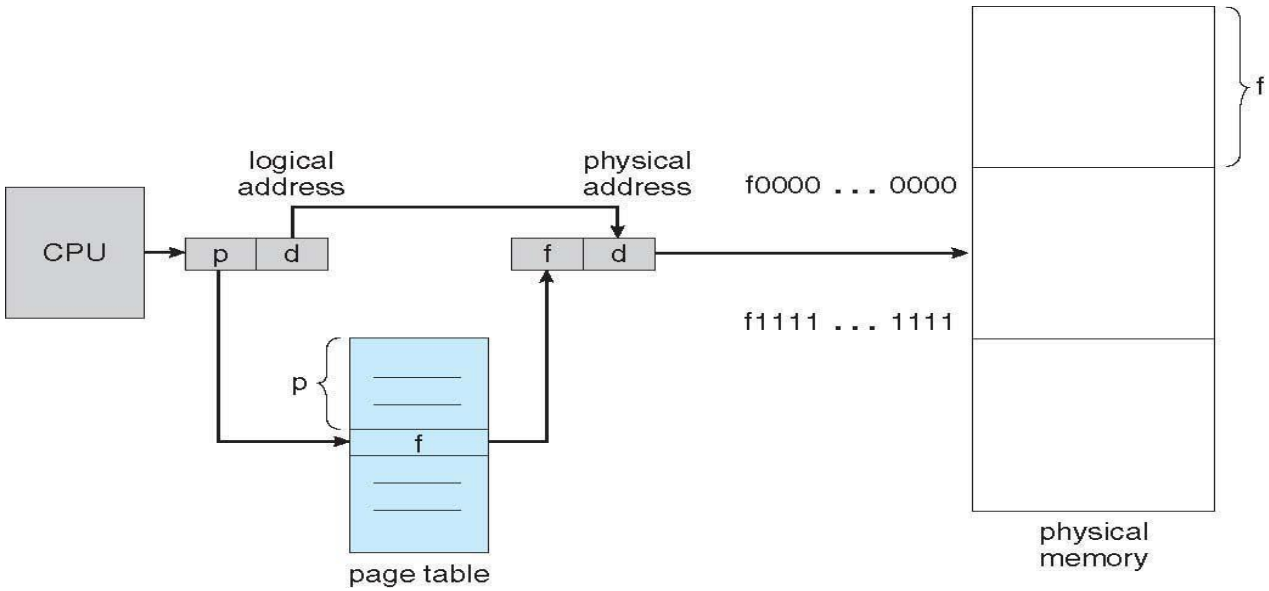
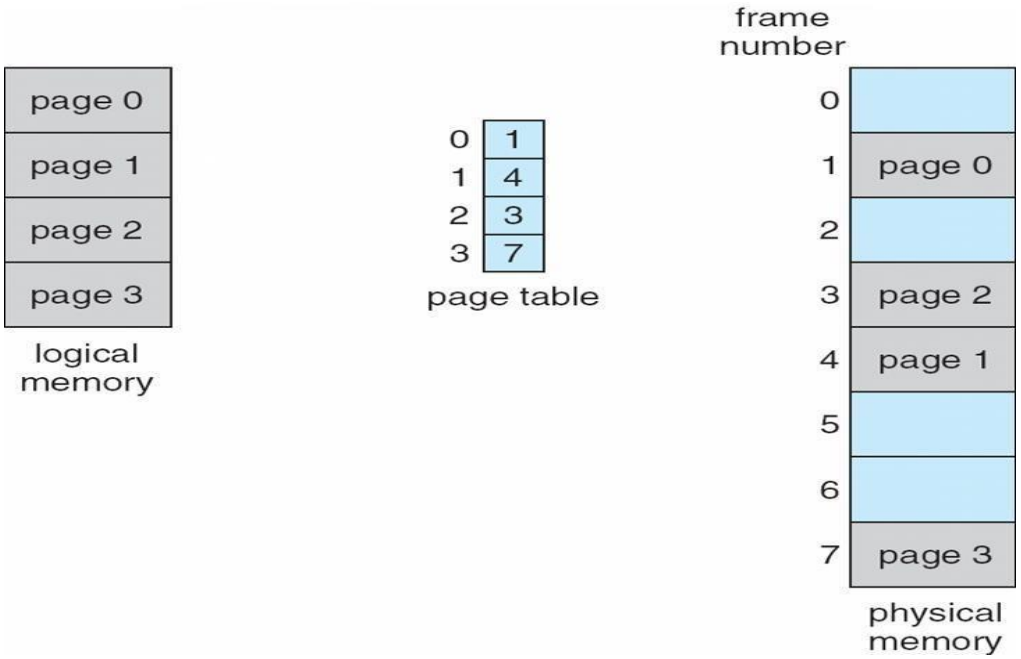


Fig : Paging hardware

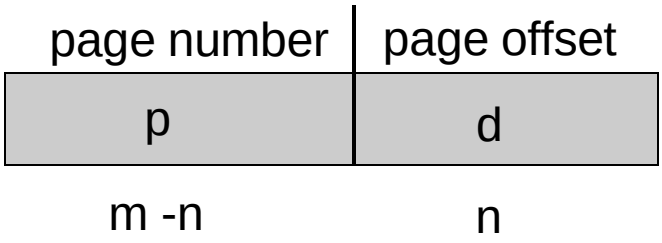
When a process arrives in the system to be executed, its size, expressed in pages, is examined. Each page of the process needs one frame. Thus, if the process requires 11 pages, at least 11 frames must be available in memory. If n frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process.

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 the logical address space is 2^m, and a page size is 2^n, addressing units (bytes or words) 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. Thus, the logical address is as follows:



Address generated by CPU is divided into:

- **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
- **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



For given logical address space 2^m and page size 2^n

where p is an index into the page table and d is the displacement within the page. Here, in the logical address, n= 2 and m = 4. 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 page 0 is in frame 5. Thus, logical address 0 maps to physical address 20 [= (5 x 4) + 0]. Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 x 4) + 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 [= (6 x 4) + O]. Logical address 13 maps to physical address 9.

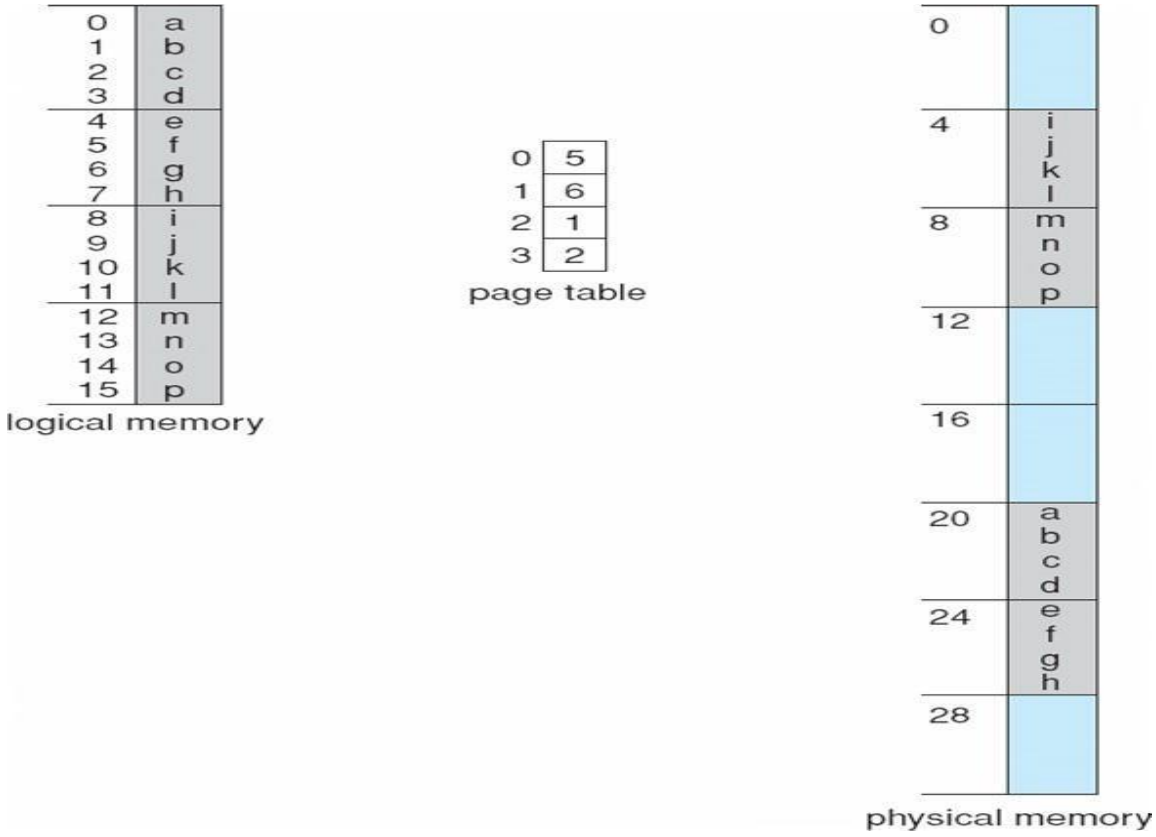


Fig: Paging example for a 32-byte memory with 4-byte pages

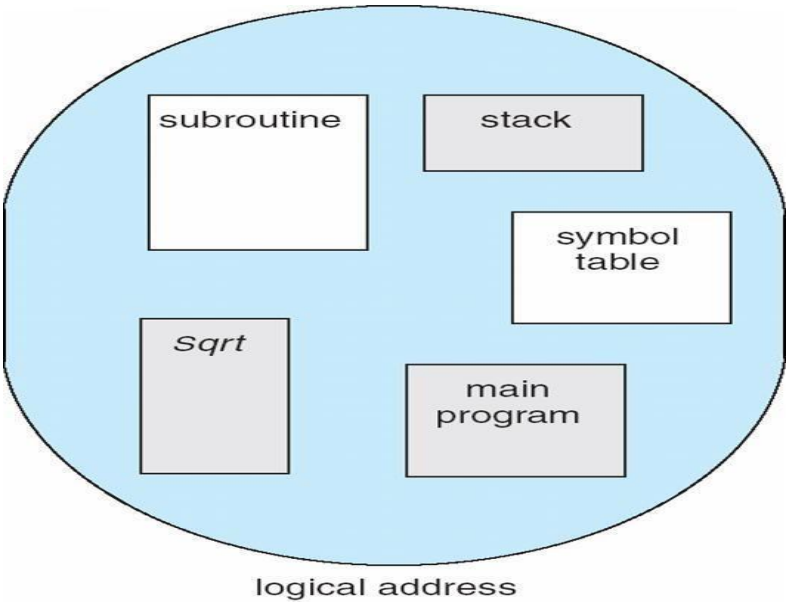
Segmentation :

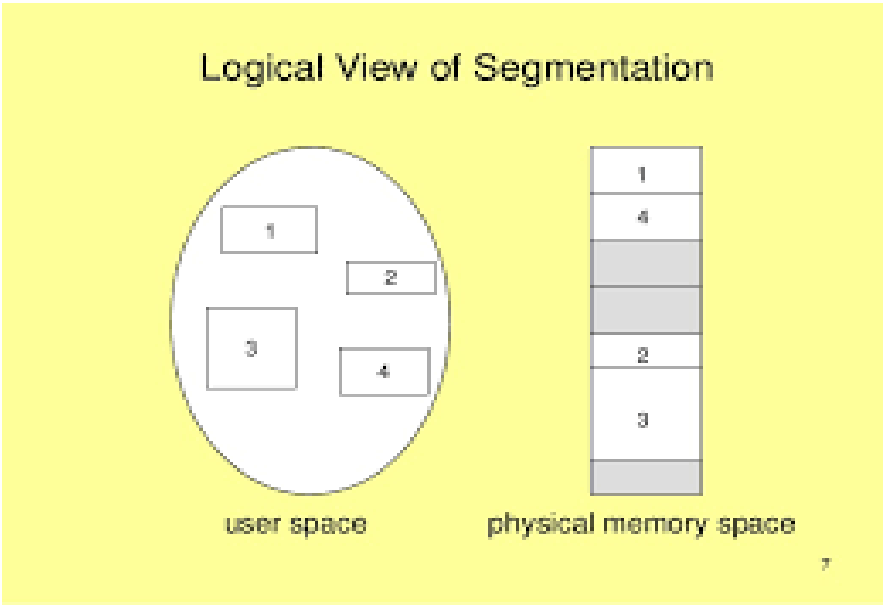
It is a memory-management scheme that supports the user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities: a segment name and an offset. (Contrast this scheme with the paging scheme, in which the user specifies only a single address, which is partitioned by the hardware into a page number and an offset, all invisible to the programmer.) 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>

A program is a collection of segments

- A segment is a logical unit such as:

main program
procedure
function
method
object
local variables, global variables
common block
stack
symbol table
arrays



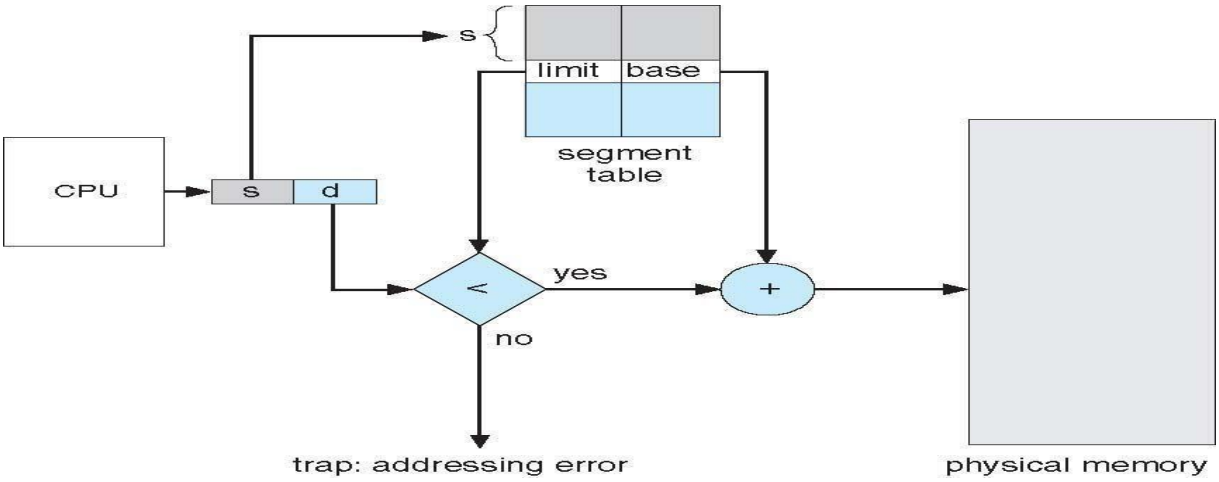


- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment

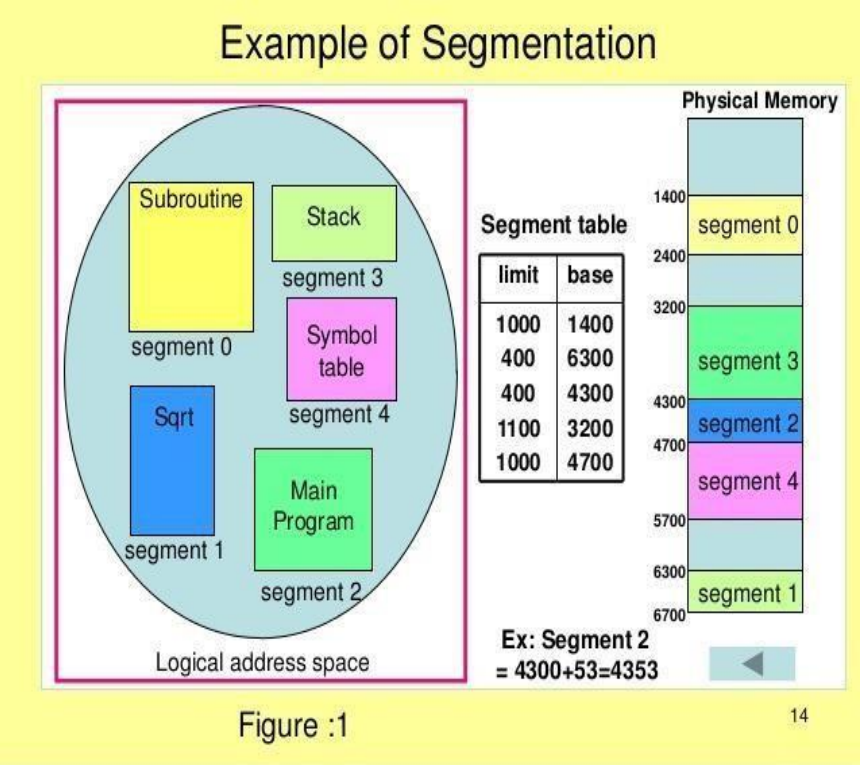
    Thus, the segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.

- **Segment-table base register (STBR)** points to the segment table's location in memory

Segmentation hardware :



A logical address consists of two parts: a segment number, s, and an offset into that segment, d. The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs. As an example, consider the situation shown below. We have five segments numbered from 0 through 4. The segments are stored in physical memory as shown. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit). For example, segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53= 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1000 bytes long.



Consider the following segment table :

Segment	Base	Limit
0	219	600
1	2300	14
2		
4	1952	96

What will the physical addresses for the logical address 0,430?

In a segmentation scheme, the generated logical address consists of two parts-

1. Segment Number
2. Segment Offset

We know-

- Segment Offset must always lie in the range [0, limit-1].

- If segment offset becomes greater than or equal to the limit of segment, then trap addressing error is produced.

Here Segment Number = 0 and Segment Offset = 430

We have,

- In the segment table, limit of segment-0 is 600.
- Thus, segment offset must always lie in the range = [0, 600-1] = [0, 599]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 219 + 430 = 649

_____