Paging:

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory by dividing total program space (logical memory) into equal size partition and also main memory into equal size partition. When the logical memory space is divided into equal size partition then it is known as page and whenever a physical memory space is divided into equal size partition then it is known as frame. The main concept of paging is that whenever a particular page is getting loaded to certain frame there should be no any free space remaining. Page size is equal to frame size.

The following figure shows the general concept of the paging:

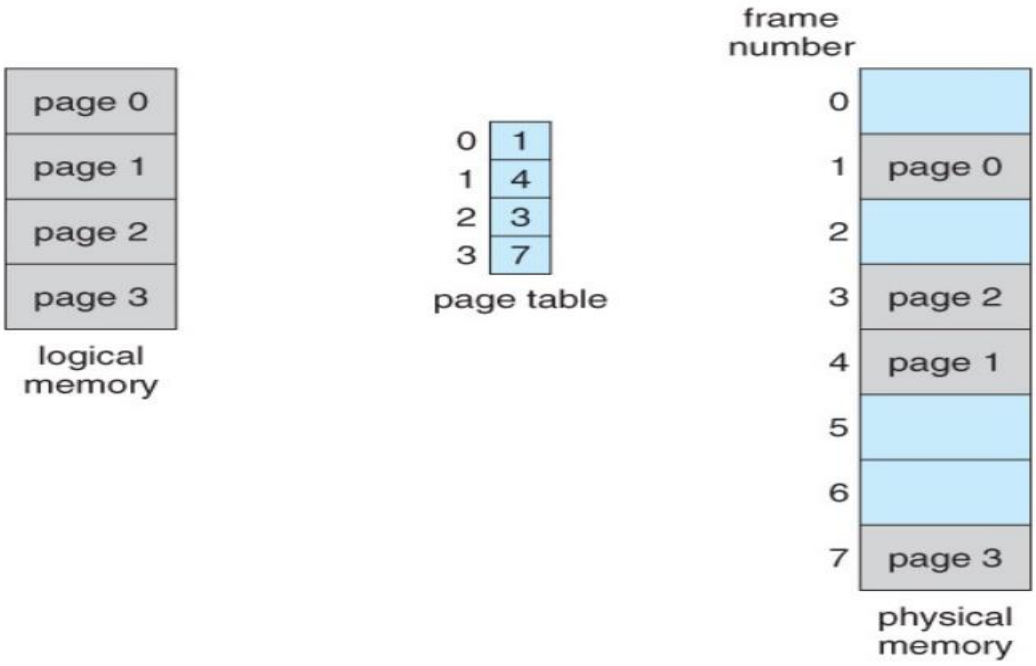


Figure: paging model of logical and physical memory

In above figure, logical memory is divided into fixed size multiple page like page 0, page 1, page 2 and page 3. Such page number is used as index on page table to find out the frame number. Page table is showing that page number 0 is in frame1. Similarly page number 2 is in frame 3.

Working procedure:

The hardware support for paging is illustrated in following figure.

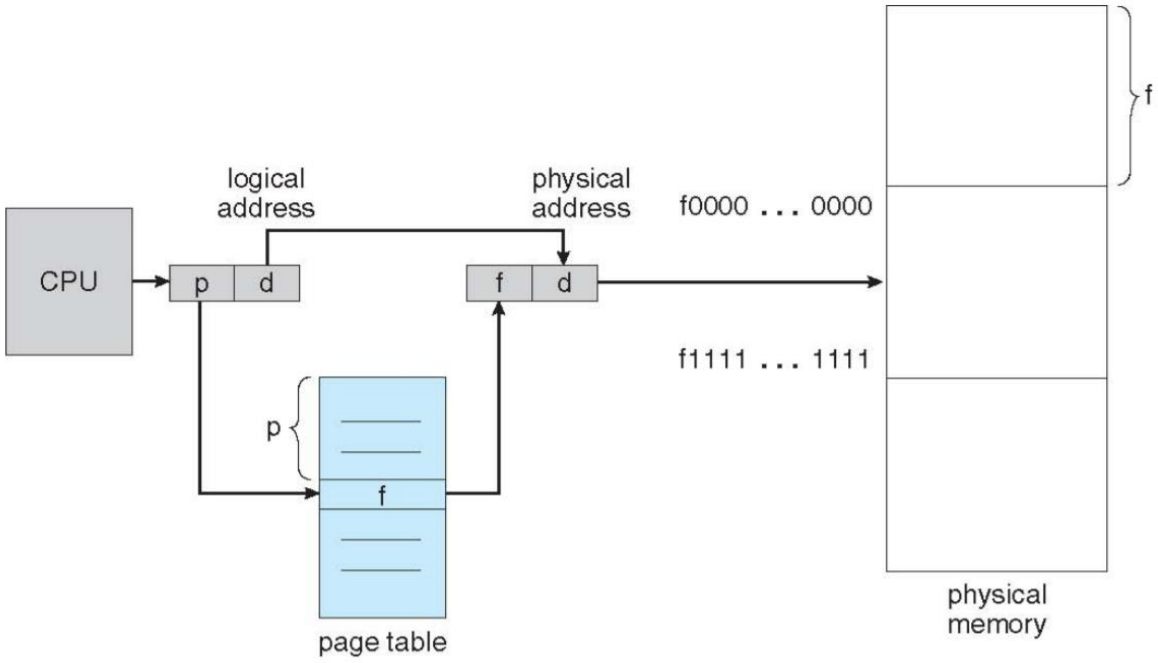


Fig: paging hardware

Here every address generated by the CPU is divided into two parts: a page number (p) and a page offset (d). Page offset determines how many parts or location does each page contains. The page number is used as an index into a page table. The page size is defined by the hardware. The size of page is power of 2 and makes the translation of a logical address into a page number and page offset particularly easy. If the size of the logical address is 2^m and a page size is 2^n bytes then the high order bit m-n forms page number and low order bit n forms offset which is shown in following figure:

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.

Example of paging:

Let us consider, in the logical address n=2 and m = 4. Then:

Logical memory's size = $2^m = 2^4 = 16$ byte

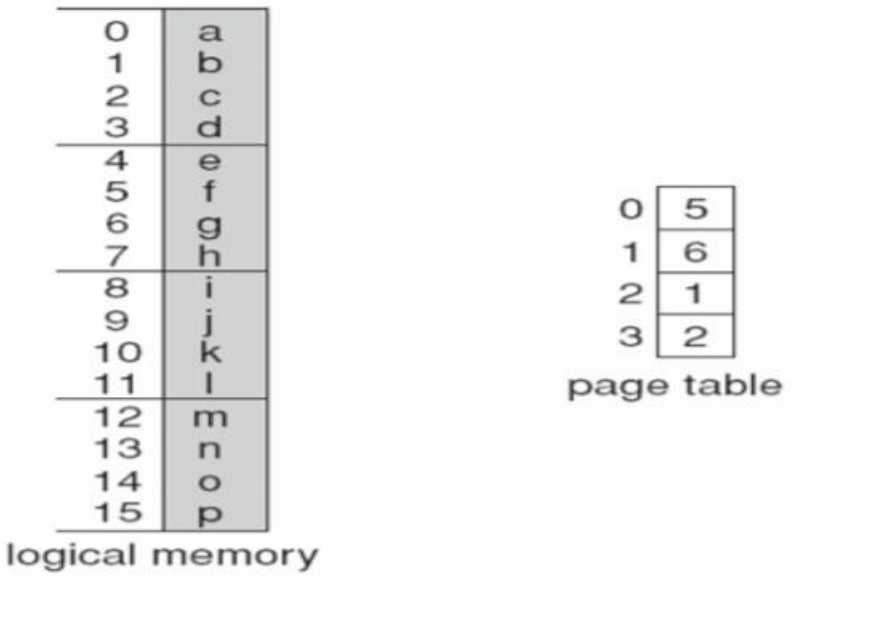
Page size = $2^n = 22 = 4$ byte.

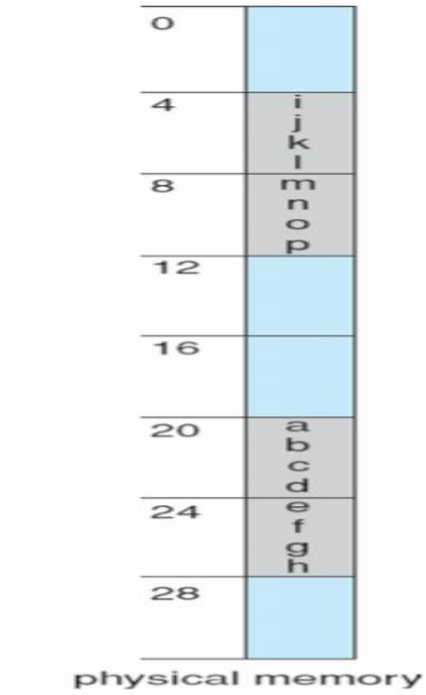
Page number (p) = m-n = 4-2 = 2 i.e. $2^2 = 4$ [0 – 3 page number]

Page offset (d) = n = 2 i.e. $2^2 = 4$byte [every 4 location or partition of logical memory is equal to one page number] i.e. page no. 0 contain location 0-3. Similarly, page no. 0 contain location 4-7 and so on.

Physical memory size = 32 byte = 8 frame (2^8) and 1 frame = 4 location (0-3) same of page size.

The following figure shows an example:





In logical memory: location 0-3 is in page no. 0, 4-7 is in page 1, 8-11 is in page 2 and 12 to 15 is in page no. 3. Page table indicates that page 0 is in frame 5, page 1 is in frame 6 and so on.

Now suppose CPU generates logical address of 0 then this logical address 0 is in page 0 and offset 0. Indexing into page table we find that page 0 is in frame 5. This logical address 0 maps to physical address by = [frame number * page size + page offset] = 5 * 4 +0 = 20 i.e. logical address 0 maps to physical address 20.

Next time suppose CPU generates logical address 5 then this logical address 5 is in page 1 and offset is 2. Indexing into page table we find that page 1 is in frame 6. This logical address 5 maps to physical address by = [frame number * page size + page offset] = 6 * 4 + 2 = 26 i.e. logical address 0 maps to physical address 26.

Structure of page table / types:

1. **Hierarchical paging:**

   Most modern computer system support a large logical address space (2^32 to 2^64). In such an environment the page table itself becomes excessively large. One simple solution to this problem is to divide the page table into smaller pieces. One way is to use a two level paging algorithm in which the page table is also paged as shown in figure below. For e.g. let us consider a system with 32 bit logical address space and a page size of 4KB. A logical address is divided into a page number consisting of 20 bits and a page offset consisting of 12 bits.

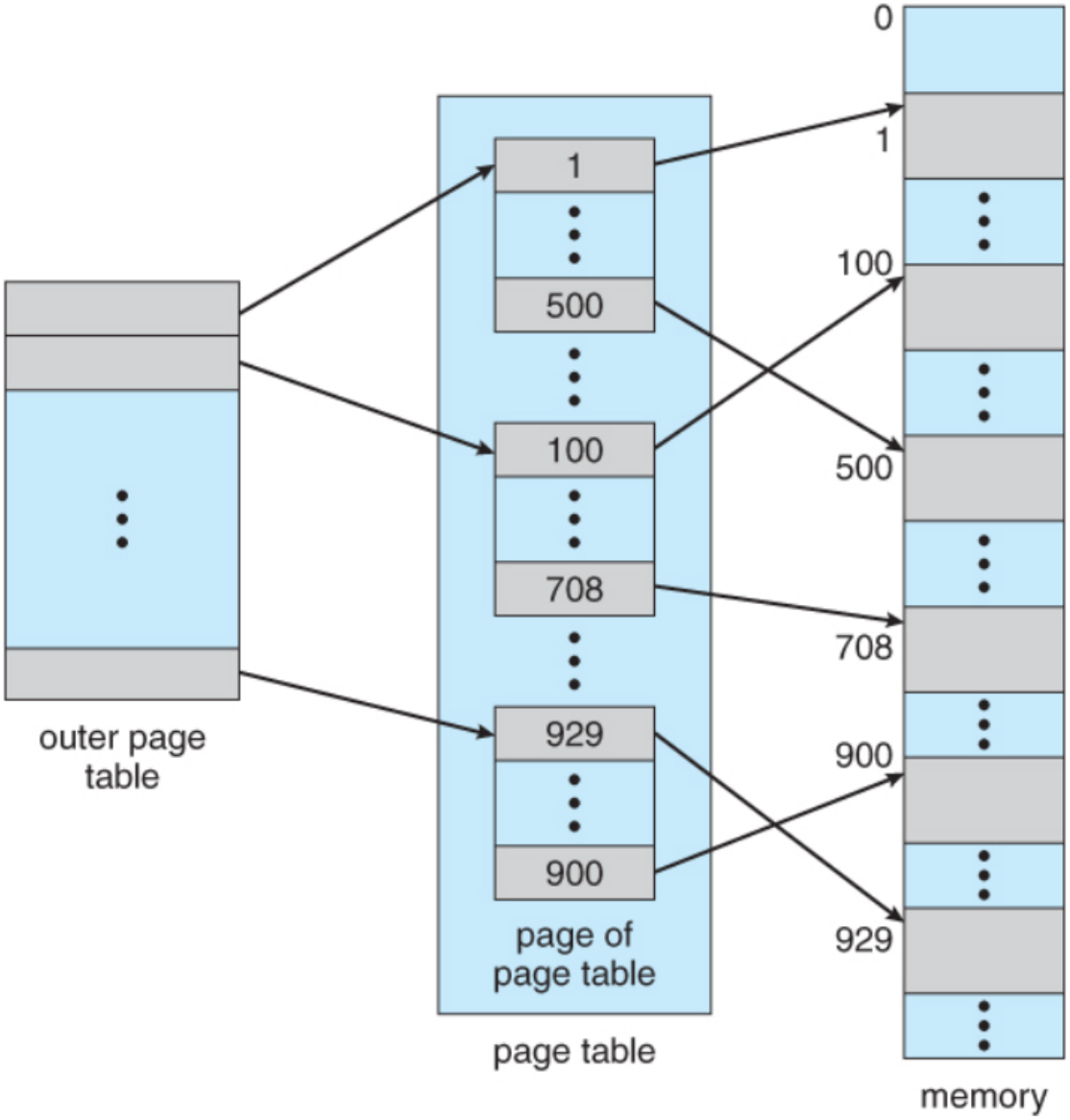
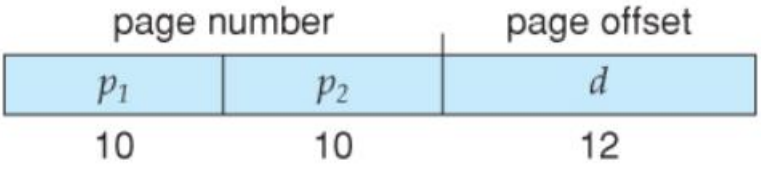


Figure: a two level page table scheme

As we page the page table, the page number is further divided into a 10 bit page number and a 10 bit page offset. Thus a logical address is as follows:



P1 is an index into the outer page table and p2 is the displacement within the page of the inner page table. The first identifies an entry in the outer page table which identifies where in memory to find one page of an inner page table. The second 10 bits finds a specific entry in that inner page table which in turn identifies a particular frame in physical memory. The address translation is shown by following figure:

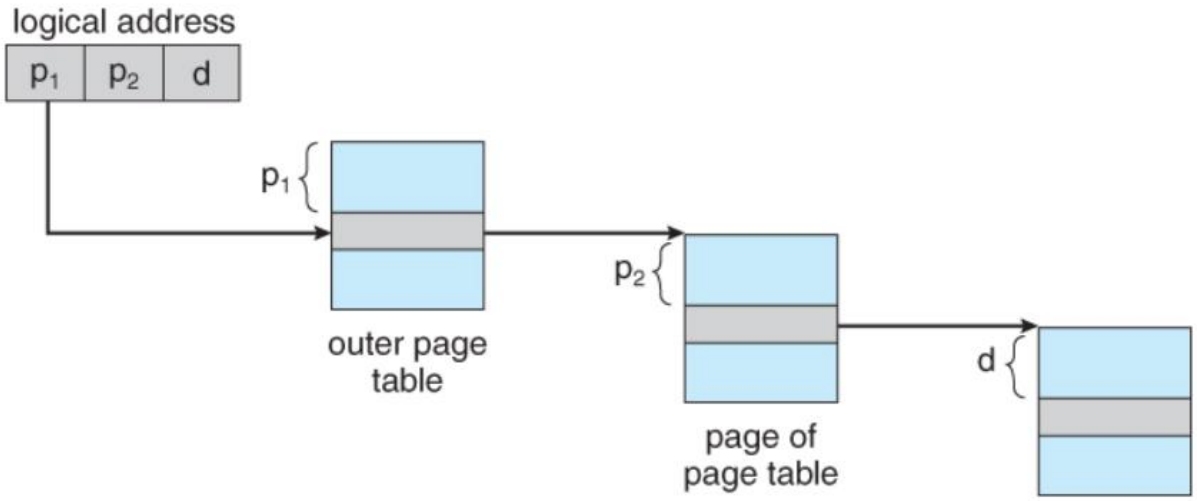


Figure: address translation

2. Hashed page table:

If the address space is larger than 32 bits is to use a hashed page table with the hash value being the virtual page number. Each entry in the hash table contains a linked list of element that hash to the same location. Each element consist of three fields: virtual page number, the value of the mapped page frame and a pointer to the next element in the linked list.

This work as: the virtual page number in the virtual address is hashed into the hash table. Such virtual page number is compared with field 1 in the first element in the linked list. If there is a match the corresponding page frame (field 2) is used to form the desired physical address. If there is no

match then entries in the linked list are searched for a matching virtual page number.