Virtual Memory

Virtual Memory: Basic Concepts

Virtual memory is a technique that allows the execution of processes that may not be completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.
There are two types of addresses in virtual memory systems: those referenced by processes (virtual addresses) and those available in main memory (physical or real addresses).
Virtual memory systems contain special-purpose hardware called the memory management unit (MMU) that quickly maps virtual addresses to physical addresses.
A key to implementing virtual memory systems is how to map virtual addresses to physical addresses as process execute. Dynamic address translation (DAT) mechanisms convert virtual addresses to physical addresses during execution.

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 broken into fixed-sized blocks called frames. Logical memory is also 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. The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.
The hardware support for paging is illustrated in Figure (1). Every address generated by the CPU is divided into two parts:

- **Page number (p):** Number of bits required to represent the pages in Logical Address Space.
- **Page offset (d):** Number of bits required to represent particular word in a page or page size of Logical Address Space.

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.

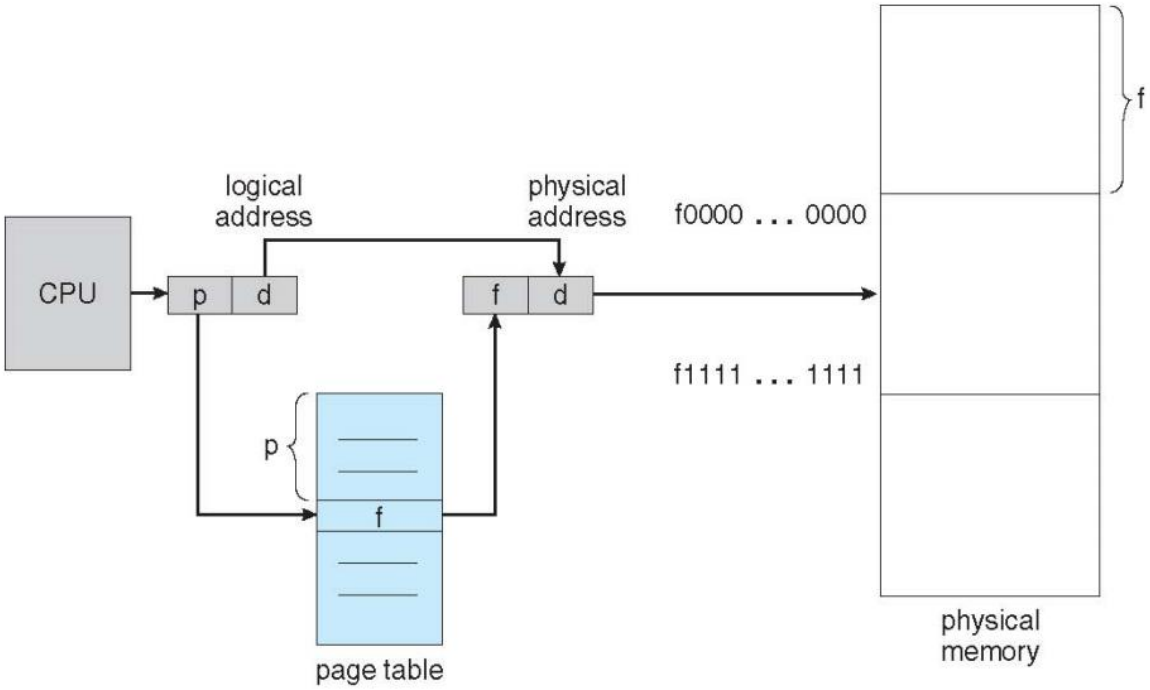

Figure 1: Paging hardware

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.

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB (translation Look-aside buffer), a special, small, fast look up hardware cache. The TLB is used with page tables in the following way.

The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory. The whole task may take less than 10 percent longer than it would if an unmapped memory reference were used.

If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory. In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference.

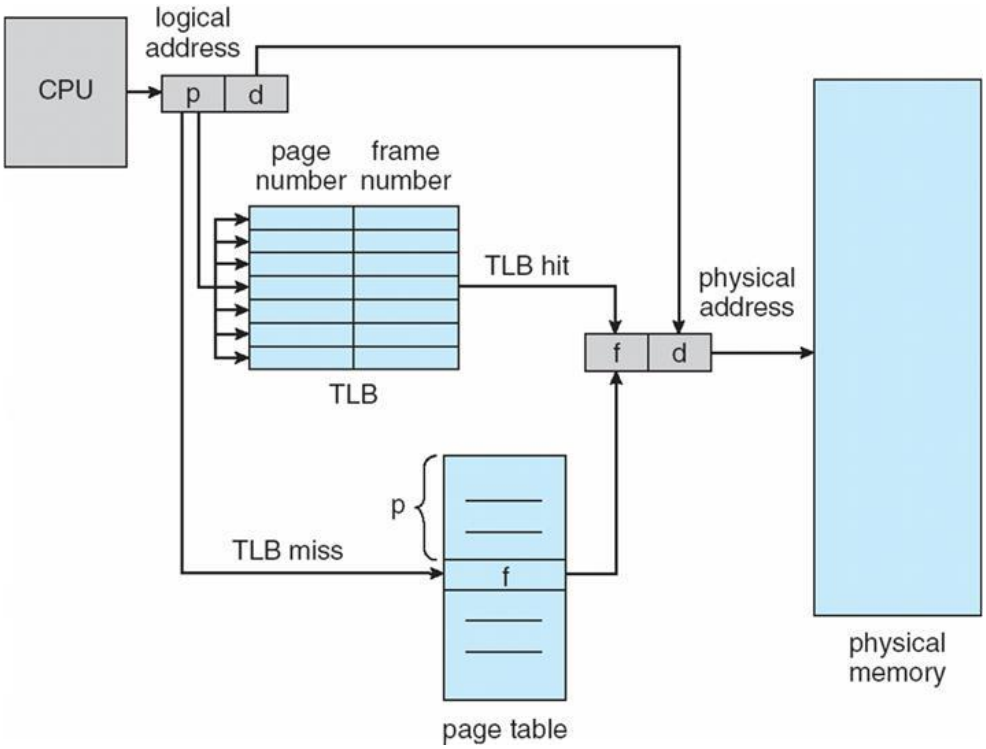

Figure 2: Paging hardware with TLB

Protection

Memory protection in a paged environment is accomplished by protection bits that are associated with each frame. Normally, these bits are kept in the page table. One bit can define a page to be read-write or read-only. Every reference to memory goes through the page table to find the correct frame number. At the same time that the physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read-only page. An attempt to write to a read-only page causes a hardware trap to the operating system (or memory-protection violation). We can easily expand this approach to provide a finer level of protection. We can create hardware to provide read-only, read-write, or execute-only protection. Or, by providing separate protection bits for

each kind of access, we can allow any combination of these accesses; illegal attempts will be trapped to the operating system.

One more bit is generally attached to each entry in the page table: a valid-invalid bit. When this bit is set to "valid," this value indicates that the associated page is in the process' logical-address space, and is thus a legal (or valid) page. If the bit is set to "invalid", this value indicates that the page is not in the process' logical-address space. Illegal addresses are trapped by using the valid-invalid bit. The operating system sets this bit for each page to allow or disallow accesses to that page.

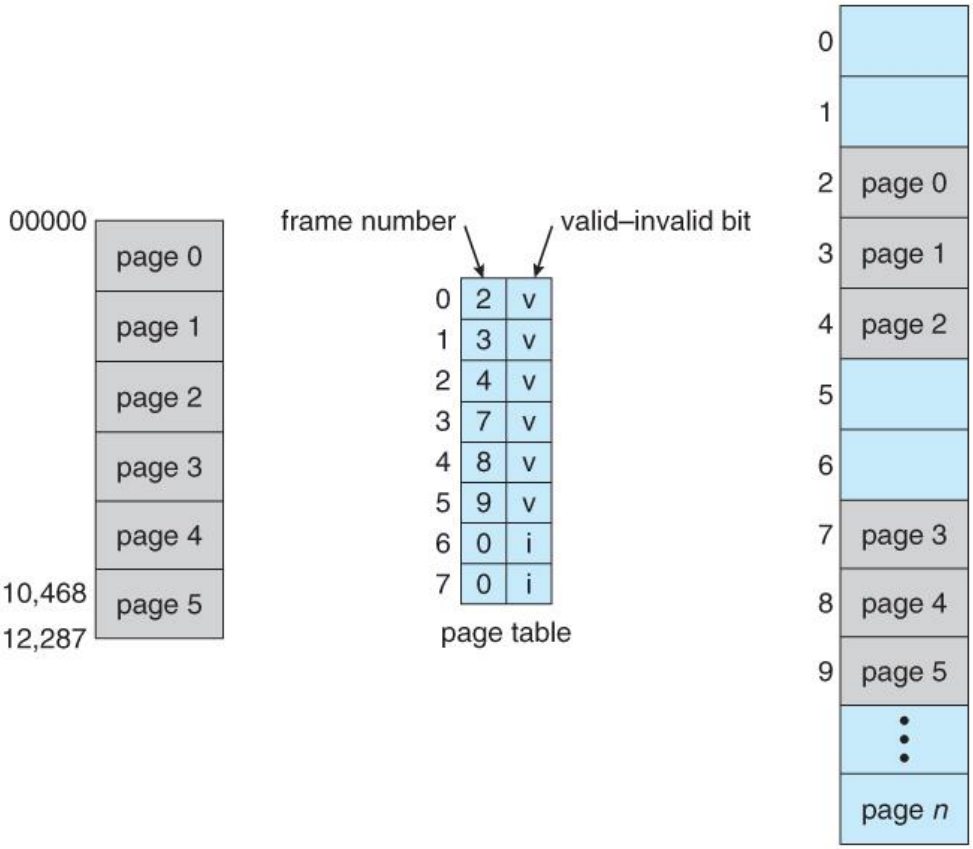

Figure 3: Valid (v) or invalid (i) bit in a page table.