Effective access Time

Each operating system has its own methods for storing page tables. Most allocate a page table for each process. A pointer to the page table is stored with the other register values (like the instruction counter) in the process control block. The hardware implementation of the page table can be done in several ways. In the simplest case, the page table is implemented as a set of dedicated registers. These registers should be built with very high-speed logic to make the paging-address translation efficient. The use of registers for the page table is satisfactory if the page table is reasonably small (for example, 256 entries).

The standard solution to this problem is to use a special, small, fastlookup hardware cache, called translation look-aside buffer **(TLB).** The TLB is associative, high-speed memory. Each entry in the TLB consists of two parts: a key (or tag) and a value. When the associative memory is presented with an item, it is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast; the hardware, however, is expensive. Typically, the number of entries in a TLB is small, often numbering between 64 and 1,024.

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 (Figure below). In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, the operating system must select one for replacement. Replacement policies range from least recently used (LRU) to random. Furthermore, some TLBs allow entries to be wired down, meaning that they cannot be removed from the TLB. Typically, TLB entries for kernel code are often wired down.

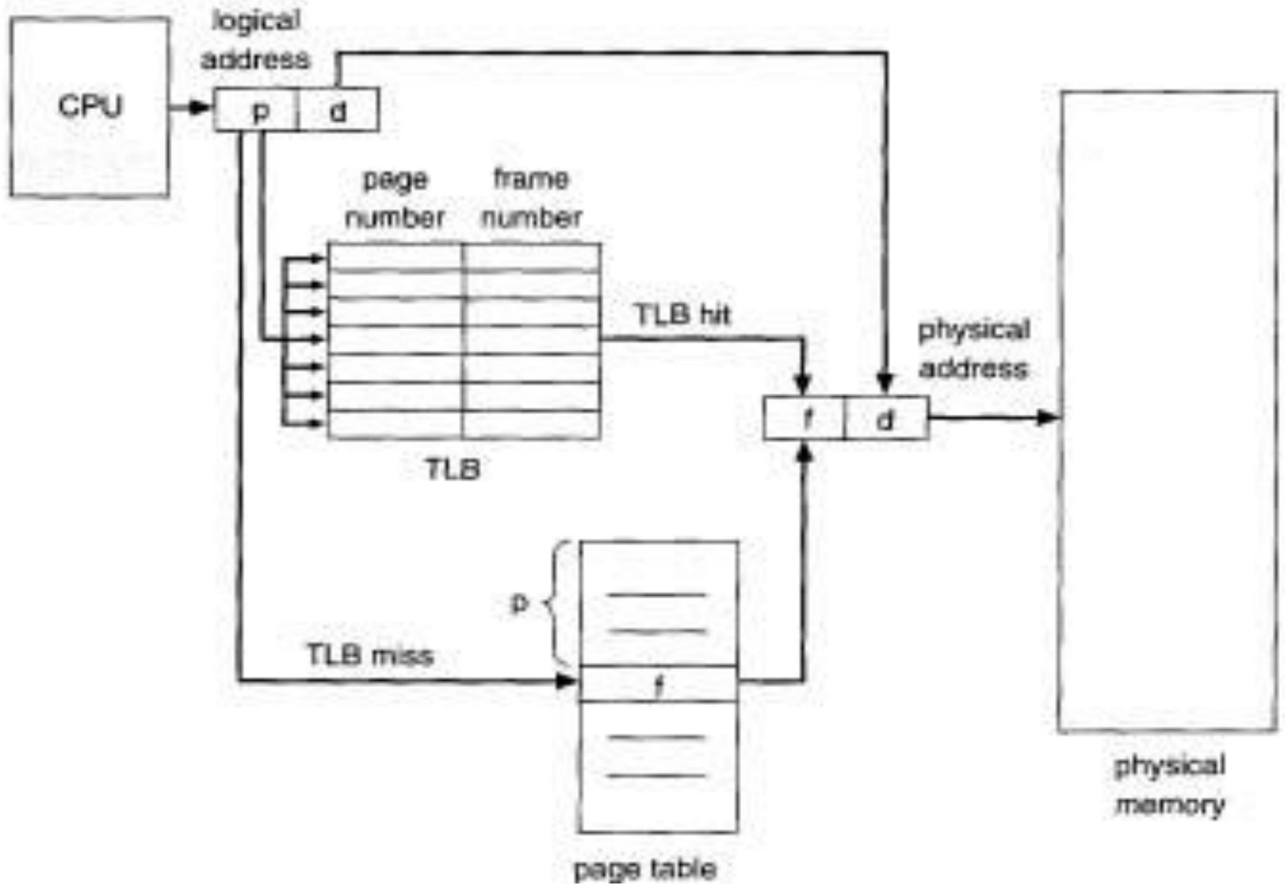

Figure: Paging hardware with TLB.

The percentage of times that a particular page number is found in the TLB is called the **hit ratio.** An 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time. If it takes 20 nanoseconds to search the TLB, and 100 nanoseconds to access memory, then a mapped memory access takes 120 nanoseconds when the page number is in the TLB.

If we fail to find the page number in the TLB (20 nanoseconds), then we must first access memory for the page table and frame number (100 nanoseconds), and then access the desired byte in memory (100 nanoseconds), for a total of 220 nanoseconds. To find the **effective memory-access time,** we must weigh each case by its probability:

EAT(effective access time)= P x hit memory time + (1-P) x miss memory time.

effective access time = 0.80 x 120 + 0.20 x 220

= 140 nanoseconds.

In this example, we suffer a 40-percent slowdown in memory access time (from 100 to 140 nanoseconds).

Q1. For a 98-percent hit ratio, we have

EAT(effective access time)= P x hit memory time + (1-P) x miss memory time.
Where: P is Hit ratio.
effective access time = 0.98 x 120 + 0.02 x 220
= 122 nanoseconds.
This increased hit rate produces only a 22-percent slowdown in access time.

Q2. What will be the EAT if hit ratio is 70%, time for TLB is 30ns and access to main memory is 90ns?

Sol. P = 70% = 70/100 = **0.7**

Hit memory time = 30ns + 90ns = **120ns**

Miss memory time = 30ns + 90ns + 90ns = **210ns**

Therefore, EAT = P x Hit + (1-P) x Miss

= 0.7 x 120 + 0.3 x 210

=840 + 63.0

=**147 ns**