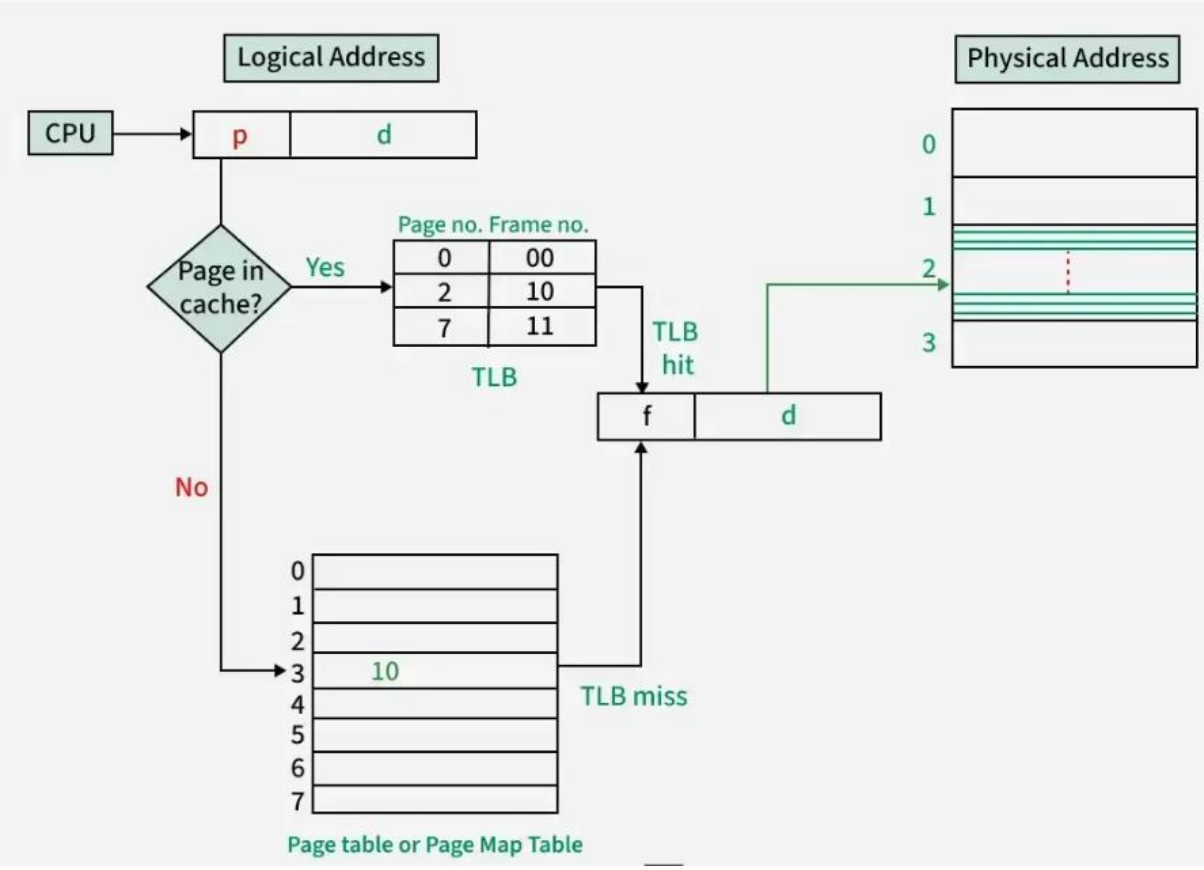
Translation Lookaside Buffer (TLB) in Paging

Paging allows efficient mapping between virtual addresses and the physical addresses in main memory. Each process has its own Page Table, which stores Page Table Entries (PTEs) that map virtual page numbers to physical frame numbers.



TLB in Paging

Now the question is where to place the page table, such that overall access time (or reference time) will be less.

The Problem of Access Time in Paging

When the CPU generates a virtual address, it must be translated into a physical address to access data in main memory. The naive approach stores the entire page table in main memory. Therefore, for every memory access, the following two memory accesses are required:

1. Access the page table in main memory to get the frame number.
2. Access the actual data in the main memory frame.

Note: *This leads to a performance problem, as every memory access now requires two memory accesses, causing a significant slowdown.*

Why Not Store Page Table in Registers?

Initially, it seemed ideal to store the page table in the high-speed CPU registers to speed up lookups since registers provide the fastest access time. However:

- Registers are limited in size and can store only a small number of entries (usually between 0.5k to 1k PTEs).
- For large processes, the required page table size can be huge (e.g., 1 million entries for a 32-bit address space with 4 KB pages).

Therefore, this approach is impractical and the entire page table is kept in main memory instead.

- To find the frame number
- To go to the address specified by frame number

How Does the TLB Work?

When the CPU generates a virtual address, it first looks up the page number in the TLB.

- **TLB Hit:** It is the situation when, the page number is found in the TLB, the corresponding frame number is retrieved instantly.
- **TLB Miss:** It is the situation when, the page number is not found in the TLB, the CPU accesses the page table in main memory.

Step-by-Step Process

Steps in TLB Hit:

1. CPU generates a virtual (logical) address.
2. It is checked in TLB (present).
3. The corresponding frame number is retrieved, which now tells where the main memory page lies.

Steps in TLB miss:

1. CPU generates a virtual (logical) address.
2. It is checked in TLB (not present).
3. Now the page number is matched to the page table residing in the main memory (assuming the page table contains all PTE).

4. The corresponding frame number is retrieved, which now tells where the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement techniques comes into the picture i.e either FIFO, LRU or MFU etc).

Effective memory access time(EMAT)

TLB is used to reduce adequate memory access time as it is a high-speed associative cache.

EMAT=h×(c+m)+(1−h)×(c+n×m)$EMAT = h \times (c+m) + (1-h) \times (c+n \times m)$

where:

- h is the hit ratio of the TLB,
- m is the memory access time,
- c is the TLB access time and
- n represents the system level.

Note: *A higher hit ratio hhh significantly reduces EMAT, making the system efficient.*

Pros of Using TLB in Paging

- **Faster Address Translation**: Reduces the need for repeated page table accesses.
- **Better Performance**: Reduces average memory access time dramatically.
- **Efficient Memory Utilization**: Only the most frequently used PTEs are cached in the small, fast TLB.
- **Scalability**: Especially useful for large address spaces in 64-bit architectures.