Chapter 19: Paging: Faster Translations (TLBs)

This chapter addresses the critical performance problem introduced by paging. While paging offers great flexibility, it requires an extra memory lookup for the page table on every single memory reference, which is prohibitively slow. The solution is a hardware mechanism called the **Translation Lookaside Buffer (TLB)**.

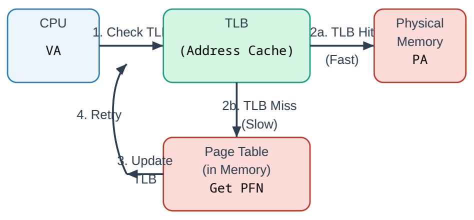
> **The Crux of the Problem: How to Speed Up Address Translation?**
>
> How can we speed up address translation and avoid the extra memory reference that paging seems to require? What hardware support is required? What OS involvement is needed?

1. The Translation Lookaside Buffer (TLB)

The TLB is a hardware **cache** for address translations. It is a small, fast memory located within the CPU's Memory Management Unit (MMU) that stores recent, popular virtual-to-physical page translations.

- **Core Idea:** Before accessing the page table in main memory, the hardware first checks the TLB.
  - If the translation is found in the TLB (a **TLB hit**), the physical address is formed immediately, and the slow memory access to the page table is avoided.
  - If the translation is not in the TLB (a **TLB miss**), the hardware (or OS) must access the page table to find the translation, update the TLB with the new entry, and then retry the instruction.
- **Goal:** To make the common case fast. Because programs exhibit **locality** (both temporal and spatial), most memory references will hopefully result in a TLB hit.

Address Translation with a TLB



2. Example: Accessing an Array

Let's trace the TLB activity for a program looping through an array of 10 integers. Assume the array starts at virtual address 100 and the page size is 16 bytes.

- The array elements will be spread across three virtual pages:
  - `a[0]`, `a[1]`, `a[2]` are on **VPN 6**.
  - `a[3]`, `a[4]`, `a[5]`, `a[6]` are on **VPN 7**.
  - `a[7]`, `a[8]`, `a[9]` are on **VPN 8**.
- **Access Trace:**
  1. Access `a[0]` (VPN 6): **TLB Miss**. The translation for VPN 6 is fetched from the page table and loaded into the TLB.
  2. Access `a[1]` (VPN 6): **TLB Hit**. The translation is already in the TLB.
  3. Access `a[2]` (VPN 6): **TLB Hit**.
  4. Access `a[3]` (VPN 7): **TLB Miss**. A new page is accessed. The translation for VPN 7 is loaded into the TLB.
  5. Access `a[4]-a[6]` (VPN 7): **TLB Hit, Hit, Hit**.
  6. Access `a[7]` (VPN 8): **TLB Miss**. The translation for VPN 8 is loaded.
  7. Access `a[8]-a[9]` (VPN 8): **TLB Hit, Hit**.
- **Result:** Out of 10 memory accesses, there were 3 misses and 7 hits (a 70% hit rate). This demonstrates the power of **spatial locality**. Even on the first pass, the TLB significantly improves performance. On subsequent passes over the array, all accesses would be hits due to **temporal locality**.

---

3. Who Handles TLB Misses?

There are two main approaches for handling a TLB miss.

- **Hardware-Managed TLB (CISC-style):**
  - The hardware has complete knowledge of the page table's location and format.
  - On a miss, the hardware itself "walks" the page table in memory, finds the correct PTE, updates the TLB, and retries the instruction.
  - This is common in Complex Instruction Set Computer (CISC) architectures like **x86**.
- **Software-Managed TLB (RISC-style):**
  - The hardware's role is minimal. On a miss, it simply raises an exception (a trap) and transfers control to the OS.
  - An OS **trap handler** is responsible for looking up the translation in its page table data structure, updating the TLB using privileged instructions, and returning from the trap.
  - The hardware then retries the original instruction, which now results in a TLB hit.

    - This is common in Reduced Instruction Set Computer (RISC) architectures like **MIPS** and **SPARC**.
    - **Advantage:** This approach offers **flexibility**, as the OS can use any page table structure it wants without requiring hardware changes.

---

4. TLB Contents and Context Switching

4.1. What's in a TLB Entry?

A typical TLB entry contains:

- **VPN (Virtual Page Number)**
- **PFN (Physical Frame Number)**
- **Valid Bit:** Is this a valid entry?
- **Protection Bits:** Read, Write, Execute permissions.
- **Dirty Bit:** Has the page been written to?
- **Address Space Identifier (ASID):** A field to identify which process this translation belongs to.

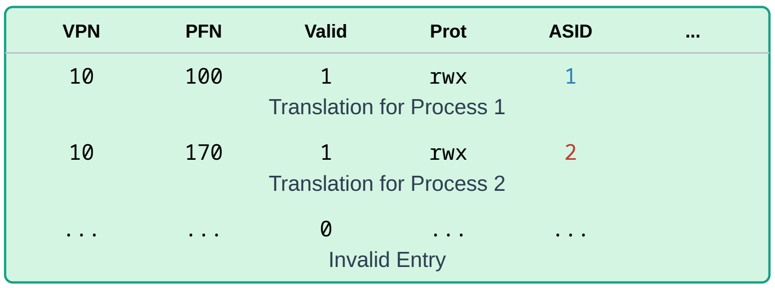
4.2. The Context Switch Problem

The translations in the TLB are specific to the currently running process. When the OS switches to a new process, these old translations are invalid and could cause the new process to access the wrong memory.

> **The Crux of the Problem: How to Manage TLB Contents on a Context Switch?**
>
> When context-switching between processes, the translations in the TLB for the last process are not meaningful to the about-to-be-run process. What should the hardware or OS do to solve this problem?

- **Solution 1: Flush the TLB:**
    - On every context switch, the OS issues a privileged instruction to flush the entire TLB, setting all valid bits to 0.
    - **Pro:** Simple and guarantees correctness.
    - **Con:** High performance cost. The new process will suffer a storm of TLB misses as it starts running.
- **Solution 2: Use an Address Space Identifier (ASID):**
    - The hardware includes an **ASID** field in each TLB entry. The OS assigns a unique ASID to each process.
    - On a context switch, the OS tells the hardware the ASID of the new process.
    - The hardware then matches both the VPN *and* the current ASID to find a translation.
    - **Pro:** Allows translations for different processes to co-exist in the TLB, avoiding a flush and reducing miss rates.

TLB with Address Space Identifier (ASID)



VPN	PFN	Valid	Prot	ASID	...
10	100	1	rwx	1	
		Translation for Process 1			
10	170	1	rwx	2	
		Translation for Process 2			
...	...	0	...	...	
		Invalid Entry			

The ASID allows translations for different processes to coexist in the TLB.

---

5. TLB Replacement Policy

When the TLB is full and a new entry must be added, an old one must be evicted.

- **Goal:** The replacement policy should aim to minimize the miss rate.
- **Common Policies:**
  - **Least-Recently-Used (LRU):** Evicts the entry that has not been used for the longest time. Good for exploiting locality.
  - **Random:** Evicts a random entry. Simple and avoids worst-case performance scenarios that can plague LRU.

---

6. Summary

The **Translation Lookaside Buffer (TLB)** is a hardware cache that makes paging performant. By storing popular virtual-to-physical address translations, it avoids the slow trip to the page table in main memory for most memory accesses.

- The effectiveness of the TLB depends on program **locality**.
- Handling **TLB misses** can be done by hardware or software (OS).
- On a **context switch**, the TLB must be managed, either by flushing it or by using **ASIDs** to distinguish between processes.
- Like any cache, the TLB needs a **replacement policy** to decide which entry to evict when it's full.

While the TLB is essential, it is not a perfect solution. Programs that access more pages than can fit in the TLB will suffer from poor performance, a phenomenon known as **exceeding the TLB coverage**.