Faster Paging with TLBs

Chapter 19

Previously in CS212…

- Paging is a flexible means of allocating memory
  - Fixed sized blocks means no external fragmentation
  - If we “outgrow” a page, we can allocate another page for more storage

- Page Tables are:
  - Per process
  - Large and need to be stored in memory
  - Memory access is slow, but needed for each data/instruction address translation to locate the page in physical memory

- We need a way to do this better!

Translation-lookaside buffer (TLB)

- Located on the CPU's memory-management unit (MMU)
- A hardware cache of popular virtual-to-physical address translations
  - Smaller storage space compared to primary memory (RAM), but faster
- Each time a virtual memory request is made, the hardware checks the TLB first to see if we already know that translation
  - If the translation is present, we don't need to check the page table

The Basics*

- A request is made for a virtual address
  - Extract the VPN (virtual page number)
  - Check the TLB to see if we know the associated PFN (page frame number)

- If we find the translation, we have a **TLB hit**
  - append the PFN from the TLB to the virtual address offset bits to generate a physical address

- If we don’t find the translation, we have a **TLB miss**
  - Look up the VPN in the page table and store the value in the TLB if it is valid
  - Check the TLB again

*Assumes linear page table and hardware based TLB management

Why is it faster?

- The TLB is located near the CPU, so access is faster than primary memory (raw speed)
- Caching common translations on average should produce more hits than misses and reduce the expensive page table lookup in primary memory
- Leverages the Principles of:
  - Spatial locality – related things may be near each other
  - Temporal locality – things recently access may be needed again soon

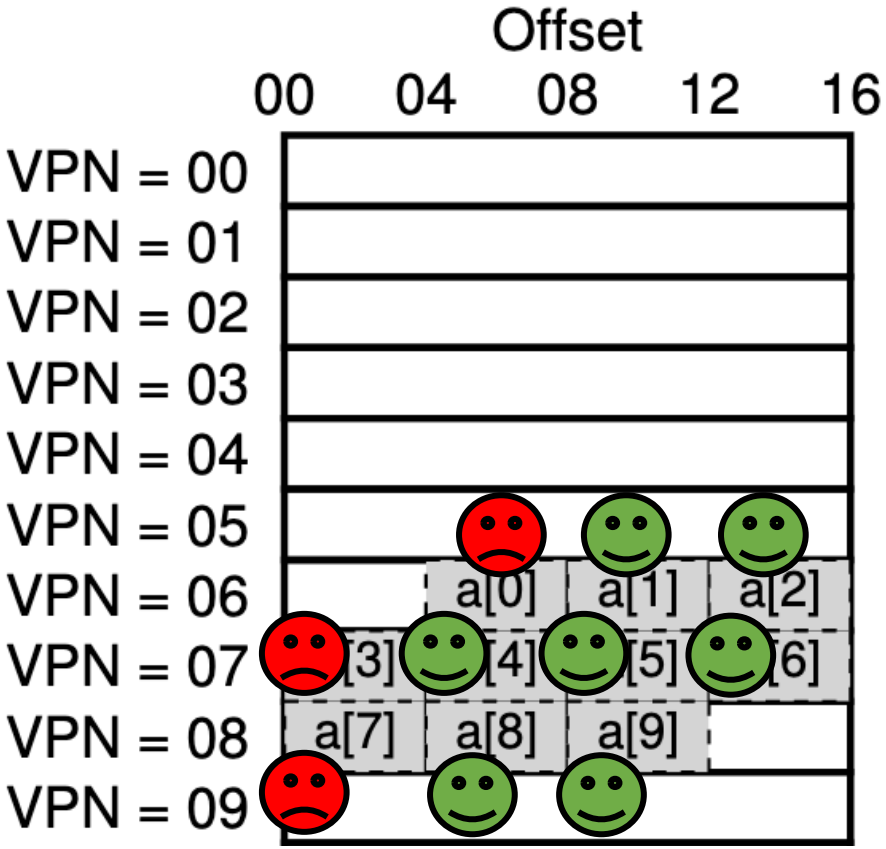
Example

- Our first request to array index 0 is a TLB Miss. We look up the PFN and cache it for VPN 06
- The subsequent two access are in the same page, so we have TLB hits
- VPN 7 is a miss the first time, but the next three access are hits
- The process repeats for VPN 08
- 3 misses and 7 hits
  - Hit rate = 70%

```
int sum = 0;
for (i = 0; i < 10; i++) {
    sum += a[i];
}
```

TLB Cache:

- 06
- 07
- 08



A Swing and a TBL Miss

- Either hardware or software can respond to a TLB miss
- Hardware requires a page-table base register and must know the *exact format* of the table in memory
  - Less trust in the OS
- Software uses the LDE concept to raise an exception, raise privilege, and jump to trap handler code to find the translation in the page table and update the TLB
  - Simplicity and flexibility
- Note that the TLB is checked twice on a miss. What is the implication to the LDE process?

Challenges

- Context Switching
  - Page table address translations are per process!
    - Should we flush the TLB when we switch process?
    - Save an address space identifier (ASID) to link the translation to a process?
- Cache Replacement Policy
  - We can’t hold everything; how do we decide what to keep and what to purge from the cache?
    - Least-recently-used (LRU), random, etc.
- TLB Coverage
  - If a program requests more unique pages than the TLB can hold, we will start to generate many misses and lose performance

Next Time…

- We've found a way to improve performance for page tables with TLB caching, but we haven't addressed the amount of storage required for page tables in memory

- We will look at a more complex variation of page tables to help mitigate that issue