EC 513
Computer Architecture

Modern Virtual Memory and Virtualization

Prof. Michel A. Kinsky
Address Translation

Virtual Address

TLB Lookup

Page Table Walk

Protection Check

Update TLB

Page Fault (OS loads page)

Protection Fault

Segment Fault

Where?

Hardware

Hardware or software

Software

Physical Address (to cache)

hit

miss

is in memory

is not in memory

denied

permitted
Page Fault Handler

- When the referenced page is not in DRAM:
  - The missing page is located (or created)
  - It is brought in from disk, and page table is updated
    - Another job may be run on the CPU while the first job waits for the requested page to be read from disk
  - If no free pages are left, a page is swapped out
    - Approximate LRU replacement policy
- Since it takes a long time to transfer a page (msecs), page faults are handled completely in software by the OS
  - Untranslated addressing mode is essential to allow kernel to access page tables
Translation for Page Tables

- Can references to page tables cause TLB misses?

- User VA translation causes a TLB miss
- Page table walk: User PTE Base and appropriate bits from VA are used to obtain virtual address VP for page table entry
- Get a TLB miss when we try to translate VP
When we get a TLB miss on VP translation, OS adds System PTE Base to bits from VP to find physical address of page table entry for VP.
Swapping a Page of a Page Table

- A PTE in primary memory contains primary or secondary memory addresses
- A PTE in secondary memory contains only secondary memory addresses
- A page of a PT can be swapped out only if none of its PTE’s point to pages in the primary memory
Software handlers need a restartable exception on page fault or protection violation.

Handling a TLB miss needs a hardware or software mechanism to refill TLB.
Address Translation in CPU Pipeline

- Need mechanisms to cope with the additional latency of a TLB
  - Slow down the clock
  - Pipeline the TLB and cache access
  - Virtual address caches
  - Parallel TLB/cache access
Physical or Virtual Address Caches?

- One-step process in case of a hit (+)
- Cache needs to be flushed on a context switch unless address space identifiers (ASIDs) included in tags (-)
- Aliasing problems due to the sharing of pages (-)

Alternative: place the cache before the TLB

Diagram:

- CPU → Virtual Cache → TLB → Primary Memory
- CPU → TLB → Physical Cache → Primary Memory
Aliasing in Virtual-Address Caches

**General Solution:** Disallow aliases to coexist in cache

Two virtual pages share one physical page.

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!
Aliasing in Virtual-Address Caches

- Software (i.e., OS) solution for direct-mapped cache
  - VAs of shared pages must agree in cache index bits; this ensures all VAs accessing same PA will conflict in direct-mapped cache (early SPARC)

Virtual cache can have two copies of same physical data. Writes to one copy not visible to reads of other!
Concurrent Access to TLB & Cache

- Index L is available without consulting the TLB
- Cache and TLB accesses can begin simultaneously
Concurrent Access to TLB & Cache

- Tag comparison is made after both accesses are completed
  - Cases: \( L + b = k \) \( L + b < k \) \( L + b > k \) what happens here?
  - \( L + b > k \): Partially VA cache. But it may be more effective to increase the way of the cache and decrease the index bits for same cache capacity
After the PPN is known, W physical tags are compared.

- Allows cache size to be greater than $2^{L+b}$ bytes.
Anti-Aliasing Using L2

- Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2)
  - After VA2 is resolved to PA, collision is detected in L2
  - Collision: VA1 will be purged from L1 and L2, and VA2 will be loaded → no aliasing!

Field a is different.
Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1
Virtual Memory Use Today

- Desktops/servers have full demand-paged virtual memory
  - Portability between machines with different memory sizes
  - Protection between multiple users or multiple tasks
  - Share small physical memory among active tasks
  - Simplifies implementation of some OS features
Virtual Memory Use Today

- Vector supercomputers have translation and protection but not demand-paging (Older Crays: base&bound, Japanese & Cray X1: pages)
  - Do not waste expensive CPU time thrashing to disk (make jobs fit in memory)
  - Mostly run in batch mode (run set of jobs that fits in memory)
  - Difficult to implement restartable vector instructions
Virtual Memory Use Today

- Most embedded processors and DSPs provide physical addressing only
  - Cannot afford area/speed/power budget for virtual memory support
  - Often there is no secondary storage to swap to!
  - Programs custom written for particular memory configuration in product
  - Difficult to implement restartable instructions for exposed architectures
Next Class

- Synchronization and Sequential Consistency