Last time in Lecture 9

- Protection and translation required for multiprogramming
  - Base and bounds was early simple scheme
- Page-based translation and protection avoids need for memory compaction, easy allocation by OS
  - But need to indirect in large page table on every access
- Address spaces accessed sparsely
  - Can use multi-level page table to hold translation/protection information, but implies multiple memory accesses per reference
- Address space access with locality
  - Can use “translation lookaside buffer” (TLB) to cache address translations (sometimes known as address translation cache)
  - Still have to walk page tables on TLB miss, can be hardware or software talk
- Virtual memory uses DRAM as a “cache” of disk memory, allows very cheap main memory
Memory Management

- Can separate into orthogonal functions:
  - Translation (mapping of virtual address to physical address)
  - Protection (permission to access word in memory)
  - Virtual memory (transparent extension of memory space using slower disk or flash storage)
- But most modern systems provide support for all the above functions with a single page-based system

Modern Virtual Memory Systems

*Illusion of a large, private, uniform store*

Protection & Privacy
- several users, each with their private address space and one or more shared address spaces
- page table = name space

Demand Paging
- Provides the ability to run programs larger than the primary memory
- Hides differences in machine configurations

*The price is address translation on each memory reference*
Hierarchical Page Table

Virtual Address
31 22 21 12 11 0

p1 p2 offset

10-bit 10-bit
L1 index L2 index

Root of Current Page Table
(Processor Register)

Level 1 Page Table

Level 2 Page Tables

Data Pages

- page in primary memory
- page in secondary memory
- PTE of a nonexistent page

Page-Based Virtual-Memory Machine
(Hardware Page-Table Walk)

- Assumes page tables held in untranslated physical memory
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
    - Pseudo-LRU replacement policy, implemented in software

- 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
Handling VM-related exceptions

- Handling a TLB miss needs a hardware or software mechanism to refill TLB
- Handling a page fault (e.g., page is on disk) needs a restartable exception so software handler can resume after retrieving page
  - Precise exceptions are easy to restart
  - Can be imprecise but restartable, but this complicates OS software
- Handling protection violation may abort process
  - But often handled the same as a page fault

Address Translation in CPU Pipeline

- Need to cope with additional latency of TLB:
  - slow down the clock?
  - pipeline the TLB and cache access?
  - virtual address caches
  - parallel TLB/cache access
Virtual-Address Caches

Alternative: place the cache before the TLB

- 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 (-)
- maintaining cache coherence (-)  (see later in course)

Virtually Addressed Cache (Virtual Index/Virtual Tag)

Translate on miss
**Aliasing in Virtual-Address Caches**

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!

General Solution: *Prevent aliases coexisting in cache*

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)

---

**Concurrent Access to TLB & Cache (Virtual Index/Physical Tag)**

Index L is available without consulting the TLB

⇒ *cache and TLB accesses can begin simultaneously!*

Tag comparison is made after both accesses are completed

*Cases:* $L + b = k$, $L + b < k$, $L + b > k$
Virtual-Index Physical-Tag Caches:  
**Associative Organization**

After the PPN is known, $2^a$ physical tags are compared

*How does this scheme scale to larger caches?*

---

Concurrent Access to TLB & Large L1

The problem with $L_1 > \text{Page size}$

*Can VA$_1$ and VA$_2$ both map to PA?*
CS152 Administrivia

- PS 2 and Lab 2 out
- Short time frame – only one week left till due so start soon
- Quiz 2, Tuesday March 5
  - Lectures 6-9, PS 2, Lab 2, readings

A solution via Second Level Cache

Usually a common L2 cache backs up both Instruction and Data L1 caches

L2 is “inclusive” of both Instruction and Data caches
- Inclusive means L2 has copy of any line in either L1
Suppose VA1 and VA2 both map to PA and VA1 is already in L1, L2 (VA1 ≠ VA2).
After VA2 is resolved to PA, a collision will be detected in L2.
VA1 will be purged from L1 and L2, and VA2 will be loaded ⇒ no aliasing!
Atlas Revisited

- One PAR for each physical page

- PAR’s contain the VPN’s of the pages resident in primary memory

- Advantage: The size is proportional to the size of the primary memory

- What is the disadvantage?

Hashed Page Table: Approximating Associative Addressing

- Hashed Page Table is typically 2 to 3 times larger than the number of PPN’s to reduce collision probability

- It can also contain DPN’s for some non-resident pages (not common)

- If a translation cannot be resolved in this table then the software consults a data structure that has an entry for every existing page (e.g., full page table)
Power PC: Hashed Page Table

- Each hash table slot has 8 PTE's <VPN,PPN> that are searched sequentially
- If the first hash slot fails, an alternate hash function is used to look in another slot
  
  *All these steps are done in hardware!*
- Hashed Table is typically 2 to 3 times larger than the number of physical pages
- The full backup Page Table is managed in software

VM features track historical uses:

- **Bare machine, only physical addresses**
  - One program owned entire machine
- **Batch-style multiprogramming**
  - Several programs sharing CPU while waiting for I/O
  - Base & bound: translation and protection between programs (supports swapping entire programs but not demand-paged virtual memory)
  - Problem with external fragmentation (holes in memory), needed occasional memory defragmentation as new jobs arrived
- **Time sharing**
  - More interactive programs, waiting for user. Also, more jobs/second.
  - Motivated move to fixed-size page translation and protection, no external fragmentation (but now internal fragmentation, wasted bytes in page)
  - Motivated adoption of virtual memory to allow more jobs to share limited physical memory resources while holding working set in memory
- **Virtual Machine Monitors**
  - Run multiple operating systems on one machine
  - Idea from 1970s IBM mainframes, now common on laptops
    - e.g., run Windows on top of Mac OS X
  - Hardware support for two levels of translation/protection
    - Guest OS virtual -> Guest OS physical -> Host machine physical
Virtual Memory Use Today - 1

- Servers/desktops/laptops/smartphones 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
- Vector supercomputers have translation and protection but rarely complete demand-paging
  - (Older Crays: base&bound, Japanese & Cray X1/X2: pages)
    - Don’t 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

2/21/2013

Virtual Memory Use Today - 2

- Most embedded processors and DSPs provide physical addressing only
  - Can’t 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

2/21/2013
Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252