Modern Virtual Memory Systems

Nathan Beckmann
Computer Science and Artificial Intelligence Laboratory
M.I.T.

http://www.csg.csail.mit.edu/6.823
From last time...

- Virtual memory gives programs the illusion of a large, private memory
Address Translation: putting it all together

Virtual Address → TLB Lookup

- TLB Lookup
  - miss
  - hit

- Page Table Walk
  - the page is \(\notin\) memory
  - the page is \(\in\) memory

- Protection Check
  - denied
  - permitted

- Page Fault (OS loads page)
- Update TLB
- Protection Fault (SEGFAULT)

Where? → Physical Address (to cache)
Topics

- Interrupts

- Speeding up the common case:
  - TLB & Cache organization

- Speeding up page table walks

- Modern Usage
Interrupts: altering the normal flow of control

An external or internal event that needs to be processed by another (system) program. The event is usually unexpected or rare from program’s point of view.
Causes of Interrupts

Interrupt: an event that requests the attention of the processor

- Asynchronous: an external event
  - input/output device service-request
  - timer expiration
  - power disruptions, hardware failure

- Synchronous: an internal event (a.k.a exceptions)
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception
  - misaligned memory access
  - virtual memory exceptions: page faults, TLB misses, protection violations
  - traps: system calls, e.g., jumps into kernel
Asynchronous Interrupts: invoking the interrupt handler

- An I/O device requests attention by asserting one of the prioritized interrupt request lines

- When the processor decides to process the interrupt
  - It stops the current program at instruction $I_i$, completing all the instructions up to $I_{i-1}$ (precise interrupt)
  - It saves the PC of instruction $I_i$ in a special register (EPC)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode
Interrupt Handler

- Saves EPC before enabling interrupts to allow nested interrupts ⇒
  - need an instruction to move EPC into GPRs
  - need a way to mask further interrupts at least until EPC can be saved
- Needs to read a *status register* that indicates the cause of the interrupt
- Uses a special indirect jump instruction RFE *(return-from-exception)* which
  - enables interrupts
  - restores the processor to the user mode
  - restores hardware status and control state
Synchronous Interrupts

• A synchronous interrupt (exception) is caused by a *particular instruction*

• In general, the instruction cannot be completed and needs to be *restarted* after the exception has been handled
  – requires undoing the effect of one or more partially executed instructions

• In case of a trap (system call), the instruction is considered to have been completed
  – a special jump instruction involving a change to privileged kernel mode
Exception Handling 5-Stage Pipeline

- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?
Exception Handling 5-Stage Pipeline

- **PC**: Instruction Memory
- **D**: Decode
- **E**: Execution
- **M**: Memory
- **W**: Writeback

- **Illegal Opcode**
- **Overflow**
- **Data address Exceptions**
- **Asynchronous Interrupts**

- **Select Handler PC**
- **Kill F Stage**
- **Kill D Stage**
- **Kill E Stage**
- **Asynchronous Interrupts**

**Commit Point**

**EPC**

**Kill Writeback**
Exception Handling  5-Stage Pipeline

- Hold exception flags in pipeline until commit point (M stage)

- Exceptions in earlier pipe stages override later exceptions *for a given instruction*

- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

- Inject external interrupts at commit point (override others)
Topics

• Interrupts

• Speeding up the common case:
  – TLB & Cache organization

• Speeding up page table walks

• Modern Usage
Address Translation in CPU Pipeline

- 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
- 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
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

CPU \(\rightarrow\) Virtual Cache \(\rightarrow\) TLB \(\rightarrow\) Primary Memory

CPU \(\rightarrow\) Virtual Cache \(\rightarrow\) TLB \(\rightarrow\) Primary Memory

(StrongARM)
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: **Disallow aliases to coexist 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 SPARCs)
Concurrent Access to TLB & Cache

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:** \( L + b = k \) \quad L + b < k \quad L + b > k
Virtual-Index Physical-Tag Caches: Associative Organization

Is this scheme realistic?
Concurrent Access to TLB & Large L1
The problem with L1 > Page size

Can $VA_1$ and $VA_2$ both map to $PA$?
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.

L2 is *physically addressed*.
Anti-Aliasing Using L2: \textit{MIPS R10000}

- 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. Field a is different \(\Rightarrow\) Collision!
- VA1 will be purged from L1 and L2, and VA2 will be loaded \(\Rightarrow\) no aliasing!
Virtually-Addressed L1: Anti-Aliasing using L2

Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1.
Topics

- Interrupts

- Speeding up the common case:
  - TLB & Cache organization

- Speeding up page table walks

- Modern Usage
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*

- 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?

A program that traverses the page table needs a “no translation” addressing mode.
Hierarchical Page Table Walk: SPARC v8

Virtual Address

Context Table Register → Context Table

Index 1: 31
Index 2: 23
Index 3: 17
Offset: 11

Many page table levels ➔ Large TLB miss penalty

Can we improve this?

Add a layer of cache!
hashed Page Table: Approximating Associative Addressing

Virtual Address

Virtual Page Number (VPN)  
Physical Page Number (PPN)

Virtual Address

Virtual Page Number (VPN)  
Physical Page Number (PPN)

Virtual Page Number (VPN)  
Physical Directory Number (DPN)

Virtual Page Number (VPN)  
Physical Page Number (PPN)

- Caching layer *in memory* (not a hardware cache)
- Hashed Page Table is typically 2 to 3 times larger than the number of PPN’s to reduce collision probability
- Typically only contains in-memory PTEs
- If a translation cannot be resolved in this table then invoke *software*
- ...Or add a L2 LTB in hardware
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 its PTE’s point to pages in the primary memory

Why?

http://www.csg.csail.mit.edu/6.823
Topics

• Interrupts

• Speeding up the common case:
  – TLB & Cache organization

• Speeding up page table walks

• Modern Usage
Virtual Memory Use Today - 1

• 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

• Vector supercomputers have translation and protection but not demand-paging
  (Older Crays: base&bound, Japanese & Cray X1: 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
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

Given the software demands of modern embedded devices (e.g., cell phones, PDAs) all this may change in the near future!
Next lecture:
Complex pipelining
Atlas Revisited

• One PAR for each physical page

• PAR’s contain the VPN’s of the pages \textit{resident in primary memory}

• \textit{Advantage:} The size is proportional to the size of the primary memory

• \textit{What is the disadvantage?}
Global System Address Space

- Level A maps users’ address spaces into the global space providing privacy, protection, sharing etc.
- Level B provides demand-paging for the large global system address space
- Level A and Level B translations may be kept in separate TLB’s
Hashed Page Table Walk:
PowerPC Two-level, Segmented Addressing

Hashed Segment Table
64-bit user VA

Hashed Page Table

Global Seg ID          Page     Offset
0                             35              51              63

Hashed Page Table

PA of Seg Table
per process

PA of Page Table
system-wide

[ IBM numbers bits with MSB=0 ]

40-bit PA

PPN

Offset

offset
## Power PC: Hashed Page Table

### Diagram

- **VPN** → **d** → **80-bit VA**
  - **Offset** → **PA of Slot**
  
- **Base of Table**
  
- **Page Table**
  
- **Primary Memory**

### Points

- 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 a software data structure.