Virtual Memory:  
Part Deux
Need to restart instruction.
Address Translation in CPU Pipeline

- Need **restartable** exception on page fault/protection violation for software handler
- Either *hardware* or *software* TLB refill on TLB miss

- Coping 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

Instead of

+ 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
**Aliasing in Virtual-Address Caches**

Two virtual pages share one physical page

<table>
<thead>
<tr>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA₁</td>
<td>1st Copy of Data at PA</td>
</tr>
<tr>
<td>VA₂</td>
<td>2nd Copy of Data at PA</td>
</tr>
</tbody>
</table>

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)*

Two processes sharing the same file,
Map the same memory segment to different
Parts of their address space.
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: $L + b = k$, $L + b < k$, $L + b > k$
Virtual-Index Physical-Tag Caches:
Associative Organization

Consider 4-Kbyte pages and caches with 32-byte blocks

32-Kbyte cache \( \Rightarrow 2^a = 8 \)

4-Mbyte cache \( \Rightarrow 2^a = 1024 \) No!

Is this scheme realistic?
Concurrent Access to TLB & Large L1

Can VA₁ and VA₂ both map to PA?

If they differ in the lower ‘a’ bits alone, and share a physical page.
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
Anti-Aliasing Using L2: (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, a collision will be detected in L₂. VA₁ will be purged from L₁ and L₂, and VA₂ will be loaded ⇒ no aliasing!
Virtually-Addressed L1:
Anti-Aliasing using L2

Physically-addressed L2 can also be used to avoid aliases in virtually-addressed L1
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  
  (A different job may start running 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 (μsecs), page faults are handled completely in software by the operating system  
- Untranslated addressing mode is essential to allow kernel to access page tables

Deleted pseudo-LRU policy here. Pseudo-LRU means you can’t afford to do the real thing.
Essentially has 1 bit to indicate if a page has been recently used or not. A round-robin policy is used to find the first page with a value of 0.
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?__________________________

What is the worst thing you can do with respect to storing page tables?
Storing page table on disk for whose entries point to phys. Mem.
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?*

Good old Atlas which had all the ideas
(Was an improvement on most of its successors.)
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.
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

64-bit user VA

Hashed Segment Table

80-bit System VA

Hashed Page Table

40-bit PA

Page 17
**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 a software data structure.
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

An interrupt is 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
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*)

- disables interrupts and transfers control to a designated interrupt handler running in kernel mode
Interrupt Handler

To allow nested interrupts, EPC is saved before enabling interrupts ⇒

- need an instruction to move EPC into GPRs
- need a way to mask further interrupts at least until EPC can be saved

A status register indicates the cause of the interrupt - it must be visible to an interrupt handler

The return from an interrupt handler is a simple indirect jump but usually involves

- enabling interrupts
- restoring the processor to the user mode
- restoring hardware status and control state

⇒ a special return-from-exception instruction (RFE)
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
How to handle multiple simultaneous exceptions in different pipeline stages?

How and where to handle external asynchronous interrupts?
Exception Handling (5-Stage Pipeline)

1. **Inst. Mem**
   - PC Address Exceptions
   - Select Handler PC

2. **Decode**
   - Illegal Opcode
   - Kill D Stage

3. **E**
   - Overflow
   - Kill E Stage

4. **M**
   - Data Address Exceptions
   - Asynchronous Interrupts
   - Kill F Stage

5. **Data Mem**
   - Kill D Stage
   - Commit Point
   - Writeback

- **Select Handler PC**
- **Asynchronous Interrupts**
- **Illegal Opcode**
- **Overflow**
- **Data Address Exceptions**
- **Kill D Stage**
- **Kill E Stage**
- **Commit Point**
- **Writeback**

Page 25
**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*

- Inject external interrupts at commit point (override others)

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

• Vector supercomputers have translation and protection but not demand-paging
  (Crays: base&bound, Japanese: 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)
  – More difficult to implement restartable vector instructions

• 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