Virtual Memory: Systems

CSci 2021: Machine Architecture and Organization
Lecture #29-30, April 6-8th, 2015

Your instructor: Stephen McCamant

Based on slides originally by:
Randy Bryant, Dave O’Hallaron, Antonia Zhai

Outline
- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping

Review of Symbols
- Basic Parameters
  - \( N = 2^n \): Number of addresses in virtual address space
  - \( M = 2^m \): Number of addresses in physical address space
  - \( P = 2^p \): Page size (bytes)
- Components of the virtual address (VA)
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: Virtual page offset
  - VPN: Virtual page number
- Components of the physical address (PA)
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag

Simple Memory System Example
- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes

Simple Memory System Page Table
Only show first 16 entries (out of 256)

Simple Memory System TLB
- 16 entries
- 4-way associative
**Simple Memory System Cache**
- 16 lines, 4-byte block size
- Physically addressed
- Direct mapped

---

**Address Translation Example #1**
Virtual Address: 0x03D4

<table>
<thead>
<tr>
<th>TLBT</th>
<th>TLBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1111</td>
</tr>
</tbody>
</table>

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

---

**Address Translation Example #2**
Virtual Address: 0x0BBF

<table>
<thead>
<tr>
<th>TLBT</th>
<th>TLBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1111</td>
</tr>
</tbody>
</table>

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

---

**Address Translation Example #3**
Virtual Address: 0x0200

<table>
<thead>
<tr>
<th>TLBT</th>
<th>TLBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1111</td>
</tr>
</tbody>
</table>

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

---

**Today**
- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping

---

**Intel Core i7 Memory System**
Processor package
- Core 2x4
  - Instruction fetch
  - L1 i-cache
    - 32 KB, 8-way
  - L1 d-cache
    - 32 KB, 8-way
  - L2 unified TLB
    - 512 entries, 4-way
    - 32 GB/s total (shared by all cores)
  - L1 unified TLB
    - 128 entries, 4-way
    - 32 GB/s total (shared by all cores)
  - QPI interconnect
  - QuickPath interconnect
  - Physical memory controller
  - DDR3 Memory controller
  - 3 x 64 bit @ 10.66 GB/s
  - Main memory

---
Review of Symbols

- **Basic Parameters**
  - \( N = 2^M \): Number of addresses in virtual address space
  - \( M \): Number of addresses in physical address space
  - \( P = 2^L \): Page size (bytes)

- **Components of the virtual address (VA)**
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPN: Virtual page number
  - VPO: Virtual page offset
  - CI: Cache index
  - CT: Cache tag

- **Components of the physical address (PA)**
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CD: Cache disabled or enabled for the child page table.

- **End-to-end Core i7 Address Translation**

- **Core i7 Page Table Translation**

- **Core i7 Level 1-3 Page Table Entries**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>Unused</td>
</tr>
<tr>
<td>59</td>
<td>Page table physical base address</td>
</tr>
<tr>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>49</td>
<td>Unused</td>
</tr>
<tr>
<td>48</td>
<td>Page physical base address</td>
</tr>
<tr>
<td>47</td>
<td>46</td>
</tr>
</tbody>
</table>

Each entry references a 4K child page table

- R/W: Read-only or read/write access permission for all reachable pages
- U/S: User or supervisor (kernel) mode access
- WT: Write-through or write-back cache policy for the child page table
- CD: Caching disabled (1) or enabled (0)
- A: Reference bit (set by MMU on reads and writes, cleared by software)
- PS: “Page size”: if set, entry points to a large page (1GB or 2MB) not a page table

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

- **Core i7 Level 4 Page Table Entries**

Each entry references a 4K child page

- R/W: Read-only or read/write access permission for child page
- U/S: User or supervisor (kernel) mode access
- WT: Write-through or write-back cache policy for this page
- CD: Caching disabled (1) or enabled (0)
- A: Reference bit (set by MMU on reads and writes, cleared by software)
- D: Dirty bit (set by MMU on writes, cleared by software)

Page physical base address: 40 most significant bits of physical/page address

- **Cute Trick for Speeding Up L1 Access**

- **Observation**
  - Bits that determine CI identical in virtual and physical address
  - Can index into cache while address translation taking place
  - Generally we hit in TLB, so PPN bits (CT bits) available next
  - “Virtually indexed, physically tagged”
  - Cache carefully sized to make this possible
Discussion Point: TLBs and Processes

- Our system will have multiple processes running at once, each with their own address space. How does this affect the way the TLB has to work? (There are several choices)
  1. Clear the TLB when you switch processes: simple, but hurts performance
  2. Include a process/address space identifier as part of TLB entry tags ("tagged TLB"): needs larger TLB
  3. Global page flag: non-global entries are cleared on process switch
     - Compromise makes kernel faster
     - Used in current x86 processors

Virtual Memory of a Linux Process

Linux Organizes VM as Collection of "Areas"

- pgd: Page global directory address
  - Points to L1 page table
- vm_prot: Read/write permissions for this area
- vm_flags: Pages shared with other processes or private to this process

Linux Page Fault Handling

- Segmentation fault: accessing a non-existing page
- Protection exception: e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)

Memory Mapping

- VM areas initialized by associating them with disk objects.
  - Process is known as memory mapping.

- Area can be backed by (i.e., get its initial values from):
  - Regular file on disk (e.g., an executable object file)
    - Initial page bytes come from a section of a file
  - Anonymous file (e.g., nothing)
    - First fault will allocate a physical page full of 0's (demand-zero page)
    - Once the page is written to (dirtied), it is like any other page

- Dirty pages are copied back and forth between memory and a special swap file.
**Demand paging**

- **Key point:** no virtual pages are copied into physical memory until they are referenced!
  - Known as **demand paging**
- Improves time and space efficiency

**Sharing Revisited: Shared Objects**

- Process 1 maps the shared object.
- Notice how the virtual addresses can be different.

**Sharing Revisited: Private Copy-on-write (COW) Objects**

- Instruction writing to private page triggers protection fault.
- Handler creates new R/W page.
- Instruction restarts upon handler return.
- Copying deferred as long as possible!