Lecture 17: Virtual Memory (2)
TLBs and Multi-level page tables

Some slides modified from originals by Dave O’hallaron
Speeding up Translation with a TLB

- Page table entries (PTEs) are cached in L1 like any other memory word
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay

- Solution: *Translation Lookaside Buffer* (TLB)
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages
A TLB hit eliminates a memory access
A TLB miss incurs an additional memory access (the PTE)
Fortunately, TLB misses are rare. Why?
Reloading the TLB

- If the TLB does not have mapping, two possibilities:
  1. MMU loads PTE from page table in memory
     » Hardware managed TLB, OS not involved in this step
     » OS has already set up the page tables so that the hardware can access it directly
  2. Trap to the OS
     » Software managed TLB, OS intervenes at this point
     » OS does lookup in page table, loads PTE into TLB
     » OS returns from exception, TLB continues

- A machine will only support one method or the other
- At this point, there is a PTE for the address in the TLB
Page Faults

- PTE can indicate a protection fault
  - Read/write/execute – operation not permitted on page
  - Invalid – virtual page not allocated, or page not in physical memory

- TLB traps to the OS (software takes over)
  - R/W/E – OS usually will send fault back up to process, or might be playing games (e.g., copy on write, mapped files)
  - Invalid
    - Virtual page not allocated in address space
      - OS sends fault to process (e.g., segmentation fault)
    - Page not in physical memory
      - OS allocates frame, reads from disk, maps PTE to physical frame
Multi-Level Page Tables

- Suppose:
  - 4KB \( (2^{12}) \) page size, 48-bit address space, 8-byte PTE

- Problem:
  - Would need a 512 GB page table!
    - \( 2^{48} \times 2^{-12} \times 2^3 = 2^{39} \) bytes

- Common solution:
  - Multi-level page tables
  - Example: 2-level page table
    - Level 1 table: each PTE points to a page table (always memory resident)
    - Level 2 table: each PTE points to a page (paged in and out like any other data)
A Two-Level Page Table Hierarchy

**Level 1** page table
- PTE 0
- PTE 1
- PTE 2 (null)
- PTE 3 (null)
- PTE 4 (null)
- PTE 5 (null)
- PTE 6 (null)
- PTE 7 (null)
- PTE 8
- (1K - 9) null PTEs

**Level 2** page tables
- PTE 0
- ... PTE 1023

**Virtual memory**
- VP 0
- VP 1023
- VP 1024
- VP 2047
- Gap
- 1023 unallocated pages
- VP 9215

32 bit addresses, 4KB pages, 4-byte PTEs

- 2K allocated VM pages for code and data
- 6K unallocated VM pages
- 1023 unallocated pages
- 1 allocated VM page for the stack
TLB Misses (2)

Note that:

- Page table lookup (by HW or OS) can cause a recursive fault if page table is paged out
  - Assuming page tables are in OS virtual address space
  - Not a problem if tables are in physical memory
  - Yes, this is a complicated situation!

- When TLB has PTE, it restarts translation
  - Common case is that the PTE refers to a valid page in memory
    » These faults are handled quickly, just read PTE from the page table in memory and load into TLB
  - Uncommon case is that TLB faults again on PTE because of PTE protection bits (e.g., page is invalid)
    » Becomes a page fault…
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)

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>0A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>0B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0C</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>0D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>0E</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>0F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
Simple Memory System TLB

- 16 entries
- 4-way associative

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>–</td>
<td>0</td>
<td>09</td>
<td>0D</td>
<td>1</td>
<td>00</td>
<td>–</td>
<td>0</td>
<td>07</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>2D</td>
<td>1</td>
<td>02</td>
<td>–</td>
<td>0</td>
<td>04</td>
<td>–</td>
<td>0</td>
<td>0A</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>–</td>
<td>0</td>
<td>08</td>
<td>–</td>
<td>0</td>
<td>06</td>
<td>–</td>
<td>0</td>
<td>03</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>–</td>
<td>0</td>
<td>03</td>
<td>0D</td>
<td>1</td>
<td>0A</td>
<td>34</td>
<td>1</td>
<td>02</td>
<td>–</td>
<td>0</td>
</tr>
</tbody>
</table>
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>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

VPN: \(0x0F\)  TLBI: \(0x3\)  TLBT: \(0x03\)  TLB Hit? \(Y\)  Page Fault? \(N\)  PPN: \(0xD\)

Physical Address

CT

<table>
<thead>
<tr>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

PPN

<table>
<thead>
<tr>
<th>CO</th>
<th>CI</th>
<th>CT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

CO: \(0\)  CI: \(0x5\)  CT: \(0xD\)  Hit? \(Y\)  Byte: \(0x36\)
### Address Translation Example #2

**Virtual Address:** 0x0B8F

<table>
<thead>
<tr>
<th>TLBI</th>
<th>TLBT</th>
<th>VPN</th>
<th>VPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1 0 1 1</td>
<td>0 0 0 1 1 1</td>
</tr>
</tbody>
</table>

- **VPN:** 0x2E
- **TLBI:** 2
- **TLBT:** 0x0B
- **TLB Hit?** N
- **Page Fault?** Y
- **PPN:** TBD

**Physical Address**

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **CO:** ___
- **CI:** ___
- **CT:** ___
- **Hit?** ___
- **Byte:** ___
Address Translation
Example #3

Virtual Address: 0x0020

Physical Address

CO 0 CI 0x8 CT 0x28 Hit? N Byte: Mem
Intel Core i7 Memory System

Processor package

Core x4

Registers

L1 d-cache 32 KB, 8-way

L1 i-cache 32 KB, 8-way

L2 unified cache 256 KB, 8-way

Instruction fetch

L1 d-TLB 64 entries, 4-way

L1 i-TLB 128 entries, 4-way

L2 unified TLB 512 entries, 4-way

MMU (addr translation)

QuickPath interconnect 4 links @ 25.6 GB/s each

DDR3 Memory controller 3 x 64 bit @ 10.66 GB/s 32 GB/s total (shared by all cores)

Main memory

To other cores

To I/O bridge
## End-to-end Core i7 Address Translation

![Diagram of address translation process]

**CPU**
- 36 virtual address (VA)
- 12
- VPN
- VPO
- TLBI
- TLBT
- VPN1
- VPN2
- VPN3
- VPN4
- CR3
- PTE
- PTE
- PTE
- PTE
- PPN
- PPO
- L1 TLB (16 sets, 4 entries/set)
- L1 TLB (64 sets, 8 lines/set)
- CR3
- CT
- CI
- CO
- Physical address (PA)

**Result**
- 32/64
- L1 hit
- L1 miss
- L2, L3, and main memory

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

Each entry references a 4K child page table

- **P**: Child page table present in physical memory (1) or not (0).
- **R/W**: Read-only or read-write access access permission for all reachable pages.
- **U/S**: user or supervisor (kernel) mode access permission for all reachable pages.
- **WT**: Write-through or write-back cache policy for the child page table.
- **CD**: Caching disabled or enabled for the child page table.
- **A**: Reference bit (set by MMU on reads and writes, cleared by software).
- **PS**: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).
- **G**: Global page (don’t evict from TLB on task switch)

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

- Programmer’s view of virtual memory
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes

- System view of virtual memory
  - Uses memory efficiently by caching virtual memory pages
    » Efficient only because of locality
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient interpositioning point to check permissions
Summary (2)

Paging mechanisms:

- Optimizations
  - Managing page tables (space)
  - Efficient translations (TLBs) (time)
  - Demand paged virtual memory (space)

- Recap address translation

- Advanced Functionality
  - Sharing memory
  - Copy on Write
  - Mapped files

Next time: Paging policies
MAPPED FILES

- Mapped files enable processes to do file I/O using loads and stores
  - Instead of “open, read into buffer, operate on buffer, …”
- Bind a file to a virtual memory region (mmap() in Unix)
  - PTEs map virtual addresses to physical frames holding file data
  - Virtual address base + N refers to offset N in file
- Initially, all pages mapped to file are invalid
  - OS reads a page from file when invalid page is accessed
  - OS writes a page to file when evicted, or region unmapped
  - If page is not dirty (has not been written to), no write needed
    » Another use of the dirty bit in PTE