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

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

32 bit addresses, 4KB pages, 4-byte PTEs
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></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<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**

### TLB Diagram

![TLB Diagram](image)

### TLB Table

<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

![Diagram of a cache with addresses and tags]

```
Idx | Tag | Valid | B0 | B1 | B2 | B3
--- |-----|-------|----|----|----|----
 0  |   19|      1| 99 | 11 | 23 | 11
 1  |   15|      0|    |    |    |    
 2  |   1B|      1| 00 | 02 | 04 | 08
 3  |   36|      0|    |    |    |    
 4  |   32|      1| 43 | 6D | 8F | 09
 5  |   0D|      1| 36 | 72 | F0 | 1D
 6  |   31|      0|    |    |    |    
 7  |   16|      1| 11 | C2 | DF | 03

Idx | Tag | Valid | B0 | B1 | B2 | B3
--- |-----|-------|----|----|----|----
 8  |   24|      1| 3A | 00 | 51 | 89
 9  |   2D|      0|    |    |    |    
 A  |   2D|      1| 93 | 15 | DA | 3B
 B  |   0B|      0|    |    |    |    
 C  |   12|      0|    |    |    |    
 D  |   16|      1| 04 | 96 | 34 | 15
 E  |   13|      1| 83 | 77 | 1B | D3
 F  |   14|      0|    |    |    |    
```
Address Translation
Example #1

Virtual Address: 0x03D4

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

VPN: 0x0F | TLB Hit? Y | Page Fault? N | PPN: 0x0D

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

PPN: 0x0D | Hit? Y | Byte: 0x36

CO: 0 | CI: 0x5 | CT: 0x0D
Address Translation
Example #2

Virtual Address: \texttt{0x0B8F}

<table>
<thead>
<tr>
<th>13</th>
<th>12</th>
<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>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

VPN \texttt{0xE}  TLBI \texttt{2}  TLBT \texttt{0x0B}  TLB Hit? \texttt{N}  Page Fault? \texttt{Y}  PPN: TBD

Physical Address

<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>
</table>

CO ____  CI ____  CT ____  Hit? ____  Byte: ____
Address Translation
Example #3

Virtual Address: \(0x0020\)

Physical Address

Byte: Mem
Intel Core i7 Memory System

Process package

Core x4

- Registers
- Instruction fetch
- L1 d-cache 32 KB, 8-way
- L1 i-cache 32 KB, 8-way
- L2 unified cache 256 KB, 8-way
- MMU (addr translation)
  - L1 d-TLB 64 entries, 4-way
  - L1 i-TLB 128 entries, 4-way
- L2 unified TLB 512 entries, 4-way
- 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

CPU

Virtual address (VA)

36

12

VPN VPO

32 4

TLBT TLBI

TLB miss

L1 TLB (16 sets, 4 entries/set)

VPN1 VPN2 VPN3 VPN4

9 9 9 9

CR3

PTE Page tables

32/64

32/64

Result

L1 hit

L1 d-cache (64 sets, 8 lines/set)

L1 miss

L2, L3, and main memory

40 12

PPN PPO

Physical address (PA)

40 6 6

CT CI CO
## Core i7 Level 1-3 Page Table Entries

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>52</th>
<th>51</th>
<th>12</th>
<th>11</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>XD</td>
<td>Unused</td>
<td>Page table physical base address</td>
<td>Unused</td>
<td>G</td>
<td>PS</td>
<td>A</td>
<td>CD</td>
<td>WT</td>
<td>U/S</td>
<td>R/W</td>
<td>P=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Available for OS (page table location on disk)

P=0

---

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