Virtual Memory III
CSE 351 Spring 2017

Instructor:
Ruth Anderson

Teaching Assistants:
Dylan Johnson
Kevin Bi
Linxing Preston Jiang
Cody Ohlsen
Yufang Sun
Joshua Curtis
Administrivia

- Homework 4 – Due TONIGHT, this Friday 5/19
  - Cache questions
- Lab 4 – Due Tuesday 5/23
  - Cache runtimes and parameter puzzles
Quick Review

- What do Page Tables map?
- Where are Page Tables located?
- How many Page Tables are there?
- Can your program tell if a page fault has occurred?
- What is thrashing?
- True / False: Virtual Addresses that are contiguous will always be contiguous in physical memory
- TLB stands for _______________________ and stores ____________________
Quick Review Answers

- What do Page Tables map?
  - VPN → PPN or disk address

- Where are Page Tables located?
  - In physical memory

- How many Page Tables are there?
  - One per process

- Can your program tell if a page fault has occurred?
  - Nope, but it has to wait a long time

- What is thrashing?
  - Constantly paging out and paging in

- True / False: Virtual Addresses that are contiguous will always be contiguous in physical memory
  - Could fall across a page boundary

- TLB stands for Translation Lookaside Buffer and stores page table entries
Example from Section
Simple Memory System Example (small)

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

- Only showing first 16 entries (out of _____)
  - **Note**: showing 2 hex digits for PPN even though only 6 bits
  - **Note**: management bits not shown, but part of each PTE

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>
Simple Memory System: TLB

- 16 entries total
- 4-way set associative

Why does the TLB ignore the page offset?
Simple Memory System: Cache

- Direct-mapped with $K = 4$ B, $C/K = 16$
- Physically addressed

Note: It is just coincidence that the PPN is the same width as the cache Tag
# Current State of Memory System

## TLB:

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>V</th>
<th>Tag</th>
<th>PPN</th>
<th>V</th>
<th>Tag</th>
<th>PPN</th>
<th>V</th>
<th>Tag</th>
<th>PPN</th>
<th>V</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>

## Page Table (partial):

<table>
<thead>
<tr>
<th>VPN</th>
<th>PPN</th>
<th>V</th>
<th>VPN</th>
<th>PPN</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>28</td>
<td>1</td>
<td>8</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>–</td>
<td>0</td>
<td>9</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>1</td>
<td>A</td>
<td>09</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>02</td>
<td>1</td>
<td>B</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>–</td>
<td>0</td>
<td>C</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
<td>D</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>0</td>
<td>E</td>
<td>–</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>0</td>
<td>F</td>
<td>0D</td>
<td>1</td>
</tr>
</tbody>
</table>

## Cache:

### Index 0

<table>
<thead>
<tr>
<th>Tag</th>
<th>V</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>1</td>
<td>99</td>
<td>11</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>1B</td>
<td>1</td>
<td>00</td>
<td>02</td>
<td>04</td>
<td>08</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>43</td>
<td>6D</td>
<td>8F</td>
<td>09</td>
</tr>
<tr>
<td>0D</td>
<td>1</td>
<td>36</td>
<td>72</td>
<td>F0</td>
<td>1D</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>11</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
</tr>
</tbody>
</table>

### Index 8

<table>
<thead>
<tr>
<th>Tag</th>
<th>V</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>1</td>
<td>3A</td>
<td>00</td>
<td>51</td>
<td>89</td>
</tr>
<tr>
<td>2D</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>93</td>
<td>1</td>
<td>DA</td>
<td>3B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0B</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>04</td>
<td>96</td>
<td>34</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>83</td>
<td>77</td>
<td>1B</td>
<td>D3</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Memory Request Example #1

- **Virtual Address:** 0x03D4

  - TLBT: 13-12-11-10-9-8-7-6-5-4-3-2-1-0
  - TLBI: 0-0-0-0-1-1-1-1-0-1-0-1-0-0

- **Physical Address:**

  - CT: 11-10-9-8-7-6-5-4-3-2-1-0
  - CI
  - CO
  - PPN
  - PPO

**Note:** It is just coincidence that the PPN is the same width as the cache Tag
Memory Request Example #2

- **Virtual Address:** 0x038F

```plaintext
0 0 0 0 1 1 1 0 0 0 1 1 1 1
```

- **Physical Address:**

```plaintext
CT  Cl  CO
11 10 9 8 7 6 5 4 3 2 1 0
```

Note: It is just coincidence that the PPN is the same width as the cache Tag
Memory Request Example #3

- Virtual Address: \(0x0020\)

- Physical Address:

Note: It is just coincidence that the PPN is the same width as the cache Tag
Memory Request Example #4

- **Virtual Address**: 0x036B

- **Physical Address**:

  - **VPN**
  - **VPO**
  - **TLBT**
  - **TLBI**
  - **VPN _____ TLBT _____ TLBI _____ TLB Hit? ____ Page Fault? ____ PPN _____**

  - **CT**
  - **Cl**
  - **CO**
  - **PPN**
  - **PPO**

  - **CT _____ CI _____ CO _____ Cache Hit? ____ Data (byte) ________**

**Note:** It is just coincidence that the PPN is the same width as the cache Tag
Virtual Memory 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 sharing
  - Simplifies protection by providing permissions checking
Address Translation

- VM is complicated, but also elegant and effective
  - Level of indirection to provide isolated memory & caching
  - TLB as a cache of page tables avoids two trips to memory for every memory access
Memory System Summary

- Memory Caches (L1/L2/L3)
  - Purely a speed-up technique
  - Behavior invisible to application programmer and (mostly) OS
  - Implemented totally in hardware

- Virtual Memory
  - Supports many OS-related functions
    - Process creation, task switching, protection
  - Operating System (software)
    - Allocates/shares physical memory among processes
    - Maintains high-level tables tracking memory type, source, sharing
    - Handles exceptions, fills in hardware-defined mapping tables
  - Hardware
    - Translates virtual addresses via mapping tables, enforcing permissions
    - Accelerates mapping via translation cache (TLB)
Memory Overview

- `movl 0x8043ab, %rdi`
Memory System – Who controls what?

- Memory Caches (L1/L2/L3)
  - Controlled by hardware
  - Programmer cannot control it
  - Programmer can write code to take advantage of it

- Virtual Memory
  - Controlled by OS and hardware
  - Programmer cannot control mapping to physical memory
  - Programmer can control sharing and some protection
    - via OS functions (not in CSE 351)
Page Table Reality

- Just one issue... the numbers don’t work out for the story so far!

- The problem is the page table for each process:
  - Suppose 64-bit VAs, 8 KiB pages, 8 GiB physical memory
  - How many page table entries is that?
  - About how long is each PTE?

- **Moral:** Cannot use this naïve implementation of the virtual→physical-page mapping – it’s way too big
A Solution: Multi-level Page Tables

This is called a *page walk*

This is extra (non-testable) material
Multi-level Page Tables

- A tree of depth $k$ where each node at depth $i$ has up to $2^j$ children if part $i$ of the VPN has $j$ bits
- Hardware for multi-level page tables inherently more complicated
  - But it’s a necessary complexity – 1-level does not fit
- Why it works: Most subtrees are not used at all, so they are never created and definitely aren’t in physical memory
  - Parts created can be evicted from cache/memory when not being used
  - Each node can have a size of ~1-100KB
- But now for a $k$-level page table, a TLB miss requires $k + 1$ cache/memory accesses
  - Fine so long as TLB misses are rare – motivates larger TLBs

This is extra (non-testable) material
Practice VM Question

- Our system has the following properties:
  - 1 MiB of physical address space
  - 4 GiB of virtual address space
  - 32 KiB page size
  - 4-entry fully associative TLB with LRU replacement

a) Fill in the following blanks:

- ________ Entries in page table
- ________ Minimum bit-width of PTBR
- ________ TLBT bits
- ________ Max # of valid entries in a page table
Practice VM Question

- One process uses a page-aligned *square* matrix `mat[]` of 32-bit integers in the code shown below:

  ```
  #define MAT_SIZE = 2048
  for(int i = 0; i < MAT_SIZE; i++)
      mat[i * (MAT_SIZE + 1)] = i;
  ```

b) What is the largest stride (in bytes) between successive memory accesses (in the VA space)?
Practice VM Question

- One process uses a page-aligned square matrix `mat[]` of 32-bit integers in the code shown below:
  ```c
  #define MAT_SIZE = 2048
  for(int i = 0; i < MAT_SIZE; i++)
      mat[i * (MAT_SIZE + 1)] = i;
  ```

  c) What are the following hit rates for the first execution of the for loop?

  ________  TLB Hit Rate  ________  Page Table Hit Rate