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
- Homework 5 – coming soon!
  - Processes and VM
- Lab 5 – coming soon!
  - Memory Allocator (malloc)
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: \( n = 14 \) bits \( \iff \) \( N = 16 \text{ KiB} \) VA space
  - 12-bit physical address: \( m = 12 \) bits \( \iff \) \( M = 4 \text{ KiB} \) PA space
  - Page size = 64 bytes: \( P = 64 \text{ B} \) \( \iff \) \( p = 6 \) bits

![Diagram of memory addressing](image-url)
Simple Memory System: Page Table

- Only showing first 16 entries (out of $2^8 = 256$)
  - **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

\[
\text{16/4} = 4 \text{ sets}
\]

Why does the TLB ignore the page offset?

not part of its job!
(address translation)

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

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

```
<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>1</td>
<td>99</td>
<td>11</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1B</td>
<td>1</td>
<td>00</td>
<td>02</td>
<td>04</td>
<td>08</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1</td>
<td>43</td>
<td>6D</td>
<td>8F</td>
<td>09</td>
</tr>
<tr>
<td>5</td>
<td>0D</td>
<td>1</td>
<td>36</td>
<td>72</td>
<td>F0</td>
<td>1D</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>11</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Valid</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24</td>
<td>1</td>
<td>3A</td>
<td>00</td>
<td>51</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>2D</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>2D</td>
<td>1</td>
<td>93</td>
<td>15</td>
<td>DA</td>
<td>3B</td>
</tr>
<tr>
<td>B</td>
<td>0B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>16</td>
<td>1</td>
<td>04</td>
<td>96</td>
<td>34</td>
<td>15</td>
</tr>
<tr>
<td>E</td>
<td>13</td>
<td>1</td>
<td>83</td>
<td>77</td>
<td>1B</td>
<td>D3</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

Circled #s refer to Memory Request Example #

**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>0X</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>0X</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>
</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>

**Cache:**

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>V</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
<td>1</td>
<td>99</td>
<td>11</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>15</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1B</td>
<td>1</td>
<td>00</td>
<td>02</td>
<td>04</td>
<td>08</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td>1</td>
<td>43</td>
<td>6D</td>
<td>8F</td>
<td>09</td>
</tr>
<tr>
<td>5</td>
<td>0D</td>
<td>1</td>
<td>36</td>
<td>72</td>
<td>F0</td>
<td>1D</td>
</tr>
<tr>
<td>6</td>
<td>31</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>11</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>V</th>
<th>B0</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>24</td>
<td>1</td>
<td>3A</td>
<td>00</td>
<td>51</td>
<td>89</td>
</tr>
<tr>
<td>9</td>
<td>2D</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>A</td>
<td>2D</td>
<td>1</td>
<td>93</td>
<td>15</td>
<td>DA</td>
<td>3B</td>
</tr>
<tr>
<td>B</td>
<td>0B</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>D</td>
<td>16</td>
<td>1</td>
<td>04</td>
<td>96</td>
<td>34</td>
<td>15</td>
</tr>
<tr>
<td>E</td>
<td>13</td>
<td>1</td>
<td>83</td>
<td>77</td>
<td>1B</td>
<td>D3</td>
</tr>
<tr>
<td>F</td>
<td>14</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Memory Request Example #1

- **Virtual Address:** 0x03D4

  - **VPN:** 0xF
  - **TLBT:** 0x03 (check this entry of the page table)
  - **TLBI:** 3 (look for this tag within TLB set)
  - **TLB Hit?** Y
  - **Page Fault?** N
  - **PPN:** 0x0D

- **Physical Address:**

  - **CT:** 0x0D (look for this tag within cache set)
  - **CI:** 5 (check this set of the cache)
  - **CO:** 0 (which byte of cache block)
  - **Cache Hit?** Y
  - **Data (byte):** 0x36
Memory Request Example #2

- **Virtual Address:** $0x038F$

  ![TLB Diagram]

  - VPN: $0x0E$
  - TLBT: $0x03$
  - TLBI: $2$
  - TLB Hit?: $N$
  - Page Fault?: $Y$
  - PPN: n/a

- **Physical Address:**

  ![Physical Address Diagram]

  - CT: $____$
  - CI: $____$
  - CO: $____$
  - Cache Hit?: $____$
  - Data (byte): $____$

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

- **Virtual Address:** 0x0020

  ![TLB Diagram]

  - VPN
  - TLB:
  - TLBI
  - TLBT

  - VPO
  - VPO

  - PPO
  - PPO

- **Physical Address:**

  ![Cache Diagram]

  - CT
  - CI
  - CO

  - PPN
  - PPO

  - Data (byte): Unknown

  ![Page Fault]

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

- **Virtual Address:** 0x036B
  - TLBT: 00000110110101010111
  - VPN: 000111011010101011
  - TLBT: 0x03
  - TLBI: 1
  - TLB Hit? Y
  - Page Fault? N
  - PPN: 0x2D

- **Physical Address:**
  - CT: 0x2D
  - CI: A
  - CO: 3
  - Cache Hit? Y
  - Data (byte): 0x3B
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?
    - 1 PTE for every virtual page
      \[ 2^{n-p} = 2^{51} PTE \]
    - How many bits for PTE?
      \[ m-p=20 \text{ bits} \rightarrow 24 \text{ bits} = 3 \text{ B} \]
  - 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
Practice VM Question

- Our system has the following properties:
  - 1 MiB of physical address space \( m = 20 \)
  - 4 GiB of virtual address space \( n = 32 \)
  - 32 KiB page size \( p = 15 \)
  - 4-entry fully associative TLB with LRU replacement

\[ \text{1 set} \]

a) Fill in the following blanks:

- \( \frac{2^{17}}{2^{n-p}} \) Entries in page table \( \leq \) # of virtual pages
- \( 2^5 \) TLBT bits

\[ \text{VPN} \rightarrow \text{TLBT/TLBI} \]
\[ \text{here TLBI} = 0 \]

- \( 2^{20} \) Minimum bit-width of PTBR \( \leftarrow \) physical address of PT

\[ m \]

- \( \frac{2^{5}}{2^{n-p}} \) Max # of valid entries in a page table \( \leq \) # of pages in physical memory
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 = 2^11
  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)?

2049 ints = \[
\begin{array}{c}
  \text{0} \\
  \text{2049} \\
  \text{2*2049} \\
\end{array}
\]  

2049 * 4 B = \text{stride}
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;
```

c) What are the following hit rates for the first execution of the for loop? (assume all of mat[] starts on disk)

- **TLB Hit Rate:** \(\frac{3}{4} = 75\%\)
- **Page Table Hit Rate:** 0%

**access pattern:**
- Single write to index
- Never revisit indices (always increasing)
- We access every row of matrix exactly once
- Each page holds \(\frac{2^{15}}{2^{13}} = 4\) rows of matrix
- Within each page: MTHHL

**Note:**
- Only access PT on TLB Miss
- Because mat[] on disk, each first access to page causes page fault.