Virtual Memory II
CSE 351 Autumn 2016

Instructor:
Justin Hsia

Teaching Assistants:
Chris Ma
Hunter Zahn
John Kaltenbach
Kevin Bi
Sachin Mehta
Suraj Bhat
Thomas Neuman
Waylon Huang
Xi Liu
Yufang Sun

https://xkcd.com/1495/
Administrivia

- Lab 4 due Monday, Nov. 28

- Wednesday’s Lecture A will be recorded via Panopto
  - Supposedly live-streamed, too (on Canvas?)

- “Virtual section” on virtual memory over the break
  - Worksheet (and solutions) will be posted for extra practice
Virtual Memory (VM)

- Overview and motivation
- VM as a tool for caching
- Address translation
- VM as a tool for memory management
- VM as a tool for memory protection
Review: Terminology

- Context switch
  - Switch between processes on the same CPU
- Page in
  - Move pages of virtual memory from disk to physical memory
- Page out
  - Move pages of virtual memory from physical memory to disk
- Thrashing
  - Total working set size of processes is larger than physical memory and causes excessive paging in and out instead of doing useful computation
VM for Managing Multiple Processes

- Key abstraction: each process has its own virtual address space
  - It can view memory as a simple linear array
- With virtual memory, this simple linear virtual address space need not be contiguous in physical memory
  - Process needs to store data in another VP? Just map it to any PP!

![Address translation diagram]

Virtual Address Space for Process 1:
- 0 to N-1
- VP 1, VP 2, ...

Virtual Address Space for Process 2:
- 0 to N-1
- VP 1, VP 2, ...

Physical Address Space (DRAM):
- 0 to M-1
- PP 2, PP 6, PP 8, ...
- (e.g., read-only library code)
Simplifying Linking and Loading

- **Linking**
  - Each program has similar virtual address space
  - Code, Data, and Heap always start at the same addresses

- **Loading**
  - `execve` allocates virtual pages for `.text` and `.data` sections & creates PTEs marked as invalid
  - The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system
VM for Protection and Sharing

- The mapping of VPs to PPs provides a simple mechanism to protect memory and to share memory between processes
  - **Sharing:** map virtual pages in separate address spaces to the same physical page (here: PP 6)
  - **Protection:** process can’t access physical pages to which none of its virtual pages are mapped (here: Process 2 can’t access PP 2)
Memory Protection Within Process

- VM implements read/write/execute permissions
  - Extend page table entries with permission bits
  - MMU checks these permission bits on every memory access
    - If violated, raises exception and OS sends SIGSEGV signal to process (segmentation fault)

<table>
<thead>
<tr>
<th>Process $i$:</th>
<th>Valid</th>
<th>READ</th>
<th>WRITE</th>
<th>EXEC</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP 0:</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>PP 6</td>
</tr>
<tr>
<td>VP 1:</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>PP 4</td>
</tr>
<tr>
<td>VP 2:</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>PP 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process $j$:</th>
<th>Valid</th>
<th>READ</th>
<th>WRITE</th>
<th>EXEC</th>
<th>PPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP 0:</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>PP 9</td>
</tr>
<tr>
<td>VP 1:</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>PP 6</td>
</tr>
<tr>
<td>VP 2:</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>PP 11</td>
</tr>
</tbody>
</table>

Physical Address Space
Address Translation: Page Hit

1) Processor sends *virtual* address to MMU (*memory management unit*)

2-3) MMU fetches PTE from page table in cache/memory  
    (Uses PTBR to find beginning of page table for current process)

4) MMU sends *physical* address to cache/memory requesting data

5) Cache/memory sends data to processor

VA = Virtual Address   PTEA = Page Table Entry Address   PTE = Page Table Entry
PA = Physical Address  Data = Contents of memory stored at VA originally requested by CPU
Address Translation: Page Fault

1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in cache/memory
4) Valid bit is zero, so MMU triggers page fault exception
5) Handler identifies victim (and, if dirty, pages it out to disk)
6) Handler pages in new page and updates PTE in memory
7) Handler returns to original process, restarting faulting instruction
Hmm... Translation Sounds Slow

- The MMU accesses memory twice: once to get the PTE for translation, and then again for the actual memory request
  - The PTEs may be cached in L1 like any other memory word
    - But they may be evicted by other data references
    - And a hit in the L1 cache still requires 1-3 cycles

- What can we do to make this faster?
  - Solution: add another cache! 🎉
Speeding up Translation with a TLB

- **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
    - Modern Intel processors have 128 or 256 entries in TLB
  - Much faster than a page table lookup in cache/memory

![TLB Diagram]
A TLB hit eliminates a memory access!
A TLB miss incurs an additional memory access (the PTE)
- Fortunately, TLB misses are rare
Fetching Data on a Memory Read

1) Check TLB
   - **Input**: VPN, **Output**: PPN
   - **TLB Hit**: Fetch translation, return PPN
   - **TLB Miss**: Check page table (in memory)
     - **Page Table Hit**: Load page table entry into TLB
     - **Page Fault**: Fetch page from disk to memory, update corresponding page table entry, then load entry into TLB

2) Check cache
   - **Input**: physical address, **Output**: data
   - **Cache Hit**: Return data value to processor
   - **Cache Miss**: Fetch data value from memory, store it in cache, return it to processor
Address Translation

Virtual Address

TLB Lookup

- TLB Miss
  - Page Table “Walk”
    - Page not in Mem
      - Page Fault (OS loads page)
        - Find in Disk
  - Page in Mem
    - Update TLB
      - Find in Mem

- TLB Hit
  - Protection Check
    - Access Denied
      - Protection Fault
        - SIGSEGV
        - Check cache
    - Access Permitted
      - Physical Address


Summary of Address Translation 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)**
  - VPO  Virtual page offset
  - VPN  Virtual page number
  - TLBI  TLB index
  - TLBT  TLB tag

- **Components of the physical address (PA)**
  - PPO  Physical page offset (same as VPO)
  - PPN  Physical page number
Simple Memory System Example (small)

- **Addressing**
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes

![Diagram of memory addressing](image)

- **Virtual Page Number (VPN)**
  - 14 bits
  - Positions: 13 to 0

- **Virtual Page Offset (VPO)**
  - 5 bits
  - Positions: 11 to 7

- **Physical Page Number (PPN)**
  - 12 bits
  - Positions: 11 to 0

- **Physical Page Offset (PPO)**
  - 4 bits
  - Positions: 11 to 8
Simple Memory System: Page Table

- Only showing first 16 entries (out of _____)
  - **Note:** showing 2 hex digits for PPN even though only 6 bits

<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 $K = 4B$, $C/K = 16$
- Physically addressed

Note: It is just coincidence that the PPN is the same width as the cache Tag.

<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>

Diagram:
- Cache tag
- Cache index
- Cache offset
- Physical page number
- Physical page offset
# 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:

<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

  ![Virtual Address Diagram]

  VPN _____ TLBT _____ TLBI _____ TLB Hit? ___ Page Fault? ___ PPN _____

- **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 #2

- **Virtual Address**: 0x038F

  - **TLBT**: 1011
  - **TLBI**: 1100
  - **VPN**: 1110

  - **VPO**: 1111

- **Physical Address**:
  - **CT**: 1000
  - **CI**: 0000
  - **CO**: 0000

  - **PPN**: 1111
  - **PPO**: 0000

  - **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

  - TLBT: 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
  - VPN: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

- **Physical Address:**

  - CT: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  - CO: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

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

- **Virtual Address:** \(0x036B\)

  - TLBT: \(0000\)
  - TLBI: \(1101\)
  - VPO: \(1110\)
  - VPN: \(1101\)

  ![Diagram of TLB and VPN](image1.png)

- **Physical Address:**

  - CT: \(1101\)
  - CI: \(0110\)
  - CO: \(1001\)
  - PPN: \(1101\)

  ![Diagram of Cache and PPN](image2.png)

**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
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 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)