Virtual Memory II
CSE 351 Winter 2024

Instructor:
Justin Hsia

Teaching Assistants:
Adithi Raghavan
Aman Mohammed
Connie Chen
Eyoel Gebre
Jiawei Huang
Malak Zaki
Naama Amiel
Nathan Khuat
Nikolas McNamee
Pedro Amarante
Will Robertson

https://xkcd.com/1495/
Relevant Course Information

- HW22 due tonight, HW23 due Wednesday, HW24 due Friday
- Today is the last day to submit Lab 4
- Lab 5 due Friday (3/8)
  - The most significant amount of C programming you will do in this class – combines lots of topics from this class: pointers, bit manipulation, structs, examining memory
  - Understanding the concepts first and efficient debugging will save you lots of time
  - Light style grading

- No lessons in Week 11 – “normal” lectures
- Final exam: 3/11-13
  - Final review session on 3/8 @ 4:30 pm in CSE G01 and on Zoom
Virtual Memory II
Lesson Summary (1/2)

- Address translation done via **page tables**
  - Lookup tables (one per process) that map VPN → PPN
  - Uses management bits: valid bit, access rights (read, write, execute)
  - Stored in memory – page table for currently-running process is pointed to by **page table base register** (PTBR)

```
Virtual address (VA)

Valid bit = 0: page not in memory (page fault)
```

<table>
<thead>
<tr>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page table base register (PTBR)</td>
</tr>
</tbody>
</table>

```
Physical address (PA)

Valid
```

```
Physical page number (PPN) Virtual page number (VPN) Virtual page offset (VPO)
```

```
Page table address for process
```

```
Physical page offset (PPO)
```

```
Virtual page offset (VPO)
```

```
Virtual page number (VPN)
```

```
Valid
```

```
PPN
```

```
Valid
```
Lesson Summary (2/2)

- Introduced the **translation lookaside buffer** (TLB) as a cache for page table entries (PTEs)
  - Try to avoid accessing the page table in memory
  - Split VPN into **TLB Tag** and **TLB Index** based on # of sets in TLB
  - Management bits include valid (like $, not PT) and TLB tag
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
Fetching Data on a Memory Read

1) Address Translation (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) Fetch Data (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 Manipulation

request from CPU: \( n \)-bit virtual address

split to access TLB: TLB Tag \( \rightarrow \) TLB Index \( \rightarrow \) Page Offset

(on TLB miss) access PT: \( m \)-bit physical address:

split to access cache: Cache Tag \( \rightarrow \) Cache Index \( \rightarrow \) Block offset

TRANSLATION
Lesson Q&A

❖ Learning Objectives:
  ▪ Determine virtual memory parameters related to addresses, page tables, and TLBs.
  ▪ Perform address translations (virtual address → physical address).
  ▪ Describe the relationships between virtual memory parameters and policies.

❖ What lingering questions do you have from the lesson?
  ▪ Chat with your neighbors about the lesson for a few minutes to come up with questions
Virtual Memory II — Practice
Virtual Memory Concept Questions

❖ Which terms from caching are most similar/analogous to the new virtual memory terms?

- block #, block size, cache line, cache set, index width, management bits, offset width, tag width
- page size
- page offset width
- virtual page number
- physical page number
- page table entry
- access rights
VM Parameters Questions

❖ 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 a page table  ________ Minimum bit-width of PTBR

________ TLBT bits  ________ Max # of valid entries in a page table
Memory Translation Examples
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
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: other management bits not shown, but part of 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>
<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**

![Diagram of TLB index and tag]

**Why does the TLB ignore the page offset?**

<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>
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

<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>
### Current State of Memory System

#### TLB:

<table>
<thead>
<tr>
<th>Set</th>
<th>Tag</th>
<th>PPN</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>03</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>09</td>
<td>0D</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>03</td>
<td>2D</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>07</td>
<td>0D</td>
<td>1</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>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</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>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>1</td>
<td>C2</td>
<td>DF</td>
<td>03</td>
<td>03</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`

  ![TLB Diagram](diagram)

  - **VPN** ______  **TLBT** _____  **TLBI** _____  **TLB Hit?** ___  **Page Fault?** ___  **PPN** _____

- **Physical Address**:

  ![Cache Diagram](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

![Diagram showing TLB, VPN, VPO, TLBT, TLBI, VPN, TLB Hit, Page Fault, PPN]

- **Physical Address:**

![Diagram showing CT, CI, CO, PPN, PPO, CT Hit, Data (byte)]

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

❖ Virtual Address: \(0x036B\)

Virtual Address Example:

\[
\begin{array}{ccccccccccccc}
13 & 12 & 11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\end{array}
\]

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

❖ Physical Address:

Physical Address Example:

\[
\begin{array}{ccccccccccccc}
11 & 10 & 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\
\end{array}
\]

CT ______  CI _____  CO _____  Cache Hit? ___  Data (byte) _______

Note: It is just coincidence that the PPN is the same width as the cache Tag
We made it! 😊 😎 😂

- **Topic Group 1: Data**
  - Memory, Data, Integers, Floating Point, Arrays, Structs

- **Topic Group 2: Programs**
  - x86-64 Assembly, Procedures, Stacks, Executables

- **Topic Group 3: Scale & Coherence**
  - Caches, Memory Allocation, Processes, Virtual Memory

We’ll explore the OUTSIDE of the House of Computing next lecture!
Group Work Time

❖ During this time, you are encouraged to work on the following:

1) If desired, continue your discussion
2) Work on the homework problems
3) Work on the current lab

❖ Resources:

- You can revisit the lesson material
- Work together in groups and help each other out
- Course staff will circle around to provide support