Virtual Memory: Systems

15-213: Introduction to Computer Systems
18th Lecture, March 22, 2016

Instructors:
Franz Franchetti & Seth Copen Goldstein, Ralf Brown, and Brian Raling
Cheating: Description

■ What is cheating?
  ▪ **Sharing code**: by copying, retyping, **looking at**, or supplying a file
  ▪ **Describing**: verbal description of code from one person to another.
  ▪ **Coaching**: helping your friend to write a lab, line by line
  ▪ **Searching the Web** for solutions
  ▪ **Copying code** from a previous course or online solution
    ▪ You are only allowed to use code we supply, or from the CS:APP website

■ What is NOT cheating?
  ▪ Explaining how to use systems or tools
  ▪ Helping others with high-level design issues

■ See the course syllabus for details.
  ▪ Ignorance is not an excuse
Cheating: Consequences

■ Penalty for cheating:
  ▪ If you do cheat – come clean asap!
  ▪ Removal from course with failing grade (no exceptions!)
  ▪ Permanent mark on your record
  ▪ Your instructors’ personal contempt

■ Detection of cheating:
  ▪ We have sophisticated tools for detecting code plagiarism
  ▪ Last Fall, 20 students were caught cheating and failed the course.
  ▪ Some were expelled from the University

■ Don’t do it!
  ▪ Start early
  ▪ Ask the staff for help when you get stuck

We found 10+ cheating cases in CacheLab
Regrading and Style Grading

- CacheLab Style Grades about to be released
- Regrade request deadline: 3/22, 11:59:00 pm EDT *tonight one minute before midnight*
- Need one hardcopy request and one email
  Gave hardcopy to TA after exam or to instructor, slide hardcopy under Seth’s or Franz’ door
- Result *may* show up in exam server
  unless you are a “special case”
- Final score will be uploaded to autolab later this week
MallocLab

- Out tonight
- Due April 8
- Checkpoint due March 31
Office Hours

- I am there every week...
- It is usually not very crowded
- Just come by to check in even if you do not have a question
Recap: Hmmm, How Does This Work?!
Transaction Lookaside Buffer (TLB)

A TLB hit eliminates a memory access
Virtual Memory and Physical Memory

- **Page hit:** reference to VM word that is in physical memory (DRAM cache hit)

![Diagram of virtual memory and physical memory relationship]

**Virtual address**

**Physical page number or disk address**

**Valid**

0: null
1: Memory resident page table (DRAM)

**Physical memory (DRAM)**

- VP 1
- VP 2
- VP 7
- VP 4

**Physical memory (disk)**

- VP 1
- VP 2
- VP 3
- VP 4
- VP 6
- VP 7
VM as a Tool for Memory Management

- Simplifying memory allocation
- Sharing code and data among processes

Virtual Address Space for Process 1:

0
VP 1
VP 2
...
N-1

0
Physical Address Space (DRAM)

0
PP 2
PP 6
PP 8
...
M-1

(e.g., read-only library code)

Virtual Address Space for Process 2:

0
VP 1
VP 2
...
N-1
Translating with a k-level Page Table

- **Page table base register (PTBR)**
- **VIRTUAL ADDRESS**
  - Level 1 page table
  - Level 2 page table
  - Level k page table
- **PHYSICAL ADDRESS**
  - PPN
  - PPO
Set Associative Cache: Read

E = \(2^e\) lines per set

S = \(2^s\) sets

Address of word:
- t bits
- s bits
- b bits

CT: tag
CI: index
CO: offset

data begins at this offset

- Locate set
- Check if any line in set has matching tag
- Yes + line valid: hit
- Locate data starting at offset

B = \(2^b\) bytes per cache block (the data)
Today

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping
Review of 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)**
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: Virtual page offset
  - VPN: Virtual page number

- **Components of the physical address (PA)**
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag
Simple Memory System Example

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

![Diagram showing virtual and physical page numbers and offsets with VPN, VPO, PPN, and PPO labels]
Simple Memory System TLB

- **16 entries**
- **4-way associative**

![Diagram of simple memory system TLB]

VPN = 0b1101 = 0x0D

Transaction Lookaside Buffer (TLB)

<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 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>
</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>OD</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>

0x0D → 0x2D
Simple Memory System Cache

- 16 lines, 4-byte block size
- Physically addressed
- Direct mapped

V[0b00001101101001] = V[0x369]
P[0b101101101001] = P[0xB69] = 0x15

<table>
<thead>
<tr>
<th>Idx</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>Idx</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>
### Address Translation Example

<table>
<thead>
<tr>
<th>Idx</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>Idx</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>

### Transaction Lookaside Buffer (TLB)

<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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tag</th>
<th>PPN</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>07</td>
<td>02</td>
<td>1</td>
</tr>
<tr>
<td>0A</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>03</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>
## Address Translation Example: TLB/Cache Miss

### TLB/Cache Miss Table

<table>
<thead>
<tr>
<th>Idx</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>

### Page Table

<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>
</tbody>
</table>

### Physical Address

- **CT**: 11 10 9 8 7 6 5 4 3 2 1 0
- **Cl**: 1 0 1 0 0 0 1 0 0 0 0 0
- **CO**: 1 0 1 0 0 0 1 0 0 0 0 0
- **PPN**: CO 0 Cl 0x8 CT 0x28
- **PPO**: Hit? N Byte: Mem
Virtual Memory Exam Question

Problem 5. (10 points):
Assume a System that has

1. A two way set associative TLB
2. A TLB with 8 total entries
3. $2^8$ byte page size
4. $2^{16}$ bytes of virtual memory
5. one (or more) boats

A. Use the TLB to fill in the table. Strike out anything that you don’t have enough information to fill in.

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Frame Number</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x13</td>
<td>0x30</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0x34</td>
<td>0x58</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0x1F</td>
<td>0x80</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0x2A</td>
<td>0x72</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0x1F</td>
<td>0x95</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0x20</td>
<td>0xAA</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0x3F</td>
<td>0x20</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0x3E</td>
<td>0xFF</td>
<td>0</td>
</tr>
</tbody>
</table>

B. TLBT

C. TLBI

D. CI = 0x2

E. CT = 0x1F

F. $0x7E85 = 0x0111111010000101$

G. $0x7E85 \rightarrow 0x9585$

Exam: [http://www.cs.cmu.edu/~213/oldexams/exam2b-s11.pdf (solution)]
Today

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping
Intel Core i7 Memory System

Processor package

Core x4

- Registers
- 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

Virtual address (VA) → CPU

VPN VPO

36 → 12

TLBT TLBI

L1 TLB (16 sets, 4 entries/set)

VPN1 VPN2 VPN3 VPN4

9 9 9 9

CR3

PTE PTE PTE PTE

Page tables

TLB

hit

TLB

miss

L1 TLB (16 sets, 4 entries/set)

VPN1 VPN2 VPN3 VPN4

9 9 9 9

CR3

PTE PTE PTE PTE

Page tables

32/64

Result

L2, L3, and main memory

L1 hit

L1 d-cache

(64 sets, 8 lines/set)

L1 miss

Physical address (PA)

CT CI CO

PPN PPO

40 12

L1 hit

L1 d-cache

(64 sets, 8 lines/set)

L1 miss

Physical address (PA)

CT CI CO

PPN PPO

40 12
### Core i7 Level 1-3 Page Table Entries

<table>
<thead>
<tr>
<th></th>
<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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page table physical base address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U/S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Each entry references a 4K child page table. Significant fields:

- **P**: Child page table present in physical memory (1) or not (0).
- **R/W**: Read-only or read-write 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.
- **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).
- **Page table physical base address**: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)
- **XD**: Disable or enable instruction fetches from all pages reachable from this PTE.
**Core i7 Level 4 Page Table Entries**

<table>
<thead>
<tr>
<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 physical base address</td>
<td>Unused</td>
<td>G</td>
<td>D</td>
<td>A</td>
<td>CD</td>
<td>WT</td>
<td>U/S</td>
<td>R/W</td>
<td>P=1</td>
</tr>
</tbody>
</table>

Available for OS (page location on disk)  

<table>
<thead>
<tr>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>XD</td>
<td>Unused</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Each entry references a 4K child page. Significant fields:**

**P:** Child page is present in memory (1) or not (0)

**R/W:** Read-only or read-write access permission for child page

**U/S:** User or supervisor mode access

**WT:** Write-through or write-back cache policy for this page

**A:** Reference bit (set by MMU on reads and writes, cleared by software)

**D:** Dirty bit (set by MMU on writes, cleared by software)

**Page physical base address:** 40 most significant bits of physical page address (forces pages to be 4KB aligned)

**XD:** Disable or enable instruction fetches from this page.
Core i7 Page Table Translation

CR3

Physical address of L1 PT

L1 PT

Page global directory

512 GB region per entry

L1 PTE

VPN 1

9

L2 PT

Page upper directory

1 GB region per entry

L2 PTE

VPN 2

9

L3 PT

Page middle directory

2 MB region per entry

L3 PTE

VPN 3

9

L4 PT

Page table

4 KB region per entry

L4 PTE

VPN 4

9

VPO

12

VPN

9

Offset into physical and virtual page

Physical address

40

40

40

40

40

40

40

Physical address of page

VPN 1

VPN 2

VPN 3

VPN 4

Virtual address

VPN

9

9

9

9

12

VPN

PPO

PPN

12

512 GB region per entry

1 GB region per entry

2 MB region per entry

4 KB region per entry
Cute Trick for Speeding Up L1 Access

Observation
- Bits that determine CI identical in virtual and physical address
- Can index into cache while address translation taking place
- Generally we hit in TLB, so PPN bits (CT bits) available next
- "Virtually indexed, physically tagged"
- Cache carefully sized to make this possible
Virtual Address Space of a Linux Process

- **Process-specific data structs** (ptables, task and mm structs, kernel stack)
- **Physical memory**
- **Kernel code and data**
- **Memory mapped region** for shared libraries
- **Runtime heap** (malloc)
- **Uninitialized data** (.bss)
- **Initialized data** (.data)
- **Program text** (.text)
- **User stack**

**Identical for each process**

**Different for each process**

**Kernel virtual memory**

**Process virtual memory**

- **brk**
- **%rsp**

- **Physical memory** is identical for each process.
- **Kernel virtual memory** is different for each process.
- **Process virtual memory** is different for each process.
Linux Organizes VM as Collection of “Areas”

- **pgd:**
  - Page global directory address
  - Points to L1 page table

- **vm_prot:**
  - Read/write permissions for this area

- **vm_flags**
  - Pages *shared* with other processes or *private* to this process
Linux Page Fault Handling

Segmentation fault: accessing a non-existing page

Normal page fault

Protection exception: e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)
Today

- Simple memory system example
- Case study: Core i7/Linux memory system
- Memory mapping
Memory Mapping

- VM areas initialized by associating them with disk objects.
  - Process is known as *memory mapping*.

- Area can be *backed by* (i.e., get its initial values from):
  - *Regular file* on disk (e.g., an executable object file)
    - Initial page bytes come from a section of a file
  - *Anonymous file* (e.g., nothing)
    - First fault will allocate a physical page full of 0's (*demand-zero page*)
    - Once the page is written to (*dirtied*), it is like any other page

- Dirty pages are copied back and forth between memory and a special *swap file*. 
Sharing Revisited: Shared Objects

- Process 1 maps the shared object.
Sharing Revisited: Shared Objects

- Process 2 maps the shared object.
- Notice how the virtual addresses can be different.
Sharing Revisited:
Private Copy-on-write (COW) Objects

- Two processes mapping a *private copy-on-write (COW)* object.
- Area flagged as private copy-on-write
- PTEs in private areas are flagged as read-only
Sharing Revisited: Private Copy-on-write (COW) Objects

- Instruction writing to private page triggers protection fault.
- Handler creates new R/W page.
- Instruction restarts upon handler return.
- Copying deferred as long as possible!

Diagram:

- Process 1 virtual memory
- Physical memory
- Process 2 virtual memory
- Private copy-on-write object
- Copy-on-write
- Write to private copy-on-write page
The `fork` Function Revisited

- VM and memory mapping explain how `fork` provides private address space for each process.

- To create virtual address for new new process
  - Create exact copies of current `mm_struct`, `vm_area_struct`, and page tables.
  - Flag each page in both processes as read-only
  - Flag each `vm_area_struct` in both processes as private COW

- On return, each process has exact copy of virtual memory

- Subsequent writes create new pages using COW mechanism.
To load and run a new program `a.out` in the current process using `execve`

- Free `vm_area_struct`'s and page tables for old areas
- Create `vm_area_struct`'s and page tables for new areas
  - Programs and initialized data backed by object files.
  - `.bss` and stack backed by anonymous files.
- Set PC to entry point in `.text`
  - Linux will fault in code and data pages as needed.

Memory mapped region for shared libraries

Program text (.text)

Initialized data (.data)

Uninitialized data (.bss)

Runtime heap (via malloc)

User stack

Private, demand-zero

Shared, file-backed

Private, file-backed

Private, demand-zero

Private, demand-zero

Private, demand-zero

Shared, file-backed
User-Level Memory Mapping

void *mmap(void *start, int len,
            int prot, int flags, int fd, int offset)

- Map `len` bytes starting at offset `offset` of the file specified by file description `fd`, preferably at address `start`
  - `start`: may be 0 for “pick an address”
  - `prot`: PROT_READ, PROT_WRITE, ...
  - `flags`: MAP_ANON, MAP_PRIVATE, MAP_SHARED, ...

- Return a pointer to start of mapped area (may not be `start`)
User-Level Memory Mapping

```c
void *mmap(void *start, int len,
           int prot, int flags, int fd, int offset)
```

- `len`: bytes
- `start`: (or address chosen by kernel)
- `len bytes` (or address chosen by kernel)
- `offset`: (bytes)
- `0` to `0`

Disk file specified by file descriptor `fd`

Process virtual memory
### Example: Using `mmap` to Copy Files

- Copying a file to `stdout` without transferring data to user space.

```c
#include "csapp.h"

void mmapcopy(int fd, int size)
{
    char *bufp;

    /* Ptr to memory mapped area */
    bufp = Mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
    Write(1, bufp, size);
    return;
}

int main(int argc, char **argv)
{
    struct stat stat;
    int fd;

    /* Check for required cmd line arg */
    if (argc != 2) {
        printf("usage: %s <filename>\n", argv[0]);
        exit(0);
    }

    /* Copy input file to stdout */
    fd = Open(argv[1], O_RDONLY, 0);
    Fstat(fd, &stat);
    mmapcopy(fd, stat.st_size);
    exit(0);
}
```

```c
/* mmapcopy driver */
int main(int argc, char **argv)
{
    struct stat stat;
    int fd;

    /* Check for required cmd line arg */
    if (argc != 2) {
        printf("usage: %s <filename>\n", argv[0]);
        exit(0);
    }

    /* Copy input file to stdout */
    fd = Open(argv[1], O_RDONLY, 0);
    Fstat(fd, &stat);
    mmapcopy(fd, stat.st_size);
    exit(0);
}
```
Summary: Virtual Memory & Caches

- **Components of the virtual address (VA)**
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: Virtual page offset
  - VPN: Virtual page number

- **Components of the physical address (PA)**
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag