Lecture 16: Paging
Instructor: Chengyu Song
Slide contributions from
Nael Abu-Ghazaleh, Harsha Madhyvasta and Zhiyun Qian

Some slides modified from originals by Dave O’hallaron
Recap: Address Spaces

- **Linear address space**: Ordered set of contiguous non-negative integer addresses:
  \[ \{0, 1, 2, 3 \ldots \} \]

- **Virtual address space**: Set of \( N = 2^n \) virtual addresses
  \[ \{0, 1, 2, 3, \ldots, N-1\} \]

- **Physical address space**: Set of \( M = 2^m \) physical addresses
  \[ \{0, 1, 2, 3, \ldots, M-1\} \]

- Clean distinction between data (bytes) and their attributes (addresses)
- Each object can now have multiple addresses
- Every byte in main memory:
  one physical address, one (or more) virtual addresses
Recap: Paging

- Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory.
Segmentation and Paging

- Can combine segmentation and paging
  - The x86 supports segments and paging
- Use segments to manage logically related units
  - Module, procedure, stack, file, data, etc.
  - Segments vary in size, but usually large (multiple pages)
- Use pages to partition segments into fixed size chunks
  - Makes segments easier to manage within physical memory
    » Segments become “pageable” – rather than moving segments into and out of memory, just move page portions of segment
  - Need to allocate page table entries only for those pieces of the segments that have themselves been allocated
- Tends to be complex...
Managing Page Tables

- Last lecture we computed the size of the page table for a 32-bit address space w/ 4K pages to be 4MB
  - This is far too much overhead for each process

- How can we reduce this overhead?
  - Observation: Only need to map the portion of the address space actually being used (tiny fraction of entire addr space)

- How do we only map what is being used?
  - Can dynamically extend page table

- Use another level of indirection: two-level page tables
Two-Level Page Tables

- Two-level page tables
  - Virtual addresses (VAs) have three parts:
    - Master page number, secondary page number, and offset
  - Master page table maps VAs to secondary page table
  - Secondary page table maps page number to physical page
  - Offset indicates where in physical page address is located
One-Level Page Lookups

Virtual Address

Page number

Offset

Page Table

Page frame

Physical Address

Page frame

Offset

Physical Memory
Two-Level Page Lookups

Virtual Address

Master page number  Secondary  Offset

Page table

Master Page Table

Secondary Page Table

Physical Address

Page frame  Offset

Physical Memory
Example

- How many bits in offset? $4K = 12$ bits
- 4KB pages, 4 bytes/PTE
- Want master page table in one page: $4K/4$ bytes = 1K entries
- Hence, 1K secondary page tables
- How many bits?
  - Master page number = 10 bits (because 1K entries)
  - Offset = 12 bits
  - Secondary page number = $32 - 10 - 12 = 10$ bits
A Two-Level Page Table Hierarchy

Level 1
page table

Level 2
page tables

Virtual memory

VP 0

... VP 1023

... VP 1024

... VP 2047

Gap

2K allocated VM pages for code and data

6K unallocated VM pages

1023 unallocated pages

1 allocated VM page for the stack

PTE 0

... PTE 1023

PTE 0

... PTE 1023

1023 null PTEs

PTE 1023

PTE 0

... PTE 1023

PTE 0

... PTE 1023

(1K - 9) null PTEs

PTE 8

PTE 7 (null)

PTE 6 (null)

PTE 5 (null)

PTE 4 (null)

PTE 3 (null)

PTE 2 (null)

PTE 1

PTE 0

32 bit addresses, 4KB pages, 4-byte PTEs

CS153 – Lecture 16 – Paging
Two-level Paging

- Two-level paging reduces memory overhead of paging
  - Only need one master page table and one secondary page table when a process begins
  - As address space grows, allocate more secondary page tables and add PTEs to master page table

- What problem remains?
  - Hint: what about memory lookups?
Efficient Translations

- Recall that our original page table scheme doubled the latency of doing memory lookups
  - One lookup into the page table, another to fetch the data
- Now two-level page tables triple the latency!
  - Two lookups into the page tables, a third to fetch the data
  - And this assumes the page table is in memory
- How can we use paging but also have lookups cost about the same as fetching from memory?
  - Cache translations in hardware
  - Translation Lookaside Buffer (TLB)
  - TLB managed by Memory Management Unit (MMU)
**TLBs**

- **Translation Lookaside Buffers**
  - Translate *virtual page #s* into PTEs *(not physical addr)*
  - Can be done in a single machine cycle

- **TLBs implemented in hardware**
  - Fully associative cache (all entries looked up in parallel)
    - Keys are virtual page numbers
    - Values are PTEs (entries from page tables)
  - With PTE + offset, can directly calculate physical address

- **Why does this help?**
  - Exploits locality: Processes use only handful of pages at a time
    - 16-48 entries/pages (64-192K)
    - Only need those pages to be “mapped”
  - Hit rates are therefore very important
A TLB hit eliminates one or more memory accesses
A TLB miss incurs an additional memory access (the PTE)
Fortunately, TLB misses are rare. Why?
Managing TLBs

- **Hit rate**: Address translations for most instructions are handled using the TLB
  - >99% of translations, but there are misses (**TLB miss**)...

- **Who places translations into the TLB (loads the TLB)?**
  - **Hardware (Memory Management Unit) [x86]**
    - Knows where page tables are in main memory
    - OS maintains tables, HW accesses them directly
    - Tables have to be in HW-defined format (inflexible)
  - **Software loaded TLB (OS) [MIPS, Alpha, Sparc, PowerPC]**
    - TLB faults to the OS, OS finds appropriate PTE, loads it in TLB
    - Must be fast (but still 20-200 cycles)
    - CPU ISA has instructions for manipulating TLB
    - Tables can be in any format convenient for OS (flexible)
Managing TLBs (2)

- OS ensures that TLB and page tables are consistent
  - When it changes the protection bits of a PTE, it needs to invalidate the PTE if it is in the TLB (special hardware instruction)
- Reload TLB on a process context switch
  - Invalidate all entries
  - Why? Who does it?
- When the TLB misses and a new PTE has to be loaded, a cached PTE must be evicted
  - Choosing PTE to evict is called the TLB replacement policy
  - Implemented in hardware, often simple, e.g., Least Recently Used (LRU)
Simple Memory System Example

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

![Virtual memory address diagram]

![Physical memory address diagram]
## 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>0D</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>
Simple Memory System TLB

- 16 entries
- 4-way associative

<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>

TLBT: 13 12 11 10 9 8 7 6 5 4 3 2 1 0
TLBI: 13 12 11 10 9 8 7 6 5 4 3 2 1 0
VPN: CS153 – Lecture 16 Paging
VPO: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
Simple Memory System Cache

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

<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 #1

Virtual Address: \(0x03D4\)

Physical Address

CO 0x0D  CT 0x0D  CT 0x0D  Hit? Y  Byte: 0x36

CS153 – Lecture 16 – Paging  22
Address Translation Example #2

Virtual Address: \texttt{0x0B8F}

<table>
<thead>
<tr>
<th>TLBT</th>
<th>TLBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VPN</th>
<th>VPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

VPN: \texttt{0x2E} \hspace{1cm} TLBI: 2 \hspace{1cm} TLBT: \texttt{0x0B} \hspace{1cm} TLB Hit? N \hspace{1cm} Page Fault? Y \hspace{1cm} PPN: TBD

Physical Address

<table>
<thead>
<tr>
<th>CT</th>
<th>CI</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PPN</th>
<th>PPO</th>
</tr>
</thead>
</table>

CO: \_\_ \hspace{1cm} CI: \_\_ \hspace{1cm} CT: \_\_ \hspace{1cm} Hit?: \_\_ \hspace{1cm} Byte: \_\_
Address Translation Example #3

Virtual Address: 0x0020

Physical Address

Byte: Mem
Intel Core i7 Memory System

Processor package

Core x4

- Registers
- Instruction fetch
- L1 d-cache 32 KB, 8-way
- L1 i-cache 32 KB, 8-way
- L2 unified cache 256 KB, 8-way
- 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)
- L3 unified cache 8 MB, 16-way (shared by all cores)

To other cores
To I/O bridge

Main memory
End-to-end Core i7 Address Translation

CPU

Virtual address (VA)

VPN
VPO

TLBT
TLBI

TLB
miss

TLB hit

L1 TLB (16 sets, 4 entries/set)

VPN1
VPN2
VPN3
VPN4

PTE

Page tables

CR3

VPN
VPO

36
12

32
4

9
9
9
9

9
9

9

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

32/64

L1
hit

Physical address (PA)

CS153 – Lecture 16 – Paging
### Core i7 Level 1-3 Page Table Entries

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

Available for OS (page table location on disk)  
P=0

**Each entry references a 4K child page table**

**P:** Child page table present in physical memory (1) or not (0).

**R/W:** Read-only or read-write access 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.

**CD:** Caching disabled or enabled 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).

**G:** Global page (don’t evict from TLB on task switch)

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

- Page Optimizations
  - Managing page tables (space)
  - Efficient translations (TLBs) (time)
Next time...

- Advanced Paging
- Preparation
  - Read Module 20