CSE 141 – Computer Architecture
Fall 2005

Lectures 17
Virtual Memory

Pramod V. Argade
November 23, 2005
Announcements

- **Final Review Discussion Section**
  - Wed. Nov. 30, 6:30 - 7:50, Center 216

- **Final Exam**
  **When:** Thursday., December 8, 7 - 9:59 PM
  **Where:** Center 216
# Course Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Lecture Topic</th>
<th>Quiz Topic</th>
<th>Homework Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9/26</td>
<td>Monday</td>
<td>Introduction, Ch. 1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>9/28</td>
<td>Wednesday</td>
<td>ISA, Ch. 2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>10/3</td>
<td>Monday</td>
<td>Arithmetic Part 1, Ch. 4</td>
<td>ISA</td>
<td>#1</td>
</tr>
<tr>
<td>4</td>
<td>10/5</td>
<td>Wednesday</td>
<td>Arithmetic Part 2, Ch. 4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>10/10</td>
<td>Monday</td>
<td>Performance, Ch. 3</td>
<td>Arithmetic</td>
<td>#2</td>
</tr>
<tr>
<td>6</td>
<td>10/12</td>
<td>Wednesday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>10/17</td>
<td>Monday</td>
<td>Single cycle CPU, Ch. 5</td>
<td>Performance</td>
<td>#3</td>
</tr>
<tr>
<td>8</td>
<td>10/19</td>
<td>Wednesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>10/24</td>
<td>Monday</td>
<td>Multi-cycle CPU, Ch. 5</td>
<td>Single Cycle CPU</td>
<td>#4</td>
</tr>
<tr>
<td>10</td>
<td>10/26</td>
<td>Wednesday</td>
<td>Review for the Midterm</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10/31</td>
<td>Monday</td>
<td>Mid-term Exam</td>
<td></td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>11/2</td>
<td>Wednesday</td>
<td>Exceptions, Ch. 5 and Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>11/7</td>
<td>Monday</td>
<td>Pipelining, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>11/9</td>
<td>Wednesday</td>
<td>Data and control hazards, Ch. 6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>11/14</td>
<td>Monday</td>
<td>Data and control hazards, Ch. 6</td>
<td>Pipeline Hazards</td>
<td>#5</td>
</tr>
<tr>
<td>15</td>
<td>11/16</td>
<td>Wednesday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>11/21</td>
<td>Monday</td>
<td>Memory &amp; cache design, Ch. 7</td>
<td>Cache</td>
<td>#6</td>
</tr>
<tr>
<td>17</td>
<td>11/23</td>
<td>Wednesday</td>
<td>Virtual Memory &amp; cache design, Ch. 7</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>18</td>
<td>11/28</td>
<td>Monday</td>
<td>Course Review</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12/8</td>
<td>Thursday</td>
<td>Final Exam 7:00 - 9:59 PM Center 216</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
Virtual Memory
Virtual Memory

- Virtual memory is the name of a technique that allows us to view main memory as a cache of a larger memory space (on disk).
  - Allows efficient and safe sharing of memory among programs
  - Creates an illusion of providing unlimited memory to programs
Virtual Memory (VM)

- Each program is compiled to run in its own address space
- A single program may exceed the size of primary memory
- Multiple programs may dynamically share portions of memory
- Main memory need contain only the active portions of the program
- VM and caching have different historical roots
- VM is like caching, but uses different terminology

<table>
<thead>
<tr>
<th>Cache</th>
<th>VM</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>page</td>
</tr>
<tr>
<td>cache miss</td>
<td>page fault</td>
</tr>
<tr>
<td>address</td>
<td>virtual address</td>
</tr>
<tr>
<td>index</td>
<td>physical address (sort of)</td>
</tr>
</tbody>
</table>
Advantages of Virtual Memory

- **Performance**
  - Large amount of memory accessed efficiently

- **Memory sharing among multiple programs**

- **Protection**
  - Simultaneous (time-sharing) execution of multiple programs
  - Use of “kernel space” and “user space”

- **Ease of programming/compilation**

- **Efficient use of memory**
Memory Mapping/Address Translation

- Virtual to physical address mapping
- Page may be present or absent in main memory
- Page may be resident on the disk
- Two virtual pages may map to the same physical address
Mapping Virtual to Physical Address

Virtual Address

Virtual Page Number  Page Offset

Page Table

Physical Page Number

Main Memory
Mapping from a Virtual to a Physical Address

Example:
- Virtual address space 4 Gbytes
- Physical address space 1 Gbytes
- Page size 4 Kbytes

Virtual Address

Virtual address

31 30 29 28 27 15 14 13 12 11 10 9 8 3 2 1 0

Virtual page number  Page offset

Translation

Physical page number  Page offset

Physical address
Address Translation via the Page Table

<table>
<thead>
<tr>
<th>Virtual address</th>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 15 14 13 12 11 10 9 8 3 2 1 0</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Valid</th>
<th>Physical page number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Physical address</th>
<th>Physical page number</th>
<th>Page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>29 28 27 15 14 13 12 11 10 9 8 3 2 1 0</td>
<td>18</td>
<td></td>
</tr>
</tbody>
</table>

Notes:
- The page table contains mapping for every possible virtual page
- Valid bit indicates whether the page is present in the main memory
- Extra bits in the page table are used for protection information
Cache vs Virtual Memory Access

- **Access time**
  
  time between when a read is requested and when the desired word arrives

- **Transfer time**
  
  time it takes to transfer the whole request

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level Cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (Page) size</td>
<td>16 - 128 bytes</td>
<td>4096 - 65,536 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1 - 2 clock cycles</td>
<td>40 - 100 clock cycles</td>
</tr>
<tr>
<td>Miss Penalty (Access time)</td>
<td>8 - 100 clock cycles</td>
<td>700,000 - 6,000,000 clock cycles</td>
</tr>
<tr>
<td>Miss Rate</td>
<td>0.5 - 10%</td>
<td>0.00001 - 0.001%</td>
</tr>
<tr>
<td>Data memory size</td>
<td>0.016 - 1 MB</td>
<td>16 - 8192 MB</td>
</tr>
</tbody>
</table>

- VM has very high miss penalty
  - large pages (4 KB to MBs)
  - associative mapping of pages (typically fully associative)
  - software handling of misses (but not hits!!)
  - *write-through* not an option, only *write-back*
Translation Look-aside Buffer

- Address translation could be expensive to perform for every memory access
  - page tables are stored in main memory
  - need to access page table before accessing data location

- Solution is to remember the last address translation so the mapping lookup can be skipped
  - use a translation buffer to hold the last N translations
TLB: Making Address Translation Fast

Translation Lookaside Buffer: A cache for address translations

![Diagram of TLB and Page Table]

- **Virtual page number**
- **Valid**
- **Tag**
- **Physical page address**

**TLB**

**Page Table Register**

**Physical memory**

**Disk storage**

**Page table**

**Physical page valid/disk address**
TLB and Cache

Virtual address

31 30 29  ... 15 14 13 12 11 10 9 8  ... 3 2 1 0

Virtual page number  Page offset

Valid

Dirty

Tag

Physical page number

Physical page number

Physical address tag

Physical address

Cache index

Byte offset

16

14

2

Valid

Tag

Data

Cache

Cache hit

Data

Physical address

Cache index

32
Example: Page Table Size

• What is the total page table size if:
  – Virtual address is 32 bits
  – 8 Kbytes page size
  – Assume that each page table entry is 4 bytes

• Page Table Size computation
Example: Address translation

- Using the page table shown, translate following 32-bit virtual addresses into physical addresses. Make entries in the TLB assuming LRU replacement.
  - The page size is 4 Kbytes
  - Addresses: 0x0000 3040, 0x0000 1040, 0x0000 2040

Virtual | Physical
--- | ---
0x0000 3040 | 0x0010 0
0x0000 1040 | 0x000D 2
0x0000 2040 | 0xdead 1

TLB

<table>
<thead>
<tr>
<th>Valid</th>
<th>Tag</th>
<th>Physical Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0010 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x000D 2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0xdead 1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x0023 9</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Page Table Register

<table>
<thead>
<tr>
<th>Valid</th>
<th>Page Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000 0</td>
</tr>
<tr>
<td>1</td>
<td>0x0000 1</td>
</tr>
<tr>
<td>0</td>
<td>0x0000 2</td>
</tr>
<tr>
<td>1</td>
<td>0x0000 3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Memory Access Events

- **TLB Miss**
  - Entry does not exist in the TLB
  - A page table entry is brought into the TLB

- **Page fault**
  - The valid bit is not set for the page table entry
  - A page of data from disk is brought in to the main memory

- **Cache miss**
  - Tag mismatch or valid bit not set
  - Cache line is brought from next level of memory hierarchy (depending on the policy for a write)
Processing a Page Fault

- If the valid bit for a virtual page is off, page fault occurs
- CPU generates an exception
- OS takes control
- OS finds the page in the next level of hierarchy (disk)
- OS decides where to place the requested page in memory
- OS copies the page from next level of hierarchy to memory
- OS sets valid bit in the page table entry for the virtual address
- OS returns from the exception
- Program re-executes the same instruction
- Page translation finds valid bit set for the virtual page
- Data access succeeds
What is a Process?

- Program state consists of:
  - Page tables, PC and the registers
- This state is referred to as a process
- Process is an instance of a program executing on a CPU
Implementing Protection with VM

- Protection is essential for:
  - Allowing single main memory to be shared among multiple processes
  - Prevent one process from writing into the memory space of another
  - Prevent a user process from modifying its own page tables
  - Controlling raw access to peripheral devices

- Hardware capabilities needed for protection
  - Two operating modes: user mode and kernel mode of execution
  - A portion of the CPU state that a user process can read, but not write
    - This is the usr/kernel mode bit
  - A mechanism to switch between user mode and kernel mode
    - Accomplished by a system call
Additional bits in the Page Table

- **User or Kernel bit**
  - This bit restricts access to some pages to kernel only

- **Write bit**
  - This bit restricts read-only or read/write access to a page

- **Referenced bit**
  - OS periodically sets this bit to zero
  - It is set by CPU hardware when the page is referenced
  - Used by OS for replacing the page with other memory pages

- **Dirty bit**
  - If a process writes to a page, the dirty bit is set
  - It is used by OS to write the page to secondary storage before replacing it
Virtual Memory Key Points

- How does virtual memory provide:
  - illusion of large main memory?
  - sharing?
  - performance?
  - protection?

- Virtual Memory requires twice as many memory accesses, so we cache page table entries in the TLB.

- Three things can go wrong on a memory access: cache miss, TLB miss, page fault.