CSE 141 – Computer Architecture
Fall 2003

Lectures 16
Virtual Memory

Pramod V. Argade
Schedule

<table>
<thead>
<tr>
<th>Lecture #</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Sep. 25</td>
<td>Thursday</td>
<td>Introduction, Ch. 1</td>
</tr>
<tr>
<td>2</td>
<td>Sep. 30</td>
<td>Tuesday</td>
<td>Performance, Ch. 2</td>
</tr>
<tr>
<td>3</td>
<td>Oct. 2</td>
<td>Thursday</td>
<td>ISA, Ch. 3</td>
</tr>
<tr>
<td>4</td>
<td>Oct. 7</td>
<td>Tuesday</td>
<td>Arithmetic, Ch. 4, Continued</td>
</tr>
<tr>
<td>5</td>
<td>Oct. 9</td>
<td>Thursday</td>
<td>ISA, Ch. 3</td>
</tr>
<tr>
<td>6</td>
<td>Oct. 14</td>
<td>Tuesday</td>
<td>Single cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>7</td>
<td>Oct. 16</td>
<td>Thursday</td>
<td>Single-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>8</td>
<td>Oct. 21</td>
<td>Tuesday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>9</td>
<td>Oct. 23</td>
<td>Thursday</td>
<td>Multi-cycle CPU, Ch. 5</td>
</tr>
<tr>
<td>10</td>
<td>Oct. 28</td>
<td>Tuesday</td>
<td>Classes cancelled due to wildfires</td>
</tr>
<tr>
<td>11</td>
<td>Oct. 30</td>
<td>Thursday</td>
<td>Exceptions and Review for Midterm</td>
</tr>
<tr>
<td>12</td>
<td>Nov. 4</td>
<td>Tuesday</td>
<td>Mid-term Exam</td>
</tr>
<tr>
<td>13</td>
<td>Nov. 6</td>
<td>Thursday</td>
<td>Pipeline, Ch. 6</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 11</td>
<td>Tuesday</td>
<td>Veteran's Day Holiday</td>
</tr>
<tr>
<td>14</td>
<td>Nov. 13</td>
<td>Thursday</td>
<td>Data hazards, Ch. 6</td>
</tr>
<tr>
<td>15</td>
<td>Nov. 18</td>
<td>Tuesday</td>
<td>Control hazards, Ch. 6</td>
</tr>
<tr>
<td>16</td>
<td>Nov. 20</td>
<td>Thursday</td>
<td>Memory Hierarchy Ch. 7</td>
</tr>
<tr>
<td>17</td>
<td>Nov. 25</td>
<td>Tuesday</td>
<td>Caches, Ch. 7</td>
</tr>
<tr>
<td>No Class</td>
<td>Nov. 27</td>
<td>Thursday</td>
<td>Thanksgiving Holiday</td>
</tr>
<tr>
<td>18</td>
<td>Dec. 2</td>
<td>Tuesday</td>
<td>Virtual Memory, Ch. 7</td>
</tr>
<tr>
<td>19</td>
<td>Dec. 4</td>
<td>Thursday</td>
<td>Course Review</td>
</tr>
<tr>
<td>20</td>
<td>Dec. 9</td>
<td>Tuesday</td>
<td>Final Exam</td>
</tr>
</tbody>
</table>

Virtual Memory

- Virtual memory is the name of the 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**
  - No need for overlays
- **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
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

Physical address

Physical page number Page offset

Physical page number Valid

If 0 then page is not present in memory

Address Translation via the Page Table

Page table register

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

Virtual page number Page offset

Valid

Physical page number

Page table

IF 0 then page is not present in memory

Physical page number Page offset

Physical address

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 (ties up bandwidth)

<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</td>
<td>8 - 100 clock cycles</td>
<td>700,000 - 6,000,000 clock cycles</td>
</tr>
<tr>
<td>(Access time)</td>
<td>(6 - 60 clock cycles)</td>
<td>(500,000 - 4,000,000 clock cycles)</td>
</tr>
<tr>
<td>(Transfer time)</td>
<td>(2 - 40 clock cycles)</td>
<td>(200,000 - 2,000,000 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

TLB and Cache
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

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

<table>
<thead>
<tr>
<th>Virtual Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000 3040</td>
<td>0x0010</td>
</tr>
<tr>
<td>0x0000 1040</td>
<td>0x000D</td>
</tr>
<tr>
<td>0x0000 2040</td>
<td>0xdead</td>
</tr>
</tbody>
</table>

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

- TLB Miss
  - Entry does not exist in the TLB
  - A mechanism must be provided to bring a page table entry into the TLB
- Page fault
  - The valid bit is not set for the page table entry
- Cache miss
  - Tag mismatch or valid bit not set
  - Cache line is brought in (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 returns from the exception
- Program re-executes the same instruction
- Page translation finds valid bit on 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 to share a single main memory among multiple processes
  - Prevent once 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 mode: 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 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.