Memory Hierarchy and Cache Design (4)

Virtual vs Real Address Spaces

Virtual vs Real Address Spaces

Virtual Memory - the logical concept

Virtual Memory vs Cache

<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>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>0.5–4%</td>
<td>0.00001–0.001%</td>
</tr>
<tr>
<td>Data memory size</td>
<td>0.016–1MB</td>
<td>16–4192 MB</td>
</tr>
</tbody>
</table>
Virtual Memory vs Cache

- Replacement
  - Cache - controlled by hardware
  - Virtual memory - controlled by operating system
  - HUGE miss penalty
- Size
  - Virtual memory size is determined by processor address size
  - Cache size is independent of processor address size
- Combined usage of secondary memory
  - Backing store of main memory
  - File system

Types of Virtual Memory

- Paged
  - Fixed size blocks (pages)
  - Size usually 4K-64K bytes
- Segmented
  - Variable size blocks
  - Sizes vary

Mapping

![Mapping Diagram]

Segmentation and Paging

- Segmentation
  - Organize Virtual Address Space in variable size blocks
  - Need information about the length of blocks also
  - Addresses from one segment to the next may not be considered contiguous
  - Virtual address
    - (Segment ID, Offset in Segment)
- Paging
  - Organize Virtual Address Space in fixed size blocks
  - Virtual Address
    - (Single Address)
Segmentation vs Paging

<table>
<thead>
<tr>
<th>Page</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words per address</td>
<td>One</td>
</tr>
<tr>
<td>Programmers visible?</td>
<td>Invisible to application programmer</td>
</tr>
<tr>
<td>Replacing a block</td>
<td>Trivial (all blocks are the same size)</td>
</tr>
<tr>
<td>Memory use inefficiency</td>
<td>Internal fragmentation (misalignment of page)</td>
</tr>
<tr>
<td>Efficient disk traffic</td>
<td>Yes (adjust page size to balance access time and transfer time)</td>
</tr>
</tbody>
</table>

External Fragmentation

Internal Fragmentation
Translating Virtual Addresses

Segmentation

Paging Address Translation

Four Questions for Virtual Memory

• Block Placement: Fully Associative
• Block Identification: Fully Associative
• Block Replacement: LRU or its variants
• Write Strategy: Write back & Write allocate
Block Placement

- Fully Associative - the obvious choice
  - Huge miss penalty - low miss rate is extremely important
  - VM managed by software in OS - much more flexible than cache managed by hardware

Block Identification

Paging

Logical to Physical Addresses

VAX Memory Management

- 32 Bit addresses
- 4 GB Virtual Address Space
- 512 Bytes per page 2**23 pages
- Real Address Space is limited to 1GB
Block Replacement

- Pure LRU is difficult, if not impossible, to implement
- The Clock Algorithm
  - Maintain a circular list of pages in physical memory
  - Use a use bit to track how recently a page is referenced
  - Use bit set whenever a page is referenced
  - On a page fault, look for a page with use bit = 0 (in the clock order)
  - Replaces a page that hasn’t been referenced for one complete revolution of the clock

Write Strategy

- Write back - the obvious choice

Fast Address Translation

- Page tables are large and often paged
- Need fast mechanism for address translation
- USE
  - TRANSLATION LOOK-ASIDE BUFFERS (TLB)
    - Associative memory
Page Size

- Size of page table is inversely proportional to the page size
- Transferring larger pages to and from the secondary storage is more efficient than smaller pages
- As TLB sizes are restricted, larger page sizes imply more address space can be translated
- Larger page sizes lead to higher internal fragmentation
- Larger pages lead to higher startup time for a process
Segmentation and Paging

<table>
<thead>
<tr>
<th></th>
<th>Segmentation</th>
<th>Paging</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Block size</td>
<td>Variable</td>
<td>Fixed</td>
</tr>
<tr>
<td>2. Placement</td>
<td>First fit, et al.</td>
<td>Fully associative</td>
</tr>
<tr>
<td>3. How found</td>
<td>Size+location</td>
<td>Page table registers</td>
</tr>
<tr>
<td>4. Replacement</td>
<td>by programmer</td>
<td>LRU</td>
</tr>
<tr>
<td>5. Write action</td>
<td>Write back</td>
<td>Write back</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragmentation</td>
<td>External</td>
</tr>
<tr>
<td>Programming</td>
<td>Visible</td>
</tr>
<tr>
<td>Protection</td>
<td>RO, RW, EX</td>
</tr>
<tr>
<td></td>
<td>Internal</td>
</tr>
<tr>
<td></td>
<td>Transparent</td>
</tr>
<tr>
<td></td>
<td>Difficult</td>
</tr>
</tbody>
</table>

Protection

- Have bounds register and check it for each access
  - Have at least two modes - Kernel and User
  - Have CPU states that can be used by a user but not modified - Base and Bound Registers...
  - Controlled change of mode - System Call

Example - AXP 21064

- 64-bit addresses
  - kseg - (bits 63 and 62 = 10)
    - Reserved for Operating system kernel
    - Uses no memory management
    - Uniform protection
  - Seg0 - Bit 63 = 0
  - seg 1 - (bits 63 and 62 = 11)
    - User processes use Seg 0 and Seg1
      - Mapped to pages with individual protection
      - Seg0 grows upwards and Seg 1 downwards

- Three level hierarchical page table
8KB pages
- Entries are 64 bit (8 bytes) long
  - 32 bit contain physical page frame number
  - 5 protection fields
    - Valid
    - User read enable
    - Kernel read enable
    - User write enable
    - Kernel write enable
  - Fields reserved for system software use
    - One page contains 1024 PTEs
    - Each level is 10 bits long
    - Leaves 21 bits to be defined!!!
  - For Seg0 all are 0 and for Seg1 all are 1
- Virtual addresses are 43 bits long and not 64
- Physical address are 34 bits, not 32+13
  - Requirement: physical address < virtual address

Address Spaces
- Influenced by page size
- E.g. Page Size: 64 KB
- Virtual address - 3 x 13 + 16 = 55 bits
- Physical address - 32 + 16 or 48 bits

Memory Hierarchy TLB Parameters
Use two TLBs - one for instructions and one for data
OS can tell TLB to treat contiguous sequences of pages as one
- Block Size 1 PTE (8Bytes)
- Hit Time 1 clock cycle
- Miss Penalty(Avg) 20 cycles
- TLB Size Instruction: 8PTE for 8KB pages
  4 PTE for 4 MB pages
  (96 bytes total)
  Data: 32 PTE for 8-KB, 64 KB, 512KB, 4 MB
  (256 Bytes total)
- Block Selection Random, but not last used
- Write Strategy (Not Applicable)
- Block Placement Fully Associative
VAX Memory Management

- 32 Bit addresses
- 4 GB Virtual Address Space
- 512 Bytes per page $2^{23}$ pages
- Real Address Space is limited to 1GB

Access Control

- Access Level
  - No Access
  - Read Only Access
  - Read/Write Access

- Access Mode
  - Kernel
  - Kernel functions of the OS
  - Executive
  - System Calls
  - Supervisor
  - Operating system services e.g. responses to user commands
  - User

Each page in memory is defined to have a particular access level for each access mode.
VAX Address Translation

VAX TLB

- 2 Way Set Associative
- 64 entries per set
  - Divided into 32 entries for system and 32 for process

VAX TLB

VAX Virtual Memory Scheme
System/370 Memory Management

- Two level
  - Segments
  - Pages

- Page size
  - 2KB or 4KB
- Segment Size
  - 64KB or 1MB

- For 370/XA 4KB pages and 1M segments

Address Formats

INTEL 80386 Memory Management

- Both segmentation and paging
- Four views of the memory
  - Unsegmented and Unpaged
  - Virtual addresses are same as real addresses
  - Unsegmented paged
    - used by Berkeley UNIX
  - Segmented unpaged
    - Segmented andpaged
    - used by Unix System 5

Structure

- One segment table for each virtual address space
- One page table for each segment
### Summary

<table>
<thead>
<tr>
<th>TLB</th>
<th>First-level cache</th>
<th>Second-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block size</td>
<td>4-8 bytes (1 PTE)</td>
<td>4-32 bytes</td>
<td>32-256 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1 clock cycle</td>
<td>1-2 clock cycles</td>
<td>6-15 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>16-30 clock cycles</td>
<td>4-6 clock cycles</td>
<td>30-200 clock cycles</td>
</tr>
<tr>
<td>Miss rate (limit)</td>
<td>0.1-2%</td>
<td>0.3-2%</td>
<td>15-30%</td>
</tr>
<tr>
<td>Size</td>
<td>32-64KB bytes (8-1024 PTEs)</td>
<td>1-128 KB</td>
<td>256 KB-16MB</td>
</tr>
<tr>
<td>Backing store</td>
<td>First level cache</td>
<td>Second level cache</td>
<td>Page mode DRAM</td>
</tr>
<tr>
<td>Q1: Block placement</td>
<td>Fully associative or set associative</td>
<td>Direct mapped</td>
<td>Direct mapped or set associative</td>
</tr>
<tr>
<td>Q1: Block identification</td>
<td>Tag/block</td>
<td>Tag/block</td>
<td>Tag/block</td>
</tr>
<tr>
<td>Q1: Block replacement</td>
<td>Random</td>
<td>N/A (direct mapped)</td>
<td>Random</td>
</tr>
<tr>
<td>Q1: write strategy</td>
<td>Flush on a write to page table</td>
<td>Write through or write back</td>
<td>Write back</td>
</tr>
</tbody>
</table>