CMSC 611: Advanced Computer Architecture

Virtual Memory
Memory Hierarchy

Capacity
- CPU Registers: 100s Bytes, <10s ns
- Cache: K Bytes, 10-40 ns
- Main Memory: M Bytes, 70ns-1us
- Disk: G Bytes, ms
- Tape: infinite sec-min

Access Time
- Prog./compiler: 1-8 bytes
- cache cntl: 8-128 bytes
- OS: 512-4K bytes
- user/operator: Mbytes

Upper Level
- faster

Lower Level
- Larger

Staging Transfer Unit

Registers

Instr. Operands

Cache

Blocks

Main Memory

Pages

Disk

Files

Tape

Upper Level

faster

Lower Level

Larger
Virtual Memory

- Using virtual addressing, main memory plays the role of cache for disks.
- The virtual space is much larger than the physical memory space.
- Physical main memory contains only the active portion of the virtual space.
- Address space can be divided into fixed size (pages) or variable size (segments) blocks.

Cache

Virtual memory

Block ➞ Page

Cache miss ➞ page fault

Block addressing ➞ Address translation
### Virtual Memory

**Advantages**
- Allows efficient and safe data sharing of memory among multiple programs
- Moves programming burdens of a small, limited amount of main memory
- Simplifies program loading and avoid the need for contiguous memory block
- allows programs to be loaded at any physical memory location

<table>
<thead>
<tr>
<th><strong>Cache</strong></th>
<th><strong>Virtual memory</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>Page</td>
</tr>
<tr>
<td>Cache miss</td>
<td>page fault</td>
</tr>
<tr>
<td>Block addressing</td>
<td>Address translation</td>
</tr>
</tbody>
</table>
Virtual Addressing

- Page faults are costly and take millions of cycles to process (disks are slow)
- Optimization Strategies:
  - Pages should be large enough to amortize the access time
  - Fully associative placement of pages reduces page fault rate
  - Software-based so can use clever page placement
  - Write-through can make writing very time consuming (use copy back)
Page Table

- Page table:
  - Resides in main memory
  - One entry per virtual page
  - No tag is required since it covers all virtual pages
  - Point directly to physical page
  - Table can be very large
  - Operating sys. may maintain one page table per process
  - A dirty bit is used to track modified pages for copy back
Page Faults

- A page fault happens when the valid bit of a virtual page is off.
- A page fault generates an exception to be handled by the operating system to bring the page to main memory from a disk.
- The operating system creates space for all pages on disk and keeps track of the location of pages in main memory and disk.
- Page location on disk can be stored in page table or in an auxiliary structure.
- LRU page replacement strategy is the most common.
- Simplest LRU implementation uses a reference bit per page and periodically reset reference bits.
Optimizing Page Table Size

With a 32-bit virtual address, 4-KB pages, and 4 bytes per page table entry:

Number of page table entries = \( \frac{2^{32}}{2^{12}} = 2^{20} \)

Size of page table = \( 2^{20} \) page table entries \( \times \) \( \frac{2^2}{\text{bytes}} \) page table entry = 4 MB

- Optimization techniques:
  - Keep bound registers to limit the size of page table for given process in order to avoid empty slots
  - Store only physical pages and apply hashing function of the virtual address (inverted page table)
  - Use multi-level page table to limit size of the table residing in main memory
  - Allow paging of the page table
  - Cache the most used pages ⇒ Translation Look-aside Buffer
Multi-Level Page Table

32-bit address:

<table>
<thead>
<tr>
<th>P1 index</th>
<th>P2 index</th>
<th>page offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

- 2 GB virtual address space
- 4 MB of PTE2
  - paged, holes
- 4 KB of PTE1

Inverted page table can be the only practical solution for huge address space, e.g., 64-bit address space
### Translation Look-aside Buffer

- Special cache for recently used translation
- TLB misses are typically handled as exceptions by operating system
- Simple replacement strategy since TLB misses happen frequently

#### Diagram

- **Virtual page number**
- **Valid**
- **Tag**
- **Physical page address**
- **Page table**
- **Physical page**
- **Valid or disk address**
- **Physical memory**
- **Disk storage**
Avoiding Address Translation

- Send virtual address to cache?
  - Called Virtually Addressed Cache or just Virtual Cache vs. Physical Cache
  - Every time process is switched logically must flush the cache; otherwise get false hits
    - Cost is time to flush + “compulsory” misses from empty cache
  - Dealing with aliases (sometimes called synonyms)
    - Two different virtual addresses map to same physical address causing unnecessary read misses or even RAW
  - I/O must interact with cache, so need virtual address
- Solution to aliases
  - HW guarantees that every cache block has unique physical address (simply check all cache entries)
  - SW guarantee: lower n bits must have same address so that it overlaps with index; as long as covers index field & direct mapped, they must be unique; called page coloring

- Solution to cache flush
  - Add process identifier tag that identifies process as well as address within process: cannot get a hit if wrong process
Impact of Using Process ID

- Miss rate vs. virtually addressed cache size of a program measured three ways:
  - Without process switches (uniprocessor)
  - With process switches using a PID tag (PID)
  - With process switches but without PID (purge)
Virtually Addressed Caches

VA: Virtual address  TB: Translation buffer  PA: Page address

Conventional Organization

Virtually Addressed Cache

Translate only on miss

Synonym Problem

Overlap $ access with VA translation: requires $ index to remain invariant across translation
Indexing via Physical Addresses

• If index is physical part of address, can start tag access in parallel with translation
• To get the best of the physical and virtual caches, use the page offset (not affected by the address translation) to index the cache
• The drawback is that direct-mapped caches cannot be bigger than the page size (typically 4-KB)

• To support bigger caches and use same technique:
  – Use higher associativity since the tag size gets smaller
  – OS implements page coloring since it will fix a few least significant bits in the address (move part of the index to the tag)
TLB and Cache in MIPS

Fully associative TLB

Address translation and block identification

Direct-mapped Cache
A cache hit can only occur after TLB hit
(TLB miss & No Page fault ➔ load page address to TLB)

Virtual address ➔ TLB access

TLB hit? No ➔ TLB miss exception
Yes ➔ Physical address

Write? No ➔ Try to read data from cache
Yes ➔ Write access bit on?

Write access bit on? No ➔ Write protection exception
Yes ➔ Write data into cache, update the tag, and put the data and the address into the write buffer

Cache hit? No ➔ Cache miss stall
Yes ➔ Deliver data to the CPU

Write?
## Memory Related Exceptions

**Possible exceptions:**

**Cache miss:** referenced block not in cache and needs to be fetched from main memory

**TLB miss:** referenced page of virtual address needs to be checked in the page table

**Page fault:** referenced page is not in main memory and needs to be copied from disk

<table>
<thead>
<tr>
<th>Cache</th>
<th>TLB</th>
<th>Page fault</th>
<th>Possible? If so, under what condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>miss</td>
<td>hit</td>
<td>hit</td>
<td>Possible, although the page table is never really checked if TLB hits</td>
</tr>
<tr>
<td>hit</td>
<td>miss</td>
<td>hit</td>
<td>TLB misses, but entry found in page table and data found in cache</td>
</tr>
<tr>
<td>miss</td>
<td>miss</td>
<td>hit</td>
<td>TLB misses, but entry found in page table and data misses in cache</td>
</tr>
<tr>
<td>miss</td>
<td>miss</td>
<td>miss</td>
<td>TLB misses and followed by page fault. Data must miss in cache</td>
</tr>
<tr>
<td>miss</td>
<td>hit</td>
<td>miss</td>
<td>Impossible: cannot have a translation in TLB if page is not in memory</td>
</tr>
<tr>
<td>hit</td>
<td>hit</td>
<td>miss</td>
<td>Impossible: cannot have a translation in TLB if page is not in memory</td>
</tr>
<tr>
<td>hit</td>
<td>miss</td>
<td>miss</td>
<td>Impossible: data is not allowed in cache if page is not in memory</td>
</tr>
</tbody>
</table>
Memory Protection

• Want to prevent a process from corrupting memory space of other processes
  – Privileged and non-privileged execution
• Implementation can map independent virtual pages to separate physical pages
• Write protection bits in the page table for authentication
• Sharing pages through mapping virtual pages of different processes to same physical pages
Memory Protection

• To enable the operating system to implement protection, the hardware must provide at least the following capabilities:
  – Support at least two mode of operations, one of them is a user mode
  – Provide a portion of CPU state that a user process can read but not write,
    • e.g. page pointer and TLB
  – Enable change of operation modes through special instructions