Review

- **Examples** of direct mapped cache and memory address partition
- Direct-mapped cache with **larger block size**
- Memory organization supports **larger block size**  
  - With **interleaved memory bank**
- **Cache Performance**  
  - $\text{execution time} = (\text{execution cycles} + \text{memory stall cycles}) \times \text{cycle time}$, where  
    $\text{stall cycles} = \# \text{ of instructions} \times \text{miss ratio} \times \text{miss penalty}$  
  - $\text{AMAT} = \text{hit time} + \text{miss rate} \times \text{miss penalty}$
- How to improve performance?  
  - Larger block size, larger cache, higher associativity, use split caches, Multi-level cache

5.4 Virtual Memory

- **Physical Memory**  
  - Installed in Computer: for example 512MB RAM in PC  
  - Limited by **size and power** supply of processor  
  - Potential extent limited by size of **address space**
- **Virtual Memory**  
  - system memory simulated by the hard drive  
  - How to put $2^{32}$ **words** in your PC?  
  - Not as RAM – not enough space or power  
  - Make the CPU believe it has $2^{32}$ **words**  
  - Have the Main Memory act as a long-term Cache  
  - **Page the main memory contents to/from disk**  
  - Paging table (address system) works with Memory Management Unit to control page storage/retrieval
Virtual Memory Benefits

- Virtual memory (VM) allows main memory (DRAM) to act like a cache for secondary storage (magnetic disk).
- VM address translation provides a mapping from the virtual address of the processor to the physical address in main memory and secondary storage.
- VM provides the following benefits
  - Allows multiple programs to share the same physical memory
  - Allows programmers to write code (or compilers to generate code) as though they have a very large amount of main memory
  - Automatically handles bringing in data from disk

Virtual Memory (VM)

Observe: Cache is a cache for main memory

*** Main Memory is a Cache for Disk Storage ***

Justification:

- VM allows efficient and safe sharing of memory among multiple programs (multiprogramming support)
- VM removes the programming headaches of a small amount of physical memory
- VM simplifies loading the program by supporting relocation

History: VM was developed first, then cache

Cache is based on VM technology, not conversely
### Virtual Memory Terms

**Page**: A virtual memory block (cache = block, VM = page)

**Page Fault**: A miss on MemRead (cache = miss, VM = page fault)

**Physical Address**: Where data is stored in physical memory

**Virtual Address**: Produced by CPU which sees a big address space, then translated by HW + SW to yield a physical address

**Memory Mapping or Address Translation**:
- Process of transforming Virtual Address to Physical Address

**Translation Lookaside Buffer (TLB)**:
- Helps make memory mapping more efficient

### Comparing the 2 levels of hierarchy

- **Cache Version**
  - Block (or Line)
  - Miss
  - Block Size: 32-64B
  - Placement: Direct Mapped, N-way Set Associative
  - Replacement: Least Recently Used (LRU) or Random (LRU)
  - Write Thru or Back
  - Miss handling: hardware

- **Virtual Memory version**
  - Page
  - Page Fault
  - Page Size: 4K-8KB
  - Fully Associative
  - Least Recently Used (LRU)
  - Write Back
  - Software
Virtual Memory Cost

Page Fault: Re-fetch the data from disk \[\text{millisecond latency}\]

Reducing VM Cost:

• Make pages large enough to amortize high disk access time
• Allow fully associative page placement to reduce page fault rate
• Handle page faults in software, because you have a lot of time between disk accesses (versus cache, which is v. fast)
• Use clever software algorithms to place pages (we have the time)

Random page replacement, versus Least Recently Used
• Use Write-back (faster) instead of write-through (too long)

4 Qs for Virtual Memory

• Q1: Where can a block be placed in the upper level?
  — Miss penalty for virtual memory is very high
  — Have software determine location of block while accessing disk
  — Allow blocks to be placed anywhere in memory (fully associative) to reduce miss rate.
• Q2: How is a block found if it is in the upper level?
  — Address divided into page number and page offset
  — Page table and translation buffer used for address translation
• Q3: Which block should be replaced on a miss?
  — Want to reduce miss rate & can handle in software
  — Least Recently Used typically used
• Q4: What happens on a write?
  — Writing to disk is very expensive
  — Use a write-back strategy
Virtual Memory Process

- A virtual address consists of a virtual page number and a page offset.
- The virtual page number gets translated to a physical page number.
- The page offset is not changed.

Virtual Address:

<table>
<thead>
<tr>
<th>N bits</th>
<th>K</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual page number</td>
<td>Page offset</td>
<td></td>
</tr>
</tbody>
</table>

Translation

Physical Address:

<table>
<thead>
<tr>
<th>M bits</th>
<th>K</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Page Number</td>
<td>Page offset</td>
<td></td>
</tr>
</tbody>
</table>

M is always less than N

Otherwise, why have VM?

Paging Organization (assume 1 KB pages)

Physical Address

| 0   | page 0  |
| 1024 | page 1  |
| ...  | ...     |
| 7168 | page 7  |

Virtual Address

| 0   | page 0  |
| 1024 | page 1  |
| 2048 | page 2  |
| ...  | ...     |
| 31744| page 31 |

Page is unit of mapping

Page also unit of transfer from disk to physical memory

Physical Memory

Virtual Memory
Address Translation with Page Tables

- A page table translates a virtual page number into a physical page number.
- A page table register indicates the start of the page table.
- The virtual page number is used as an index into the page table that contains
  - The physical page number
  - A valid bit that indicates if the page is present in main memory
  - A dirty bit to indicate if the page has been written
  - Protection information about the page (read only, read/write, etc.)
- Since page tables contain a mapping for every virtual page, no tags are required.

Page Table Diagram
Accessing Main Memory or Disk

- If the valid bit of the page table is zero, this means that the page is not in main memory.
- In this case, a page fault occurs, and the missing page is read in from disk.

Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written
Determining Page Table Size

- Assume
  - 32-bit virtual address
  - 30-bit physical address
  - 4 KB pages => 12 bit page offset
  - Each page table entry is one word (4 bytes)
- How large is the page table?
  - Virtual page number = 32 - 12 = 20 bytes
  - Number of entries = number of pages = 2^20
  - Total size = number of entries x bytes/entry
    = 2^20 x 4 = 4 Mbytes
  - Each process running needs its own page table
- Since page tables are very large, they are almost always stored in main memory, which makes them slow.

Caching Virtual Addresses

- Virtual memory seems to be really slow:
  - Must access memory on load/store -- even cache hits!
  - Worse, if translation not completely in memory, may need to go to disk before hitting in cache!
- Solution: Caching! (surprise!)
  - Keep track of most common translations and place them in a “Translation Lookaside Buffer” (TLB)
Making address translation practical: TLB

- Virtual memory => memory acts like a cache for the disk
- Page table maps virtual page numbers to physical frames
- Translation Look-aside Buffer (TLB) is a cache for translations

Translation-Lookaside Buffer (TLB)

- A TLB acts as a cache for the page table, by storing physical addresses of pages that have been recently accessed.
TLB Characteristics

- The following are characteristics of TLBs
  - TLB size: 32 to 4,096 entries
  - Block size: 1 or 2 page table entries (4 or 8 bytes each)
  - Hit time: 0.5 to 1 clock cycle
  - Miss penalty: 10 to 30 clock cycles (go to page table)
  - Miss rate: 0.01% to 0.1%
  - Associative: Fully associative or set associative
  - Write policy: Write back (replace infrequently)
- The MIPS R2000 TLB has the following characteristics
  - TLB size: 64 entries
  - Block size: 1 entry of 64 bits (20 bit tag, 1 valid bit, 1 dirty bit, several bookkeeping bits)
  - Hit time: 0.5 clock cycles
  - Miss penalty: Average of 16 cycles
  - Associative: Fully associative
  - Write policy: write back

TLB and Cache Interaction

- If cache tag uses physical address
  - Need to translate before cache lookup
- Alternative: use virtual address tag
  - Complications due to aliasing
    - Different virtual addresses for shared physical address
Handling TLB Misses and Page Faults

- When a TLB miss occurs either
  - Page is present in memory and update the TLB
    - occurs if valid bit of page table is set
  - Page is not present in memory and O.S. gets control to handle a page fault
- If a page fault occur, the operating system
  - Access the page table to determine the physical location of the page on disk
  - Chooses a physical page to replace - if the replaced page is dirty it is written to disk
  - Reads a page from disk into the chosen physical page in main memory.
- Since the disk access takes so long, another process is typically allowed to run during a page fault.
### Virtual Memory Summary

- Virtual memory (VM) allows main memory (DRAM) to act like a cache for secondary storage (magnetic disk).
- Page tables and TLBS are used to translate the virtual address to a physical address.
- The large miss penalty of virtual memory leads to different strategies from cache:
  - Fully associative
  - LRU or LRU approximation
  - Write-back
  - Done by software

### 5.6 Virtual Machines

- Host computer emulates guest operating system and machine resources
  - Improved isolation of multiple guests
  - Avoids security and reliability problems
  - Aids sharing of resources
- Virtualization has some performance impact
  - Feasible with modern high-performance computers
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare
  - Microsoft Virtual PC
Virtual Machine Monitor

- Maps virtual resources to physical resources
  - Memory, I/O devices, CPUs
- Guest code runs on native machine in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

Example: Timer Virtualization

- In native machine, on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs
Instruction Set Support

• User and System modes
• Privileged instructions only available in system mode
  – Trap to system if executed in user mode
• All physical resources only accessible using privileged instructions
  – Including page tables, interrupt controls, I/O registers
• Renaissance of virtualization support
  – Current ISAs (e.g., x86) adapting

Conclusions

• Caching improves memory efficiency by:
  – Restricting frequently-occurring read/write operations to fast memory
  – Buffering register-to-memory access, thereby leveling resource use
  – Keeping the I/O pipeline almost always full (occupied) to maximize hierarchical memory system throughput
• Virtual Memory is useful because:
  – Maps a big symbolic address space to a small physical address space – works almost the same as cache