EE 156/CS140: Advanced Topics in Computer Architecture

Spring 2023

Tufts University

Instructor: Prof. Mark Hempstead
mark@ece.tufts.edu

Lecture 5: Virtual Memory

Unit 1: The Memory Hierarchy

Unit 1: The Memory Hierarchy (3-4 weeks)

- **Introduction and Performance Metrics [Chapter 1]**
- **Review of Basic Caches and Set Associativity [Appendix B]**
- **Advanced Cache Optimization Techniques and Replacement policies [Ch 2]**
  - **Cache Miss Types (3 Cs)**
  - **Cache Hierarchies**
  - **10 Optimizations (From Textbook)**
- **Prefetching [Extra Slides]**
- **Memory consistency and Cache coherence [Chapter 5]**
- **Software interfaces and memory consistency**
- **Transactional memory (paper)**
- **Review of Virtual Memory and TLBs [Appendix B]**
- **Advanced Virtual Memory [SLCA: Bhattacharjee and Lustig] (Paper)**
- New Non-Volatile Memory (NVM) technologies

- **Textbook Reading: Read Sections B.4, B.5**

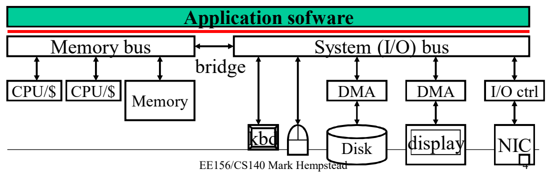
A Review of

VIRTUAL MEMORY

[B.4, B.5]

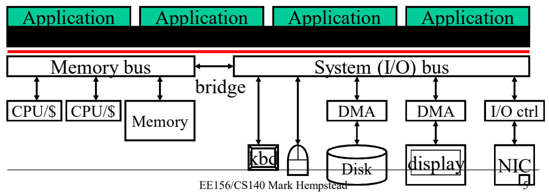
A Computer System: + App Software

- **Application software**: computer must do something

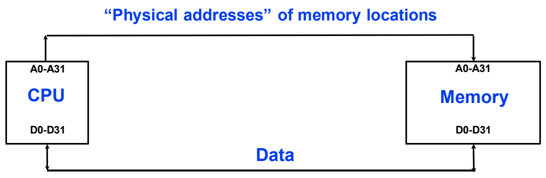


A Computer System: + OS

- **Operating System (OS):** virtualizes hardware for apps
  - **Abstraction**: provides **services** (e.g., threads, files, etc.)
    - + Simplifies app programming model, raw hardware is nasty
  - **Isolation**: gives each app illusion of private CPU, memory, I/O
    - + Simplifies app programming model
    - + Increases hardware resource utilization



The Limits of Physical Addressing

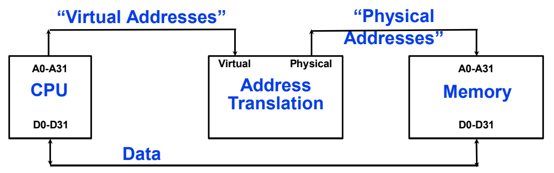


- •All programs share one address space: The **physical** address space
- •Machine language programs must be aware of the machine organization
- •No way to prevent a program from accessing **any machine resource**

Virtualizing Main Memory

- How do multiple apps (and the OS) share main memory?
  - **Goal: each application thinks it has infinite memory**
- One app may want more memory than is in the system
  - App's insn/data footprint may be larger than main memory
  - **Requires main memory to act like a cache**
    - With disk as next level in memory hierarchy (slow)
    - Write-back, write-allocate, large blocks or "pages"
  - No notion of "program not fitting" in registers or caches (why?)
- Solution:
  - Part #1: treat memory as a "cache"
    - Store the overflowed blocks in "swap" space on disk
  - Part #2: add a level of indirection (address translation)

Solution: Add a Layer of Indirection

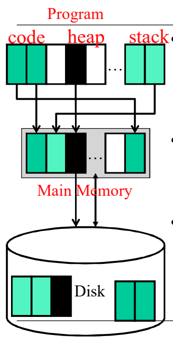


User programs run in an standardized virtual address space

Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

Hardware supports "modern" OS features: Protection, Translation, Sharing

Virtual Memory (VM)

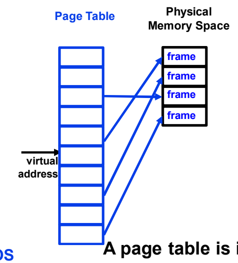


- Programs use **virtual addresses (VA)**
  - $0 \ldots 2^N-1$
  - VA size also referred to as machine size
  - E.g., 32-bit (embedded) or 64-bit (server)
- Memory uses **physical addresses (PA)**
  - $0 \ldots 2^M-1$ (typically M<N, especially if N=64)
  - 2^M is most physical memory machine supports
- VA→PA at **page** granularity (VP→PP)
  - By "system"
  - Mapping need not preserve contiguity
  - VP need not be mapped to any PP
  - Unmapped VPs live on disk (swap)

Three Advantages of Virtual Memory

- Translation:
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- Protection:
  - Different threads (or processes) protected from each other.
  - Different pages can be given special behavior (Read Only, Invisible to user programs, etc).
  - Kernel data protected from User programs
  - Very important for protection from malicious programs
- Sharing:
  - Can map same physical page to multiple users ("Shared memory")

Page tables encode virtual address spaces



A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):

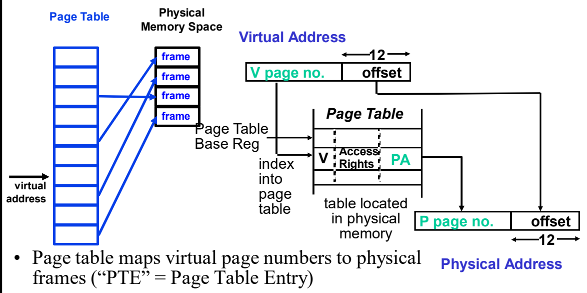
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

A page table is indexed by a virtual address

OS manages the page table for each ASID

A valid page table entry codes physical memory "frame" address for the page

Details of Page Table



- Page table maps virtual page numbers to physical frames ("PTE" = Page Table Entry)
- Virtual memory => treat memory ≈ cache for disk
- "Dirty bit" used to mark a page in use

Page tables may not fit in memory!

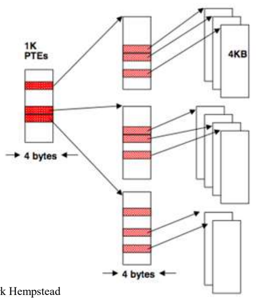
A table for 4KB pages for a 32-bit address space has 1M entries

Each process needs its own address space!

Two-level Page Tables

32 bit virtual address

31 — 22	21 — 12	11 — 0
P1 index	P2 index	Page Offset

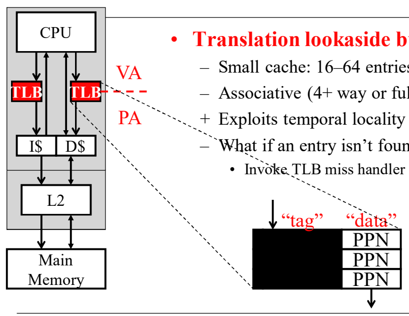


Top-level table wired in main memory

Subset of 1024 second-level tables in main memory; rest are on disk or unallocated

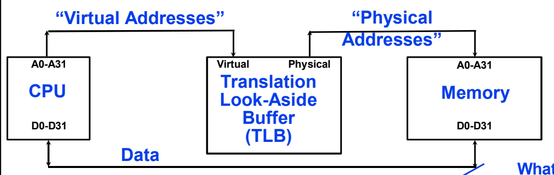
TLB DESIGN CONCEPTS (FAST TRANSLATION OF VIRTUAL ADDRESSES) [B.4]

Translation Lookaside Buffer



- **Translation lookaside buffer (TLB)**
  - – Small cache: 16–64 entries
  - – Associative (4+ way or fully associative)
  - + Exploits temporal locality in page table
  - – What if an entry isn't found in the TLB?
    - Invoke TLB miss handler

MIPS Address Translation: How does it work?



Translation Look-Aside Buffer (TLB) A small fully-associative cache of mappings from virtual to physical addresses

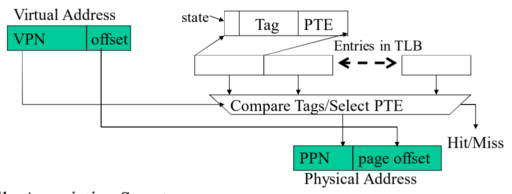
What is the table of mappings that it caches?

TLB also contains protection bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read/write it.

EE156/CS140 Mark Hempstead 16

Basic TLB Organization



- Fully Associative Structure
- Example: VA = 44bits, Page Size = 4MB, PA Space = 1GB
- VPN bits = bits (VA) - $\log_2$(page size) = 44 – 22 = 22 bits
- Physical Addr. = $\log_2$(PA Size) = 30 bits (8 PPN + 22 Page Offset)

EE156/CS140 Mark Hempstead 17

TLB Organization

- **Like caches**: TLBs also have ABCs
  - Capacity
  - Associativity (At least 4-way associative, fully-associative common)
  - What does it mean for a TLB to have a block size of two?
    - Two consecutive VPs share a single tag
  - **Like caches**: there can be second-level TLBs
- Example: AMD Opteron
  - 32-entry fully-assoc. TLBs, 512-entry 4-way L2 TLB (insn & data)
  - 4KB pages, 48-bit virtual addresses, four-level page table
- **Rule of thumb**: TLB should "cover" size of on-chip caches
  - In other words: (#PTEs in TLB) * page size $\geq$ cache size
  - Why? Consider relative miss latency in each…

EE156/CS140 Mark Hempstead 18

TLB Misses

- **TLB miss:** translation not in TLB, but in page table
  - Two ways to "fill" it, both relatively fast
- **Software-managed TLB**: e.g., Alpha, MIPS
  - Short (~10 insn) OS routine walks page table, updates TLB
  - + Keeps page table format flexible
  - Latency: one or two memory accesses + OS call (pipeline flush)
- **Hardware-managed TLB**: e.g., x86, recent SPARC, ARM
  - Page table root in hardware register, hardware "walks" table
  - + Latency: saves cost of OS call (avoids pipeline flush)
  - Page table format is hard-coded
- Trend is towards hardware TLB miss handler

Page Faults

- **Page fault**: PTE not in TLB or page table
  - → page not in memory
  - Or no valid mapping → segmentation fault
  - Starts out as a TLB miss, detected by OS/hardware handler
- **OS software routine**:
  - Choose a physical page to replace
    - **"Working set"**: refined LRU, tracks active page usage
  - If dirty, write to disk
  - Read missing page from disk
    - Takes so long (~10ms), OS schedules another task
  - Requires yet another data structure: **frame map**
    - Maps physical pages to <process, virtual page> pairs
  - Treat like a normal TLB miss from here

Virtual Memory Summary

- OS virtualizes memory and I/O devices
- Virtual memory
  - "infinite" memory, isolation, protection, inter-process communication
  - Page tables
  - Translation buffers
    - Parallel vs serial access, interaction with caching
  - Page faults
- Want to know more virtual memory implementation details:
  - Read: "Architectural and Operating System Support for Virtual Memory"
    https://www.morganclaypool.com/toc/cac/1/1