Address Translation

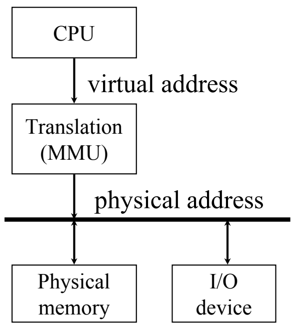
Tore Larsen

Material developed by:
Kai Li, Princeton University

Why Virtual Memory?

- Use secondary storage
  - Extend expensive DRAM with reasonable performance
- Provide Protection
  - Programs do not step over each other, communicate with each other require explicit IPC operations
- Convenience
  - Flat address space and programs have the same view of the world

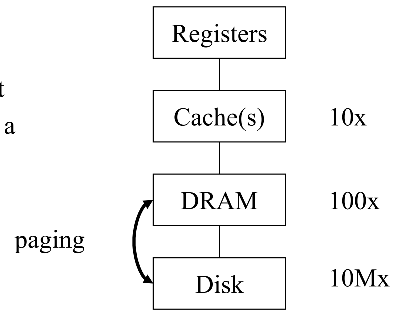
Translation Overview



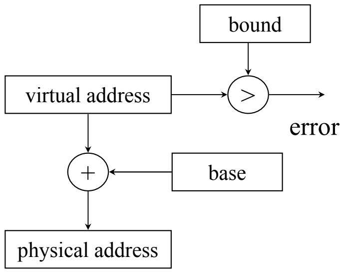
- Actual translation is in hardware (MMU)
- Controlled in software
- CPU view
  - what program sees, virtual memory
- Memory view
  - physical memory

Goals of Translation

- Implicit translation for each memory reference
- A hit should be very fast
- Trigger an exception on a miss
- Protected from user's faults

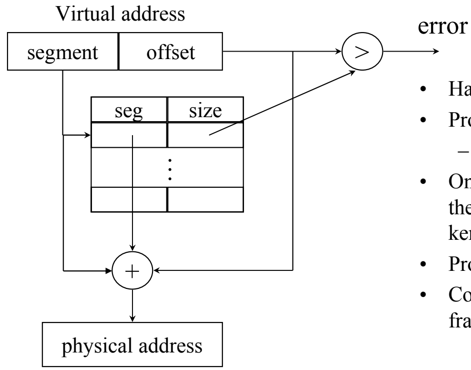


Base and Bound



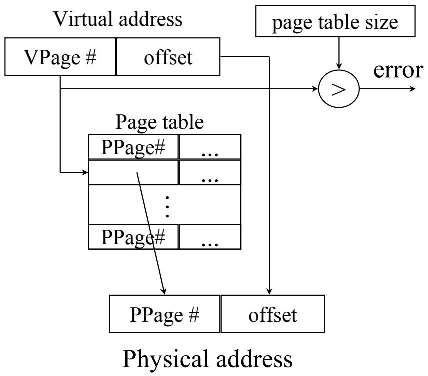
- Built in Cray-1
- A program can only access physical memory in [base, base+bound]
- On a context switch: save/restore base, bound registers
- Pros: Simple
- Cons: fragmentation, hard to share, and difficult to use disks

Segmentation



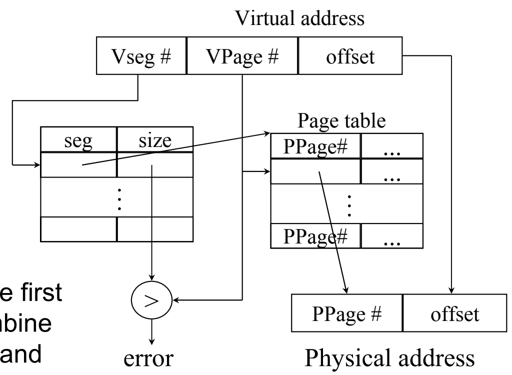
- Have a table of (seg, size)
- Protection: each entry has
  - (nil,read,write)
- On a context switch: save/restore the table or a pointer to the table in kernel memory
- Pros: Efficient, easy to share
- Cons: Complex management and fragmentation within a segment

Paging



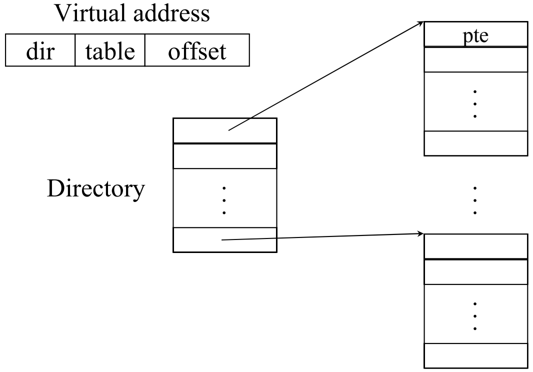
- Use a page table to translate
- Various bits in each entry
- Context switch: similar to the segmentation scheme
- What should be the page size?
- Pros: simple allocation, easy to share
- Cons: big page table and cannot deal with holes easily

Segmentation with Paging



Multics was the first system to combine segmentation and paging.
www.multicians.org

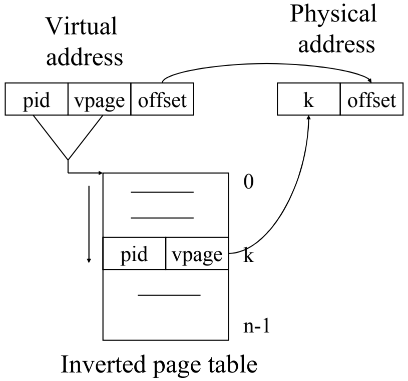
Multiple-Level Page Tables



How Many PTEs Do We Need?

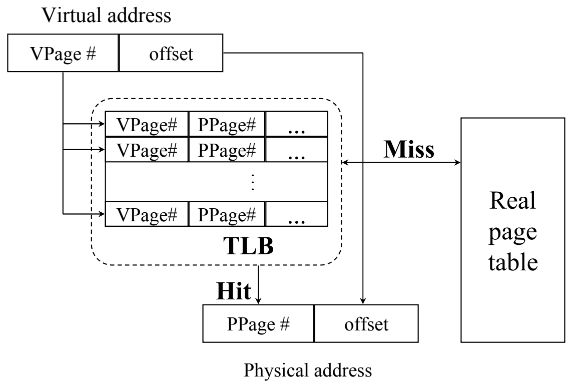
- Worst case for 32-bit address machine
  - # of processes × 2^{20} (if page size is 4096 bytes)
- What about 64-bit address machine?
  - # of processes × 2^{52}

Inverted Page Tables



- Main idea
  - One PTE for each physical page frame
  - Hash (Vpage, pid) to Ppage#
- Pros
  - Small page table for large address space
- Cons
  - Lookup is difficult
  - Overhead of managing hash chains, etc

Translation Look-aside Buffer (TLB)



Bits in A TLB Entry

- Common (necessary) bits
  - Virtual page number: match with the virtual address
  - Physical page number: translated address
  - Valid
  - Access bits: kernel and user (nil, read, write)
- Optional (useful) bits
  - Process tag
  - Reference
  - Modify
  - Cacheable

Hardware-Controlled TLB

- On a TLB miss
  - Hardware loads the PTE into the TLB
    - Need to write back if there is no free entry
  - Generate a fault if the page containing the PTE is invalid
  - VM software performs fault handling
  - Restart the CPU
- On a TLB hit, hardware checks the valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Software-Controlled TLB

- On a miss in TLB
  - Write back if there is no free entry
  - Check if the page containing the PTE is in memory
  - If no, perform page fault handling
  - Load the PTE into the TLB
  - Restart the faulting instruction
- On a hit in TLB, the hardware checks valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Hardware vs. Software Controlled

- Hardware approach
  - Efficient
  - Inflexible
  - Need more space for page table
- Software approach
  - Flexible
  - Software can do mappings by hashing
    - PP# → (Pid, VP#)
    - (Pid, VP#) → PP#
  - Can deal with large virtual address space

Cache vs. TLB

- Similarity
  - Both cache a portion of memory
  - Both write back on a miss
- Differences
  - TLB is usually fully set-associative
  - Cache can be direct-mapped
  - TLB does not deal with consistency with memory
  - TLB can be controlled by software
- Combine L1 cache with TLB
  - Virtually addressed cache
  - Why wouldn't everyone use virtually addressed cache?

Issues

- What TLB entry to be replaced?
  - Random
  - Pseudo LRU
- What happens on a context switch?
  - Process tag: change TLB registers and process register
  - No process tag: Invalidate the entire TLB contents
- What happens when changing a page table entry?
  - Change the entry in memory
  - Invalidate the TLB entry

Consistency Issue

- Snoopy cache protocols can maintain consistency with DRAM, even when DMA happens
- No hardware maintains consistency between DRAM and TLBs: you need to flush related TLBs whenever changing a page table entry in memory
- On multiprocessors, when you modify a page table entry, you need to do "TLB shootdown" to flush all related TLB entries on all processors