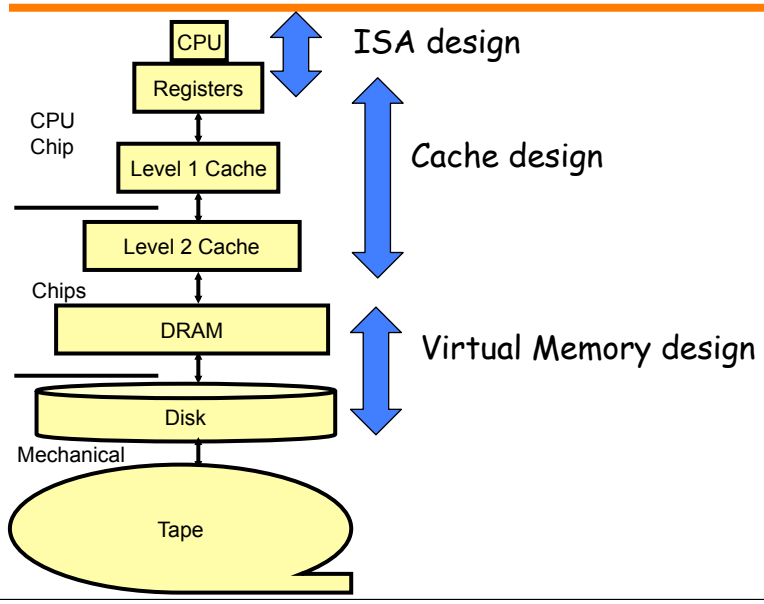
Lecture 17: Virtual Memory

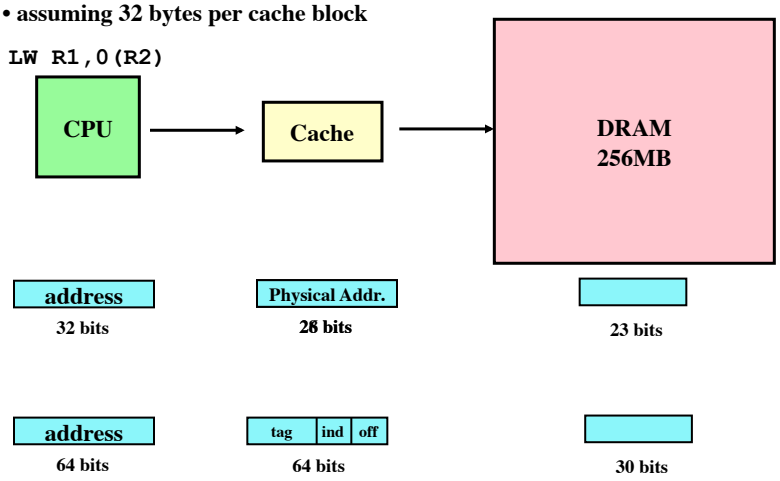
- Administration
  - Take QUIZ 11 over P&H 5.4 before 11:59pm today
  - Homework 6 due now
  - Homework 7 due Thursday April 1, 2010 no joke ☺
  - Exam April 6 (review April 1)
- Last Time
  - Using a memory address to find location in cache
  - Deciding what to evict from the cache
  - Improving cache optimization
- Today
  - What is Virtual Memory?

The Memory Hierarchy



Physical Memory Addressing

• assuming 32 bytes per cache block



Webster's definition of "virtual"

Pronunciation: 'v&r-ch&-w&l, -ch&l; 'v&rch-w&l
Function: *adjective*
Etymology: Middle English, possessed of certain physical virtues, from Medieval Latin *virtualis,* from Latin *virtus* strength, virtue

1 : being such in essence or effect though not formally recognized or admitted <a *virtual* dictator>

2 : of, relating to, or using virtual memory

3 : of, relating to, or being a hypothetical particle whose existence is inferred from indirect evidence <*virtual* photons>

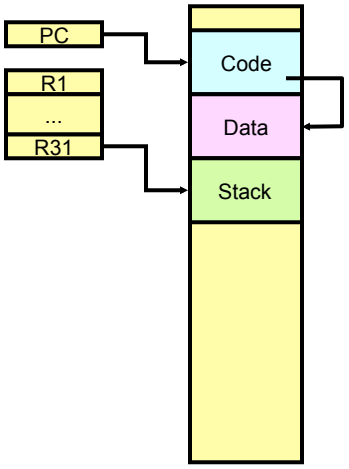
The goal of virtual memory

- Make it appear as if each process has:
  - Its own private memory
  - The memory is nearly infinite in size
- The challenge… Physical memory is:
  - Limited in size
  - Shared by all of the processes running on the machine
- The job of the virtual memory system is to maintain the illusion we want, given the physical limitations.

What if?

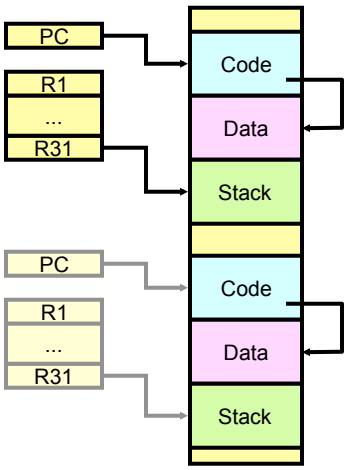
- A program is loaded into different places in memory each time it runs?
  - Relocation
- A program wants to use more memory than physically exists?
  - Page to disk
- We want to switch between multiple programs that use different data?
  - Protection

Simple View of Memory



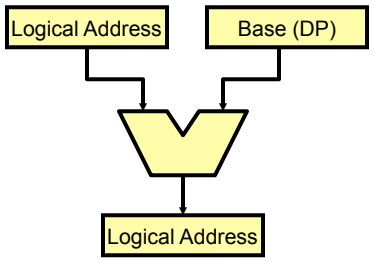
- Single *program* runs at a time
- Code and static data are at fixed locations
  - code starts at fixed location, e.g., 0x100
  - subroutines may be at fixed locations (absolute jumps)
- data locations may be *wired* into code
- Stack accesses relative to stack pointer.

Running Two Programs (Relocation)



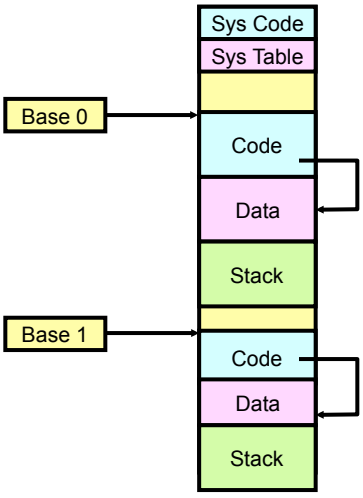
- Need to relocate *logical* addresses to *physical* locations
- Stack is already relocatable
  - all accesses relative to SP
- Code can be made relocatable
  - allow only relative jumps
  - all accesses relative to PC
- Data segment
  - at load time, calculate all addresses relative to a DP
    - expensive
  - faster with hardware support
    - base register

Base-Register Addressing



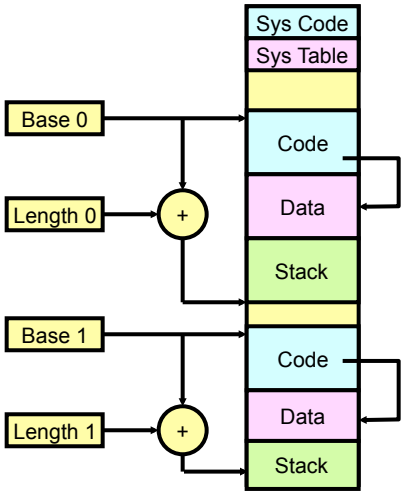
- Add a single base register, BR, to hardware
- Base register loaded with data pointer (DP) for current program
- All data addresses added to base before accessing memory
  - Can relocate code too
- Often implemented with an adder

Base Register Addressing – Context switching



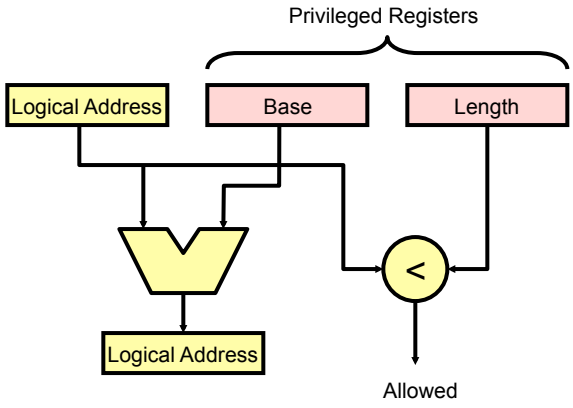
- System code handles switching between programs
  - Place to stand: Mechanism to bypass the base address
- System table contains
  - Base address of each program
  - Saved state of non-running programs

Providing Protection Between Programs (Length Registers)



- Add a *Length Register* LR to the hardware
- A program is only allowed to access memory from BR to B R+Length-1
- A program cannot set BR or LR
  - they are *privileged registers*
- But how do we switch programs?

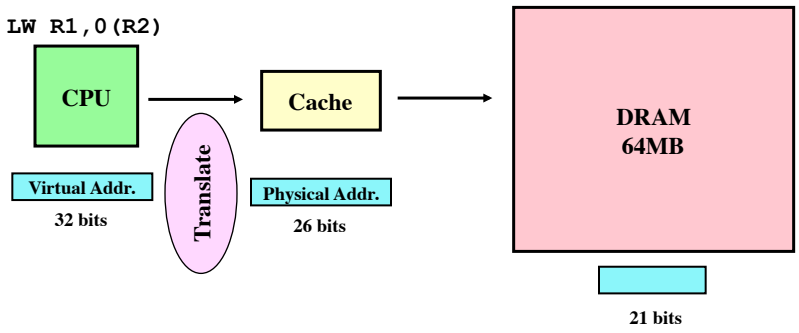
Base + Length Addressing



What a mess!

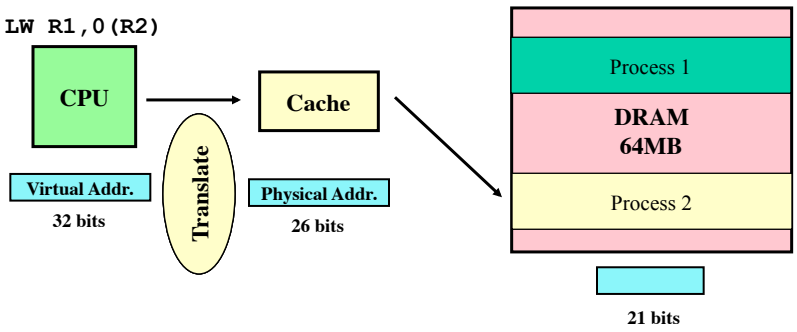
- Is there a better way that:
  - Provides a standard virtual address space to each process
  - Simplifies protection
  - Enables relocation
  - Extends the physical memory capacity

A Load from Virtual Memory



- Translate from virtual space to physical space
  - VA ⇒ PA
  - May need to go to disk

A Load from Virtual Memory



- Both programs can use the same set of addresses!
  - Change translation tables to point same VA to different PA for different programs

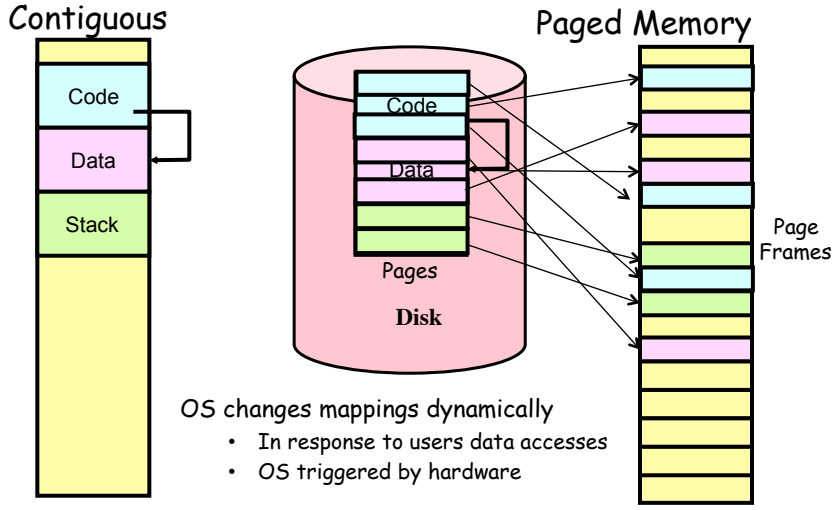
What a mess!

- Is there a better way? We need:
  - Provides a standard virtual address space to each process
  - Simplifies protection
  - Enables relocation
  - Extends the physical memory capacity

Demand Paging

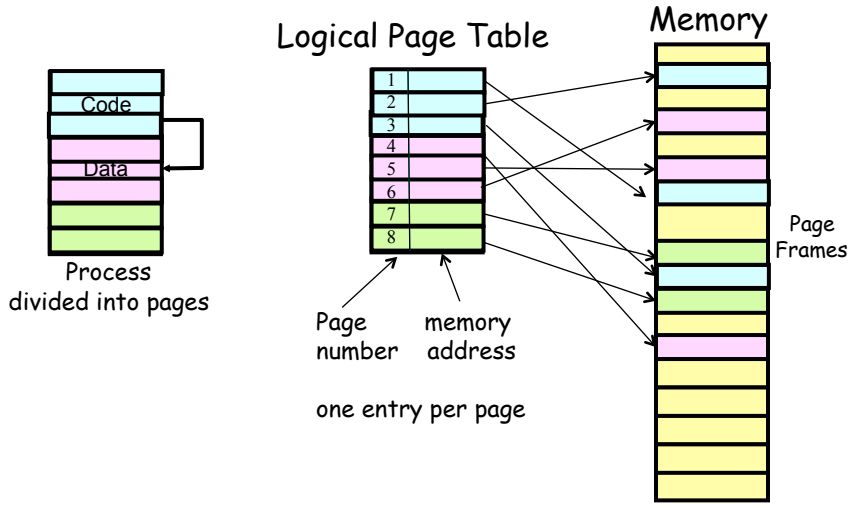
- Virtual address space is split into "pages"
  - 4-256 KB
  - Pages in main-memory are called "page-frames"
- Main memory holds a subset of all pages of a process
- How to locate a page?
  - Virtual to physical address translation
- How to allocate frames to processes?
  - Main memory as a cache for the disk

Contiguous logical view still, but Paged Memory: Any Page in any Page Frame!

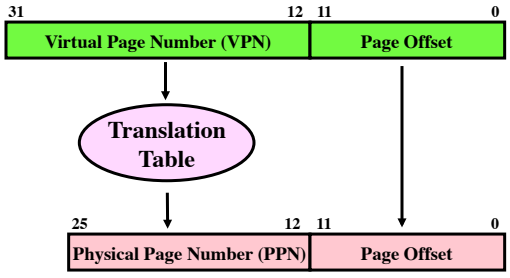


OS changes mappings dynamically
- In response to users data accesses
- OS triggered by hardware

What is the mapping from a page to a page frame?



Virtual Address Translation

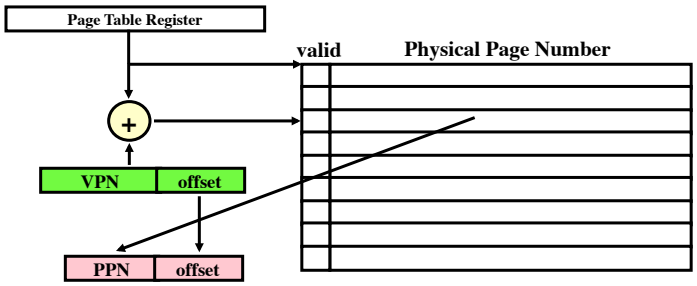


Example

- Main Memory = 64MB
- Page Size = 4KB
- VPN = 20 bits
- PPN = 14 bits

- Translation table
  - aka "Page Table"

Page Table Construction

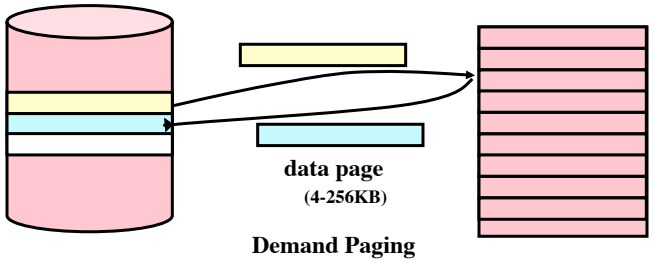


- Page table size
  - (14 + 1) * 2^{20} ~ 2MB
- Where to put the page table?

Paging and Protection

- How to ensure that processes can't access each other's data
  - Put them in separate virtual address spaces
  - Control the mappings of VA to PA for each process
    - Separate page tables
- How can you share data between processes?
  - Yes! Give them each a VA mapping to the same PA
    - Matching entry in each process' page table

Paging: Main Memory as a Cache for Disk



Demand Paging

- 32 bit addresses = 4GB, Main Memory = 64MB
- Dynamically adjust what data stays in main memory
  - Page similar to cache block
- Note: file system >> 4GB, managed by O/S

What if Data is Not in DRAM?

1) Examine page table

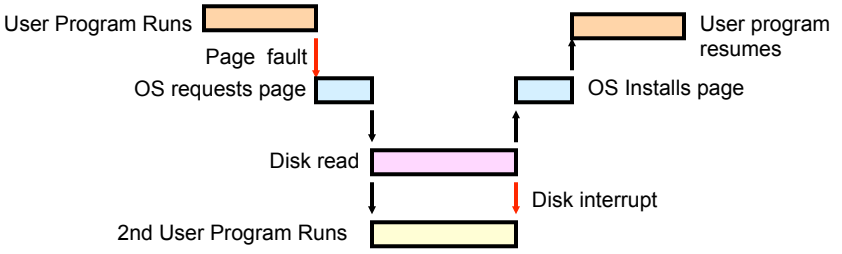
2) Discover that no mapping exists

3) Select page to evict, store back to disk

4) Bring in new page from disk

5) Update page table

Page Fault



Summary

- Virtual memory provides
  - Illusion of private memory system for each process
  - Protection
  - Relocation in memory system
  - Demand paging
- But – page tables can be large
  - Motivates: paging page tables, multi-level tables, inverted page tables, TLB
- Next time
  - Integration of virtual memory into cache hierarchy
  - DRAM memory organization, TLBs
  - Homework 7 is due 4/1
  - Reading: P&H 5.6, 5.11-13