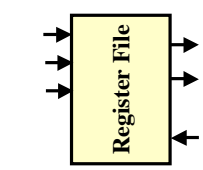
Lecture 16: Virtual Memory

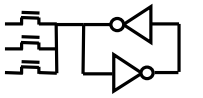
- Administrative
  - HW #5 due
  - HW #6 handed out (due in one week)
  - Start finding partners for final project
  - Exam on April 12 -- schedule on web site
  - Remember -- old lectures are also online
- Last Time:
  - Caches
- Today
  - Virtual memory

Memory Systems

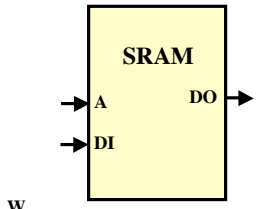
- Memory Technology
  - SRAM, DRAM
- Higher order memory functions
  - Relocation, protection
- Virtual memory

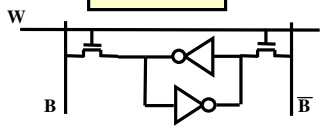
Typical Memory Technologies



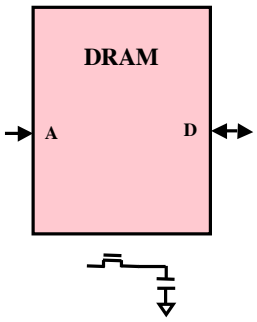


- Integrated into CPU
- Fast, many ports





- Caches
  - On chip L1
  - On/off chip L2
  - Off chip L3
- More bits, fewer ports

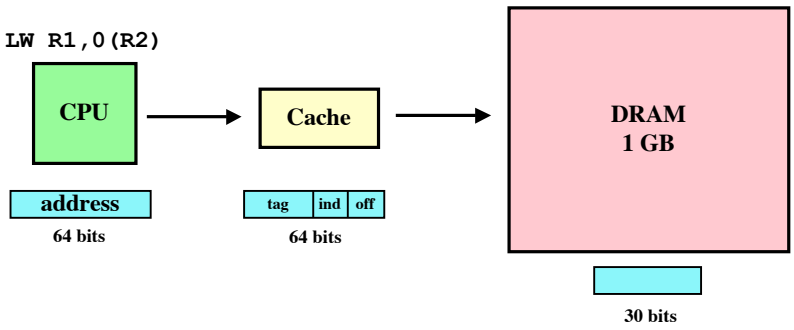


- Main Memory
- Very dense
- Slower to access, one port

SRAM vs. DRAM

- SRAM
  - Smaller capacity
  - Low latency for access
  - Storage cells are self-restoring
  - 4 times cost per bit (vs DRAM)
- DRAM
  - Larger capacity
  - High latency for access
  - Reads destroy data
    - Must write data back
    - Refresh periodically
  - Lower cost per bit

Physical Memory Addressing



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

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>

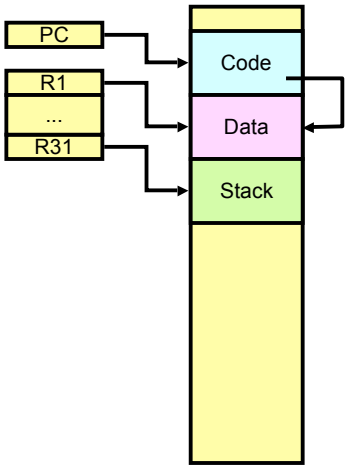
2 : of, relating to, or using virtual memory

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

The goal of virtual memory

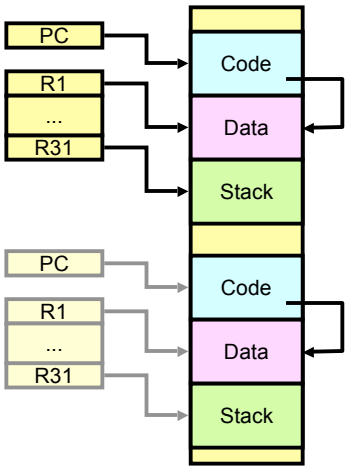
- Make it appear as if each process has:
  - Its own private memory
  - The memory is nearly infinite in size
- The challenge... Physical memory is:
  - Limited in size
  - Shared by all of the processes running on the machine
- The job of the virtual memory system is to maintain the illusion we want,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

Simple View of Memory



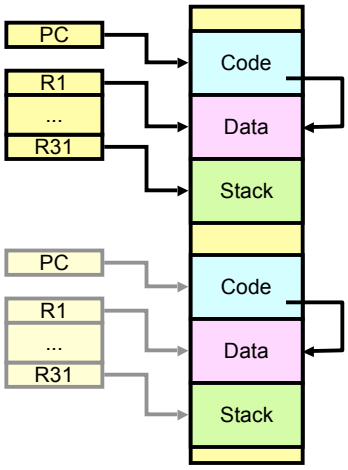
- Single *program* runs at a time
- Code and static data are at fixed locations
  - code starts at fixed location, e.g., 0x100
  - subroutines may be at fixed locations (absolute jumps)
- data locations may be *wired* into code
- Stack accesses relative to stack pointer.

Running Two Programs (Relocation) No Protection

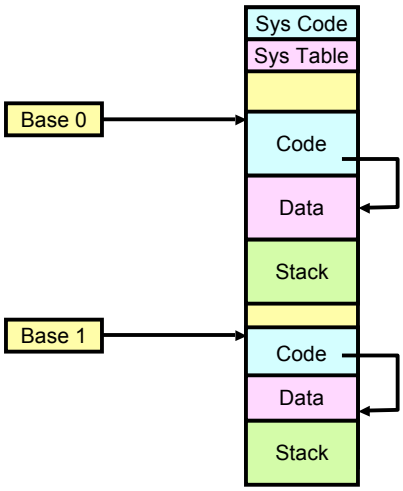


- Need to relocate *logical* addresses to *physical* locations
- Stack is already relocatable
  - all accesses relative to SP
- Code can be made relocatable
  - allow only relative jumps
  - all accesses relative to PC
- Data segment
  - can calculate all addresses relative to a DP
    - expensive
 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

Can you think of a simpler strategy?



Base Register Addressing

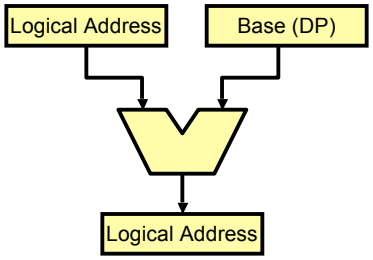


System code handles switching between programs

System table contains:

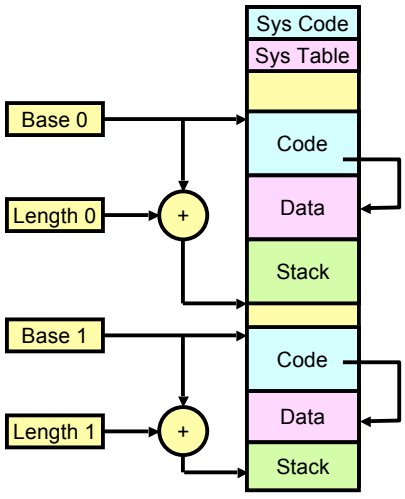
- Base address of each program
- Saved state of non-running programs

Implement base register with extra adder



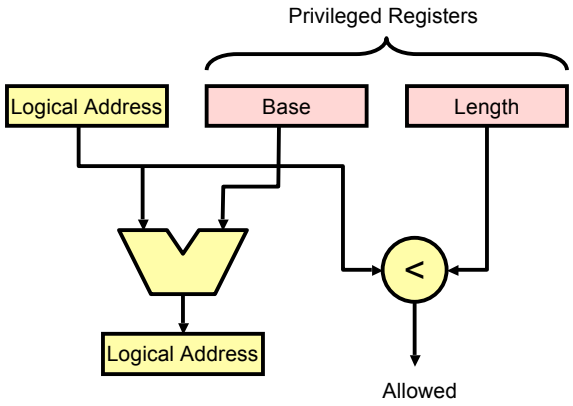
- Add a single base register, BR, to hardware
- Base register loaded with data pointer (DP) for current program
- All data addresses added to base before accessing memory
  - Can relocate code too
- Often implemented with a three-input adder
- Need to bypass base register to access system tables for program switching
  - a place to stand

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 BR+Length-1
- A program cannot set BR or LR
  - they are *privileged registers*
- But how do we switch programs?

Base + Length Addressing

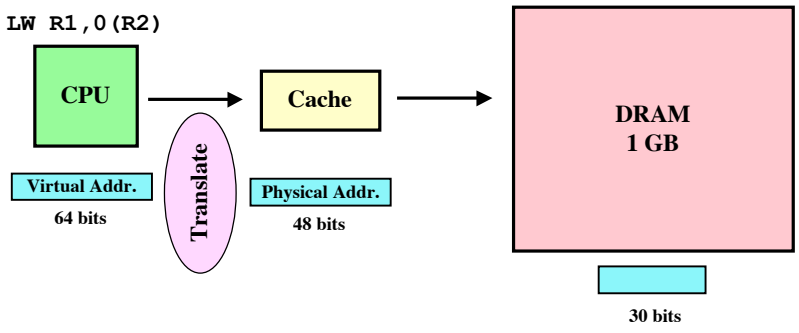


What a mess!

- Is there a better way that:
  - Simplifies protection
  - Enables relocation
  - Extends the physical memory capacity

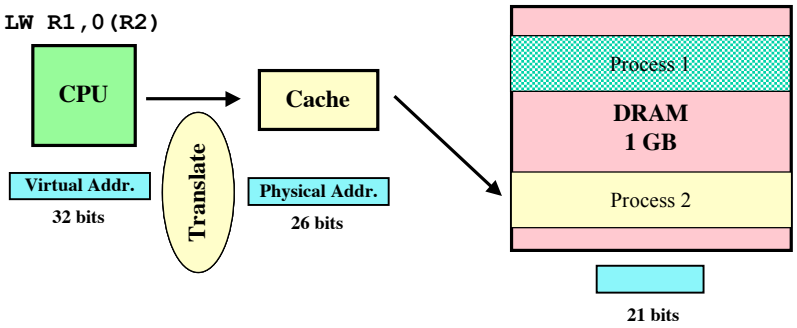
Analogy - Rental storage units

A Load to Virtual Memory



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

A Load to Virtual Memory

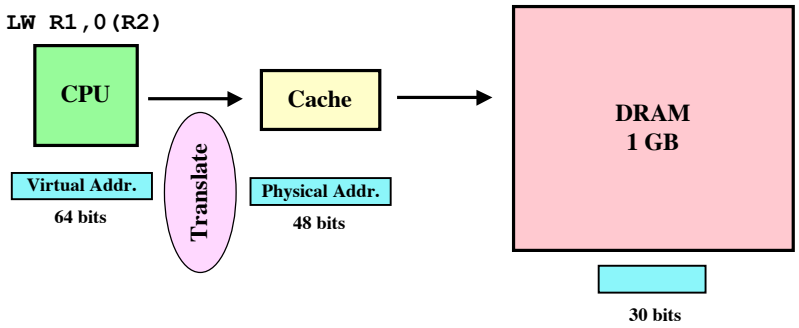


- Both programs can use the same set of addresses!
  - Change translation tables to point same VA to different PA for different programs

Paging and Protection

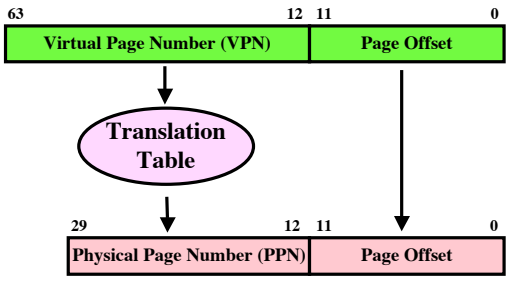
- How to ensure that processes can't access each other's data
  - Put them in separate virtual address spaces
  - Control the mappings of VA to PA for each process
    - Separate page tables
- How can you share data between processes
  - Give them each a VA mapping to the same PA
    - Similar entry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

How do we implement "translate" bubble?



- List some possibilities…

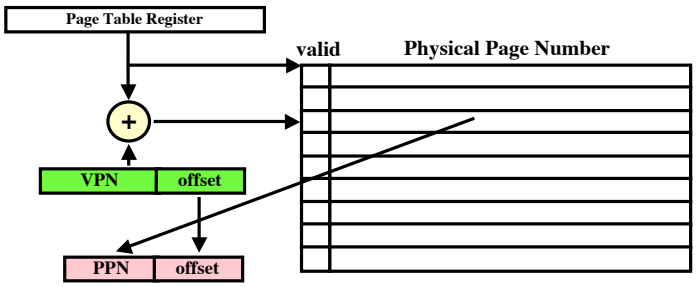
Virtual Address Translation



- Main Memory = 1 GB
- Page Size = 4KB
- VPN = 52 bits
- PPN = 18 bits

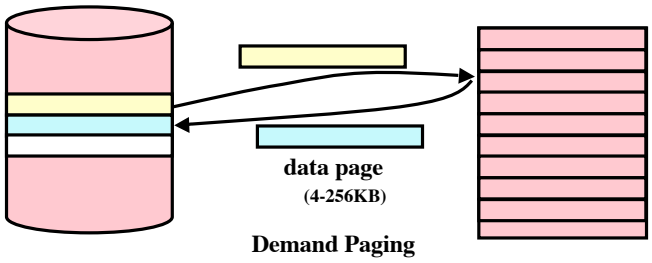
- Translation table
  - aka "Page Table"

Page Table Construction



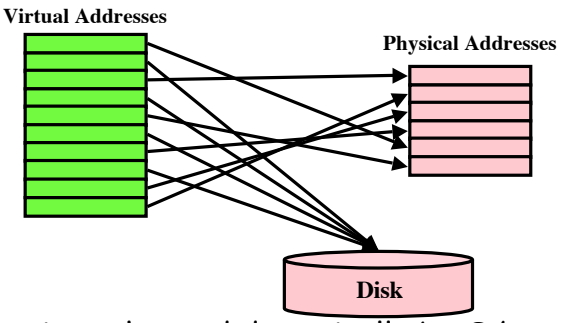
- Page table size
  - (14 + 1) * 2^{52} = enormous
- Where to put the page table?

Paging: Main Memory as a Cache for Disk



- 64 bit addresses = 2^32 * 4GB, Main Memory = 1 GB
- Dynamically adjust what data stays in main memory
  - Page similar to cache block
- Note: file system >> 32 GB, managed by O/S

Virtual Addresses Span Memory+Disk



- Mappings changed dynamically by O/S
  - In response to users data accesses
  - OS triggered by hardware

What if Data is Not in DRAM?

1) Examine page table

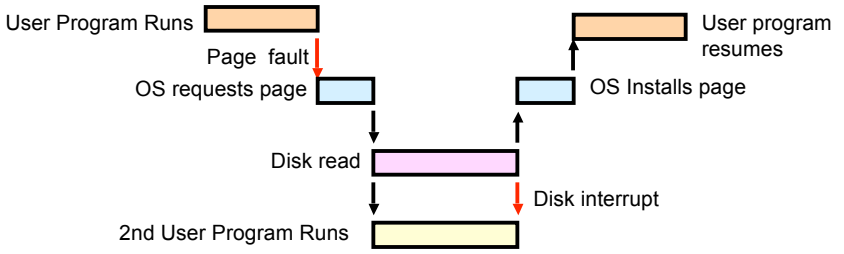
2) Discover that no mapping exists

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
- Next time
  - Integration of virtual memory into cache hierarchy
  - DRAM memory organization