inst.eecs.berkeley.edu/~cs61c

UCB CS61C : Machine Structures

Lecture 34 – Virtual Memory II
2010-04-19

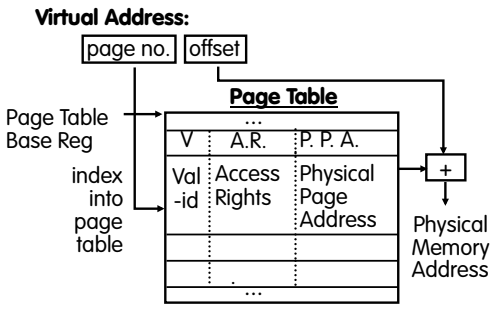
Lecturer SOE
Dan Garcia

$31,600 FOR AN ATARI 2600 VIDEO GAME?

Tanner Sandlin found a rare video game in his house, put it on eBay and just sold it for almost $32K! Some articles have called this the "holy grail" of Atari 2600 games…

www.neoseeker.com/news/13615-very-rare-atari-2600-air-raid-game-sells-for-31-600-usd/

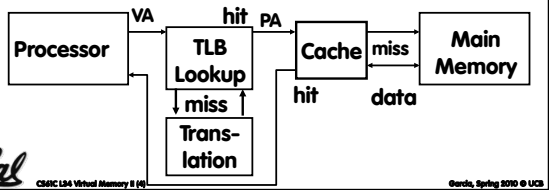
Review Address Mapping: Page Table



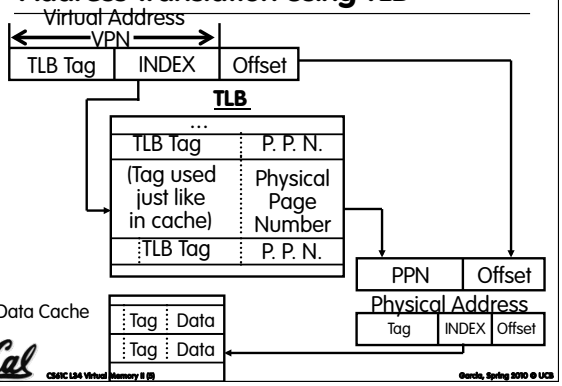
Page Table located in physical memory

Fetching data on a memory read

- **Check TLB (input: VPN, output: PPN)**
  - hit: fetch translation
  - miss: check page table (in memory)
    - Page table hit: fetch translation
    - Page table miss: page fault, fetch page from disk to memory, return translation to TLB
- **Check cache (input: PPN, output: data)**
  - hit: return value
  - miss: fetch value from memory, remember it in cache, return value



Address Translation using TLB



Typical TLB Format

Tag	Physical Page #	Dirty	Ref	Valid	Access Rights

- **TLB just a cache on the page table mappings**
- **TLB access time comparable to cache (much less than main memory access time)**
- **Dirty: since use write back, need to know whether or not to write page to disk when replaced**
- **Ref: Used to help calculate LRU on replacement**
  - Cleared by OS periodically, then checked to see if page was referenced

What if not in TLB?

- **Option 1: Hardware checks page table and loads new Page Table Entry into TLB**
- **Option 2: Hardware traps to OS, up to OS to decide what to do**
  - MIPS follows Option 2: Hardware knows nothing about page table
  - A trap is a synchronous exception in a user process, often resulting in the OS taking over and performing some action before returning to the program.
    - More about exceptions next lecture

What if the data is on disk?

- We load the page off the disk into a free block of memory, using a DMA transfer (Direct Memory Access – special hardware support to avoid processor)
  - Meantime we switch to some other process waiting to be run
- When the DMA is complete, we get an interrupt and update the process's page table
  - So when we switch back to the task, the desired data will be in memory

What if we don't have enough memory?

- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
- And update that program's page table to reflect the fact that its memory moved somewhere else
- If continuously swap between disk and memory, called Thrashing

Question (1/3)

- 40-bit virtual address, 16 KB page

Virtual Page Number (? bits)	Page Offset (? bits)

- 36-bit physical address

Physical Page Number (? bits)	Page Offset (? bits)

- Number of bits in Virtual Page Number/Page offset, Physical Page Number/Page offset?
  1: 22/18 (VPN/PO), 22/14 (PPN/PO)
  2: 24/16, 20/16
  3: 26/14, 22/14
  4: 26/14, 26/10
  5: 28/12, 24/12

Question (2/3): 40b VA, 36b PA

- 2-way set-assoc. TLB, 512 entries, 40b VA:

TLB Tag (? bits)	TLB Index (? bits)	Page Offset (14 bits)

- TLB Entry: Valid bit, Dirty bit, Access Control (say 2 bits), Virtual Page Number, Physical Page Number

V	D	Access (2 bits)	TLB Tag (? bits)	Physical Page No. (? bits)

- Number of bits in TLB Tag / Index / Entry?
  1: 12 / 14 / 38 (TLB Tag / Index / Entry)
  2: 14 / 12 / 40
  3: 18 / 8 / 44
  4: 18 / 8 / 58

Question (3/3)

- 2-way set-assoc, 64KB data cache, 64B block

Cache Tag (? bits)	Cache Index (? bits)	Block Offset (? bits)

Physical Page Address (36 bits)

- Data Cache Entry: Valid bit, Dirty bit, Cache tag + ? bits of Data

V	D	Cache Tag (? bits)

Cache Data (? bits)

- Number of bits in Data cache Tag / Index / Offset / Entry?
  1: 12 / 9 / 14 / 87 (Tag/Index/Offset/Entry)
  2: 20 / 10 / 6 / 86
  3: 20 / 10 / 6 / 534
  4: 21 / 9 / 6 / 87
  5: 21 / 9 / 6 / 535

And in Conclusion…

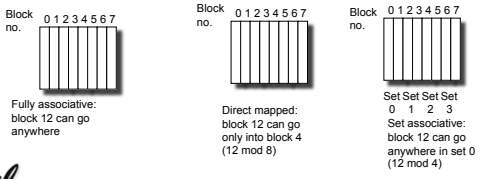
- Virtual memory to Physical Memory Translation too slow?
  - Add a cache of Virtual to Physical Address Translations, called a TLB
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well
- Virtual Memory allows protected sharing of memory between processes with less swapping to disk

4 Qs for any Memory Hierarchy

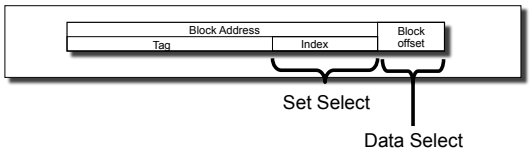
- **Q1: Where can a block be placed?**
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)
- **Q2: How is a block found?**
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - Separate lookup table (as in a page table)
- **Q3: Which block is replaced on a miss?**
  - Least recently used (LRU)
  - Random
- **Q4: How are writes handled?**
  - Write through (Level never inconsistent w/lower)
  - Write back (Could be "dirty", must have dirty bit)

Q1: Where block placed in upper level?

- **Block #12 placed in 8 block cache:**
  - Fully associative
  - Direct mapped
  - 2-way set associative
    - Set Associative Mapping = Block # Mod # of Sets



Q2: How is a block found in upper level?



- **Direct indexing (using index and block offset), tag compares, or combination**
- **Increasing associativity shrinks index, expands tag**

Q3: Which block replaced on a miss?

- **Easy for Direct Mapped**
- **Set Associative or Fully Associative:**
  - Random
  - LRU (Least Recently Used)

Miss Rates

Associativity:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What to do on a write hit?

- **Write-through**
  - update the word in cache block and corresponding word in memory
- **Write-back**
  - update word in cache block
  - allow memory word to be "stale"
  - => add 'dirty' bit to each line indicating that memory be updated when block is replaced
  - => OS flushes cache before I/O !!!
- **Performance trade-offs?**
  - WT: read misses cannot result in writes
  - WB: no writes of repeated writes

Three Advantages of Virtual Memory

- **1) Translation:**
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multiple processes reasonable
  - 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

Three Advantages of Virtual Memory

- **2) Protection:**
  - Different 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 ⇒ Far more "viruses" under Microsoft Windows
  - Special Mode in processor ("Kernel mode") allows processor to change page table/TLB
- **3) Sharing:**
  - Can map same physical page to multiple users ("Shared memory")

Why Translation Lookaside Buffer (TLB)?

- **Paging is most popular implementation of virtual memory (vs. base/bounds)**
- **Every paged virtual memory access must be checked against Entry of Page Table in memory to provide protection / indirection**
- **Cache of Page Table Entries (TLB) makes address translation possible without memory access in common case to make fast**

Bonus slide: Virtual Memory Overview (1/3)

- **User program view of memory:**
  - Contiguous
  - Start from some set address
  - Infinitely large
  - Is the only running program
- **Reality:**
  - Non-contiguous
  - Start wherever available memory is
  - Finite size
  - Many programs running at a time

Bonus slide: Virtual Memory Overview (2/3)

- **Virtual memory provides:**
  - illusion of contiguous memory
  - all programs starting at same set address
  - illusion of ~ infinite memory (232 or 264 bytes)
  - protection

Bonus slide: Virtual Memory Overview (3/3)

- **Implementation:**
  - Divide memory into "chunks" (pages)
  - Operating system controls page table that maps virtual addresses into physical addresses
  - Think of memory as a cache for disk
  - TLB is a cache for the page table

Address Map, Mathematically

V = {0, 1, . . . , n - 1} virtual address space (n > m)
M = {0, 1, . . . , m - 1} physical address space
MAP: V --> M U {θ} address mapping function

MAP(a) = a' if data at virtual address a
is present in physical address a' and a' in M
= θ if data at virtual address a is not present in M