inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #26 Virtual Memory II & I/O Intro

2007-8-8

Scott Beamer, Instructor

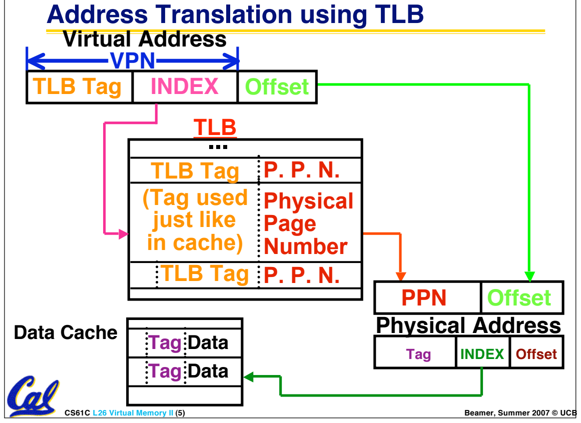
Apple Releases new iMac

www.apple.com

Review

- Manage memory to disk? Treat as cache
  - Included protection as bonus, now critical
  - Use Page Table of mappings for each process vs. tag/data in cache
- Virtual Memory allows protected sharing of memory between processes
- Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well

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

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?

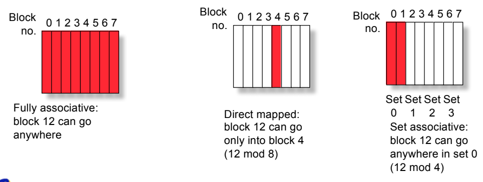
- We chose some other page belonging to a program and transfer it onto the disk if it is dirty
  - If clean (disk copy is up-to-date), just overwrite that data in memory
  - We chose the page to evict based on replacement policy (e.g., LRU)
- And update that program's page table to reflect the fact that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

4 Qs for any Memory Hierarchy

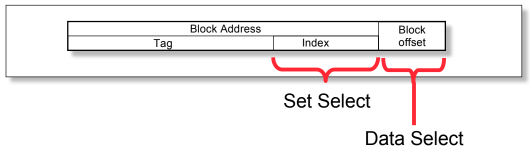
- Q1: Where can a block be placed?
  - One place (direct mapped)
  - A few places (set associative)
  - Any place (fully associative)
- Q2: How is a block found?
  - Indexing (as in a direct-mapped cache)
  - Limited search (as in a set-associative cache)
  - Full search (as in a fully associative cache)
  - 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


Block no.
0 1 2 3 4 5 6 7
Fully associative: block 12 can go anywhere
Block no.
0 1 2 3 4 5 6 7
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Block no.
0 1 2 3 4 5 6 7
Set Set Set Set
0 1 2 3
Set associative: block 12 can go anywhere in set 0 (12 mod 4)

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?

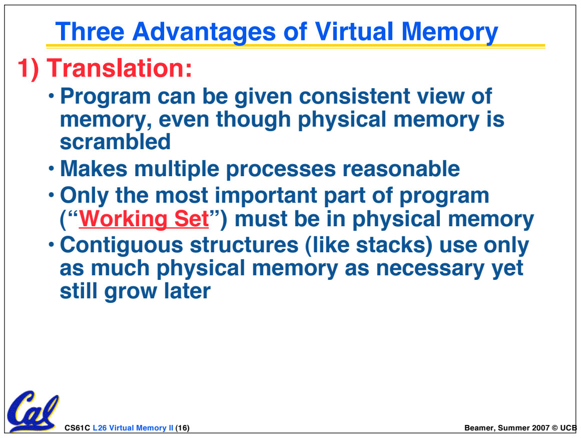
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

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?

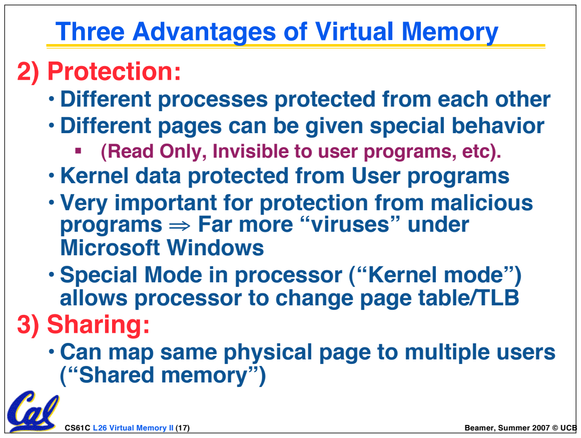
- Write-through
  - update the word in cache block and corresponding word in memory
- Write-back
  - update word in cache block
  - allow memory word to be "stale"
  - => add 'dirty' bit to each line indicating that memory be updated when block is replaced
  - => OS flushes cache before I/O !!!
- Performance 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



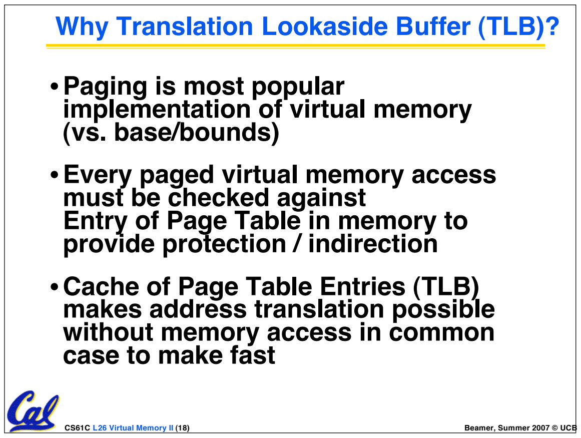
Three Advantages of Virtual Memory

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

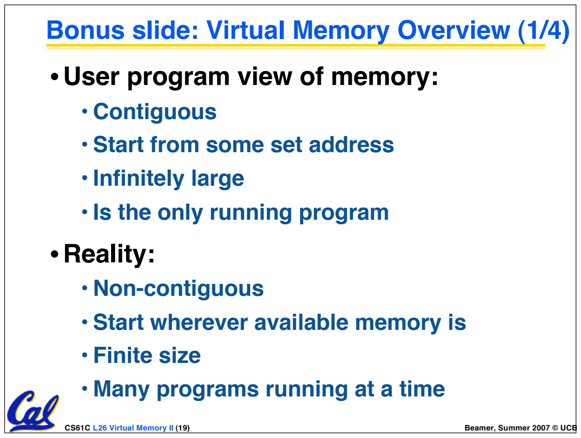
3) Sharing:

- Can map same physical page to multiple users ("Shared memory")



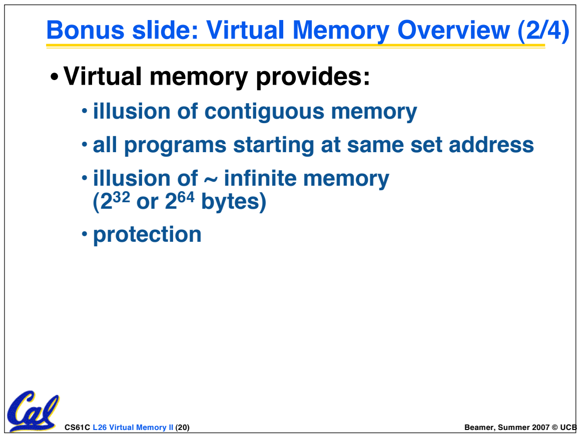
Why Translation Lookaside Buffer (TLB)?

- 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



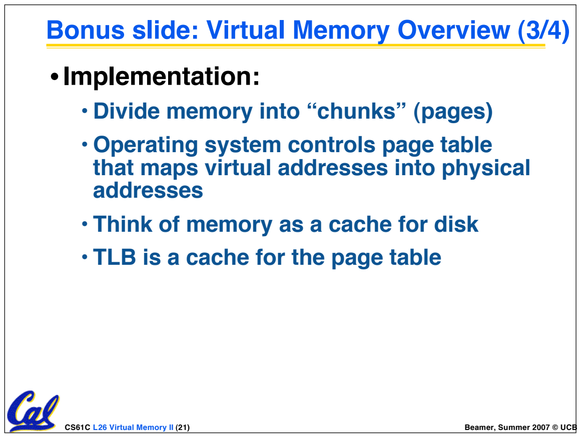
Bonus slide: Virtual Memory Overview (1/4)

- 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 (2/4)

- Virtual memory provides:
  - illusion of contiguous memory
  - all programs starting at same set address
  - illusion of ~ infinite memory (2^{32} or 2^{64} bytes)
  - protection



Bonus slide: Virtual Memory Overview (3/4)

- 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

Bonus slide: Virtual Memory Overview (4/4)

- Let's say we're fetching some data:
  - 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

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
```

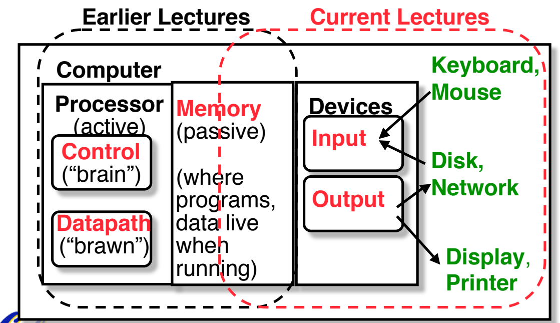
And in Conclusion…

- 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

Administrivia

- Assignments
  - Proj4 due 8/12
  - HW8 due 8/14
- Proj3 Face-to-Face grading started Monday
  - Make an appointment (and show up)
- Final Review Session in works
- Course Survey during last lecture
  - 2 points extra added for taking survey (still anonymous)
- Midterm Regrades done

Recall : 5 components of any Computer



Motivation for Input/Output

- I/O is how humans interact with computers
- I/O gives computers long-term memory.
- I/O lets computers do amazing things:
  - Read pressure of synthetic hand and control synthetic arm and hand of fireman
  - Control propellers, fins, communicate in BOB (Breathable Observable Bubble)
- Computer without I/O like a car without wheels; great technology, but won't get you anywhere

I/O Device Examples and Speeds

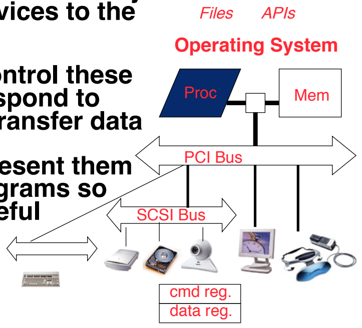
- I/O Speed: bytes transferred per second (from mouse to Gigabit LAN: 7 orders of mag!)

• Device	Behavior	Partner	Data Rate (KBytes/s)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Wireless Network	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00
Wired LAN Network	I or O	Machine	125,000.00
OC-768	I or O	Machine	5,000,000.00

When discussing transfer rates, use 10^x

What do we need to make I/O work?

- A way to connect many types of devices to the Proc-Mem
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful

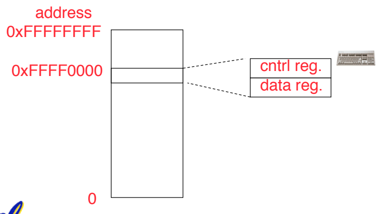


Instruction Set Architecture for I/O

- What must the processor do for I/O?
  - Input: reads a sequence of bytes
  - Output: writes a sequence of bytes
- Some processors have special input and output instructions
- Alternative model (used by MIPS):
  - Use loads for input, stores for output
  - Called "Memory Mapped Input/Output"
  - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)

Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices



Processor-I/O Speed Mismatch

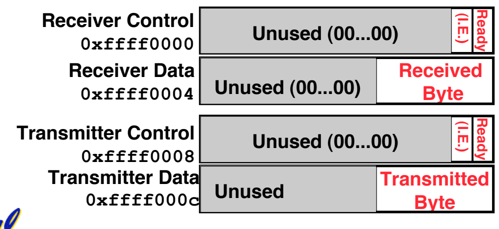
- 1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate
  - I/O devices data rates range from 0.01 KB/s to 125,000 KB/s
- Input: device may not be ready to send data as fast as the processor loads it
  - Also, might be waiting for human to act
- Output: device not be ready to accept data as fast as processor stores it
- What to do?

Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - Control Register, says it's OK to read/write (I/O ready) [think of a flagman on a road]
  - Data Register, contains data
- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg (0 ⇒ 1) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit (1 ⇒ 0) of Control Register

SPIM I/O Simulation

- SPIM simulates 1 I/O device: memory-mapped terminal (keyboard + display)
  - Read from keyboard (receiver); 2 device regs
  - Writes to terminal (transmitter); 2 device regs



SPIM I/O

- Control register rightmost bit (0): Ready
  - Receiver: Ready==1 means character in Data Register not yet been read; 1 ⇒ 0 when data is read from Data Reg
  - Transmitter: Ready==1 means transmitter is ready to accept a new character; 0 ⇒ Transmitter still busy writing last char
    - I.E. bit discussed later
- Data register rightmost byte has data
  - Receiver: last char from keyboard; rest = 0
  - Transmitter: when write rightmost byte, writes char to display

I/O Example

- Input: Read from keyboard into `$v0`

```
               lui  $t0, 0xffff #ffff0000
Waitloop:      lw   $t1, 0($t0) #control
               andi $t1,$t1,0x1
               beq  $t1,$zero, Waitloop
               lw   $v0, 4($t0) #data
```

- Output: Write to display from `$a0`

```
               lui  $t0, 0xffff #ffff0000
Waitloop:      lw   $t1, 8($t0) #control
               andi $t1,$t1,0x1
               beq  $t1,$zero, Waitloop
               sw   $a0, 12($t0) #data
```

- Processor waiting for I/O called "Polling"
- "Ready" bit is from processor's point of view!

I/O Conclusion

- Critical for computing, without it no way for humans to interact with them
  - Or interact with other computers
- Great diversity in bandwidth, latency, and type of I/O devices
- Need architecture (OS and hardware) to control them and make them easy to program for
- MIPS uses Memory Mapped I/O