Lecture 3 – Memory Hierarchy

Slides were used during lectures by
David Patterson, Berkeley, spring 2006

Review from last lecture

- **Quantify and summarize performance**
  - Ratios, Geometric Mean, Multiplicative Standard Deviation
- **F&P: Benchmarks age, disks fail,1 point fail danger**
- **Control VIA State Machines and Microprogramming**
- **Just overlap tasks; easy if tasks are independent**
- **Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:**

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Cycle Time}_{\text{unpipelined}}}{\text{Cycle Time}_{\text{pipelined}}}$$

- **Hazards limit performance on computers:**
  - Structural: need more HW resources
  - Data (RAW,WAR,WAW): need forwarding, compiler scheduling
  - Control: delayed branch, prediction
- **Exceptions, Interrupts add complexity**

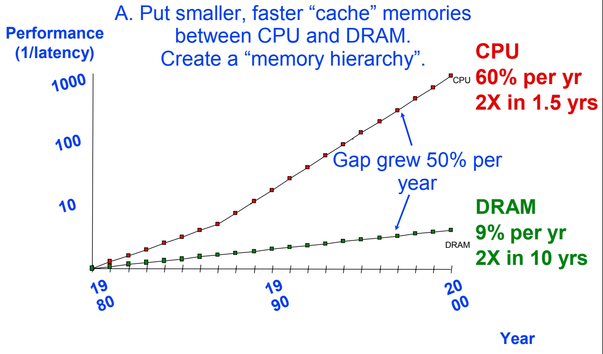
Outline

- Review
- Memory hierarchy
- Locality
- Cache design
- Virtual address spaces
- Page table layout
- TLB design options
- Conclusion

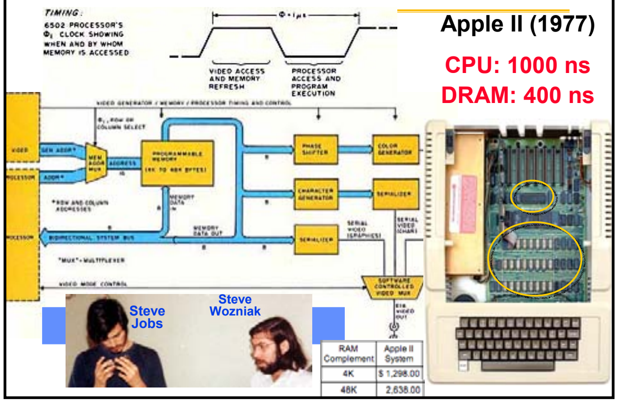
Since 1980, CPU has outpaced DRAM ...

Q. How do architects address this gap?

A. Put smaller, faster "cache" memories between CPU and DRAM. Create a "memory hierarchy".



1977: DRAM faster than microprocessors



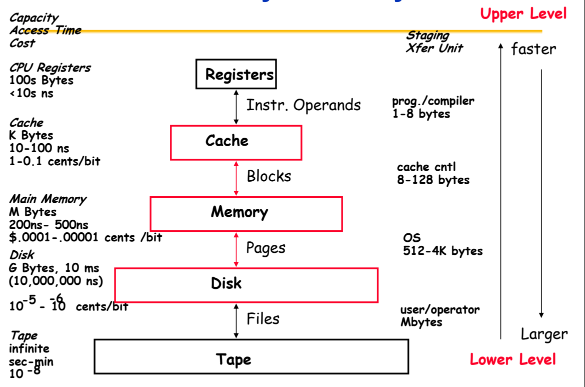
TIMING: 6502 PROCESSOR'S Φ₂ CLOCK SHOWING WHEN AND BY WHOM MEMORY IS ACCESSED
VIDEO ACCESS AND MEMORY REFRESH
PROCESSOR ACCESS AND PROGRAM EXECUTION
Steve Jobs
Steve Wozniak

RAM Complement	Apple II System
4K	$ 1,298.00
48K	2,638.00

Apple II (1977)

CPU: 1000 ns
DRAM: 400 ns

Levels of the Memory Hierarchy



Memory Hierarchy: Apple iMac G5

Managed by compiler

Managed by hardware

Managed by OS, hardware, application

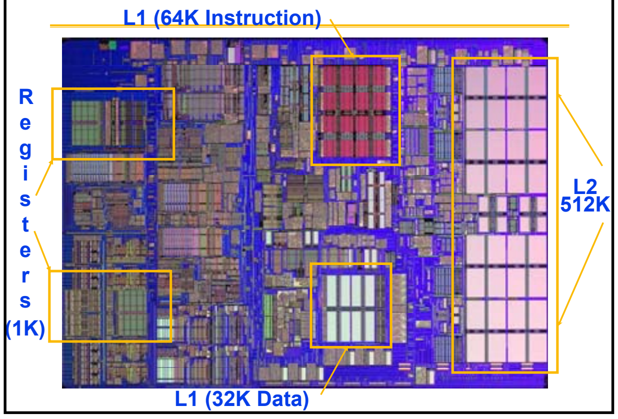
07	Reg	L1 Inst	L1 Data	L2	DRAM	Disk
Size	1K	64K	32K	512K	256M	80G
Latency Cycles, Time	1, 0.6 ns	3, 1.9 ns	3, 1.9 ns	11, 6.9 ns	88, 55 ns	10^7, 12 ms

iMac G5 1.6 GHz

Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

iMac's PowerPC 970: All caches on-chip

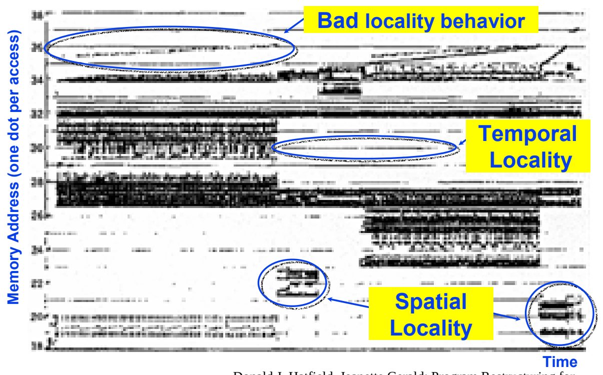


The Principle of Locality

- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.
- **Two Different Types of Locality:**
  - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)
- **Last 15 years, HW relied on locality for speed**

It is a property of programs which is exploited in machine design.

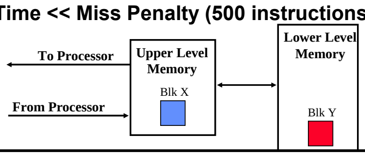
Programs with locality cache well ...



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

Memory Hierarchy: Terminology

- **Hit: data appears in some block in the upper level (example: Block X)**
  - **Hit Rate**: the fraction of memory access found in the upper level
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss: data needs to be retrieve from a block in the lower level (Block Y)**
  - **Miss Rate = 1 − (Hit Rate)**
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block the processor
- **Hit Time << Miss Penalty (500 instructions on 21264!)**



Cache Measures

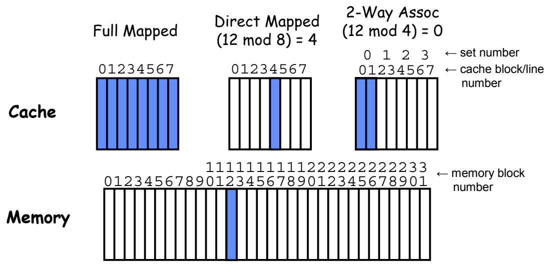
- ***Hit rate*: fraction found in that level**
  - So high that usually talk about *Miss rate*
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- **Average memory-access time**
  = Hit time + Miss rate x Miss penalty (ns or clocks)
- ***Miss penalty*: time to replace a block from lower level, including time to replace in CPU**
  - ***access time***:
    time to lower level = *f*(latency to lower level)
  - ***transfer time***:
    time to transfer block = *f*(bandwidth between upper & lower levels)

4 Questions for Memory Hierarchy

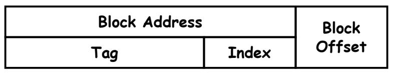
- **Q1: Where can a block be placed in the upper level?** ***(Block placement)***
- **Q2: How is a block found if it is in the upper level?** ***(Block identification)***
- **Q3: Which block should be replaced on a miss?** ***(Block replacement)***
- **Q4: What happens on a write?** ***(Write strategy)***

Q1: Where can a block be placed in the upper level?

- **Block 12 placed in 8 block cache:**
  - Fully associative, direct mapped, set associative
  - S.A. Mapping = Block Number modulo #Sets


Full Mapped
Direct Mapped (12 mod 8) = 4
2-Way Assoc (12 mod 4) = 0
0 1 2 3 ← set number
01234567 ← cache block/line number
Cache
1111111111222222222233
01234567890123456789012345678901 ← memory block number
Memory

Q2: How is a block found if it is in the upper level?



- **Tag on each block**
  - No need to check index or block offset
- **Increasing associativity shrinks index, expands tag**

Q3: Which block should be replaced on a miss?

- **Easy for Direct Mapped**
- **Set Associative or Fully Associative:**
  - **LRU (Least Recently Used)**; appealing, but hard to implement for high associativity
  - **Random**; easy to implement, how well does it work?

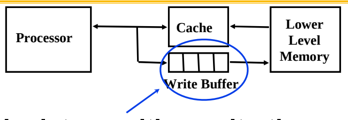
Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Q4: What happens on a write?

	Write-Through	Write-Back
Policy	Data written to cache block also written to lower-level memory	Write data only to the cache Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option: let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower level memory

Q. Why a write buffer ?	A. So CPU doesn't stall
Q. Why a buffer, why not just one register ?	A. Bursts of writes are common.
Q. Are Read After Write (RAW) hazards an issue for write buffer?	A. Yes! Drain buffer before next read, or send read first after check write buffers.

Cache misses

Cache misses can be divided into three categories

Compulsory	First access miss, cold start miss.
Capacity	Cache is full.
Conflict	Two blocks are mapped to the same location.

6 Basic Cache Optimizations

Reducing Miss Rate

1. Larger Block size (compulsory misses)
2. Larger Cache size (capacity misses)
3. Higher Associativity (conflict misses)

Reducing Miss Penalty

4. Multilevel Caches

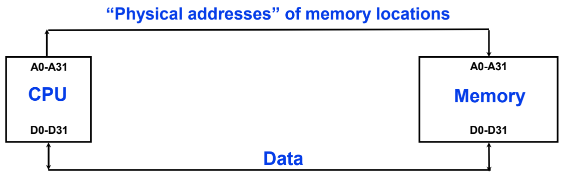
Reducing hit time

5. Giving Reads Priority over Writes
   E.g., Read complete before earlier writes in write buffer
6. Avoiding Address Translation during Indexing of the Cache

Outline

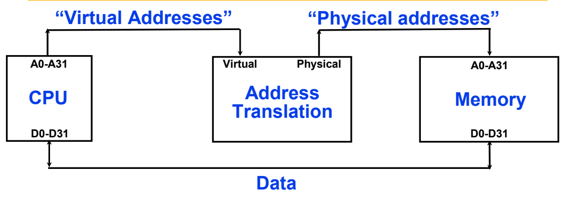
- Review
- Memory hierarchy
- Locality
- Cache design
- **Virtual address spaces**
- **Page table layout**
- **TLB design options**
- **Conclusion**

The Limits of Physical Addressing



- All programs share one address space: The physical address space
- Machine language programs must be aware of the machine organization
- No way to prevent a program from accessing any machine resource

Solution: Add a Layer of Indirection



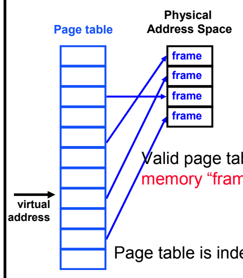
- User programs run in a standardized virtual address space
- Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory
- Hardware supports "modern" OS features: Protection, Translation, Sharing

Three Advantages of Virtual Memory

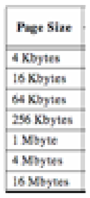
- **Translation:**
  - Program can be given consistent view of memory, even though physical memory is scrambled
  - Makes multithreading reasonable (now used a lot!)
  - Only the most important part of program ("Working Set") must be in physical memory.
  - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
- **Protection:**
  - Different threads (or 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
- **Sharing:**
  - Can map same physical page to multiple users ("Shared memory")

Page tables encode virtual address spaces

A virtual address space is divided into blocks of memory called pages



A machine usually supports pages of a few sizes (MIPS R4000):



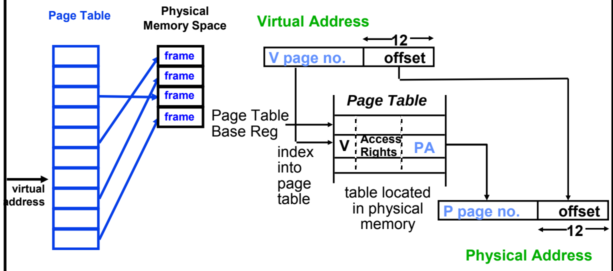
Page Size
4 Kbytes
16 Kbytes
64 Kbytes
256 Kbytes
1 Mbyte
4 Mbytes
16 Mbytes

Valid page table entry codes physical memory "frame" address for the page

Page table is indexed by a virtual address

OS manages the page table

Details of Page Table



- Page table maps virtual page numbers to physical frames ("PTE" = Page Table Entry)
- Virtual memory => treat memory ≈ cache for disk

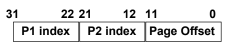
Page tables may not fit in memory!

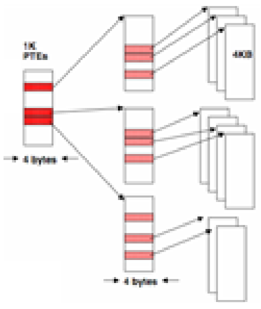
A table for 4KB pages for a 32-bit address space has 1M entries

Each process needs its own address space!

Two-level Page Tables

32 bit virtual address





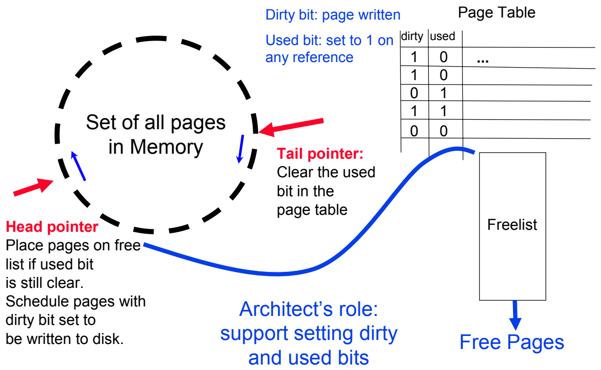
Top-level table wired in main memory

Subset of 1024 second-level tables in main memory; rest are on disk or unallocated

VM and Disk: Page replacement policy

Dirty bit: page written

Used bit: set to 1 on any reference



dirty	used	
1	0	...
1	0	
0	1	
1	1	
0	0	

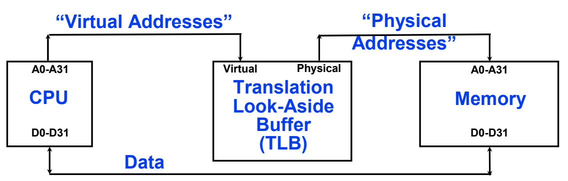
Tail pointer: Clear the used bit in the page table

Head pointer Place pages on free list if used bit is still clear. Schedule pages with dirty bit set to be written to disk.

Architect's role: support setting dirty and used bits

TLB Design Concepts

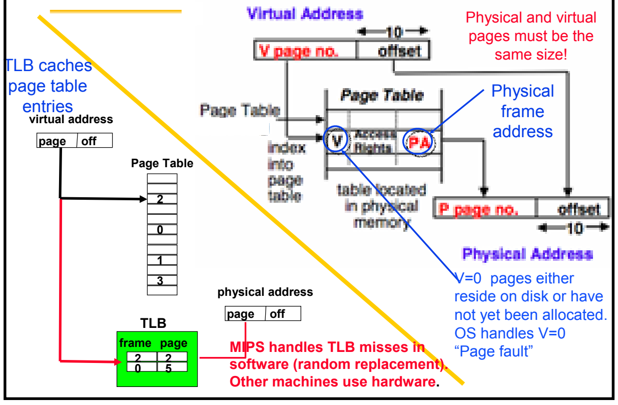
MIPS Address Translation: How does it work?



Translation Look-Aside Buffer (TLB)

- A small fully-associative cache of mappings from virtual to physical addresses
- TLB also contains protection bits for virtual address
- Fast common case: Virtual address is in TLB, process has permission to read/write it.

The TLB caches page table entries

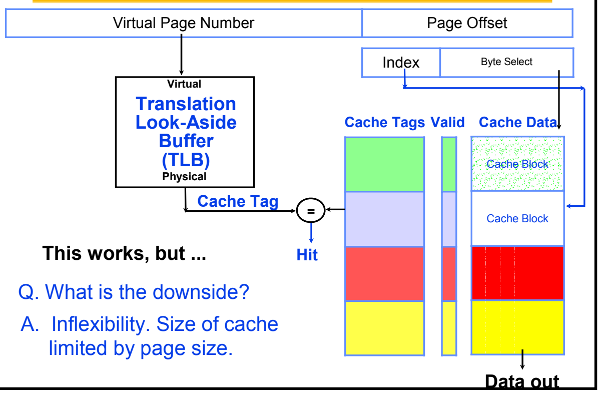


Physical and virtual pages must be the same size!

V=0 pages either reside on disk or have not yet been allocated. OS handles V=0 "Page fault"

MIPS handles TLB misses in software (random replacement). Other machines use hardware.

Can TLB and caching be overlapped?



This works, but ...

Q. What is the downside?

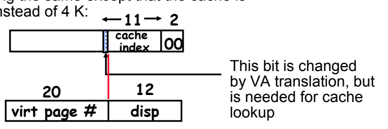
A. Inflexibility. Size of cache limited by page size.

Problems With Overlapped TLB Access

Overlapped access only works as long as the address bits used to index into the cache *do not change* as the result of VA translation

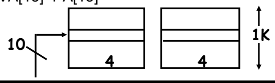
This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:



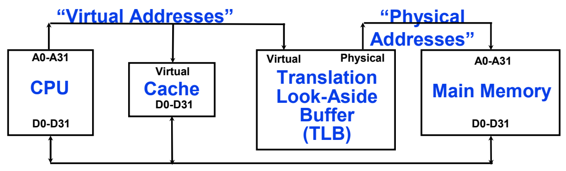
This bit is changed by VA translation, but is needed for cache lookup

Solutions:
go to 8K byte page sizes;
go to 2 way set associative cache;
or SW guarantee VA[13]=PA[13]



2 way set assoc cache

Use virtual addresses for cache?



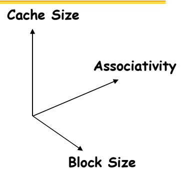
Only use TLB on a cache miss !

Downside: a subtle, fatal problem. What is it?

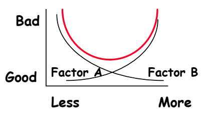
A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a nightmare.

Summary #1/3: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back
  - write allocation



- The optimal choice is a compromise
  - depends on access characteristics
    - » workload
    - » use (I-cache, D-cache, TLB)
  - depends on technology / cost



- Simplicity often wins

Summary #2/3: Caches

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - » Temporal Locality: Locality in Time
    - » Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity. Nightmare Scenario: ping pong effect!
- Write Policy: Write Through vs. Write Back
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): affects Compilers, Data structures, and Algorithms

Summary #3/3: TLB, Virtual Memory

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
  - funny times, as most systems can't access all of 2nd level cache without TLB misses!
- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is more important than memory hierarchy benefits, but computers insecure

Reading

- **This lecture: appendix C *Memory Hierarchy***
- **Next lecture: chapter 2 *Instruction-Level Parallelism***