EECS 252 Graduate Computer Architecture
Lecture 4

Review Continued: Caching & VM
Exceptions
January 30th, 2012

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

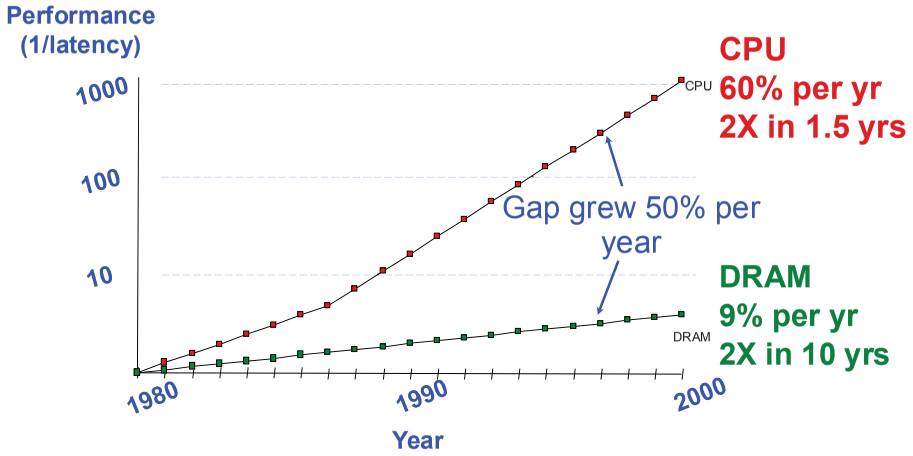
Review: Pipelining

- 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

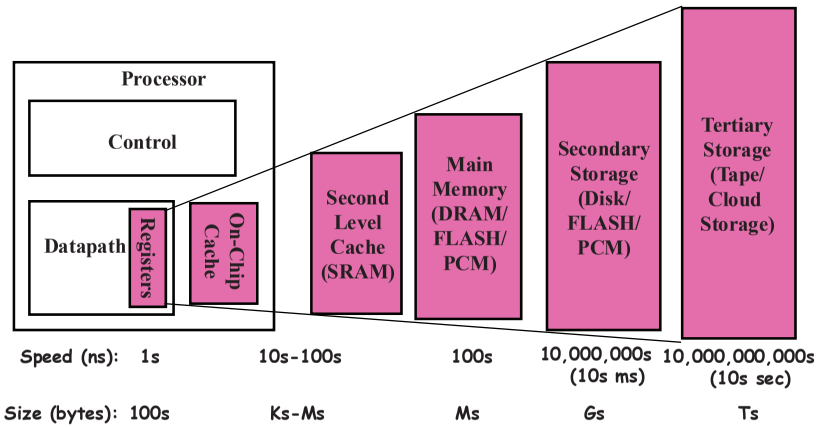
Since 1980, CPU has outpaced DRAM ...



- How do architects address this gap?
  - Put small, fast "cache" memories between CPU and DRAM.
  - Create a "memory hierarchy"

Memory Hierarchy

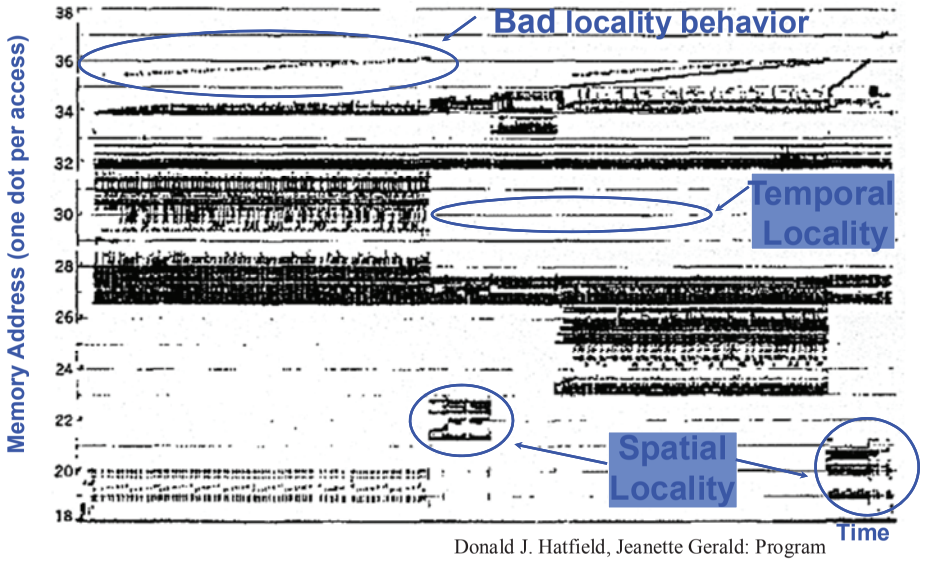
- Take advantage of the principle of locality to:
  - Present as much memory as in the cheapest technology
  - Provide access at speed offered by the fastest technology



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

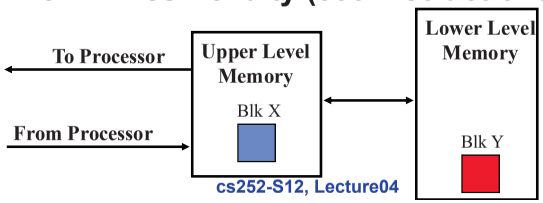
Programs with locality cache well ...



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

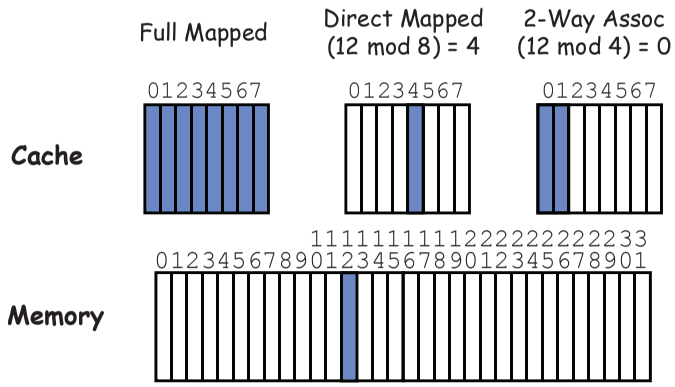
Memory Hierarchy: Terminology

- 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!)



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

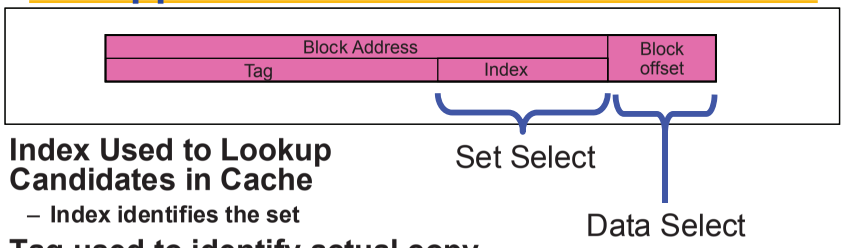
- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant
- **Capacity**:
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- **Conflict** (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- **Coherence** (Invalidation): other process (e.g., I/O) updates memory

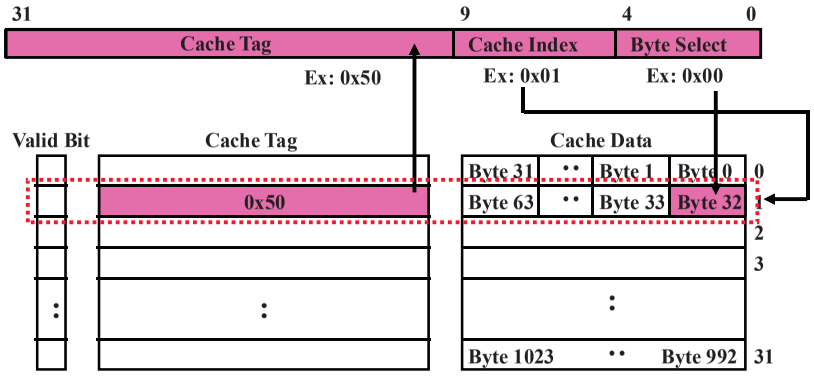
Q2: How is a block found if it is in the upper level?



- **Index Used to Lookup Candidates in Cache**
  - Index identifies the set
- **Tag used to identify actual copy**
  - If no candidates match, then declare cache miss
- **Block is minimum quantum of caching**
  - Data select field used to select data within block
  - Many caching applications don't have data select field
- **Larger block size has distinct hardware advantages:**
  - less tag overhead
  - exploit fast burst transfers from DRAM/over wide busses
- **Disadvantages of larger block size?**
  - Fewer blocks ⇒ more conflicts. Can waste bandwidth

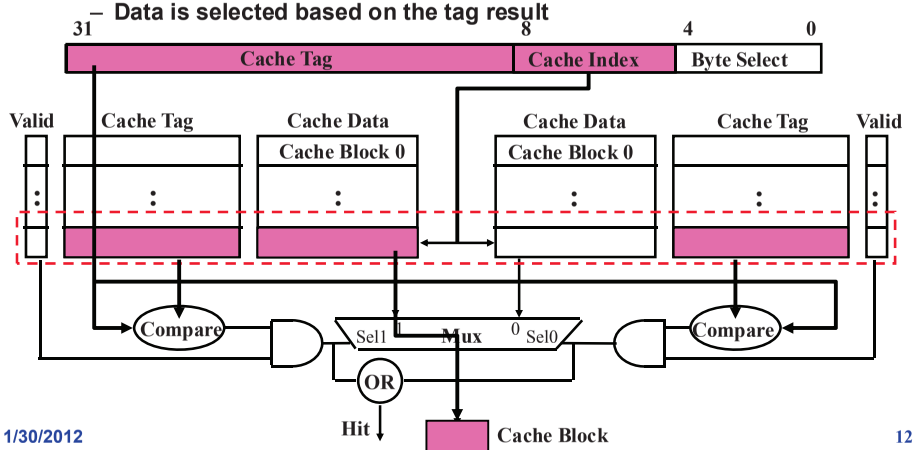
Review: Direct Mapped Cache

- **Direct Mapped 2^N byte cache**:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = 2^M)
- **Example: 1 KB Direct Mapped Cache with 32 B Blocks**
  - Index chooses potential block
  - Tag checked to verify block
  - Byte select chooses byte within block



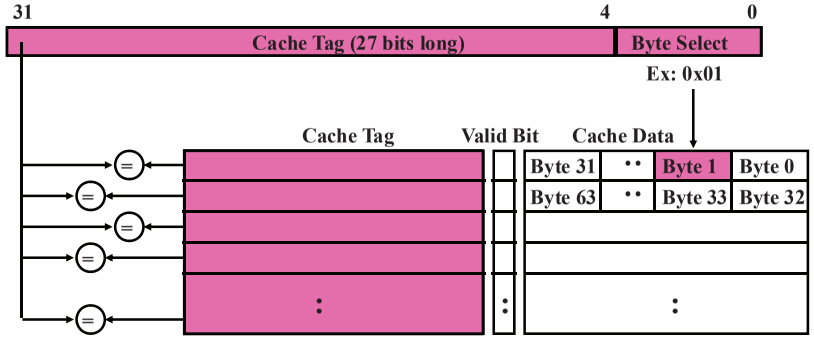
Review: Set Associative Cache

- **N-way set associative**: N entries per Cache Index
  - N direct mapped caches operates in parallel
- **Example: Two-way set associative cache**
  - Cache Index selects a "set" from the cache
  - Two tags in the set are compared to input in parallel
  - Data is selected based on the tag result



Review: Fully Associative Cache

- Fully Associative: Every block can hold any line
  - Address does not include a cache index
  - Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size=32B blocks
  - We need N 27-bit comparators
  - Still have byte select to choose from within block



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, but – how well does it work?

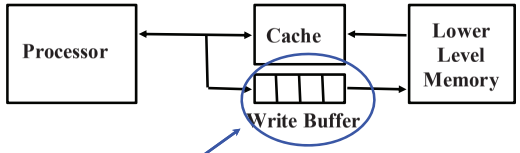
Assoc:	2-way		4-way		8-way	
Size	LRU	Ran	LRU	Ran	LRU	Ran
16K	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64K	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256K	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

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 check write buffers for match on reads

5 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

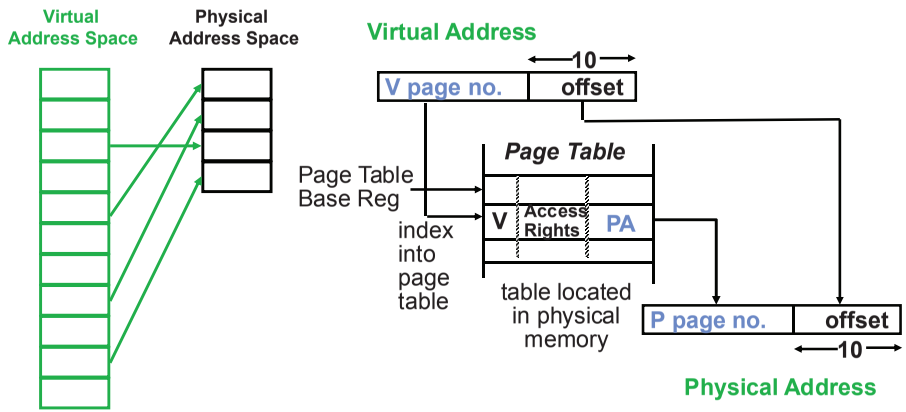
CS 252 Administrivia

- Textbook Reading for next few lectures:
  - Computer Architecture: A Quantitative Approach, Chapter 2
- Don't forget to try to do the prerequisite exams (look at handouts page):
  - See if you have good enough understanding of prerequisite material
- Reading assignment for this week are up. Will work to get much further ahead.
  - Should be using web form to submit your summaries
  - Remember: I've given you 5 slip days on paper summaries
  - Technically, these paper summaries are due before class (1:00)
- Go to handouts page:
  - All readings put up so far are there
- Start thinking about projects!
  - Find a partner, start thinking of topics

Precise Exceptions: Ability to Undo!

- Readings for today:
  - James Smith and Andrew Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors
  - Gurindar Sohi and Sriram Vajapeyam, "Instruction Issue Logic for High-Performance, Interruptable Pipelined Processors"
- Basic ideas:
  - Prevent out of order commit
  - Either delay execution or delay commit
- Options:
  - Reorder Buffer with/without bypassing
  - Future File
- We will see an explicit use of both reorder buffer and future file in next couple of lectures

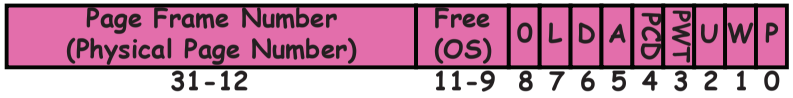
What is virtual memory?



- Virtual memory => treat memory as a cache for the disk
- Terminology: blocks in this cache are called "Pages"
  - Typical size of a page: 1K — 8K
- Page table maps virtual page numbers to physical frames
  - "PTE" = Page Table Entry

What is in a Page Table Entry (PTE)?

- What is in a Page Table Entry (or PTE)?
  - Pointer to next-level page table or to actual page
  - Permission bits: valid, read-only, read-write, write-only
- Example: Intel x86 architecture PTE:
  - Address same format previous slide (10, 10, 12-bit offset)
  - Intermediate page tables called "Directories"

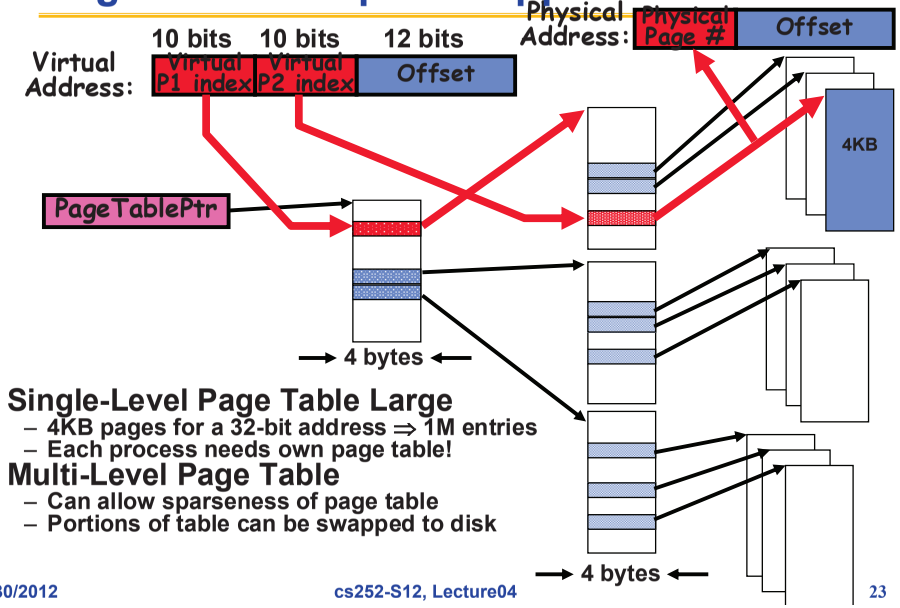


P: Present (same as "valid" bit in other architectures)
W: Writeable
U: User accessible
PWT: Page write transparent: external cache write-through
PCD: Page cache disabled (page cannot be cached)
A: Accessed: page has been accessed recently
D: Dirty (PTE only): page has been modified recently
L: L=1⇒4MB page (directory only).
Bottom 22 bits of virtual address serve as offset

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")

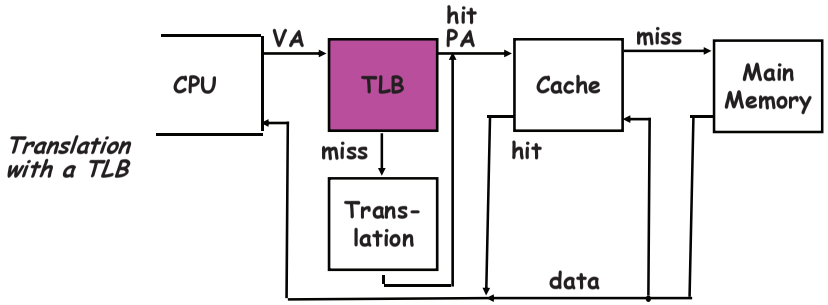
Large Address Space Support



- Single-Level Page Table Large
  - 4KB pages for a 32-bit address ⇒ 1M entries
  - Each process needs own page table!
- Multi-Level Page Table
  - Can allow sparseness of page table
  - Portions of table can be swapped to disk

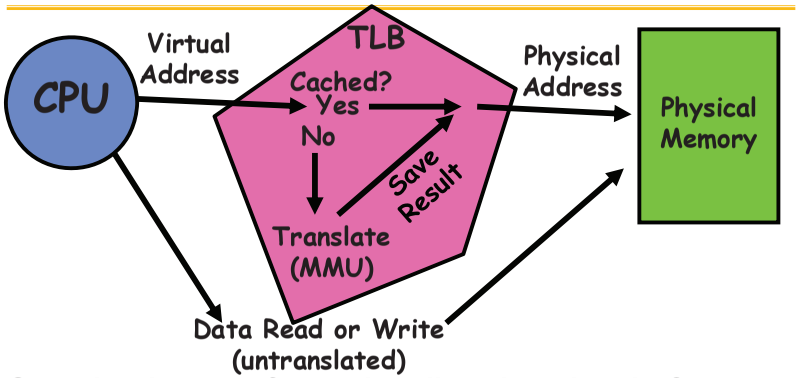
Translation Look-Aside Buffers

- Translation Look-Aside Buffers (TLB)
  - Cache on translations
  - Fully Associative, Set Associative, or Direct Mapped



- TLBs are:
  - Small – typically not more than 128 – 256 entries
  - Fully Associative

Caching Applied to Address Translation

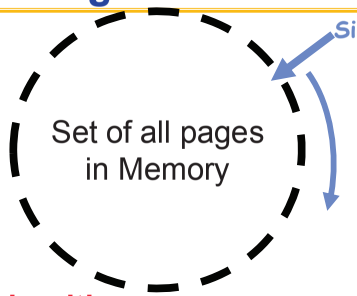


- Question is one of page locality: does it exist?
  - Instruction accesses spend a lot of time on the same page (since accesses sequential)
  - Stack accesses have definite locality of reference
  - Data accesses have less page locality, but still some…
- Can we have a TLB hierarchy?
  - Sure: multiple levels at different sizes/speeds

What Actually Happens on a TLB Miss?

- Hardware traversed page tables:
  - On TLB miss, hardware in MMU looks at current page table to fill TLB (may walk multiple levels)
    - » If PTE valid, hardware fills TLB and processor never knows
    - » If PTE marked as invalid, causes Page Fault, after which kernel decides what to do afterwards
- Software traversed Page tables (like MIPS)
  - On TLB miss, processor receives TLB fault
  - Kernel traverses page table to find PTE
    - » If PTE valid, fills TLB and returns from fault
    - » If PTE marked as invalid, internally calls Page Fault handler
- Most chip sets provide hardware traversal
  - Modern operating systems tend to have more TLB faults since they use translation for many things
  - Examples:
    - » shared segments
    - » user-level portions of an operating system

Clock Algorithm: Not Recently Used



Single Clock Hand:
Advances only on page fault!
Check for pages not used recently
Mark pages as not used recently

Page Table

dirty	used	
1	0	…
1	0	
0	1	
1	1	
0	0	

- Clock Algorithm:
  - Approximate LRU (approx to approx to MIN)
  - Replace an old page, not the oldest page
- Details:
  - Hardware "use" bit per physical page:
    - » Hardware sets use bit on each reference
    - » If use bit isn't set, means not referenced in a long time
  - On page fault:
    - » Advance clock hand (not real time)
    - » Check use bit: 1→used recently; clear and leave alone
      0→selected candidate for replacement

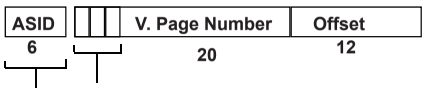
Example: R3000 pipeline

MIPS R3000 Pipeline

Inst Fetch	Dcd/ Reg	ALU / E.A	Memory	Write Reg
TLB I-Cache	RF	Operation		WB
		E.A. TLB	D-Cache	

TLB
64 entry, on-chip, fully associative, software TLB fault handler

Virtual Address Space

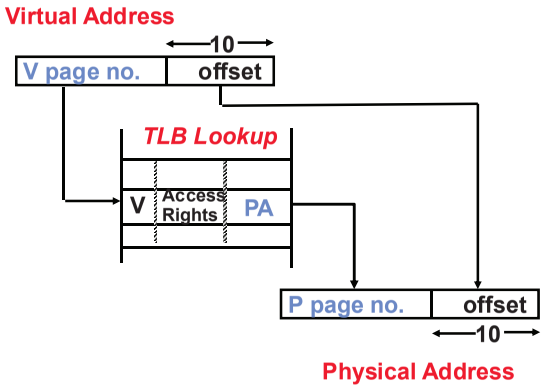


0xx User segment (caching based on PT/TLB entry)
100 Kernel physical space, cached
101 Kernel physical space, uncached
11x Kernel virtual space

Allows context switching among
64 user processes without TLB flush

Reducing translation time further

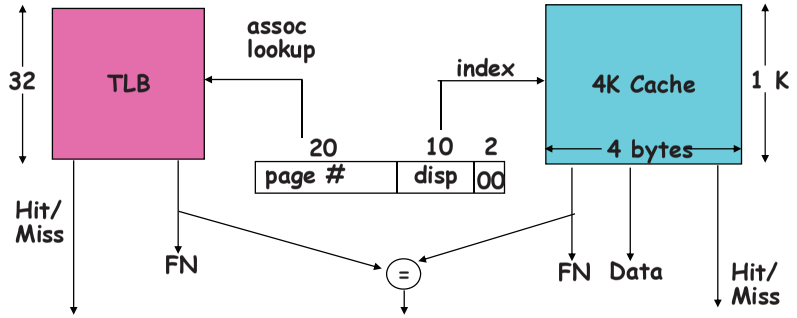
- As described, TLB lookup is in serial with cache lookup:



- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
  - Works because offset available early

Overlapping TLB & Cache Access

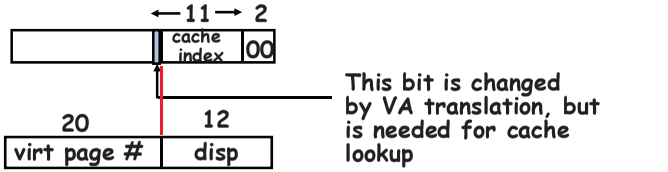
- Here is how this might work with a 4K cache:



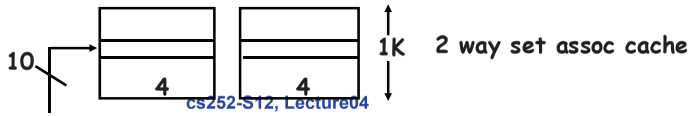
- What if cache size is increased to 8KB?
  - Overlap not complete
  - Need to do something else. See CS152/252
- Another option: Virtual Caches
  - Tags in cache are virtual addresses
  - Translation only happens on cache misses

Problems With Overlapped TLB Access

- Overlapped access requires address bits used to index into cache *do not change* as result 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:



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

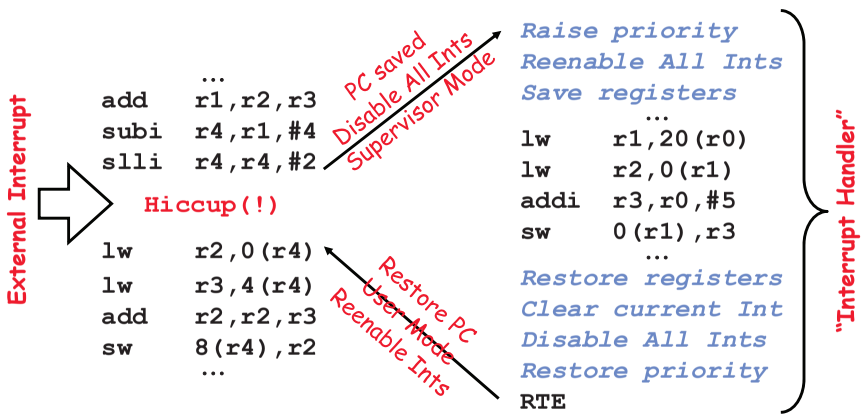


Exceptions: Traps and Interrupts

(Hardware)

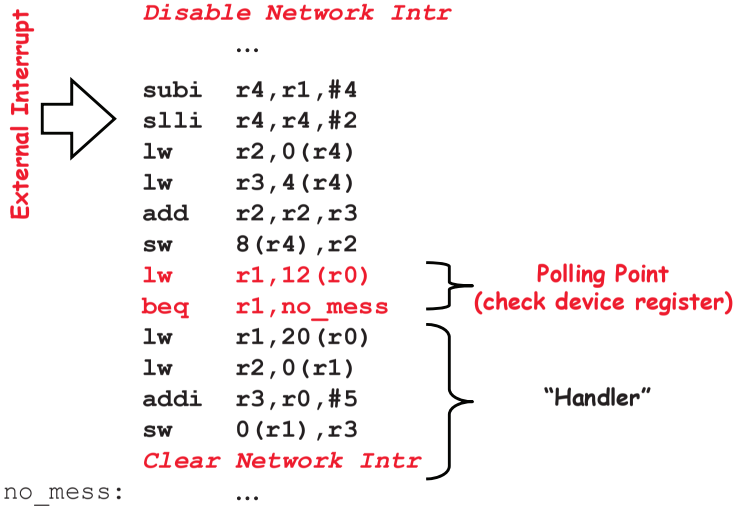
Example: Device Interrupt

(Say, arrival of network message)



Alternative: Polling

(again, for arrival of network message)



Polling is faster/slower than Interrupts.

- Polling is faster than interrupts because
  - Compiler knows which registers in use at polling point. Hence, do not need to save and restore registers (or not as many).
  - Other interrupt overhead avoided (pipeline flush, trap priorities, etc).
- Polling is slower than interrupts because
  - Overhead of polling instructions is incurred regardless of whether or not handler is run. This could add to inner-loop delay.
  - Device may have to wait for service for a long time.
- When to use one or the other?
  - Multi-axis tradeoff
    - » Frequent/regular events good for polling, *as long as device can be controlled at user level.*
    - » Interrupts good for infrequent/irregular events
    - » Interrupts good for ensuring regular/predictable service of events.

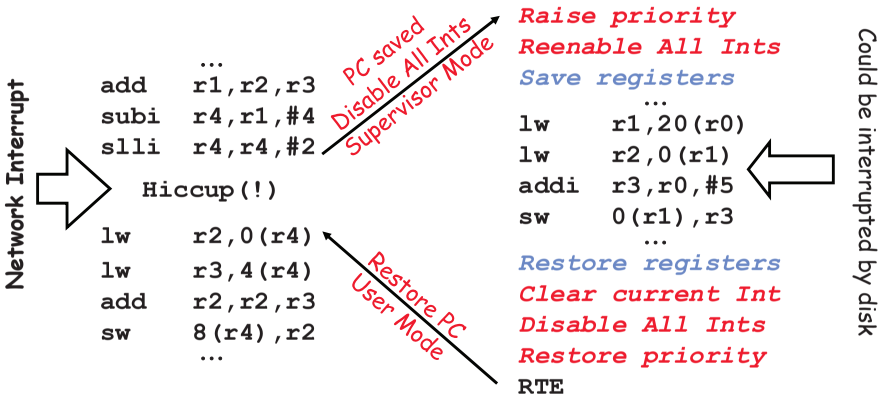
Trap/Interrupt classifications

- *Traps:* relevant to the current process
  - Faults, arithmetic traps, and synchronous traps
  - Invoke software on behalf of the currently executing process
- *Interrupts:* caused by asynchronous, outside events
  - I/O devices requiring service (DISK, network)
  - Clock interrupts (real time scheduling)
- *Machine Checks:* caused by serious hardware failure
  - Not always restartable
  - Indicate that bad things have happened.
    - » Non-recoverable ECC error
    - » Machine room fire
    - » Power outage

A related classification: Synchronous vs. Asynchronous

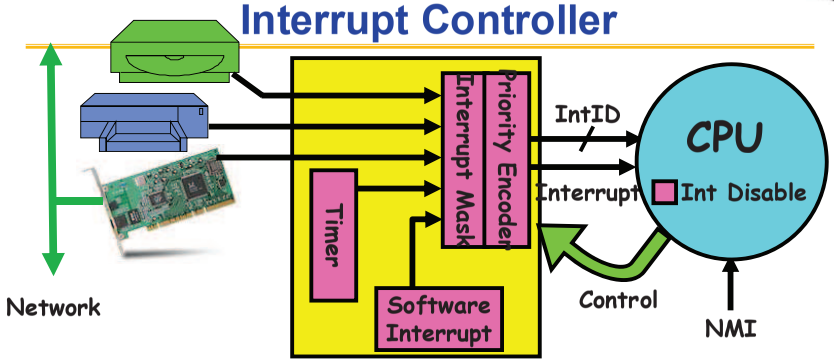
- ***Synchronous:*** **means related to the instruction stream, i.e. during the execution of an instruction**
  - Must stop an instruction that is currently executing
  - Page fault on load or store instruction
  - Arithmetic exception
  - Software Trap Instructions
- ***Asynchronous:*** **means unrelated to the instruction stream, i.e. caused by an outside event.**
  - Does not have to disrupt instructions that are already executing
  - Interrupts are asynchronous
  - Machine checks are asynchronous
- ***SemiSynchronous (or high-availability interrupts):***
  - Caused by external event but may have to disrupt current instructions in order to guarantee service

Interrupt Priorities Must be Handled



Note that priority must be raised to avoid recursive interrupts!

Interrupt Controller



- **Interrupts invoked with interrupt lines from devices**
- **Interrupt controller chooses interrupt request to honor**
  - Mask enables/disables interrupts
  - Priority encoder picks highest enabled interrupt
  - Software Interrupt Set/Cleared by Software
  - Interrupt identity specified with ID line
- **CPU can disable all interrupts with internal flag**
- **Non-maskable interrupt line (NMI) can't be disabled**

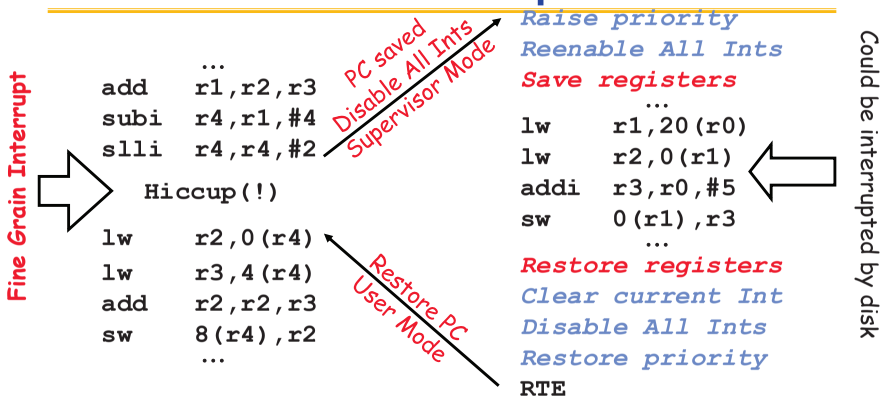
Interrupt controller hardware and mask levels

- **Operating system constructs a hierarchy of masks that reflects some form of interrupt priority.**
- **For instance:**

Priority	Examples
0	Software interrupts
2	Network Interrupts
4	Sound card
5	Disk Interrupt
6	Real Time clock
∞	Non-Maskable Ints (power)

  - This reflects the an order of urgency to interrupts
  - For instance, this ordering says that disk events can interrupt the interrupt handlers for network interrupts.

Can we have fast interrupts?



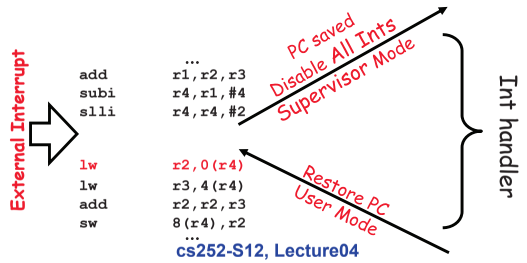
- Pipeline Drain: Can be very Expensive
- Priority Manipulations
- Register Save/Restore
  - 128 registers + cache misses + etc.

SPARC (and RISC I) had register windows

- On interrupt or procedure call, simply switch to a different set of registers
- Really saves on interrupt overhead
  - Interrupts can happen at any point in the execution, so compiler cannot help with knowledge of live registers.
  - Conservative handlers must save all registers
  - Short handlers might be able to save only a few, but this analysis is compilcated
- Not as big a deal with procedure calls
  - Original statement by Patterson was that Berkeley didn't have a compiler team, so they used a hardware solution
  - Good compilers can allocate registers across procedure boundaries
  - Good compilers know what registers are live at any one time
- However, register windows have returned!
  - IA64 has them
  - Many other processors have shadow registers for interrupts

Precise Interrupts/Exceptions

- An interrupt or exception is considered *precise* if there is a single instruction (or interrupt point) for which:
  - All instructions before that have committed their state
  - No following instructions (including the interrupting instruction) have modified any state.
- This means, that you can restart execution at the interrupt point and "get the right answer"
  - Implicit in our previous example of a device interrupt:
    - » Interrupt point is at first lw instruction



Precise interrupt point may require multiple PCs

```
      addi  r4,r3,#4
      sub   r1,r2,r3
PC:   bne   r1,there
PC+4: and   r2,r3,r5
      <other insts>
```

⇦ Interrupt point described as <PC,PC+4>

```
      addi  r4,r3,#4
      sub   r1,r2,r3
PC:   bne   r1,there
PC+4: and   r2,r3,r5
      <other insts>
```

⇦ Interrupt point described as:
<PC+4,there> (branch was taken)
or
<PC+4,PC+8> (branch was not taken)

- On SPARC, interrupt hardware produces "pc" and "npc" (next pc)
- On MIPS, only "pc" – must fix point in software

Why are precise interrupts desirable?

- Many types of interrupts/exceptions need to be restartable. Easier to figure out what actually happened:
  - I.e. TLB faults. Need to fix translation, then restart load/store
  - IEEE gradual underflow, illegal operation, etc:

    e.g. Suppose you are computing: $f(x) = \frac{\sin(x)}{x}$
    Then, for $x \rightarrow 0$

    $$f(0) = \frac{0}{0} \Rightarrow NaN + illegal_operation$$

    Want to take exception, replace *NaN* with 1, then restart.
- Restartability doesn't *require* preciseness. However, preciseness makes it *a lot easier* to restart.
- Simplify the task of the operating system a lot
  - Less state needs to be saved away if unloading process.
  - Quick to restart (making for fast interrupts)

Precise Exceptions in simple 5-stage pipeline:

- Exceptions may occur at different stages in pipeline (I.e. out of order):
  - Arithmetic exceptions occur in execution stage
  - TLB faults can occur in instruction fetch or memory stage
- What about interrupts? The doctor's mandate of "do no harm" applies here: try to interrupt the pipeline as little as possible
- All of this solved by tagging instructions in pipeline as "cause exception or not" and wait until end of memory stage to flag exception
  - Interrupts become marked NOPs (like bubbles) that are placed into pipeline instead of an instruction.
  - Assume that interrupt condition persists in case NOP flushed
  - Clever instruction fetch might start fetching instructions from interrupt vector, but this is complicated by need for supervisor mode switch, saving of one or more PCs, etc

Another look at the exception problem

- Use pipeline to sort this out!
  - Pass exception status along with instruction.
  - Keep track of PCs for every instruction in pipeline.
  - Don't act on exception until it reache WB stage
- Handle interrupts through "faulting noop" in IF stage
- When instruction reaches WB stage:
  - Save PC ⇒ EPC, Interrupt vector addr ⇒ PC
  - Turn all instructions in earlier stages into noops!

Summary: 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: 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;

Summary: Interrupts

- Interrupts and Exceptions either interrupt the current instruction or happen between instructions
  - Possibly large quantities of state must be saved before interrupting
- Machines with precise exceptions provide one single point in the program to restart execution
  - All instructions before that point have completed
  - No instructions after or including that point have completed