Lecture 2: Review of Instruction Sets, Pipelines, and Caches

Prof. David A. Patterson
Computer Science 252
Spring 1998

Review, #1

- Designing to Last through Trends

	Capacity	Speed
Logic	2x in 3 years	2x in 3 years
DRAM	4x in 3 years	2x in 10 years
Disk	4x in 3 years	2x in 10 years
Processor	(n.a.)	2x in 1.5 years

- Time to run the task
  - Execution time, response time, latency
- Tasks per day, hour, week, sec, ns, …
  - Throughput, bandwidth
- "X is n times faster than Y" means

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}}$$

Review, #2

- Amdahl's Law:

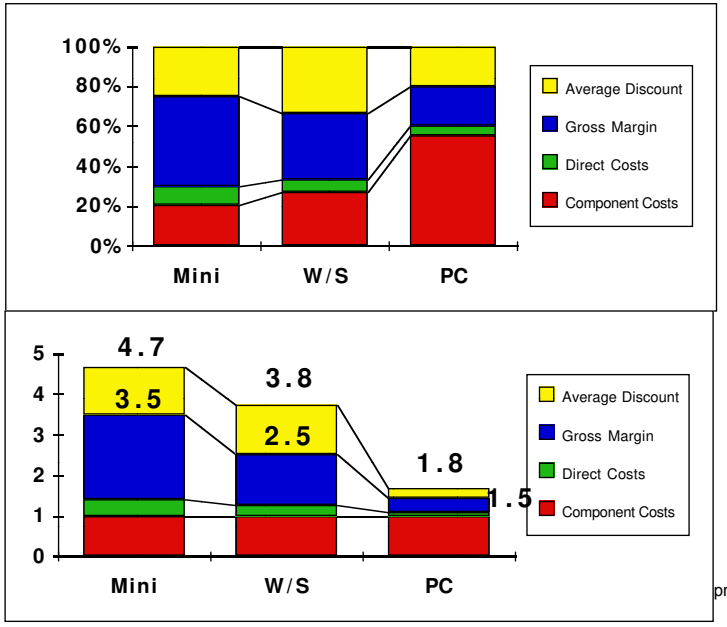
$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

- CPI Law:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

- Execution time is the REAL measure of computer performance!
- Good products created when have:
  - Good benchmarks
  - Good ways to summarize performance
- Die Cost goes roughly with die area^4

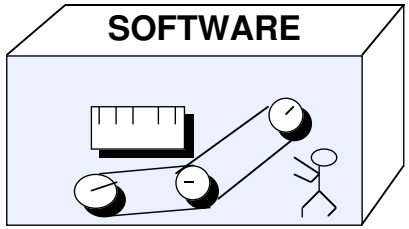
Review, #3: Price vs. Cost



Computer Architecture Is …

the attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

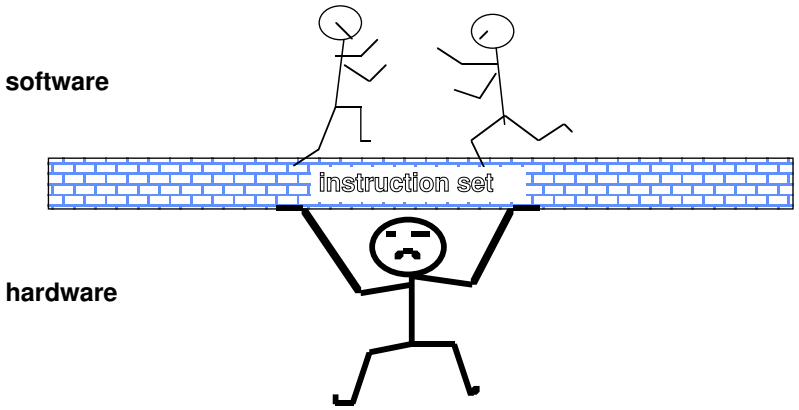
Amdahl, Blaaw, and Brooks, 1964



Computer Architecture's Changing Definition

- 1950s to 1960s:
  Computer Architecture Course = Computer Arithmetic
- 1970s to mid 1980s:
  Computer Architecture Course = Instruction Set Design, especially ISA appropriate for compilers
- 1990s:
  Computer Architecture Course = Design of CPU, memory system, I/O system, Multiprocessors

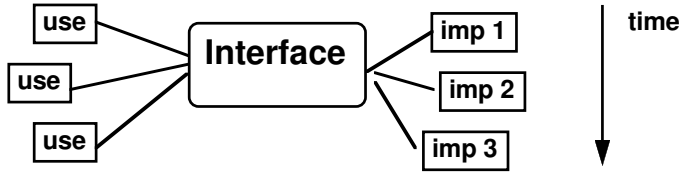
Instruction Set Architecture (ISA)



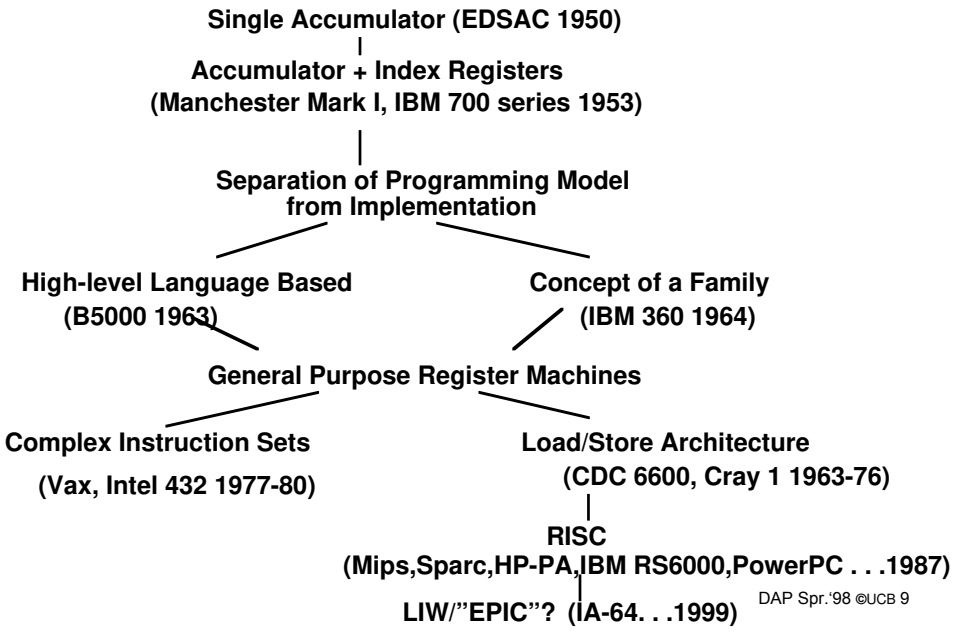
Interface Design

A good interface:

- Lasts through many implementations (portability, compatability)
- Is used in many differeny ways (generality)
- Provides convenient functionality to higher levels
- Permits an efficient implementation at lower levels



Evolution of Instruction Sets



Evolution of Instruction Sets

- Major advances in computer architecture are typically associated with landmark instruction set designs
  - Ex: Stack vs GPR (System 360)
- Design decisions must take into account:
  - technology
  - machine organization
  - programming langauges
  - compiler technology
  - operating systems
- And they in turn influence these

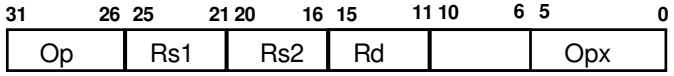
A "Typical" RISC

- 32-bit fixed format instruction (3 formats)
- 32 32-bit GPR (R0 contains zero, DP take pair)
- 3-address, reg-reg arithmetic instruction
- Single address mode for load/store: base + displacement
  - no indirection
- Simple branch conditions
- Delayed branch

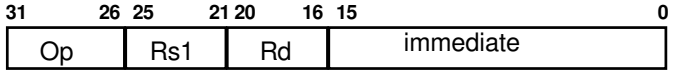
see: SPARC, MIPS, HP PA-Risc, DEC Alpha, IBM PowerPC, CDC 6600, CDC 7600, Cray-1, Cray-2, Cray-3

Example: MIPS (≈ DLX)

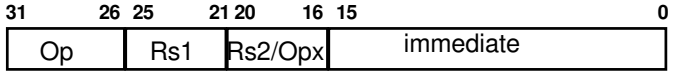
Register-Register



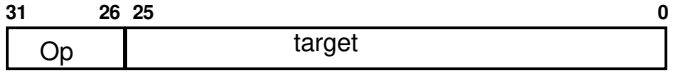
Register-Immediate



Branch

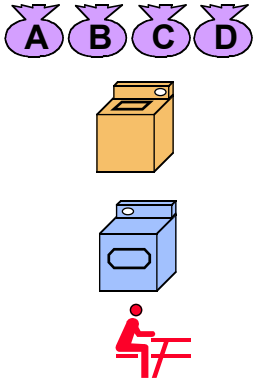


Jump / Call

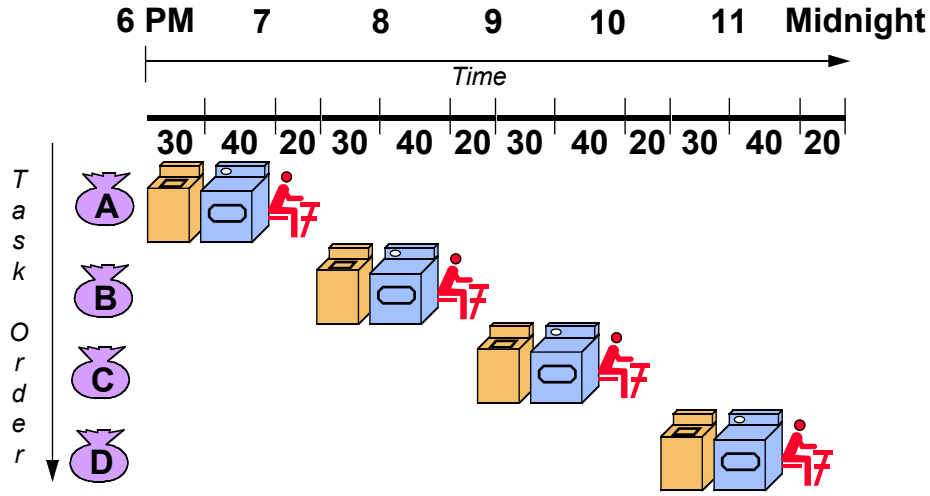


Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- "Folder" takes 20 minutes

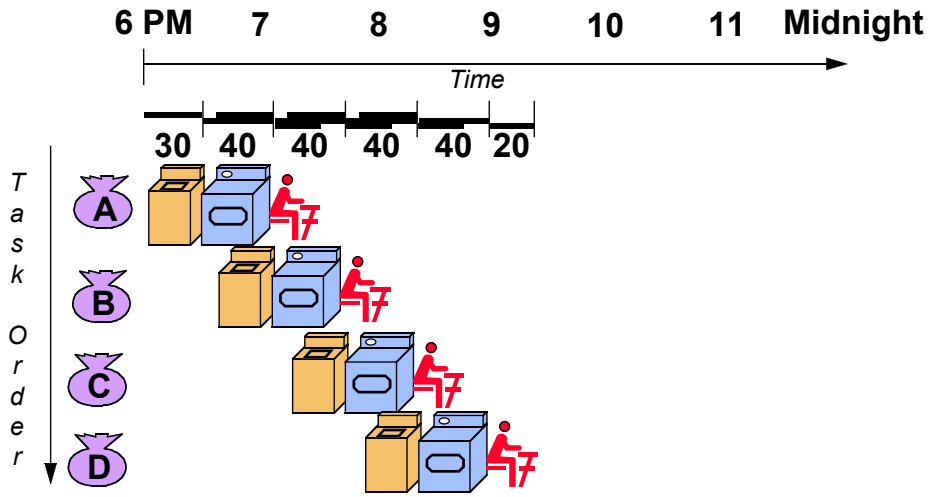


Sequential Laundry



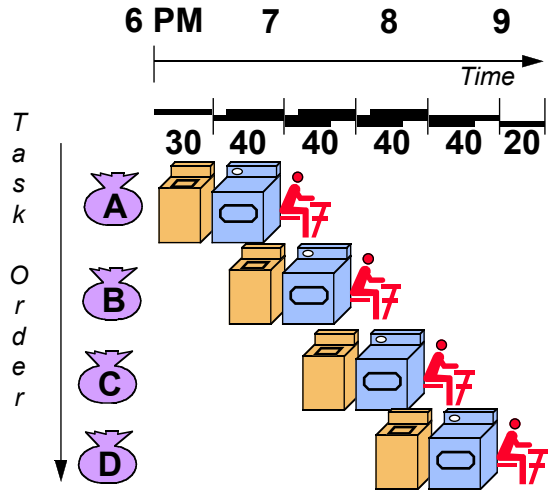
- Sequential laundry takes 6 hours for 4 loads
- If they learned pipelining, how long would laundry take?

Pipelined Laundry Start work ASAP



- Pipelined laundry takes 3.5 hours for 4 loads

Pipelining Lessons



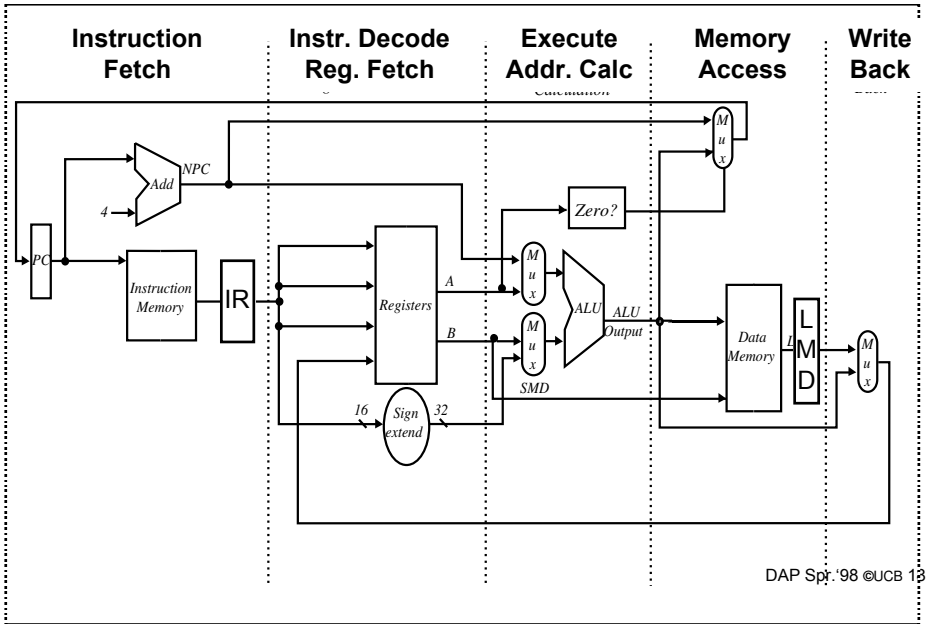
- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

Computer Pipelines

- Execute billions of instructions, so throughout is what matters
- DLX desirable features: all instructions same length, registers located in same place in instruction format, memory operands only in loads or stores

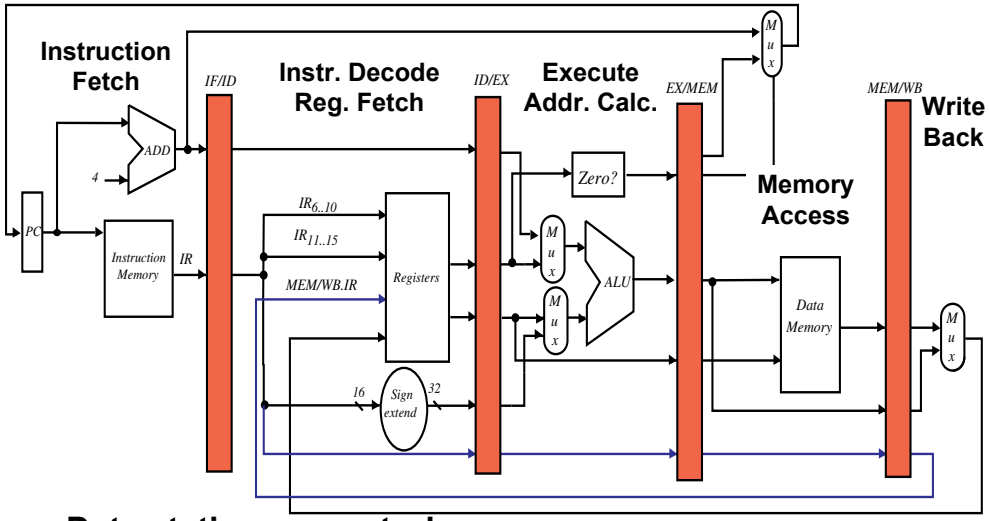
5 Steps of DLX Datapath

Figure 3.1, Page 130



Pipelined DLX Datapath

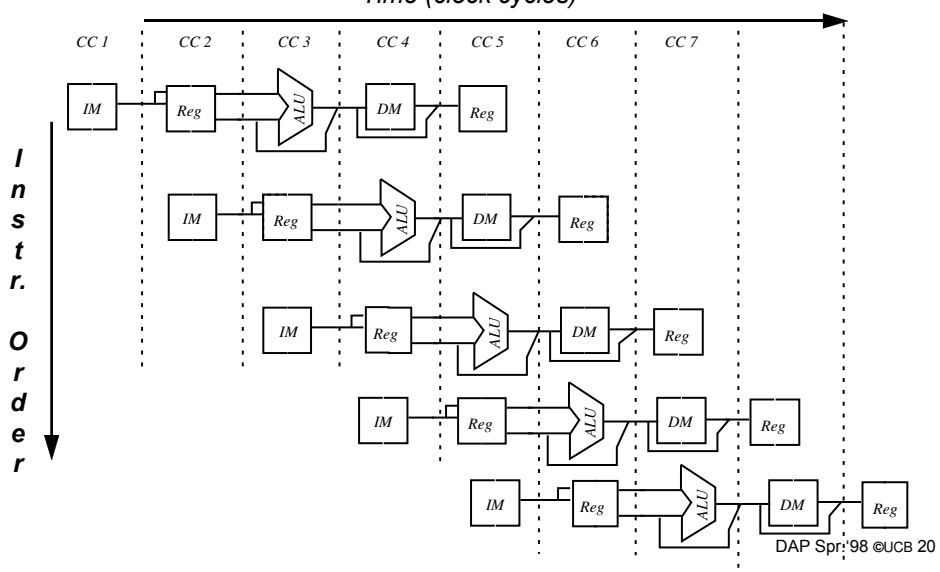
Figure 3.4, page 137



- Data stationary control
  - local decode for each instruction phase / pipeline stage

Visualizing Pipelining

Figure 3.3, Page 133

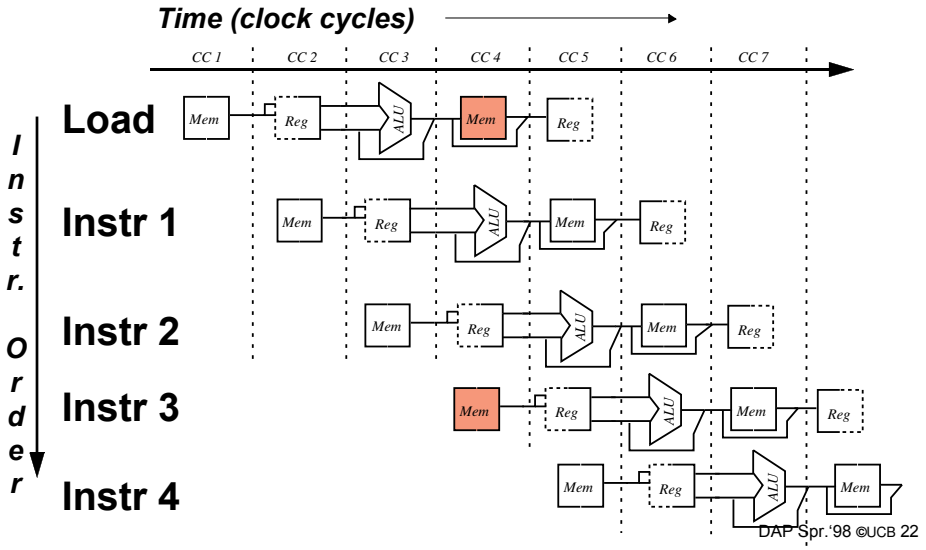


Its Not That Easy for Computers

- Limits to pipelining: Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: HW cannot support this combination of instructions (single person to fold and put clothes away)
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline (missing sock)
  - Control hazards: Pipelining of branches & other instructions that change the PC
  - Common solution is to stall the pipeline until the hazard is resolved, inserting one or more "bubbles" in the pipeline

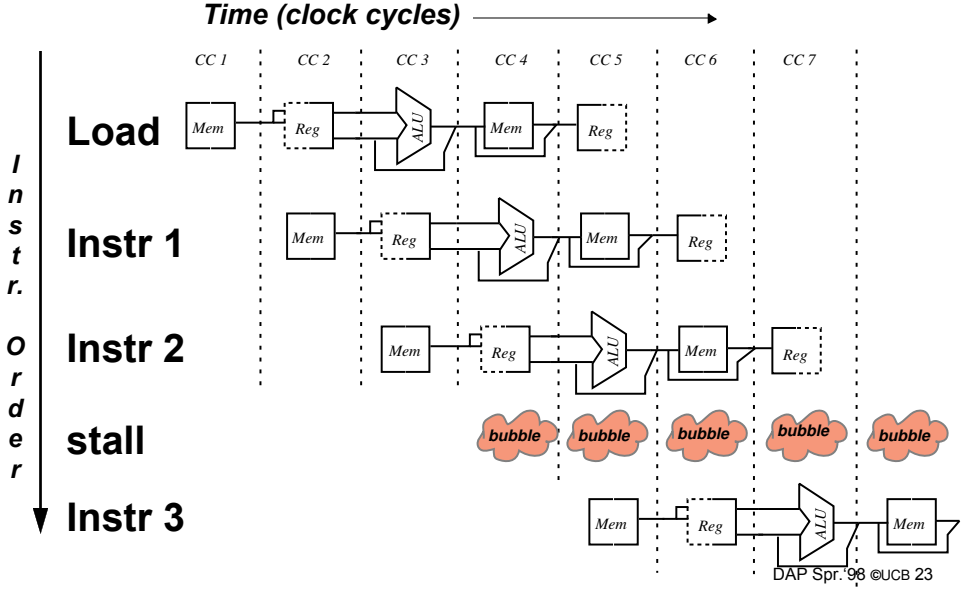
One Memory Port/Structural Hazards

Figure 3.6, Page 142



One Memory Port/Structural Hazards

Figure 3.7, Page 143



Speed Up Equation for Pipelining

$$CPI_{pipelined} = \text{Ideal CPI} + \text{Pipeline stall clock cycles per instr}$$

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

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

Example: Dual-port vs. Single-port

- Machine A: Dual ported memory
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

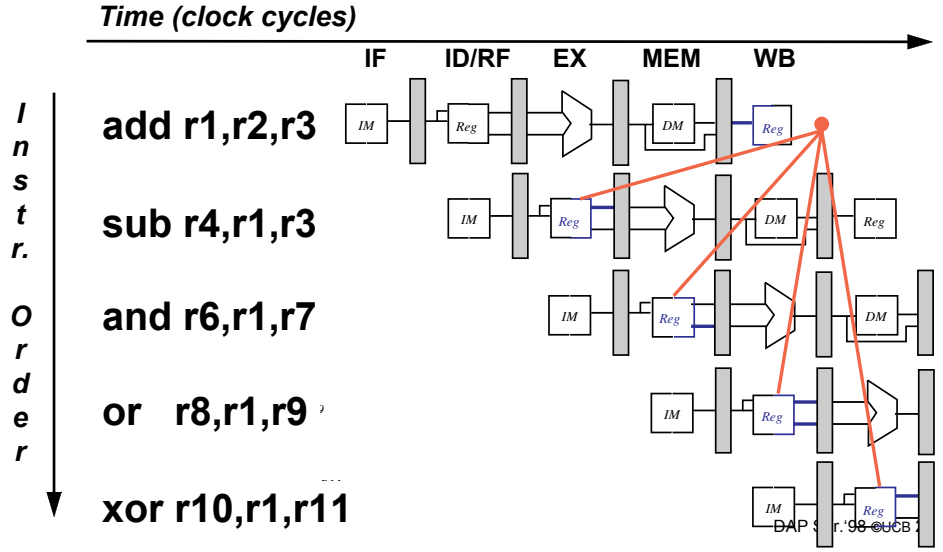
$$SpeedUp_A = Pipeline\ Depth/(1 + 0) \times (clock_{unpipe}/clock_{pipe})$$
$$= Pipeline\ Depth$$
$$SpeedUp_B = Pipeline\ Depth/(1 + 0.4 \times 1) \times (clock_{unpipe}/(clock_{unpipe} / 1.05)$$
$$= (Pipeline\ Depth/1.4) \times 1.05$$
$$= 0.75 \times Pipeline\ Depth$$

$SpeedUp_A$ / $SpeedUp_B$ = Pipeline Depth/(0.75 x Pipeline Depth) = 1.33

- Machine A is 1.33 times faster

Data Hazard on R1

Figure 3.9, page 147



Three Generic Data Hazards

$Instr_I$ followed by $Instr_J$

- Read After Write (RAW)
  $Instr_J$ tries to read operand before $Instr_I$ writes it

Three Generic Data Hazards

$Instr_I$ followed by $Instr_J$

- Write After Read (WAR)
  $Instr_J$ tries to write operand ***before*** $Instr_I$ reads i
  - Gets wrong operand
- Can't happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Three Generic Data Hazards

Instr$_I$ followed by Instr$_J$

- Write After Write (WAW)
  Instr$_J$ tries to write operand ***before*** Instr$_I$ writes it
  - Leaves wrong result (Instr$_I$ not Instr$_J$)
- Can't happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5
- Will see WAR and WAW in later more complicated pipes

CS 252 Administrivia

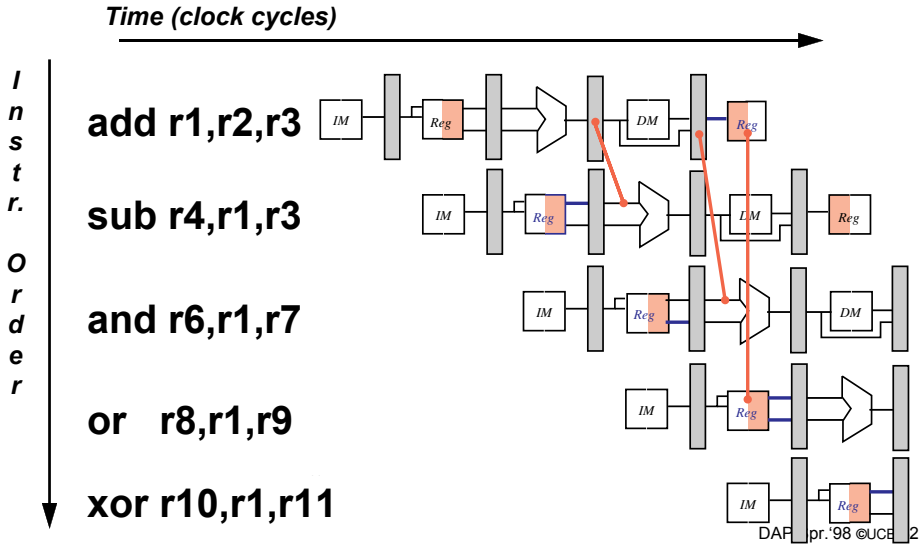
- Too many students with too varied background?
  - In past, CS grad students took written prelim exams on undergraduate material in hardware, software, and theory
  - Prelims were dropped => some unprepared for CS 252?
- In class exam on Wednesday January 28
  - Improve 252 experience if recapture common background
  - Bring 1 sheet of paper with notes on both sides
  - Doesn't affect grade, only admission into class
  - 2 grades: Admitted or audit/take CS 152 1st (before class Friday)
- Review: Chapters 1- 3, CS 152 home page, maybe "Computer Organization and Design (COD)2/e"
  - If did take a class, be sure COD Chapters 2, 6, 7 are familiar
  - Copies in Bechtel Library on 2-hour reserve

CS 252 Administrivia

- Too many students?
- 61 students at 1st lecture
  - To give proper attention to projects (as well as homeworks and quizes), I can handle up to 36 students
- Limiting Number of Students
  - First priority is first year CS/ EECS grad students (32)
  - Second priority is N-th year CS/ EECS grad students (21)
  - Third priority is College of Engineering grad students (1)
  - Fourth priority is CS/EECS undegraduate seniors (7)
    (Note: 1 graduate course unit = 2 undergraduate course units)
  - All other categories
- If not this semester, 252 is offered regularily (Fall)

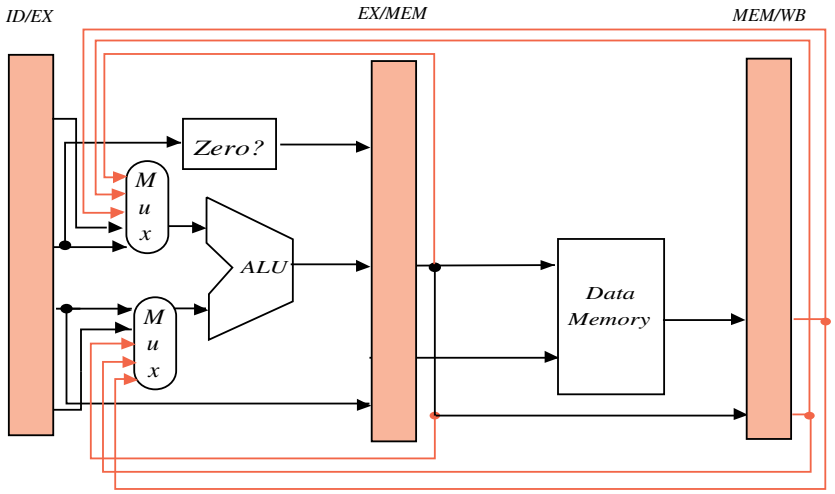
Forwarding to Avoid Data Hazard

Figure 3.10, Page 149



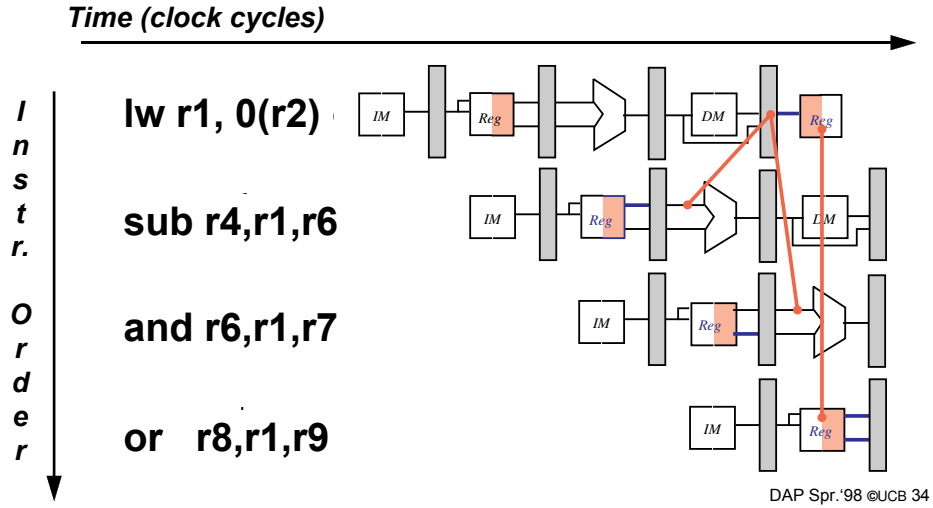
HW Change for Forwarding

Figure 3.20, Page 161



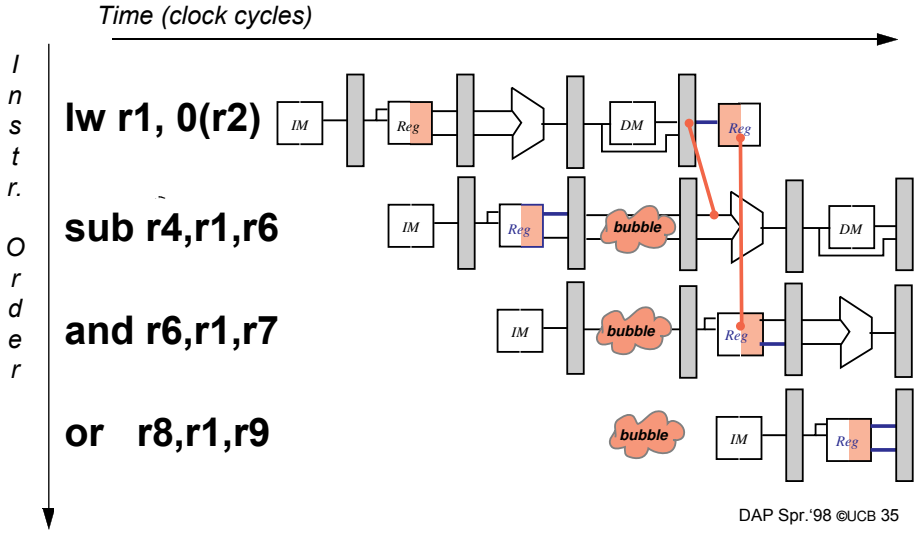
Data Hazard Even with Forwarding

Figure 3.12, Page 153



Data Hazard Even with Forwarding

Figure 3.13, Page 154



Software Scheduling to Avoid Load Hazards

Try producing fast code for

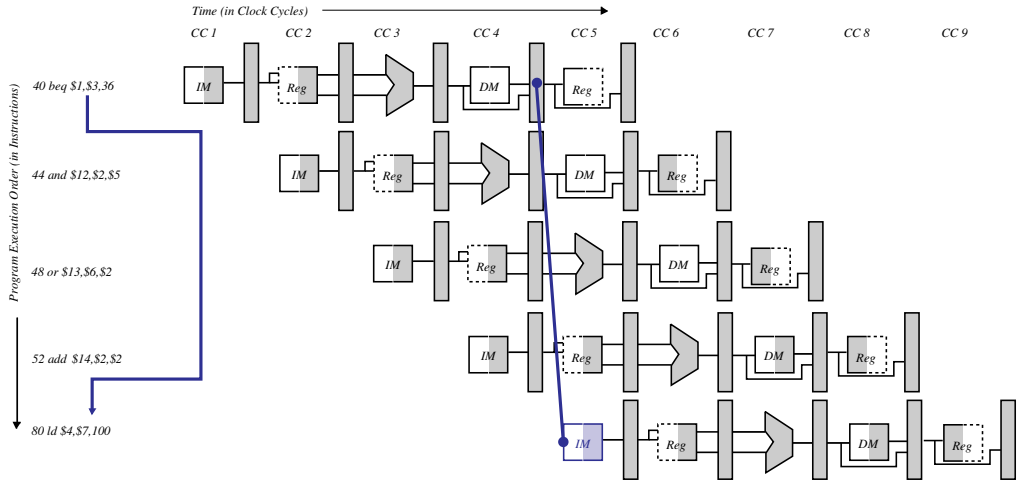
a = b + c;

d = e – f;

assuming a, b, c, d ,e, and f in memory.

Slow code:		Fast code:	
LW	Rb,b	LW	Rb,b
LW	Rc,c	LW	Rc,c
ADD	Ra,Rb,Rc	LW	Re,e
SW	a,Ra	ADD	Ra,Rb,Rc
LW	Re,e	LW	Rf,f
LW	Rf,f	SW	a,Ra
SUB	Rd,Re,Rf	SUB	Rd,Re,Rf
SW	d,Rd	SW	d,Rd

Control Hazard on Branches Three Stage Stall

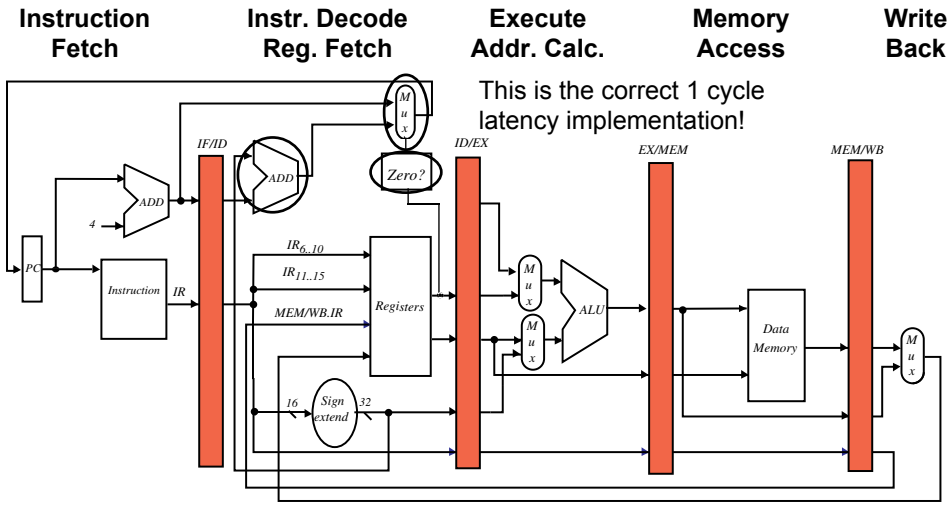


Branch Stall Impact

- If CPI = 1, 30% branch, Stall 3 cycles => new CPI = 1.9!
- Two part solution:
  - Determine branch taken or not sooner, AND
  - Compute taken branch address earlier
- DLX branch tests if register = 0 or ≠ 0
- DLX Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - 1 clock cycle penalty for branch versus 3

Pipelined DLX Datapath

Figure 3.22, page 163



Four Branch Hazard Alternatives

#1: Stall until branch direction is clear

#2: Predict Branch Not Taken

- Execute successor instructions in sequence
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% DLX branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch Taken

- 53% DLX branches taken on average
- But haven't calculated branch target address in DLX
  - » DLX still incurs 1 cycle branch penalty
  - » Other machines: branch target known before outcome

Four Branch Hazard Alternatives

#4: Delayed Branch

- Define branch to take place AFTER a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  ........
  sequential successorn
branch target if taken
```

Branch delay of length *n*

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- DLX uses this

Delayed Branch

- **Where to get instructions to fill branch delay slot?**
  - Before branch instruction
  - From the target address: only valuable when branch taken
  - From fall through: only valuable when branch not taken
  - Cancelling branches allow more slots to be filled
- **Compiler effectiveness for single branch delay slot:**
  - Fills about 60% of branch delay slots
  - About 80% of instructions executed in branch delay slots useful in computation
  - About 50% (60% x 80%) of slots usefully filled
- **Delayed Branch downside: 7-8 stage pipelines, multiple instructions issued per clock (superscalar)**

Evaluating Branch Alternatives

$$\text{Pipeline speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch frequency} \times \text{Branch penalty}}$$

Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. unpipelined*	*speedup v. stall*
Stall pipeline	3	1.42	3.5	1.0
Predict taken	1	1.14	4.4	1.26
Predict not taken	1	1.09	4.5	1.29
Delayed branch	0.5	1.07	4.6	1.31

Conditional & Unconditional = 14%, 65% change PC

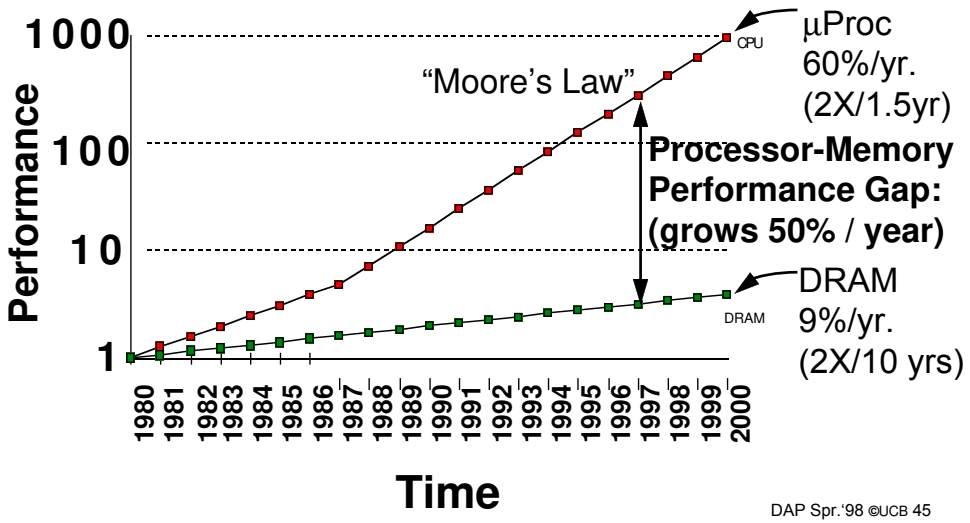
Pipelining Introduction Summary

- **Just overlap tasks, and easy if tasks are independent**
- **Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:**

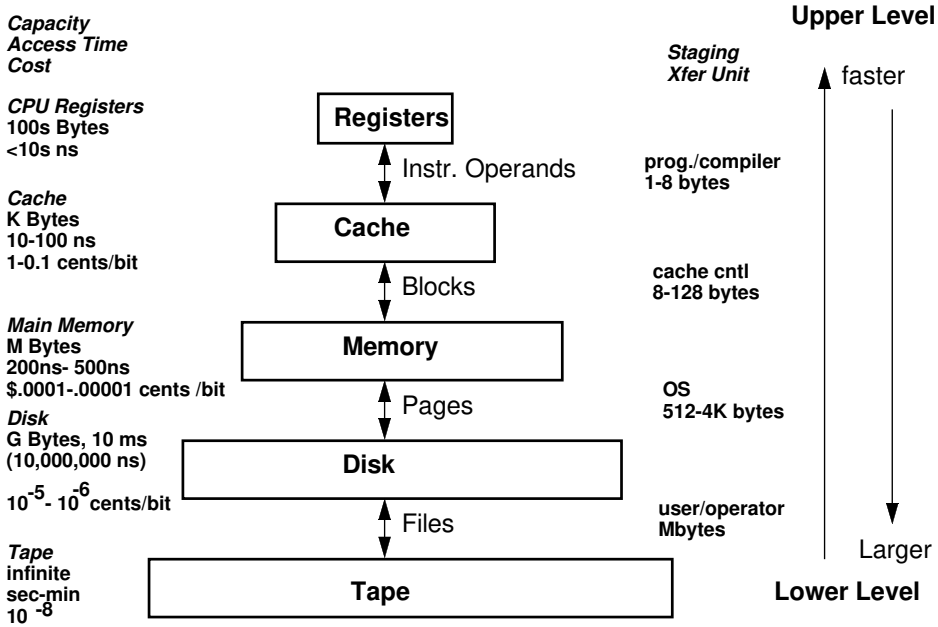
$$\text{Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle Unpipelined}}{\text{Clock Cycle Pipelined}}$$

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

Recap: Who Cares About the Memory Hierarchy?



Levels of the Memory Hierarchy

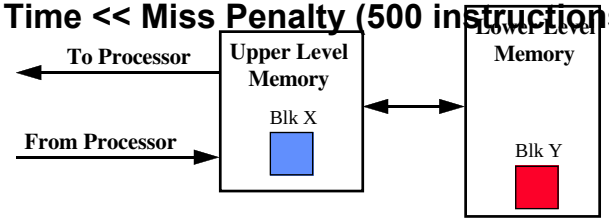


The Principle of Locality

- 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 localilty for speed

Memory Hierarchy: Terminology

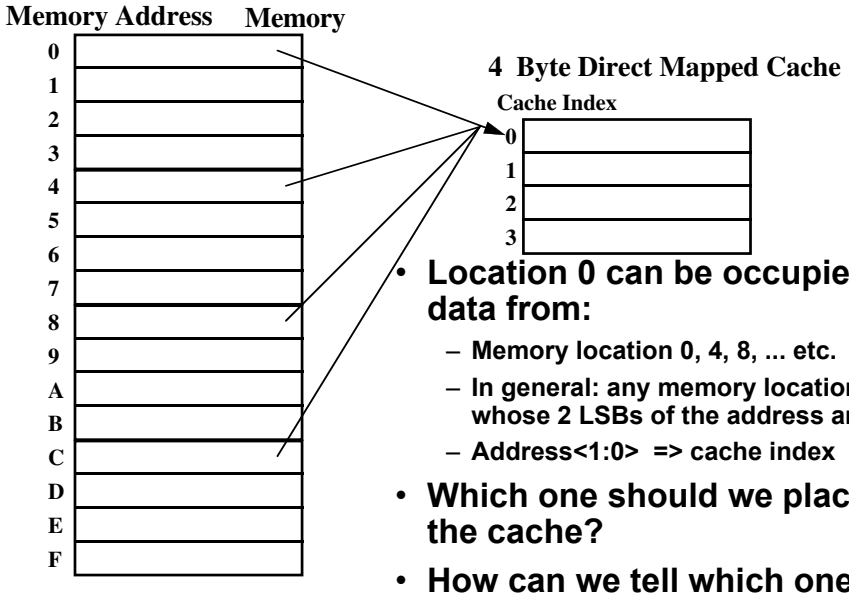
- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: the fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- Miss: data needs to be retrieve from a block in the 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(BW between upper & lower levels)

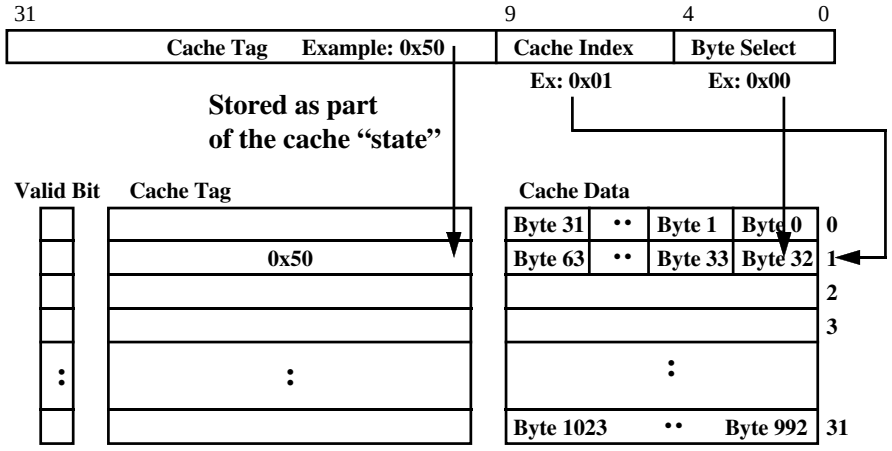
Simplest Cache: Direct Mapped



- Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ... etc.
  - In general: any memory location whose 2 LSBs of the address are 0s
  - Address<1:0> => cache index
- Which one should we place in the cache?
- How can we tell which one is in the cache?

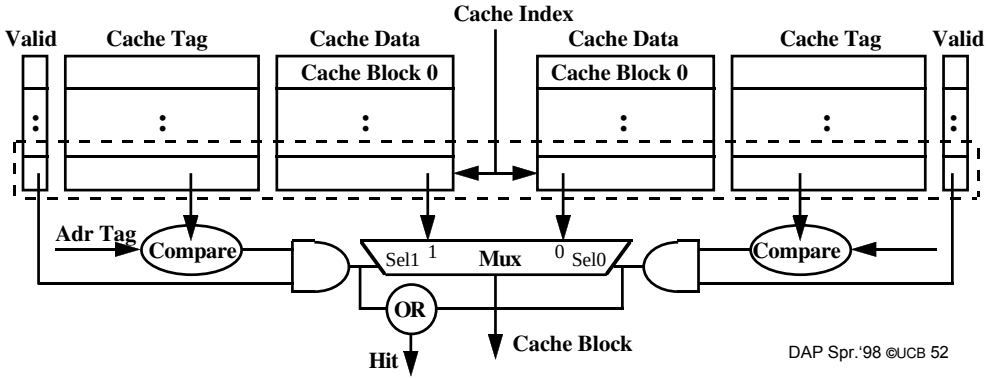
1 KB Direct Mapped Cache, 32B blocks

- For a 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)



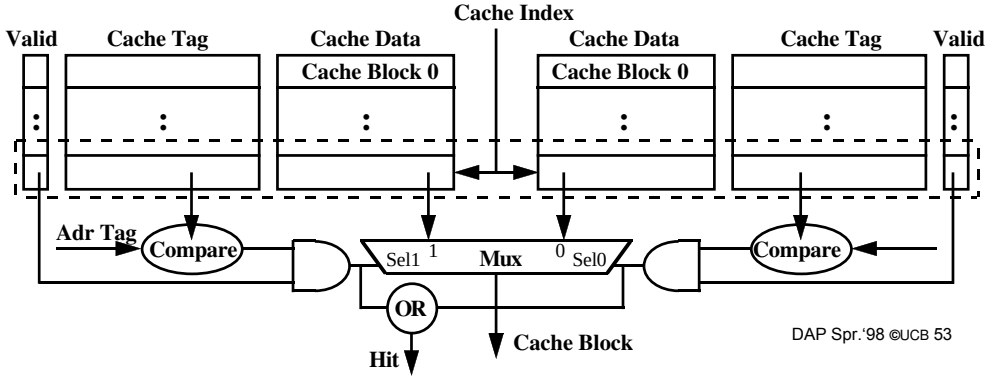
Two-way Set Associative Cache

- N-way set associative: N entries for each Cache Index
  - N direct mapped caches operates in parallel (N typically 2 to 4)
- Example: Two-way set associative cache
  - Cache Index selects a "set" from the cache
  - The two tags in the set are compared in parallel
  - Data is selected based on the tag result



Disadvantage of Set Associative Cache

- N-way Set Associative Cache v. Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.

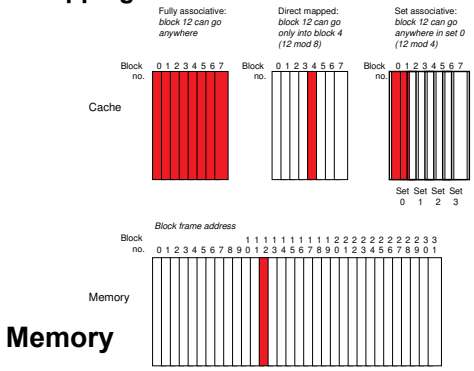


4 Questions for Memory Hierarchy

- 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)*

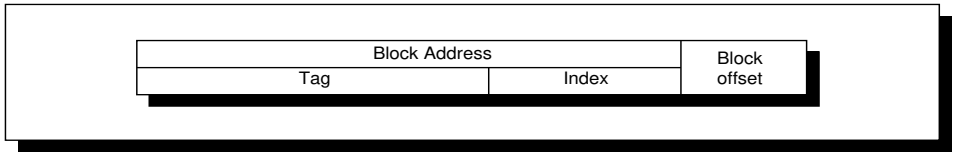
Q1: Where can a block be placed in the upper level?

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



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?

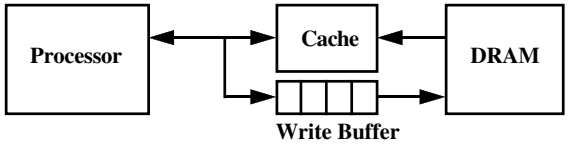
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity:	2-way		4-way		8-way	
Size	LRU	Random	LRU	Random	LRU	Random
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*—The information is written to both the block in the cache and to the block in the lower-level memory.**
- ***Write back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.**
  - is block clean or dirty?
- Pros and Cons of each?
  - WT: read misses cannot result in writes
  - WB: no repeated writes to same location
- WT always combined with write buffers so that don't wait for lower level memory

Write Buffer for Write Through

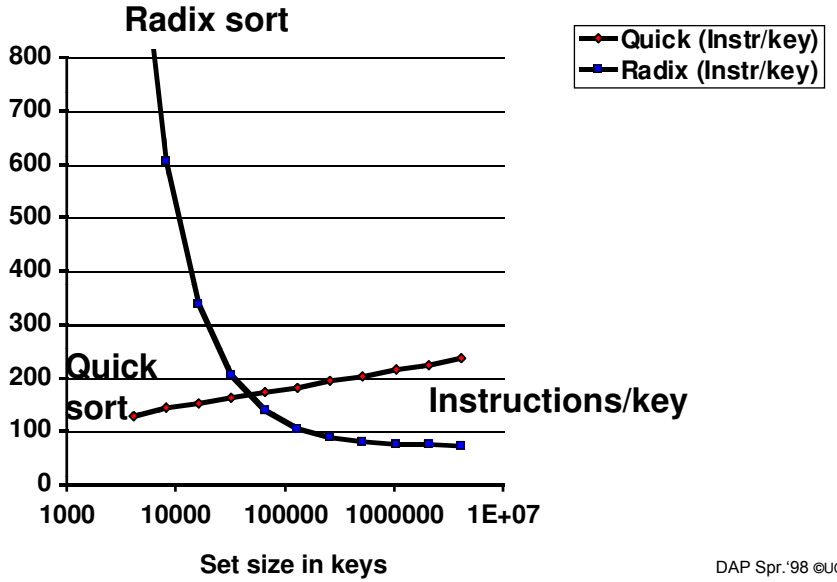


- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare:
  - Store frequency (w.r.t. time) -> 1 / DRAM write cycle
  - Write buffer saturation

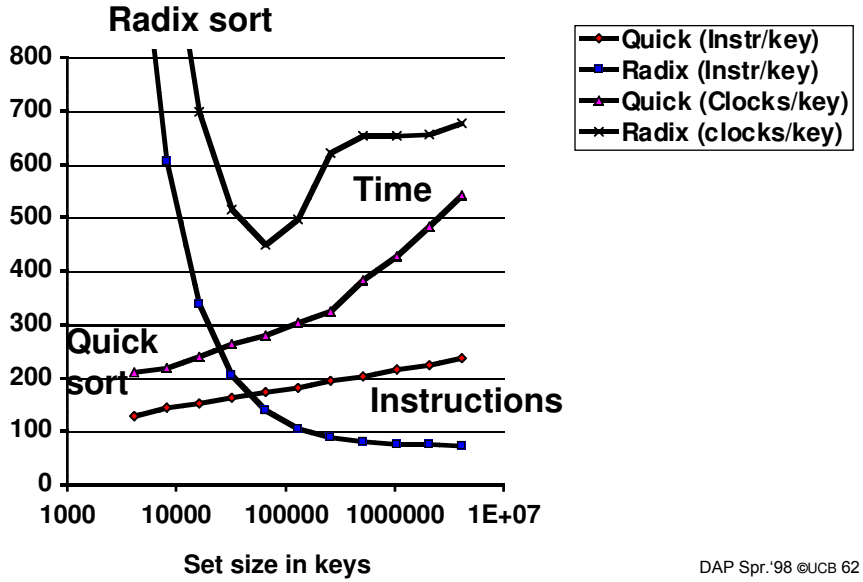
Impact of Memory Hierarchy on Algorithms

- Today CPU time is a function of (ops, cache misses) vs. just f(ops):
  What does this mean to Compilers, Data structures, Algorithms?
- "The Influence of Caches on the Performance of Sorting" by A. LaMarca and R.E. Ladner. *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, January, 1997, 370-379.
- Quicksort: fastest comparison based sorting algorithm when all keys fit in memory
- Radix sort: also called "linear time" sort because for keys of fixed length and fixed radix a constant number of passes over the data is sufficient independent of the number of keys
- For Alphastation 250, 32 byte blocks, direct mapped L2 2MB cache, 8 byte keys, from 4000 to 4000000

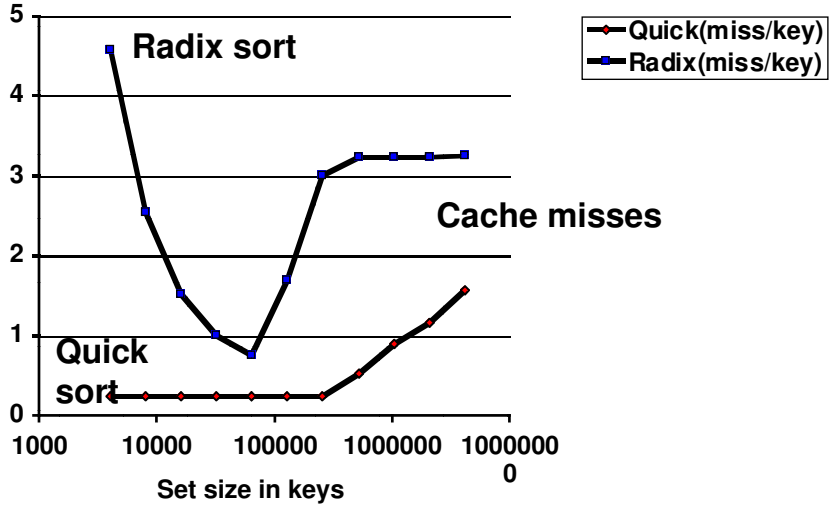
Quicksort vs. Radix as vary number keys: Instructions



Quicksort vs. Radix as vary number keys: Instrs & Time



Quicksort vs. Radix as vary number keys: Cache misses



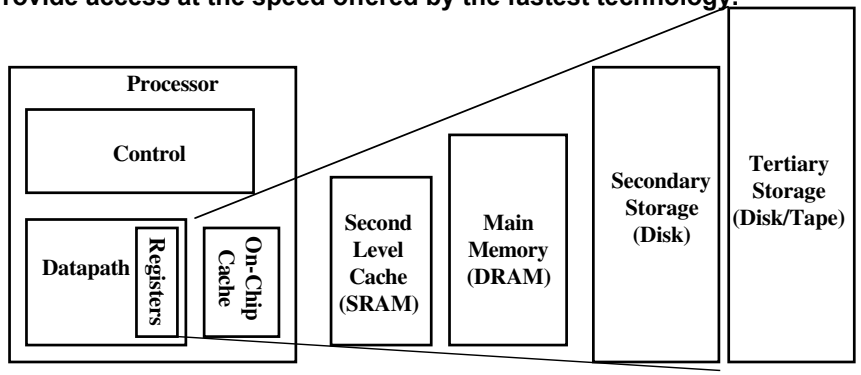
What is proper approach to fast algorithms?

5 minute Class Break

- 80 minutes straight is too long for me to lecture (12:40:00 – 2:00:00):
  - ≈ 1 minute: review last time & motivate this lecture
  - ≈ 20 minute lecture
  - ≈ 3 minutes: discuss class manangement
  - ≈ 25 minutes: lecture
  - 5 minutes: break
  - ≈25 minutes: lecture
  - ≈1 minute: summary of today's important topics

A Modern Memory Hierarchy

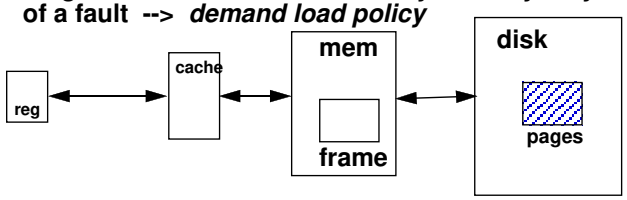
- By taking advantage of the principle of locality:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

Basic Issues in VM System Design

- size of information blocks that are transferred from secondary to main storage (M)
- block of information brought into M, and M is full, then some region of M must be released to make room for the new block --> *replacement policy*
- which region of M is to hold the new block --> *placement policy*
- missing item fetched from secondary memory only on the occurrence of a fault --> *demand load policy*



Paging Organization

virtual and physical address space partitioned into blocks of equal size

page frames

pages

Address Map

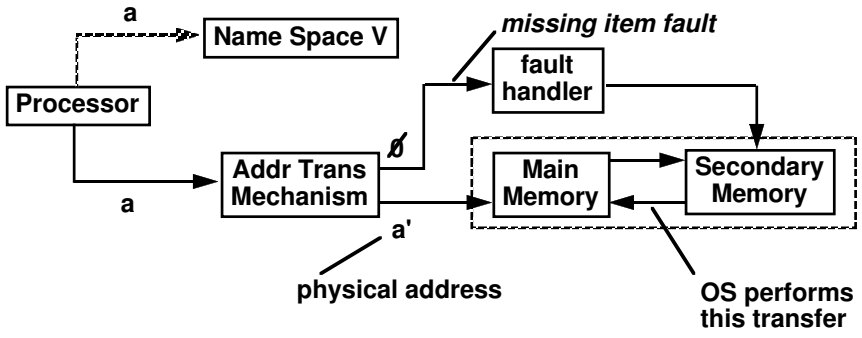
V = {0, 1, . . . , n - 1} virtual address space
M = {0, 1, . . . , m - 1} physical address space

n > m

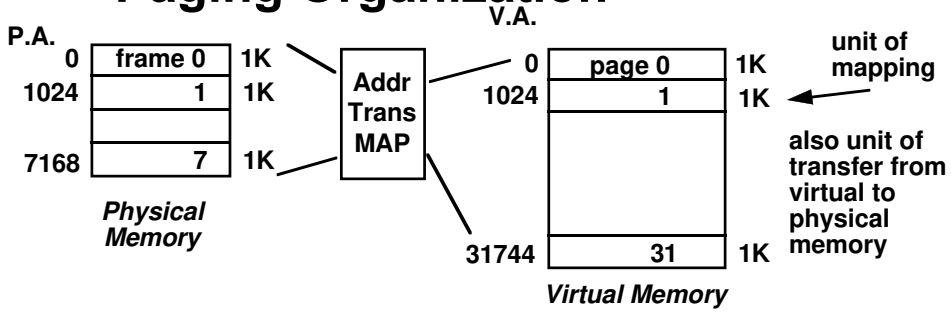
MAP: V --> M U {ø} address mapping function

MAP(a) = a' if data at virtual address a is present in physical address a' and a' in M

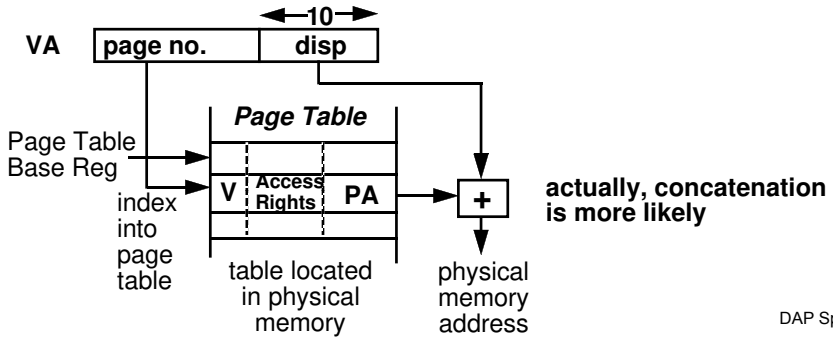
= ø if data at virtual address a is not present in M



Paging Organization

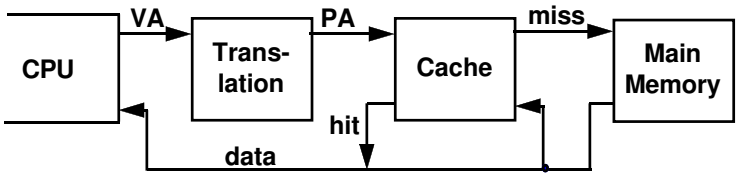


Address Mapping



actually, concatenation is more likely

Virtual Address and a Cache



It takes an extra memory access to translate VA to PA

This makes cache access very expensive, and this is the "innermost loop" that you want to go as fast as possible

ASIDE: Why access cache with PA at all? VA caches have a problem!
synonym / alias problem: two different virtual addresses map to same physical address => two different cache entries holding data for the same physical address!

for update: must update all cache entries with same physical address or memory becomes inconsistent

determining this requires significant hardware, essentially an associative lookup on the physical address tags to see if you have multiple hits; or

software enforced alias boundary: same lsb of VA &PA > cache size

TLBs

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

Really just a cache on the page table mappings

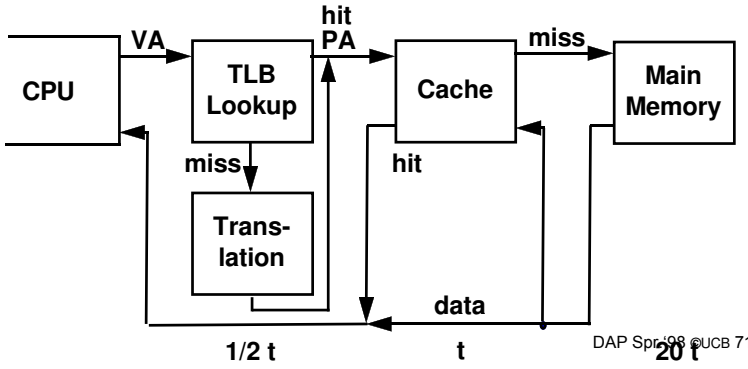
TLB access time comparable to cache access time (much less than main memory access time)

Translation Look-Aside Buffers

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

Translation with a TLB



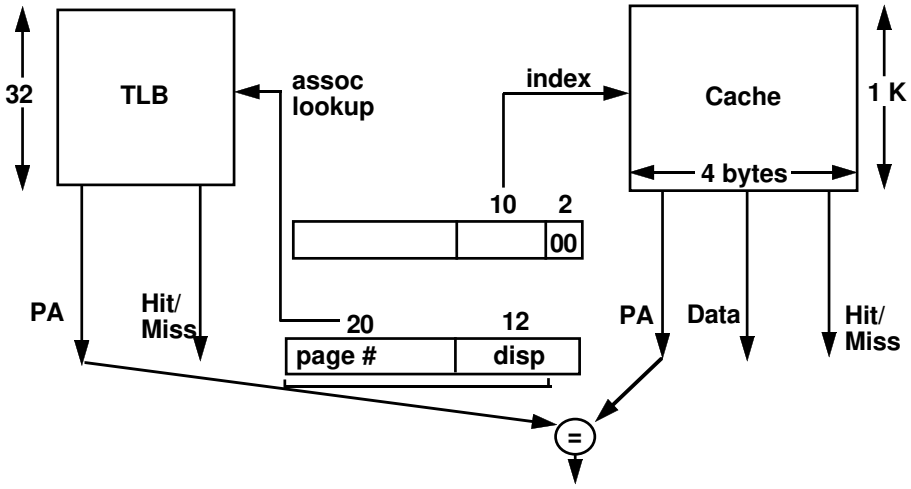
Reducing Translation Time

Machines with TLBs go one step further to reduce # cycles/cache access

They overlap the cache access with the TLB access:

high order bits of the VA are used to look in the TLB while low order bits are used as index into cache

Overlapped Cache & TLB Access



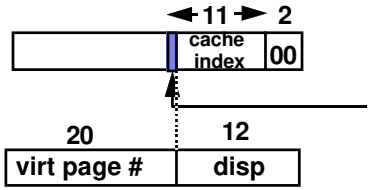
IF cache hit AND (cache tag = PA) then deliver data to CPU
ELSE IF [cache miss OR (cache tag = PA)] and TLB hit THEN
access memory with the PA from the TLB
ELSE do standard VA translation

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

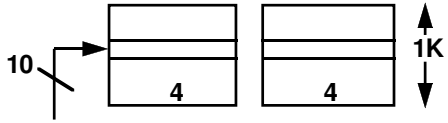
This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache

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]



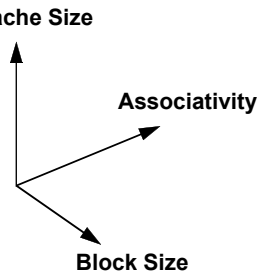
2 way set assoc cache

Summary #1/4:

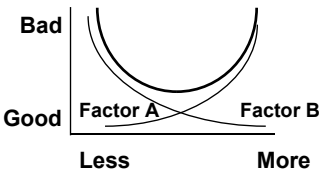
- **The Principle of Locality:**
  - Program access a relatively small portion of the address space at any instant of time.
    - » Temporal Locality: Locality in Time
    - » Spatial Locality: Locality in Space
- **Three Major Categories of Cache Misses:**
  - Compulsory Misses: sad facts of life. Example: 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: needs a write buffer. Nightmare: WB saturation
  - Write Back: control can be complex

Summary #2 / 4: 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 #3/4: TLB, Virtual Memory

- 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 repalced on miss? 4) How are writes handled?
- 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!

Summary #4/4: Memory Hierachy

- VIrtual memory was controversial at the time: can SW automatically manage 64KB across many programs?
  - 1000X DRAM growth removed the controversy
- 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
- Today CPU time is a function of (ops, cache misses) vs. just f(ops): What does this mean to Compilers, Data structures, Algorithms?