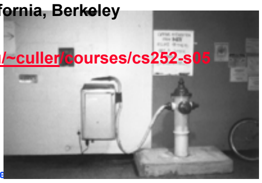
CS252
Graduate Computer Architecture
Lecture 2

Review of Instruction Sets, Pipelines, and Caches

Prof. David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler/courses/cs252-s05



Review, #1

- Technology is changing rapidly:

	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

- What was true five years ago is not necessarily true now.
- Execution time is the REAL measure of computer performance!
  - Not clock rate, not CPI
- "X is n times faster than Y" means:

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

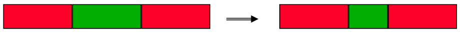
Amdahl's Law

$$\text{ExTime}_{new} = \text{ExTime}_{old} \times \left[(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}} \right]$$

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

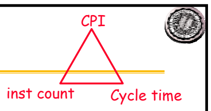
Best you could ever hope to do:

$$\text{Speedup}_{maximum} = \frac{1}{(1 - \text{Fraction}_{enhanced})}$$



Today:
Quick review of everything you should have learned

Computer Performance



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

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

Cycles Per Instruction (Throughput)

"Average Cycles per Instruction"

CPI = (CPU Time * Clock Rate) / Instruction Count
= Cycles / Instruction Count

$$\text{CPU time} = \text{Cycle Time} \times \sum_{j=1}^{n} \text{CPI}_j \times \text{I}_j$$

$$\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times \text{F}_j \qquad \text{where } \text{F}_j = \frac{\text{I}_j}{\text{Instruction Count}}$$

"Instruction Frequency"

Example: Calculating CPI bottom up

Run benchmark and collect workload characterization (simulate, machine counters, or sampling)

Base Machine (Reg / Reg)

Op	Freq	Cycles	CPI(i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix of instruction types in program

Design guideline: Make the common case fast

MIPS 1% rule: only consider adding an instruction of it is shown to add 1% performance improvement on reasonable benchmarks.

Example: Branch Stall Impact

- Assume CPI = 1.0 ignoring branches (ideal)
- Assume solution was stalling for 3 cycles
- If 30% branch, Stall 3 cycles on 30%

Op	Freq	Cycles	CPI(i)	(% Time)
Other	70%	1	.7	(37%)
Branch	30%	4	1.2	(63%)

⇒ new CPI = 1.9

- New machine is 1/1.9 = 0.52 times faster (i.e. slow!)

SPEC: System Performance Evaluation Cooperative

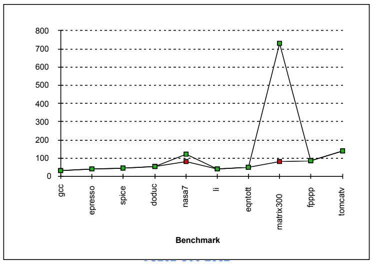
- First Round 1989
  - 10 programs yielding a single number ("SPECmarks")
- Second Round 1992
  - SPECInt92 (6 integer programs) and SPECfp92 (14 floating point programs)
    - » Compiler Flags unlimited. March 93 of DEC 4000 Model 610:

```
spice: unix.c:/def=(sysv,has_bcopy,"bcopy(a,b,c)=
memcpy(b,a,c)"
wave5: /ali=(all,dcom=nat)/ag=a/ur=4/ur=200
nasa7: /norecu/ag=a/ur=4/ur2=200/lc=blas
```

- Third Round 1995
  - new set of programs: SPECint95 (8 integer programs) and SPECfp95 (10 floating point)
  - "benchmarks useful for 3 years"
  - Single flag setting for all programs: SPECint_base95, SPECfp_base95
- Fourth Round 2000: 26 apps
  - analysis and simulation programs
  - Compression: bzip2, gzip,
  - Integrated circuit layout, ray tracing, lots of others

SPEC First Round

- One program: 99% of time in single line of code
- New front-end compiler could improve dramatically

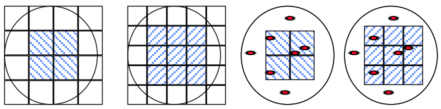


Integrated Circuits Costs

$$\text{IC cost} = \frac{\text{Die cost} + \text{Testing cost} + \text{Packaging cost}}{\text{Final test yield}}$$

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi\ (\text{Wafer\_diam}/2)^2}{\text{Die\_Area}} - \frac{\pi \times \text{Wafer\_diam}}{\sqrt{2 \cdot \text{Die\_Area}}} - \text{Test\_Die}$$



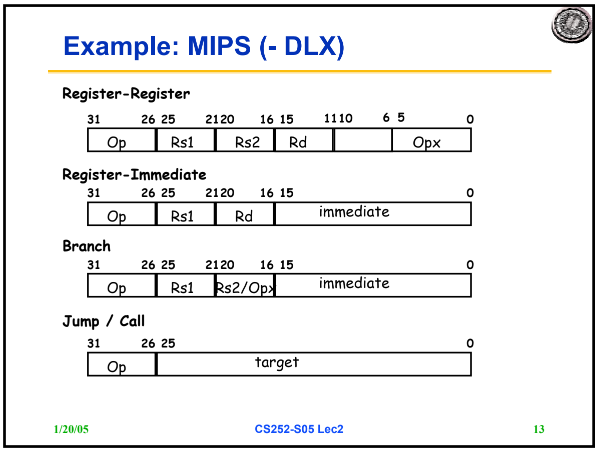
$$\text{Die Yield} = \text{Wafer\_yield} \times \left\{1 + \left(\frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha}\right)\right\}^{-\alpha}$$

Die Cost goes roughly with die area^4

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

31 26	25 21	20 16	15 11	10 6	5 0
Op	Rs1	Rs2	Rd		Opx

Register-Immediate

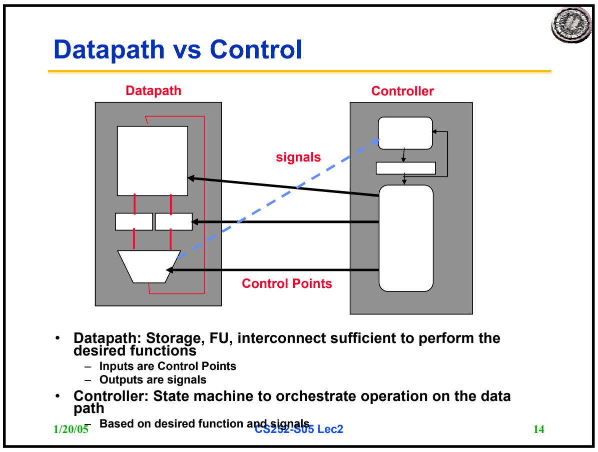
31 26	25 21	20 16	15 0
Op	Rs1	Rd	immediate

Branch

31 26	25 21	20 16	15 0
Op	Rs1	Rs2/Opx	immediate

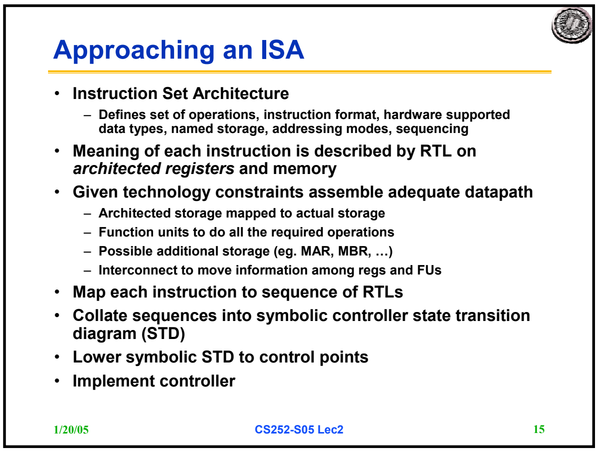
Jump / Call

31 26	25 0
Op	target



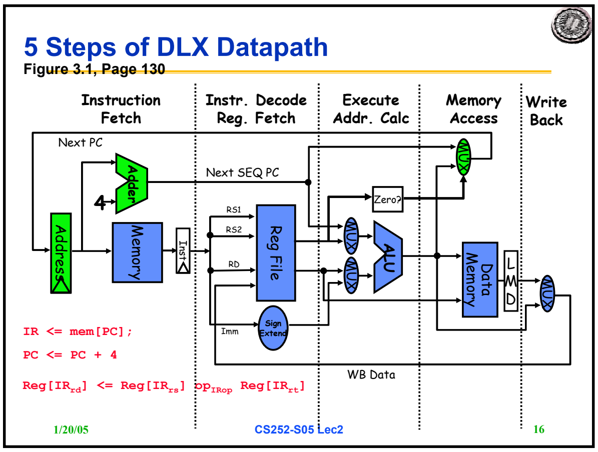
Datapath vs Control

- Datapath: Storage, FU, interconnect sufficient to perform the desired functions
  - Inputs are Control Points
  - Outputs are signals
- Controller: State machine to orchestrate operation on the data path
  - Based on desired function and signals



Approaching an ISA

- Instruction Set Architecture
  - Defines set of operations, instruction format, hardware supported data types, named storage, addressing modes, sequencing
- Meaning of each instruction is described by RTL on *architected registers* and memory
- Given technology constraints assemble adequate datapath
  - Architected storage mapped to actual storage
  - Function units to do all the required operations
  - Possible additional storage (eg. MAR, MBR, …)
  - Interconnect to move information among regs and FUs
- Map each instruction to sequence of RTLs
- Collate sequences into symbolic controller state transition diagram (STD)
- Lower symbolic STD to control points
- Implement controller



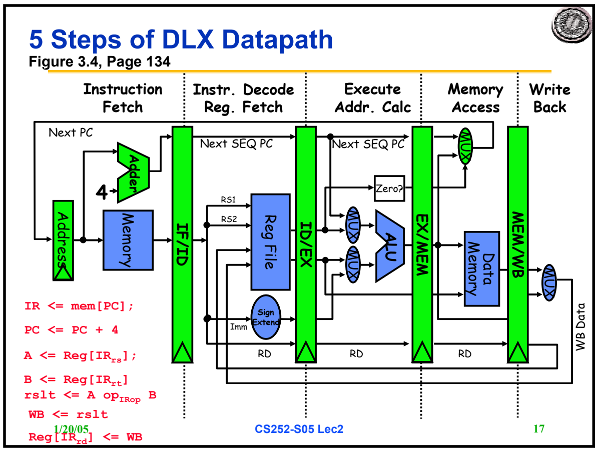
5 Steps of DLX Datapath

Figure 3.1, Page 130

IR <= mem[PC];

PC <= PC + 4

Reg[IR_{rd}] <= Reg[IR_{rs}] op_{IRop} Reg[IR_{rt}]



5 Steps of DLX Datapath

Figure 3.4, Page 134

IR <= mem[PC];

PC <= PC + 4

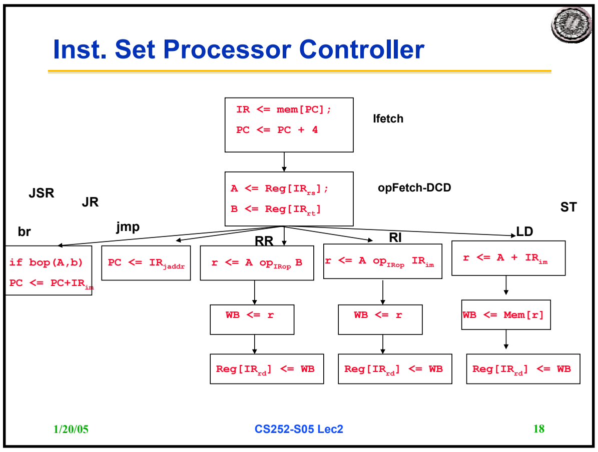
A <= Reg[IR_{rs}];

B <= Reg[IR_{rt}]

rslt <= A op_{IRop} B

WB <= rslt

Reg[IR_{rd}] <= WB



Inst. Set Processor Controller

IR <= mem[PC];
PC <= PC + 4

A <= Reg[IR_{rs}];
B <= Reg[IR_{rt}]

if bop(A,b)
PC <= PC+IR_{im}

PC <= IR_{jaddr}

r <= A op_{IRop} B

WB <= r

Reg[IR_{rd}] <= WB

r <= A op_{IRop} IR_{im}

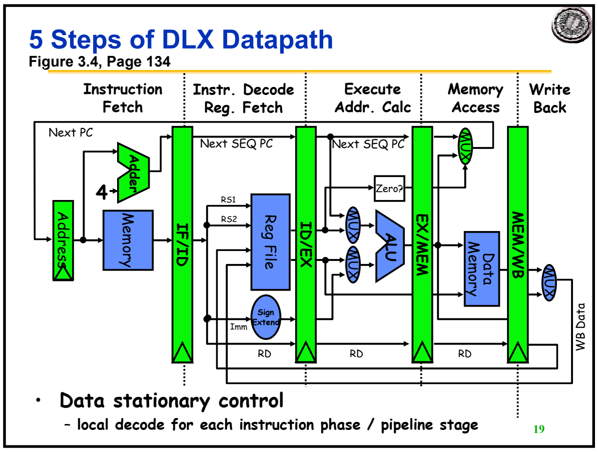
WB <= r

Reg[IR_{rd}] <= WB

r <= A + IR_{im}

WB <= Mem[r]

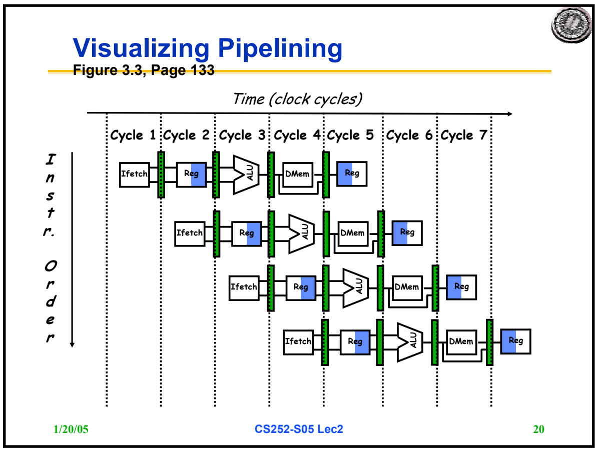
Reg[IR_{rd}] <= WB



5 Steps of DLX Datapath

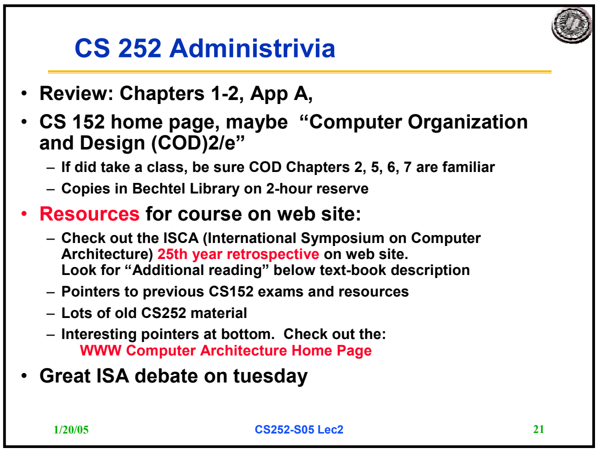
Figure 3.4, Page 134

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



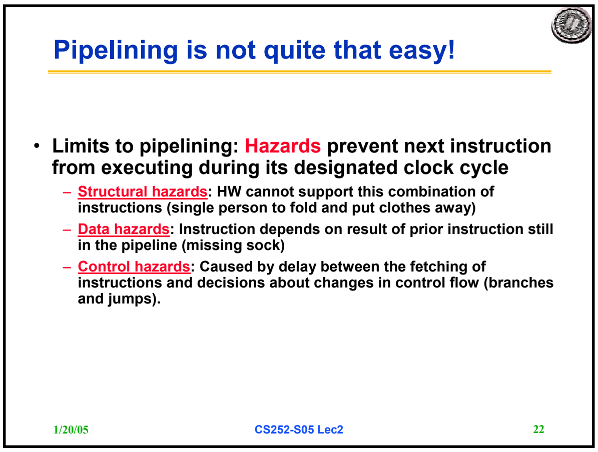
Visualizing Pipelining

Figure 3.3, Page 133



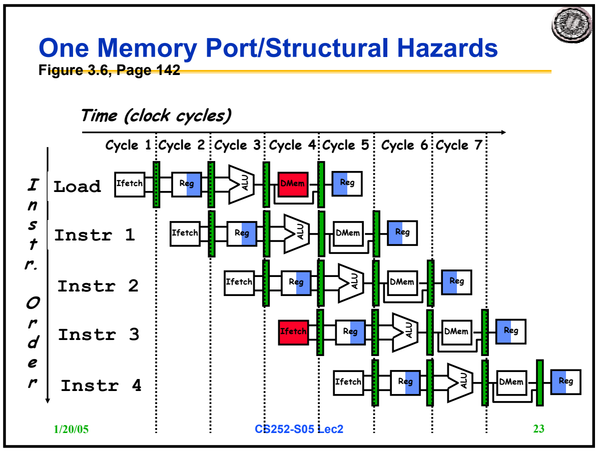
CS 252 Administrivia

- Review: Chapters 1-2, App A,
- CS 152 home page, maybe “Computer Organization and Design (COD)2/e”
  - If did take a class, be sure COD Chapters 2, 5, 6, 7 are familiar
  - Copies in Bechtel Library on 2-hour reserve
- Resources for course on web site:
  - Check out the ISCA (International Symposium on Computer Architecture) 25th year retrospective on web site. Look for “Additional reading” below text-book description
  - Pointers to previous CS152 exams and resources
  - Lots of old CS252 material
  - Interesting pointers at bottom. Check out the: WWW Computer Architecture Home Page
- Great ISA debate on tuesday



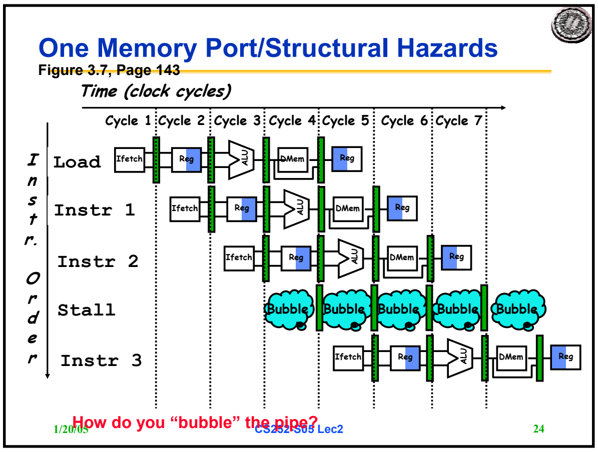
Pipelining is not quite that easy!

- 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: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).



One Memory Port/Structural Hazards

Figure 3.6, Page 142



One Memory Port/Structural Hazards

Figure 3.7, Page 143

How do you “bubble” the pipe?

Speed Up Equation for Pipelining

$$\text{CPI}_{\text{pipelined}} = \text{Ideal CPI} + \text{Average Stall cycles per Inst}$$

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

For simple RISC pipeline, CPI = 1:

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

Example: Dual-port vs. Single-port

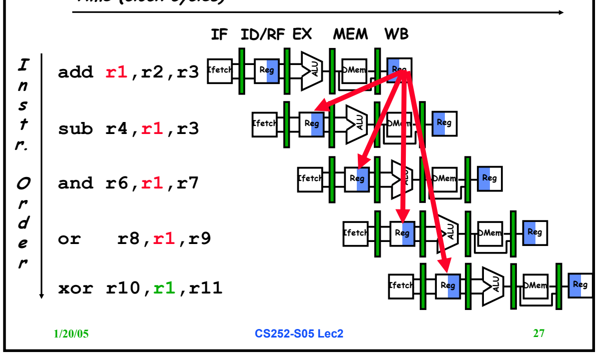
- Machine A: Dual ported memory ("Harvard Architecture")
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

SpeedUp_A = Pipeline Depth/(1 + 0) x ($\text{clock}_{\text{unpipe}}/\text{clock}_{\text{pipe}}$)
= Pipeline Depth
SpeedUp_B = Pipeline Depth/(1 + 0.4 x 1) x ($\text{clock}_{\text{unpipe}}$/($\text{clock}_{\text{unpipe}}$ / 1.05)
= (Pipeline Depth/1.4) x 1.05
= 0.75 x 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



Time (clock cycles)

IF ID/RF EX MEM WB

Instr. Order

```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
or   r8,r1,r9
xor r10,r1,r11
```

Three Generic Data Hazards

- Read After Write (RAW)
  Instr$_J$ tries to read operand before Instr$_I$ writes it

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Three Generic Data Hazards

- Write After Read (WAR)
  Instr$_J$ writes operand *before* Instr$_I$ reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1".
- 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

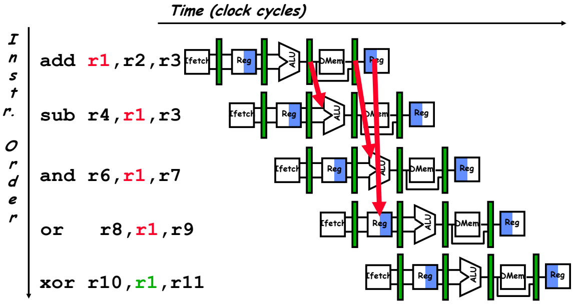
- Write After Write (WAW)
  Instr$_J$ writes operand *before* Instr$_I$ writes it.

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "output dependence" by compiler writers This also results from the reuse of name "r1".
- 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 more complicated pipes

Forwarding to Avoid Data Hazard

Figure 3.10, Page 149



add r1,r2,r3

sub r4,r1,r3

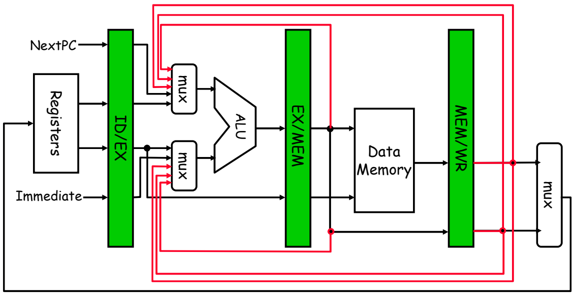
and r6,r1,r7

or r8,r1,r9

xor r10,r1,r11

HW Change for Forwarding

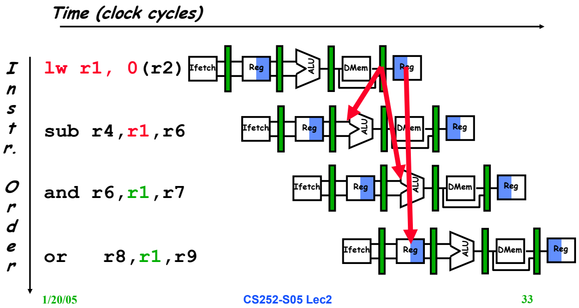
Figure 3.20, Page 161



What circuit detects and resolves this hazard?

Data Hazard Even with Forwarding

Figure 3.12, Page 153



lw r1, 0(r2)

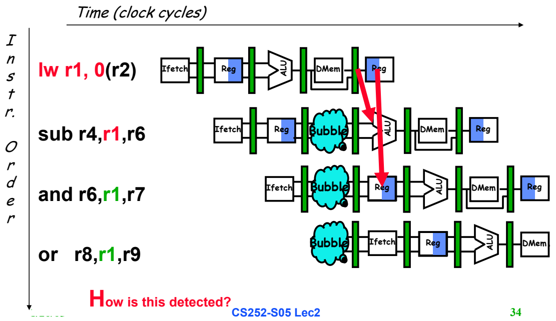
sub r4,r1,r6

and r6,r1,r7

or r8,r1,r9

Data Hazard Even with Forwarding

Figure 3.13, Page 154



lw r1, 0(r2)

sub r4,r1,r6

and r6,r1,r7

or r8,r1,r9

How is this detected?

Software Scheduling to Avoid Load Hazards

Try producing fast code for

a = b + c;

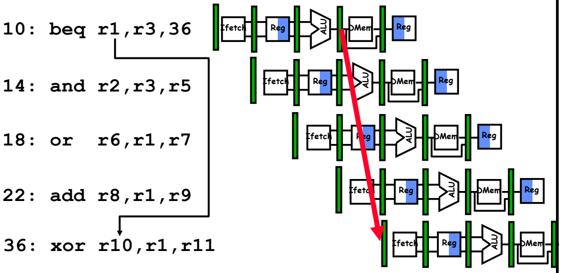
d = e – f;

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

Compiler optimizes for performance. Hardware checks for safety.

Control Hazard on Branches
Three Stage Stall



10: beq r1,r3,36

14: and r2,r3,r5

18: or r6,r1,r7

22: add r8,r1,r9

36: xor r10,r1,r11

What do you do with the 3 instructions in between?

How do you do it?

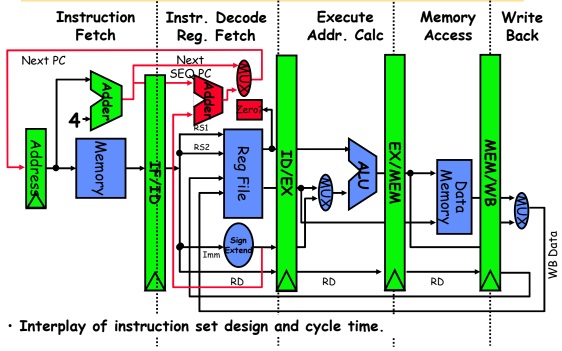
Where is the “commit”?

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



- Interplay of instruction set design and cycle time.

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   \
  ........                 > Branch delay of length n
  sequential successorn   /
branch target if taken
```

- 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
  - Canceling 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

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

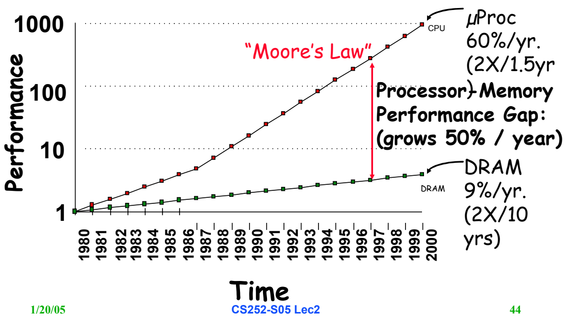
Scheduling scheme	*Branch penalty*	*CPI*	*speedup v. unpipelined*	*speedup v. stall*
Stall pipeline	3	1.42	3.5	1.0
Predict taken	1	1.14	4.4	1.26
Predict not taken	1	1.09	4.5	1.29
Delayed branch	0.5	1.07	4.6	1.31

Conditional & Unconditional = 14%, 65% change PC

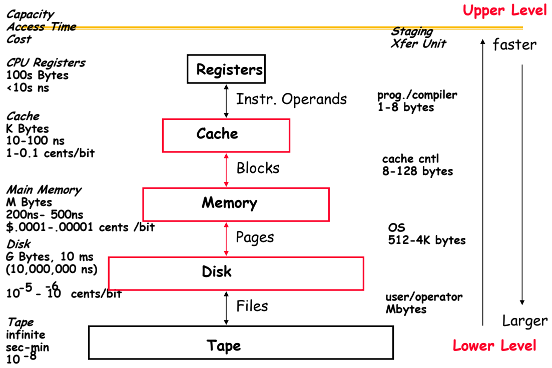
Now, Review of Memory Hierarchy

Recap: Who Cares About the Memory Hierarchy?

Processor-DRAM Memory Gap (latency)



Levels of the Memory Hierarchy



Upper Level

Capacity
Access Time
Cost

Staging
Xfer Unit

faster

CPU Registers
100s Bytes
<10s ns

Registers

Instr. Operands

prog./compiler
1-8 bytes

Cache
K Bytes
10-100 ns
1-0.1 cents/bit

Cache

Blocks

cache cntl
8-128 bytes

Main Memory
M Bytes
200ns- 500ns
$.0001-.00001 cents /bit

Memory

Pages

OS
512-4K bytes

Disk
G Bytes, 10 ms
(10,000,000 ns)

10^{-5} - 10^{-6} cents/bit

Disk

Files

user/operator
Mbytes

Tape
infinite
sec-min
10^{-8}

Tape

Larger

Lower Level

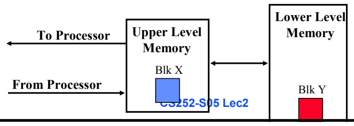
The Principle of Locality

- 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.

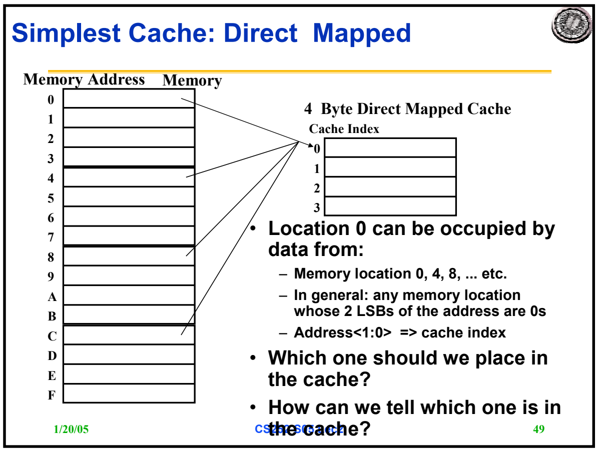
Memory Hierarchy: Terminology

- 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!)



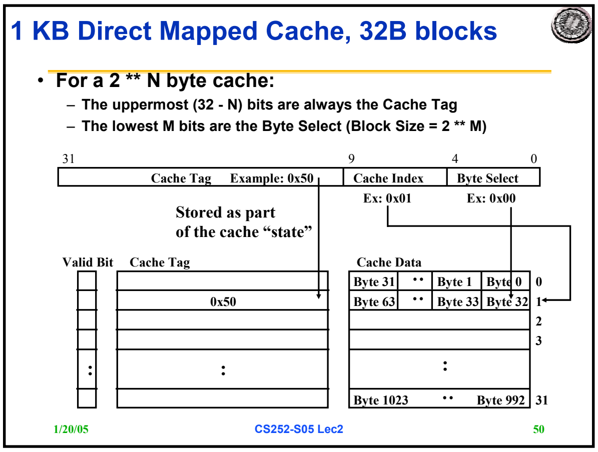
Cache Measures

- *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)



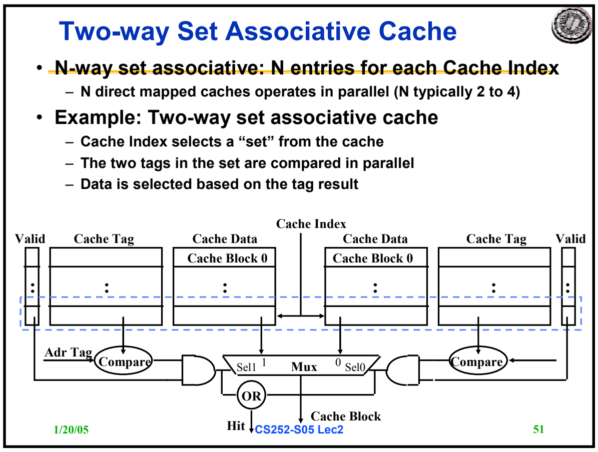
Simplest Cache: Direct Mapped

- 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?



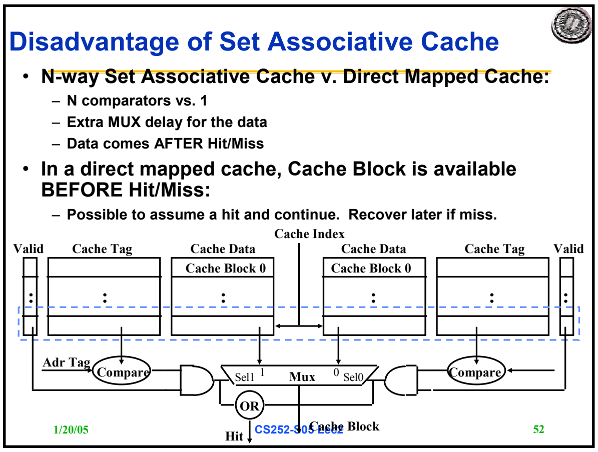
1 KB Direct Mapped Cache, 32B blocks

- 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)



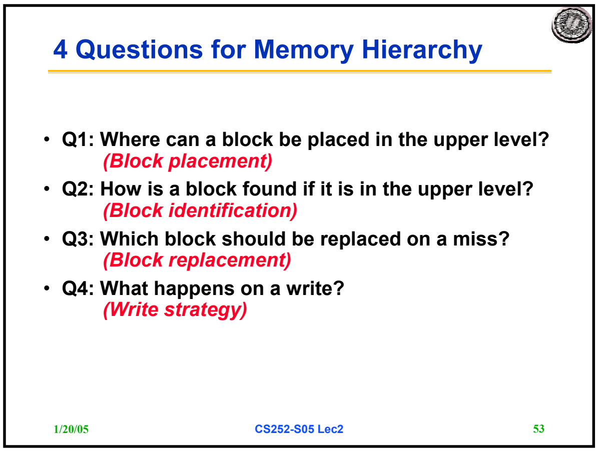
Two-way Set Associative Cache

- 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



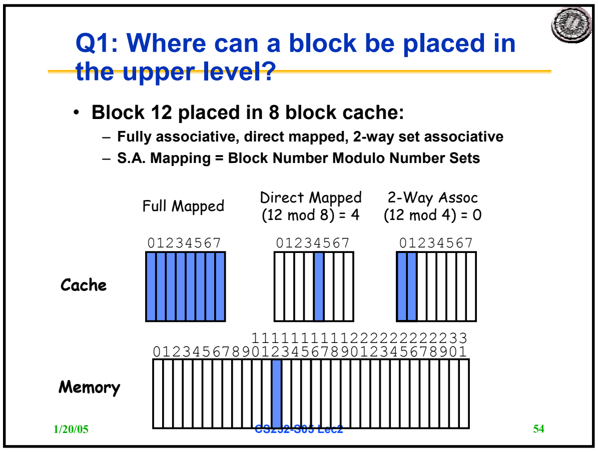
Disadvantage of Set Associative Cache

- 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.



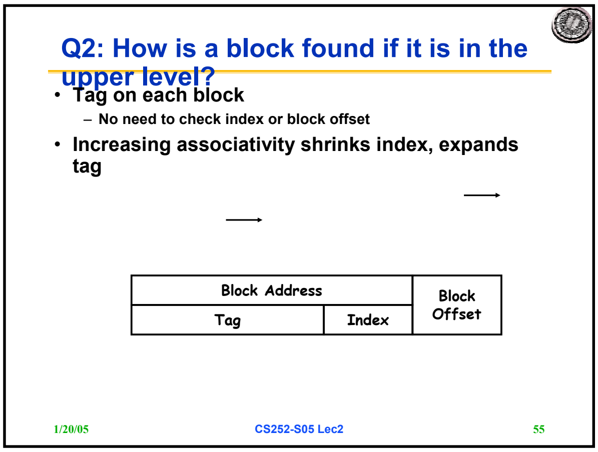
4 Questions for Memory Hierarchy

- 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?

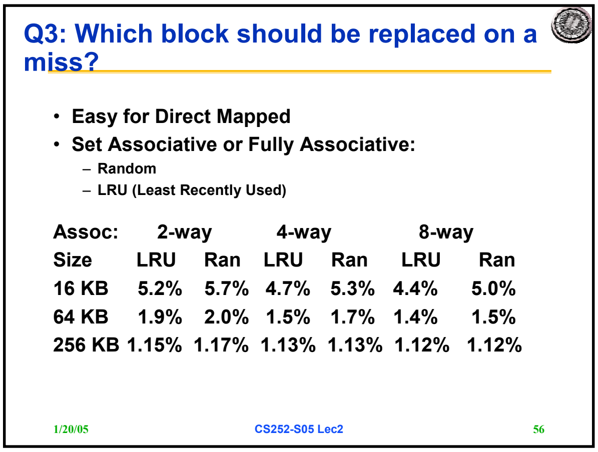
- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets



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

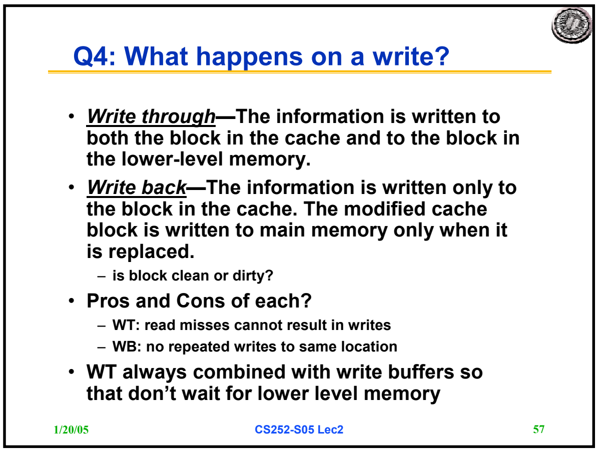
Block Address		Block Offset
Tag	Index	



Q3: Which block should be replaced on a miss?

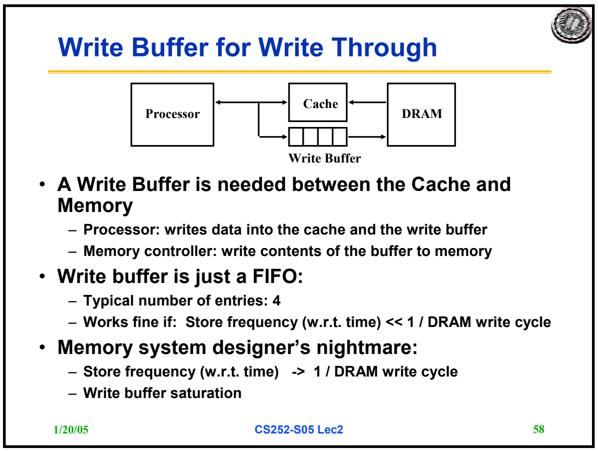
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - – Random
  - – LRU (Least Recently Used)

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?

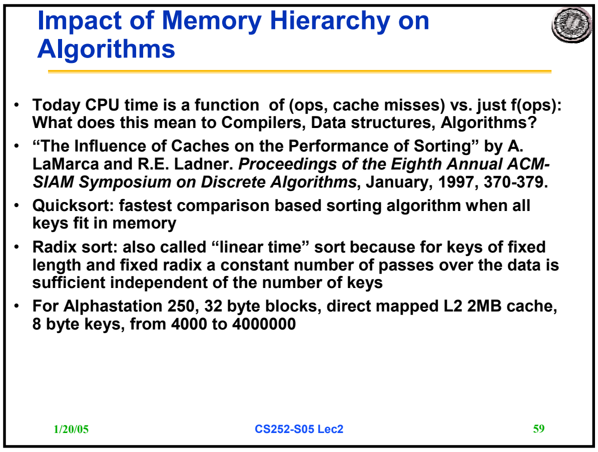
- ***Write through***—The information is written to both the block in the cache and to the block in the lower-level memory.
- ***Write back***—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - – is block clean or 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

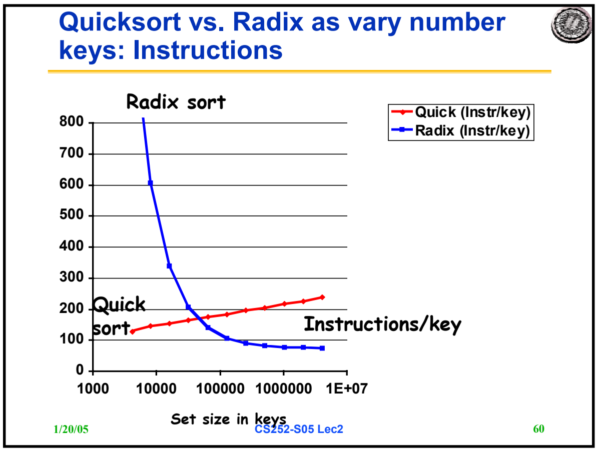
Processor
Cache
DRAM
Write Buffer

- 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



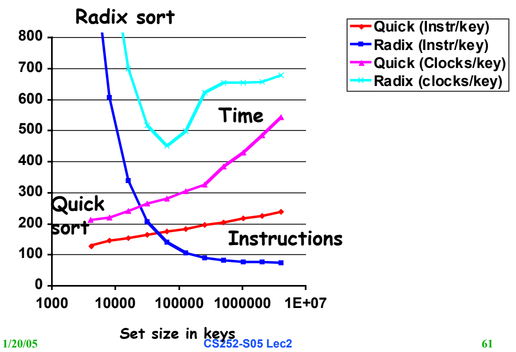
Impact of Memory Hierarchy on Algorithms

- 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

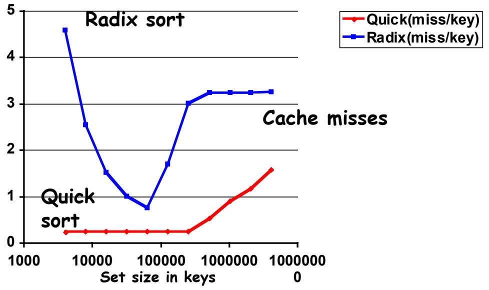


Quicksort vs. Radix as vary number keys: Instructions

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



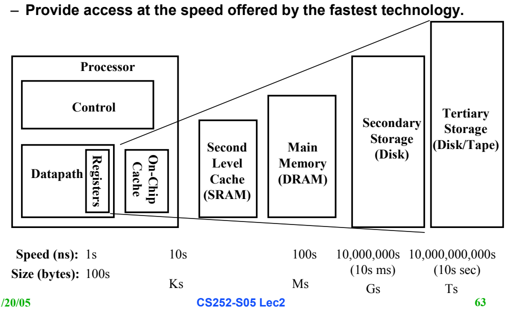
Quicksort vs. Radix as vary number keys: Cache misses



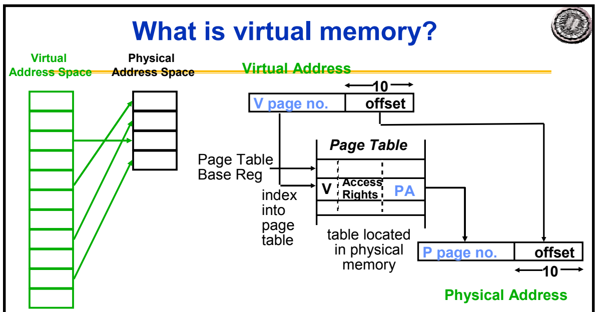
What is proper approach to fast algorithms?

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.



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

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 => Far more "viruses" under Microsoft Windows
- Sharing:
  - Can map same physical page to multiple users ("Shared memory")

Issues in Virtual Memory System Design

What is the size of information blocks that are transferred from secondary to main storage (M)? ⇒ *page size*
(Contrast with physical block size on disk, I.e. *sector size*)

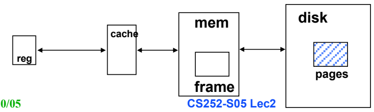
Which region of M is to hold the new block ⇒ *placement policy*

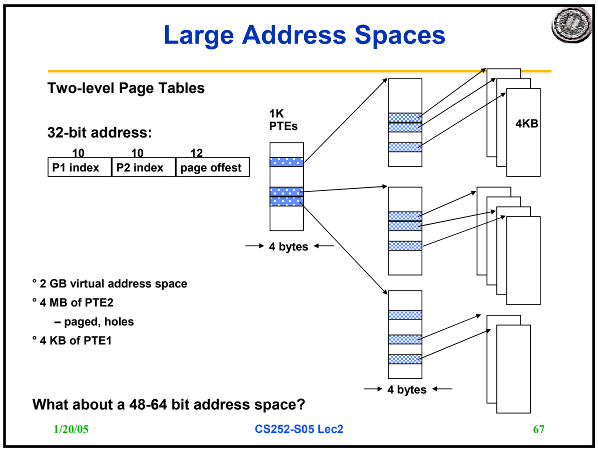
How do we find a page when we look for it? ⇒ *block identification*

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*

What do we do on a write? ⇒ *write policy*

Missing item fetched from secondary memory only on the occurrence of a fault ⇒ *demand load policy*





Large Address Spaces

Two-level Page Tables

32-bit address:

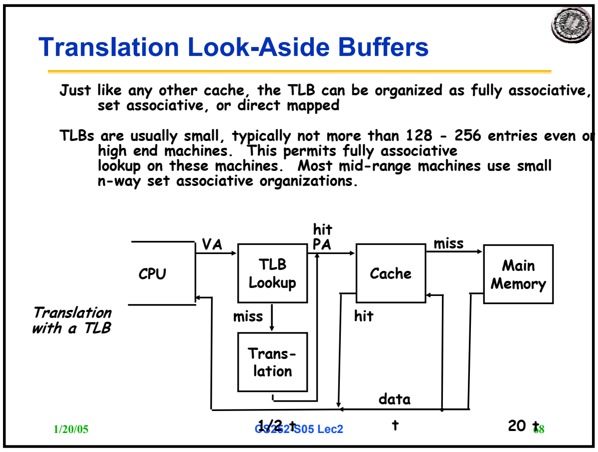
° 2 GB virtual address space

° 4 MB of PTE2

– paged, holes

° 4 KB of PTE1

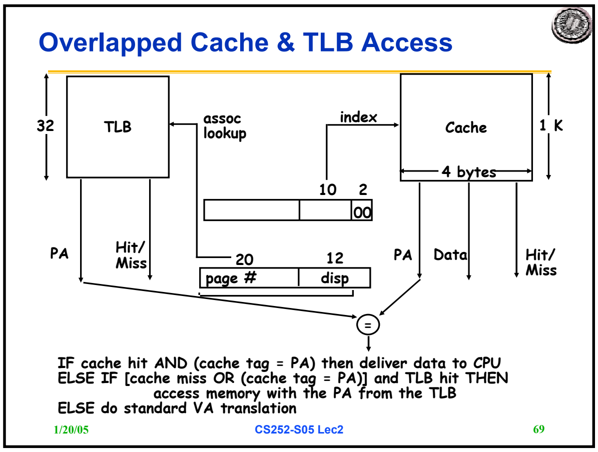
What about a 48-64 bit address space?



Translation Look-Aside Buffers

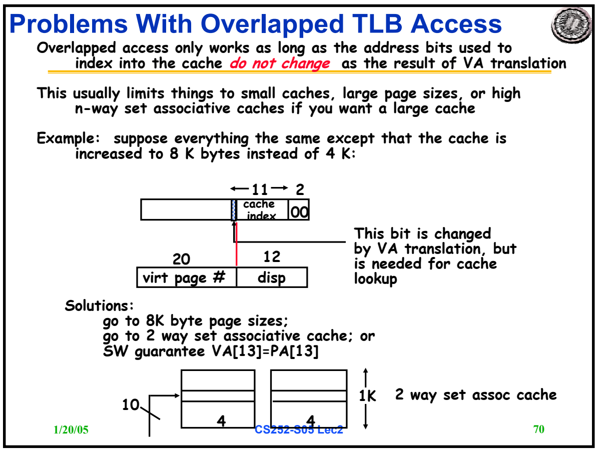
Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

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



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

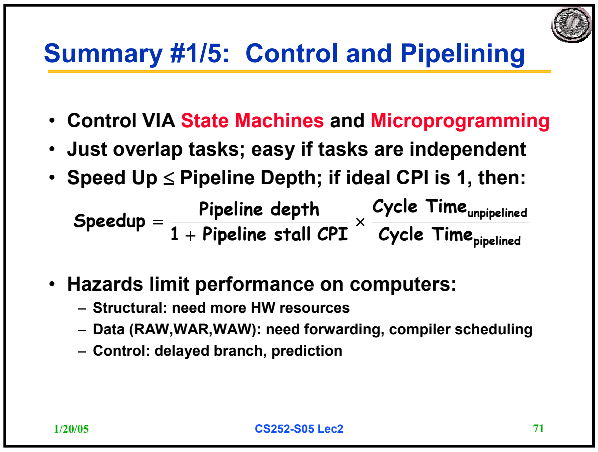
Example: suppose everything the same except that the cache is increased to 8 K bytes instead of 4 K:

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

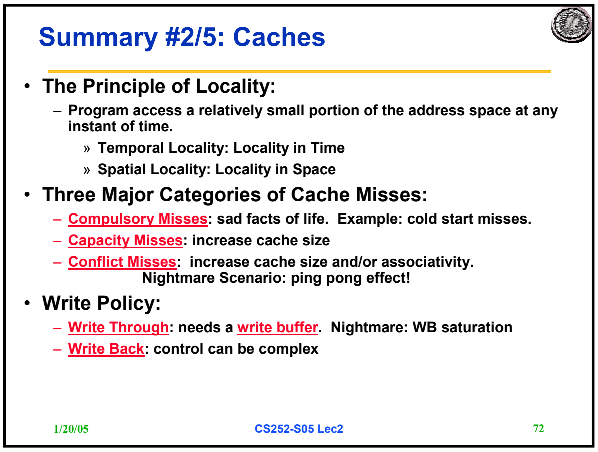


Summary #1/5: Control and Pipelining

- 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

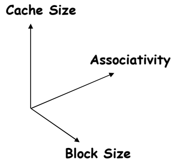


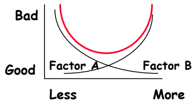
Summary #2/5: Caches

- 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: needs a write buffer. Nightmare: WB saturation
  - Write Back: control can be complex

Summary #3/5: 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 #4/5: 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 #5/5: 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?