CDA 5106 Advanced Computer Architecture 1

Final Exam | Course Review

Chapter 1 Review Topics

- Instruction Set Architecture
- Bandwidth vs. Latency
- MTTF Concept, calculating combined MTTF
- Benchmarking
- Amdahl's Law, Speedup calculation
- Processor Performance Equation, what is CPI

2

Instruction Set Architecture

"... 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, Blaauw, and Brooks, 1964

- Organization of Programmable Storage
- Data Types & Data Structures: Encodings & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions

3

Tracking Technology Performance Trends

- Compare Bandwidth vs. Latency improvements in performance over time
- Bandwidth: number of events per unit time
  - E.g., Mbits/sec over network, MB/sec from disk
- Latency: elapsed time for a single event
  - E.g., one-way network delay in microseconds, average disk access time in milliseconds

4

Calculating MTTF and Reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

$$\begin{aligned} FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\ &= 10 + 2 + 5/1{,}000{,}000 \\ &= 17/1{,}000{,}000 \\ &= 17{,}000\,FIT \\ MTTF &= 1{,}000{,}000{,}000/17{,}000 \\ &\approx 59{,}000\,hours \end{aligned}$$

5

Measuring Performance

- Benchmarking various components of a computer
- Summarizing performance as a score

$$n = \frac{Performance_X}{Performance_Y}$$

- Desktop Benchmarks
  - FLOPS, MIPS, Graphics subsystem
  - SPEC 2006
- Server Benchmarks
  - Transactions, throughput
  - TPC

6

Amdahl's Law Example

- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

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

$$= \frac{1}{(1 - 0.4) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster

7

Processor Performance Equation

$$CPUtime = \frac{CPUClockCycles}{ClockRate}$$

$$CPI = \frac{CPUClockCycles}{IC}$$

$$CPUClockCycles = \sum_{i=1}^{n} IC_i \times CPI_i$$

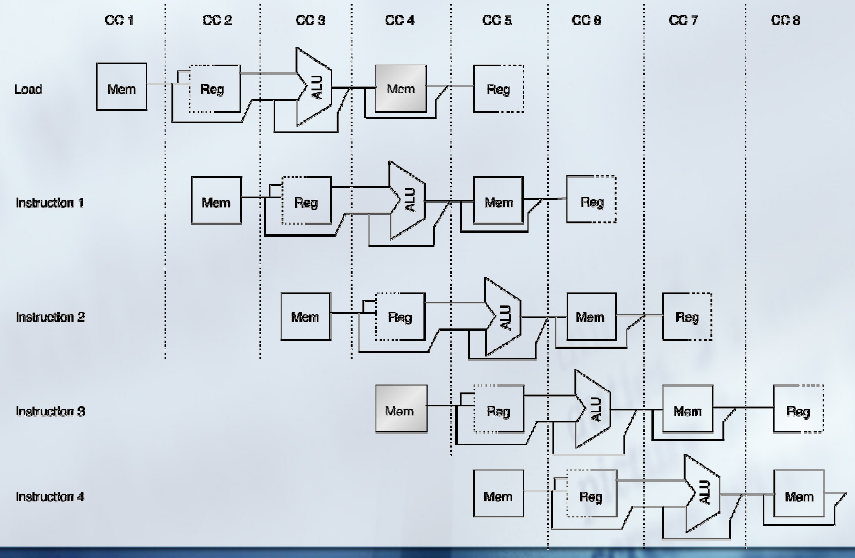
8

Appendix A Review Topics

- MIPS Integer Pipeline
- Pipeline Stages, what happens at each stage
- Data forwarding
- Pipeline hazards
- Data hazards
- Know basic MIPS instructions
- Branches, branch delay slot
- Instruction Latency
- Floating point pipeline

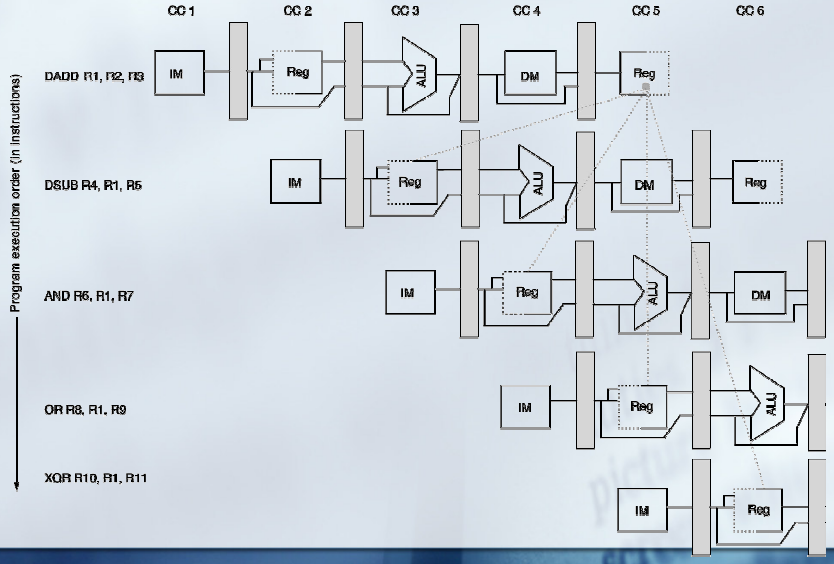
9

Structural hazard example



10

Data hazard examples



11

Stalled pipeline

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	EX	MEM	WB			
AND	R6,R1,R7			IF	ID	EX	MEM	WB		
OR	R8,R1,R9				IF	ID	EX	MEM	WB	

LD	R1,0(R2)	IF	ID	EX	MEM	WB				
DSUB	R4,R1,R5		IF	ID	stall	EX	MEM	WB		
AND	R6,R1,R7			IF	stall	ID	EX	MEM	WB	
OR	R8,R1,R9				stall	IF	ID	EX	MEM	WB

12

Four Branch Hazard Alternatives

#4: Delayed Branch

- Define branch to take place AFTER a following instruction

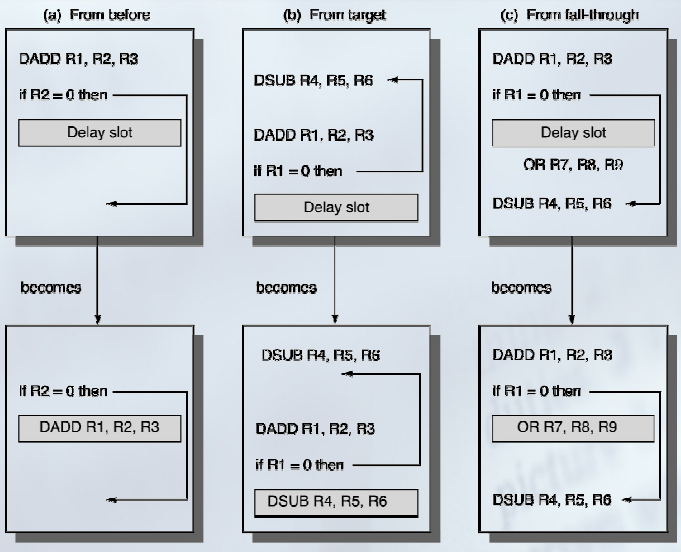
```
branch instruction
 sequential successor1
 sequential successor2
 ........
 sequential successorn
branch target if taken
```

Branch delay of length *n*

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS has one delay slot

13

Branch Delay Slot Scheduling



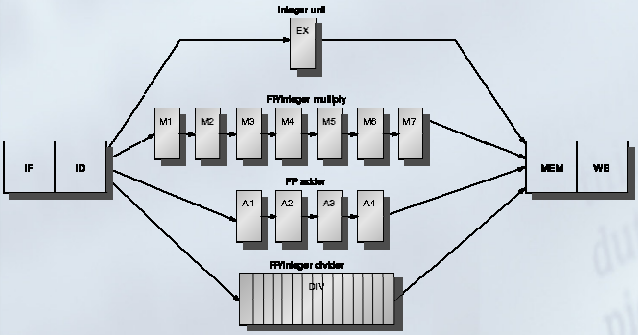
14

FP Pipeline Stages

Functional Unit	Latency	Initiation Interval
Integer ALU	0	1
Data Memory	1	1
FP Add	3	1
FP/Int Multiply	6	1
FP/Int Divide/Sqrt	24	25

- Latency = time between FU result being produced and when an instruction can use it

  Latency determines number of stalls required if the next instruction needs result for this instruction's EX stage



- Initiation Interval = number of cycles required between issuing 2 of the same type of instruction
- Divider has an interval > 1 since it is not pipelined

We pipeline the FP Adder and FP Multiply units to provide overlap in their execution, but not the FP divider since divisions are fairly rare

15

Chapter 2 Review Topics

- Instruction-level parallelism
- Branch prediction
- Loop unrolling
- Converting MIPS to pseudo-code and vice versa
- Register renaming
- Dynamic Scheduling – what happens in Tomasulo's, ROB, VLIW vs. simple pipeline; instruction lifecycle

16

Unroll Loop Four Times (straightforward way)

```
1  Loop:L.D    F0,0(R1)
3       ADD.D  F4,F0,F2
6       S.D    0(R1),F4      ;drop DSUBUI & BNEZ
7       L.D    F6,-8(R1)
9       ADD.D  F8,F6,F2
12      S.D    -8(R1),F8     ;drop DSUBUI & BNEZ
13      L.D    F10,-16(R1)
15      ADD.D  F12,F10,F2
18      S.D    -16(R1),F12   ;drop DSUBUI & BNEZ
19      L.D    F14,-24(R1)
21      ADD.D  F16,F14,F2
24      S.D    -24(R1),F16
25      DADDUI R1,R1,#-32    ;alter to 4*8
27      BNEZ   R1,LOOP
```

1 cycle stall

2 cycles stall

Rewrite loop to minimize stalls?

27 clock cycles, or 6.75 per iteration
(Assumes R1 is multiple of 4)

17

Unrolled Loop Scheduling That Minimizes Stalls

```
1  Loop:L.D    F0,0(R1)
2       L.D    F6,-8(R1)
3       L.D    F10,-16(R1)
4       L.D    F14,-24(R1)
5       ADD.D  F4,F0,F2
6       ADD.D  F8,F6,F2
7       ADD.D  F12,F10,F2
8       ADD.D  F16,F14,F2
9       S.D    0(R1),F4
10      S.D    -8(R1),F8
11      S.D    -16(R1),F12
12      DSUBUI R1,R1,#32
13      S.D    8(R1),F16 ; 8-32 = -24
14      BNEZ   R1,LOOP
```

14 clock cycles, or 3.5 per iteration

18

Speculation to greater ILP

- Greater ILP: Overcome control dependence by hardware speculating on outcome of branches and executing program as if guesses were correct
  - Speculation ⇒ fetch, issue, and execute instructions as if branch predictions were always correct
  - Dynamic scheduling ⇒ only fetches and issues instructions
- Essentially a data flow execution model: Operations execute as soon as their operands are available

19

HW Schemes: Instruction Parallelism

- Key idea: Allow instructions behind stall to proceed

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F12,F8,F14
```

- Enables out-of-order execution and allows out-of-order completion
- Will distinguish when an instruction *begins execution* and when it *completes execution*; between 2 times, the instruction is *in execution*
- In a dynamically scheduled pipeline, all instructions pass through issue stage in order (in-order issue)

20

Tomasulo's can overlap iterations of loops

- Register renaming
  - Multiple iterations use different physical destinations for registers (dynamic loop unrolling).
- Reservation stations
  - Permit instruction issue to advance past integer control flow operations
  - Also buffer old values of registers - totally avoiding the WAR stall that we saw in the scoreboard.
- Other perspective: Tomasulo building data flow dependency graph on the fly.

21

What about Precise Interrupts?

- Tomasulo had:
  In-order issue, out-of-order execution, and out-of-order completion
- Need to "fix" the out-of-order completion aspect so that we can find precise breakpoint in instruction stream.

22

4 Steps of Speculative Tomasulo/ROB Algorithm

1. Issue—get instruction from FP Op Queue

   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")

2. Execution—operate on operands (EX)

   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW

3. Write result—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

4. Commit—update register with reorder result

   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

23

Loop Unrolling in VLIW

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
L.D F0,0(R1)	L.D F6,-8(R1)				1
L.D F10,-16(R1)	L.D F14,-24(R1)				2
L.D F18,-32(R1)	L.D F22,-40(R1)	ADD.D F4,F0,F2	ADD.D F8,F6,F2		3
L.D F26,-48(R1)		ADD.D F12,F10,F2	ADD.D F16,F14,F2		4
		ADD.D F20,F18,F2	ADD.D F24,F22,F2		5
S.D 0(R1),F4	S.D -8(R1),F8	ADD.D F28,F26,F2			6
S.D -16(R1),F12	S.D -24(R1),F16				7
S.D -32(R1),F20	S.D -40(R1),F24			DSUBUI R1,R1,#48	8
S.D -0(R1),F28				BNEZ R1,LOOP	9

Unrolled 7 times to avoid delays

7 results in 9 clocks, or 1.3 clocks per iteration (1.8X)

Average: 2.5 ops per clock, 50% efficiency

Note: Need more registers in VLIW (15 vs. 6 in SS)

24

Chapter 4 Review Topics

- Multiprocessors
- Cache coherence problem
- Cache coherence protocols – what do they do
- Cache states in the cache coherence lifecycle

25

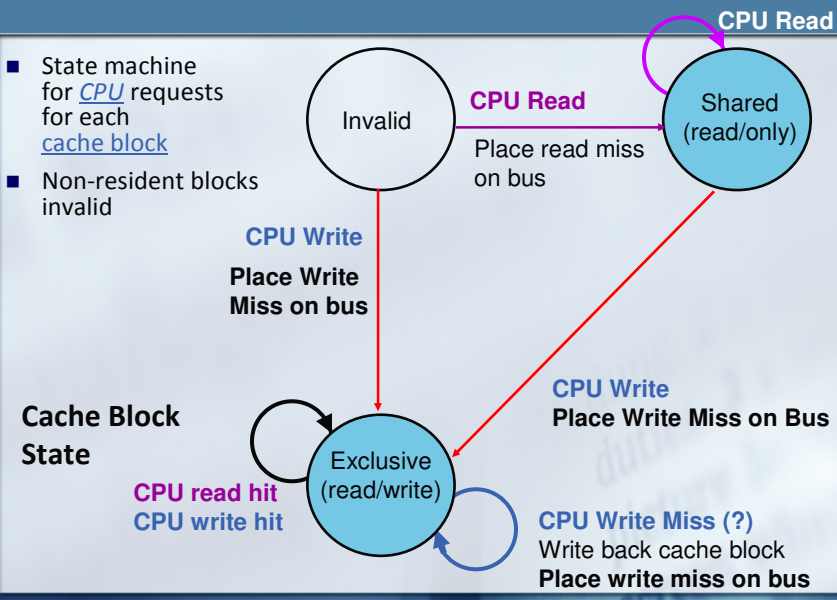
2 Classes of Cache Coherence Protocols

1. Directory based — Sharing status of a block of physical memory is kept in just one location, the directory
2. Snooping — Every cache with a copy of data also has a copy of sharing status of block, but no centralized state is kept
   - All caches are accessible via some broadcast medium (a bus or switch)
   - All cache controllers monitor or snoop on the medium to determine whether or not they have a copy of a block that is requested on a bus or switch access

26

Write-Back State Machine - CPU

- State machine for *CPU* requests for each cache block
- Non-resident blocks invalid



27

Coherence Misses

1. True sharing misses arise from the communication of data through the cache coherence mechanism
   - Invalidates due to 1st write to shared block
   - Reads by another CPU of modified block in different cache
   - Miss would still occur if block size were 1 word
2. False sharing misses when a block is invalidated because some word in the block, other than the one being read, is written into
   - Invalidation does not cause a new value to be communicated, but only causes an extra cache miss
   - Block is shared, but no word in block is actually shared
     ⇒ miss would not occur if block size were 1 word

28

Directory Protocol

- Similar to Snoopy Protocol: Three states
  - Shared: ≥ 1 processors have data, memory up-to-date
  - Uncached: (no processor has it; not valid in any cache)
  - Exclusive: 1 processor (owner) has data; memory out-of-date
- In addition to cache state, must track which processors have data when in the shared state (usually bit vector, 1 if processor has copy)
- Keep it simple:
  - Writes to non-exclusive data ⇒ write miss
  - Processor blocks until access completes
  - Assume messages received and acted upon in order sent

29

Appendix C Review Topics

- What makes caches work
- Principle of Locality
- Cache block placement schemes
- Cache page replacement policies
- Cache optimization techniques (also in Ch. 5)
- Memory access time with cache
- Multi-level caches
- Virtual Memory
- TLB

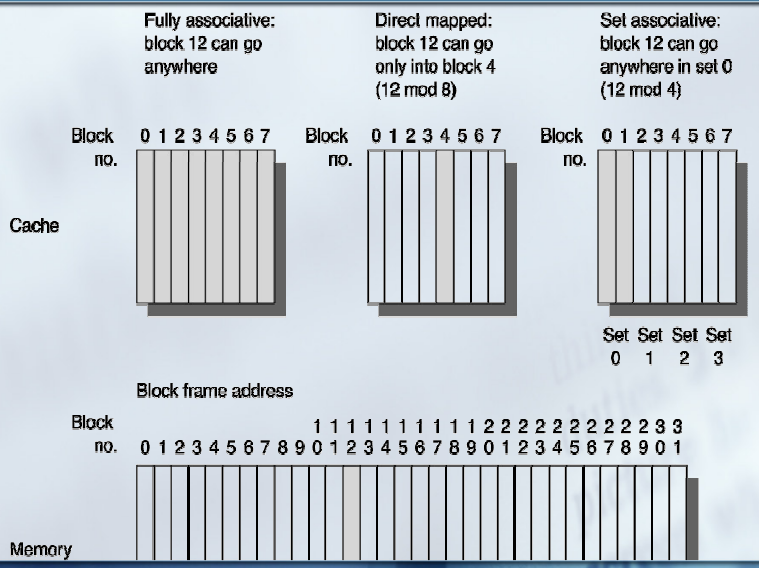
30

Cache Basics

- A *cache* is a (hardware managed) storage, intermediate in size, speed, and cost-per-bit between the programmer-visible registers and main physical memory
- The cache itself may be SRAM or fast DRAM
- There may be more than one level of caches
- Basis for cache to work: Principle of Locality
- When a location is accessed, it *and "nearby"* locations are likely to be accessed again soon
  - "Temporal" locality - Same location likely again soon
  - "Spatial" locality - Nearby locations likely

31

Block Placement Schemes



32

Replacement Strategies

- Which block do we replace when a new block comes in (on cache miss)?
- Direct-mapped: There's only one choice!
- Associative (fully- or set-):
  - If any frame in the set is empty, pick one of those.
  - Otherwise, there are many possible strategies:
    - Random: Simple, fast, and fairly effective
    - Least-Recently Used (LRU), and approximations thereof
      - Require bits to record replacement info., e.g. 4-way requires 4! = 24 permutations, need 5 bits to define the MRU to LRU positions
    - FIFO: Replace the oldest block.

33

Basic Cache Performance Formulas

$$Miss\ Rate = \frac{Misses\ per\ instruction}{Memory\ accesses\ per\ instruction}$$

- Memory access - data transfer requests (on load/store) and instruction memory access (always 1 per instruction)

$$Average\ memory\ access\ time = Hit\ time + Miss\ rate \times Miss\ penalty$$

- Units of measurement:
  - Hit time, Miss penalty – *ns* or *clock cycles*
  - Miss rate – unitless
  - Average memory access time – *ns* or *clock cycles*

34

Cache Performance Improvement

Average memory access time = (Hit time) + (Miss rate)×(Miss penalty)

"Amortized miss penalty"

- Reduce miss penalty:
  - Multilevel cache; Critical word first and early restart; priority to read miss; Merging write buffer; Victim cache
- Reduce miss rate:
  - Larger block size; Increase cache size; Higher associativity; Way prediction and Pseudo-associative caches; Compiler optimizations
- Reduce miss penalty/rate via parallelism:
  - Non-blocking cache; Hardware and Compiler-controlled prefetching
- Reduce hit time:
  - Small simple cache; Avoid address translation in indexing cache; Pipelined cache access; Trace caches

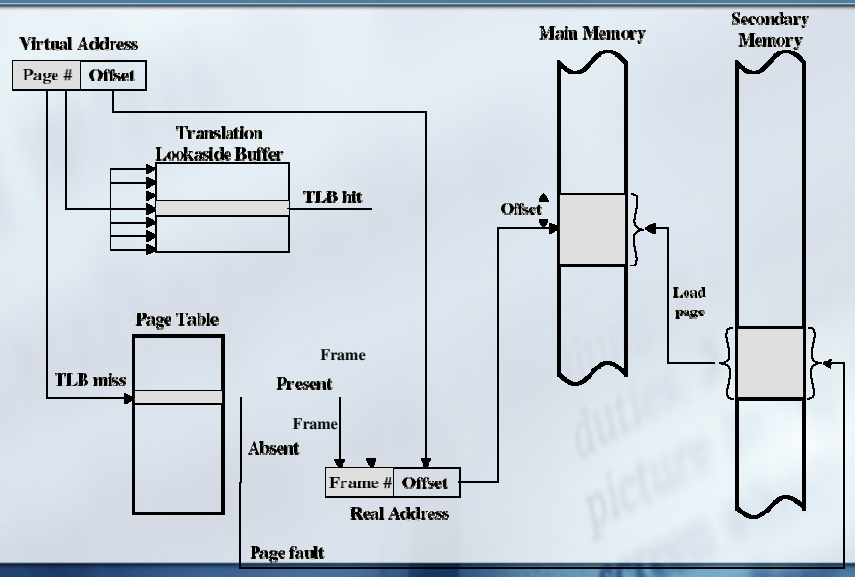
35

6 Basic Cache Optimizations

- Reducing Hit Time
  1. Avoiding Address Translation during Cache Indexing
- Reducing Miss Penalty
  2. Multilevel Caches
  3. Giving Reads Priority over Writes
     - E.g., Read complete before earlier writes in write buffer
- Reducing Miss Rate
  4. Larger Block size (Compulsory misses)
  5. Larger Cache size (Capacity misses)
  6. Higher Associativity (Conflict misses)

36

Translation Lookaside Buffer



37

Chapter 5 Review Topics

- Cache optimization techniques
- Virtual Machine concepts
- VMM

38

11 Advanced Cache Optimizations

- *Reducing hit time*

1. Small and simple caches
2. Way prediction
3. Trace caches

- *Increasing cache bandwidth*

4. Pipelined caches
5. Multibanked caches
6. Nonblocking caches

- *Reducing Miss Penalty*

7. Critical word first
8. Merging write buffers

- *Reducing Miss Rate*

9. Compiler optimizations

- *Reducing miss penalty or miss rate via parallelism*

10. Hardware prefetching
11. Compiler prefetching

39

Virtual Machine Monitors (VMMs)

- Virtual machine monitor (VMM) or hypervisor is software that supports VMs
- VMM determines how to map virtual resources to physical resources
- Physical resource may be time-shared, partitioned, or emulated in software
- VMM is much smaller than a traditional OS;
  - isolation portion of a VMM is ≈ 10,000 lines of code

40

Impact of VMs on Virtual Memory

- Virtualization of virtual memory if each guest OS in every VM manages its own set of page tables?
- VMM separates real and physical memory
  - Makes real memory a separate, intermediate level between virtual memory and physical memory
  - Some use the terms virtual memory, physical memory, and machine memory to name the 3 levels
  - Guest OS maps virtual memory to real memory via its page tables, and VMM page tables map real memory to physical memory
- VMM maintains a shadow page table that maps directly from the guest virtual address space to the physical address space of HW
  - Rather than pay extra level of indirection on every memory access
  - VMM must trap any attempt by guest OS to change its page table or to access the page table pointer

41

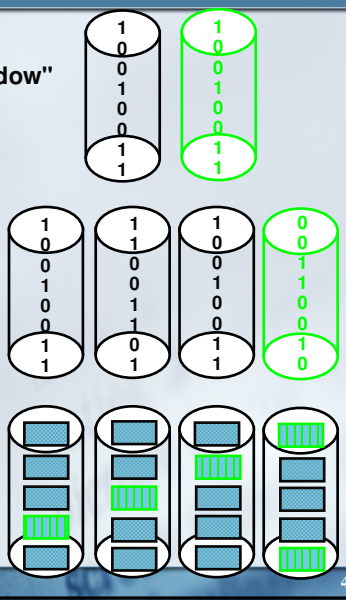
Chapter 6 Review Topics

- RAID schemes
- Queuing Theory
- Little's Law

42

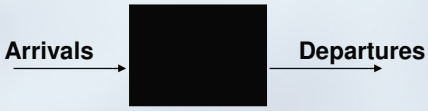
Summary: RAID Techniques

- *Disk Mirroring, Shadowing (RAID 1)*
  - Each disk is fully duplicated onto its "shadow"
  - Logical write = two physical writes
  - 100% capacity overhead
- *Parity Data Bandwidth Array (RAID 3)*
  - Parity computed horizontally
  - Logically a single high data bw disk
- *High I/O Rate Parity Array (RAID 5)*
  - Interleaved parity blocks
  - Independent reads and writes
  - Logical write = 2 reads + 2 writes



43

Introduction to Queuing Theory



- More interested in long term, steady state than in startup => Arrivals = Departures
- Little's Law:
  Mean number tasks in system = arrival rate x mean response time
  - Observed by many, Little was first to prove
- Applies to any system in equilibrium, as long as black box not creating or destroying tasks

44

Time in Queue

- All tasks in queue ($Length_{queue}$) ahead of new task must be completed before task can be serviced
  - Each task takes on average $Time_{server}$
  - Task at server takes average residual service time to complete
- Chance server is busy is *server utilization*
  $\Rightarrow$ expected time for service is Server utilization $\times$ Average residual service time
- $Time_{queue} = Length_{queue} \times Time_{server}$
  + Server utilization x Average residual service time
- Substituting definitions for $Length_{queue,}$ Average residual service time, & rearranging:

  $Time_{queue} = Time_{server} \times$ Server utilization/(1-Server utilization)

45

Example

- Little's Law: The average number of customers in a stable system (over some time interval) is equal to their average arrival rate, multiplied by their average time in the system.
- Q: At the supermarket a checkout operator has on average 4 customers and customers arrive every 2 minutes. What is the average time spent by a customer in a waiting line?

Average Response Time	Average number of customers in a system	Arrival Rate
? minutes/customer	4 customers	½ customer/minute

- A: Customers on average will be in line for 8 minutes.

46