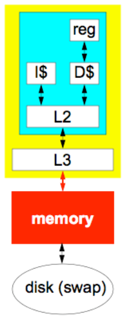
Lecture 26
Storage + I/O

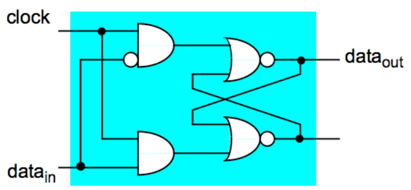
Storage Hierarchy II: Main Memory



main memory
- memory technology (DRAM)
- interleaving
- special DRAMs
- processor/memory integration

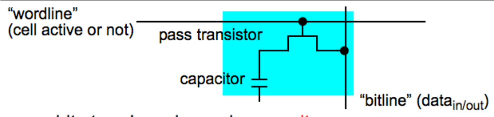
virtual memory and address translation

SRAM (Static Random Access Memory)



- "logic" (CPU process, registers are SRAM)
- store bits in flip-flops (cross-coupled NORs)
- – not very dense (six transistors per bit)
- + fast
- + doesn't need to be "refreshed" (data stays as long as power is on)

DRAM (Dynamic Random Access Memory)



- bit stored as charge in capacitor
  - optimized for density (1 transistor for DRAM vs. 6 for SRAM)
- – capacitor discharges on a read (destructive read)
  - read is automatically followed by a write (to restore bit)
- – charge leaks away over time (not static)
  - refresh by reading/writing every bit once every 2ms (row at a time)
- access time = time to read
- cycle time = time between reads > access time

Comparison with SRAM

SRAM
- optimized for speed, then density
  - + 1/4–1/8 access time of DRAM
  - – 1/4 density of DRAM
- bits stored as flip-flops (4-6 transistors per bit)
- static: bit not erased on a read
  - + no need to refresh
  - – greater power dissipated than DRAM
  - + access time = cycle time
- non-multiplexed address/data lines

DRAM Chip Specs

Year	#bits	Access Time	Cycle Time
1980	64Kb	150ns	300ns
1990	1Mb	80ns	160ns
1993	4Mb	60ns	120ns
2000	64Mb	50ns	100ns
2004	1Gb	45ns	75ns

- density: +60% annual
  - Moore's law: density doubles every 18 months
- speed: %7 annual

Example: Simple Main Memory

- 32-bit wide DRAM (1 word of data at a time)
  - pretty wide for an actual DRAM
- access time: 2 cycles (A)
- transfer time: 1 cycle (T)
  - time on the bus
- cycle time: 4 cycles (B = cycle time - access time)
  - B includes time to refresh after a read
- what is the miss penalty for a 4-word block?

Simple Main Memory

cycle	addr	mem
1	12	A
2		A
3		T/B
4		B
5	13	A
6		A
7		T/B
8		B
9	14	A
10		A
11		T/B
12		B
13	15	A
14		A
15		T/B
16		B

4-word access = 15 cycles

4-word cycle = 16 cycles

can we speed this up?

- lower latency?
  - – no
  - A,B & T are fixed
- higher bandwidth?

Bandwidth: Wider DRAMs

cycle	addr	mem
1	12	A
2		A
3		T/B
4		B
5	14	A
6		A
7		T/B
8		B

new parameter

- 64-bit DRAMs

4-word access = 7 cycles

4-word cycle = 8 cycles

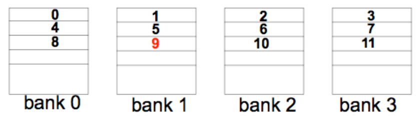
– 64-bit bus
- wide buses (especially off-chip) are hard
- electrical problems

– 64-bit DRAM is probably too wide

Bandwidth: Simple Interleaving/Banking

use multiple DRAMs, exploit their aggregate bandwidth

- each DRAM called a bank
  - not true: sometimes collection of DRAMs together called a bank
- M 32-bit banks
- simple interleaving: banks share address lines
- word A in bank (A % M) at (A div M)
  - e.g., M=4, A=9: bank 1, location 2



Simple Interleaving

cycle	addr	bank0	bank1	bank2	bank3
1	12	A	A	A	A
2		A	A	A	A
3		T/B	B	B	B
4		B	T/B	B	B
5				T	B
6					T

4-word access = 6 cycles

4-word cycle = 4 cycles

+ can start a new access in cycle 5
+ overlap access with transfer
+ and still use a 32-bit bus!

Bandwidth Determines Capacity?

aggressive configurations need a lot of banks

- 120ns DRAM (assume 64-bit banks)
- processor 1: 4ns clock, no cache ⇒ 1 64-bit ref / cycle
  - at least 32 banks (64 bits/4ns ~= 64bits/120ns * 32 banks)
- processor 2: add write-back cache ⇒ 1 64-bit ref / 4 cycles
  - at least 8 banks (64 bits/16 ns ~= 64 bits/120ns * 8 banks)
- – hard to make this many banks from *narrow DRAMs*
  - e.g., 32 64-bit banks from 1x64Mb DRAMS ⇒ 2048 DRAMS (16GB)
  - e.g., 32 64-bit banks from 4x16Mb DRAMS ⇒ 512 DRAMS (1GB)
  - can't force people to buy that much memory just to get bandwidth
- use wide DRAMs (32-bit) or optimize narrow DRAMs

Processor/Memory Integration

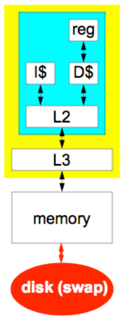
the next logical step: processor and memory on same chip

- move on-chip: FP, L2 caches, graphics. why not memory?
- – problem: processor/memory technologies incompatible
  - different number/kinds of metal layers
  - DRAM: capacitance is a good thing, logic: capacitance a bad thing

what needs to be done?

- use some DRAM area for simple processor (10% enough)
- eliminate external memory bus, milk performance from that
- integrate interconnect interfaces (processor/memory unit)
- re-examine tradeoffs: technology, cost, performance
- research projects: PIM, IRAM

Storage Hierarchy III: I/O System



- often boring, but still quite important
  - ostensibly about general I/O, mainly about disks
- performance: latency & throughput
- disks
  - parameters
  - extensions
  - redundancy and RAID
- buses
- I/O system architecture
  - DMA and I/O processors
- current research in I/O systems

I will briefly touch on all topics; (re-added to complete system architecture picture)

I/O (Disk) Performance

- who cares? you do
  - remember Amdahl's Law
  - want fast disk access (fast swap, fast file reads)
- I/O performance metrics
  - bandwidth of requests: *I/Os per second (IOPS)* → • Requests processed
  - raw data bandwidth: bytes per second → • Bytes/s
  - latency: response time → • Response time per IO

I/O Device Characteristics

- type
  - input: read only
  - output: write only
  - storage: both
- partner
  - human
  - machine
- data rate
  - peak transfer rate

Input to system

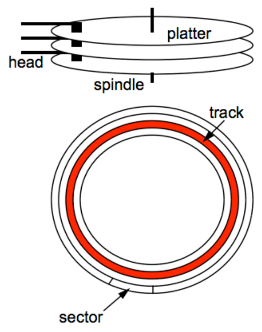
Output to display

device	type	partner	data rate KB/s
mouse	I	human	0.01
CRT	O	human	60,000
modem	I/O	machine	2-8
LAN	I/O	machine	500-6000
tape	storage	machine	2000
disk	storage	machine	2000-10,000

Both input & output

Of interest to this discussion

Disk Parameters



- 1–20 *platters* (data on both sides)
  - magnetic iron-oxide coating
  - 1 read/write head per side
- 500–2500 *tracks* per platter
- 32–128 *sectors* per track
  - sometimes fewer on inside tracks
- 512–2048 *bytes* per sector
  - usually fixed number of bytes/sector
  - data + ECC (parity) + gap
- 4–24GB total
- 3000–10000 RPM

Disk Usage Models

What metrics are important for what applications?

- data mining + supercomputing
  - large files, sequential reads
  - raw data transfer rate ($\text{rate}_{\text{transfer}}$) is most important
- transaction processing
  - large files, but random access, many small requests
  - IOPS is most important
- time sharing filesystems
  - small files, sequential accesses, potential for file caching
  - IOPS is most important

must design disk (I/O) system based on target workload

- use disk benchmarks (they exist)

Disk Alternatives

- solid state disk (SSD)
  - DRAM + battery backup with standard disk interface
  - + fast: no seek time, no rotation time, fast transfer rate
  - – expensive
- FLASH memory
  - + fast: no seek time, no rotation time, fast transfer rate
  - + non-volatile
  - – slow
  - – "wears" out over time
- optical disks (CDs, DVDs)
  - cheap if write-once, expensive if write-multiple
  - – slow

Actually, reads are proportional to normal DRAM, but writes take longer

Extensions to Conventional Disks

- increasing density: more sensitive heads, finer control
  - – increases cost
- fixed head: head per track
  - + seek time eliminated
  - – low track density
- parallel transfer: simultaneous read from multiple platters
  - – difficulty in looking onto different tracks on multiple surfaces
  - – lower cost alternatives possible (disk arrays)

More Extensions to Conventional Disks

- disk caches: disk-controller RAM buffers data
  - + fast writes: RAM acts as a write buffer
  - + better utilization of host-to-device path
  - – high miss rate increases request latency
- disk scheduling: schedule requests to reduce latency
  - e.g., schedule request with shortest seek time
  - e.g., "elevator" algorithm for seeks (head sweeps back and forth)
  - works best for unlikely cases (long queues)

Disk Fault Tolerance with RAID

- Redundant Array of Inexpensive Disks
  - Several smaller disks play a role of one big disk
- Can improve performance
  - Data spread among multiple disks
  - Accesses to different disks go in parallel
- Can improve reliability
  - Data can be kept with some redundancy

RAID 0

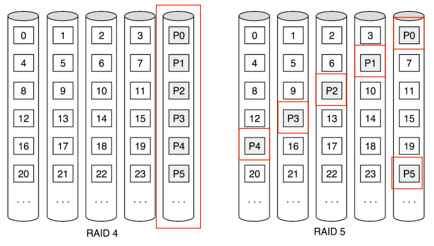
- Striping used to improve performance
  - Data stored on disks in array so that consecutive "stripes" of data are stored on different disks
  - Makes disks share the load, improving
    - Throughput: all disks can work in parallel
    - Latency: less queuing delay – a queue for each disk
- No Redundancy
  - Reliability actually lower than with single disk (if *any* disk in array fails, we have a problem)

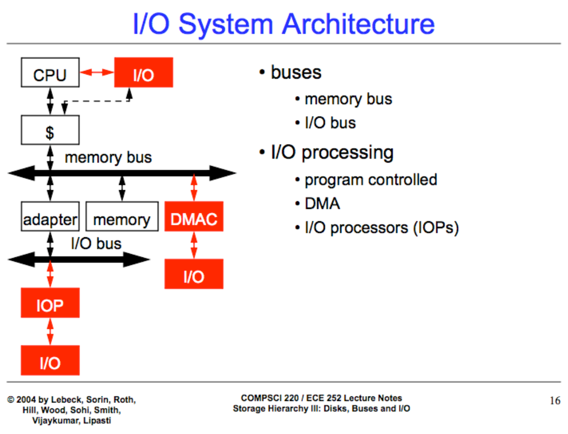
RAID 1

- Disk mirroring
  - Disks paired up, keep identical data
  - A write must update copies on both disks
  - A read can read any of the two copies
- Improved performance and reliability
  - Can do more reads per unit time
  - If one disk fails, its mirror still has the data
- If we have more than 2 disks (e.g. 8 disks)
  - "Striped mirrors" (RAID 1+0)
    - Pair disks for mirroring, striping across the 4 pairs
  - "Mirrored stripes" (RAID 0+1)
    - Do striping using 4 disks, then mirror that using the other 4

RAID 5

- Distributed block-interleaved parity
  - Like RAID 4, but parity blocks distributed to all disks
  - Read accesses only the data disk where the data is
  - A write must update the data block and its parity block
    - But now all disks share the parity update load





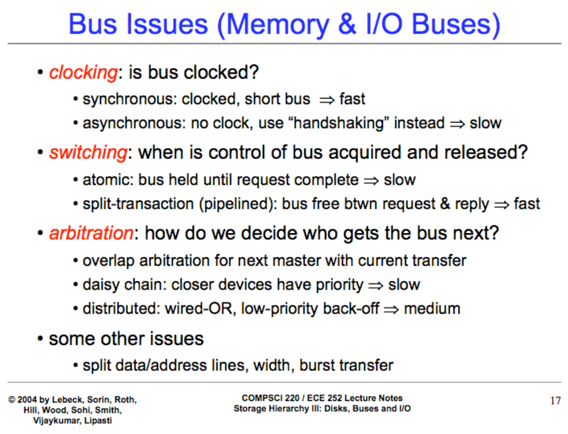
I/O System Architecture

- buses
  - memory bus
  - I/O bus
- I/O processing
  - program controlled
  - DMA
  - I/O processors (IOPs)

© 2004 by Lebeck, Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

COMPSCI 220 / ECE 252 Lecture Notes
Storage Hierarchy III: Disks, Buses and I/O

16



Bus Issues (Memory & I/O Buses)

- *clocking*: is bus clocked?
  - synchronous: clocked, short bus ⇒ fast
  - asynchronous: no clock, use "handshaking" instead ⇒ slow
- *switching*: when is control of bus acquired and released?
  - atomic: bus held until request complete ⇒ slow
  - split-transaction (pipelined): bus free btwn request & reply ⇒ fast
- *arbitration*: how do we decide who gets the bus next?
  - overlap arbitration for next master with current transfer
  - daisy chain: closer devices have priority ⇒ slow
  - distributed: wired-OR, low-priority back-off ⇒ medium
- some other issues
  - split data/address lines, width, burst transfer

© 2004 by Lebeck, Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

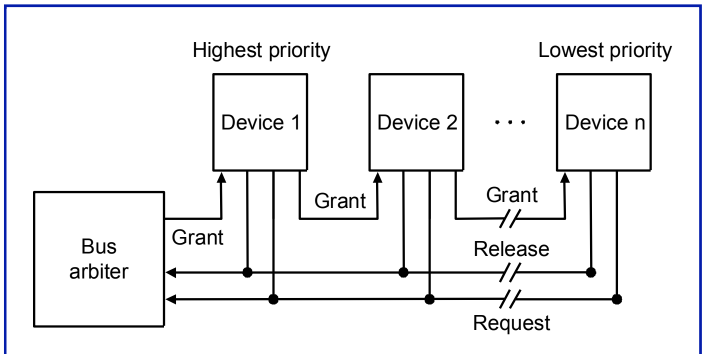
COMPSCI 220 / ECE 252 Lecture Notes
Storage Hierarchy III: Disks, Buses and I/O

17

Arbitration

- DMA implies multiple "owners" of the bus
  - must decide who owns the bus from cycle to cycle
- Arbitration
  - Daisy chain
  - Centralized parallel arbitration
  - Distributed arbitration by self selection
  - Distributed arbitration by collision detection
  - (see board for detailed examples and pictures…)

Daisy Chain



Simple but not fair and slow.

Centralized Parallel Arbitration

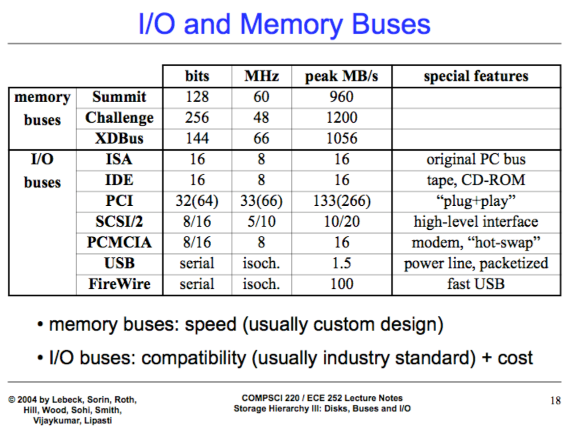
- Requires central arbiter
- Each device has separate line
- Central arbiter may become bottleneck
- Used in PCI bus

Distributed Arbitration by Self Selection

- Each device sees all requestors
- Priority scheme allows each to know if they get bus
- Requires lots of request lines

Distributed Arbitration by Collision Detection

- Devices independently request bus
- Devices have ability to detect simultaneous requests or Collisions.
- Upon collision a variety of schemes are used to select among requestors
- Used by Ethernet

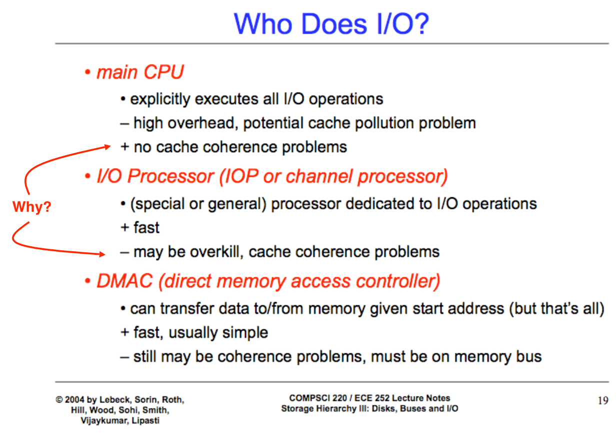


I/O and Memory Buses

		bits	MHz	peak MB/s	special features
memory buses	Summit	128	60	960	
	Challenge	256	48	1200	
	XDBus	144	66	1056	
I/O buses	ISA	16	8	16	original PC bus
	IDE	16	8	16	tape, CD-ROM
	PCI	32(64)	33(66)	133(266)	"plug+play"
	SCSI/2	8/16	5/10	10/20	high-level interface
	PCMCIA	8/16	8	16	modem, "hot-swap"
	USB	serial	isoch.	1.5	power line, packetized
	FireWire	serial	isoch.	100	fast USB

- memory buses: speed (usually custom design)
- I/O buses: compatibility (usually industry standard) + cost

© 2004 by Lebeck, Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti — COMPSCI 220 / ECE 252 Lecture Notes Storage Hierarchy III: Disks, Buses and I/O — 18

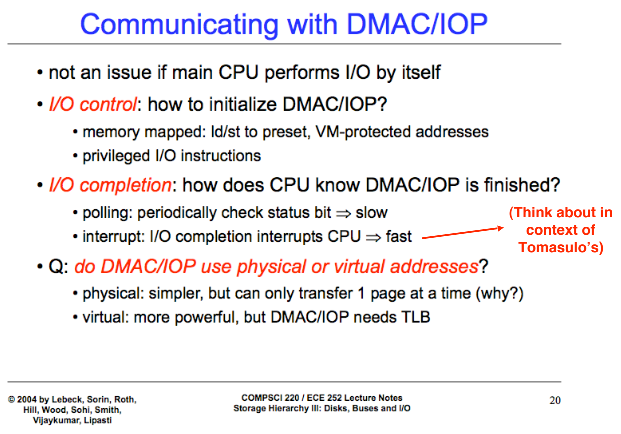


Who Does I/O?

- *main CPU*
  - explicitly executes all I/O operations
  - – high overhead, potential cache pollution problem
  - + no cache coherence problems
- *I/O Processor (IOP or channel processor)*
  - (special or general) processor dedicated to I/O operations
  - + fast
  - – may be overkill, cache coherence problems
- *DMAC (direct memory access controller)*
  - can transfer data to/from memory given start address (but that's all)
  - + fast, usually simple
  - – still may be coherence problems, must be on memory bus

Why?

© 2004 by Lebeck, Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti — COMPSCI 220 / ECE 252 Lecture Notes Storage Hierarchy III: Disks, Buses and I/O — 19

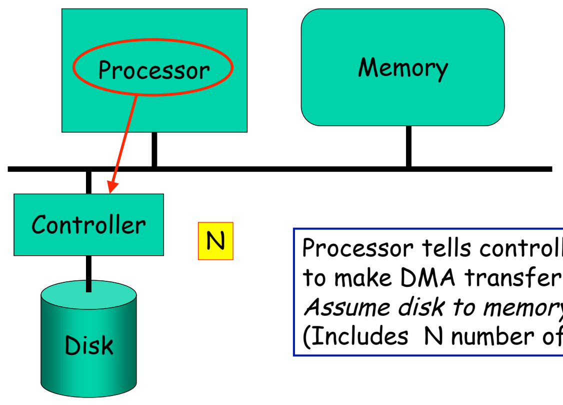


Communicating with DMAC/IOP

- not an issue if main CPU performs I/O by itself
- *I/O control*: how to initialize DMAC/IOP?
  - memory mapped: ld/st to preset, VM-protected addresses
  - privileged I/O instructions
- *I/O completion*: how does CPU know DMAC/IOP is finished?
  - polling: periodically check status bit ⇒ slow
  - interrupt: I/O completion interrupts CPU ⇒ fast (Think about in context of Tomasulo's)
- Q: *do DMAC/IOP use physical or virtual addresses*?
  - physical: simpler, but can only transfer 1 page at a time (why?)
  - virtual: more powerful, but DMAC/IOP needs TLB

© 2004 by Lebeck, Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti — COMPSCI 220 / ECE 252 Lecture Notes Storage Hierarchy III: Disks, Buses and I/O — 20

DMA



Processor tells controller to make DMA transfer. *Assume disk to memory.* (Includes N number of bytes)