CS61C

Great Ideas in Computer Architecture

RAID, Disk, I/O

Instructor: Steven Ho
Review of Last Lecture

• Great Idea: Dependability via Redundancy
  – Reliability: MTTF & Annualized Failure Rate
  – Availability: % uptime = MTTF/MTBF

• Error Correcting Code (ECC)
  – Encode data bits in larger “code words”
    • Denote valid and invalid code words
  – Hamming distance 2: Parity for Single Error Detect
  – Hamming distance 3: Single Error Correction Code + error position identification
Minimum Hamming Distance 3

Let 000 be valid

- How many bit errors can we detect?
  - Two! Takes 3 errors to reach another valid code word
- If 1-bit error, know which code word we came from?
  - Yes!

Only a quarter of the available code words are valid

Nearest 000 (one 1)

Nearest 111 (one 0)
Replace all the X’s with 1’s Everywhere else put a 0

OMFG

The parity bits form a number—the bit position in the code word!

- This tells you which bit of the code word is incorrect!
Hamming ECC Example (4/4)

Going in reverse:
We see 011100101110

Figure out which bit is wrong:

\[ p_1 : \overline{0101111} = \text{even number of 1's} : 0 \]
\[ p_2 : \overline{1101111} = \text{odd number of 1's} : 1 \]
\[ p_4 : \overline{10010} = \text{even number of 1's} : 0 \]
\[ p_8 : \overline{01110} = \text{odd number of 1's} : 1 \]

Incorrect code bit:

\[ 0b \overline{p_8p_4p_2p_1} = 0b 1010 = 10! \text{ So flip bit 10} \]
**Question:** What is the correct data word given the following SEC Hamming code: 0101101

Code word: 0 1 0 1 1 0 1 0 1

(A) 0 1 0 1
(B) 1 0 1 0
(C) 0 0 0 1
(D) 1 1 0 1
**Question:** What is the correct data word given the following SEC Hamming code: 0101101

**Code word:**

```
0 1 0 1 1 0 1
```

“X” means you check this bit position

```
p_4 0 0 0 X X X X X -> 1
p_2 0 X X 0 0 0 X X -> 0
p_1 X 0 X 0 X 0 X -> 0
```

(A) 0 1 0 1
(B) 1 0 1 0
(C) 0 0 0 1
(D) 1 1 0 1

0b 100 = 4
Flip the fourth bit—it’s a parity bit!
Data was correct
Hamming ECC “Cost”

• Space overhead in single error correction code
  – Form $p + d$ bit code word, where $p$ = # parity bits and $d$ = # data bits

• Want the $p$ parity bits to indicate either “no error” or 1-bit error in one of the $p + d$ places
  – Need $2^p \geq p + d + 1$, thus $p \geq \log_2(p + d + 1)$
  – For large $d$, $p$ approaches $\log_2(d)$
Hamming Single Error Correction, Double Error Detection (SEC/DED)

• Adding extra parity bit covering the entire SEC code word provides *double error detection* as well!

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
p_1 & p_2 & d_1 & p_3 & d_2 & d_3 & d_4 & p_4 \\
\end{array}
\]

• Let $H$ be the position of the incorrect bit we would find from checking $p_1$, $p_2$, and $p_3$ (0 means no error) and let $P$ be parity of complete code word (p’s & d’s)
  
  – $H=0$ $P=0$, no error
  – $H\neq0$ $P=1$, correctable single error ($P=1 \rightarrow$ odd # errors)
  – $H\neq0$ $P=0$, double error detected ($P=0 \rightarrow$ even # errors)
  – $H=0$ $P=1$, an error occurred in $p_4$ bit, not in rest of word
Agenda

• Dependability
• Administrivia
• Error Correcting Codes (ECC)
• RAID
Arrays of Small Disks

Can smaller disks be used to close the gap in performance between disks and CPUs?

Conventional: 4 disk types
- 3.5”
- 5.25”
- 10”
- 14”

Disk Array: 1 disk type
- 3.5”
Replace Large Disks with Large Number of Small Disks!
(Data from 1988 disks)

<table>
<thead>
<tr>
<th></th>
<th>IBM 3390K</th>
<th>IBM 3.5&quot; 0061</th>
<th>x72</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity</td>
<td>20 GBytes</td>
<td>320 MBytes</td>
<td>23 GBytes</td>
</tr>
<tr>
<td>Volume</td>
<td>97 cu. ft.</td>
<td>0.1 cu. ft.</td>
<td>11 cu. ft.</td>
</tr>
<tr>
<td>Power</td>
<td>3 KW</td>
<td>11 W</td>
<td>1 KW</td>
</tr>
<tr>
<td>Data Rate</td>
<td>15 MB/s</td>
<td>1.5 MB/s</td>
<td>120 MB/s</td>
</tr>
<tr>
<td>I/O Rate</td>
<td>600 I/Os/s</td>
<td>55 I/Os/s</td>
<td>3900 I/Os/s</td>
</tr>
<tr>
<td>MTTF</td>
<td>250 KHours</td>
<td>50 KHours</td>
<td>~700 KHours</td>
</tr>
<tr>
<td>Cost</td>
<td>$250K</td>
<td>$2K</td>
<td>$150K</td>
</tr>
</tbody>
</table>

Disk Arrays have potential for large data and I/O rates, high MB/ft$^3$, high MB/KW, but what about reliability?
RAID: Redundant Arrays of Inexpensive Disks

• Files are “striped” across multiple disks
  – Concurrent disk accesses improve throughput
• Redundancy yields high data availability
  – Service still provided to user, even if some components (disks) fail
• Contents reconstructed from data redundantly stored in the array
  – Capacity penalty to store redundant info
  – Bandwidth penalty to update redundant info
RAID 0: Data Striping

- “Stripe” data across all disks
  - Generally faster accesses (access disks in parallel)
  - No redundancy (really “AID”)
  - Bit-striping shown here, can do in larger chunks
RAID 1: Disk Mirroring

- Each disk is fully duplicated onto its “mirror”
  - Very high availability can be achieved
- Bandwidth sacrifice on write:
  - Logical write = two physical writes
  - Logical read = one physical read
- Most expensive solution: 100% capacity overhead
RAID 2-4: Data Striping + Parity

- Logical data is striped across disks
  - 2: bit, 3: byte, 4: block
- Parity disk P contains parity of other disks
- If any one disk fails, can use other disks to recover data!
  - We have to know which disk failed
- Must update Parity data on EVERY write
  - Logical write = min 2 to max N physical reads and writes
  - \( \text{parity}_{\text{new}} = \text{data}_{\text{old}} \oplus \text{data}_{\text{new}} \oplus \text{parity}_{\text{old}} \)
Updating the Parity Data

- Examine small write in RAID 3 (1 byte)
  - 1 logical write = 2 physical reads + 2 physical writes
  - Same concept applies for later RAIDs, too

What if writing halfword (2 B)? Word (4 B)?
Inspiration for RAID 5

- When writing to a disk, need to update Parity
- Small writes are bottlenecked by Parity Disk: Write to D0, D5 both also write to P disk
RAID 5: Interleaved Parity

Independent writes possible because of interleaved parity

Example: write to D0, D5 uses disks 1, 2, 4, 5

Increasing Logical Disk Addresses

Disk Columns
Modern Use of RAID and ECC (1/2)

- RAID 0 has no redundancy
- RAID 1 is too expensive
- RAID 2 is obsolete due to on-disk ECC
- RAID 3 is not commonly used (bad I/O rates)

- Typical modern code words in DRAM memory systems:
  - 64-bit data blocks (8 B) with 72-bit codes (9 B)
  - \( d = 64 \rightarrow p = 7, \) +1 for DED
Modern Use of RAID and ECC (2/2)

- Common failure mode is bursts of bit errors, not just 1 or 2
  - Network transmissions, disks, distributed storage
  - Contiguous sequence of bits in which first, last, or any number of intermediate bits are in error
  - Caused by impulse noise or by fading signal strength; effect is greater at higher data rates

- Other tools: cyclic redundancy check, Reed-Solomon, other linear codes
Question:
You have a 5-disk RAID 4 system with disk block size equal to your page size.

On a page fault, what is the worst case # of disk accesses (both reads and writes)?

(A) 1 (1 read, 0 writes)
(B) 2 (2 reads, 0 writes)
(C) 4 (2 reads, 2 writes)
(D) 5 (3 reads, 2 writes)
Question:
You have a 5-disk RAID 4 system with disk block size equal to your page size.

On a page fault, what is the worst case # of disk accesses (both reads and writes)?

(A) 1  (1 read, 0 writes)
(B) 2  (2 reads, 0 writes)
(C) 4  (2 reads, 2 writes)
(D) 5  (3 reads, 2 writes)

- Page fault -> page not in memory
- You need to fetch it from disk (1 read)
- You kick out a dirty page from the page table, so disk needs to be updated
- As we saw before, for a logical write you make two reads and two writes so 3 reads, 2 writes total
Review of RAID

• RAID: Redundant Arrays of Inexpensive Disks
  – RAID 0: data striping, no redundancy
  – RAID 1: disk mirroring
  – RAID 2: bit striping with ECC disks
  – RAID 3: byte striping with dedicated parity disk
  – RAID 4: block striping with dedicated parity disk
  – RAID 5: block striping with interleaved parity
  – Can access disks concurrently, but may require additional reads & writes for updating parity
Agenda

• RAID
• I/O Basics
• Administrivia
• Exceptions and Interrupts
• Disks
Five Components of a Computer

• Components a computer needs to work
  – Control
  – Datapath
  – Memory
  – Input
  – Output
Motivation for Input/Output

- I/O is how humans interact with computers.
- I/O gives computers long-term memory.
- I/O lets computers do amazing things:

  - Computer without I/O like a car without wheels; great technology, but gets you nowhere.

MIT Media Lab “Sixth Sense”
I/O Device Examples and Speeds

• I/O speeds: 7 orders of magnitude between mouse and LAN

<table>
<thead>
<tr>
<th>Device</th>
<th>Behavior</th>
<th>Partner</th>
<th>Data Rate (KB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keyboard</td>
<td>Input</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Mouse</td>
<td>Input</td>
<td>Human</td>
<td>0.02</td>
</tr>
<tr>
<td>Voice output</td>
<td>Output</td>
<td>Human</td>
<td>5.00</td>
</tr>
<tr>
<td>Floppy disk</td>
<td>Storage</td>
<td>Machine</td>
<td>50.00</td>
</tr>
<tr>
<td>Laser printer</td>
<td>Output</td>
<td>Human</td>
<td>100.00</td>
</tr>
<tr>
<td>Magnetic disk</td>
<td>Storage</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Wireless network</td>
<td>Input or Output</td>
<td>Machine</td>
<td>10,000.00</td>
</tr>
<tr>
<td>Graphics display</td>
<td>Output</td>
<td>Human</td>
<td>30,000.00</td>
</tr>
<tr>
<td>Wired LAN network</td>
<td>Input or Output</td>
<td>Machine</td>
<td>125,000.00</td>
</tr>
</tbody>
</table>

• When discussing transfer rates, use SI prefixes ($10^x$)
What do we need for I/O to work?

1) A way to connect many types of devices
2) A way to control these devices, respond to them, and transfer data
3) A way to present them to user programs so they are useful
Instruction Set Architecture for I/O

• What must the processor do for I/O?
  – Input: reads a sequence of bytes
  – Output: writes a sequence of bytes

• Some processors have special input and output instructions (CISC!)

• Alternative model (used by RISC-V):
  – Use loads for input, stores for output (in small pieces)
  – Called Memory Mapped Input/Output
  – A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)
Memory Mapped I/O

- Certain addresses are not regular memory
- Instead, they correspond to registers in I/O devices

![Diagram showing memory mapped I/O addresses](image)

- **address**: $0xFFFFFFFF$
- **address**: $0xFFFF0000$
- **control reg.**
- **data reg.**

8/2/2018 CS61C Su18 - Lecture 25
Processor-I/O Speed Mismatch

• 1 GHz microprocessor can execute 1 billion load or store instr/sec (4,000,000 KB/s data rate)
  – Recall: I/O devices data rates range from 0.01 KB/s to 125,000 KB/s
• Input: Device may not be ready to send data as fast as the processor loads it
  – Also, might be waiting for human to act
• Output: Device may not be ready to accept data as fast as processor stores it
Processor Checks Status Before Acting

• Path to a device generally has 2 registers:
  • Control Register says it’s OK to read/write (I/O ready)
  • Data Register contains data

1) Processor reads from control register in a loop, waiting for device to set Ready bit (0 → 1)

2) Processor then loads from (input) or writes to (output) data register
   – Resets Ready bit of control register (1 → 0)

• This process is called “Polling”
I/O Example (Polling in MIPS)

• **Input:** Read from keyboard into $v0
  
  ```
  lui $t0, 0xffff  # ffff0000
  Waitloop: lw $t1, 0($t0)  # control reg
  andi $t1,$t1,0x1
  beq$t1,$zero, Waitloop
  lw $v0, 4($t0)  # data reg
  ```

• **Output:** Write to display from $a0
  
  ```
  lui $t0, 0xffff  # ffff0000
  Waitloop: lw $t1, 8($t0)  # control reg
  andi $t1,$t1,0x1
  beq$t1,$zero, Waitloop
  sw $a0,12($t0)  # data reg
  ```

• “Ready” bit is from processor’s point of view!
Cost of Polling?

• Processor specs: 1 GHz clock, **400 clock cycles for a polling operation** (call polling routine, accessing the device, and returning)

• Determine % of processor time for polling:
  – **Mouse**: Polled 30 times/sec so as not to miss user movement
  – **Floppy disk**: Transferred data in 2-Byte units with data rate of 50 KB/sec. No data transfer can be missed.
  – **Hard disk**: Transfers data in 16-Byte chunks and can transfer at 16 MB/second. Again, no transfer can be missed.
% Processor time to poll

• Mouse polling:
  – *Time taken*: \(30 \text{ [polls/s]} \times 400 \text{ [clocks/poll]} = 12K \text{ [clocks/s]}\)
  – *% Time*: \(1.2 \times 10^4 \text{ [clocks/s]} / 10^9 \text{ [clocks/s]} = 0.0012\%
  – Polling mouse has little impact on processor

• Disk polling:
  – *Freq*: \(16 \text{ [MB/s]} / 16 \text{ [B/poll]} = 1M \text{ [polls/s]}\)
  – *Time taken*: \(1M \text{ [polls/s]} \times 400 \text{ [clocks/poll]} = 400M \text{ [clocks/s]}\)
  – *% Time*: \(4 \times 10^8 \text{ [clocks/s]} / 10^9 \text{ [clocks/s]} = 40\%
  – Unacceptable!

• **Problems**: polling, accessing small chunks
Alternatives to Polling?

• Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
• Would like an unplanned procedure call that would be invoked only when I/O device is ready

• **Solution:** Use *exception* mechanism to help trigger I/O, then *interrupt* program when I/O is done with data transfer
  – This method is discussed next
Agenda

• RAID
• I/O Basics
• Administrivia
• Exceptions and Interrupts
• Disks
Administrivia

• Proj4 due on Friday (8/03)
  – Hold off on submissions for now
• HW7 due 8/06
• Regrade requests are open for MT2 until Friday
• The final will be 8/09 7-10PM @VLSB 2040/2060!
  – If you have a conflict, and you haven’t heard from me (Damon) please contact me ASAP
Agenda

• RAID
• I/O Basics
• Administrivia
• Exceptions and Interrupts
• Disks
Exceptions and Interrupts

• “Unexpected” events requiring change in flow of control
  – Different ISAs use the terms differently

• Exception
  – Arises within the CPU
    (e.g. undefined opcode, overflow, syscall, TLB Miss)

• Interrupt
  – From an external I/O controller

• Dealing with these without sacrificing performance is difficult!
Handling Exceptions (HW)

- In RISC-V, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In RISC-V: save in special register called User Exception Program Counter (UEPC)
- Save indication of the problem
  - In RISC-V: saved cause in special register called User Cause register (ucause)
  - Also save the invalid addr/instr in utval register
- Jump to exception handler code at the address stored in user trap handler base address (utvec)
Exception Properties

• Re-startable exceptions
  – Pipeline can flush the instruction
  – Handler executes, then returns to the instruction
    • Re-fetched and executed from scratch

• PC+4 saved in UEPC register
  – Identifies causing instruction
  – PC+4 because it is the available signal in a pipelined implementation
    • Handler must adjust this value to get right address
Handler Actions (SW)

• Read Cause register, and transfer to relevant handler

• OS determines action required:
  – If restartable exception, take corrective action and then use UEPC to return to program
  – Otherwise, terminate program and report error using UEPC, Cause register, etc.
    (e.g. our best friend: SEGFAULT)
Exceptions in a Pipeline

• Another kind of control hazard
• Consider overflow on fadd.s in EX stage
  fadd.s x1, x2, x1
  1) Prevent x1 from being clobbered
  2) Complete previous instructions
  3) Flush fadd and subsequent instructions
  4) Set Cause, UEPC, UTVAL register values
  5) Transfer control to handler

• Similar to mispredicted branch
  – Use much of the same hardware
Exception Example

Time (clock cycles)

Instr. Order

and

or

fadd

slt

lw

lui
Exception Example

Time (clock cycles)

Flush add, slt, lw

1st instruction of handler

Save PC+4 into UEPC
Multiple Exceptions

• Pipelining overlaps multiple instructions
  – Could have multiple exceptions at once!
  – e.g. page fault in \texttt{lw} the same clock cycle as FP exception of following instruction \texttt{fadd}

• Simple approach: Deal with exception from \textit{earliest} instruction and flush subsequent instructions
  – Called \textit{precise exceptions}
  – In previous example, service \texttt{lw} exception first
I/O Interrupt

• An I/O interrupt is like an exception except:
  – An I/O interrupt is “asynchronous”
  – More information needs to be conveyed

• “Asynchronous” with respect to instruction execution:
  – I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  – I/O interrupt does not prevent any instruction from running to completion
Interrupt-Driven Data Transfer

1. I/O interrupt
2. Save PC
3. Jump to interrupt service routine
4. Perform transfer
5. Read
   Store
   ...
   JR

Memory

Add
Sub
And
Or

User program

Interrupt service routine
Interrupt-Driven I/O Example (1/2)

• Assume the following system properties:
  – 500 clock cycle overhead for each transfer, including interrupt
  – Disk throughput of 16 MB/s
  – Disk interrupts after transferring 16 B
  – Processor running at 1 GHz

• If disk is active 5% of program, what % of processor is consumed by the disk?
  – 5% × 16 [MB/s] / 16 [B/inter] = 50,000 [inter/s]
  – 50,000 [inter/s] × 500 [clocks/inter] = 2.5×10^7 [clocks/s]
  – 2.5×10^7 [clocks/s] / 10^9 [clock/s] = 2.5% busy
Interrupt-Driven I/O Example (2/2)

• 2.5% busy (interrupts) much better than 40% (polling)

• Real Solution: Direct Memory Access (DMA) mechanism
  – Device controller transfers data directly to/from memory without involving the processor
  – Only interrupts once per page (large!) once transfer is done
PIO vs. DMA
Direct Memory Access (DMA)

• Allows I/O devices to directly read/write main memory
• New Hardware: the DMA Engine
• DMA engine contains registers written by CPU:
  – Memory address to place data
  – # of bytes
  – I/O device #, direction of transfer
  – unit of transfer, amount to transfer per burst
Operation of a DMA Transfer

[From Section 5.1.4 Direct Memory Access in
Modern Operating Systems by Andrew S.
Tanenbaum, Herbert Bos, 2014]
DMA: Incoming Data

1. Receive interrupt from device
2. CPU takes interrupt, begins transfer
   – Instructs DMA engine/device to place data @ certain address
3. Device/DMA engine handle the transfer
   – CPU is free to execute other things
4. Upon completion, Device/DMA engine interrupt the CPU again
DMA: Outgoing Data

1. CPU decides to initiate transfer, confirms that external device is ready
2. CPU begins transfer
   – Instructs DMA engine/device that data is available @ certain address
3. Device/DMA engine handle the transfer
   – CPU is free to execute other things
4. Device/DMA engine interrupt the CPU again to signal completion
## Meet the Staff

<table>
<thead>
<tr>
<th></th>
<th>Steven</th>
<th>Nick</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming Language for life</strong></td>
<td>Go</td>
<td>Python2.7</td>
</tr>
<tr>
<td><strong>Tabs or Spaces</strong></td>
<td>Tabs that map to Spaces</td>
<td>Tabs</td>
</tr>
<tr>
<td><strong>Dipping Sauce</strong></td>
<td>Honey BBQ</td>
<td>Honey Mustard</td>
</tr>
<tr>
<td><strong>Favorite Hive Number</strong></td>
<td>2</td>
<td>18</td>
</tr>
</tbody>
</table>
Agenda

• RAID
• I/O Basics
• Administrivia
• Exceptions and Interrupts
• Disks
Computer Memory Hierarchy
One of our “Great Ideas”

- Processor
- CPU
  - Processor Register
  - CPU Cache
    - Level 1 (L1) Cache
    - Level 2 (L2) Cache
    - Level 3 (L3) Cache
- Physical Memory
  - Random Access Memory (RAM)
- Solid State Memory
  - Non-Volatile Flash-Based Memory
- Virtual Memory
  - File-Based Memory

- EDO, SD-RAM, DDR-SDRAM, RD-RAM and More...
- SSD, Flash Drive
- Mechanical Hard Drives

SUPER FAST SUPER EXPENSIVE TINY CAPACITY
FASTER EXPENSIVE SMALL CAPACITY
FAST PRICED REASONABLY AVERAGE CAPACITY
AVERAGE SPEED PRICED REASONABLY AVERAGE CAPACITY
SLOW CHEAP LARGE CAPACITY

▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng
Disks

• Nonvolatile storage
  – Retains its value without applying power to disk (SRAM/DRAM)
  – Long-term, inexpensive storage for files

• Two Types:
  – Magnetic Drives (Mechanical Drive and Floppy Disk)
    • Information stored by magnetizing ferrite material on surface of rotating disk
  – Flash Drives (Solid State Drives and Thumb Drives)
    • Information stored by trapping charge in a semiconductor and MOSFET based dual-gate transistor
Photo of Disk Internals

Actuator

Arm

Spindle

Platters (1-12)

Head

Video of hard disk in action

8/2/2018
• Several platters with information recorded magnetically on both surfaces (usually)
  – Bits recorded in *tracks*, which in turn are divided into *sectors* (usually 512 Bytes)
  – Entire package vacuum sealed to keep out dust

• Reading and writing from the disk:
  – *Actuator* moves *head* (end of *arm*) over track ("seek"), wait for sector to rotate under head, then read or write
  – *Disk Latency* = Seek Time + Rotation Time + Transfer Time + Controller Overhead
Disk Device Performance (2/2)

- Average values to plug into the formula:
- Rotation Time: Average distance of sector from head?
  - 1/2 time of a rotation
    - 7200 Revolutions Per Minute $\Rightarrow$ 120 Rev/sec
    - 1 revolution = 1/120 sec $\Rightarrow$ 8.33 milliseconds
    - 1/2 rotation (revolution) $\Rightarrow$ 4.17 ms

- Seek time: Average no. tracks to move arm?
  - Number of tracks/3 (see CS186 for the math)
  - Then, seek time = avg. number of tracks moved $\times$ time to move across one track
Flash Memory / SSD Technology

- NMOS transistor with an additional conductor between gate and source/drain which “traps” electrons. The presence/absence is a 1 or 0.
- Memory cells can withstand a limited number of program-erase cycles. Controllers use a technique called wear leveling to distribute writes as evenly as possible across all the flash blocks in the SSD.
What does Apple put in its iPods?

- Toshiba flash: 1, 2 GB
- Samsung flash: 4, 8 GB
- Toshiba 1.8-inch HDD: 80, 160 GB
- Toshiba flash: 8, 16, 32 GB

Types of iPods:
- Shuffle
- Nano
- Classic
- Touch
## Comparing Drive Types

<table>
<thead>
<tr>
<th>Feature</th>
<th>HDD</th>
<th>Flash-based SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Durability and Noise</strong></td>
<td>Loud and susceptible to shock</td>
<td>No moving parts!</td>
</tr>
<tr>
<td><strong>Access Time</strong></td>
<td>~ 12 ms</td>
<td>~ 0.1 ms</td>
</tr>
<tr>
<td></td>
<td>≈ 30M clock cycles</td>
<td>≈ 250K clock cycles</td>
</tr>
<tr>
<td><strong>Relative Power</strong></td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>~ $0.035 / GB</td>
<td>~ $0.35 / GB</td>
</tr>
<tr>
<td><strong>Capacity</strong></td>
<td>500GB - 4TB</td>
<td>128GB - 500GB</td>
</tr>
<tr>
<td><strong>Other Problems</strong></td>
<td>Fragmentation</td>
<td>Limited Writes</td>
</tr>
<tr>
<td><strong>Lifespan</strong></td>
<td>5-10 years</td>
<td>3-5 years</td>
</tr>
</tbody>
</table>
Summary (1/2)

• I/O gives computers their 5 senses + long term memory
  – I/O speed range is 7 orders of magnitude (or more!)

• Processor speed means must synchronize with I/O devices before use:
  – Polling works, but expensive due to repeated queries

• Exceptions are “unexpected” events in processor

• Interrupts are asynchronous events that are often used for interacting with I/O devices
Summary (2/2)

• Disks come in two main varieties
  – Magnetic Drives
    • Information stored by magnetizing ferrite material on surface of rotating disk
    • Loud, big, slow, cheap, ubiquitous
  – Flash Drives
    • Information stored by trapping charge in a semiconductor and MOSFET based dual-gate transistor
    • Quiet, smaller, FAST, expensive