CS 61C: Great Ideas in Computer Architecture

Input/Output

Instructor: Justin Hsia
Review of Last Lecture (1/2)

• 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
  – Hamming distance 4: SEC/Double Error Detection
Review of Last Lecture (2/2)

- **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
  - RAID 6: block striping with two interleaved parity disks for DEC
  - Can access disks concurrently, but may require additional reads & writes for updating parity
**Question:** What is the correct data word given the following SEC Hamming code: 0101101

<table>
<thead>
<tr>
<th>Code word:</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₄</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>p₂</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>p₁</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

• Nonvolatile storage
  – Information stored by magnetizing ferrite material on surface of rotating disk
  – Retains its value without applying power to disk (SRAM)
  – Long-term, inexpensive storage for files

• Two Types:
  – Floppy disks – slower, less dense, removable
  – Hard Disk Drives (HDD) – faster, more dense, non-removable
Photo of Disk Internals

- Spindle
- Arm
- Head
- Actuator
- Platters (1-12)
• Several platters, with information recorded magnetically on both surfaces (usually)
• Bits recorded in *tracks*, which in turn are divided into *sectors* (usually 512 Bytes)
• *Actuator* moves *head* (end of *arm*) over track ("seek"), wait for sector to rotate under head, then read or write
Disk Device Performance (1/3)

- **Disk Latency** = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  - **Seek Time** depends on number of tracks to move arm and speed of actuator
• *Disk Latency* = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  
  – *Rotation Time* depends on speed of disk rotation and how far sector is from head
• **Disk Latency** = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  
  – **Transfer Time** depends on size of request and data rate (bandwidth) of disk, which is a function of bit density and RPM
Disk Device Performance (2/3)

• Average distance of sector from head?
  – Average Rotation Time = time for 1/2 a rotation
    • 7200 revolutions per minute \(\rightarrow\) 1 rev/8.33 ms
    • 1/2 rotation (revolution) \(\rightarrow\) 4.17 ms

• Average no. tracks to move arm?
  – Disk industry standard benchmark:
    • Sum all time for all possible seek distances from all possible tracks / # possible
    • Assumes average seek distance is random
  – Seek Time usually given or to be solved for
Disk Device Performance (3/3)

• How long for data read/write to happen?
  – Transfer Time = size of data request / transfer rate
    • Note that transfer rates listed in SI prefixes (10^x)

• Disks have internal controllers to handle requests and data transfers
  – Controller Overhead usually given or to be solved for
  – Size of disk cache can strongly affect performance
    • Cache built into disk system, OS knows nothing
Disk Drive Performance Example

• 7200 RPM drive, 4 ms seek time, 20.48 MB/s transfer rate. Negligible controller overhead. Latency to read 100 KiB file?
  – Rotation time = 4.17 ms (from last slide)
  – Transfer time = 0.1 MiB / 20 (MiB/sec) = 5 ms
  – Latency = 4 + 4.17 + 5 = 13.17 ms
  – Throughput = 100 KiB/13.17 ms = 7.59 MiB/sec

• How do numbers change when reading bigger/smaller file? File fragmented across multiple locations?
Flash Memory

• Microdrives and Flash memory (e.g. CompactFlash) are going head-to-head
  – Both non-volatile (no power, data ok)
  – Flash benefits: durable & lower power
    (no moving parts vs. need to spin μdrives up/down)
  – Flash limitations: finite number of write cycles (wear on the insulating oxide layer around the charge storage mechanism). Most $\geq$ 100K, some $\geq$ 1M W/erase cycles.

• How does Flash memory work?
  – NMOS transistor with an additional conductor between gate and source/drain which “traps” electrons. The presence/absence is a 1 or 0.

  en.wikipedia.org/wiki/Flash_memory
What does Apple put in its iPods?

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

- Data storage devices with same electronic interfaces as HDD, but implemented (usually) with flash

<table>
<thead>
<tr>
<th></th>
<th>HDD</th>
<th>Flash-based SSD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Time</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>Relative Power</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>Cost</td>
<td>~ $0.05-$0.10 / GB</td>
<td>~ $0.65 / GB</td>
</tr>
</tbody>
</table>

- SSDs are also quieter, lighter, unsusceptible to magnetic fields and fragmentation, and start up faster
Agenda

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

- Final Review – Tue 8/13, 7-10pm in 10 Evans
- Final – Fri 8/16, 9am-12pm, 155 Dwinelle
  - 2nd half material + self-modifying MIPS
  - MIPS Green Sheet provided again
  - Two two-sided handwritten cheat sheets
    - Can re-use your midterm cheat sheet!
- “Dead Week” – WTh 8/14-15
- HKN Course Survey at end of lecture tomorrow (8/13)
Agenda

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

MIT Media Lab “Sixth Sense”

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

• Alternative model (used by MIPS):
  – 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

<table>
<thead>
<tr>
<th>address</th>
<th>control reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFFFFFFF</td>
<td></td>
</tr>
<tr>
<td>0xFFFF0000</td>
<td>data reg.</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
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 $\rightarrow$ 1)

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

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

- **Input:** Read from keyboard into \$v0
  
  ```mips
  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
  
  ```mips
  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
Get To Know Your Instructor
Agenda

- Disks
- Administrivia
- I/O Basics
- Exceptions and Interrupts
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 MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In MIPS: save in special register called *Exception Program Counter (EPC)*
- Save indication of the problem
  - In MIPS: saved in special register called *Cause* register
  - In simple implementation, might only need 1-bit (0 for undefined opcode, 1 for overflow)
- Jump to *exception handler code* at address 0x80000180
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 EPC 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 EPC to return to program
  – Otherwise, terminate program and report error using EPC, Cause register, etc.
    (e.g. our best friend the segfault)
Exceptions in a Pipeline

• Another kind of control hazard
• Consider overflow on add in EX stage
  \[
  \text{add } \$1, \ \$2, \ \$1
  \]
  1) Prevent \$1 from being clobbered
  2) Complete previous instructions
  3) Flush \text{add} and subsequent instructions
  4) Set Cause and EPC 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

add

slt

lw

lui

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg

I$

Reg

ALU

DS

Reg
Exception Example

Time (clock cycles)

Instruction Order

flush add, slt, lw

1st instruction of handler
Multiple Exceptions

• Pipelining overlaps multiple instructions
  – Could have multiple exceptions at once!
  – e.g. page fault in \( \text{lw} \) the same clock cycle as overflow of following instruction \( \text{add} \)

• **Simple approach:** Deal with exception from *earliest* instruction and flush subsequent instructions
  – Called *precise exceptions*
  – In previous example, service \( \text{lw} \) exception first

• What about multiple issue or out-of-order execution?
  – Maintaining precise exceptions can be difficult!
Imprecise Exceptions

• Just stop pipeline and save state
  – Including exception cause(s)

• Let the software handler work out:
  – Which instruction(s) had exceptions
  – Which to complete or flush
    • May require “manual” completion

• Simplifies hardware, but more complex handler software
  – Not feasible for complex multiple-issue out-of-order pipelines to always get exact instruction

• All computers today offer precise exceptions—affects performance though
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

Memory

User program

Interrupt service routine

Add
Sub
And
Or
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?
  – \(5\% \times 16 \text{ [MB/s]} / 16 \text{ [B/inter]} = 50,000 \text{ [inter/s]}\)
  – \(50,000 \text{ [inter/s]} \times 500 \text{ [clocks/inter]} = 2.5 \times 10^7 \text{ [clocks/s]}\)
  – \(2.5 \times 10^7 \text{ [clocks/s]} / 10^9 \text{ [clock/s]} = 2.5\% \text{ busy}\)
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
Summary

• Disks work by positioning head over spinning platters
  – Very slow relative to CPU, flash memory is alternative
• 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