# CS 61C: Great Ideas in Computer Architecture

*Input/Output*

**Instructor:** Alan Christopher

# 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

| Code word: | **0** | **1** | **0** | **1** | **1** | **0** | **1** |
|---|---|---|---|---|---|---|---|
| $p_4$ | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $p_2$ | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $p_1$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

| | | | | |
|---|---|---|---|---|
| **(B)** | **0** | **1** | **0** | **1** |
| **(G)** | **1** | **0** | **1** | **0** |
| **(P)** | **0** | **0** | **0** | **1** |
| **(Y)** | **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)?

| | | |
|---|---|---|
| **(B)** | **1** | **(1 read, 0 writes)** |
| **(G)** | **2** | **(2 reads, 0 writes)** |
| **(P)** | **4** | **(2 reads, 2 writes)** |
| **(Y)** | **5** | **(3 reads, 2 writes)** |

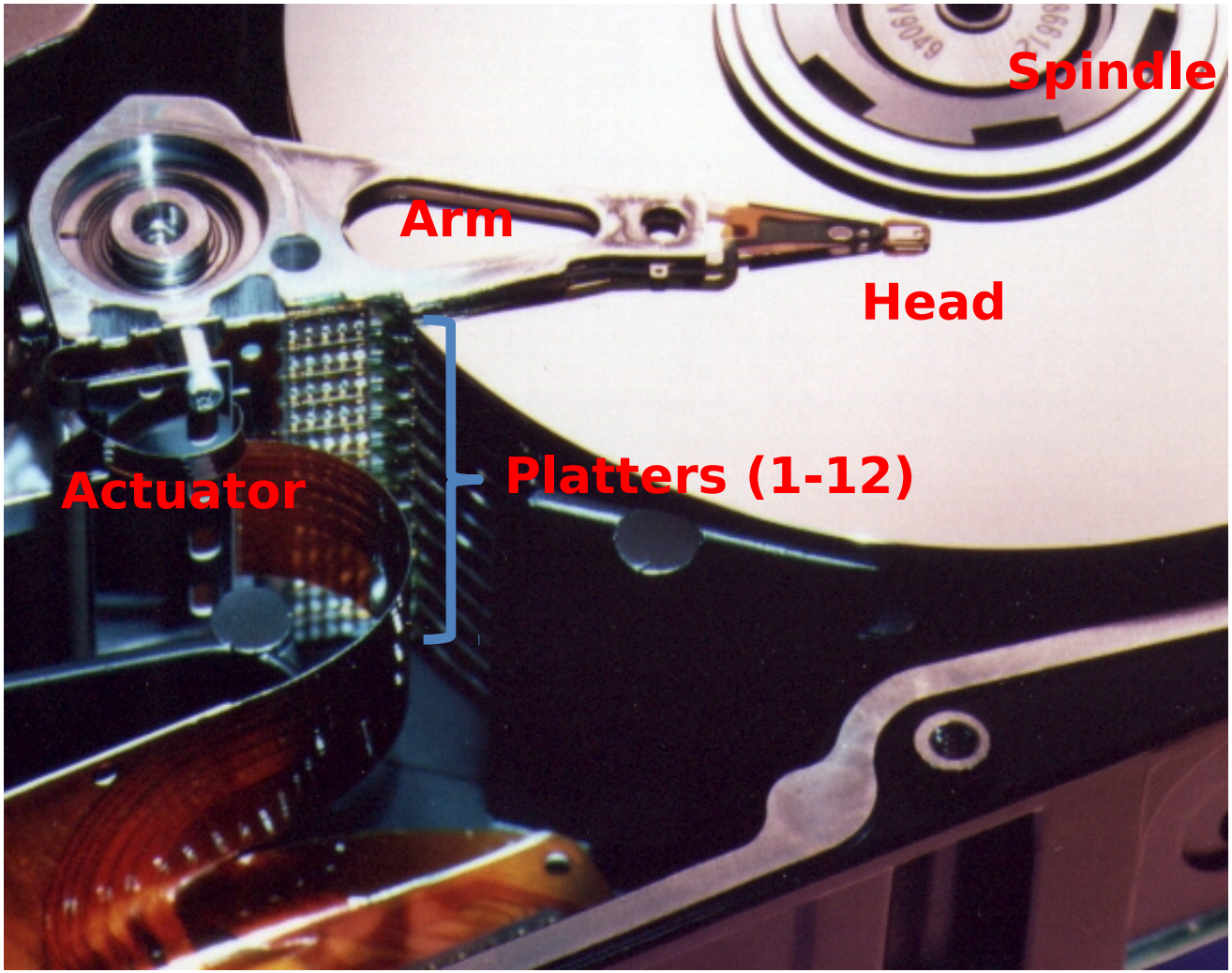
# Agenda

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

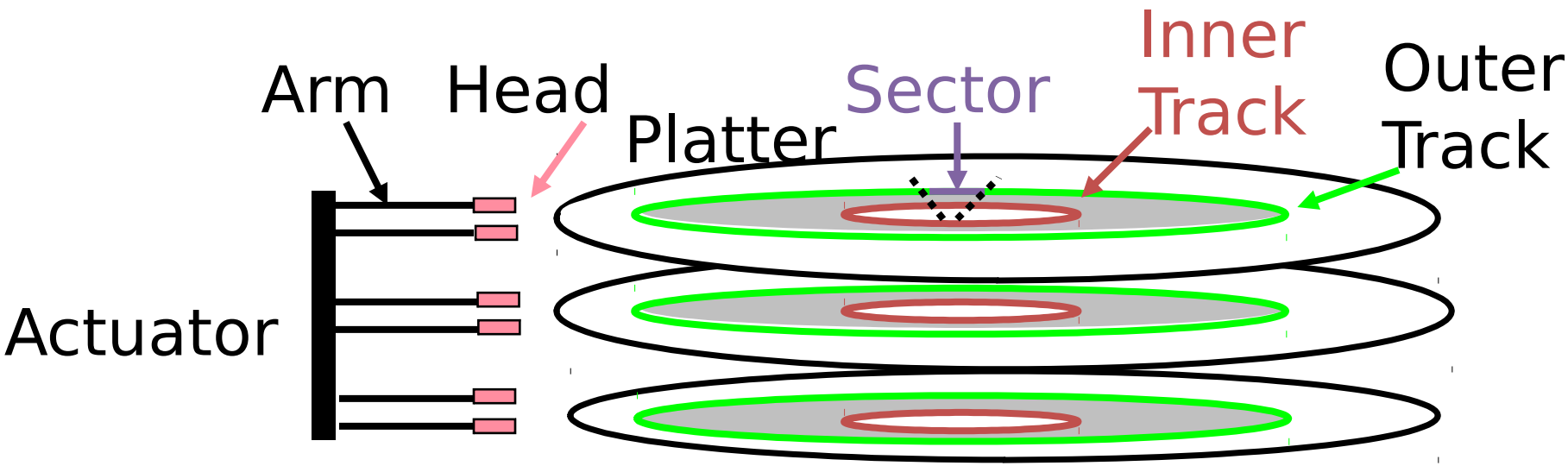
# Magnetic Disks

- 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

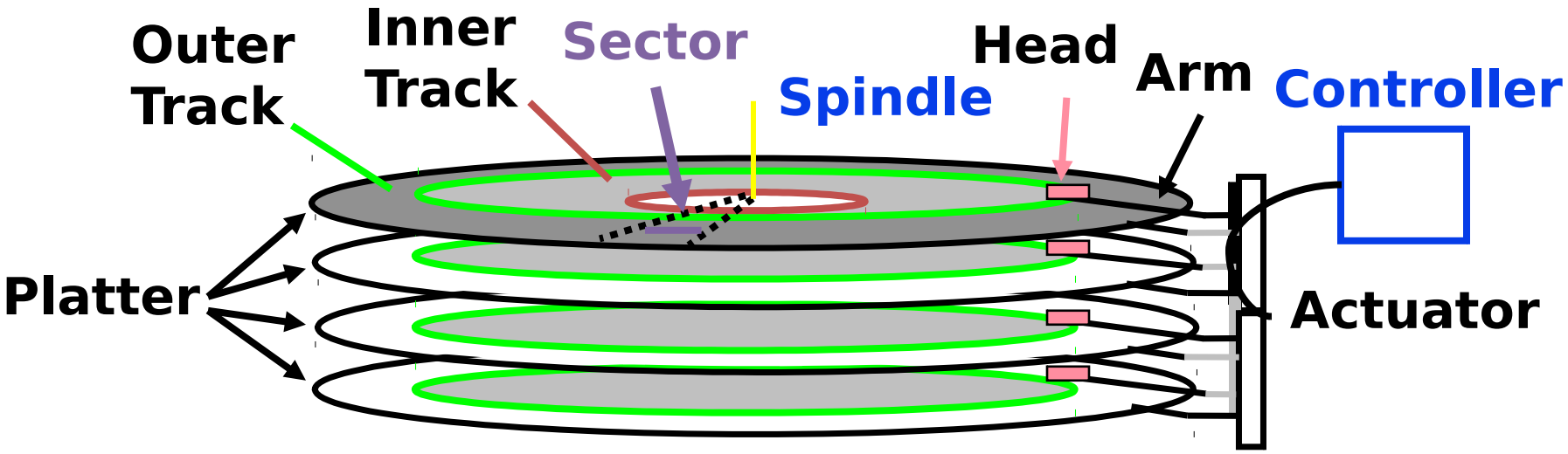


# Disk Device Terminology



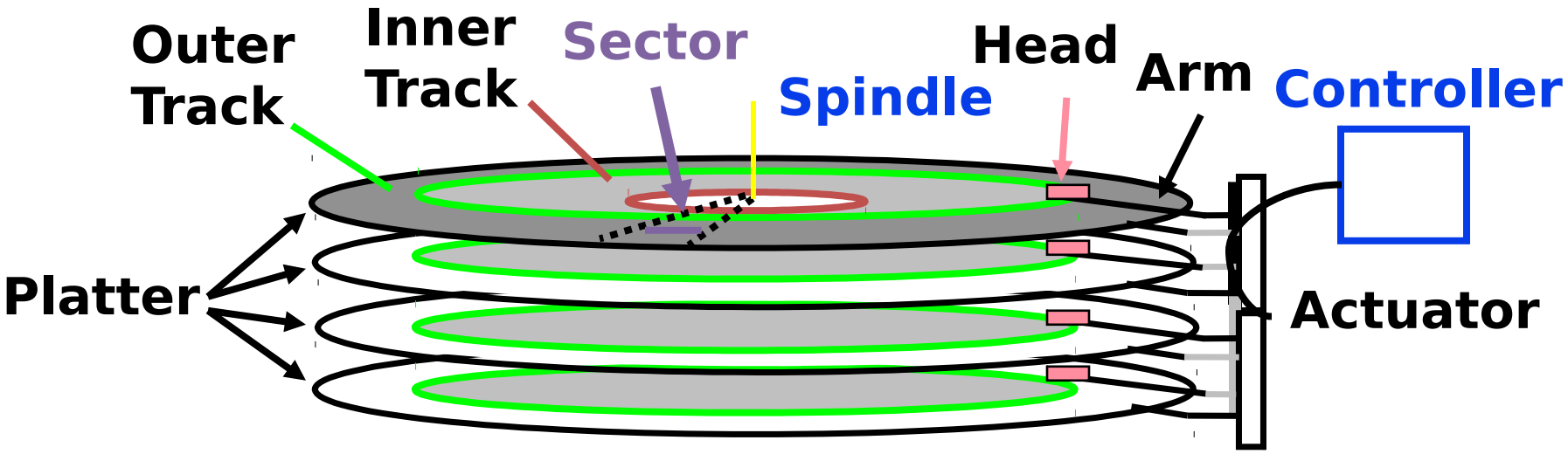

- Several platters, with information recorded magnetically on both surfaces (usually)
- Bits recorded in *tracks*, which in turn are divided into *sectors* (usually 512 Bytes)
- *Actuator* moves *head* (end of *arm*) over track (*"seek"*), wait for sector to rotate under head, then read or write

# Disk Device Performance (1/3)



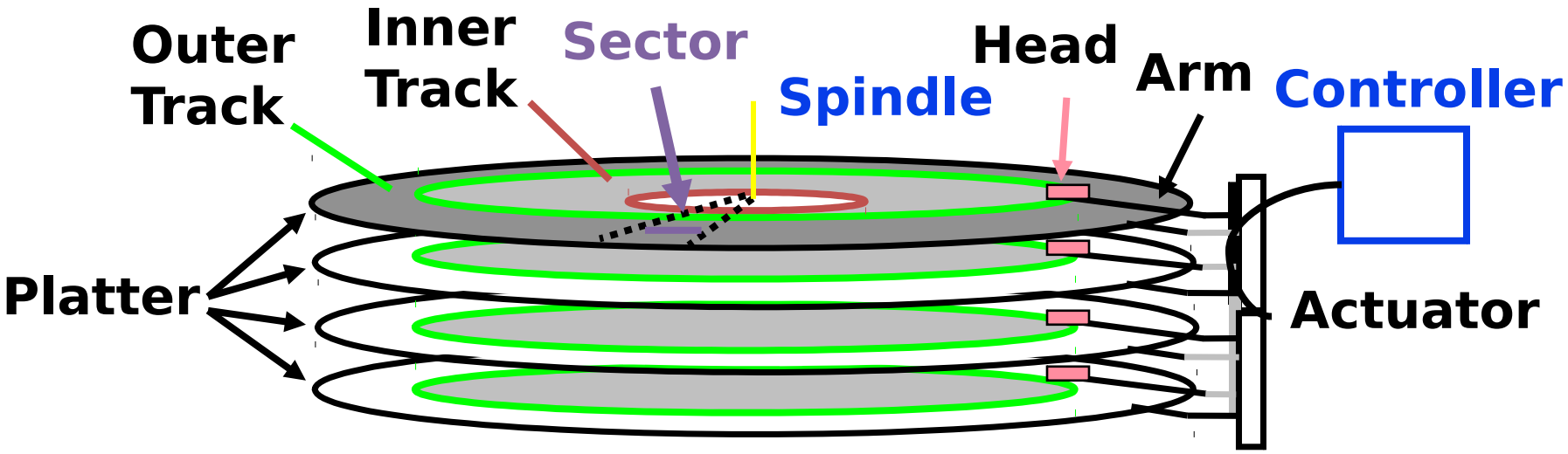

- *Disk Latency* = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  - *Seek Time* depends on number of tracks to move arm and speed of actuator

# Disk Device Performance (1/3)




- *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 Device Performance (1/3)




- *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 ⇒ 1 rev/8.33 ms
    - 1/2 rotation (revolution) ⇒ 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
- 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 ≥ 100K, some ≥ 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**

# Solid-State Drives

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

| | HDD | Flash-based SSD |
|---|---|---|
| Access Time | ~ 12 ms<br>≈ 30M clock cycles | ~ 0.1 ms<br>≈ 250K clock cycles |
| Relative Power | 1 | 1/3 |
| Cost | ~ $0.05-$0.10 / GB | ~ $0.65 / GB |

- 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

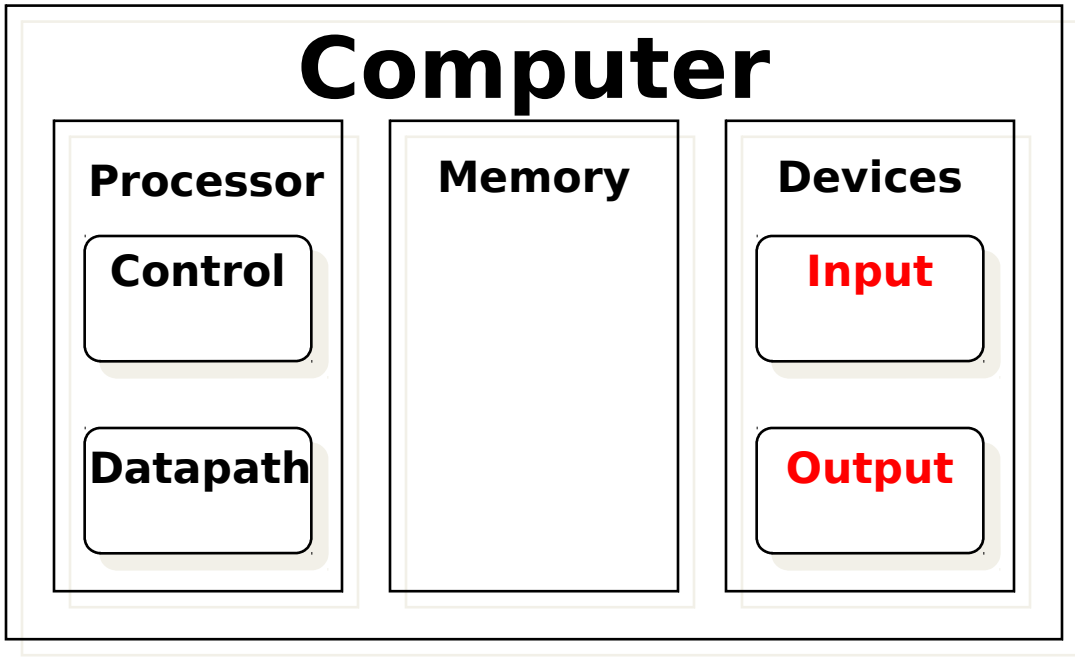
- Final – Fri 8/15, 9am-12pm, 155 Dwinelle
  - MIPS Green Sheet provided again
  - Two two-sided handwritten cheat sheets
    - Can re-use your midterm cheat sheet!
- "Dead Week" – WTh 8/14-15
- HKN Course Survey at end of lecture Wednesday (8/13)
- My office hours this week are 10am-5pm Th
- No Lab Thursday
- All grades entered by this Saturday
  - All regrade requests must be made by no later than Sunday

# 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

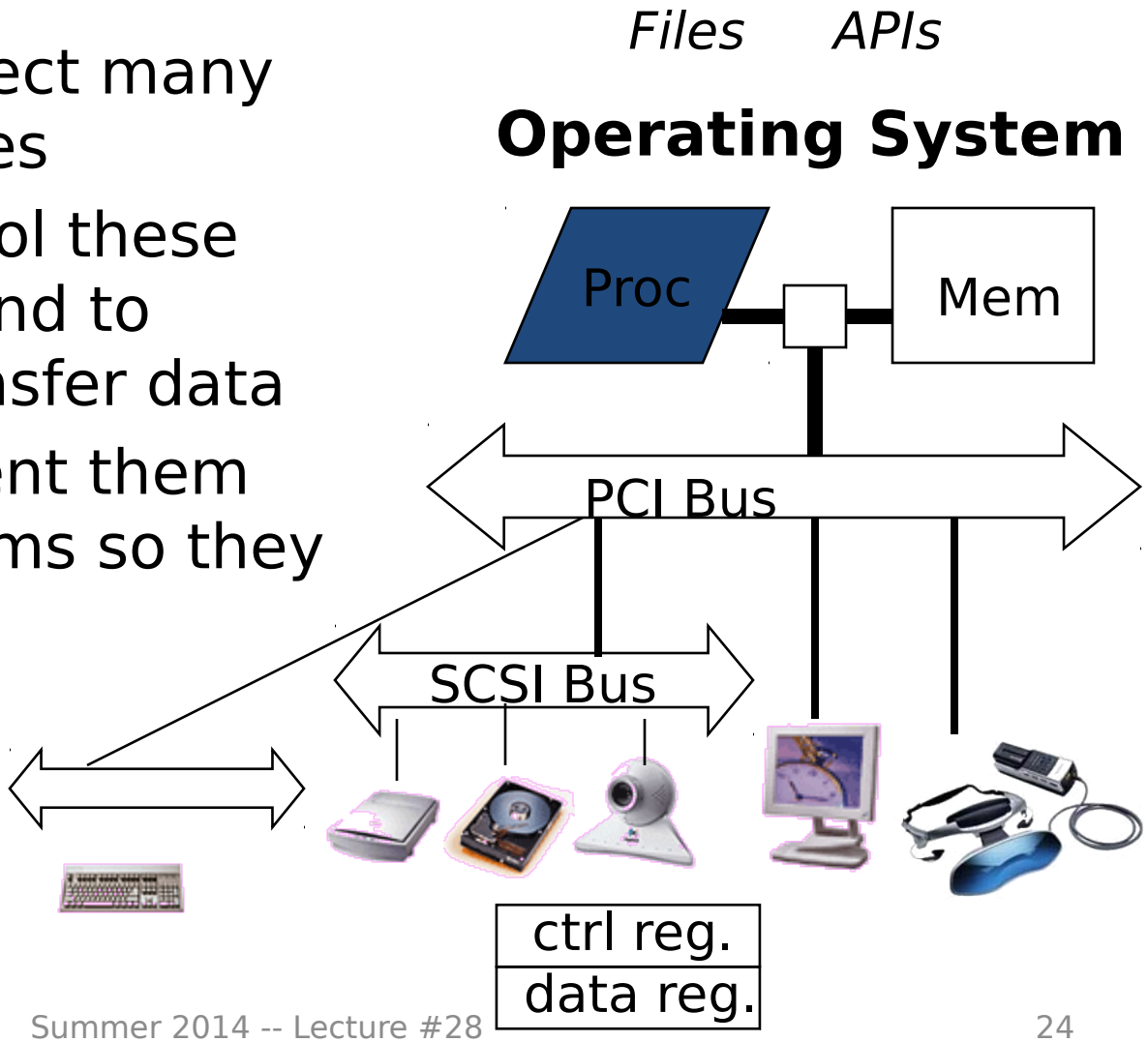
- I/O speeds: *7 orders of magnitude* between mouse and LAN

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

- When discussing transfer rates, use SI prefixes (10x), *not* IEC

# 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

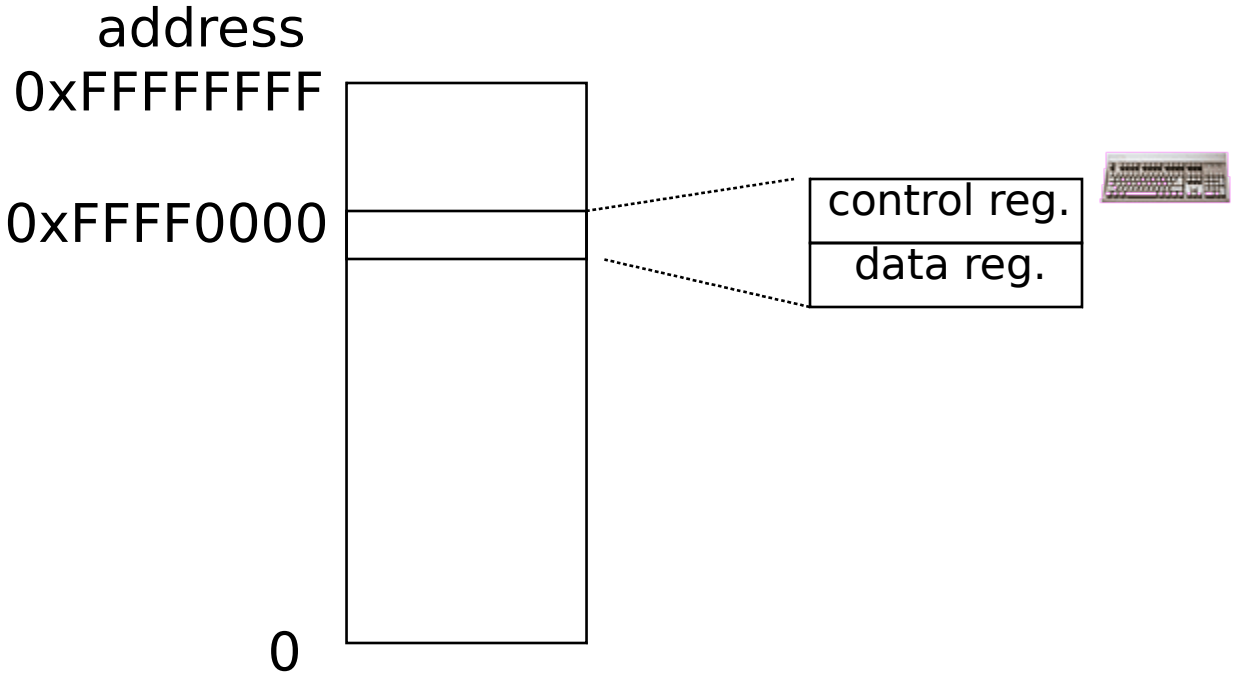


# Instruction Set Architecture for I/O

- 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



# 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 [polls/s] × 400 [clocks/poll] = 12K [clocks/s]
  - *% Time:* $1.2 \times 10^4$ [clocks/s] / $10^9$ [clocks/s] = 0.0012%
  - Polling mouse has little impact on processor
- Disk polling:
  - *Freq:* 16 [MB/s] / 16 [B/poll] = 1M [polls/s]
  - *Time taken:* 1M [polls/s] × 400 [clocks/poll] = 400M [clocks/s]
  - *% Time:* $4 \times 10^8$ [clocks/s] / $10^9$ [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

# Technology Break

# 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*

# 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)

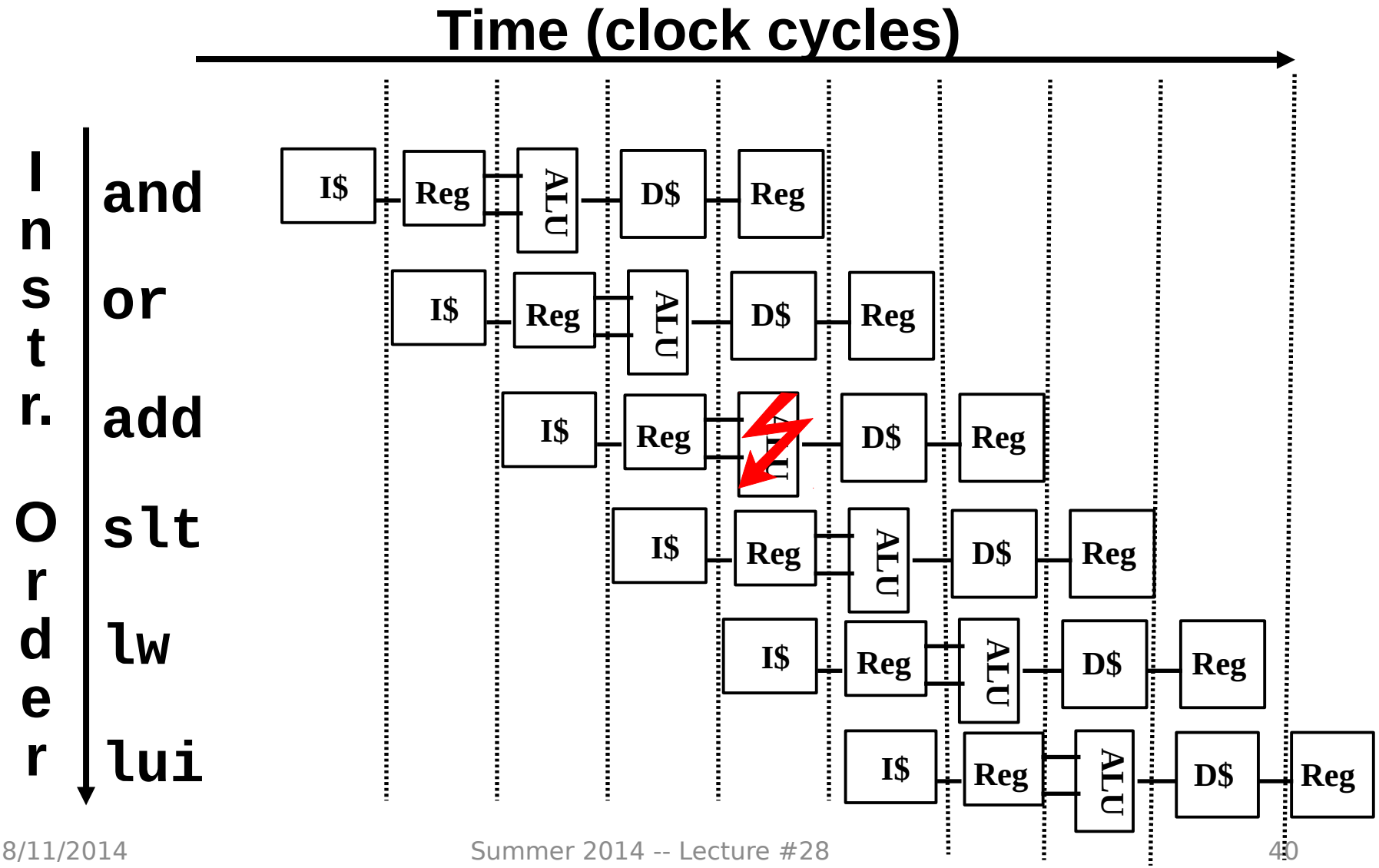
# Exceptions in a Pipeline

- Another kind of control hazard
- Consider overflow on add in EX stage

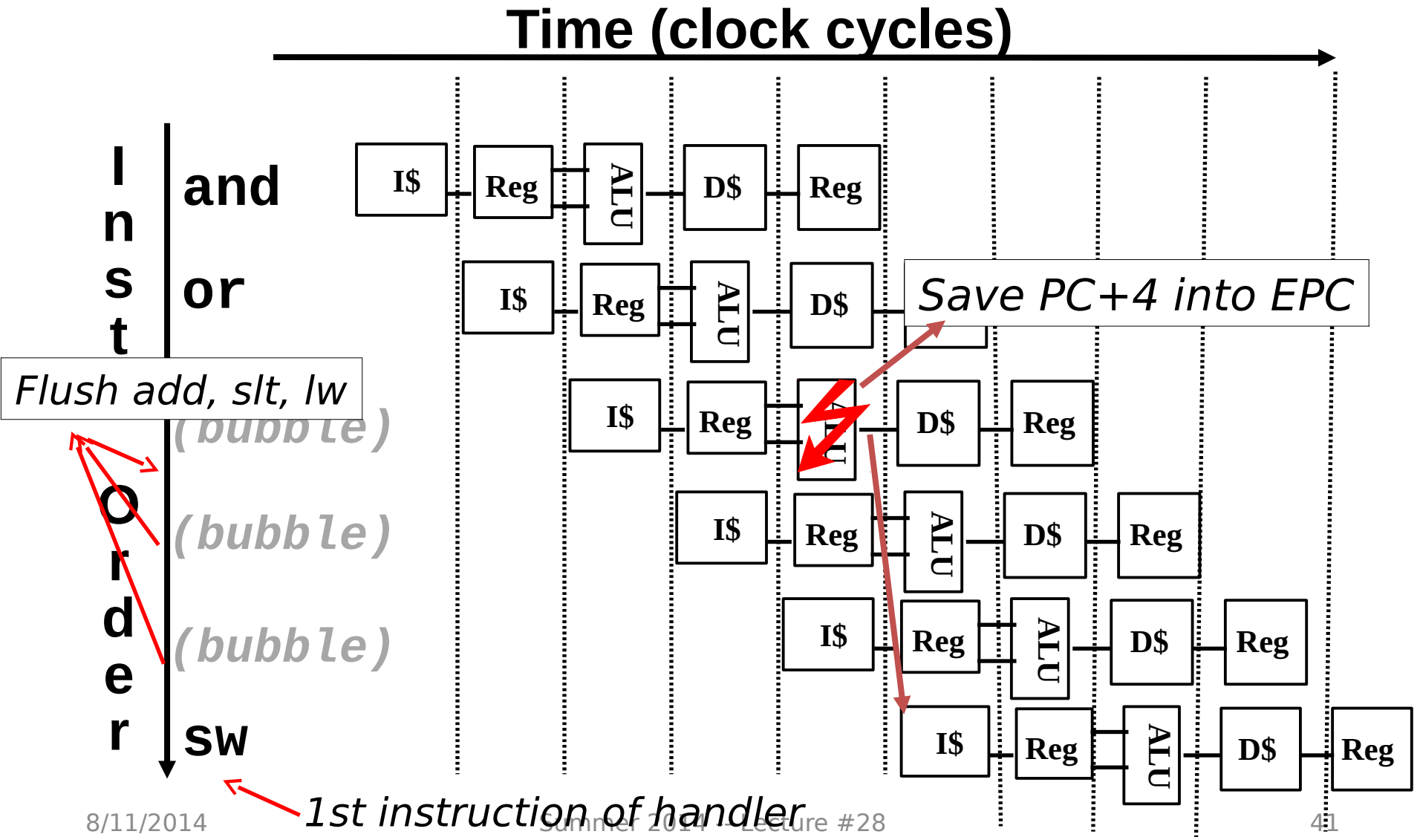
```
add $1, $2, $1
```

  - Prevent `$1` from being clobbered
  - Complete previous instructions
  - Flush `add` and subsequent instructions
  - Set Cause and EPC register values
  - Transfer control to handler
- Similar to mispredicted branch
  - Use much of the same hardware

# Exception Example



# Exception Example



# Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once!
  - e.g. page fault in `lw` the same clock cycle as overflow of following instruction `add`
- **Simple approach:** Deal with exception from *earliest* instruction and flush subsequent instructions
  - Called *precise exceptions*
  - In previous example, service `lw` exception first
- What about multiple issue or out-of-order execution?
  - Maintaining precise exceptions can be difficult!

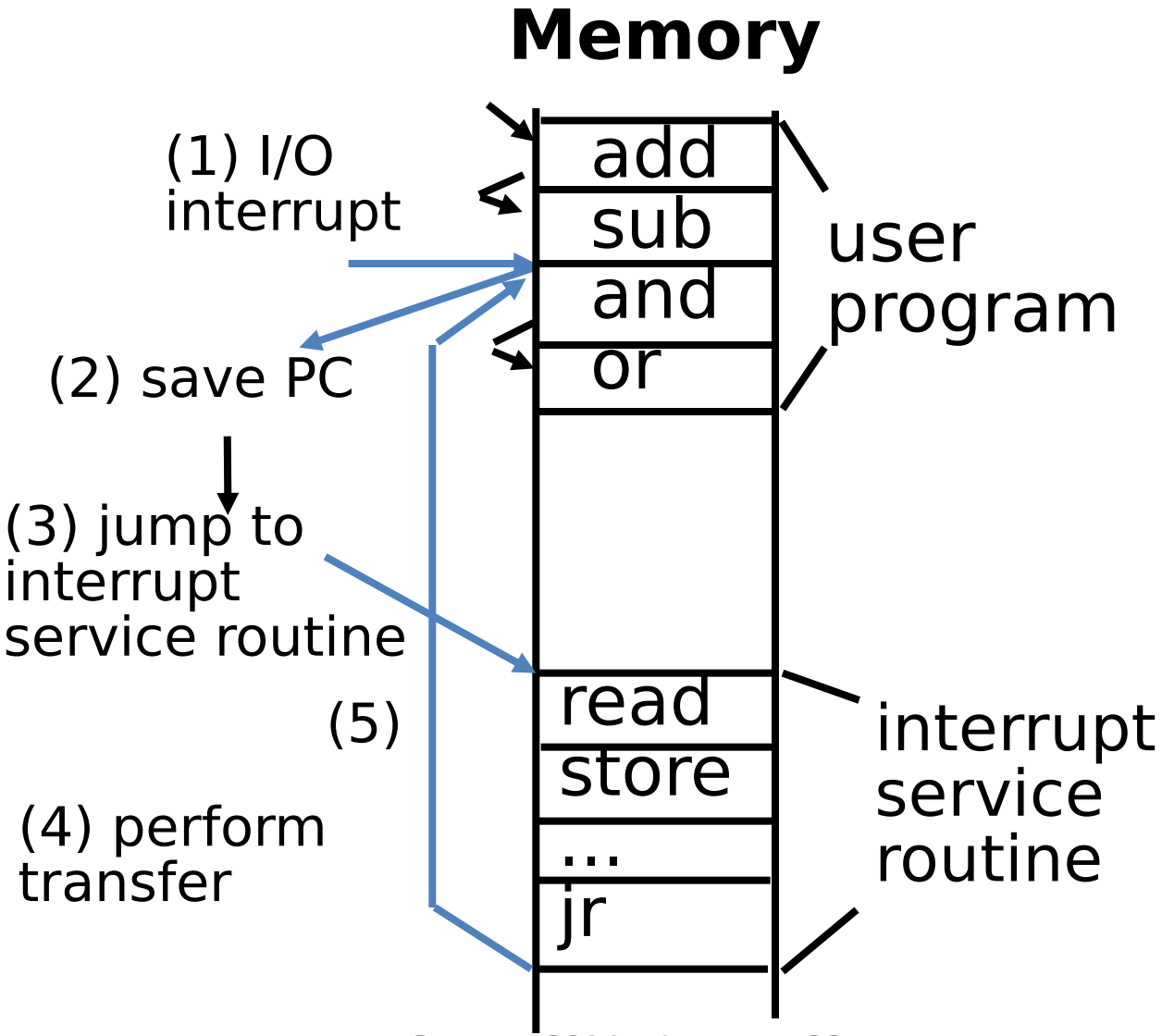
# Imprecise Exceptions

- 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



# 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] = $2.5 \times 10^7$ [clocks/s]
  - $2.5 \times 10^7$ [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

# 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 can be 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