

inst.eecs.berkeley.edu/~cs61c  
**CS61C : Machine Structures**

**Lecture 37**  
**Input / Output**



**Lecturer PSOE Dan Garcia**

[www.cs.berkeley.edu/~ddgarcia](http://www.cs.berkeley.edu/~ddgarcia)

**Email “poses threat to IQ”?! ⇒**

**UK researchers have found a link between the distractions of work-related email usage and lower IQ scores. The IQ drops were up to 10 points, which compares to only 4 points from weed!**



[www.cnn.com/2005/WORLD/europe/04/22/text.iq/](http://www.cnn.com/2005/WORLD/europe/04/22/text.iq/)  
CS61C L37 I/O (1)

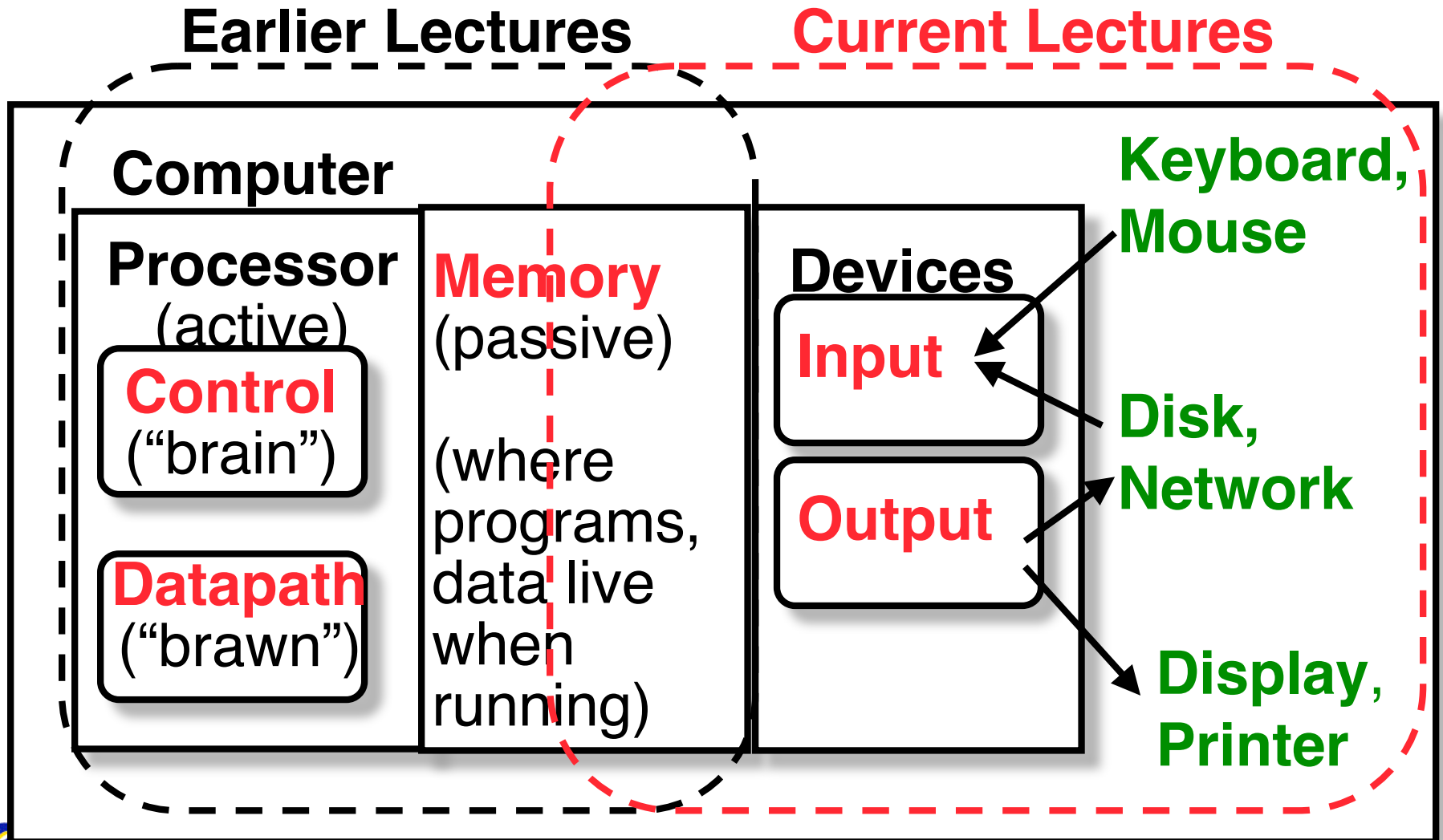
# Review

---

- **Virtual memory to Physical Memory Translation too slow?**
  - Add a cache of Virtual to Physical Address Translations, called a **TLB**
- **Spatial Locality means Working Set of Pages is all that must be in memory for process to run fairly well**
- **Virtual Memory allows protected sharing of memory between processes with less swapping to disk**



# Recall : 5 components of any Computer



# 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:



- Read pressure of synthetic hand and control synthetic arm and hand of fireman



- Control propellers, fins, communicate in BOB (Breathable Observable Bubble)

- Computer without I/O like a car without wheels; great technology, but won't get you anywhere



# I/O Device Examples and Speeds

---

- **I/O Speed: bytes transferred per second**  
(from mouse to Gigabit LAN: 10-million-to-1)

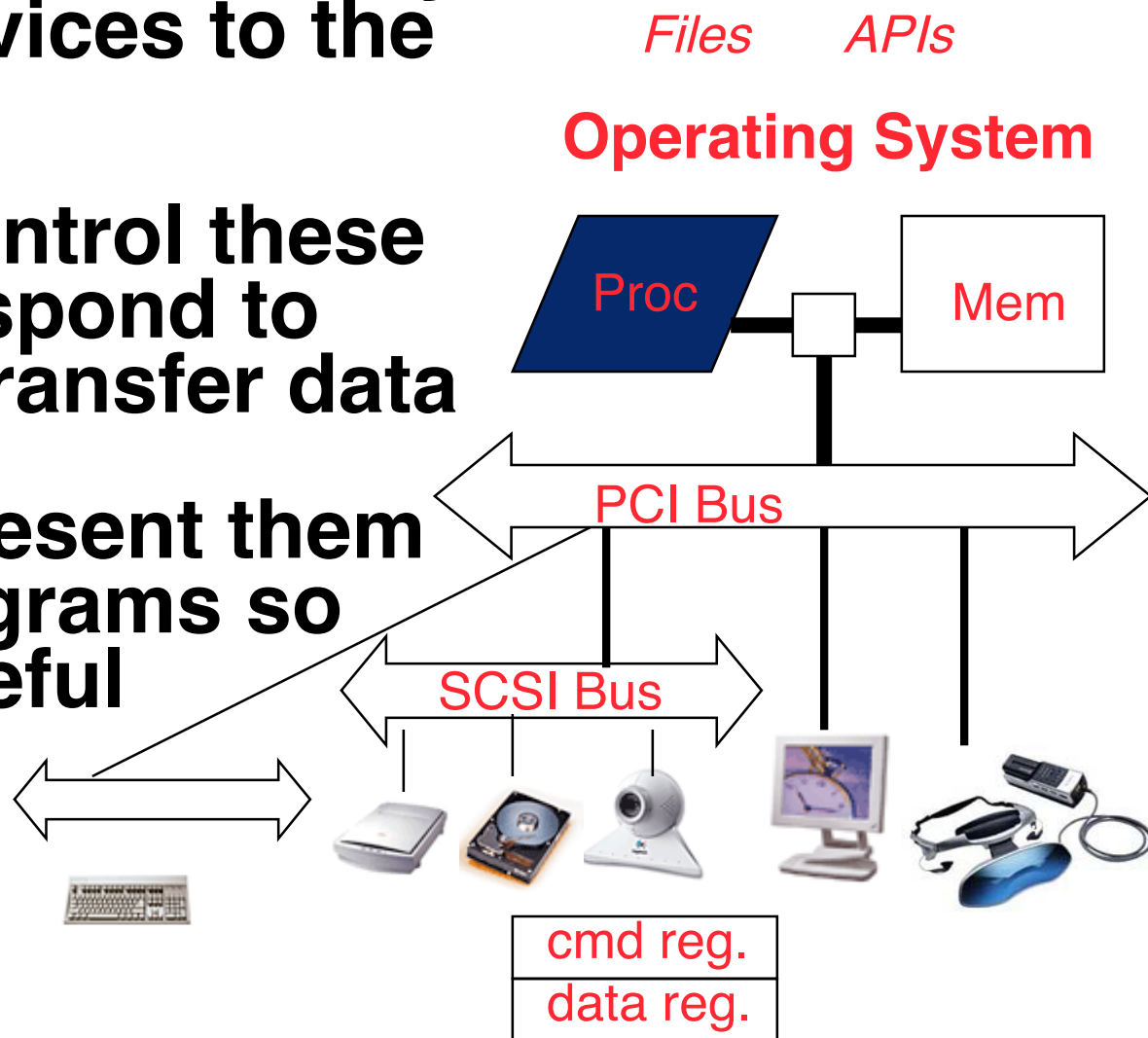
• Device	Behavior	Partner	Data Rate (KBytes/s)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Voice output	Output	Human	5.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Wireless Network	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00
Wired LAN Network	I or O	Machine	125,000.00



When discussing transfer rates, use  $10^x$

# What do we need to make I/O work?

- A way to connect many types of devices to the Proc-Mem
- A way to control these devices, respond to them, and transfer data
- A way to present them to user programs so they are useful



# Instruction Set Architecture for I/O

---

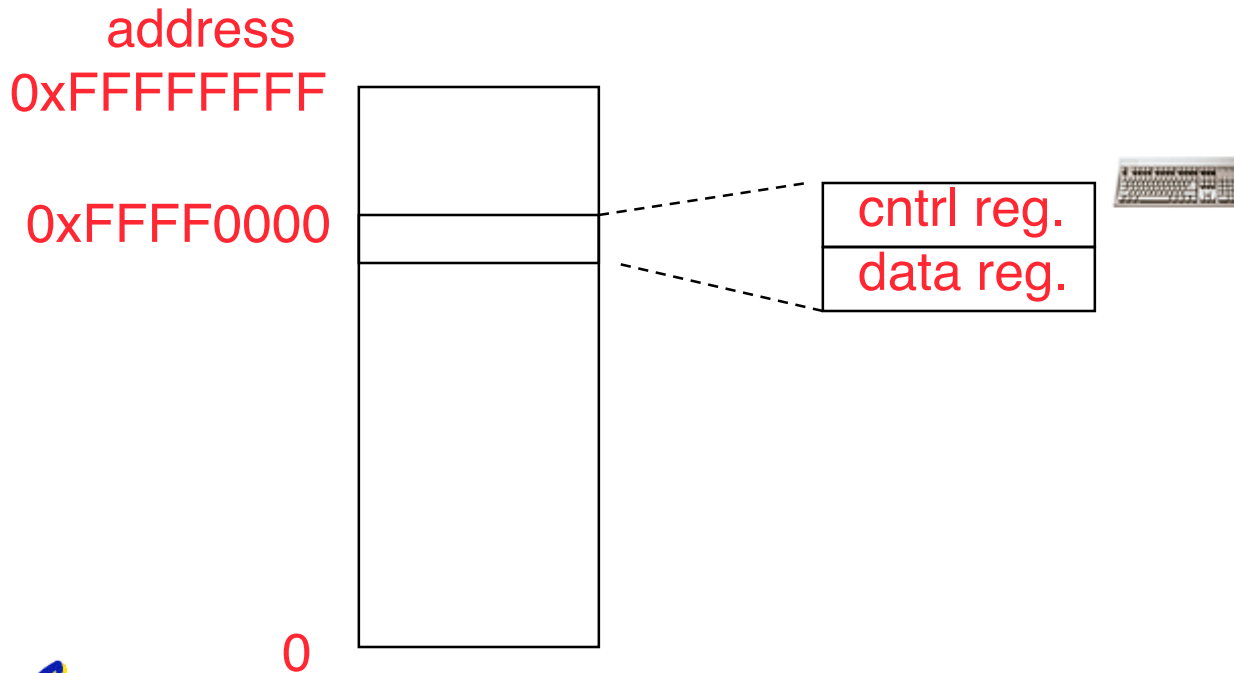
- **What must the processor do for I/O?**
  - **Input:** reads a sequence of bytes
  - **Output:** writes a sequence of bytes
- **Some processors have special input and output instructions**
- **Alternative model (used by MIPS):**
  - **Use loads for input, stores for output**
  - **Called “Memory Mapped 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

---

- **1GHz microprocessor can execute 1 billion load or store instructions per second, or 4,000,000 KB/s data rate**
  - **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 not be ready to accept data as fast as processor stores it**

• **What to do?**



# Processor Checks Status before Acting

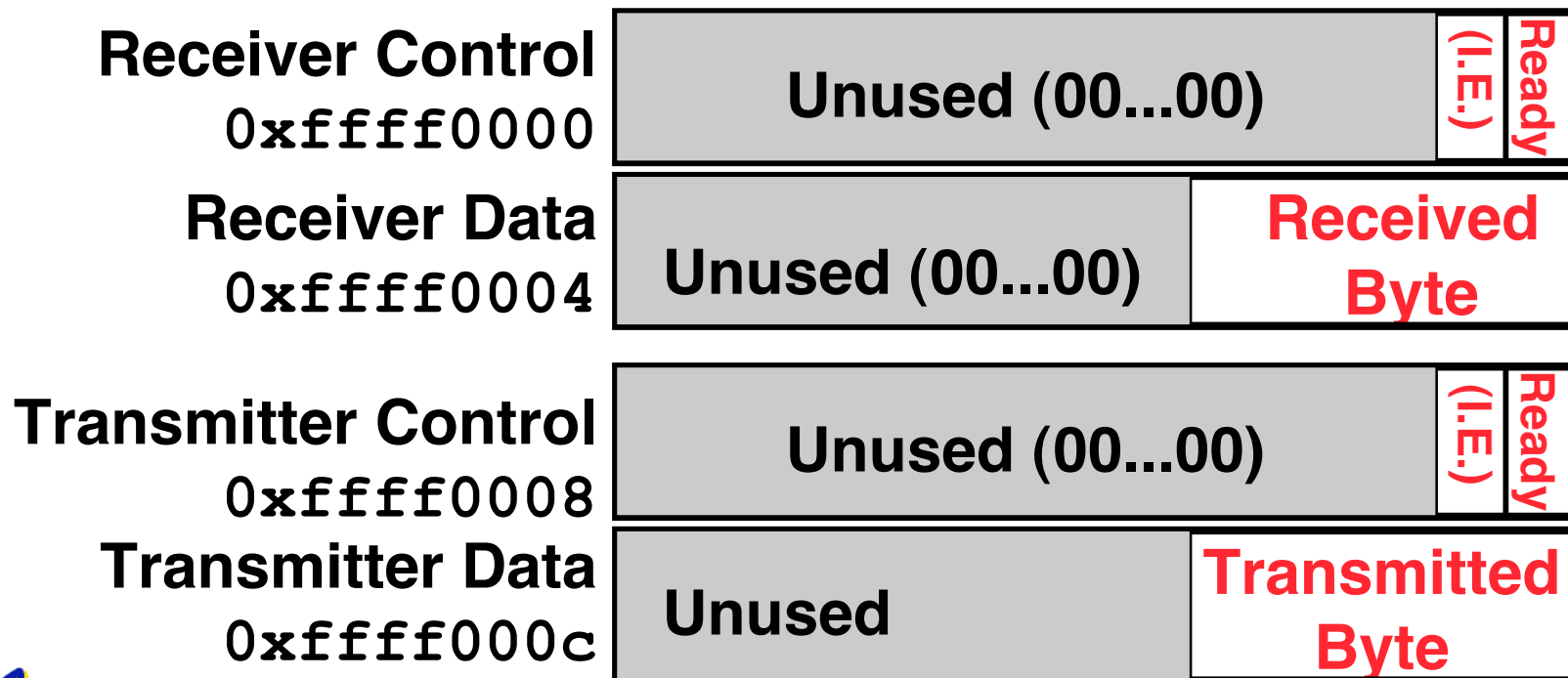
---

- Path to device generally has 2 registers:
  - **Control Register**, says it's OK to read/write (I/O ready) [think of a flagman on a road]
  - **Data Register**, contains data
- Processor reads from Control Register in loop, waiting for device to set **Ready** bit in Control reg ( $0 \Rightarrow 1$ ) to say its OK
- Processor then loads from (input) or writes to (output) data register
  - Load from or Store into Data Register resets Ready bit ( $1 \Rightarrow 0$ ) of Control Register



# SPIM I/O Simulation

- SPIM simulates 1 I/O device: memory-mapped terminal (keyboard + display)
  - Read from keyboard (receiver); 2 device regs
  - Writes to terminal (transmitter); 2 device regs



# SPIM I/O

---

- **Control register rightmost bit (0): Ready**
  - **Receiver: Ready==1 means character in Data Register not yet been read;  
1  $\Rightarrow$  0 when data is read from Data Reg**
  - **Transmitter: Ready==1 means transmitter is ready to accept a new character;  
0  $\Rightarrow$  Transmitter still busy writing last char**
    - I.E. bit discussed later
- **Data register rightmost byte has data**
  - **Receiver: last char from keyboard; rest = 0**
  - **Transmitter: when write rightmost byte, writes char to display**



# I/O Example

---

- Input: Read from keyboard into \$v0

```
Waitloop:      lui    $t0, 0xffff #ffff0000
               lw     $t1, 0($t0) #control
               andi  $t1, $t1, 0x1
               beq   $t1, $zero, Waitloop
               lw    $v0, 4($t0) #data
```

- Output: Write to display from \$a0

```
Waitloop:      lui    $t0, 0xffff #ffff0000
               lw     $t1, 8($t0) #control
               andi  $t1, $t1, 0x1
               beq   $t1, $zero, Waitloop
               sw   $a0, 12($t0) #data
```

- Processor waiting for I/O called “**Polling**”
- “Ready” bit from processor’s point of view!



# Administrivia

---

- **Only 6 lectures to go (after this one)! :(**
- **Dan will be attending the Faculty retreat on Thursday & Friday**
  - **This is where we decide who will be joining our faculty soon!**
  - **He won't be holding his thurs office hours**



# Cost of Polling?

---

- **Assume for a processor with a 1GHz clock it takes 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: transfers data in 2-Byte units and has a data rate of 50 KB/second. 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 [p. 677 in book]

### Mouse Polling, Clocks/sec

$$= 30 \text{ [polls/s]} * 400 \text{ [clocks/poll]} = 12\text{K} \text{ [clocks/s]}$$

- % Processor for polling:

$$12 * 10^3 \text{ [clocks/s]} / 1 * 10^9 \text{ [clocks/s]} = 0.0012\%$$

⇒ Polling mouse little impact on processor

### Frequency of Polling Floppy

$$= 50 \text{ [KB/s]} / 2 \text{ [B/poll]} = 25\text{K} \text{ [polls/s]}$$

- Floppy Polling, Clocks/sec

$$= 25\text{K} \text{ [polls/s]} * 400 \text{ [clocks/poll]} = 10\text{M} \text{ [clocks/s]}$$

- % Processor for polling:

$$10 * 10^6 \text{ [clocks/s]} / 1 * 10^9 \text{ [clocks/s]} = 1\%$$



⇒ OK if not too many I/O devices



# % Processor time to poll hard disk

## Frequency of Polling Disk

$$= 16 \text{ [MB/s]} / 16 \text{ [B]} = 1\text{M [polls/s]}$$

- **Disk Polling, Clocks/sec**

$$= 1\text{M [polls/s]} * 400 \text{ [clocks/poll]}$$

$$= 400\text{M [clocks/s]}$$

- **% Processor for polling:**

$$400 * 10^6 \text{ [clocks/s]} / 1 * 10^9 \text{ [clocks/s]} = 40\%$$

⇒ **Unacceptable**



# What is the alternative 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 I/O. **Interrupt** program when I/O ready, return when done with data transfer



# I/O Interrupt

---

- **An I/O interrupt is like overflow exceptions except:**
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- **An I/O interrupt is 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 completion**



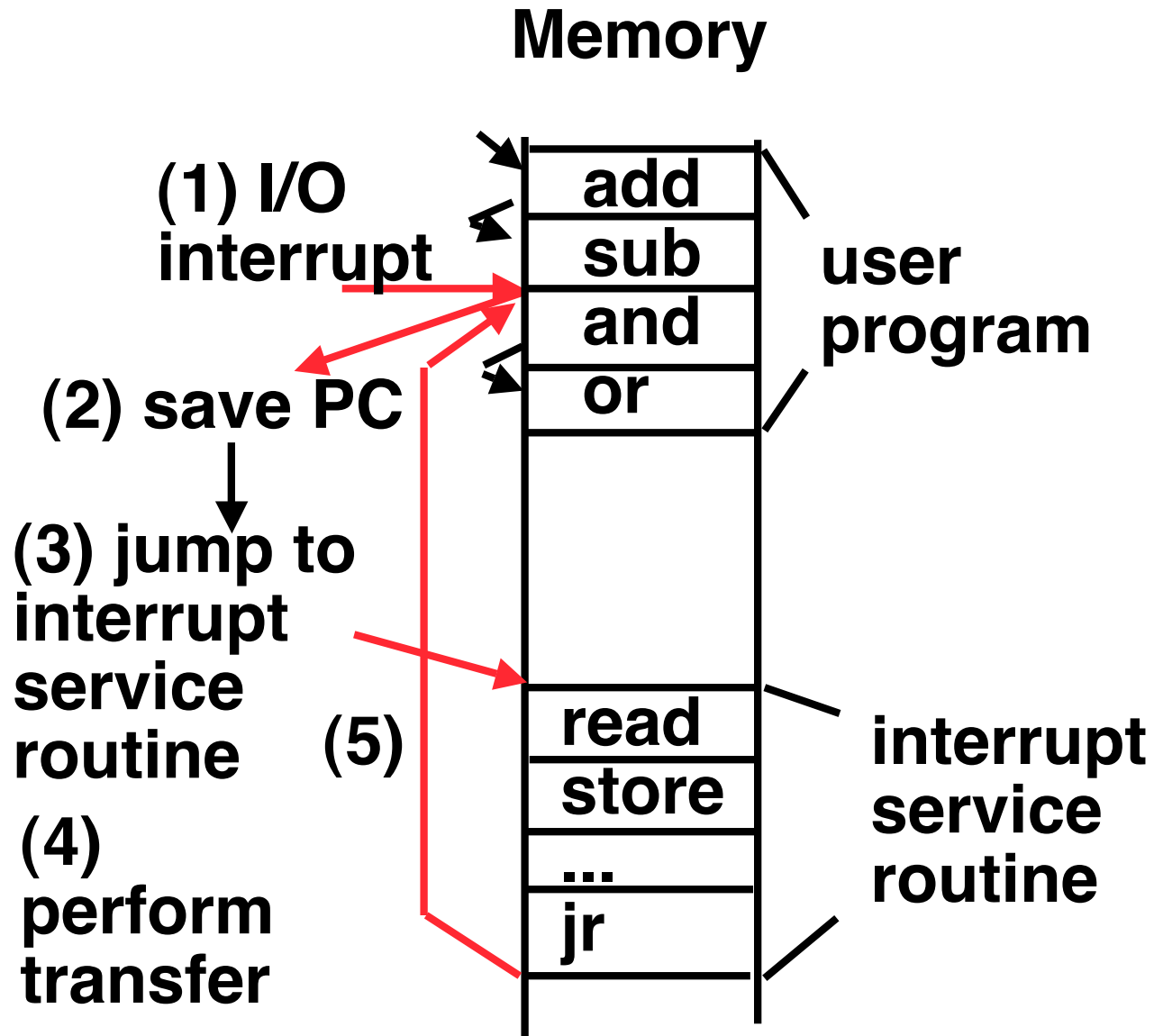
# Definitions for Clarification

---

- **Exception**: signal marking that something “out of the ordinary” has happened and needs to be handled
- **Interrupt**: asynchronous exception
- **Trap**: synchronous exception
- **Note**: Many systems folks say “interrupt” to mean what we mean when we say “exception”.



# Interrupt Driven Data Transfer



# SPIM I/O Simulation: Interrupt Driven I/O

- I.E. stands for Interrupt Enable
- Set Interrupt Enable bit to 1 have interrupt occur whenever Ready bit is set

Receiver Control 0xffff0000	Unused (00...00)	(I.E.)	Ready
Receiver Data 0xffff0004	Unused (00...00)	Received Byte	
Transmitter Control 0xffff0008	Unused (00...00)	(I.E.)	Ready
Transmitter Data 0xffff000c	Unused	Transmitted Byte	



# Benefit of Interrupt-Driven I/O

---

- Find the % of processor consumed if the hard disk is only active 5% of the time. Assuming 500 clock cycle overhead for each transfer, including interrupt:
  - Disk Interrupts/s =  $16 \text{ MB/s} / 16\text{B}/\text{interrupt} = 1\text{M interrupts/s}$
  - Disk Interrupts, clocks/s =  $1\text{M interrupts/s} * 500 \text{ clocks/interrupt} = 500,000,000 \text{ clocks/s}$
  - % Processor for during transfer:  $500 * 10^6 / 1 * 10^9 = 50\%$
- Disk active 5%  $\Rightarrow 5\% * 50\% \Rightarrow 2.5\% \text{ busy}$



# Peer Instruction

---

- A. A faster CPU will result in faster I/O.
- B. Hardware designers handle mouse input with interrupts since it is better than polling in almost all cases.
- C. Low-level I/O is actually quite simple, as it's really only reading and writing bytes.

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TF
8:	TTT





# Peer Instruction Answer

---

A. Less sync data idle time

B. Because mouse has low I/O rate polling often used

C. Concurrency, device requirements vary!

**TRUE**

A. A faster CPU will result in faster I/O.

B. Hardware designers handle mouse input with interrupts since it is better than polling in almost all cases.

C. Low-level I/O is actually quite simple, as it's really only reading and writing bytes.

	ABC
1:	FFF
2:	FFT
3:	FTF
4:	FTT
5:	TFF
6:	TFT
7:	TF
8:	TTT

**FALSE**

**FALSE**



## “And in conclusion...”

---

- I/O gives computers their 5 senses
- I/O speed range is 100-million to one
- Processor speed means must synchronize with I/O devices before use
- Polling works, but expensive
  - processor repeatedly queries devices
- Interrupts works, more complex
  - devices causes an exception, causing OS to run and deal with the device
- I/O control leads to **Operating Systems**

