

## COMP 3221

### Microprocessors and Embedded Systems

#### Lectures 26: I/O Interfacing

<http://www.cse.unsw.edu.au/~cs3221>

September, 2003

Saeid Nooshabadi

saeid@unsw.edu.au

Some of the slides are adopted from David Patterson (UCB)

COMP3221 lec26-io.1

Saeid Nooshabadi

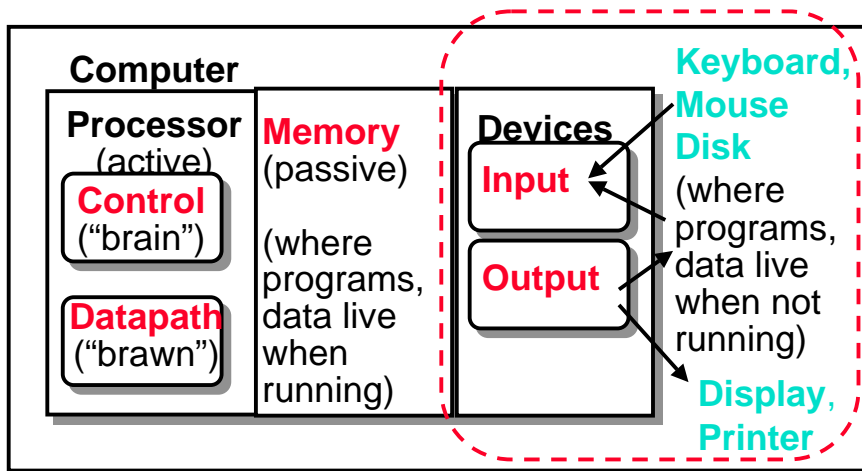
## Overview

- I/O Background
- Polling
- Interrupts

COMP3221 lec26-io.2

Saeid Nooshabadi

## Anatomy: 5 components of any Computer



COMP3221 lec26-io.3

Saeid Nooshabadi

## Motivation for Input/Output

- I/O is how humans interact with computers
- 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)
  - Read bar codes of items in refrigerator
- Computer without I/O like a car without wheels; great technology, but won't get you anywhere



COMP3221 lec26-io.4

Saeid Nooshabadi

## I/O Device Examples and Speeds

- ° I/O Speed: bytes transferred per second (from mouse to display: million-to-1)

Device	Behavior	Partner	Data Rate (Kbytes/sec)
Keyboard	Input	Human	0.01
Mouse	Input	Human	0.02
Line Printer	Output	Human	1.00
Floppy disk	Storage	Machine	50.00
Laser Printer	Output	Human	100.00
Magnetic Disk	Storage	Machine	10,000.00
Network-LAN	I or O	Machine	10,000.00
Graphics Display	Output	Human	30,000.00

COMP3221 lec26-io.5

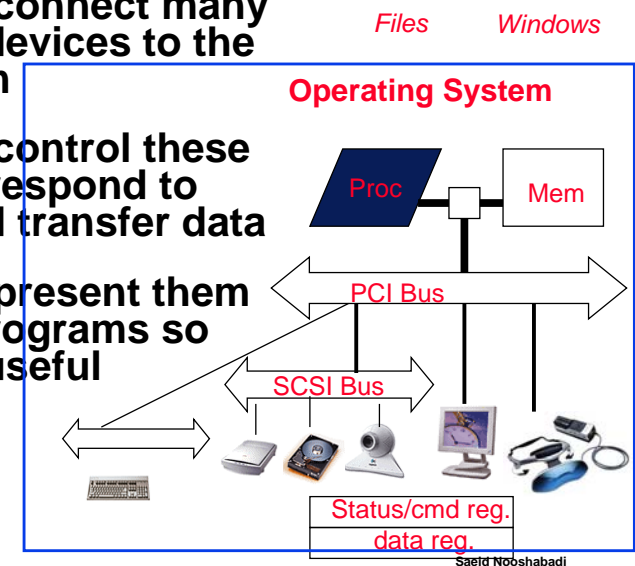
Saeid Nooshabadi

## 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

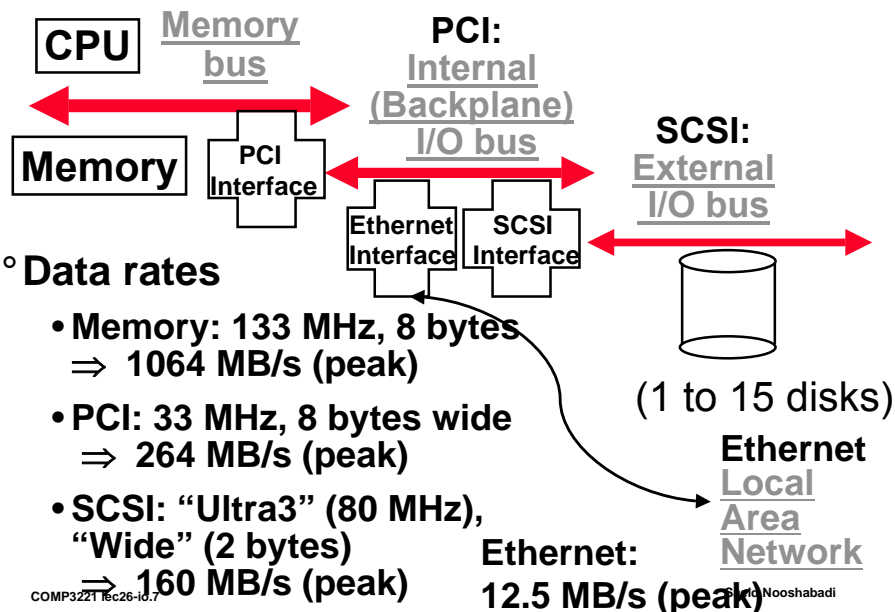
- ° A way to present them to user programs so they are useful



COMP3221 lec26-io.6

Saeid Nooshabadi

## Buses in a PC: Connect a few devices



### Data rates

- Memory: 133 MHz, 8 bytes  $\Rightarrow$  1064 MB/s (peak)
- PCI: 33 MHz, 8 bytes wide  $\Rightarrow$  264 MB/s (peak)
- SCSI: "Ultra3" (80 MHz), "Wide" (2 bytes)  $\Rightarrow$  160 MB/s (peak)
- Ethernet: 12.5 MB/s (peak)

COMP3221 lec26-io.7

Saeid Nooshabadi

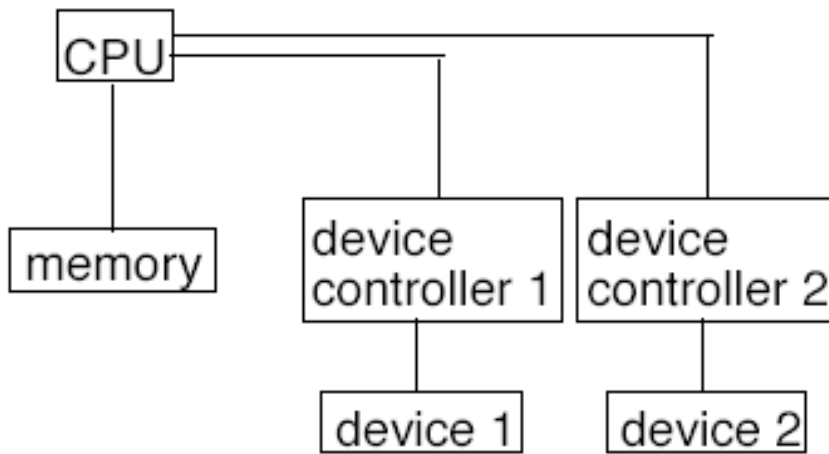
## Instruction Set Architecture for I/O

- ° Some machines have special input and output instructions
- ° Alternative model (used by ARM):
  - Input: ~ reads a sequence of bytes
  - Output: ~ writes a sequence of bytes
- ° Memory access also reading/ writing a sequence of bytes, so 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)

COMP3221 lec26-io.8

Saeid Nooshabadi

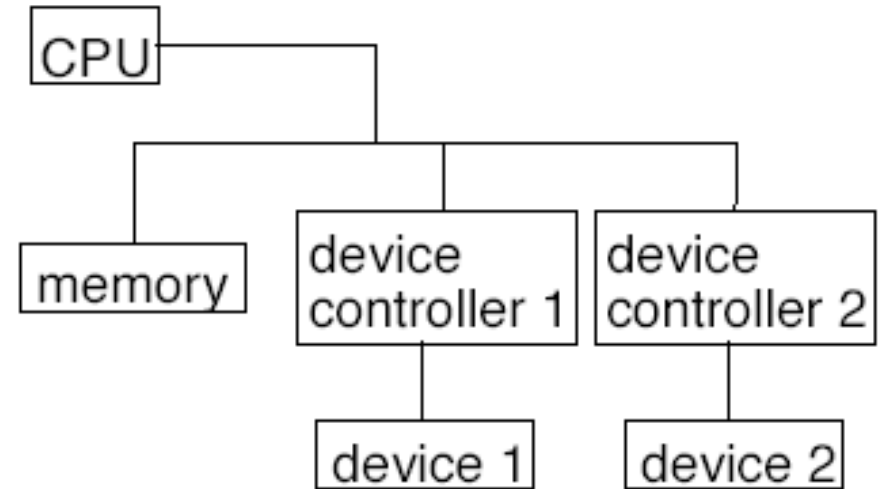
## Computers with Special Instruction for I/O



COMP3221 lec26-io.9

Saeid Nooshabadi

## Computers with Memory Mapped I/O



COMP3221 lec26-io.10

Saeid Nooshabadi

## When Memory isn't Memory

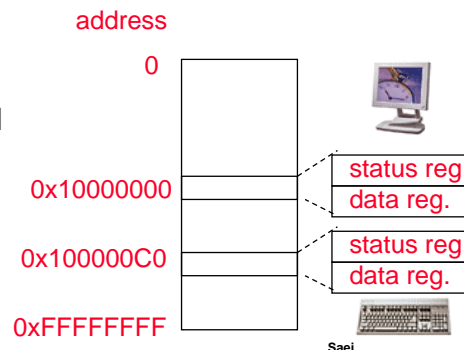
### ° I/O devices often have a few registers

- Status/ Control registers
- I/O registers

### ° If these have an interface that looks like memory, we can connect them to the memory bus

- Reads/Writes to certain locations will produce the desired change in the I/O device controller

### ° Typically, devices map to only a few bytes in memory



COMP3221 lec26-io.11

## Processor-I/O Speed Mismatch

### ° 500 MHz microprocessor can execute 500 million load or store instructions per second, or 2,000,000 KB/s data rate

- I/O devices from 0.01 KB/s to 30,000 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

### ° What to do?

COMP3221 lec26-io.12

Saeid Nooshabadi

## Processor Checks Status before Acting

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

COMP3221 lec26-10.13

Saeid Nooshabadi

## "What's This Stuff Good For?"



Remote Diagnosis:  
 "NeoRest ExII," a high-tech toilet features microprocessor-controlled seat warmers, automatic lid openers, air deodorizers, water sprays and blow-dryers that do away with the need for toilet tissue. About 25 percent of new homes in Japan have a "washlet," as these toilets are called. Toto's engineers are now working on a model that analyzes urine to determine blood-sugar levels in diabetics and then automatically sends a daily report, by modem, to the user's physician.  
*One Digital Day*, 1998  
[www.intel.com/onedigitalday](http://www.intel.com/onedigitalday)

COMP3221 lec26-10.14

Saeid Nooshabadi

## DSLMU/Komodo Address Space

Start Address	End Address	Size	Function
0x00000000 (RAM)	0x003FFFFFFF	4 MB	Read/write memory
0x00400000	0x0FFFFFFF	(252 MB)	(Unused)
0x10000000 Microcontroller I/O space	0x1FFFFFFF	256 MB	
0x20000000 Spartan space	0x2FFFFFFF	256 MB	Xilinx -XL I/O
0x30000000 Virtex-E space	0x3FFFFFFF	256 MB	Xilinx I/O
0x40000000	0xFFFFFFFF	(3072 MB)	(Unused)

COMP3221 lec26-10.15

Saeid Nooshabadi

## DSLMU I/Os

- Two RS232 serial port connectors
- LEDs on the MU Board,
- Boot Select switches
- LCD module
- Spartan-XL FPGA for I/O Expansion
- Virtex-E FPGA for Co-processing
- Single-chip 10 Mb Ethernet
- Uncommitted Peripherals
- Timers
- **Ref: Hardware Ref Manual on CD-ROM.**

COMP3221 lec26-10.16

Saeid Nooshabadi

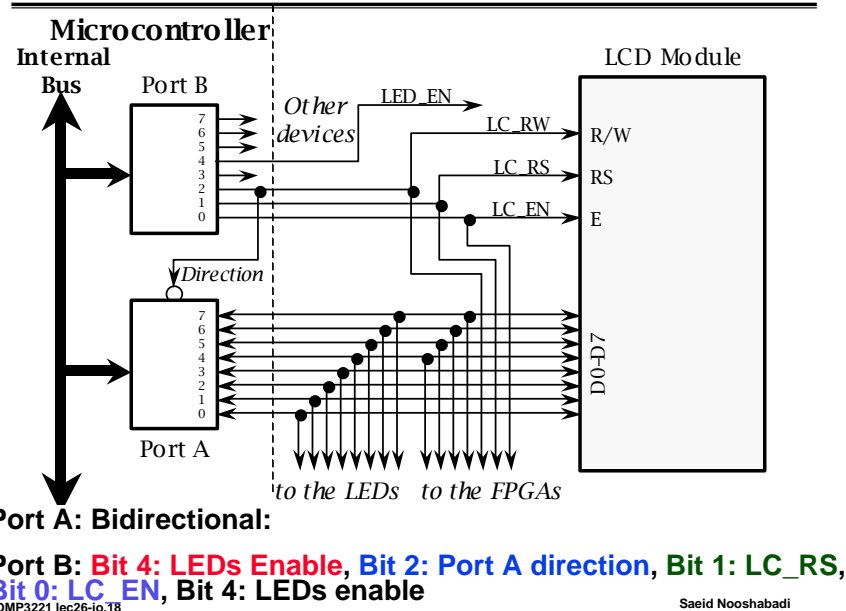
## DSLMU/Komodo I/O Addressing

Offset	Mode	Port Name	Function
0x00	R/W	Port A	Bidirectional data port to LEDs, LCD, etc.
0x04	R/W	Port B	Control port (some bits are read only)
0x08	R/W	Timer	8-bit free-running 1 kHz timer
0x0C	R/W	Timer Compare	Allows timer interrupts to be generated
0x10	RO	Serial Rx/D	Read a byte from the serial port
0x10	WO	Serial Tx/D	Write a byte to the serial port
0x14	WO	Serial Status	Serial port status port
0x18	R/W	IRQ Status	Bitmap of currently-active interrupts
0x1C	R/W	IRQ Enable	Controls which interrupts are enabled
0x20	WO	Debug Stop	Stops program execution when written to

COMP3221 lec26-io.17

Saeid Nooshabadi

## DSLMU/Komodo Ports A & B:



COMP3221 lec26-io.18

Saeid Nooshabadi

## I/O Example

### ° Output: Write to LED Port

```
.set iobase, 0x10000000 ; Base of the
                        ; DSLMU I/O space

.set portA, 0x00       ; Offset of Port A
                        ; in the I/O space

.set portB, 0x04       ; Offset of Port B
                        ; in the I/O space

mov r2, #iobase        ; Use R2 as a base
                        ; address pointer

mov r0, #0b00010000    ; Set bit 4 and reset
                        ; all other bits

strb r0, [r2, #portB]   ; Send the data to
                        ; Port B (R2 + portB)

mov r0, #0b10100101    ; R0 = data for the LEDs

strb r0, [r2, #portA]
```

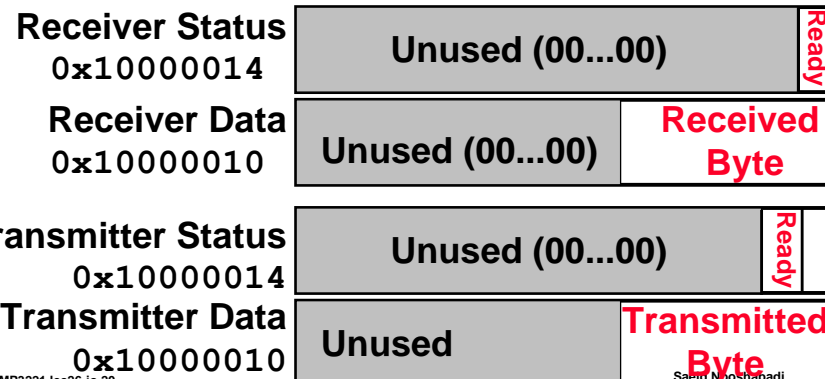
COMP3221 lec26-io.19

Saeid Nooshabadi

## DSLMU/Komodo Serial I/Os

### ° DSLMU Serial Port: memory-mapped terminal (Connected to the PC for program download and debugging)

- Read from PC Keyboard (**receiver**); 2 device regs
- Writes to PC terminal (**transmitter**); 2 device regs



COMP3221 lec26-io.20

Saeid Nooshabadi

## DSLMU/Komodo Serial I/Os

### ° Status register rightmost bit (0): Ready

- Receiver: Ready==1 means character in Data Register not yet been read (or ready to be 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

### ° Data register rightmost byte has data

- Receiver: last char from keyboard; rest = 0
- Transmitter: when write rightmost byte, writes char to display

COMP3221 lec26-io.21

Saeid Nooshabadi

## Reading Material

### ° Reading Material:

- Experiment 4 Documentation
- Hardware Reference Manual on CD-ROM

COMP3221 lec26-io.22

Saeid Nooshabadi

## Serial I/O Example (Read)

### ° Input: Read from PC keyboard into R0

```
.set iobase, 0x10000000 ; Base of DSLMU I/O space
.set ser_RxD, 0x10      ; Serial RxD port
.set ser_stat, 0x14      ; Serial Status port
.set ser_Rx_rdy, 0b01    ; Test bit 0 for RxD
                        ; ready status
```

```
readbyte:
    ldr r1,=iobase ; R1 = base address of I/O Space
```

```
Waitloop:
    ldrb r0,[r1,#ser_stat] ; Read the serial port
                        ; status
    tst r0,#ser_Rx_rdy     ; Check whether a byte
                        ; is ready to be read
    beq Waitloop          ; (No: jump back and try
                        ; again
    ldrb r0,[r1,#ser_RxD]  ; Read the available
                        ; byte into R0
    mov pc,lr
```

### ° Processor waiting for I/O called “Polling”

COMP3221 lec26-io.23

Saeid Nooshabadi

## Serial I/O Example (Write)

### ° Input: Write from to Display from R0

```
.set iobase, 0x10000000 ; Base of DSLMU I/O space
.set ser_RxD, 0x10      ; Serial TxD port
.set ser_stat, 0x14      ; Serial Status port
.set ser_Tx_rdy, 0b10    ; Test bit 1 for TxD
                        ; ready status
```

```
writebyte:
    ldr r1,=iobase ; R1 = base address of I/O Space
```

```
Waitloop:
    ldrb r2,[r1,#ser_stat] ; Read the serial port
                        ; status
    tst r2,#ser_Tx_rdy     ; Check whether is ready
                        ; to accept new data
    beq Waitloop          ; (No: jump back and try
                        ; again
    strb r0,[r1,#ser_TxD]  ; Send the next byte
                        ; from R0
    mov pc,lr
```

### ° Processor waiting for I/O called “Polling”

COMP3221 lec26-io.24

Saeid Nooshabadi



## Serial I/O Example Quiz

◦ What gets printed out?

```
ldr r1,=iobase
mov r0, 'A';
strb r0,[r1,#ser_TxD]
mov r0, 'B';
strb r0,[r1,#ser_TxD]
mov r0, 'C';
```

1. ABC
2. AB
3. AC

```
Waitloop:
ldrb r1,[r1,#ser_stat] ; Read the serial port
                        ; status
tst r1,#ser_Tx_rdy     ; Check whether is ready
                        ; to accept new data
beq Waitloop           ; (No: jump back and try
                        ; again
strb r0,[r1,#ser_TxD] ; Send the next byte
                        ; from R0
```

COMP3221 lec26-io.25

Saeid Nooshabadi

## Cost of Polling?

◦ Assume for a processor with a 500-MHz 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 8 MB/second. Again, no transfer can be missed.

COMP3221 lec26-io.26

Saeid Nooshabadi

## % Processor time to poll mouse, floppy

- Mouse Polling Clocks/sec  
 $= 30 * 400 = 12000$  clocks/sec
- % Processor for polling:  
 $12 * 10^3 / 500 * 10^6 = 0.002\%$   
 $\Rightarrow$  Polling mouse little impact on processor
- Times Polling Floppy/sec  
 $= 50 \text{ KB/s} / 2\text{B} = 25\text{K}$  polls/sec
- Floppy Polling Clocks/sec  
 $= 25\text{K} * 400 = 10,000,000$  clocks/sec
- % Processor for polling:  
 $10 * 10^6 / 500 * 10^6 = 2\%$   
 $\Rightarrow$  OK if not too many I/O devices

COMP3221 lec26-io.27

Saeid Nooshabadi

## % Processor time to hard disk

- Times Polling Disk/sec  
 $= 8 \text{ MB/s} / 16\text{B} = 500\text{K}$  polls/sec
- Disk Polling Clocks/sec  
 $= 500\text{K} * 400 = 200,000,000$  clocks/sec
- % Processor for polling:  
 $200 * 10^6 / 500 * 10^6 = 40\%$   
 $\Rightarrow$  Unacceptable

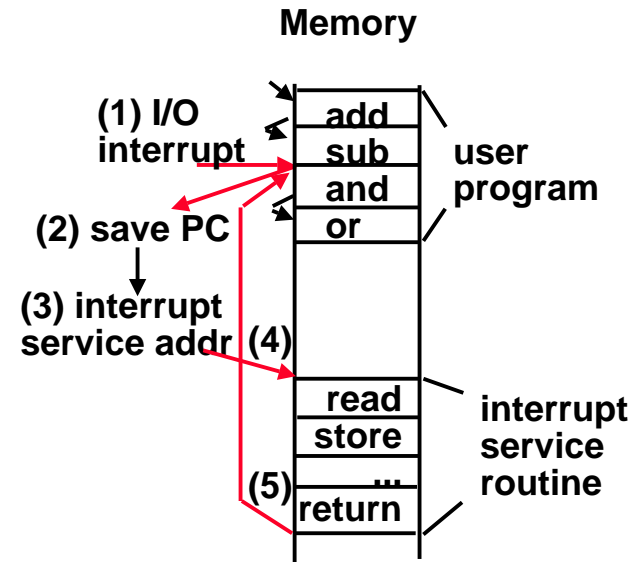
COMP3221 lec26-io.28

Saeid Nooshabadi

## What is the alternative to polling?

- Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready
- Wish we could have 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

## Interrupt Driven Data Transfer



## Benefit of Interrupt-Driven I/O

- 500 clock cycle overhead for each transfer, including interrupt. Find the % of processor consumed if the hard disk is only active 5% of the time.
- Interrupt rate = polling rate
  - Disk Interrupts/sec = 8 MB/s / 16B = 500K interrupts/sec
  - Disk Polling Clocks/sec = 500K \* 500 = 250,000,000 clocks/sec
  - % Processor for during transfer:  $250 \times 10^6 / 500 \times 10^6 = 50\%$
- Disk active 5%  $\Rightarrow$  5% \* 50%  $\Rightarrow$  2.5% busy

## Polling vs. Interrupt Analogy

- Imagine yourself on a long road trip with your 10-year-old younger brother... (You: I/O device, brother: CPU)
- Polling:
  - “Are we there yet? Are we there yet? Are we there yet? ....”
  - CPU not doing anything useful
- Interrupt:
  - Stuff him a **color** gameboy, “interrupt” him when arrive at destination
  - CPU does useful work while I/O busy



## Conclusion (#1/2)

---

- I/O is how humans interact with computers
- I/O lets computers do amazing things
- I/O devices often have a few registers
  - Status registers
  - I/O registers
  - Typically, devices map to only a few bytes in memory

## Conclusion (#2/2)

---

- I/O gives computers their 5 senses
- I/O speed range is 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