**Digression: Input/Output.** Recall this picture from lecture 1:

How does the CPU actually talk to the sensors and actuators? Two methods: 1) memory-mapped I/O

and 2) port-mapped I/O, or special instructions. In memory-mapped I/O, the CPU executes what it thinks

are reads and writes to memory. In particular, it sends out the appropriate requests on the system bus.

Devices listen on the bus and manufacture the appropriate responses. For special instructions, as seen

on Intel ia32 processors, the CPU instead executes special in and out instructions, which may transmit

data on a special bus, or set a speci1c signal on the bus.

**Pseudocode for Tight Polling Loop.** Here is pseudocode for a memory-mapped I/O system.

while( statusRegister == 0x0000 ) {

// Do nothing until statusRegister changes value

}

// Read data that has changed from a dataRegister and store in memory

incomingData = dataRegister;

We’d expect this loop to terminate based on some hardware speci1cation promising statusRegister

eventually becoming non-zero due to an external event. Data exchange occurs once the device indicates

that it is ready to emit data (by setting statusRegister); we can call this *polling synchronization*.

**Interrupts**

*“In Soviet Russia, event polls you.”*

Another way that a processor can 1nd out about an event is via an *interrupt*. Interrupts *actively synchronize*

a device and a processor. An interrupt tells the processor one bit of information: that something

worth knowing about (high-priority) is occurring. When the processor gets an interrupt, it stops what

it’s currently doing, saves its state, and starts executing a pre-de1ned *interrupt handler*. The interrupt

handler will typically read the event information (how?) and store it somewhere accessible. After the

handler returns, the processor restores its state and resumes what it was doing before.