### Autonomous Controller Operation

- **Direct Memory Access (DMA)**
  - Controller can send/read data from/to any memory address
  - CPU is oblivious to these transfers
  - DMA addresses and sizes are programmed by CPU using PIO
- **CPU Interrupts**
  - Controller can signal the CPU
  - CPU checks for interrupts on every cycle (it's a race condition)
  - CPU jumps to controller's Interrupt Service Routine if it is interrupting

### Autonomous Controller Operation

- **Direct Memory Access (DMA)**
  - Controller can send/read data from/to any memory address
  - CPU is oblivious to these transfers
  - DMA addresses and sizes are programmed by CPU using PIO
- **CPU Interrupts**
  - Controller can signal the CPU
  - CPU checks for interrupts on every cycle (it's a race condition)
  - CPU jumps to controller's Interrupt Service Routine if it is interrupting

### Adding Interrupts to Simple CPU

- **New special-purpose CPU registers**
  - `IsDeviceInterrupting` set by I/O Controller to signal interrupt
  - `InterruptControllerID` set by I/O Controller to identify interrupting device
  - `InterruptVectorBase` interrupt handler jump table, initialized at boot time
- **Modified fetch–execute cycle**

```c
while (true) {
  if (IsDeviceInterrupting) {
    // Process the interrupt
    pc = InterruptVectorBase[InterruptControllerID];
    fetch();
    execute();
  } else {
    // Continue with normal execution
    fetch();
    execute();
  }
}
```
Humans like synchrony
• we expect each step of a program to complete before the next one starts
• we use the result of previous steps as input to subsequent steps
• with disks, for example,
  - we read from a file in one step and then usually use the data we’ve read in the next step

Computer systems are asynchronous
• the disk controller takes 10-20 milliseconds (10^{-3}s) to read a block
  - CPU can execute 60 million instructions while waiting for the disk
  - we must allow the CPU to do other work while waiting for I/O completion
• many devices send unsolicited data at unpredictable times
  - e.g., incoming network packets, mouse clicks, keyboard key presses
  - we must allow programs to be interrupted many, many times a second to handle these things

Asynchrony makes programmers sad
• it makes programs more difficult to write and much more difficult to debug

Possible Solutions
• Accept the inevitable
  - use an event-driven programming model
    - event triggering and handling are decoupled
  - a common idiom in many Java programs
    - GUI programming follows this model
  - CSP is a language that boosts this idea to first-class status
    - no procedures or procedure calls
    - program code is decomposed into a set of sequential/asynchronous processes
    - processes can fire events, which can cause other processes to run in parallel
    - each process has a guard predicate that lists events that will cause it to run
• Invent a new abstraction
  - an abstraction that provides programs the illusion of synchrony
  - but, what happens when
    - a program does something asynchronous, like disk read?
    - an unanticipated device event occurs?
• What’s the right solution?
  - we still don’t know — this is one of the most pressing questions we currently face