Principles of I/O Hardware

Module 18

I/O Hardware - System Call Perspective

System calls (the OS) provides a nice standard C interface for dealing with hardware

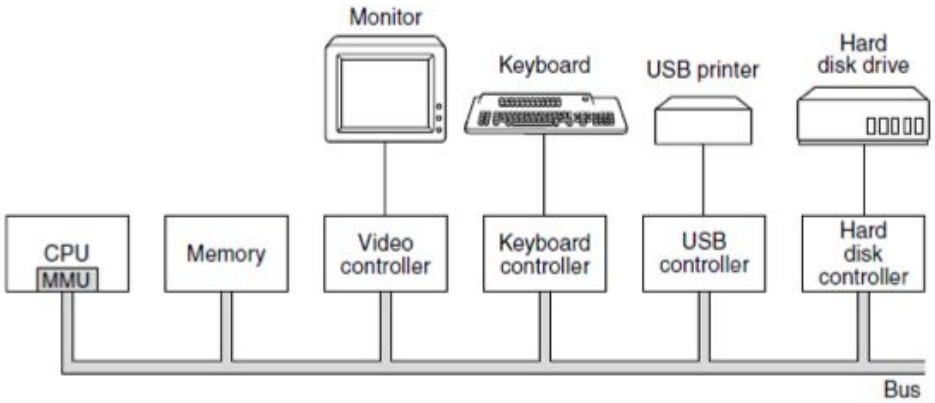
However - this is a really complex and diverse system - where each hardware device may work vastly differently!

Categories of I/O Devices

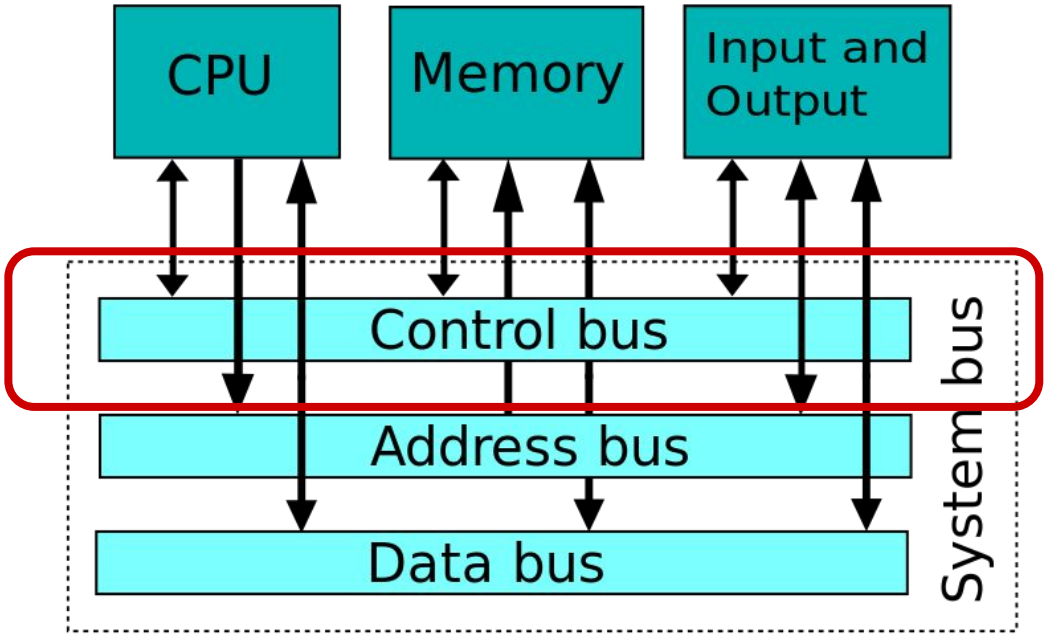
- Block Devices
  - Devices that read/write in fixed chunks
  - Each block is **independently** addressable
  - Hard Disk, Blue-Ray, USB, Tape Drives..
- Character Devices
  - Read/Write via a ***stream of characters***
  - No block structure
  - Each data item is not independently addressable
  - Printers, Network Interfaces, Mice

Transfer between CPU and Device

- At a high level, we know a “bus” connects our devices/controllers to the CPU
- The entire address space is mapped to discrete bus “lines” - which go to different controllers



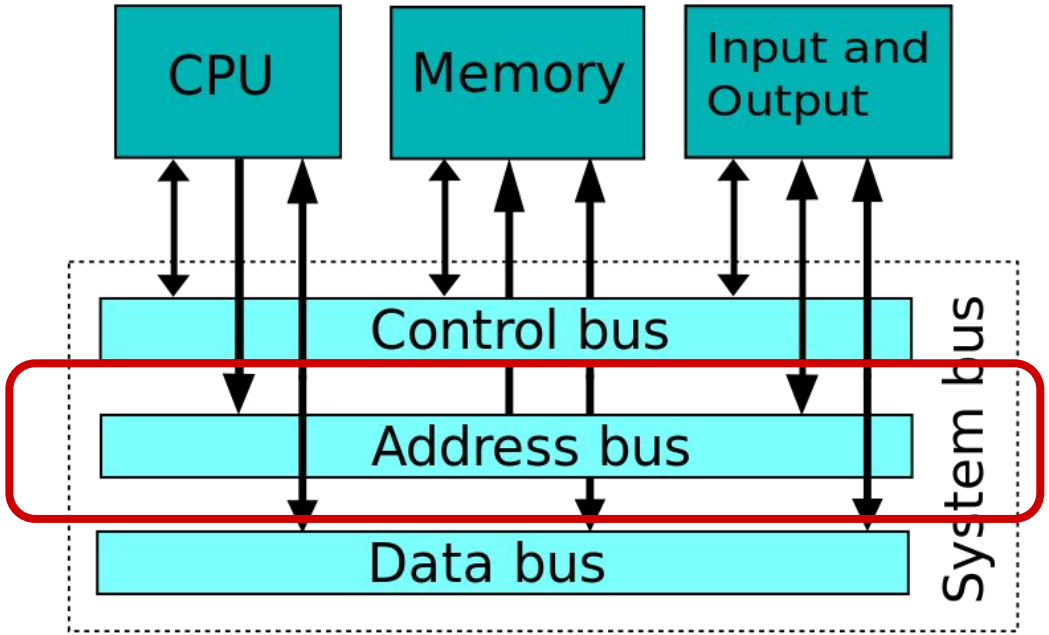
… A bit more complicated



To coordinate transfers, we need to issue “control” signals to device controllers.

Ex. “Hey Disc controller, I want to read from disc.”

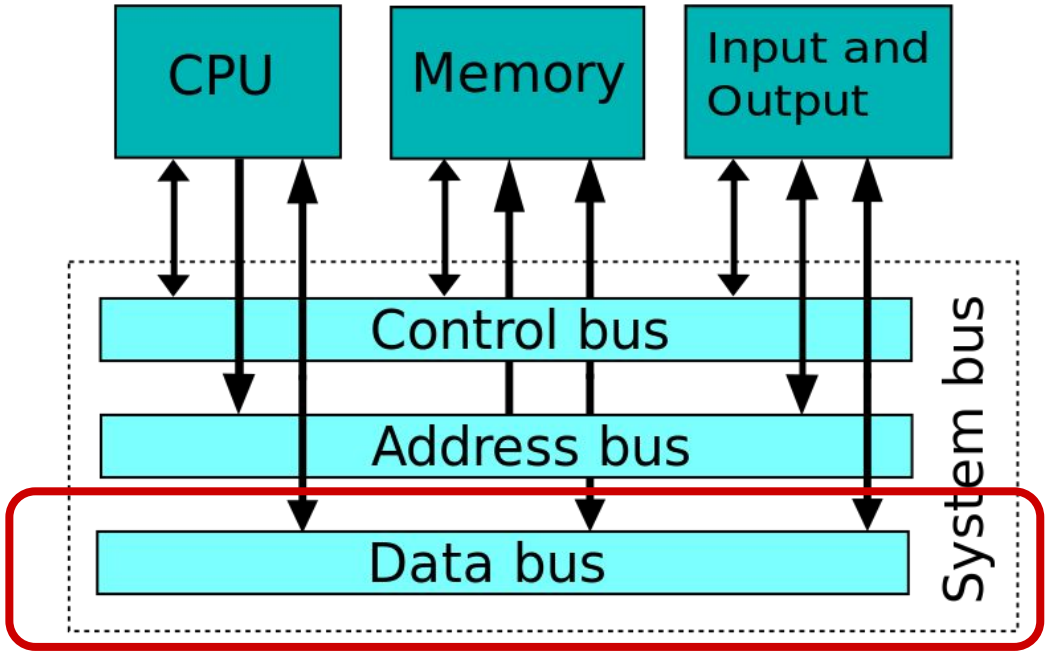
… A bit more complicated



For any transfer, we need to tell devices **where** to read/write to or from… over the address buss

Hey disc, I want to get block X and send it to memory address Y

… A bit more complicated



Then… once the control and addresses (parameters) are established - GO!

How fast can we go?

Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner at 300 dpi	1 MB/sec
Digital camcorder	3.5 MB/sec
4x Blu-ray disc	18 MB/sec
802.11n Wireless	37.5 MB/sec
USB 2.0	60 MB/sec
FireWire 800	100 MB/sec
Gigabit Ethernet	125 MB/sec
SATA 3 disk drive	600 MB/sec
USB 3.0	625 MB/sec
SCSI Ultra 5 bus	640 MB/sec
Single-lane PCIe 3.0 bus	985 MB/sec
Thunderbolt 2 bus	2.5 GB/sec
SONET OC-768 network	5 GB/sec

Data bus must be able to accommodate the *fastest* speed

Controls / Addresses

There are two parts to any I/O Device:

- Controller
  - A microchip with registers that can be written/read over the control/address bus
  - Figures out what to do based on those values
- The mechanical device
  - Does the physical/electrical work to execute the action requested (input, output)

Device Controllers

- Controllers are simple computers… the CPU (or another controller) will send input by writing to its registers.
- Controllers provide the *interface* for the CPU to control the device
- Standard interfaces (USB, SATA, SCSI) exist so device can be more usable

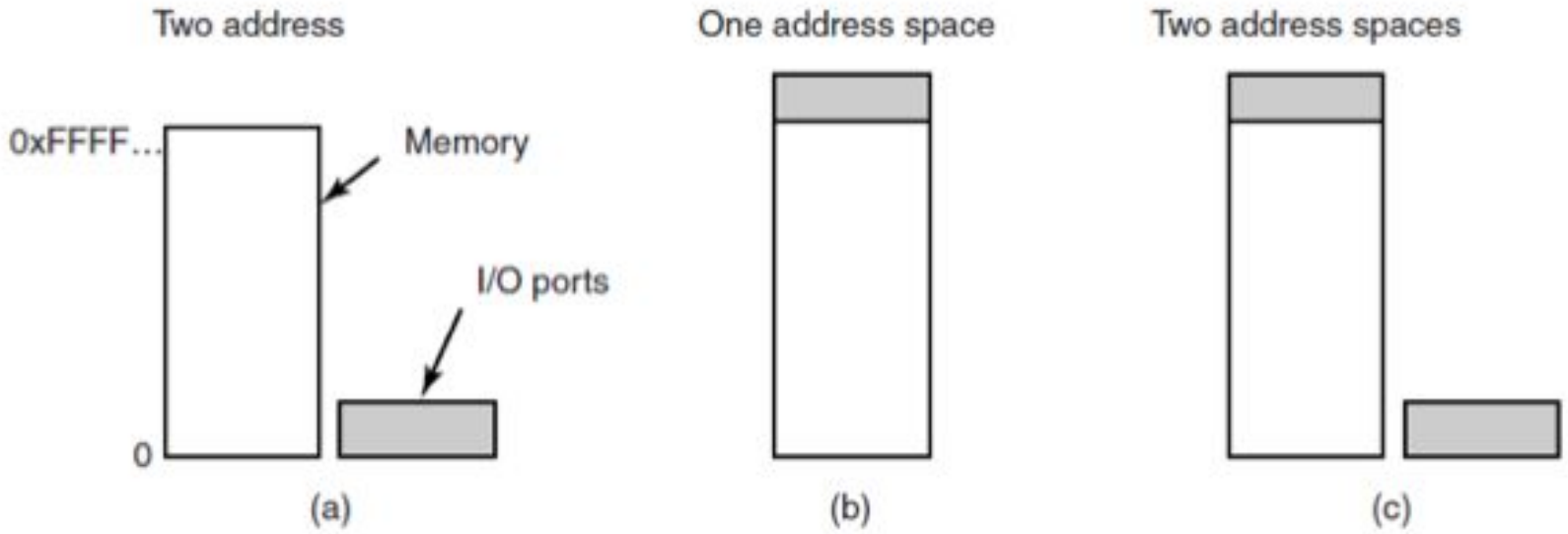
Communicating with I/O Controllers

- On the bus, some addresses correspond to memory, while others occupy “I/O port space”
- A CPU may have instructions like this:
  - `IN REG, PORT` // Move contents of REG to PORT
  - `OUT PORT, REG` // Move contents in PORT to REG
- In this scheme, Device Drivers are modules of code that translate ideas like “read from mouse” to a specific register number, port number, and what to actually write to the port itself (the correct control signal)
  - Device driver would require assembly code

Memory Mapped I/O Controllers

- A more modern approach is to assign bus lines (ports) to addresses in “memory”
  - However… these address are **not** memory - read/write is **mapped** by hardware to the correct bus line
  - Intel x86 uses this - address 640K-1M is reserved for device communication
- Since writing to devices now require the same CPU instruction as writing variables (MOV), device drivers can be written in C

I/O Space strategies



Protecting devices from user programs

- Now we see how Paging and device security is also interrelated:
  - User programs could never write to devices because I/O mapped addresses won't be put in a user program's page table!

Steps for transferring data

Let's say the CPU wants to read from Disc:

1. CPU sets control flag (read a block)
2. Disc controller reads block into internal buffer
3. Ensures ECC matches
4. Disc controller causes interrupt
5. CPU reads data from controller’s buffer - **byte-by-byte** into memory.

Problems...

- This process is SLOW:
  - Reading a byte from the controller is time-consuming
  - The CPU needs to tediously read one byte at a time into memory
- Would be nicer to free CPU from this mundane task… and allow some other hardware to ensure ALL data gets moved to memory

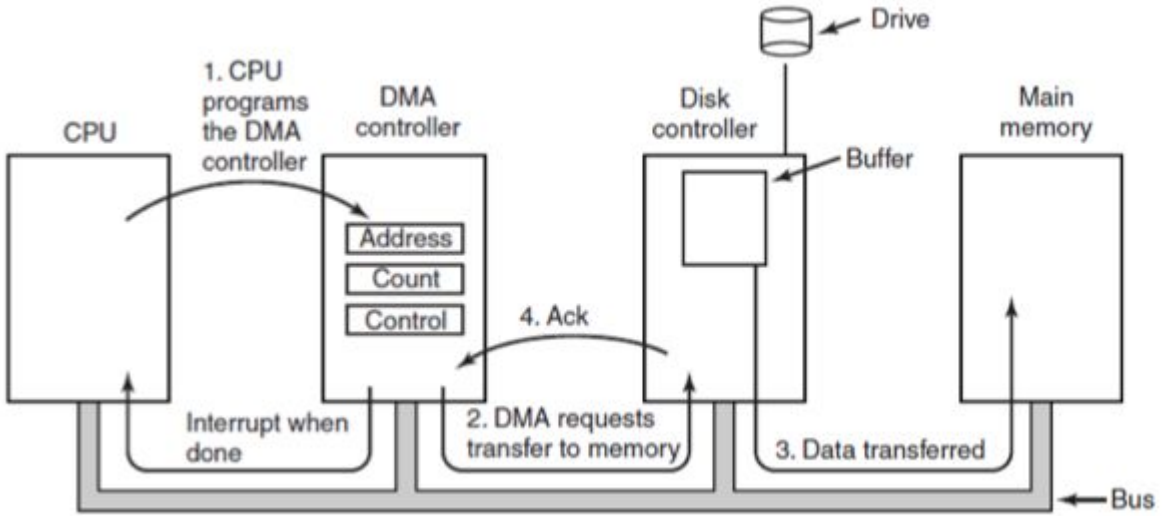
Solution: Direct Memory Access

Modern systems will have a DMA controller

The CPU delegates the task for moving each byte from device controller to memory to the DMA

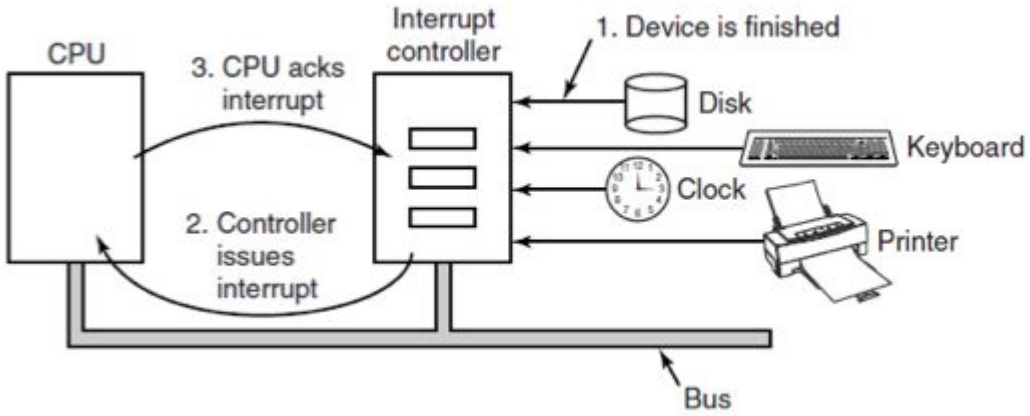
Now if the CPU want's 10MB from disk (contiguous blocks), it can ask the DMA to do this.

The CPU will only be interrupted when all 10MB are in memory.



Interrupts, revisited

We’ve had this idea of an interrupt since the first week - but how is this actually managed?



Specific bus lines are dedicated to interrupts

An interrupt controller manages complexities like simultaneous interrupts getting fired