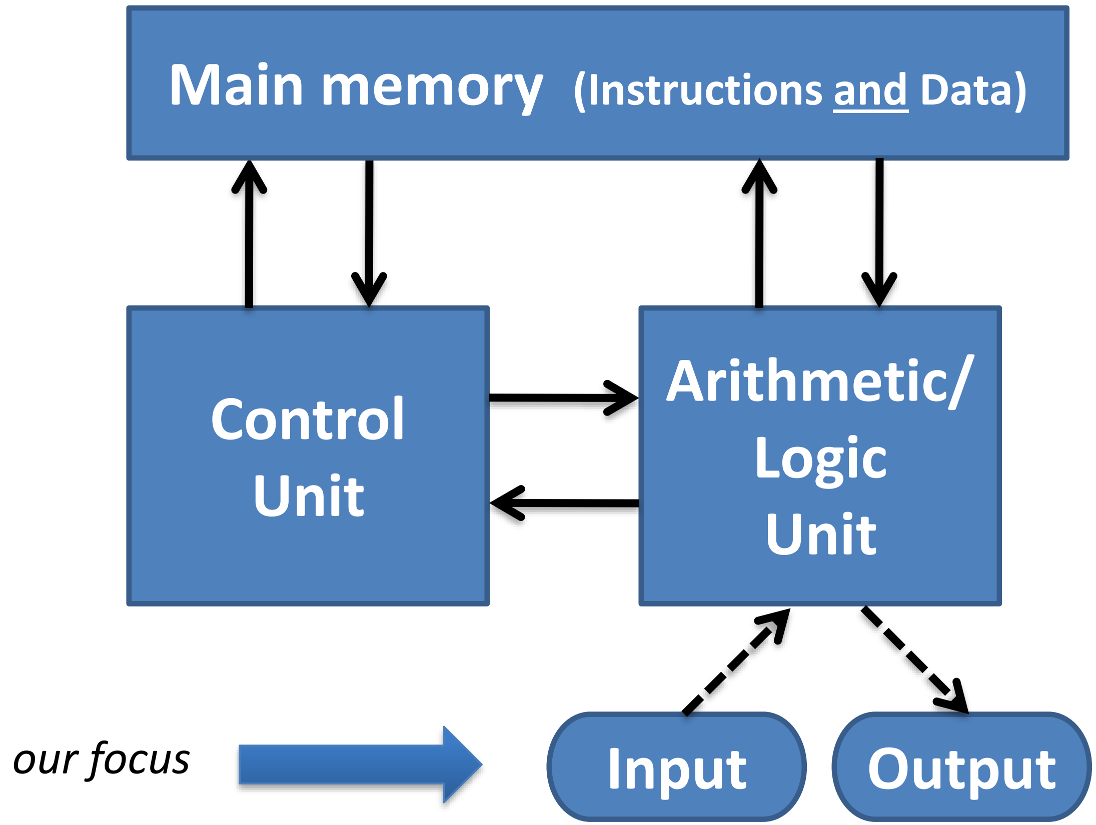
SCC131: Digital Systems

Topic 9: Building the input/output system

Reminder of the von Neumann architecture

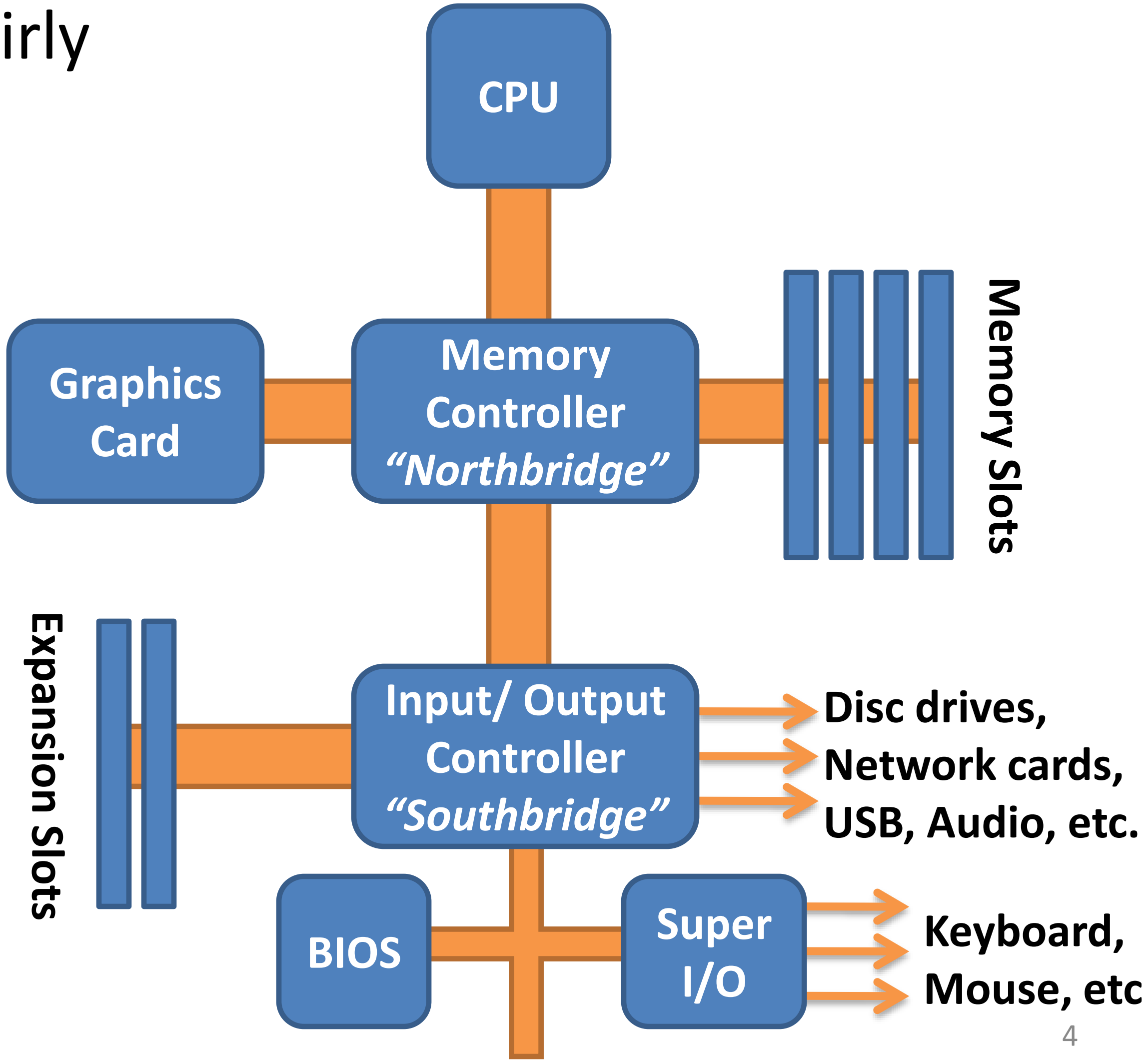


What does the input/output (a.k.a. "I/O") system do?

- Enables the attachment of input and output devices to the processor
- Examples of **input** devices
  - Keyboards, mice, track balls, touch screens, musical instruments, cameras, environmental sensors, …
- Examples of **output** devices
  - Displays, printers, speakers, environmental actuators, …
- Examples of **input-*and*-output** devices
  - Network interfaces (Ethernet, WiFi, Bluetooth, …), disks, audio cards, MIDI devices, …

Input and output

- Remember the "fairly modern PC…"?
- Connecting I/O devices is *somewhat similar* to connecting memory
- But there are I/O specific complications and challenges…



I/O challenge 1: the *speed-gap* challenge

- I/O devices are often mechanical, and therefore run *orders of magnitude slower* than the CPU...

- So: how can we ensure that the CPU is not slowed down when it interacts with I/O devices (e.g. when fetching a data from the hard disk)?

  – We can’t afford to waste CPU cycles!

- (N.B., with very slow CPUs and very fast devices the problem can be reversed - *any* difference in performance is a problem)

I/O challenge 2: the *device diversity* challenge

- Devices are extremely **diverse**, e.g., …
  - Diversity of data-access modes
    - Read-only or write-only or read-and-write
    - Access by the individual byte or by the block/ by the stream
    - Access randomly (like a disk) or sequentially (like a tape)
  - Device-specific operations
    - Change the resolution (only for screens), set the time (only for clocks), focus (only for cameras), …
  - I/O protocol
    - We may need to be concerned with potential data transfer errors (e.g. from electrical noise, wireless transmission errors)
    - Synchronous or asynchronous

Device drivers

- *These are software plug-ins inside the operating system*
- *Their job is to abstract over device diversity by "grouping" sets of "somewhat similar" types of device; e.g.*
  - *Many devices can be made to "look like a mouse" to software*
  - *All hard disks are "essentially the same"*

- *The functions of device drivers typically include…*
  - *Registering a device with the OS and initialising it*
  - *Initiating data transfers to or from a device*
  - *Monitoring status events from a device*
  - *Managing device/ system shutdown*
    - *Ensuring that the OS doesn't "stop" until all unwritten data is stored, and the device is left in a safe state*)

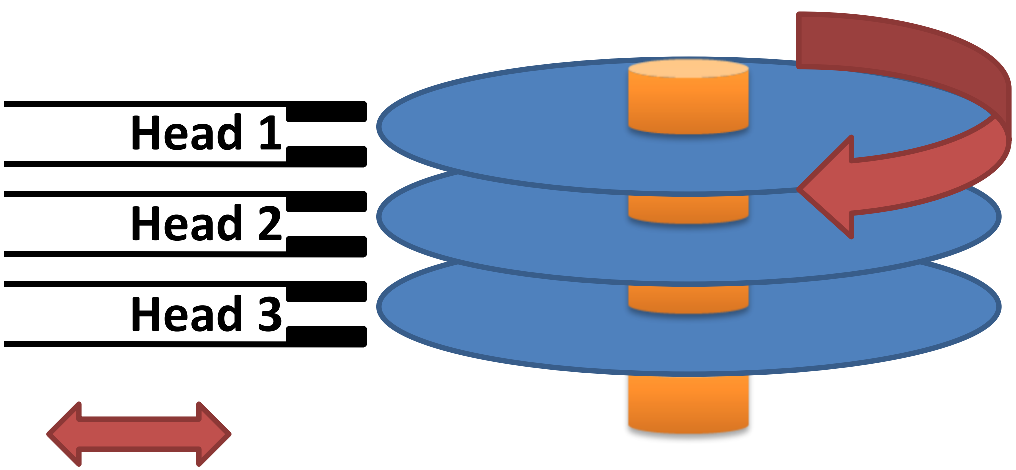
Traditional two-fold classification of device types (and device driver types)

1. Character devices
   – Send and receive **one byte** at a time
   – Classic example: the *keyboard*

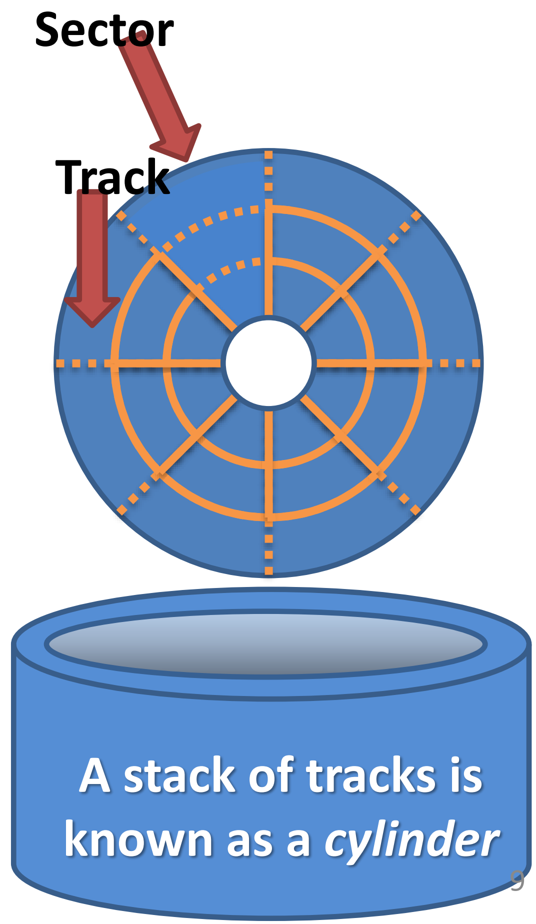
2. Block devices
   – Send and receive a multi-byte **block** at a time
   – Classic example: the *hard disk*

Block device example: the hard disk...

- Disks are naturally “block-oriented” devices: the smallest unit of disk storage is the *sector*
- A 512 byte sector (block) size is common



Addressed at the device level as
<Head, Cylinder, Sector>



Traditional two-fold classification of processor support for I/O

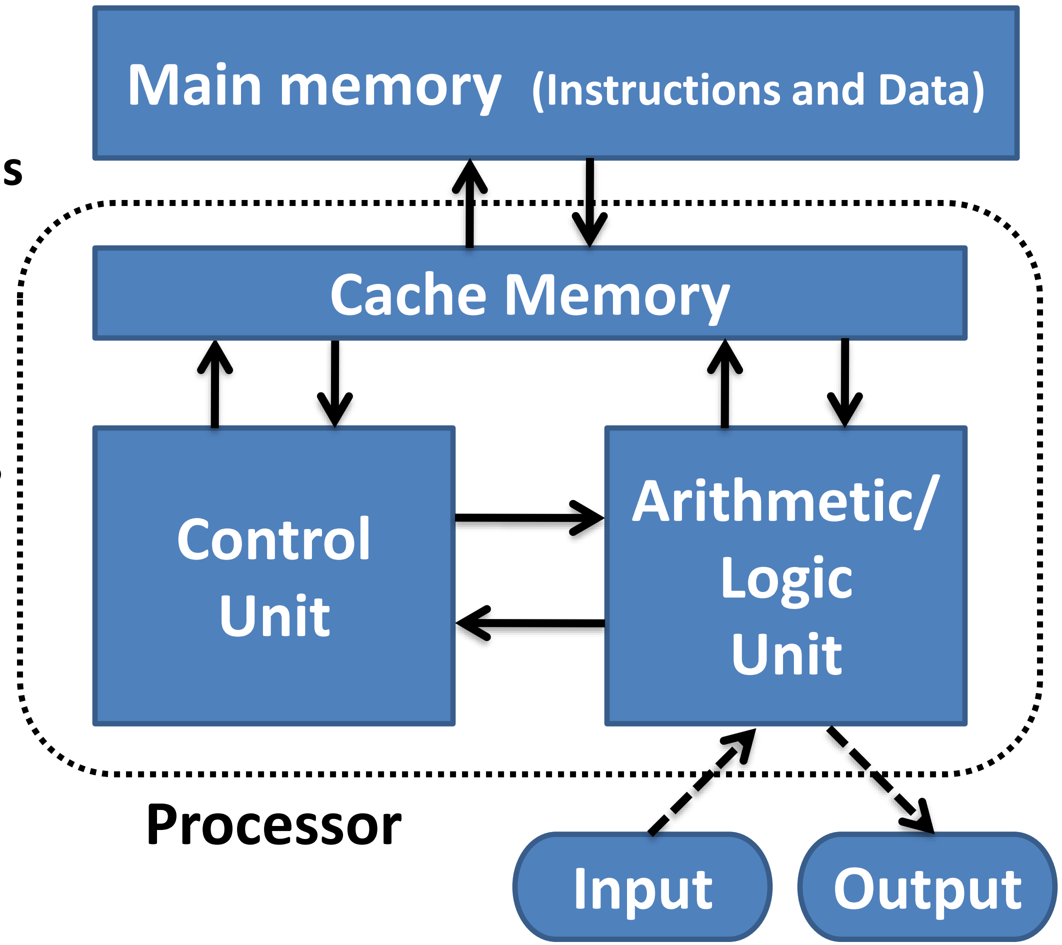
1. Isolated I/O

2. Memory-mapped I/O

Isolated I/O (1)

- The processor provides:
  - Dedicated **physical pins** for the connection of I/O devices, and
  - And dedicated **instructions** for doing I/O operations

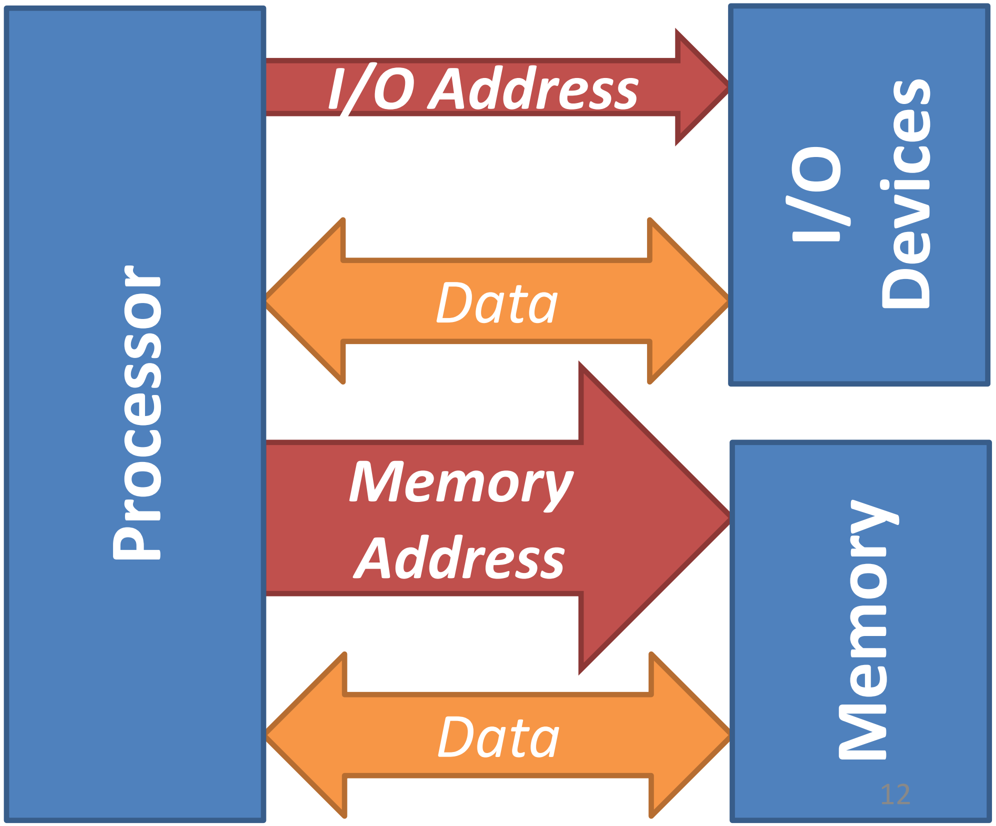
- Suited to simple devices
  - Having only a fixed set of special I/O instructions does not help much with device diversity



Isolated I/O (2)

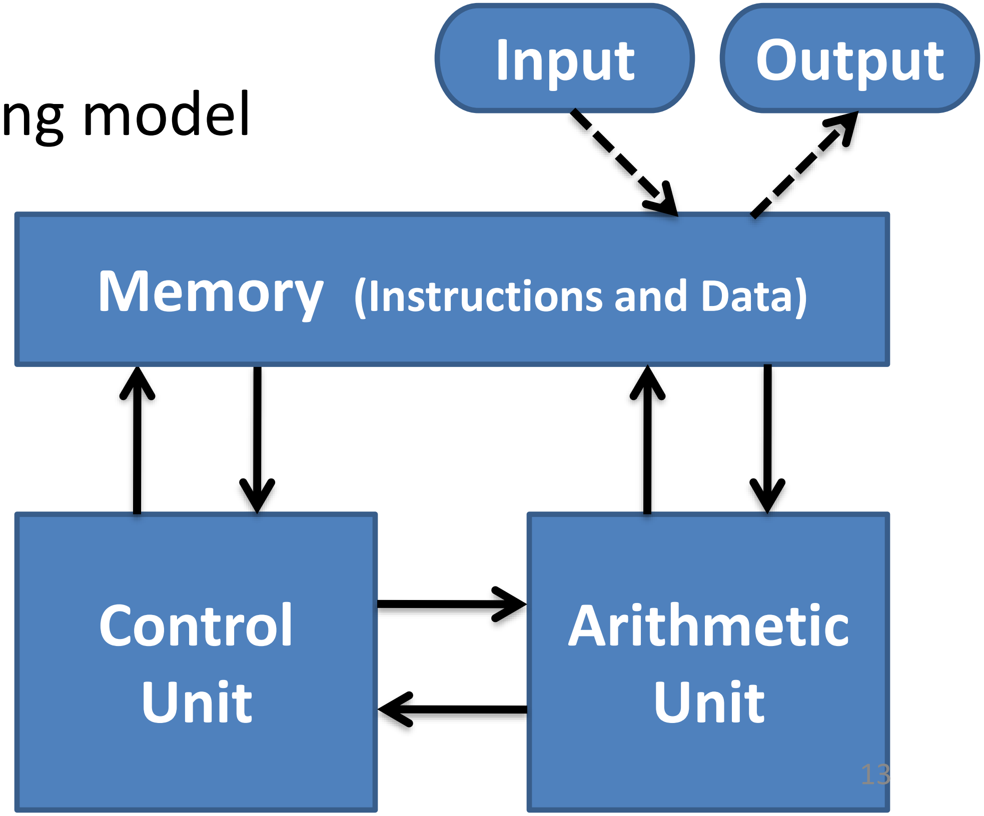
- Example Intel x86 instructions
  - **IN** destination_register, port_address
  - **OUT** source_register, port_address

- Port addresses are typically 8 bits: narrower than main memory addresses



Memory-mapped I/O

- Here, devices sit within the CPU's linear memory address space
  – E.g. we might access the next keyboard character by reading from memory address 0x40000040
- Simple, flexible programming model
- Downside is that it adds complexity to devices
  – They need to understand larger addresses and to work at memory speeds



Summary

- We appreciate the **speed-gap challenge:** the huge speed difference between CPUs and I/O devices
- We appreciate the **device diversity challenge:** the need to handle a variety of device types, including **character** and **block** devices
  - And we appreciate (very roughly!) how operating systems abstract over device classes using device drivers
- We understand, in outline, how data is stored and addressed on hard disks
- We understand the distinction between the strategies of **isolated I/O** and **memory-mapped I/O**, and their pros and cons