Principles of Operating Systems

Lecture 8 - I/O Systems

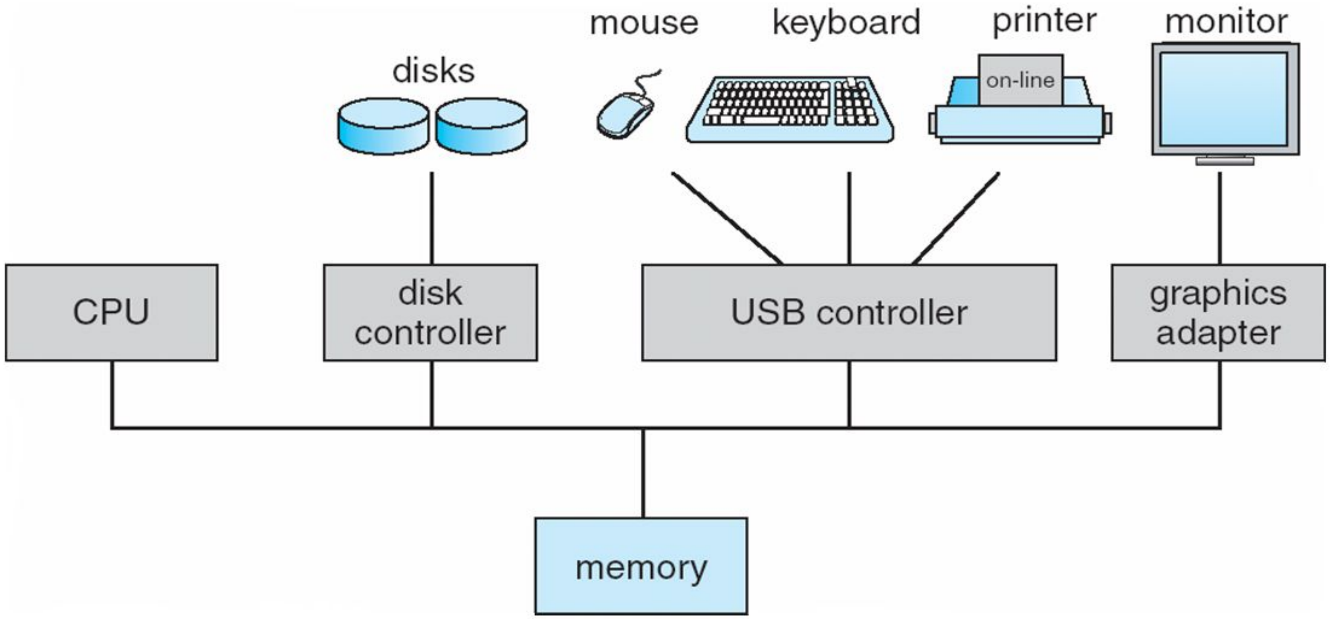
Ardalan Amiri Sani (ardalan@uci.edu)

[*lecture slides contains some content adapted from course text slides © Silberschatz*]

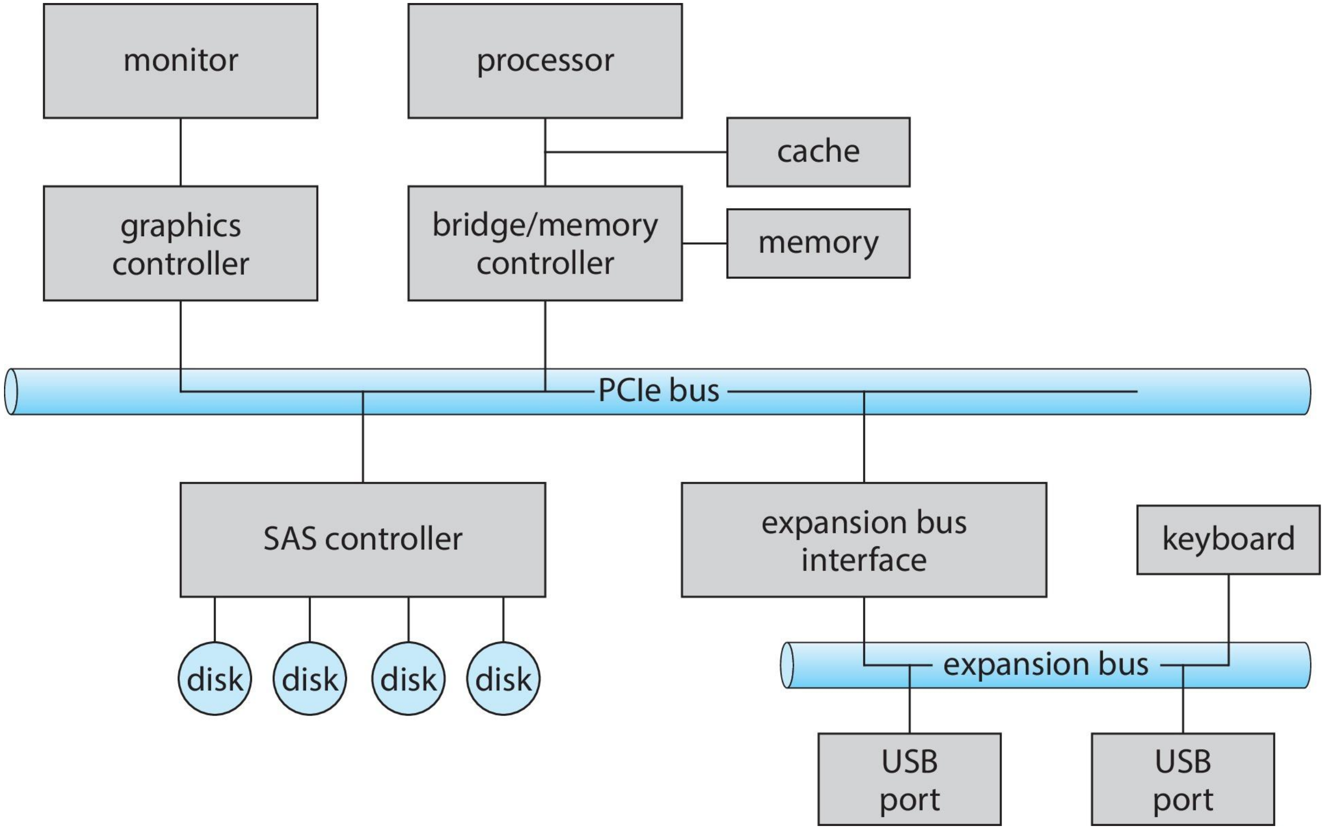
Input and output (mainly for computer to interact with the rest of the world)

- Display (and GPU)
- Mouse and keyboard
- Storage
- Network
- Sensors

Computer System Organization - simplified

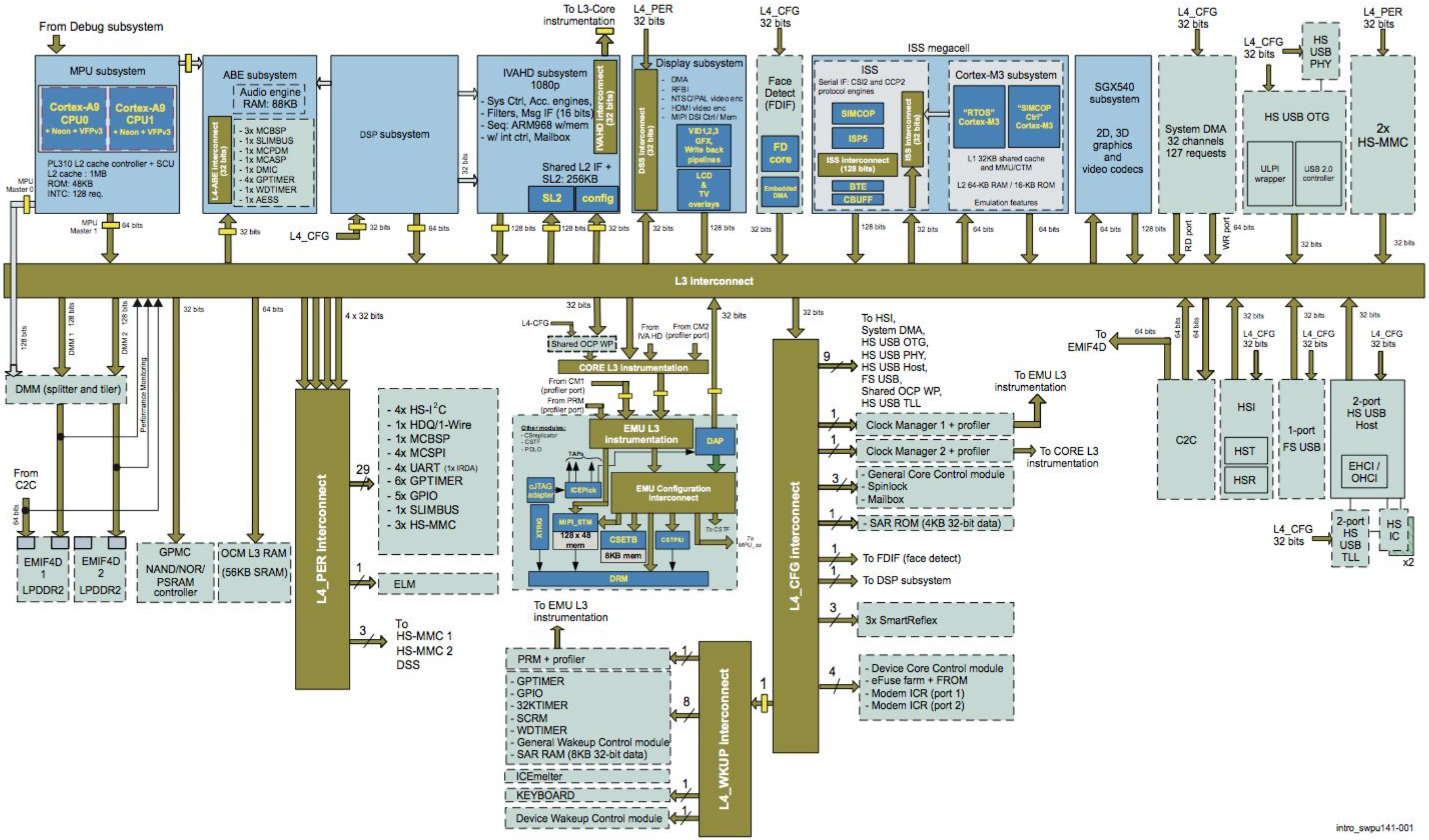


Reality is more complex…



Even more complex!!! (This slide is not covered in the exam)

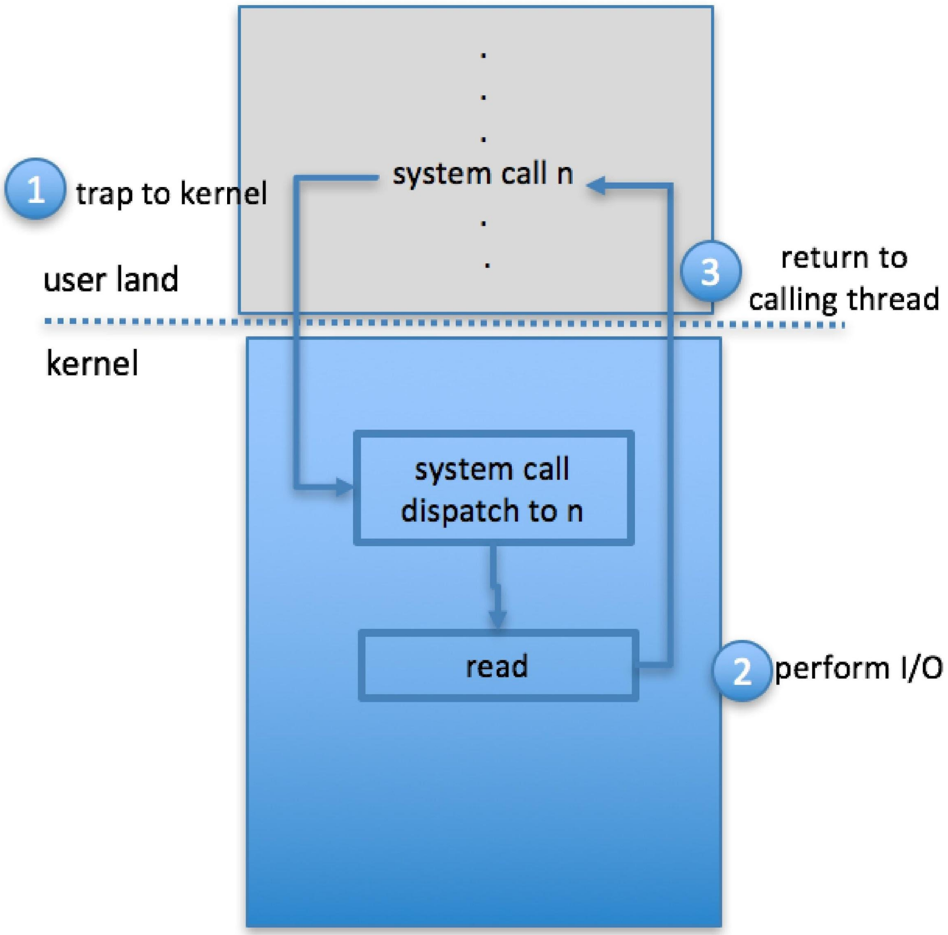
Figure 1-2. OMAP4430 Block Diagram



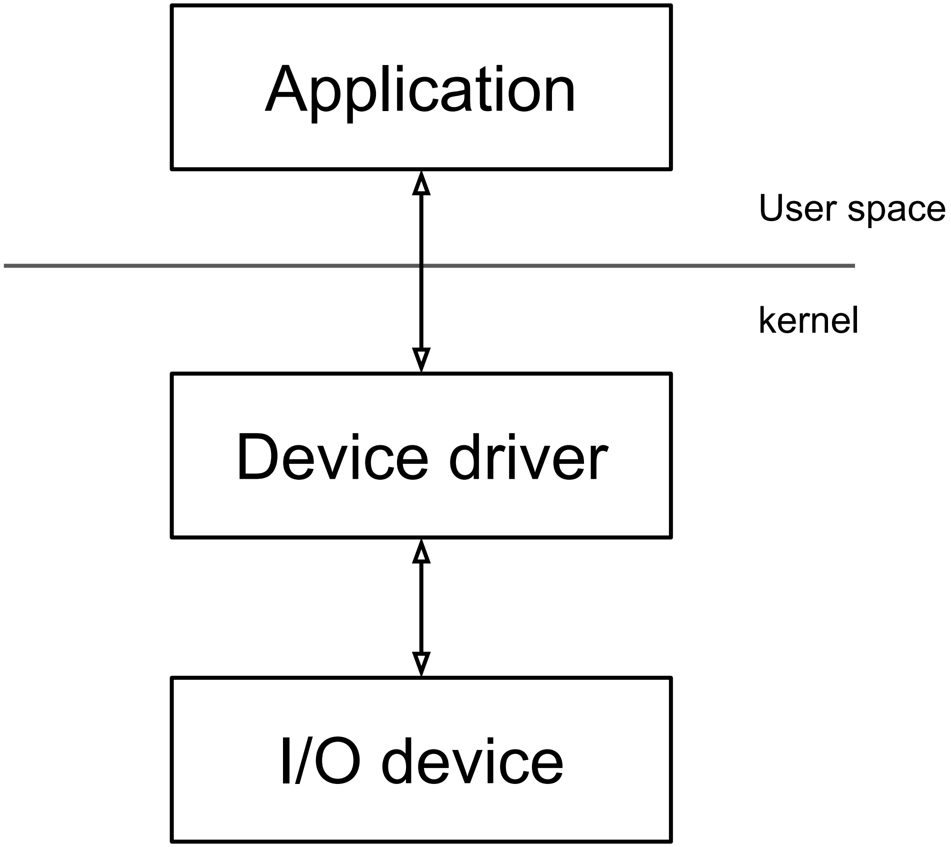
I/O Protection

- User process may accidentally or purposefully attempt to disrupt normal operation via illegal I/O instructions
  - All I/O instructions defined to be privileged
  - I/O must be performed via system calls

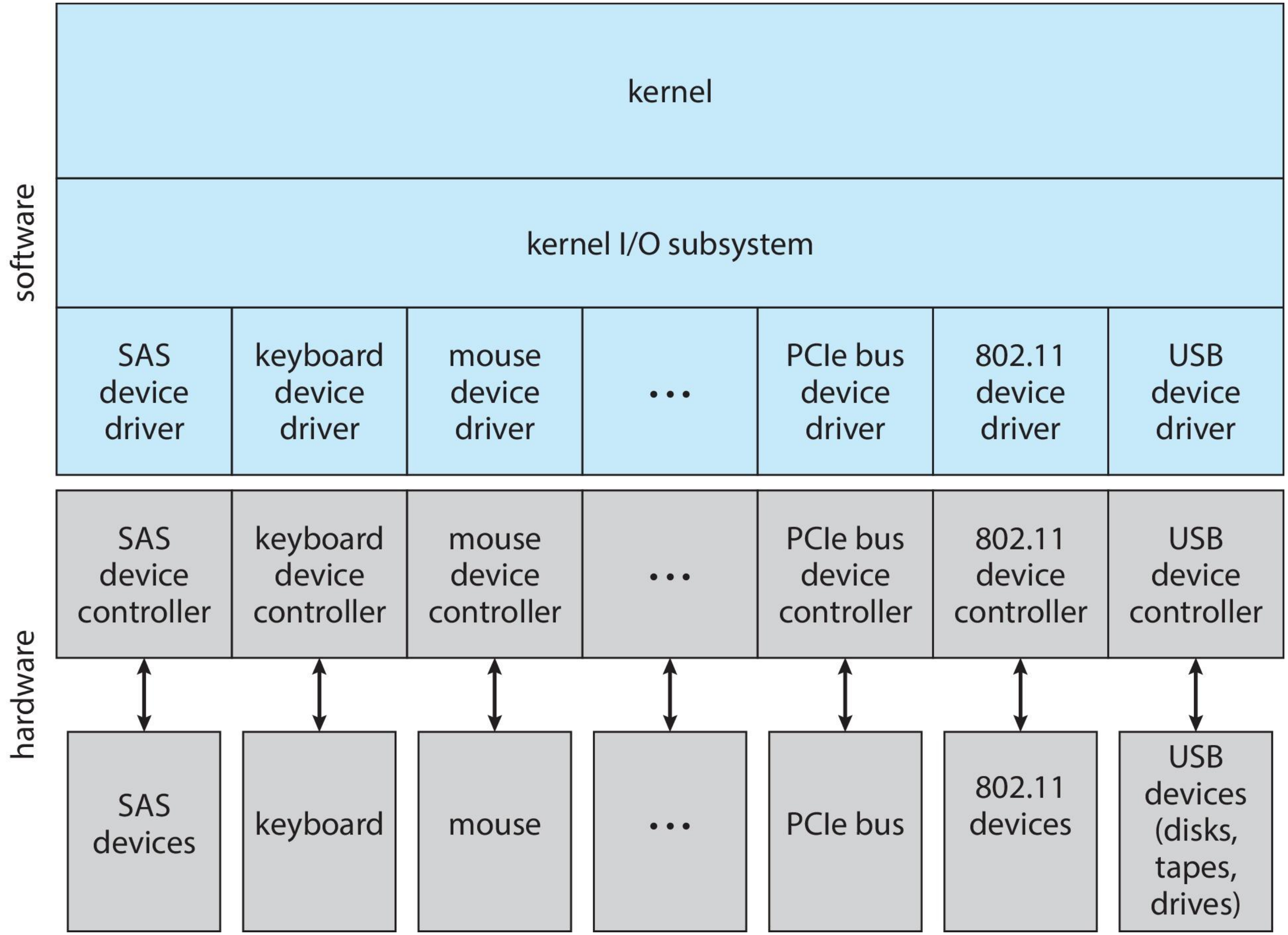
Use of a System Call to Perform I/O



Each device needs a device driver



Each device needs a device driver



Main hardware primitives for drivers to program I/O devices

Main hardware primitives for drivers to program I/O devices

- Registers
- Interrupts
- Direct Memory Access (DMA)

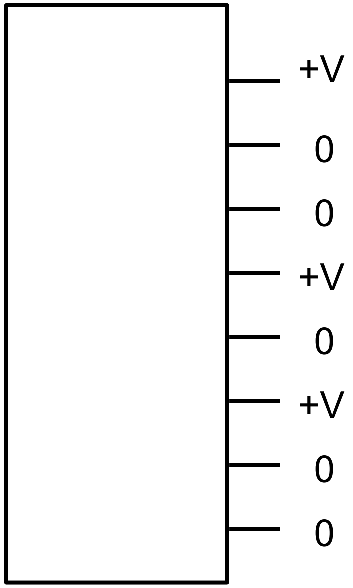
Registers are input and output variables of a device

- Write to a register will be seen by the device
- Read from a register will return the value of some state from the device

Simple example: GPIO

register value = 0b10010100

One-to-one mapping between the value of bits and the voltage on the pins



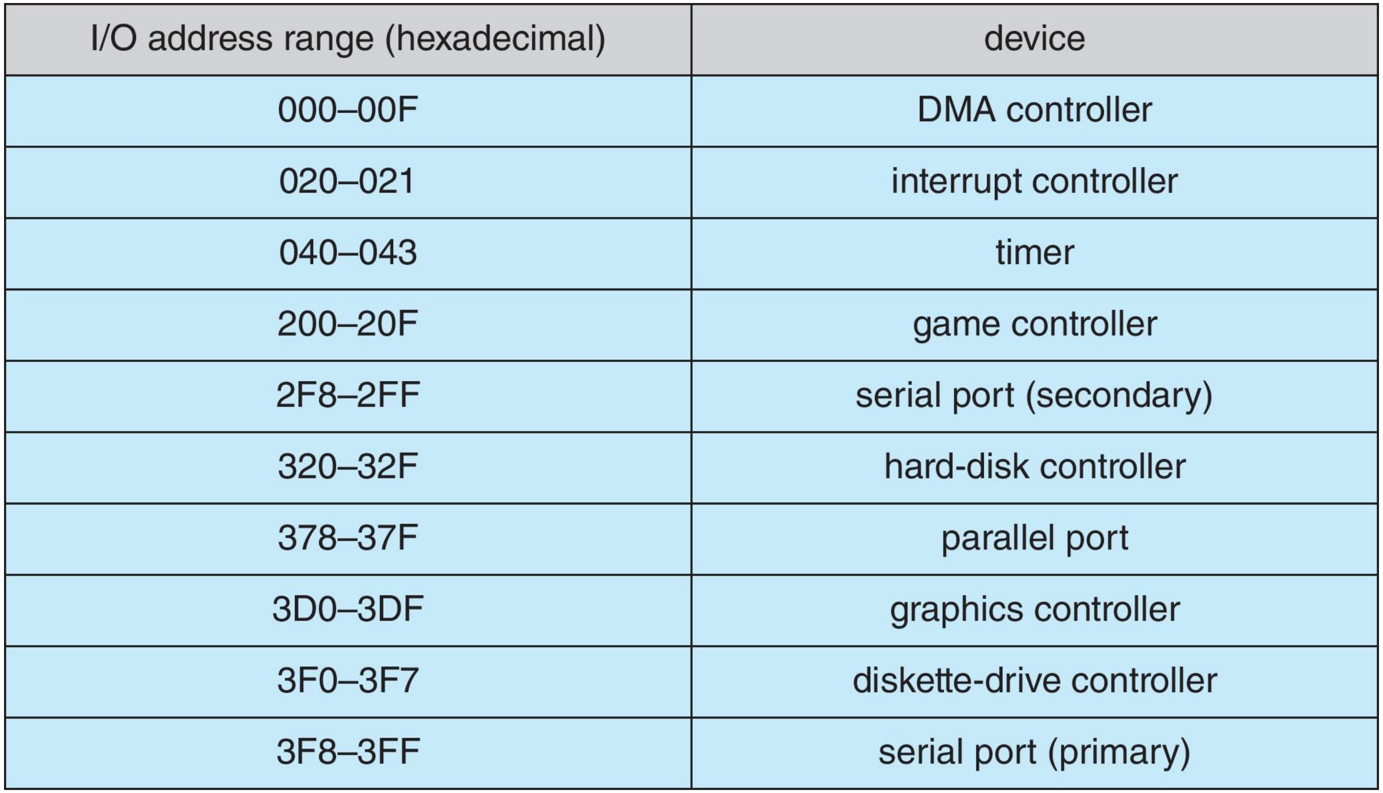
How to read/write registers?

- Port-mapped I/O (PIO or PMIO)
- Memory-mapped I/O (MMIO)

Port-mapped I/O

- Registers have their own address space
- Registers of devices are configured to non-overlapping addresses by the device and/or driver
- ISA has special instructions for these registers
  - inb and outb in x86

An example of device I/O port locations



I/O address range (hexadecimal)	device
000–00F	DMA controller
020–021	interrupt controller
040–043	timer
200–20F	game controller
2F8–2FF	serial port (secondary)
320–32F	hard-disk controller
378–37F	parallel port
3D0–3DF	graphics controller
3F0–3F7	diskette-drive controller
3F8–3FF	serial port (primary)

Memory-mapped I/O

- Registers are programmed like memory
- Each register has a physical address and can be accessed through some virtual address
- Same instructions as memory
    - ARM only supports MMIO

Interrupts

- Similar to the rest of the interrupts discussed before
- Driver registers an interrupt handler
- When device interrupts, the handler is called

Direct Memory Access (DMA)

- Device can directly read and write to memory
- Much more efficient for moving large chunks of data to/from device (compared to using device registers)
- DMA is programmed with physical addresses of memory (bypasses the virtual address translation discussed before)

Main primitives for applications to program I/O devices (UNIX)

- Device files
  - Character devices
  - Block devices
- Sockets (network devices)
- File systems (storage devices) – discussed previously
  - Read, write syscalls

Block and Character Devices

- Block devices include disk drives
  - Syscalls include read, write, seek, …
  - Direct access to disk

- Character devices include keyboards, mice, serial ports
  - Syscalls include read, write, ioctl, mmap, …

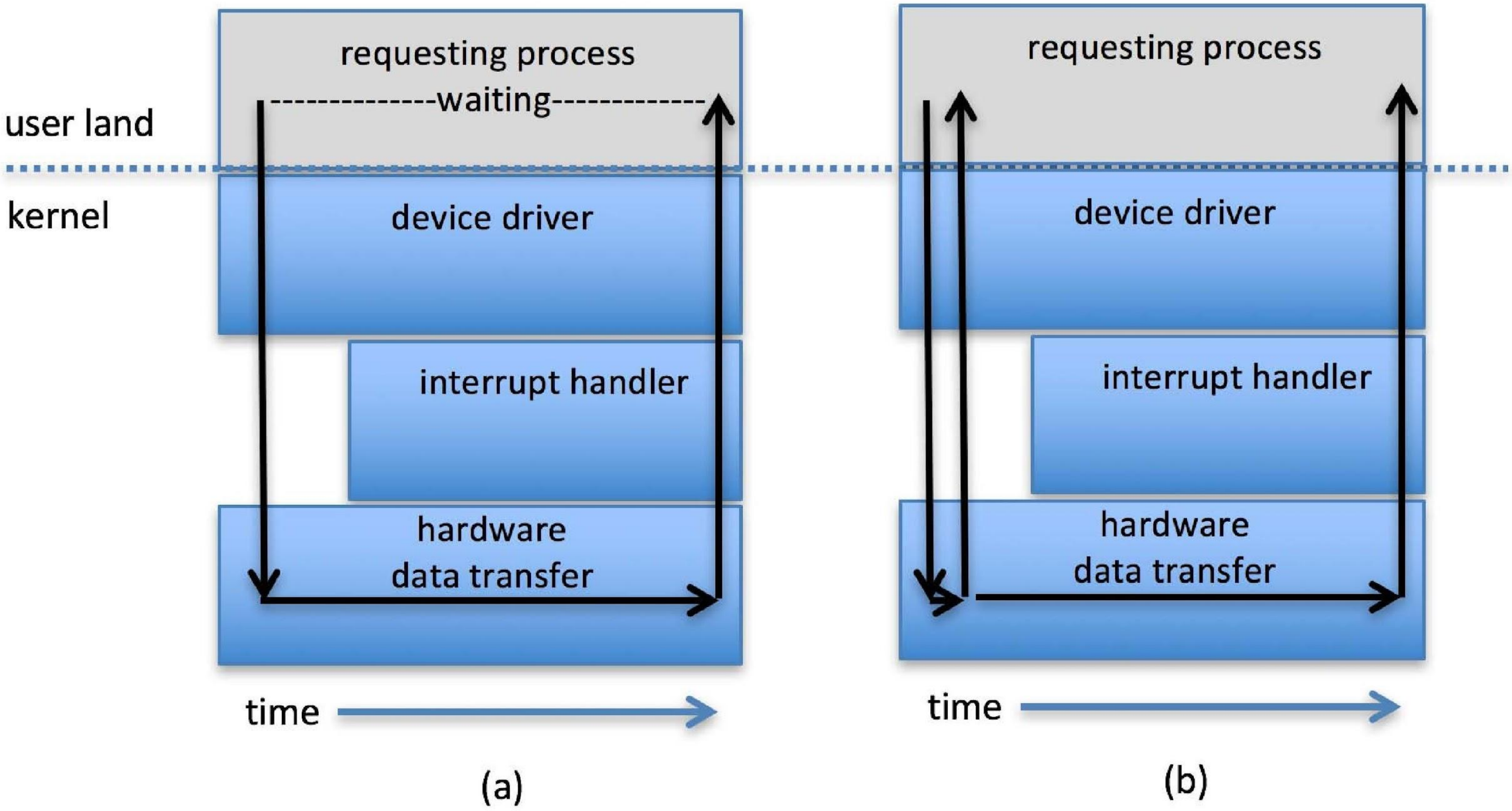
Network Devices

- Varying enough from block and character to have own interface
- Linux, Unix, Windows and many others include socket interface
  - Separates network protocol from network operation

Synchronous and Asynchronous I/O

- **Synchronous**
    - **Use blocking syscalls**
        - process suspended until I/O completed
        - Easy to use and understand
        - Insufficient for some needs
- **Asynchronous**
    - **Use Nonblocking syscalls**
        - I/O call returns as much as available or returns an error
        - How can process know when I/O is complete?
            - Can poll using syscall (e.g., poll, epoll, select syscalls)
            - Kernel can inform the process (e.g., with a signal) when I/O data is ready (Linux AIO framework)

Two I/O Methods



Scatter/gather (vectored) I/O

- **Scatter/gather I/O** allows one system call to perform multiple I/O operations
- For example, Unix **`readv()/writev()`** accept a vector of multiple buffers to read into or write from
- This scatter-gather method better for performance than multiple individual I/O calls
  - Decreases context switching and system call overhead