ECE437/CS481

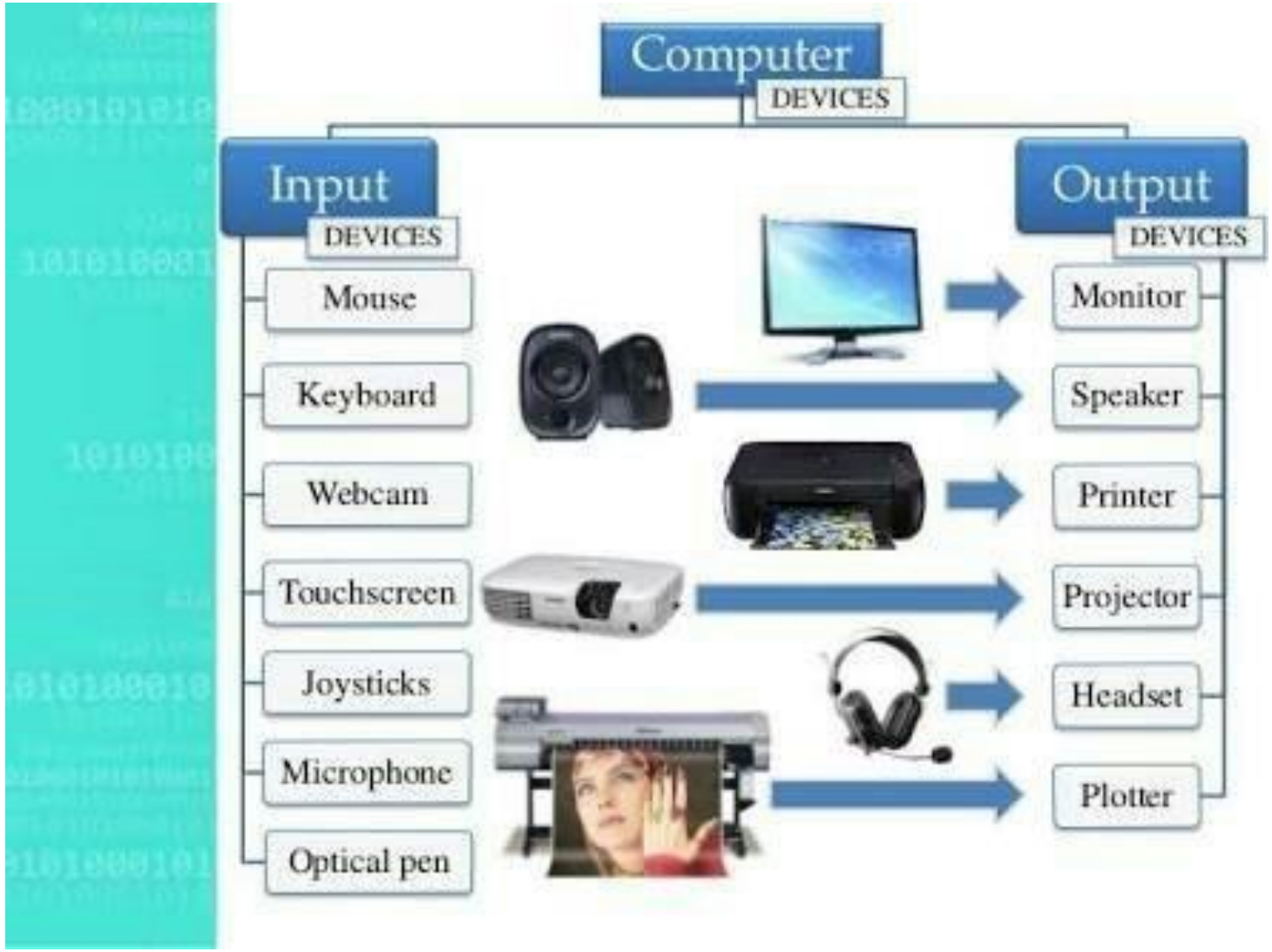
M06D: I/O SYSTEMS

CHAPTER 13

Xiang Sun

The University of New Mexico

Input and Output



© by Dr. X. Sun

Input and Output

- A computer's job is to process data
  - Computation (CPU, cache, and memory)
  - Move data into and out of a system (among I/O devices, memory and CPU)

- Challenges with I/O devices
  - Different categories: storage, networking, displays, etc.
  - Large number of device drivers to support
  - Device drivers run in kernel mode and can crash systems

- Goals of the OS
  - Provide a generic, consistent, convenient and reliable way to access I/O devices
  - As device-independent as possible
  - High performance I/O

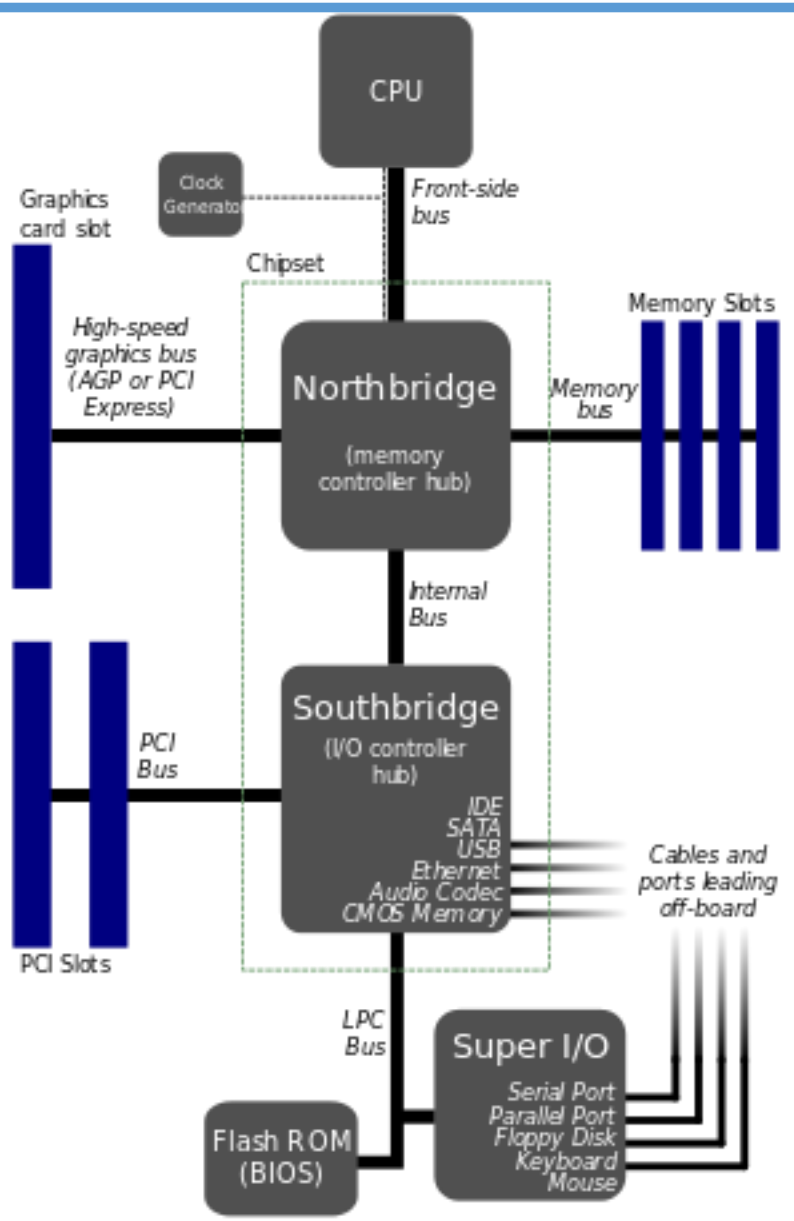
© by Dr. X. Sun

I/O System

❑ Hardware for data transferring

- ➢ Northbridge
  - ✓ Memory control hub
  - ✓ Links the CPU to very high-speed devices, e.g., memory and graphic card
  - ✓ Obsolete in modern computer architecture.

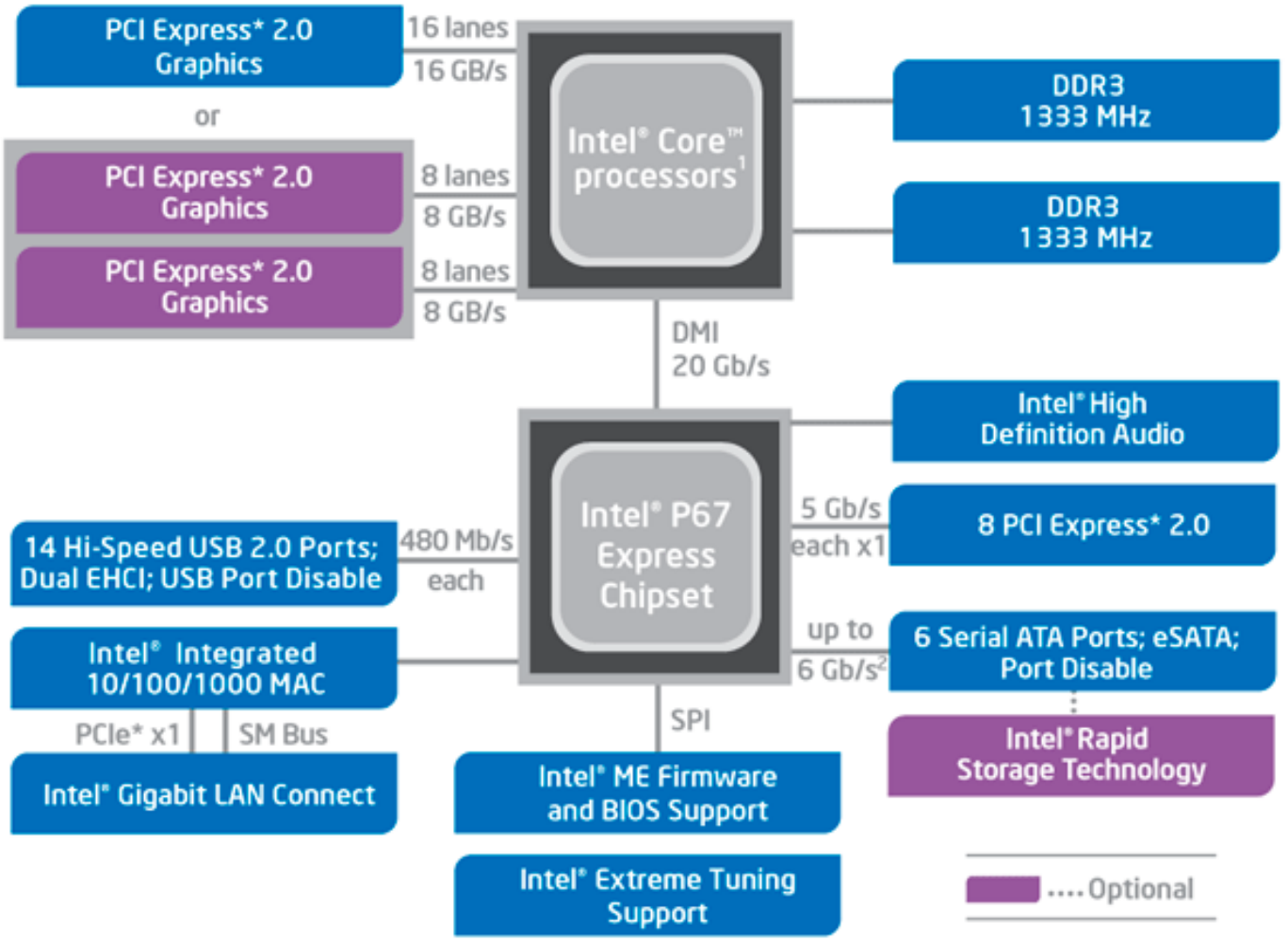
- ➢ Southbridge
  - ✓ I/O control hub
  - ✓ Connects to low-speed I/O devices, e.g., hard drives, USB devices, etc.



© by Dr. X. Sun

I/O System

- Communications between CPU and I/O devices
  - PCIe slot(s) for the GPU(s) and the DIMM slots for the RAM are now connected directly to the processor. The functionality previously provided by the Northbridge has now been integrated into the CPU.
  - Southbridge is referred to as the chipset and still handles the functionality that was handled by the Southbridge. However, it has better throughput than what the South Bridge used to have.



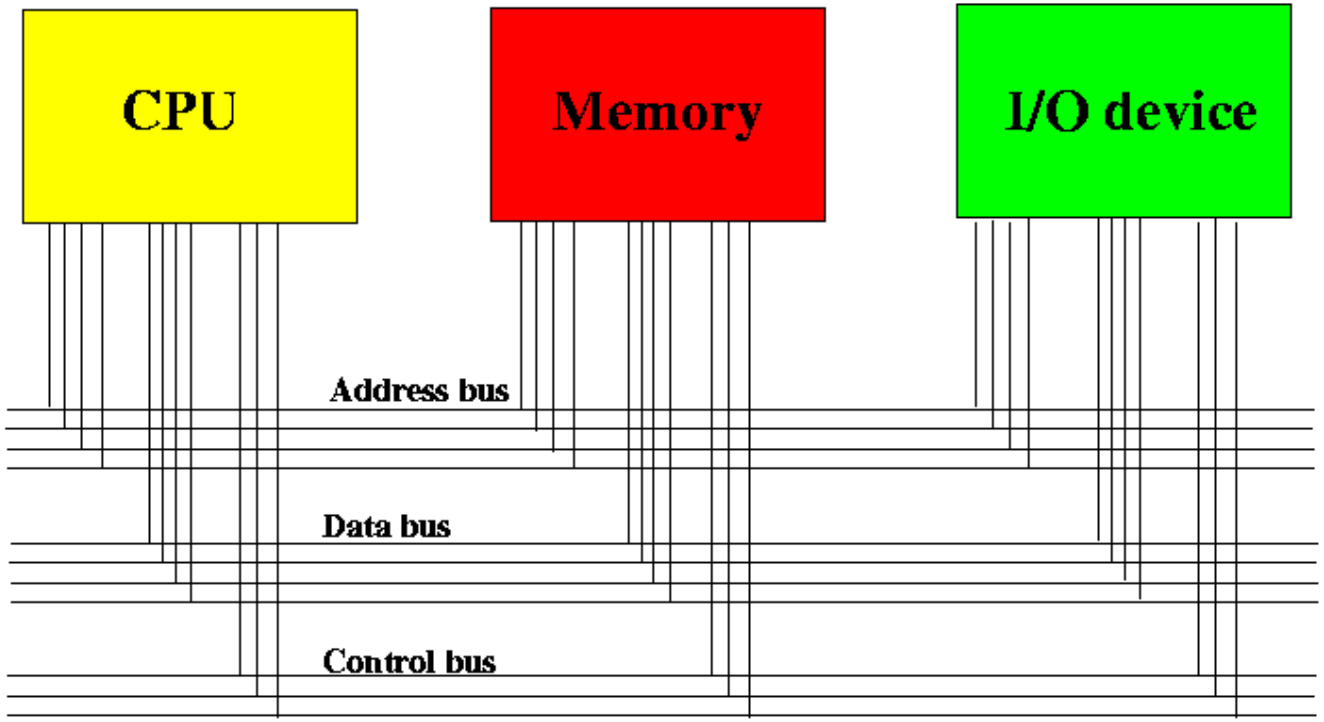
[1] Compatible with 2nd generation Intel® Core™ processor family

[2] All SATA ports capable of 3 Gb/s. 2 ports capable of 6 Gb/s.

© by Dr. X. Sun

I/O System

- Communications between CPU and I/O devices
  - Three types of buses
    - ✓ Data bus: transfer the data (information) between the CPU and other devices.
    - ✓ Address bus: identify the address of data
    - ✓ Control bus: identify the action on the address and data busses.

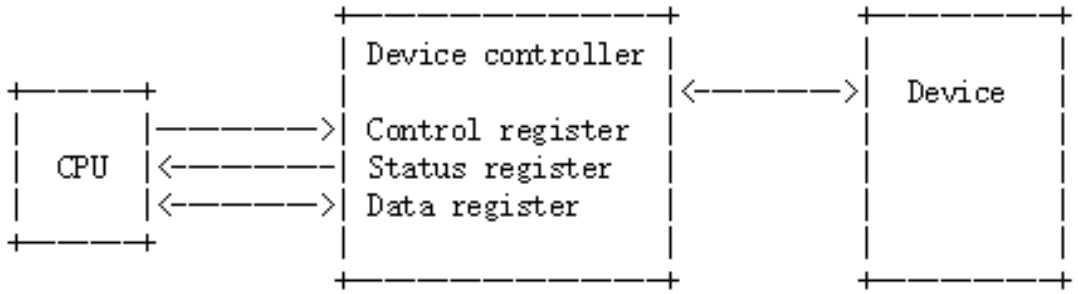


© by Dr. X. Sun

I/O System

❑ Addressing I/O devices

- CPU controls IO devices
  - ✓ by writing some command code into the command registers.
  - ✓ by reading their status registers to obtain the status (e.g., idle, busy) of I/O devices



- I/O devices need to be assigned a unique ID or address, just like memory.

- There are 2 techniques to allow IO devices to be addressed:
  - ✓ Standard I/O
  - ✓ Memory-mapped I/O

Main memory

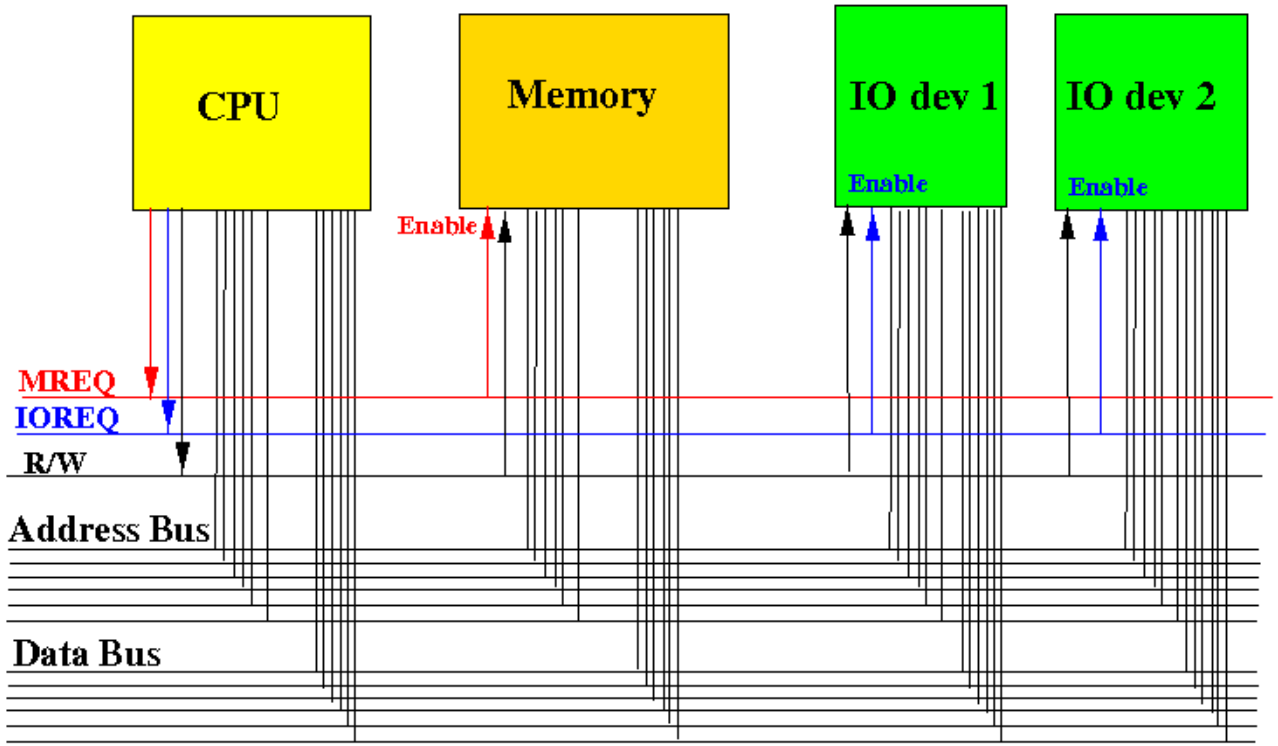
Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

© by Dr. X. Sun

I/O System

❑ Addressing I/O devices—Standard I/O

- In the standard IO technique, IO devices and memory location have separate address spaces
  - ✓ Two "spaces" are separate if there is an external (explicit) signal that identify the space.
  - ✓ For example,
    - the MREQ signal is used to signal memory requests. When the CPU wants to access (read/write) a memory location, it asserts the MREQ signal.
    - the IOREQ signal is used to signal IO requests. When the CPU wants to access (read/write) an IO device, it asserts the IOREQ signal.



- **MOV assembler instruction** to access the memory - hence, when the CPU executes a load or store instruction, it will assert the **MREQ signal**.
- **IN** (input) and **OUT** (output) **assembler instruction** to access IO devices - hence, when the CPU executes a input or output instruction, it will assert the **IOREQ signal**.

© by Dr. X. Sun

I/O System

❑ Addressing I/O devices—Memory-mapped I/O

- IO devices and the main memory share the same address space.
  - ✓ One set of memory addresses is for main memory operations
  - ✓ The other set of memory address is for I/O operations.
  - ✓ The 2 sets of addresses must be disjoint (no overlap) !!!
  - ✓ For example, last 4 bits of the address = 1111, then the CPU wants to address an IO device. Otherwise, the CPU wants to address the memory

- Three types of addresses are provided:
  - ✓ Read only address
  - ✓ Write only address
  - ✓ Read and write address

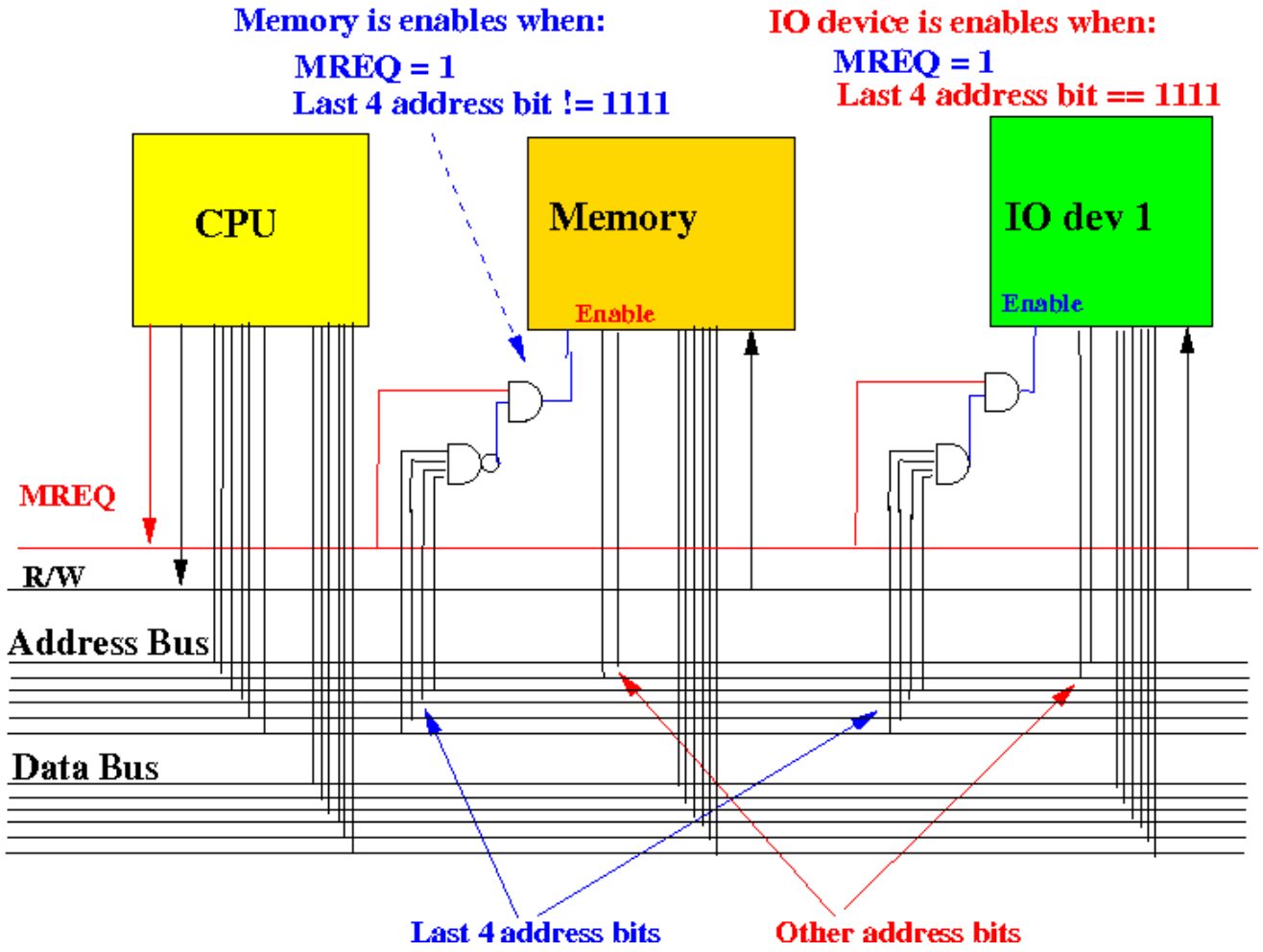
Address (binary)	Use for
00000000 00000000 00000000 00000000	Memory
00000000 00000000 00000000 00000001	Memory
00000000 00000000 00000000 00000011	Memory
....	
11100000 00000000 00000000 00000000	Memory
11100000 00000000 00000000 00000000	Memory
....	
11101111 11111111 11111111 11111111	Memory
11110000 00000000 00000000 00000000	IO device
11110000 00000000 00000000 00000001	IO device
....	
11111111 11111111 11111111 11111111	IO device

© by Dr. X. Sun

I/O System

Addressing I/O devices—Memory-mapped I/O

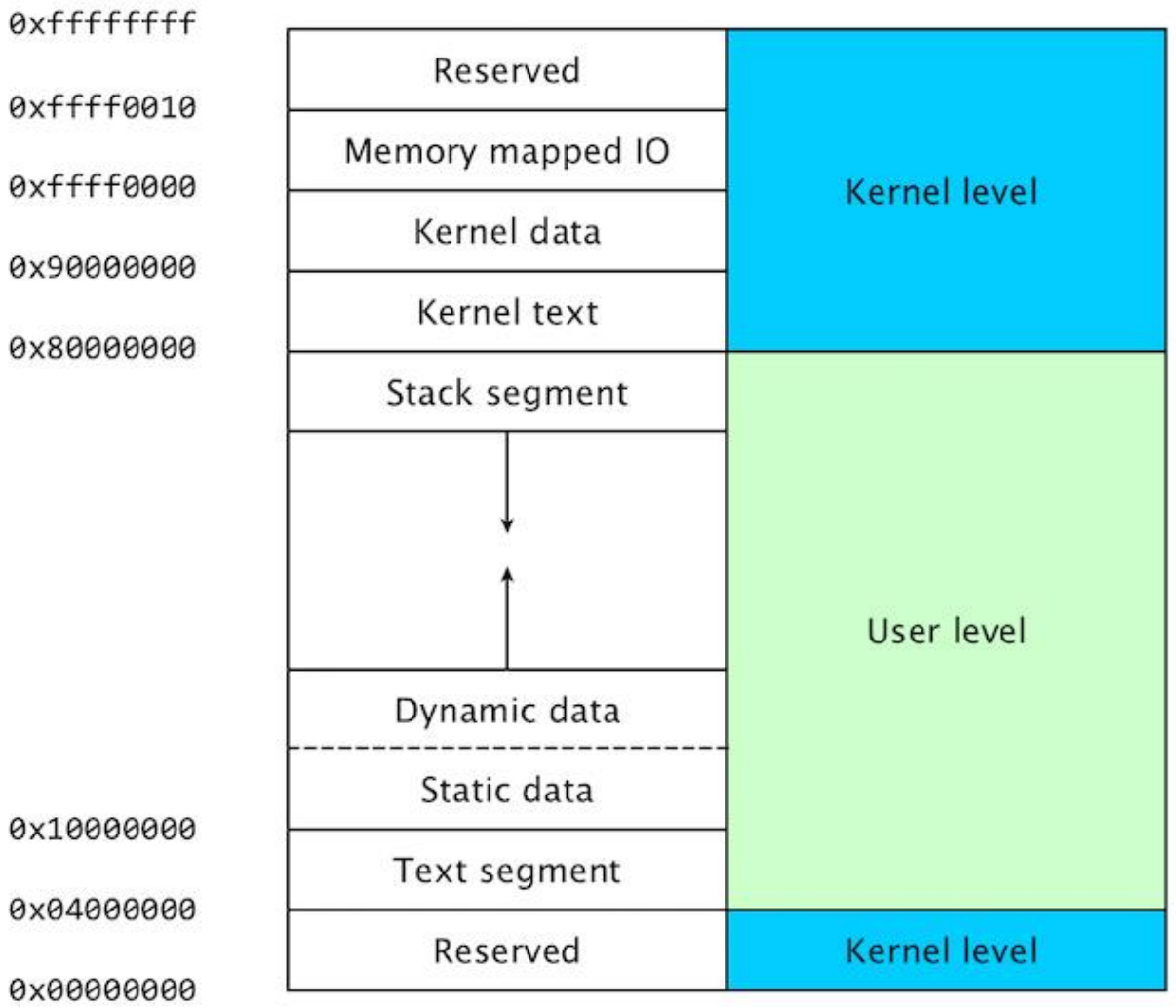
- The following circuit can determine whether the CPU wants to address the memory or an I/O device.
- When the CPU sends out an address, there is no way to tell by only looking at all the signals from the CPU if the address was for a memory location or for an IO device.
- That is, there is no need to have 2 different types of assembler instructions (one type for accessing memory and the other for accessing IO device).



© by Dr. X. Sun

I/O System

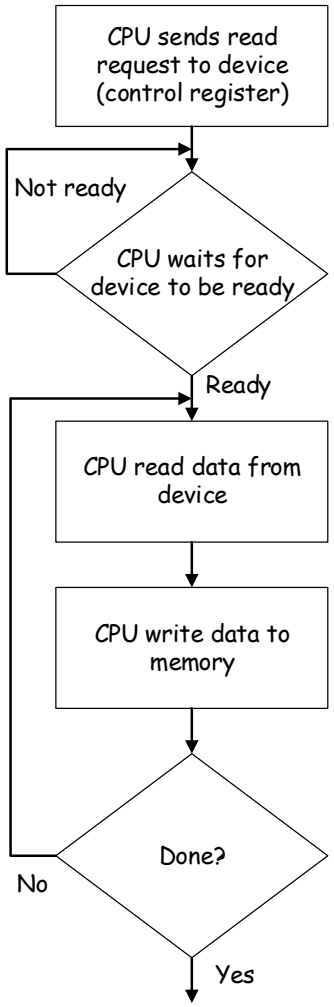
- Addressing I/O devices—Memory-mapped I/O

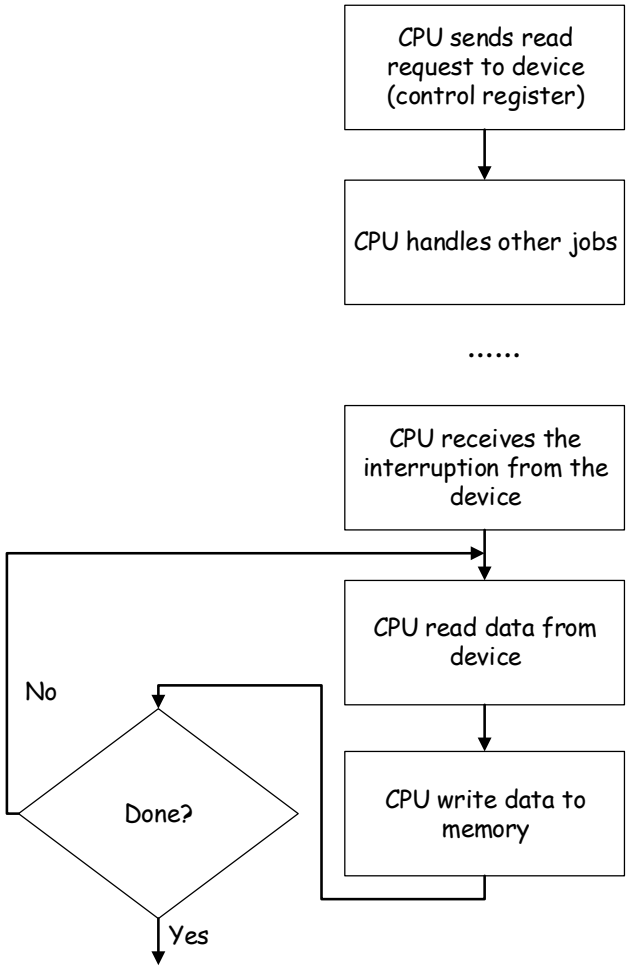


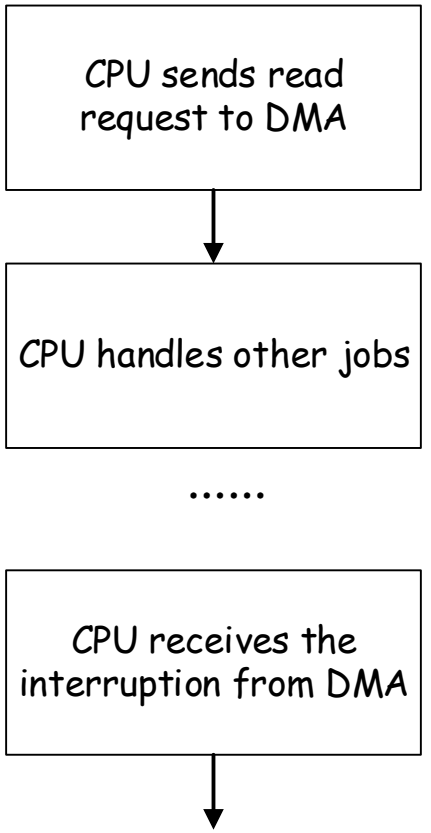
© by Dr. X. Sun

I/O System

- Data transferring between CPU and I/O devices
  - Recall: three methods of achieving data transferring:
    - ✓ Programmed I/O.
    - ✓ Interrupt- initiated I/O.
    - ✓ Direct memory access (DMA).



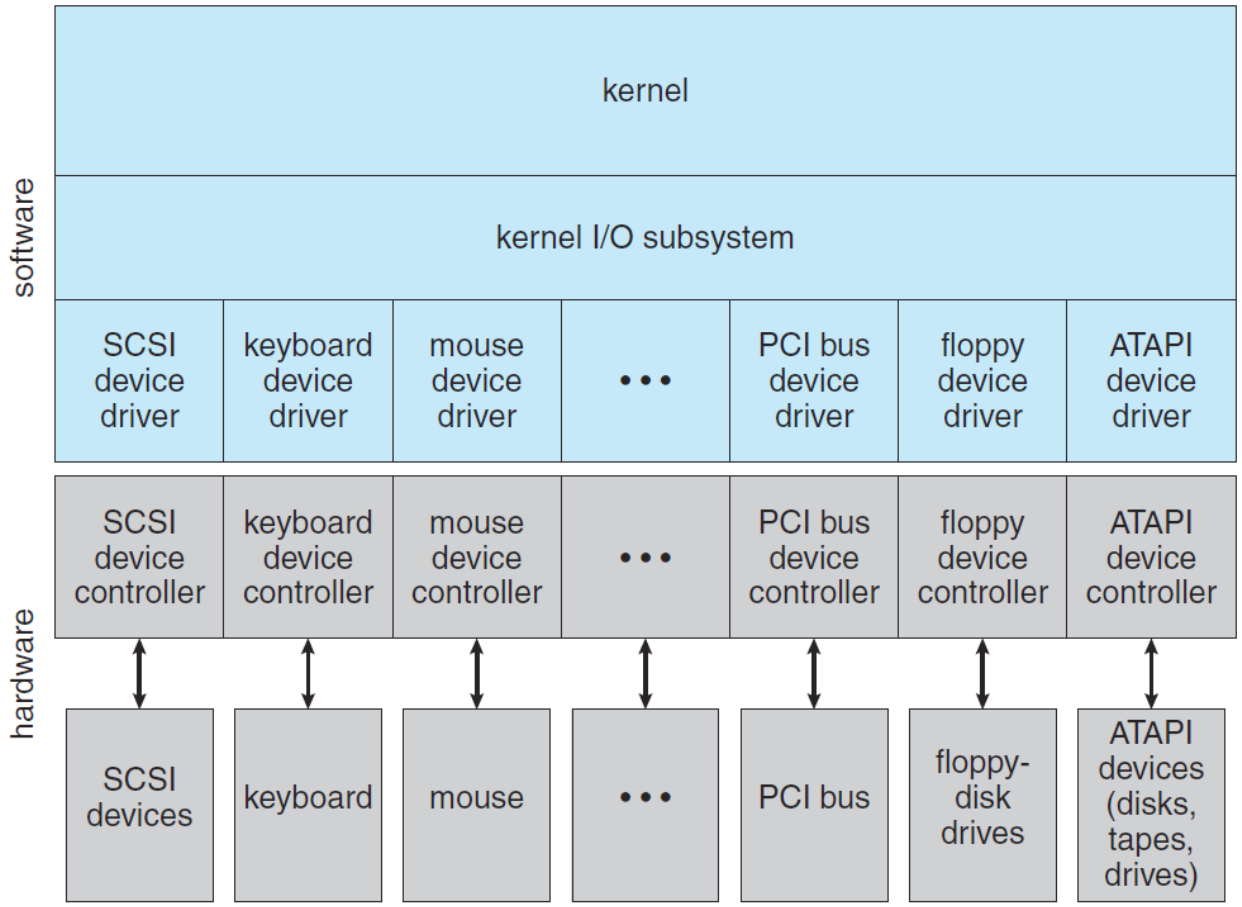




© by Dr. X. Sun

I/O System

- Hierarchical architecture of the I/O system



© by Dr. X. Sun

I/O System

Device controller—OS/user view

- Control register
- Data register
- Status register

Upper-half
- Device initialization
- Accept read-write request from the OS
- Check input parameters

Lower-half
- Start the device if necessary (e.g., start spinning the CD-ROM)
- Check if device is available: if not, wait
- Issue command(s)
- Wait for results
  - Busy wait (awakened by interrupt)
  - Block
- Check for possible errors
- Return results

Device driver

- Each type of device controller should have a specific device driver designed for it.
- A device driver can provide functionality through two separate driver components:
  - ✓ The upper-half part: interact with application and operating system—hardware independent.
  - ✓ The lower-half part: Provide the interface with device controller by accessing various registers.

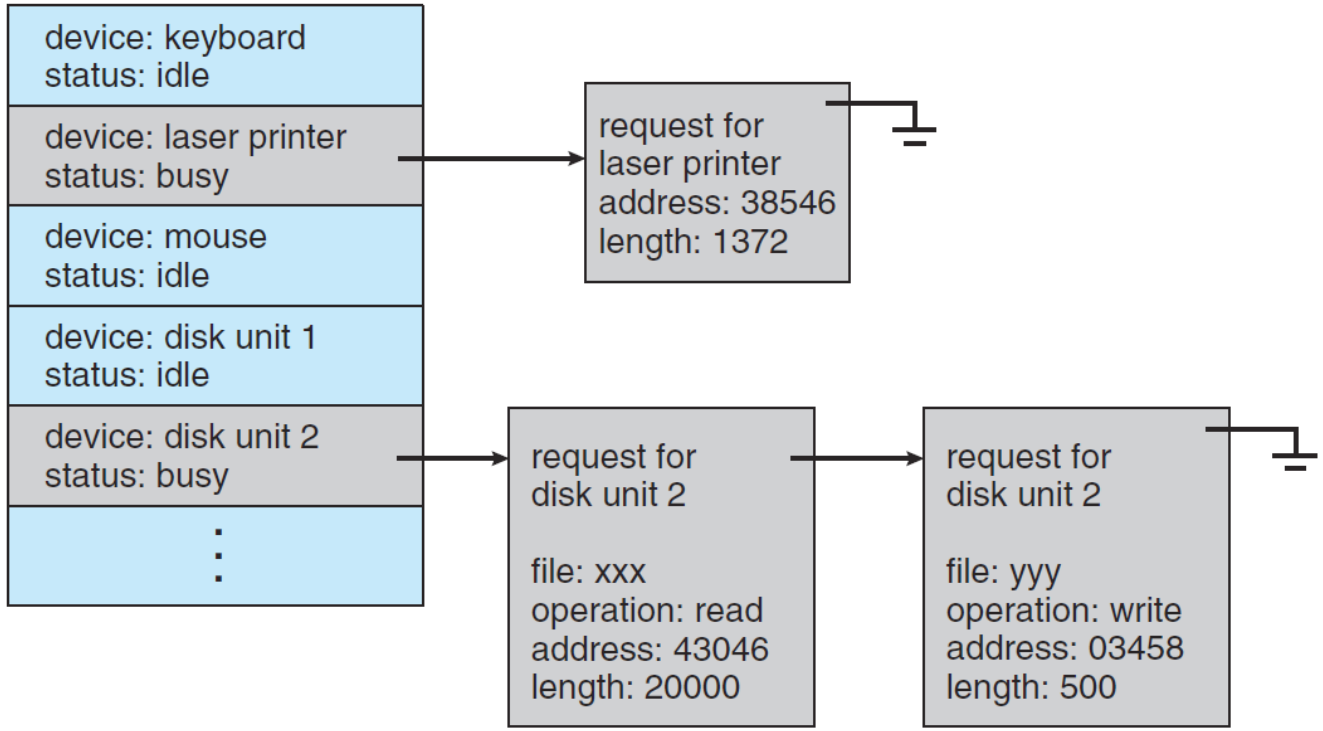
© by Dr. X. Sun

I/O System

Kernel I/O subsystem

1. I/O scheduling

- Kernel maintain a device-status table.
- If the device is busy with a request, the type of request and the related parameters will be stored in the table entry for that device.
- The OS may provide some scheduling algorithm to make IO operations more efficient.



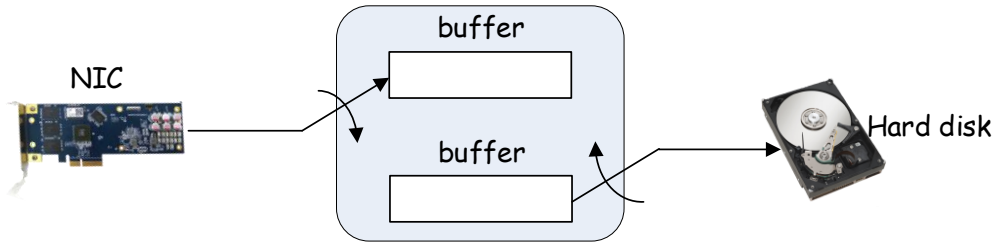
© by Dr. X. Sun

Kernel I/O subsystem

2. Buffering

- A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

- The reasons of providing buffering includes:
  - ✓ cope with a speed mismatch between two devices. For example, a file received by the NIC (which has a lower speed) need to be stored on the hard disk (which has a higher speed).--- spooling



  - ✓ Provide adaptations for devices that have different data-transfer sizes. For example, in order to transmit a video from a disk to anther host via the NIC. The video has to be fragmented into small network packages.

© by Dr. X. Sun

I/O System

Kernel I/O subsystem

3. Caching

- A cache is a memory area that stores copies of popular/recently accessed data. Access to the cached copy is more efficient than accessing to the original.
  - ✓ For instance, data stored in a disk could be cached in the memory (or even copied again in the CPU's secondary and primary caches) based on the hit rate.
  - ✓ A cache is different from a buffer: 1) different purposes; 2) a buffer may hold the only existing copy of a data; while a cache may have one of the copies of data.

4. Error handling

- I/O transferring can fail in many ways.
  - ✓ Transient failure
  - ✓ Permanent failure
- The kernel I/O subsystem can 1) deal with transient failures by attempting the data re-transmission; 2) return information to help with failure analysis.

© by Dr. X. Sun

I/O System

Kernel I/O subsystem

5. I/O protection

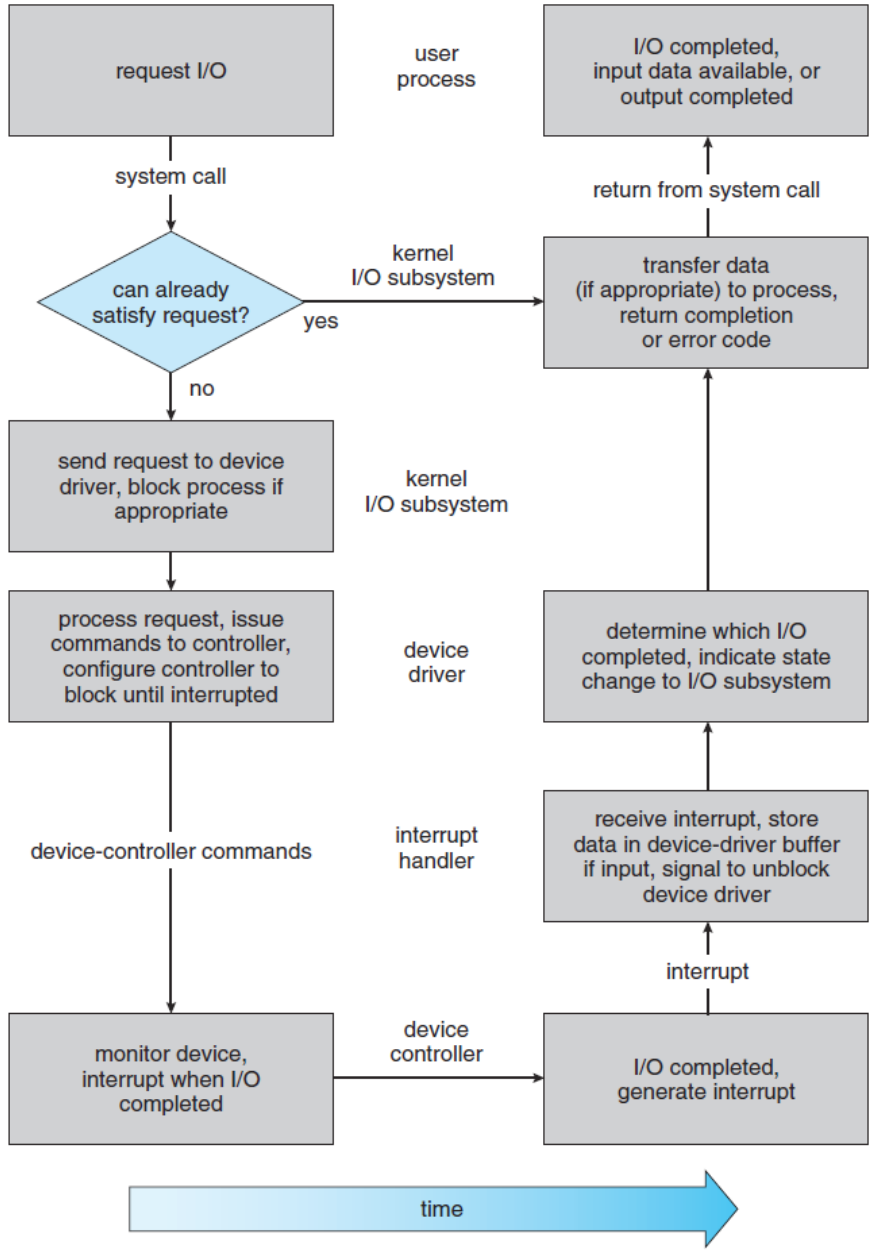
- Users cannot issue I/O instructions directly; they must do it through the operating system.
  - ✓ To conduct I/O operations, a user program executes a system call to request the OS to perform I/O on its behalf.
  - ✓ I/O subsystem in the OS validate the request. If it is, it process the request.
- Memory-mapped I/O locations must be protected from user access.

© by Dr. X. Sun

I/O System

❑ The life cycle of an I/O request

- ✓ A process issues a read() request, the kernel I/O subsystem first if the data are already available in the memory.
- ✓ If not, the process is removed from the run queue and is placed on the wait queue. Then, the kernel I/O subsystem sends the request to the related device driver.
- ✓ The device driver allocates space to receive the data, and sends the commands to the device controller by writing into the control register of the device.
- ✓ The device controller operates the device hardware to perform the data transfer. <assume that DMA is applied>
- ✓ Once data transfer is completed, DMA controller generates an interrupt to the interrupt handler, which signal the device driver.
- ✓ The device driver signals the kernel I/O subsystem that the I/O request has been completed.
- ✓ The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.



© by Dr. X. Sun