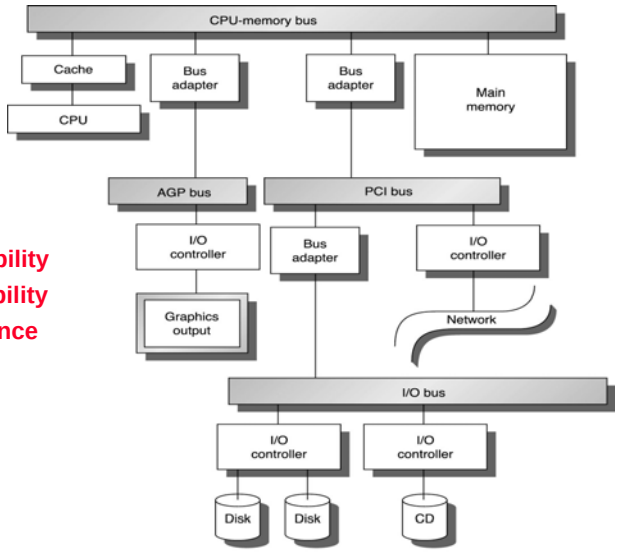
CS420/520
Computer Architecture I

OS's Responsibilities in I/O

Dr. Xiaobo Zhou
Department of Computer Science

Review: Interfacing Storage Devices to the CPU

- **Dependability**
- **Expandability**
- **Performance**



A typical interface of I/O devices and an I/O bus to the CPU-memory bus

Operating System Requirements in I/O Operations

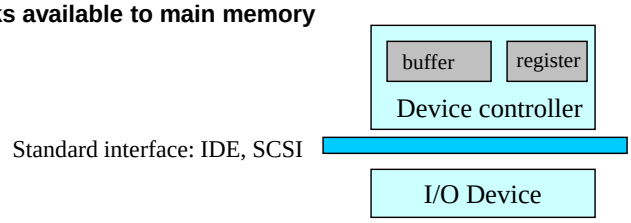
- Provide protection to shared I/O resources
  - Guarantees that a user's program can only access the portions of an I/O device to which the user has rights
- Provides OS abstraction for accessing devices:
  - Supply routines that handle low-level device operation
- Handles the interrupts generated by I/O devices
- Provide equitable access to the shared I/O resources
  - All user programs must have equal access to the I/O resources
- Schedule accesses in order to enhance system throughput

OS and I/O Systems Communication Requirements

- The Operating System must be able to prevent:
  - The user program from communicating with the I/O device directly
- If user programs could perform I/O directly:
  - Protection to the shared I/O resources could not be provided
- Three types of communication are required:
  - The OS must be able to give commands to the I/O devices
  - The I/O device must be able to notify the OS when the I/O device has completed an operation or has encountered an error
  - Data must be transferred between Memory and an I/O device

RE: Device Controllers

- I/O devices have components:
  - mechanical component
  - electronic component
- The electronic component is the *device controller*
  - may be able to handle multiple but identical devices
- Controller's tasks
  - convert serial bit stream to block of bytes
  - perform error correction as necessary
  - make blocks available to main memory



CPU→I/O: Special I/O Instructions

- How the CPU communicates with the control registers and the device data buffers? Two methods are available.
- Method I: Special (assembly) I/O Instructions
  - Each control register is assigned an I/O *port* number (device #)

```
IN REG, PORT
OUT PORT, REG
```

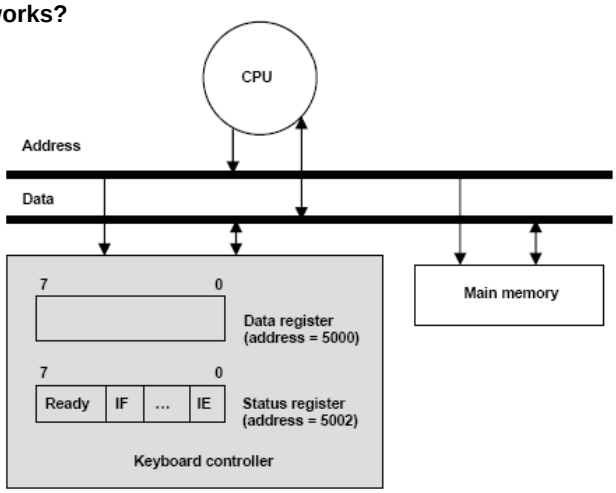
  - The memory address space and I/O address space are different

```
IN R0, 4        ; port 4
MOV R0, 4       ; memory word 4
```

I/O operations require assembly language programming

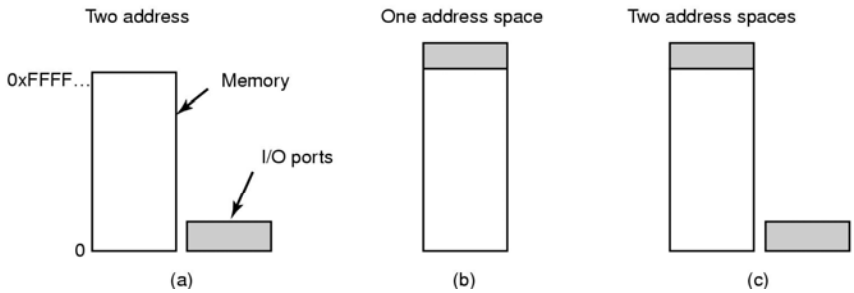
CPU→I/O: Memory-mapped I/O

- Method II: Memory-mapped I/O
  - Each register is assigned a unique memory address to which no memory is assigned; read/write to addresses are I/O operations
  - How it works?



CPU→I/O: Memory-mapped I/O (2)

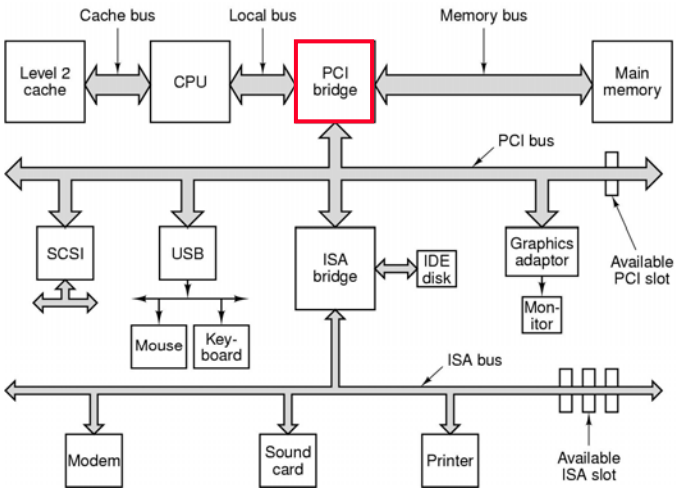
- Advantages and Disadvantages of Memory-mapped I/O
  - Pro: an I/O device driver can be written in C, instead of assembly
  - Con: how about caching a control register which says busy? What if a bus hierarchy used?
  - More in Operating Systems



(a) Separate I/O and memory space; (b) Memory-mapped I/O; (c) Hybrid
(PDP-11) (Pentium)

CPU→I/O: Memory-mapped I/O (3)

° PCI bridge chip filters addresses to different buses



Structure of a large Pentium system

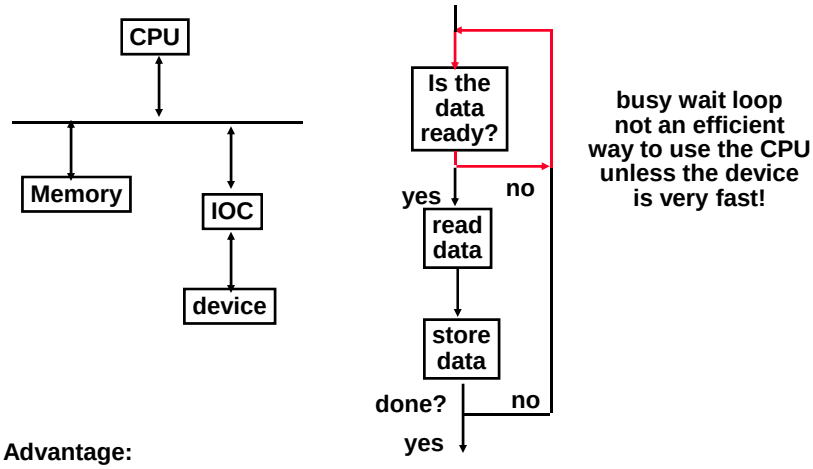
I/O→ CPU: Device Notifying the CPU

° The OS needs to know when:
- The I/O device has completed an operation
- The I/O operation has encountered an error

° This can be accomplished in two different ways:
- Polling (Busy Waiting):
  - The I/O device put information in a status register
  - The CPU periodically or continuously check the status register
- I/O Interrupt:
  - Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing.

In real-time systems, a hybrid approach is often used
- Use a clock to periodically interrupt the CPU, at which time the CPU polls all I/O devices

Data Transfer: Programmed I/O (Busy Waiting)



busy wait loop not an efficient way to use the CPU unless the device is very fast!

- Advantage:
  - Simple: the processor is totally in control and does all the work
- Disadvantage:
  - Polling overhead can consume a lot of CPU time

Example 1: Overhead of polling

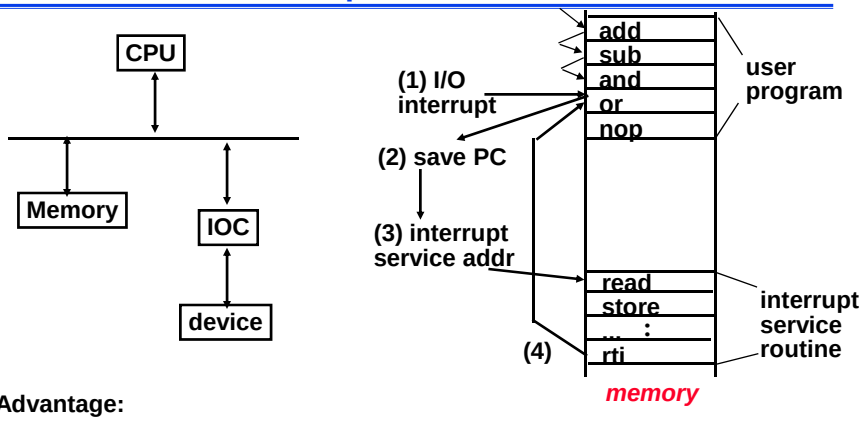
- Assume that the number of clock cycles for a polling operation is 400 and that the processor executes with a 500 MHz clock
- Determine the fraction of CPU time consumed for the following three cases, assuming that you poll often enough so that no data is ever lost and assuming that the devices are potentially always busy:
  - mouse must be polled 30 times per second (polling rate)
  - floppy disk transfers data to CPU in 16-bit units and has a data rate of 50 KB/sec.
  - hard disk transfers data four-word chunks and can transfer at 4MB/sec.

Mouse: cycles/sec for polling: 30 x 400 → fraction = 12 000 / 500 M = 0.002%

Floppy disk: cycles/sec for polling: 50 KB/sec / 2 B/polling x 400 = 25 K x 400

Hard disk: cycles/sec for polling: 4 MB/sec / 16B/polling x 400 = 250 K x 400

Data Transfer: Interrupt Driven



- Advantage:
  - User program progress is only halted during actual transfer
- Disadvantage, special hardware is needed to:
  - Cause an interrupt (I/O device)
  - Detect an interrupt (processor) (EPC, Cause Reg, Controls, etc)
  - Save the proper states to resume after the interrupt (processor)

Exception / I/O Interrupt

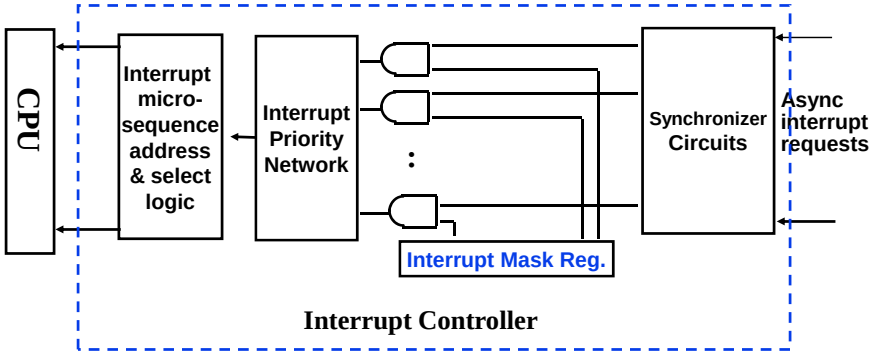
- Exception and interrupts – events other than branches and jumps that change the normal flow of instruction execution.
  - Exception is within (or outside) the Proc, e.g., overflow
  - Interrupt comes from outside (I/O devices)
- An I/O interrupt is just like the exceptions except:
  - An I/O interrupt is asynchronous
  - Further information needs to be conveyed
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction
  - I/O interrupt does not prevent any instruction from completion
    - You can pick your own convenient point to take an interrupt
    - Recall unpredictability in Pipelining Data Hazards
- I/O interrupt is more complicated than exception:
  - Needs to convey the identity of the device generating the interrupt
  - Interrupt requests can have different urgencies:
    - Interrupt request needs to be prioritized

Multiple Interrupts

- **Sequential Order**
  - Disable interrupts so processor can complete task, and processor ignores any new interrupt request signals
  - Interrupts remain pending until the processor enables interrupts
  - After interrupt handler routine completes, the processor checks for additional interrupts
- **Priorities**
  - Higher priority interrupts cause lower-priority interrupts to wait
  - Causes a lower-priority interrupt handler to be interrupted
  - Example when input arrives from communication line, it needs to be absorbed quickly to make room for more input

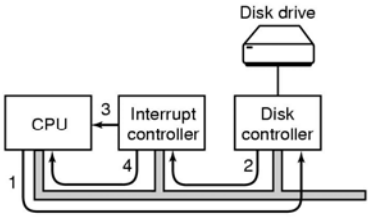
Interrupt Controller Logic

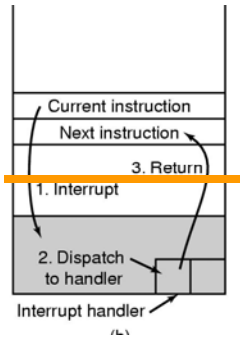
- Detect and synchronize interrupt requests
  - Ignore interrupts that are disabled (masked off)
  - Rank the pending interrupt requests
  - Create interrupt microsequence address
  - Provide select signals for interrupt microsequence



Program Interrupt/Exception Hardware

° Hardware interrupt services:
  - Save the PC (or PCs in a pipelined machine)
  - Inhibit the interrupt that is being handled
  - Branch to interrupt service routine
  - A "good thing" about interrupt:
    - Asynchronous: not associated with a particular instruction
    - Pick the most convenient place in the pipeline to handle it





Step 1: diver tells disk controller the job
Step 2: an interrupt is generated when job finished
Step 3: if interrupt controller is ready, pin to CPU
Step 4: interrupt controller gives the device # to CPU

Example 2: Overhead of Interrupt

° Consider the same hard disk and processor in Example 1 (Polling):
  - hard disk transfers data four-word chunks and can transfer at 4MB/sec.
  - the processor executes with a 500MHz clock

° Assume that the overhead for each transfer, including the interrupt, is 500 clock cycles

° Determine the fraction of CPU time consumed if the hard disk is only transferring data 5% of the time.

cycles/sec for interrupt: 4MB/sec / 16B/polling x 500 = 250 K x 500

Fraction consumed during a transfer: 250 K x 500 / 500 M = 25%

Fraction consumed on average = 25% x 0.5% = 1.25 %

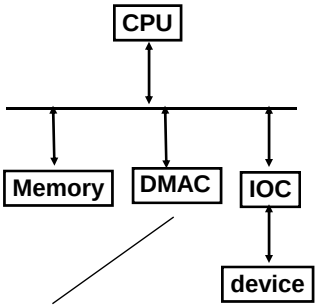
* reminder: it was 20% for polling

Delegating I/O Responsibility from the CPU: DMA

° How to reduce the CPU time spent in transferring data?
  - Recall output/input operations

° Direct Memory Access (DMA):
  - A specialized processor that transfer data between memory and I/O device while the CPU goes on with other tasks.
  - External to the CPU
  - Act as a maser on the bus
  - Transfer blocks of data to or from memory without CPU intervention
  - Once done, DMAC interrupts the CPU
  - More sophisticated DMA devices: scatter/gather, dealing with a list of separate addresses

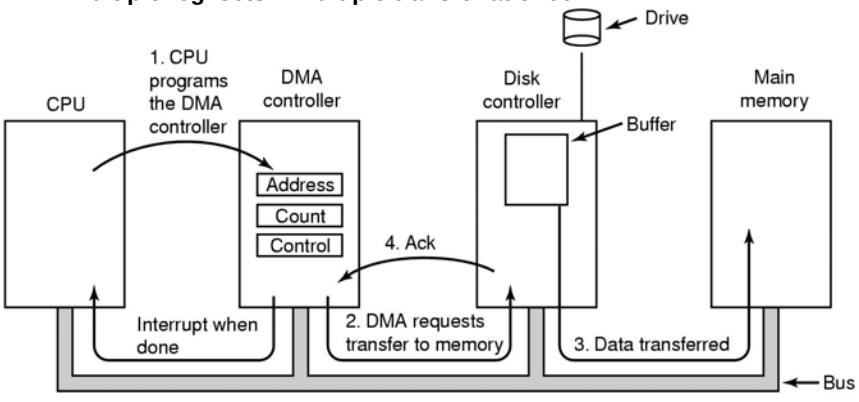
CPU sends a starting address, direction, and length count to DMA registers. Then, issues "start".



DMAC provides handshake signals for Peripheral Controller, and Memory Addresses and handshake signals for Memory.

Direct Memory Access (DMA)

° DMA controller has access to system bus independent of CPU
  - A memory address register
  - A byte count register
  - A control register (direction, transfer unit, and transfer mode)
  - Multiple reg. sets if multiple transfer at once



Operation of a DMA transfer

Example 3: Overhead of DMA

- Consider the same hard disk and processor in Examples 1 and 2:
  - hard disk transfers data four-word chunks and can transfer at 4MB/sec.
  - the processor executes with a 500MHz clock
- Assume the initial setup of a DMA transfer takes 1000 clock cycles
- the handling of the interrupt at DMA completion requires 500 cycles
- Determine the fraction of CPU time consumed if the hard disk is actively transferring 100% of the time and the average transfer from the disk is 8KB

I/O Summary:

- Storage systems and three-stage disk access
- RAID: non redundancy(0), mirroring(1), bit-interleaved parity (3), distributed block-interleaved parity (5)
- Three types of buses and bus hierarchy:
  - Processor-memory buses
  - I/O buses
  - Backplane buses
- Two methods are used to address I/O device:
  - Special I/O instructions
  - Memory-mapped I/O
- I/O device notifying the operating system:
  - Polling: it can waste a lot of processor time
  - I/O interrupt: similar to exception except it is asynchronous
- Delegating I/O responsibility from the CPU
  - Direct memory access (DMA)