

# CMPE 411

## Computer Architecture

### Lecture 27

## Interfacing I/O Devices

December 5, 2017

[www.csee.umbc.edu/~younis/CMPE411/  
CMPE411.htm](http://www.csee.umbc.edu/~younis/CMPE411/CMPE411.htm)



# Lecture's Overview

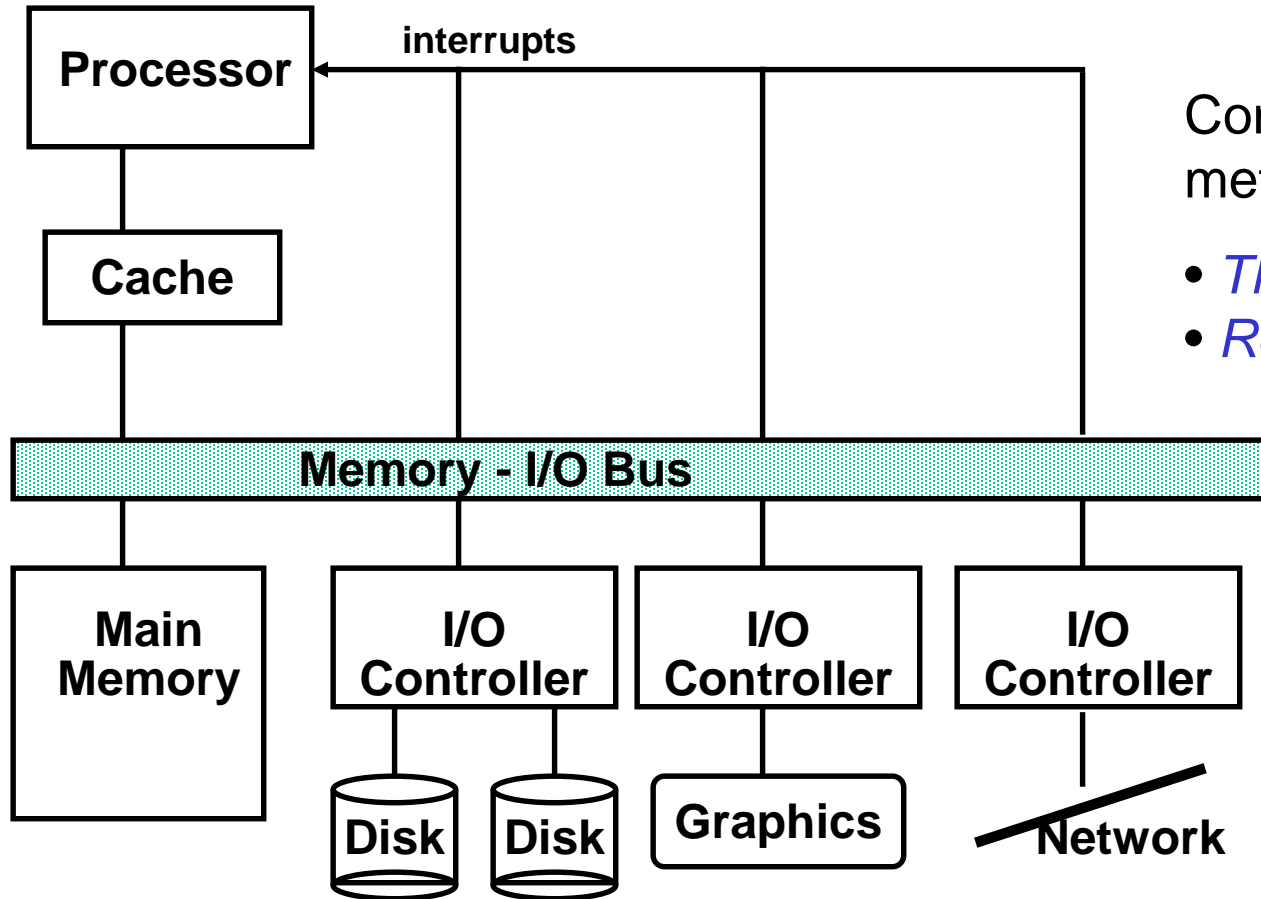
## Previous Lecture:

- Memory to processor interconnect
  - Definition of bus structure
  - Bus transactions
  - Types of buses
  - Bus Standards
- Bus Performance and Protocol
  - Synchronous versus Asynchronous buses
  - Bandwidth optimization factors
- Bus Access
  - Single versus multiple master bus
  - Bus arbitration approaches

## This Lecture:

- Interfacing I/O devices to memory, processor and OS

# Typical I/O System



Common performance metrics:

- *Throughput*: I/O bandwidth
- *Response time*: Latency

- ❑ The connection between the I/O devices, processor, and memory are usually called (local or internal) buses
- ❑ Communication among the devices and the processor use both bus protocols and interrupts

# Giving Commands to I/O Devices

Two methods are used to address the device:

## ① Special I/O instructions: (Intel 80X86, IBM 370)

- ➔ Specify both the device number and the command word
  - **Device number**: the processor communicates this via a set of wires normally included as part of the I/O bus
  - **Command word**: this is usually send on the bus's data lines
  - Each devices maintain status register to indicate progress
- ➔ Instructions are privileged to prevent user tasks from directly accessing the I/O devices

## ② Memory-mapped I/O: (Motorola/IBM PowerPC)

- ➔ Portions of the address space are assigned to I/O device
- ➔ Read and writes to those addresses are interpreted as commands to the I/O devices
- ➔ User programs are prevented from issuing I/O operations directly:
  - The I/O address space is protected by the address translation

# Communicating with I/O Devices

❑ 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:

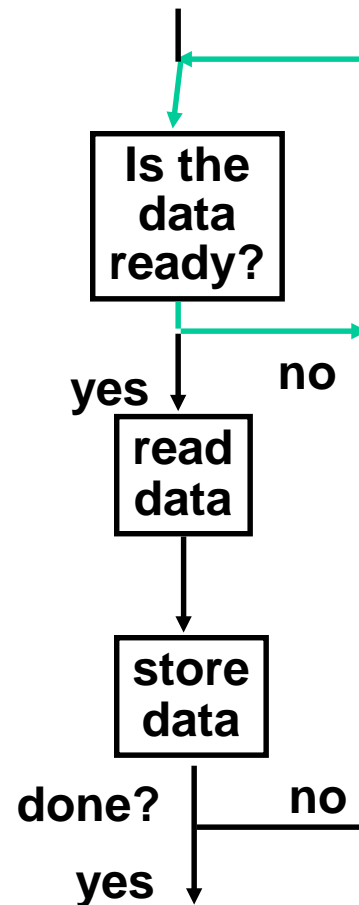
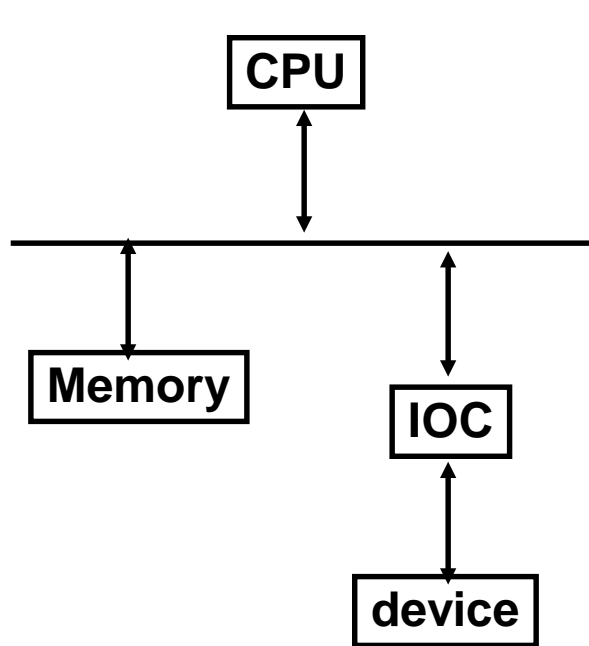
- The I/O device put information in a status register
- The OS periodically check the status register

## ➔ I/O Interrupt:

- An I/O interrupt is an externally stimulated event, asynchronous to instruction execution but does **NOT** prevent instruction completion
- Whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing
- Some processors deals with interrupts as special exceptions

These schemes requires heavy processor's involvement and suitable only for low bandwidth devices such as the keyboard

# Polling: Programmed I/O



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

**but checks for I/O  
completion can be  
dispersed among  
computation  
intensive code**

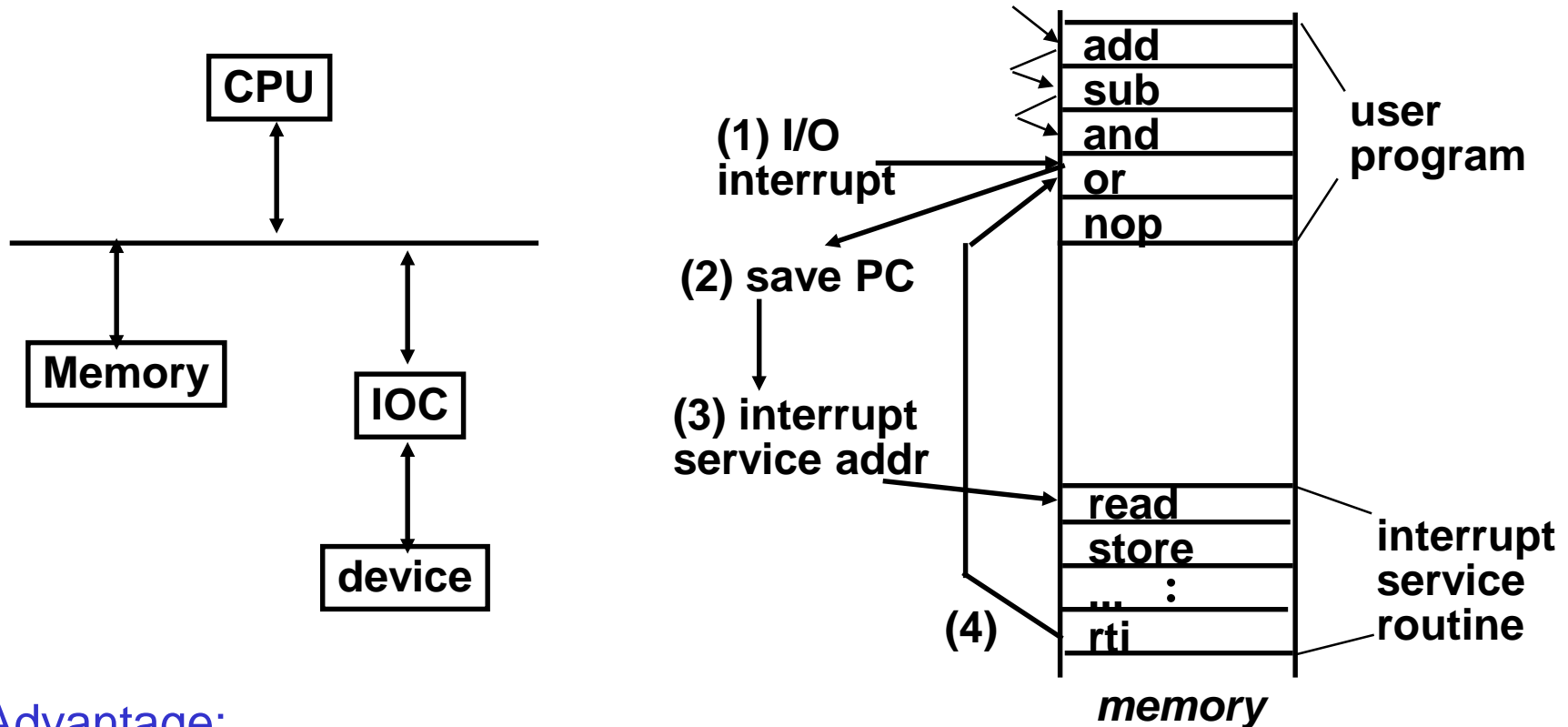
## ❑ Advantage:

➔ Simple: the processor is totally in control and does all the work

## ❑ Disadvantage:

➔ Polling overhead can consume a lot of CPU time

# Interrupt Driven Data Transfer



## □ 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)

➔ Save the proper states to resume after the interrupt (processor)

# I/O Interrupt vs. Exception

- ❑ An I/O interrupt is just like the exceptions except:
  - ➔ An I/O interrupt is asynchronous
  - ➔ Further information needs to be conveyed
  - ➔ Typically exceptions are more urgent than interrupts
- ❑ 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
- ❑ 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
    - Priority indicates urgency of dealing with the interrupt
    - high speed devices usually receive highest priority



# Direct Memory Access

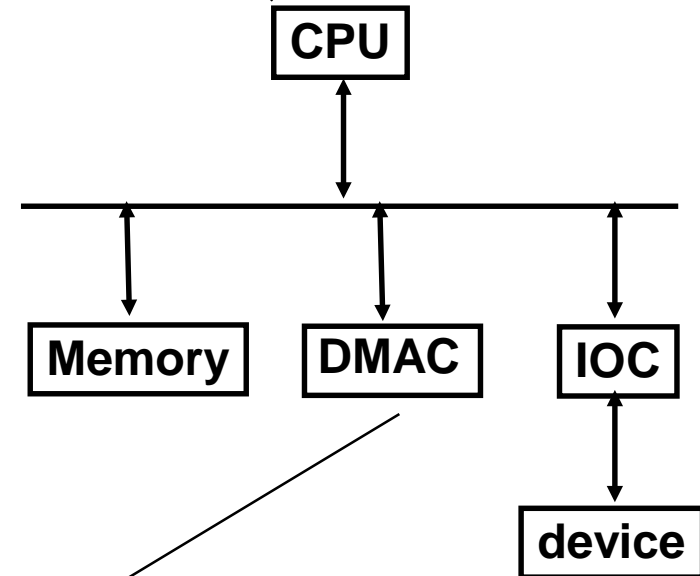
## □ Direct Memory Access (DMA):

- External to the CPU
- Use idle bus cycles (*cycle stealing*)
- Act as a master on the bus
- Transfer blocks of data to or from memory without CPU intervention
- Efficient for large data transfer, e.g. from disk
- ❖ Cache usage allows the processor to leave enough memory bandwidth for DMA

## □ How does DMA work?:

- CPU sets up and supply device id, memory address, number of bytes
- DMA controller (DMAC) starts the access and becomes bus master
- For multiple byte transfer, the DMAC increments the address
- DMAC interrupts the CPU upon completion

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



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

For multiple bus system, each bus controller often contains DMA control logic

# DMA Problems

- ① **With virtual memory systems:** (pages would have physical and virtual addresses)
- ➔ Physical pages re-mapping to different virtual pages during DMA operations
  - ➔ Multi-page DMA cannot assume consecutive addresses

## Solutions:

- ➔ Allow virtual addressing based DMA
  - Add translation logic to DMA controller
  - OS allocated virtual pages to DMA prevent re-mapping until DMA completes
- ➔ Partitioned DMA
  - Break DMA transfer into multi-DMA operations, each is single page
  - OS chains the pages for the requester

- ② **In cache-based systems:** (there can be two copies of data items)

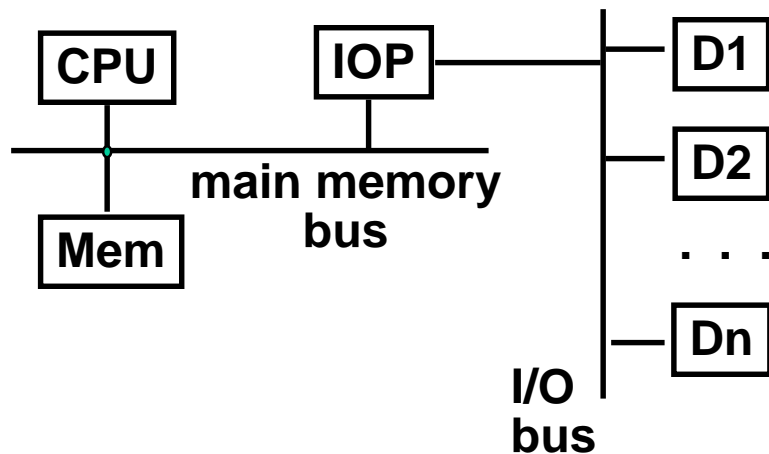
- ➔ Processor might not know that the cache and memory pages are different
- ➔ Write-back caches can overwrite I/O data or makes DMA to read wrong data

## Solutions:

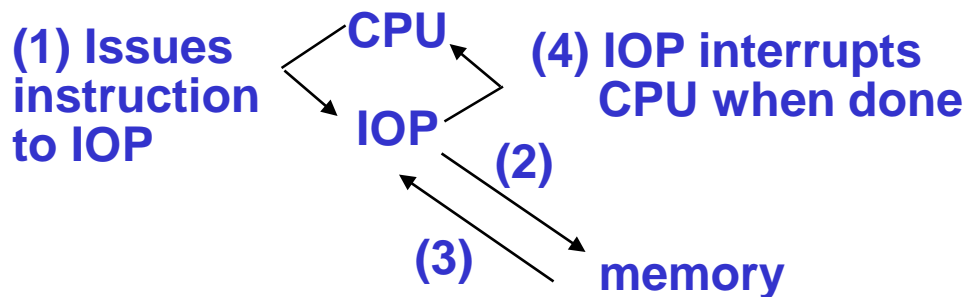
- ➔ Route I/O activities through the cache
  - Not efficient since I/O data usually is not demonstrating temporal locality
- ➔ OS selectively invalidates cache blocks before I/O read or force write-back prior to I/O write
  - Usually called cache *flushing* and requires hardware support

DMA allows another path to main memory with no cache and address translation

# I/O Processor

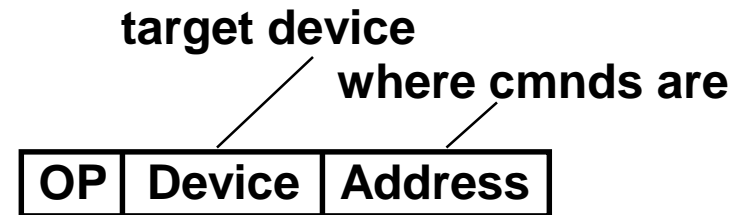


- An I/O processor (IOP) offload the CPU
- Some of the new processors, e.g. Motorola 860, include special purpose IOP for serial communication

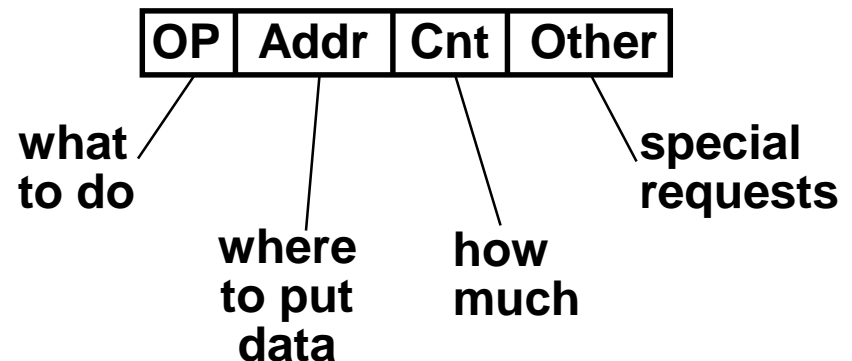


Device to/from memory transfers are controlled by the IOP directly.

IOP steals memory cycles.



IOP looks in memory for commands



# Operating System's Role

- ❑ Operating system acts as an interface between I/O hardware and programs
- ❑ Important characteristics of the I/O systems:
  - ➔ The I/O system is shared by multiple program using the processor
  - ➔ I/O systems often use interrupts to communicate information about I/O
    - Interrupts must be handled by OS because they cause a transfer to supervisor mode
  - ➔ The low-level control of an I/O device is complex:
    - Managing a set of concurrent events
    - The requirements for correct device control are very detailed
- ❑ Operating System's Responsibilities
  - ➔ Provide protection to shared I/O resources
    - Guarantees that a user's program can only access allowed set of I/O services
  - ➔ Provides 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

# Conclusion

## □ Summary

- ➔ Commanding I/O devices
  - Memory-mapped I/O
  - I/O instructions
- ➔ Communication with I/O devices
  - Device polling
  - I/O interrupts
  - Direct memory mapping
  - I/O processor
- ➔ Operating System's role
  - I/O device interfacing
  - Protection and scheduling accesses to shared devices

## □ Next Lecture

- ➔ Introduction to Multi-processor Systems

Read sections 6.6 in 4<sup>th</sup> Ed. of the textbook