香 港 中 文 大 學
The Chinese University of Hong Kong

CENG3420

Lecture 10: I/O Systems

Bei Yu

byu@cse.cuhk.edu.hk

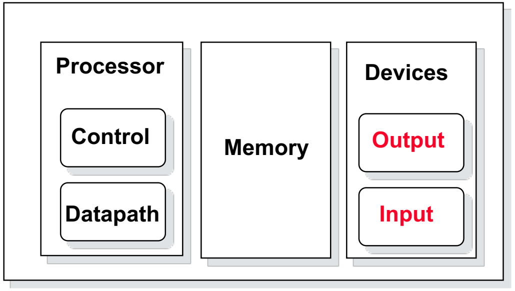
(Latest update: March 28, 2019)

Spring 2019

Overview

Overview

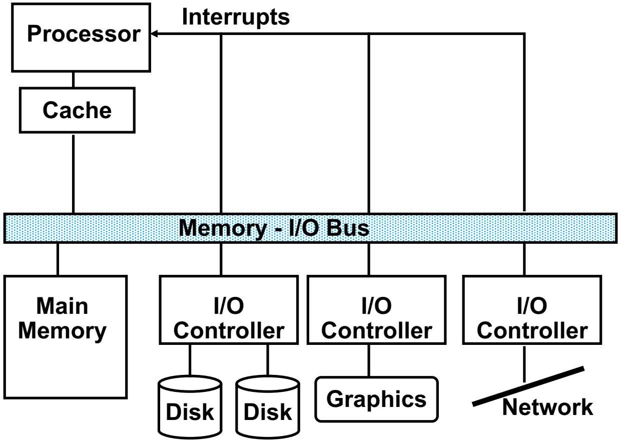
Review: Major Components of a Computer



Important metrics for an I/O system

- Performance
- Expandability
- Dependability
- Cost, size, weight
- Security

A Typical I/O System



Input and Output Devices

I/O devices are incredibly diverse with respect to

- Behavior – input, output or storage
- Partner – human or machine
- Data rate – the peak rate at which data can be transferred

Device	Behavior	Partner	Data Rate (Mb/s)
Keyboard	Input	Human	0.0001
Mouse	Input	Human	0.0038
Laser printer	Output	Human	3.2000
Flash memory	Storage	Machine	32.0000–200.0000
Magnetic disk	Storage	Machine	800.0000–3000.0000
Graphics display	Output	Human	800.0000–8000.0000
Network/LAN	Input/output	Machine	100.0000–10000.0000

I/O Performance Measures

I/O bandwidth (throughput)

- Amount of information that can be input (output) and communicated per unit time
- How much data can we move through the system in a certain time?
- How many I/O operations can we do per unit time?

I/O response time (latency)

- Total elapsed time to accomplish an input or output operation
- An especially important performance metric in real-time systems

Overview

Bus

A shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates

Advantages

- Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
- Low cost – a single set of wires is shared in multiple ways

Disadvantages

- Creates a communication bottleneck – bus bandwidth limits the maximum I/O throughput

Bus

A shared communication link (a single set of wires used to connect multiple subsystems) that needs to support a range of devices with widely varying latencies and data transfer rates

Advantages

- Versatile – new devices can be added easily and can be moved between computer systems that use the same bus standard
- Low cost – a single set of wires is shared in multiple ways

Disadvantages

- Creates a communication bottleneck – bus bandwidth limits the maximum I/O throughput

The maximum bus speed is largely limited by

- The length of the bus
- The number of devices on the bus

Types of Buses

Processor-Memory Bus (“Front Side Bus”, proprietary)

- Short and high speed
- Matched to the memory system to maximize the memory-processor bandwidth
- Optimized for cache block transfers

I/O Bus (industry standard, e.g., SCSI, USB, Firewire)

- Usually is lengthy and slower
- Needs to accommodate a wide range of I/O devices
- Use either the processor-memory bus or a backplane bus to connect to memory

Backplane Bus (industry standard, e.g., ATA, PCIexpress)

- Allow processor, memory and I/O devices to coexist on a single bus
- Used as an intermediary bus connecting I/O busses to the processor-memory bus

I/O Transactions

- An I/O transaction is a sequence of operations over the interconnect that includes a request and may include a response either of which may carry data.
- A transaction is initiated by a single request and may take many individual bus operations.
- An I/O transaction typically includes two parts
  1. Sending the address
  2. Receiving or sending the data

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
-
-
-

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
-
-
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
-
-

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
-
-
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
- ☹ Every device communicating on the bus must use same clock rate
- ☹ To avoid clock skew, they cannot be long if they are fast

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
-
-
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
- ☹ Every device communicating on the bus must use same clock rate
- ☹ To avoid clock skew, they cannot be long if they are fast

Asynchronous Bus (e.g., I/O buses)

- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
- ☺ Can accommodate a wide range of devices and device speeds
- ☺ Can be lengthened without worrying about clock skew
-
-

Synchronous and Asynchronous Buses

Synchronous Bus (e.g., processor-memory buses)

- Includes a clock in the control lines and has a fixed protocol for communication that is relative to the clock
- ☺ Involves very little logic and can run very fast
- ☹ Every device communicating on the bus must use same clock rate
- ☹ To avoid clock skew, they cannot be long if they are fast

Asynchronous Bus (e.g., I/O buses)

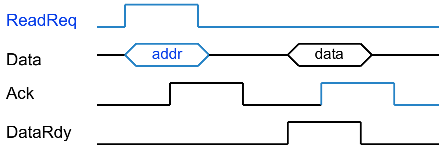
- It is not clocked, so requires a handshaking protocol and additional control lines (ReadReq, Ack, DataRdy)
- ☺ Can accommodate a wide range of devices and device speeds
- ☺ Can be lengthened without worrying about clock skew
- ☹ Slow(er)
- ☹ More complex due to handshaking protocol

Advanced Technology Attachment (ATA) Cable

- Backplane bus
- Connects hard drives, CD-ROM drives, and other drives
- [Old] Parallel ATA (PATA): synchronous
- [New] Serial ATA (SATA), much thinner, asynchronous

- **Reason:** Skew Problem

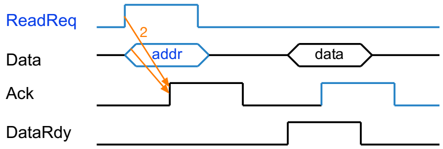
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2.
3.
4.
5.
6.
7.
8.

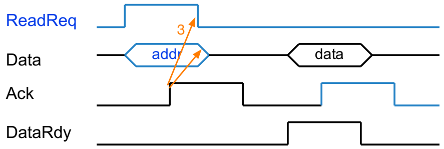
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3.
4.
5.
6.
7.
8.

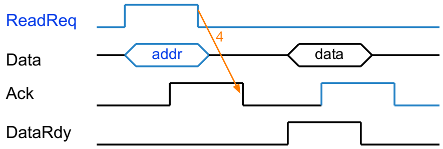
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4.
5.
6.
7.
8.

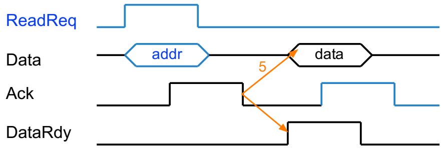
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4. Memory sees `ReadReq` go low and drops `Ack`
5.
6.
7.
8.

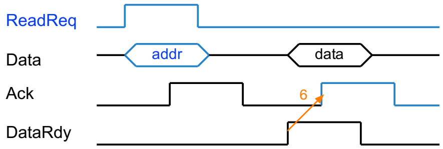
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4. Memory sees `ReadReq` go low and drops `Ack`
5. When memory ready, putting data on data lines & raises `DataRdy`
6.
7.
8.

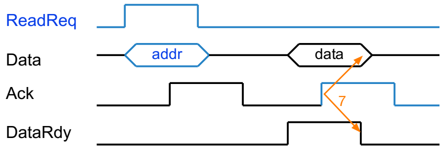
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4. Memory sees `ReadReq` go low and drops `Ack`
5. When memory ready, putting data on data lines & raises `DataRdy`
6. I/O device sees `DataRdy`, reads data from data lines & raises `Ack`
7.
8.

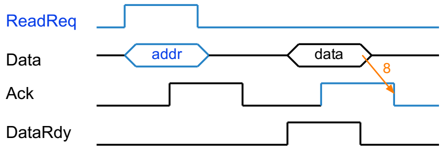
Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4. Memory sees `ReadReq` go low and drops `Ack`
5. When memory ready, putting data on data lines & raises `DataRdy`
6. I/O device sees `DataRdy`, reads data from data lines & raises `Ack`
7. Memory sees `Ack`, releases data lines, and drops `DataRdy`
8.

Asynchronous Bus Handshaking Protocol



Example: data from Memory to I/O devices

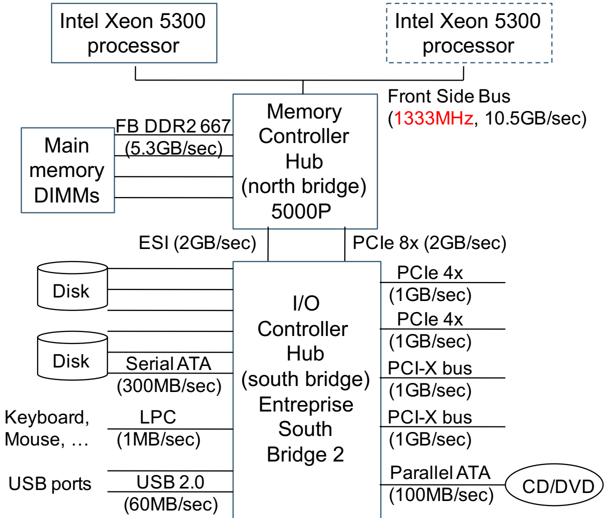
1. I/O device requests by raising `ReadReq` & putting addr on the data lines
2. Memory sees `ReadReq`, reads addr from data lines, and raises `Ack`
3. I/O device sees `Ack` and releases the `ReadReq` and data lines
4. Memory sees `ReadReq` go low and drops `Ack`
5. When memory ready, putting data on data lines & raises `DataRdy`
6. I/O device sees `DataRdy`, reads data from data lines & raises `Ack`
7. Memory sees `Ack`, releases data lines, and drops `DataRdy`
8. I/O device sees `DataRdy` go low and drops `Ack`

Key Characteristics of I/O Standards

	Firewire	**USB 2.0**	**PCIe**	**Serial ATA**	**SA SCSI**
Use	External	External	Internal	Internal	External
Devices per channel	63	127	1	1	4
Max length	4.5 meters	5 meters	0.5 meters	1 meter	8 meters
Data Width	4	2	2 per lane	4	4
Peak Bandwidth	50MB/sec (400) 100MB/sec (800)	0.2MB/sec (low) 1.5MB/sec (full) 60MB/sec (high)	250MB/sec per lane (1x) Come as 1x, 2x, 4x, 8x, 16x, 32x	300MB/sec	300MB/sec
Hot pluggable?	Yes	Yes	Depends	Yes	Yes

Hot plugging: a device does not require a restart of the system

A Typical I/O System



Overview

Interfacing I/O Devices to Processor / Memory

The operating system (OS) acts as the interface between the I/O hardware and the program requesting I/O since

- Multiple programs using the processor share the I/O system
- I/O systems usually use interrupts which are handled by the OS
- Low-level control of an I/O device is complex and detailed

OS must be able to

- give commands to the I/O devices
- be notified the status of I/O device
- transfer data between the memory and the I/O device
- protect I/O devices to which a user program doesn't have access
- schedule I/O requests to enhance system throughput

How Processor Detects I/O Devices

Port-mapped I/O (PMIO)

- special class of CPU instructions for performing I/O
- EX:

Memory-mapped I/O (MMIO)

- Portions of the high-order memory address space are assigned to each I/O device
- Read and writes to those memory addresses are interpreted as commands to the I/O devices
- Load/stores to the I/O address space can only be done by the OS
- EX:

How Processor Detects I/O Devices

Port-mapped I/O (PMIO)

- special class of CPU instructions for performing I/O
- EX: `in` and `out` instructions in x86 architecture

Memory-mapped I/O (MMIO)

- Portions of the high-order memory address space are assigned to each I/O device
- Read and writes to those memory addresses are interpreted as commands to the I/O devices
- Load/stores to the I/O address space can only be done by the OS
- EX: MIPS, LC-3b

How I/O Devices Communicate with Processor

Polling

- Processor periodically checks the status of an I/O device (through the OS) to determine its need for service
- Processor is totally in control – but does all the work
- Can waste a lot of processor time due to speed differences

Interrupt-driven I/O

- I/O device issues an interrupt to indicate that it needs attention

Interrupt Driven I/O

Asynchronous

- does NOT prevent any instruction from completing
- Need a way to identify the device generating the interrupt
- Can have different urgencies (so need a way to prioritize them)

Advantages

- Relieves the processor from having to continuously polling
- user program progress is only suspended during the actual transfer of I/O data to/from user memory space

Disadvantage

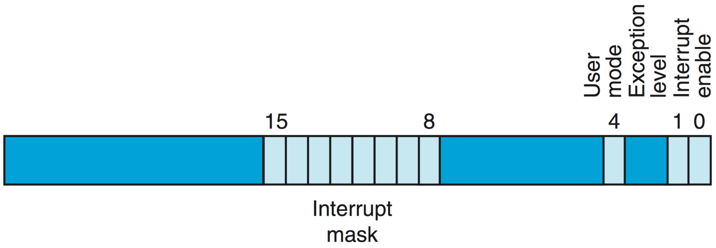
- need special hardware support

Exception Handling Registers

- MIPS uses two coprocessors: C0 and C1 for additional help.
- C0 primarily helps with exception handling
- C1 helps with floating point arithmetic.
- Each coprocessor has a few registers.

Register Number	Register Name	Usage
8	BadVAddr	Memory address where exception occurred
12	Status	Interrupt mask, enable bits, and status when exception occurred
13	Cause	Type of exception and pending interrupt bits
14	EPC	Address of instruction that caused exception

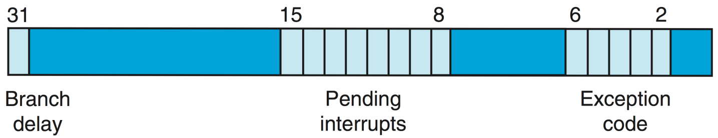
Status Register (# 12)



- `Interrupt mask` bits: whether enables 8 different exception levels
- `Exception level` bit: 1 if kernel mode; otherwise user mode.
- `Interrupt enable` bit: whether enable interrupt

Cause Register (# 13)

When an interrupt arrives, it sets its `Pending interrupt` bit in the cause register, even if the `mask` bit is disabled.

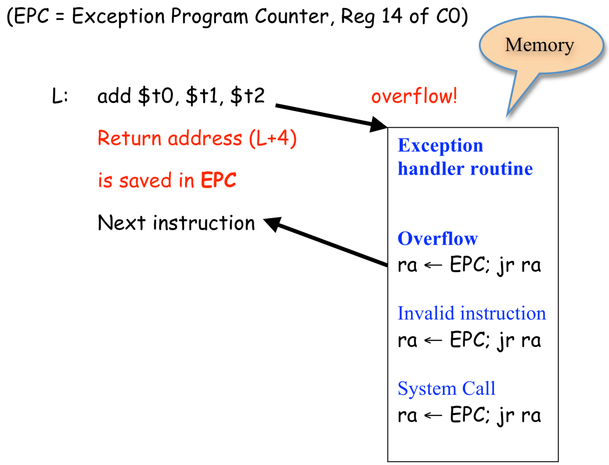


- To enable a `Pending interrupt` in cause register, the corresponding `Interrupt mask` in status register must be 1
- Once an interrupt occurs, the OS can find the reason in the `Exception code` field

  0: I/O interrupt

  1: arithmetic overflow etc

Handling Exception



The Exception Handler determines the cause of the exception by looking at the `exception code` bits. Then it jumps to the appropriate exception handling routine. Finally, it returns to the main program.

Overview

Introduction

Bus

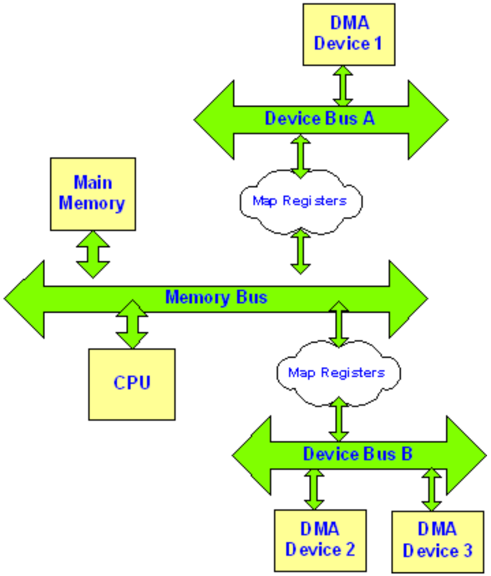
Interrupt I/O

Direct Memory Access (DMA)

Direct Memory Access (DMA)

- For high-bandwidth devices (like disks) interrupt-driven I/O would consume a lot of processor cycles
- With DMA, the DMA controller has the ability to transfer large blocks of data directly to/from the memory without involving the processor
- The processor initiates the DMA transfer by supplying the I/O device address, the operation to be performed, the memory address destination/source, the number of bytes to transfer
- The DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
- When the DMA transfer is complete, the DMA controller interrupts the processor to let it know that the transfer is complete
- There may be multiple DMA devices in one system
- Processor and DMA controllers contend for bus cycles and for memory

DMA Example


DMA Device 1
Device Bus A
Main Memory
Map Registers
Memory Bus
CPU
Map Registers
Device Bus B
DMA Device 2
DMA Device 3

DMA & Virtual Memory Considerations

Should the DMA work with virtual addresses or physical addresses?

If with Physical Address:

- Must constrain all of the DMA transfers to stay within one page because if it crosses a page boundary, then it won't necessarily be contiguous in memory
- If the transfer won't fit in a single page, it can be broken into a series of transfers (each of which fit in a page) which are handled individually and chained together

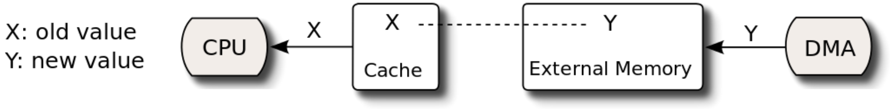
If with virtual Address:

- The DMA controller will have to translate the virtual address to a physical address (i.e., will need a TLB structure)

DMA & Virtual Memory Considerations

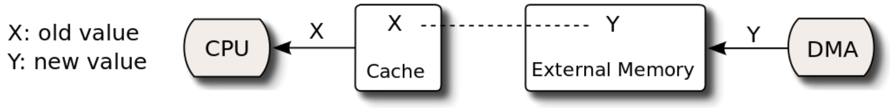
Whichever is used, the OS must cooperate by not remapping pages while a DMA transfer involving that page is in progress. Otherwise, may cause Coherency problem

Coherency Problem



- In systems with caches, there can be two copies of a data item, one in the cache and one in the main memory
- For a DMA input (from disk to memory) – the processor will be using stale data if that location is also in the cache
- For a DMA output (from memory to disk) and a write-back cache – the I/O device will receive stale data if the data is in the cache and has not yet been written back to the memory

Coherency Problem



The coherency problem can be solved by

- Routing all I/O activity through the cache – expensive and a large negative performance impact
- Having the OS invalidate all the entries in the cache for an I/O input or force write-backs for an I/O output (called a cache flush)
- Providing hardware to selectively invalidate cache entries – i.e., need a snooping cache controller