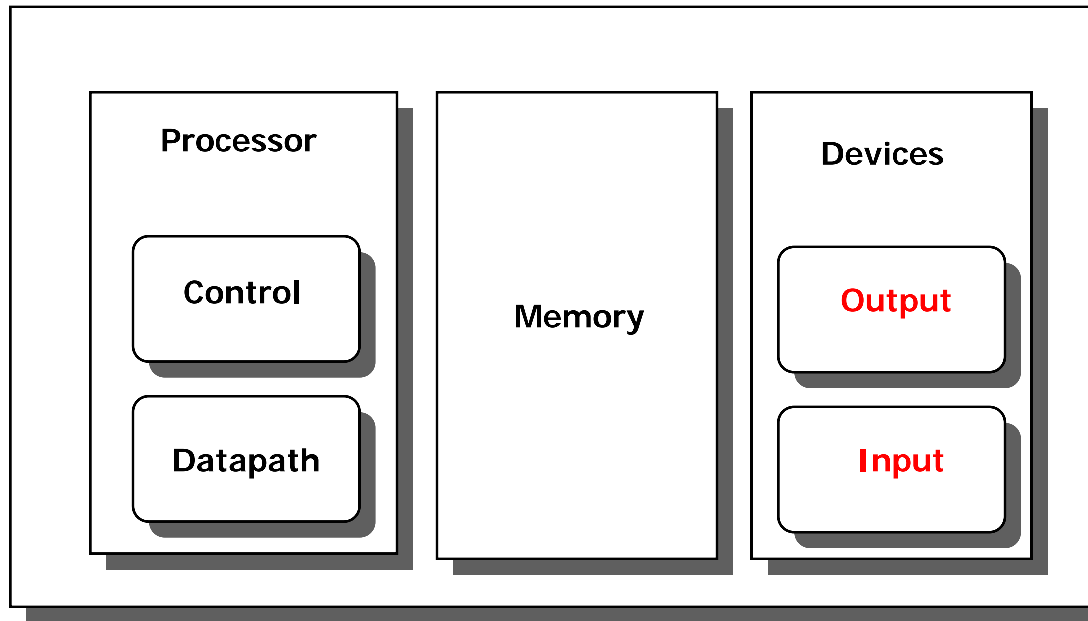
COMP303
Computer Architecture
Lecture 18

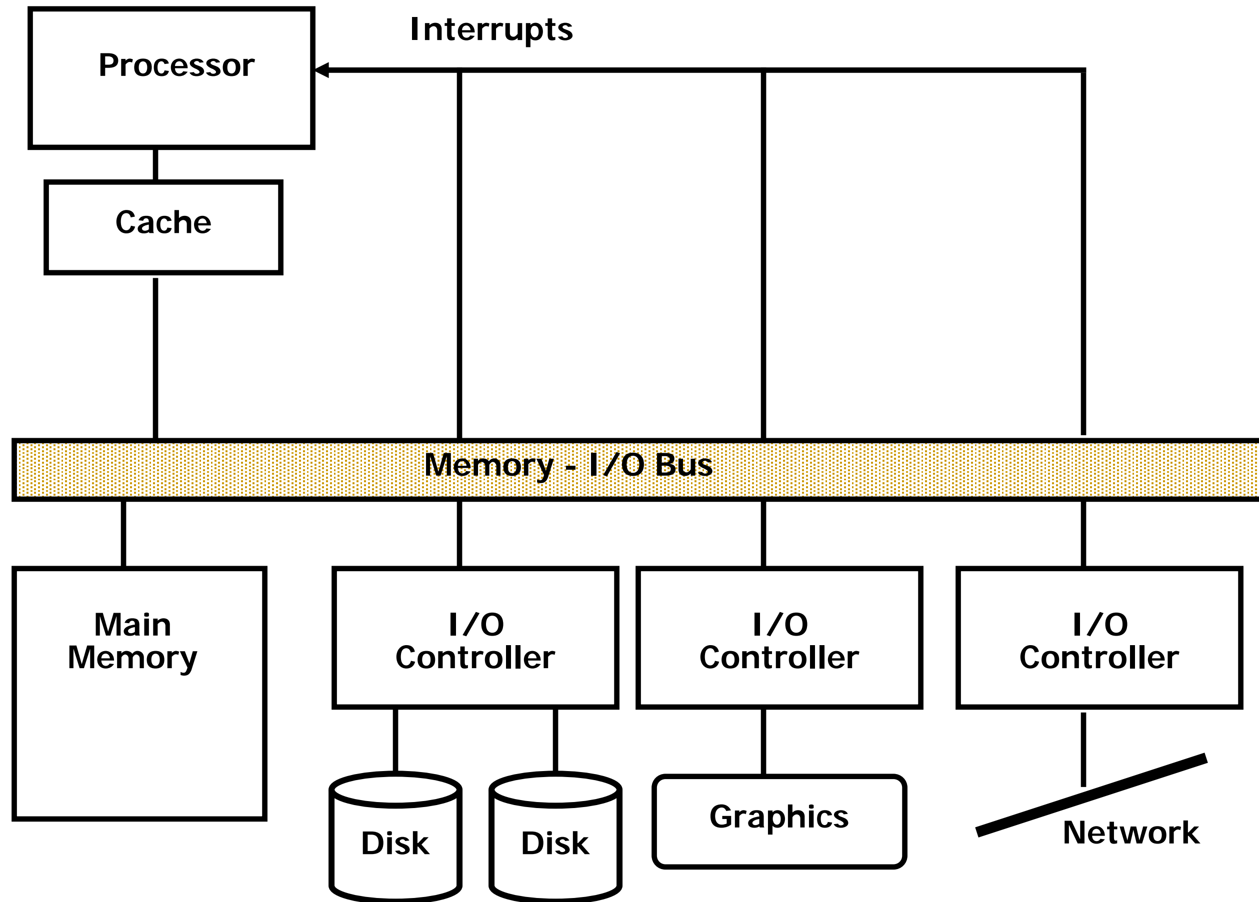
I/O System

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 between the I/O device and the main memory or processor

Device	Behavior	Partner	Data rate (Mb/s)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Laser printer	output	human	3.2000
Magnetic disk	storage	machine	800.0000-3000.0000
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-10000.0000

I/O Performance Measures

- I/O bandwidth (throughput) – amount of information that can be input (output) and communicated across an interconnect (e.g., a bus) to the processor/memory (I/O device) per unit time
  1. How much data can we move through the system in a certain time?
  2. How many I/O operations can we do per unit time?

- I/O response time (latency) – the total elapsed time to accomplish an input or output operation
  - An especially important performance metric in real-time systems

- Many applications require *both* high throughput and short response times

I/O System Interconnect Issues

- A bus is 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
- Bus transactions are defined by what they do to memory
  - output — A read transaction reads data from memory (to either the processor or an I/O device)
  - input — A write transaction writes data to the memory (from either the processor or an I/O device)

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
  - Advantage: involves very little logic and can run very fast
  - Disadvantages:
    - 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)
  - Advantages:
    - Can accommodate a wide range of devices and device speeds
    - Can be lengthened without worrying about clock skew or synchronization problems
  - Disadvantage: slow(er)

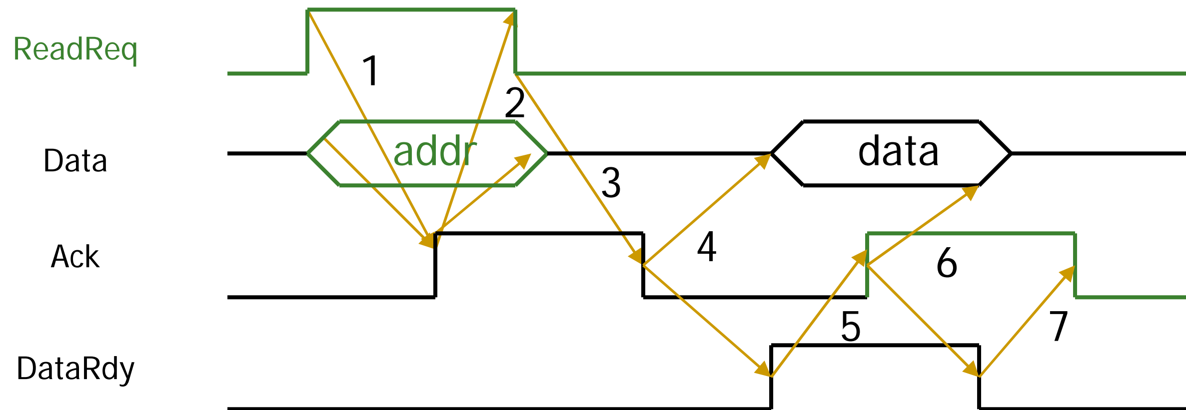
ATA Cable Sizes

- Companies have transitioned from synchronous, parallel wide buses to asynchronous narrow buses
  - Reflection on wires and clock skew makes it difficult to use 16 to 64 parallel wires running at a high clock rate (e.g., ~400MHz) so companies have moved to buses with a few one-way wires running at a very high “clock” rate (~2GHz)

  - Serial ATA cables (red) are much thinner than parallel ATA cables (green)

Asynchronous Bus Handshaking Protocol

- Output (read) data from memory to an I/O device



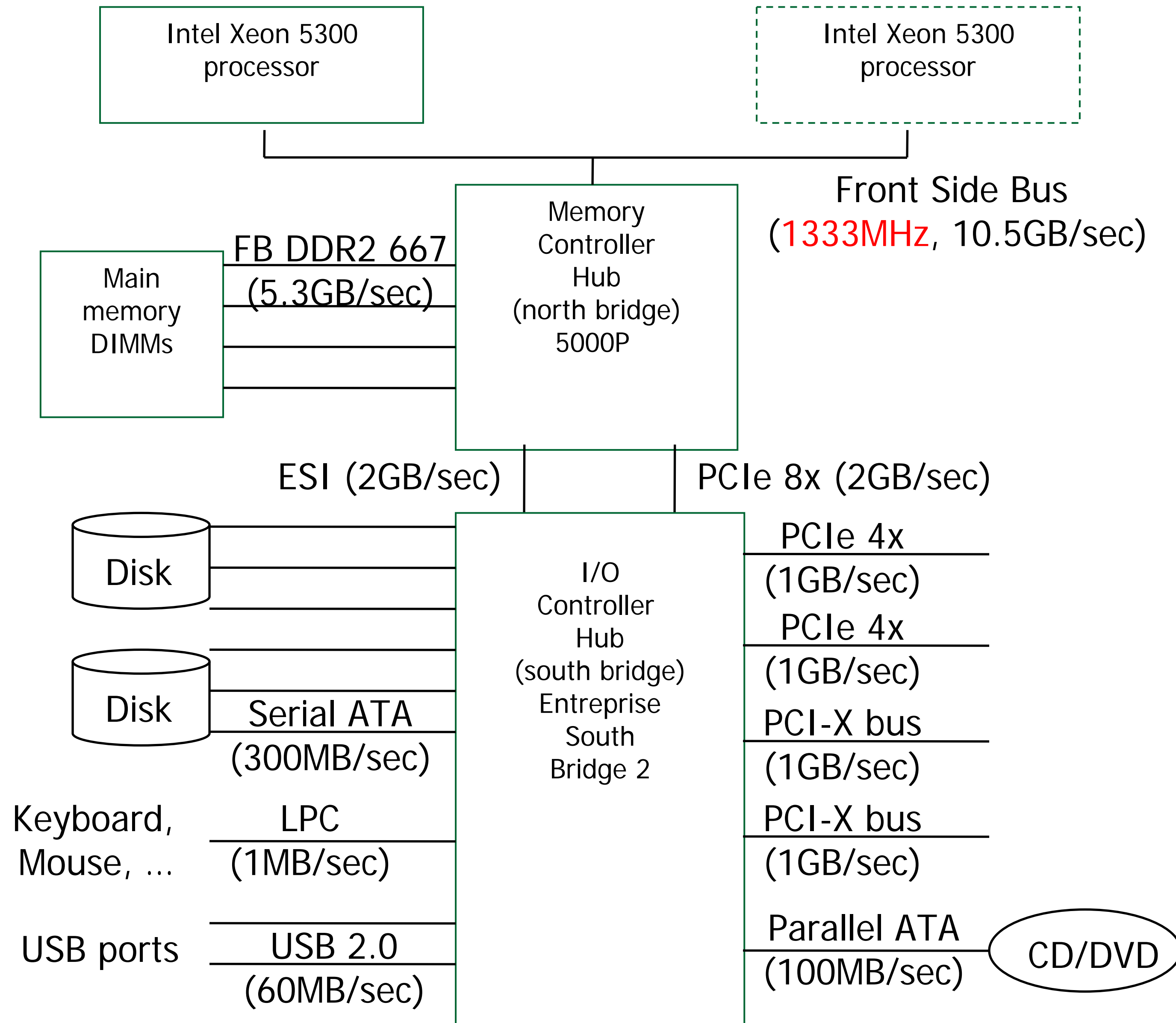
I/O device signals a request by raising ReadReq and putting the addr on the data lines

1. Memory sees ReadReq, reads addr from data lines, and raises Ack
2. I/O device sees Ack and releases the ReadReq and data lines
3. Memory sees ReadReq go low and drops Ack
4. When memory has data ready, it places it on data lines and raises DataRdy
5. I/O device sees DataRdy, reads the data from data lines, and raises Ack
6. Memory sees Ack, releases the data lines, and drops DataRdy
7. 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

A Typical I/O System



Interfacing I/O Devices to the Processor, Memory, and OS

- The operating system 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
- Thus OS must handle interrupts generated by I/O devices and supply routines for low-level I/O device operations, provide equitable access to the shared I/O resources, protect those I/O devices/activities to which a user program doesn't have access, and schedule I/O requests to enhance system throughput
  - OS must be able to give commands to the I/O devices
  - I/O device must be able to notify the OS about its status
  - Must be able to transfer data between the memory and the I/O device

Communication of I/O Devices and Processor

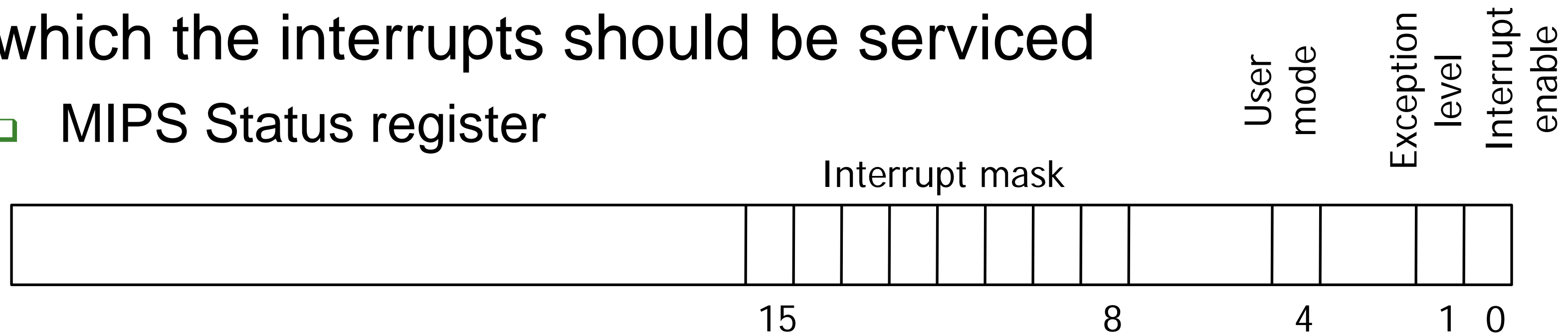
- How the processor directs the I/O devices
  - Special I/O instructions
    - Must specify both the device and the command
  - Memory-mapped I/O
    - Portions of the high-order memory address space are assigned to each I/O device
    - Read and writes to those memory addresses are interpreted as commands to the I/O devices
    - Load/stores to the I/O address space can *only* be done by the OS
- How I/O devices communicate with the processor
  - Polling – the 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 – the I/O device issues an interrupt to indicate that it needs attention

Interrupt Driven I/O

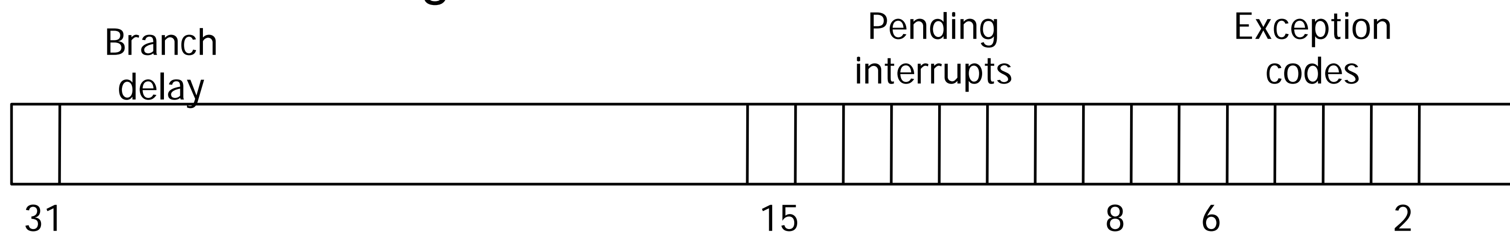
- An I/O interrupt is asynchronous wrt instruction execution
  - Is not associated with any instruction so doesn't prevent any instruction from completing
    - You can pick your own convenient point to handle the interrupt
- With I/O interrupts
  - Need a way to identify the device generating the interrupt
  - Can have different urgencies (so need a way to prioritize them)
- Advantages of using interrupts
  - Relieves the processor from having to continuously poll for an I/O event; user program progress is only suspended during the actual transfer of I/O data to/from user memory space
- Disadvantage – special hardware is needed to
  - Indicate the I/O device causing the interrupt and to save the necessary information prior to servicing the interrupt and to resume normal processing after servicing the interrupt

Interrupt Priority Levels

- Priority levels can be used to direct the OS the order in which the interrupts should be serviced
  - MIPS Status register



    - Determines who can interrupt the processor (if Interrupt enable is 0, none can interrupt)
  - MIPS Cause register



    - To enable a Pending interrupt, the corresponding bit in the Interrupt mask must be 1
    - Once an interrupt occurs, the OS can find the reason in the Exception codes field

Interrupt Handling Steps

1. Logically AND the Pending interrupt field and the Interrupt mask field to see which enabled interrupts could be the culprit. Make copies of both Status and Cause registers.
2. Select the higher priority of these interrupts (leftmost is highest)
3. Save the Interrupt mask field
4. Change the Interrupt mask field to disable all interrupts of equal or lower priority
5. Save the processor state prior to “handling” the interrupt
6. Set the Interrupt enable bit (to allow higher-priority interrupts)
7. Call the appropriate interrupt handler routine
8. Before returning from interrupt, set the Interrupt enable bit back to 0 and restore the Interrupt mask field

- Interrupt priority levels (IPLs) assigned by the OS to each process can be raised and lowered via changes to the Status’s Interrupt mask field

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
    1. 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
    2. The DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
    3. 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

The DMA Stale Data (Coherence) 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
- The coherency problem can be solved by
  1. Routing all I/O activity through the cache – expensive and a large negative performance impact
  2. 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)
  3. Providing hardware to *selectively* invalidate cache entries – i.e., need a snooping cache controller

DMA and Virtual Memory Considerations

- Should the DMA work with virtual addresses or physical addresses?
- If working with physical addresses
  - 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 working with virtual addresses
  - The DMA controller will have to translate the virtual address to a physical address (i.e., will need a TLB structure)
- Whichever is used, the OS must cooperate by not remapping pages while a DMA transfer involving that page is in progress