Memory-Mapped Input and Output

Dr. Igor Ivkovic

iivkovic@uwaterloo.ca

[with material from “Computer Organization and Design” by Patterson and Hennessy, and “Digital Design and Computer Architecture” by Harris and Harris, both published by Morgan Kaufmann]

Objectives

- Additional notes on memory management
- Using memory-mapped input and output
- Interconnecting computer components

Practical Virtual Memory Management /1

- **In the examples given before, the size of virtual address space was taken to be 2 GB**
  - In practice, for instance for 32-bit Windows processes, the amount of available virtual memory is 4 GB (or 2^{32} bytes)
  - However, that space is typically divided as 2 GB for user processes, and 2 GB for protected OS processes
  - For 64-bit operating system, the theoretical amount of available virtual memory is 2^{64}, which equals about 16 exabytes
  - In practice, for instance for 64-bit Windows processes, the amount of used virtual memory is much smaller
  - User processes are allocated 8 TB of space, while the OS processes can use up to 248 TB of space

Practical Virtual Memory Management /2

- **User space vs. system space pages:**
  - In Windows, all virtual memory pages in the user space can be paged out (moved) from physical memory to hard disk
  - However, not all virtual memory pages in the system space can be paged out from physical memory to hard disk
  - **Paged (System) Memory:** Represents the pages that can be paged out from physical memory to hard disk
  - **Nonpaged (System) Memory:** Represents the pages that cannot be paged out from physical memory to hard disk

Memory-Mapped Input and Output /1

- **Accessing Input and Output (I/O) Devices:**
  - The processor accesses I/O devices – such as keyboards, monitors, and printers – just like it accesses memory
  - **Each I/O device is assigned one or more addresses**
  - When that address is detected, data is read from or written to the I/O device instead of memory
  - **A portion of the address space is dedicated to I/O devices**
    - Example: addresses in the range 0xFFFF0000 to 0xFFFFFFFF

Memory-Mapped Input and Output /2

- **I/O is mediated by the OS:**
  - Multiple programs share I/O resources
    - Hence, the OS needs to provide protection and scheduling
  - I/O causes asynchronous interrupts
    - Same mechanism as exceptions
  - I/O interface needs to be unified
    - OS provides abstractions to programs
- **Device Driver:**
  - Software that is provided to communicate with an I/O device
  - Other programs can call on the code within the device driver to utilize the I/O hardware without knowing its internal details

Memory-Mapped Input and Output /3

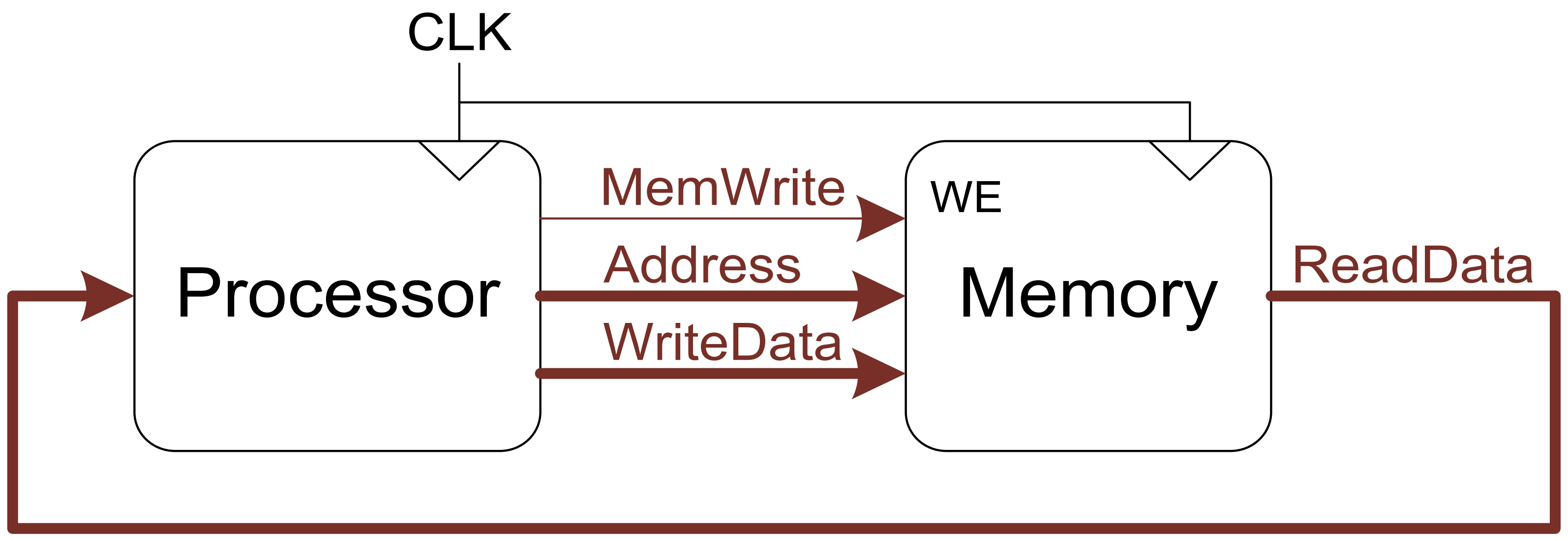
- **I/O devices are managed by I/O controller hardware:**
  - Transfers data to/from device
  - Synchronizes operations with software
- **Command registers:**
  - Cause device to do something
- **Status registers:**
  - Indicate what the device is doing and occurrence of errors
- **Data registers:**
  - **Write:** Transfer data to a device
  - **Read:** Transfer data from a device

Memory-Mapped Input and Output /4

- **I/O Register Mapping:**
  - Registers are addressed in the same space as memory
  - Address decoder distinguishes between the registers
  - OS uses address translation mechanism to make them only accessible to kernel
- **Address Decoder:**
  - Takes an address as input and determines which device/memory communicates with the processor
- **ReadData Multiplexer:**
  - A MUX that selects between memory and I/O devices as source of data sent to the processor

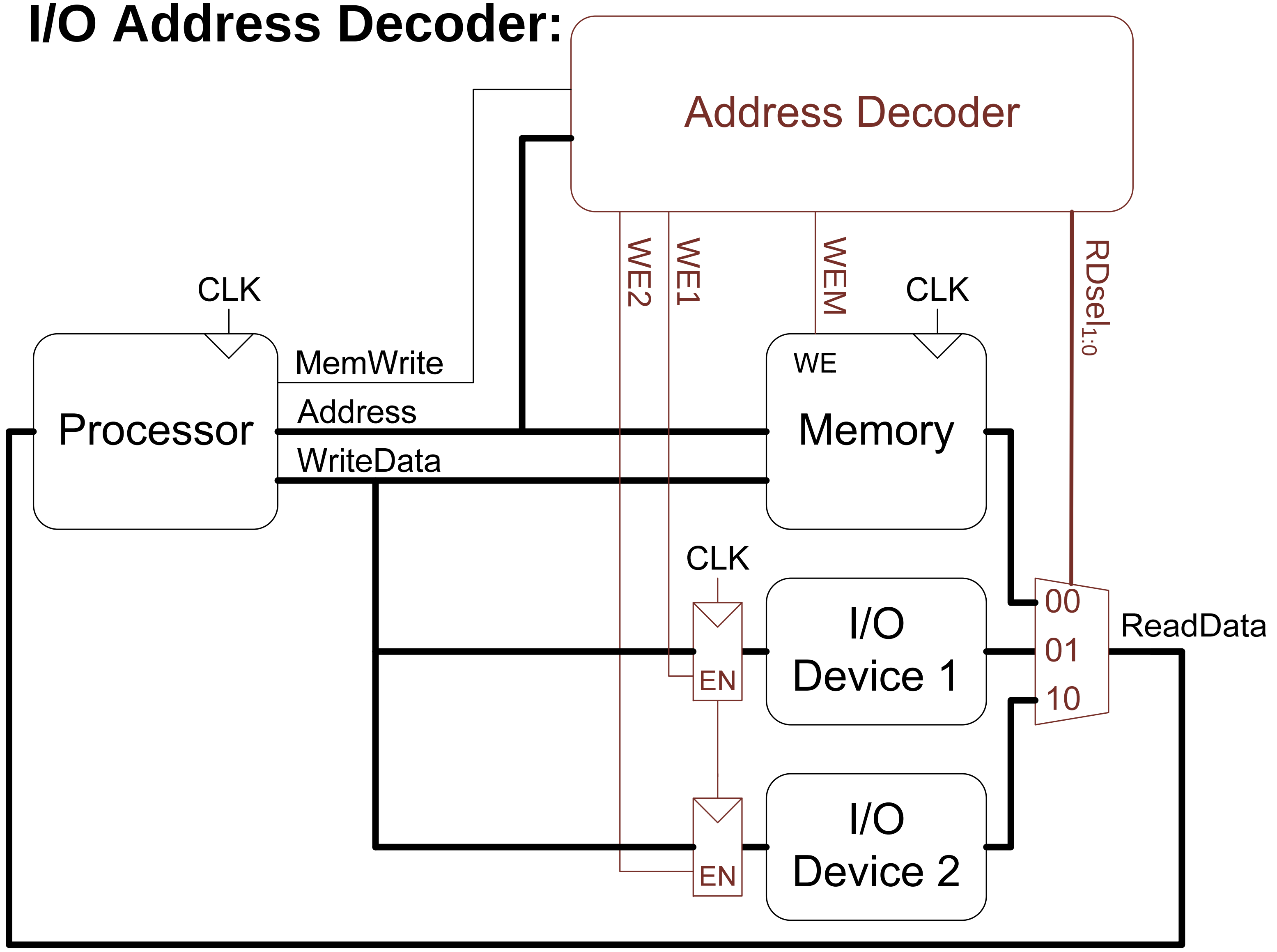
Memory-Mapped Input and Output /5

- **Review – CPU-to-Memory Interface:**
  - What we have covered so far includes an interface between the processor and memory
  - If the MemWrite is asserted as 1, Memory writes WriteData to the address specified in the Address input
  - If the MemWrite is 0, Memory reads ReadData from the address specified in the Address Input



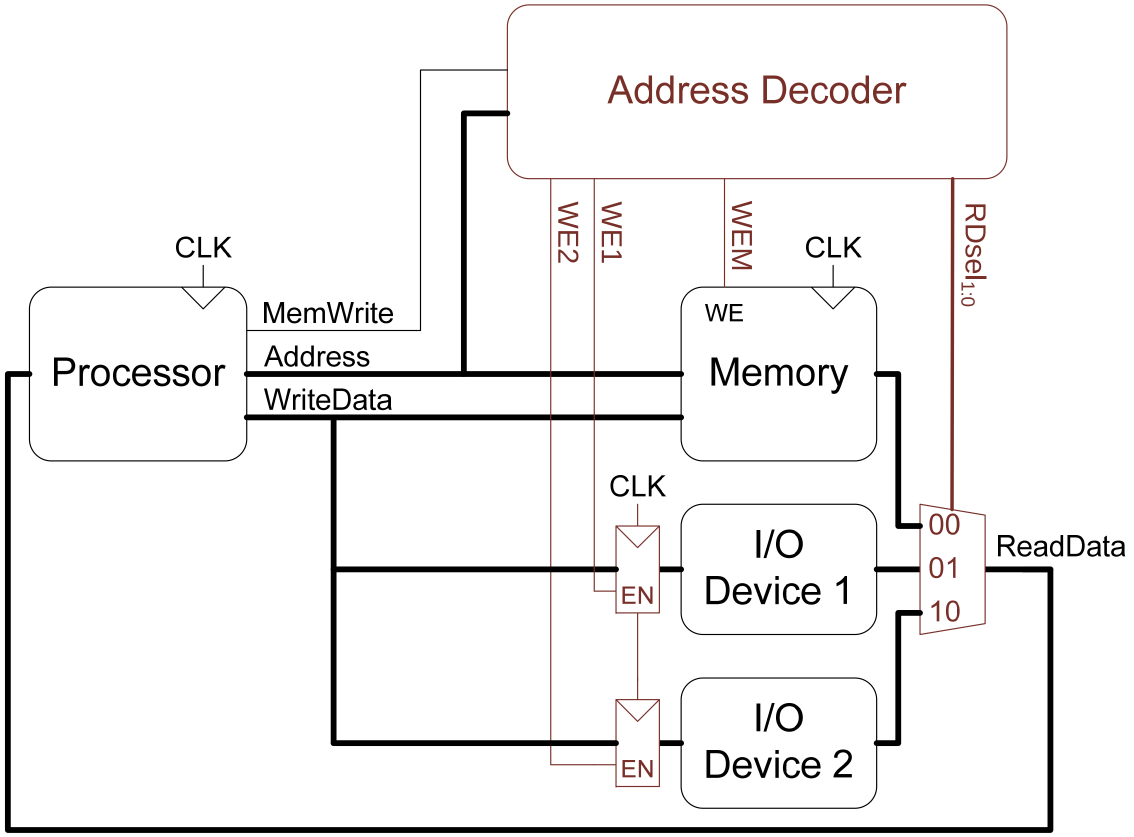
Memory-Mapped Input and Output /6

- **I/O Address Decoder:**



Memory-Mapped Input and Output /7

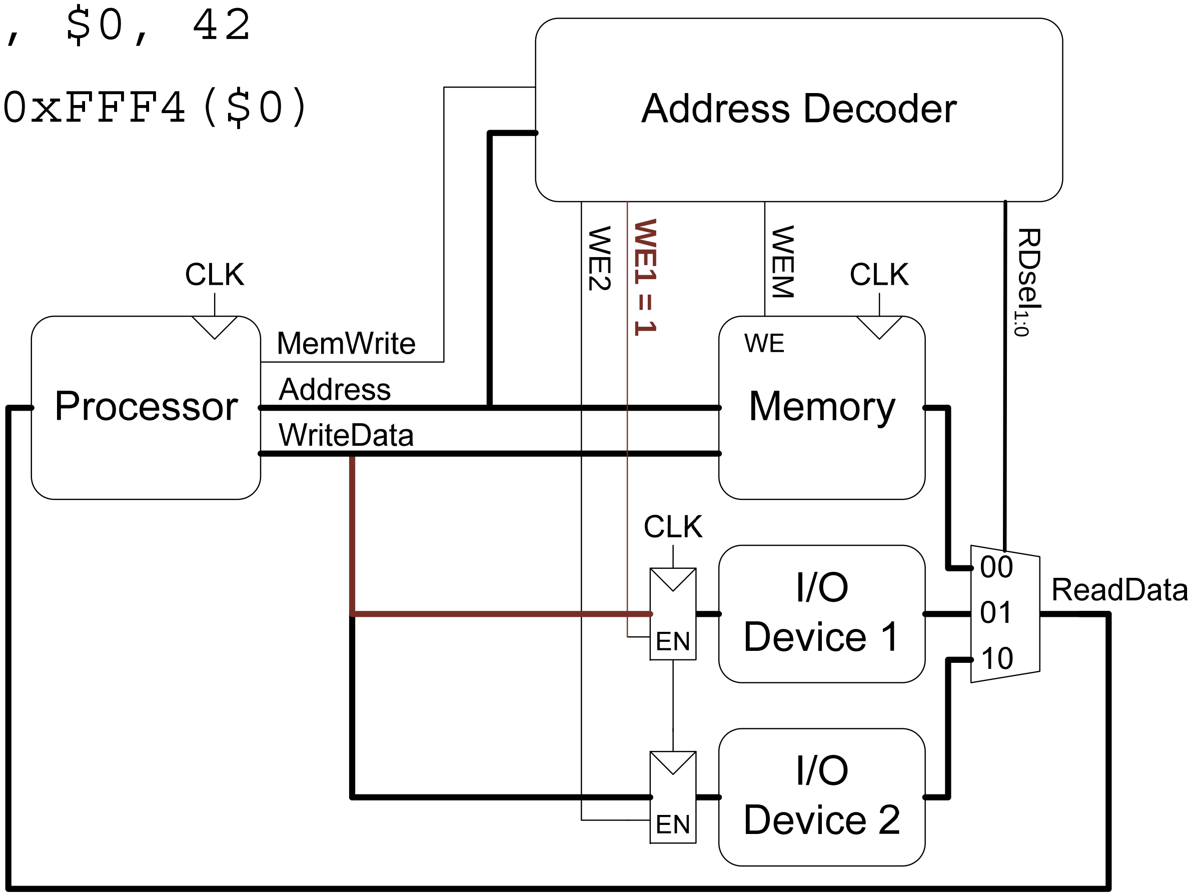
- **Address Decoder:**
  - WE1 and WE2 are Write-Enable signals which enable writes to I/O Device 1 and I/O Device 2, respectively
  - $RDsel_{1:0}$ enables read from a specific device or memory



Memory-Mapped Input and Output /8

- **Example:**
  - Suppose I/O Device 1 is assigned the address 0xFFFFFFF4
  - Write the value 42 to I/O Device 1

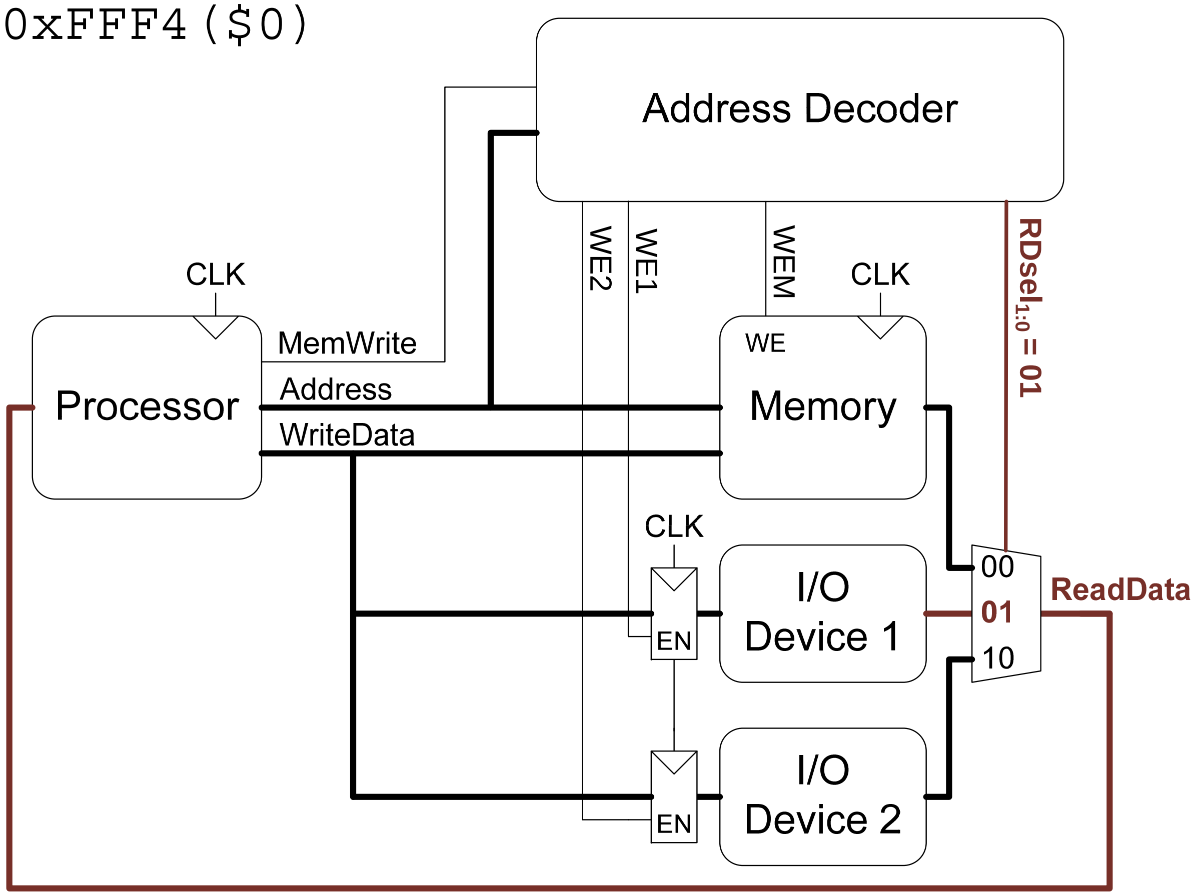
```
addi $t0, $0, 42
sw $t0, 0xFFF4($0)
```



Memory-Mapped Input and Output /9

- **Example:**
  - Suppose I/O Device 1 is assigned the address 0xFFFFFFF4
  - Read the value from I/O Device 1 and place in $t3

```
lw $t3, 0xFFF4($0)
```



Controlling Memory-Mapped I/O /1

- **I/O Polling:**
  - Periodically check the I/O status register
  - If the device is ready, do the requested operation
  - If there is an error, take an appropriate action
  - Polling is common in small or low-performance real-time embedded systems, which exhibit
    - Predictable timing, and
    - Low hardware cost
  - In other systems, it can waste CPU time

Controlling Memory-Mapped I/O /2

- **Interrupt-Driven I/O:**
  - When a device is ready or error occurs, I/O controller interrupts the CPU
  - Interrupt is like an exception (recall Cause register)
    - But interrupts are not synchronized to the instruction execution
    - Can invoke interrupt handler between instructions
    - The cause information often identifies the interrupting device
  - **Priority interrupts:**
    - Devices needing more urgent attention get higher priority
    - Can interrupt handler for a lower priority interrupt

Controlling Memory-Mapped I/O /3

- **In polling I/O and interrupt-driven I/O:**
  - CPU transfers data between memory and I/O data registers
  - **This can be time consuming for high-speed devices**
- **Direct Memory Access (DMA):**
  - Instead, OS provides a starting address in memory
  - I/O controller transfers to/from memory autonomously
  - Controller interrupts the CPU on completion or on error

Controlling Memory-Mapped I/O /4

- **DMA/Cache Interaction:**
  - If DMA writes to a main memory block that is cached, the cached copy becomes stale
  - If write-back cache has dirty block, and DMA reads the corresponding main memory block, DMA reads stale data
- **Hence, we need to ensure cache coherence**
  - Flush blocks from cache if they will be used for DMA
  - Or use non-cacheable memory locations for I/O

Controlling Memory-Mapped I/O /5

- **DMA/VM Interaction:**
  - OS uses virtual addresses for memory, but DMA blocks may not be contiguous in physical memory
  - Should the DMA use virtual addresses?
    - This would require I/O controller to do the VM translation
- **Instead, if DMA uses physical addresses**
  - Need to break transfers into page-sized data fragments
  - Or allocate contiguous physical pages for DMA

Controlling Memory-Mapped I/O /6

- **Measuring I/O Performance:**
  - I/O performance depends on
    - **Hardware:** CPU, memory, controllers, buses
    - **Software:** Operating system, application software
    - **Workload:** Request rates and patterns
  - **I/O system design can trade response time for throughput**
    - Measurements of throughput often done with constrained response-time

Interconnecting Computer Components /1

- **CPU hardware requires interconnections between**
  - CPU, memory, I/O controllers
- **(System) Bus:**
  - A shared communication channel
  - Parallel set of wires for data and synchronization of data transfer
  - Can become a performance bottleneck
  - Performance limited by physical factors, such as wire length, and number of connections

Interconnecting Computer Components /2

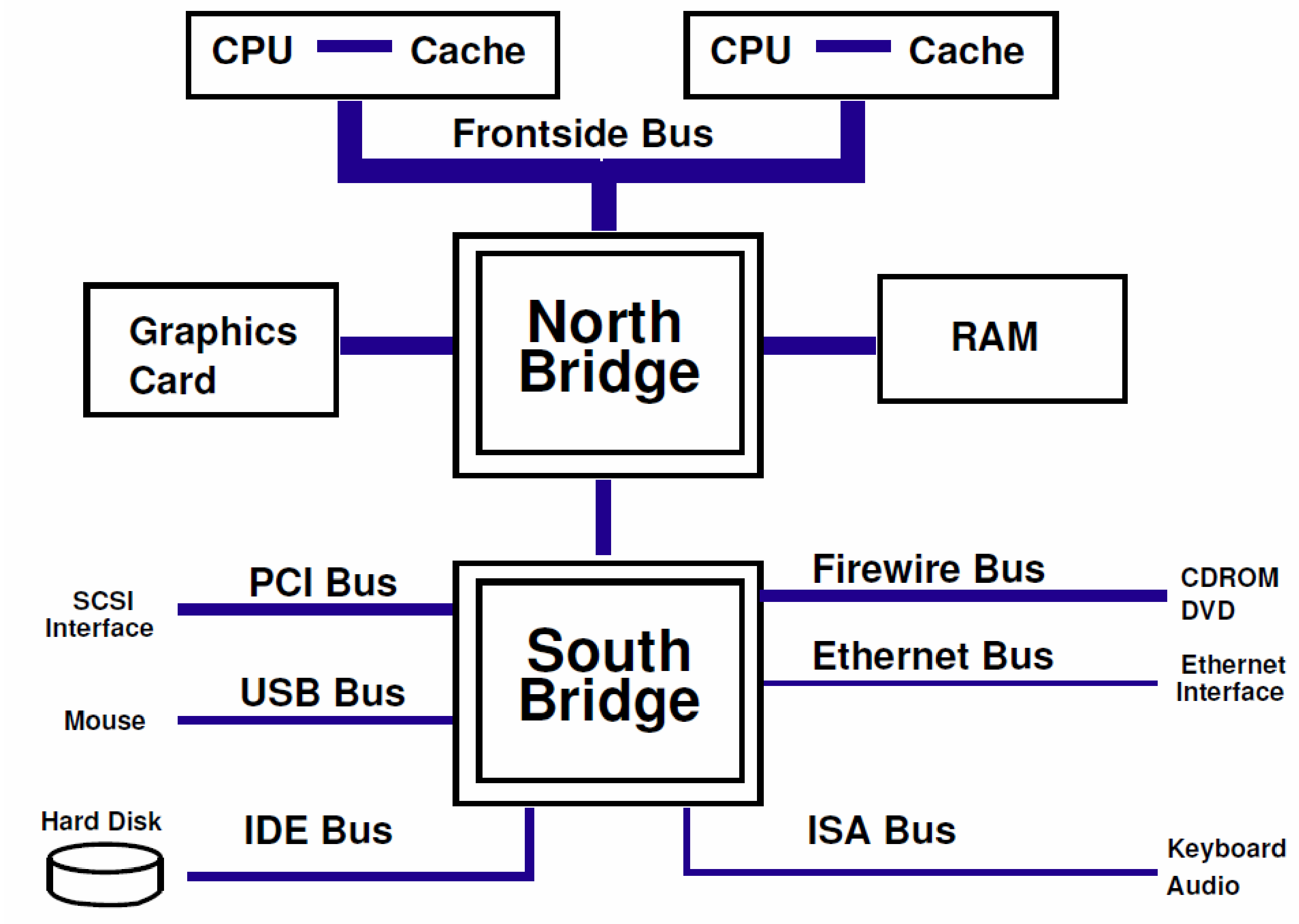
- **Data lines:**
  - Carry address and data
  - Multiplexed or separate
- **Control lines:**
  - Indicate data type, synchronize transactions
- **Synchronous line:**
  - Uses a bus clock
- **Asynchronous line:**
  - Uses request/acknowledge control lines for handshaking

Interconnecting Computer Components /3

- **Processor-Memory buses:**
  - Short in length and high speed in performance
  - Design of the memory bus is setup in a manner that matches the memory organization (e.g., presence of L1/L2/L3 caches)
- **I/O buses:**
  - Longer in length and allow multiple connections
  - Specified by standards for interoperability
  - **Connect to processor-memory bus through a bridge**
  - **North Bridge (Memory Controller):** In charge of controlling the transfers between the processor and main memory (RAM)
  - **South Bridge (I/O Controller or I/O Controller Hub (ICH)):** Controls the transfers between peripheral devices

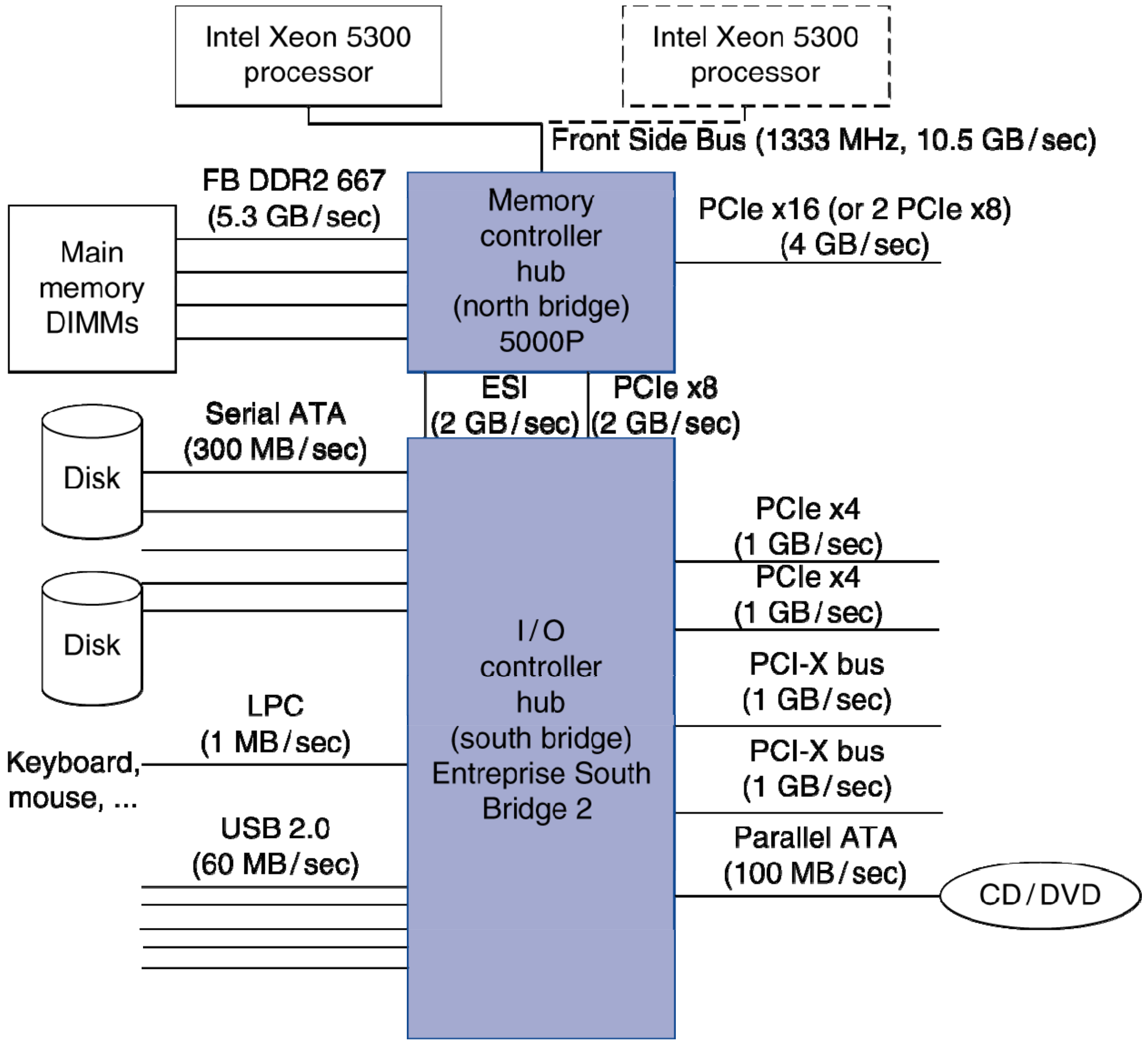
Interconnecting Computer Components /4

- **Example – Pentium Bus**



Interconnecting Computer Components /5

- **Example – Typical x86 PC I/O System:**



Intel Xeon 5300 processor
Intel Xeon 5300 processor
Front Side Bus (1333 MHz, 10.5 GB/sec)
FB DDR2 667 (5.3 GB/sec)
Main memory DIMMs
Memory controller hub (north bridge) 5000P
PCIe x16 (or 2 PCIe x8) (4 GB/sec)
ESI (2 GB/sec)
PCIe x8 (2 GB/sec)
Serial ATA (300 MB/sec)
Disk
Disk
LPC (1 MB/sec)
Keyboard, mouse, ...
USB 2.0 (60 MB/sec)
I/O controller hub (south bridge) Entreprise South Bridge 2
PCIe x4 (1 GB/sec)
PCIe x4 (1 GB/sec)
PCI-X bus (1 GB/sec)
PCI-X bus (1 GB/sec)
Parallel ATA (100 MB/sec)
CD/DVD

Interconnecting Computer Components /6

- **I/O Bus Examples:**

	Firewire	USB 2.0	PCI Express	Serial ATA	Serial Attached SCSI
Intended use	External	External	Internal	Internal	External
Devices per channel	63	127	1	1	4
Peak bandwidth	50MB/s or 100MB/s	0.2MB/s, 1.5MB/s, or 60MB/s	250MB/s/lane 1×, 2×, 4×, 8×, 16×, 32×	300MB/s	300MB/s
Hot pluggable	Yes	Yes	Depends	Yes	Yes
Max length	4.5m	5m	0.5m	1m	8m
Standard	IEEE 1394	USB Implementers Forum	PCI-SIG	SATA-IO	INCITS TC T10

Food for Thought

- **Download and Read Assignment #5 Specifications**
- **Read:**
  - Chapter 8 from the Course Notes
    - Review the material discussed in the lecture notes in more detail
    - Our course schedule follows the material in the Course Notes

  - Recommended: Chapter 8 from the Harris and Harris textbook

OR

  - Recommended: Chapter 6 from the course textbook