
I/O Systems

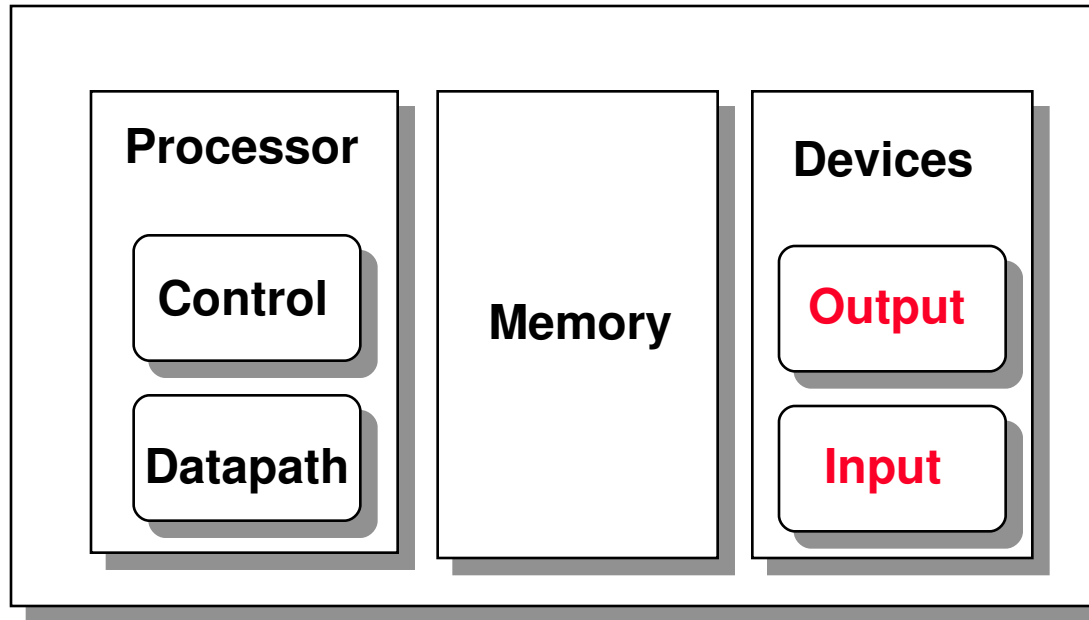
Slides Courtesy of Mary Jane Irwin (www.cse.psu.edu/~mji)

www.cse.psu.edu/~cg431

With Modifications by Nathan Sprague 2007

[Adapted from *Computer Organization and Design*, Patterson & Hennessy, ©
2005]

Review: Major Components of a Computer



- ❑ Important metrics for an I/O system
 - Performance
 - Expandability
 - Dependability
 - Cost, size, weight

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
Graphics display	output	human	800.0000-8000.0000
Network/LAN	input or output	machine	100.0000-1000.0000
Magnetic disk	storage	machine	240.0000-2560.0000

8 orders of magnitude
range

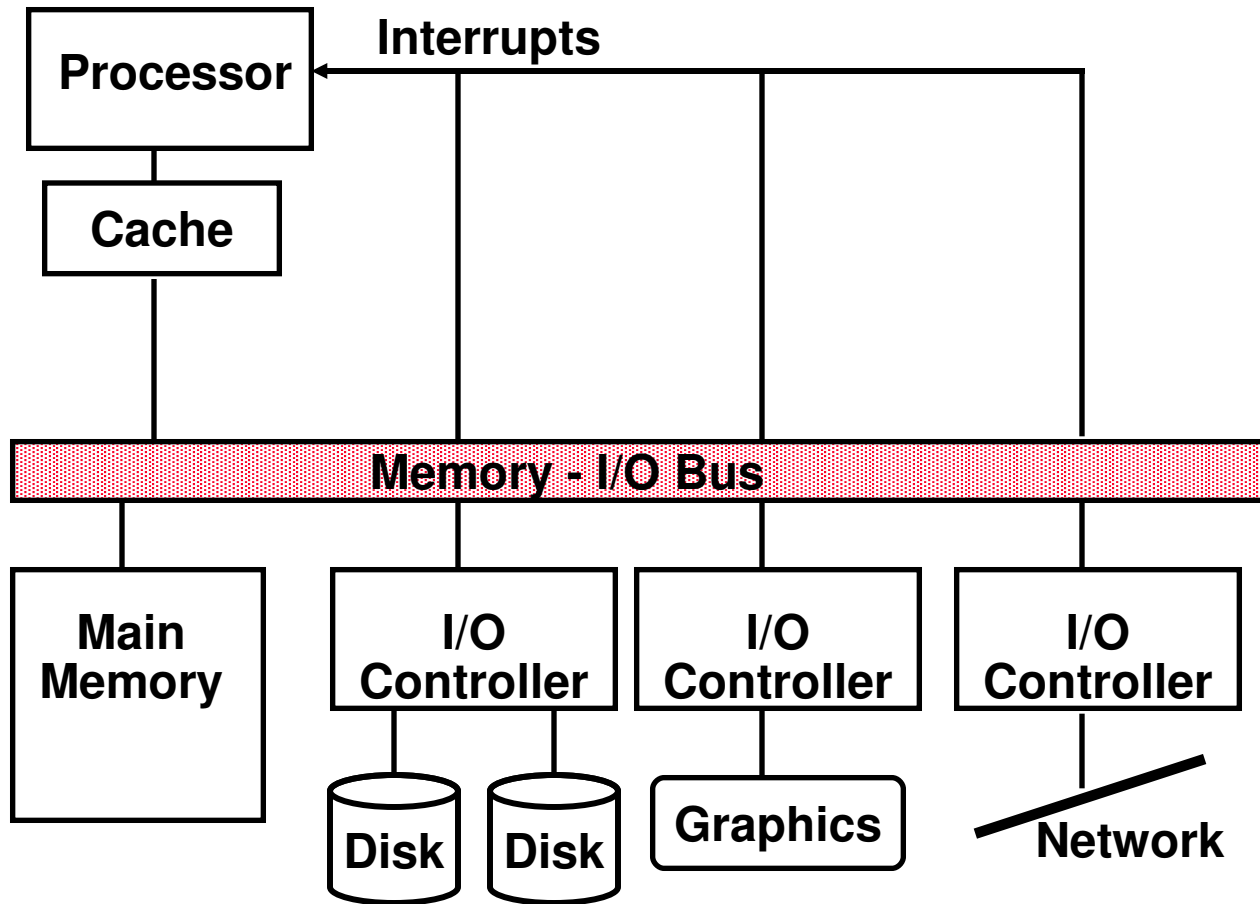
I/O Performance Measures

- ❑ **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

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

A Typical I/O System



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**

Bus Characteristics



□ Control lines

- Signal requests and acknowledgments
- Indicate what type of information is on the data lines

□ Data lines

- Data, addresses, and complex commands

□ Bus transaction consists of

- Master issuing the command (and address) – request
- Slave receiving (or sending) the data – action
- Defined by what the transaction does to memory
 - Input – inputs data from the I/O device to the memory
 - Output – outputs data from the memory to the I/O device

Types of Buses

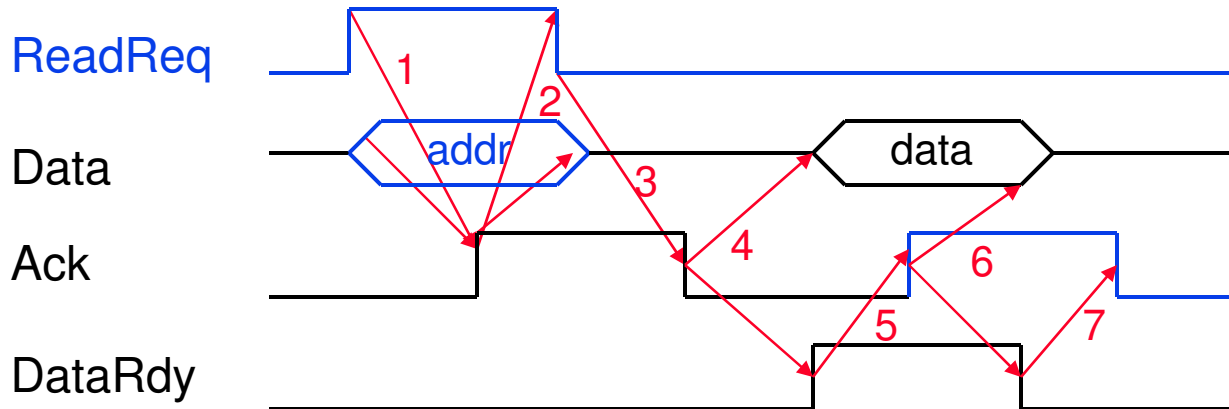
- ❑ Processor-memory bus (proprietary)
 - Short and high speed
 - Matched to the memory system to maximize the memory-processor bandwidth
 - Optimized for cache block transfers
- ❑ I/O bus (industry standard, e.g., USB, Firewire)
 - Usually is lengthy and slower
 - Needs to accommodate a wide range of I/O devices
 - Connects to the processor-memory bus or backplane bus
- ❑ Backplane bus (industry standard, e.g., ATA, PCIeexpress)
 - The backplane is an interconnection structure within the chassis
 - Used as an intermediary bus connecting I/O busses to the processor-memory bus

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)

Asynchronous Bus Handshaking Protocol

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



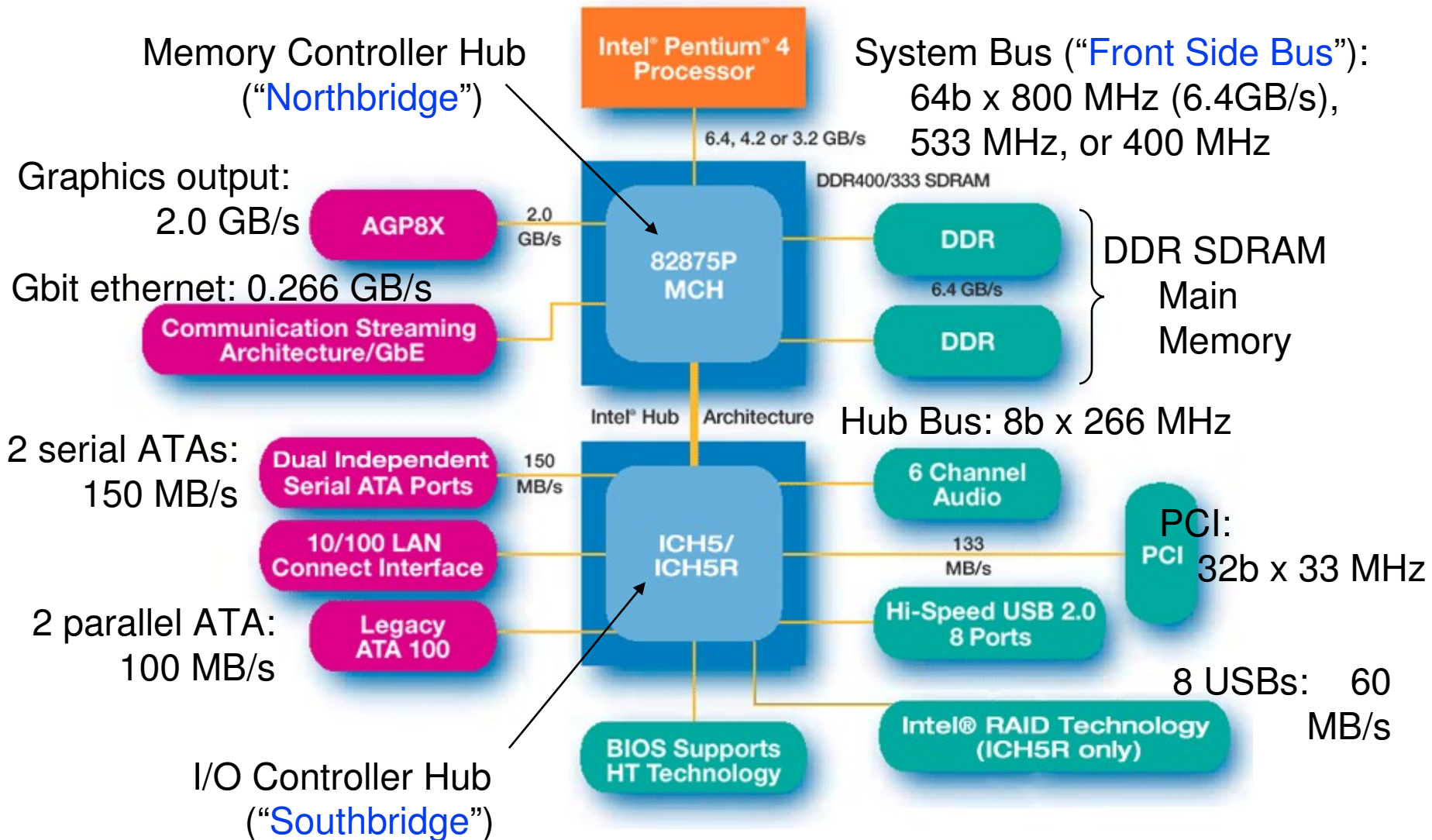
I/O device signals a request by raising **ReadReq** and putting the **addr** on the data lines

Bus Bandwidth Determinates

- ❑ The bandwidth of a bus is determined by
 - Whether its is synchronous or asynchronous and the timing characteristics of the protocol used
 - The data bus width
 - Whether the bus supports block transfers or only word at a time transfers

	Firewire	USB 2.0
Type	I/O	I/O
Data lines	4	2
Clocking	Asynchronous	Synchronous
Max # devices	63	127
Max length	4.5 meters	5 meters
Peak bandwidth	50 MB/s (400 Mbps) 100 MB/s (800 Mbps)	0.2 MB/s (low) 1.5 MB/s (full) 60 MB/s (high)

Example: The Pentium 4's Buses



Buses in Transition

- ❑ Companies are transitioning 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., ~400 MHz) so companies are transitioning to buses with a few one-way wires running at a very high “clock” rate (~2 GHz)

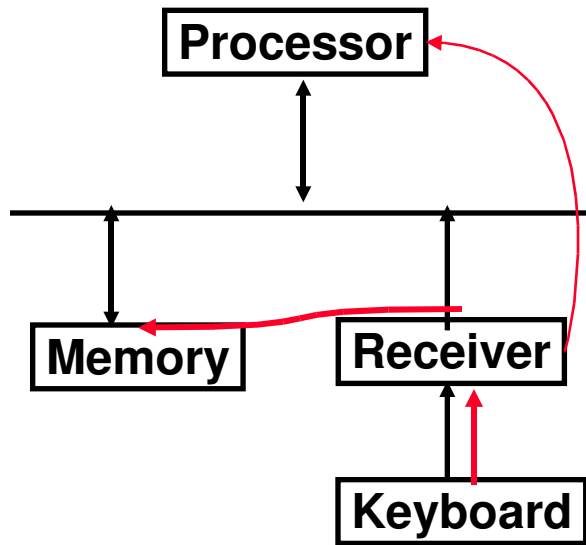
	PCI	PCIexpress	ATA	Serial ATA
Total # wires	120	36	80	7
# data wires	32 – 64 (2-way)	2 x 4 (1-way)	16 (2-way)	2 x 2 (1-way)
Clock (MHz)	33 – 133	635	50	150
Peak BW (MB/s)	128 – 1064	300	100	375 (3 Gbps)

Communication of I/O Devices and Processor

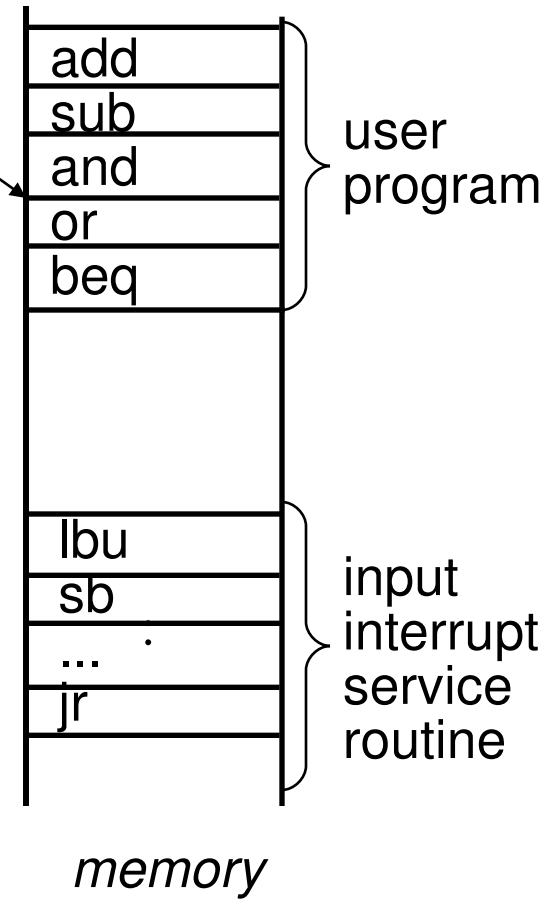
- How the processor directs the I/O devices
 - Special I/O instructions
 - Must specify both the device and the command
 - Memory-mapped I/O
 - Portions of the high-order memory address space are assigned to each I/O device
 - Read and writes to those memory addresses are interpreted as commands to the I/O devices
 - Load/stores to the I/O address space can only be done by the OS

- How the I/O device communicates with the processor
 - Polling – the processor periodically checks the status of an I/O device 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 the processor to indicate that it needs attention

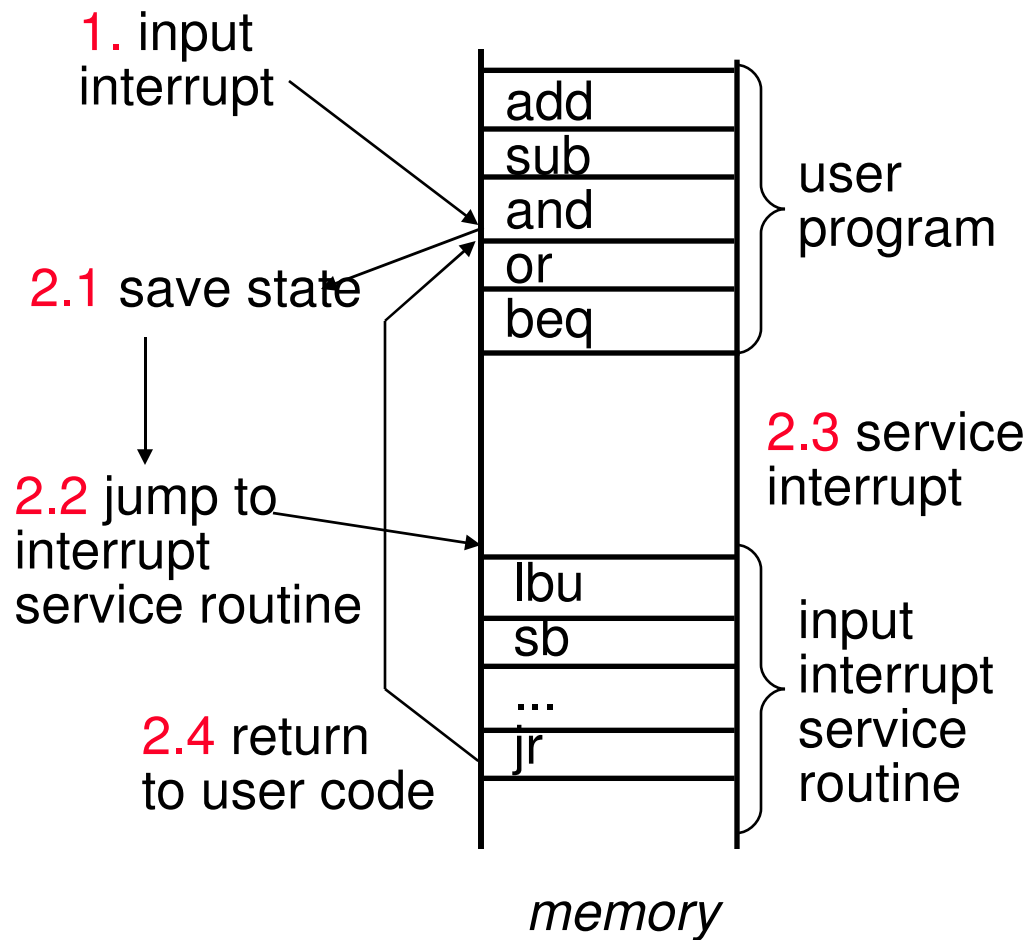
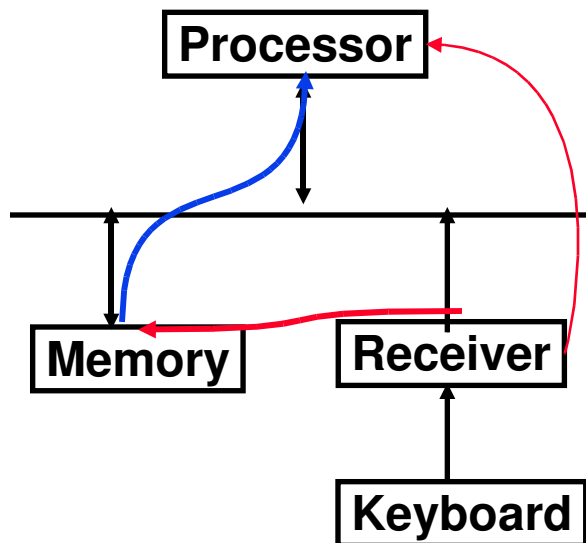
Interrupt-Driven Input



1. input interrupt



Interrupt-Driven Input



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 may need to be prioritized)
- ❑ 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
 - Cause an interrupt (I/O device) and detect an interrupt and save the necessary information to resume normal processing after servicing the interrupt (processor)

Direct Memory Access (DMA)

- ❑ For high-bandwidth devices (like disks) interrupt-driven I/O would consume a *lot* of processor cycles
- ❑ DMA – the I/O controller has the ability to transfer 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 I/O DMA controller manages the entire transfer (possibly thousand of bytes in length), arbitrating for the bus
 3. When the DMA transfer is complete, the I/O controller interrupts the processor to let it know that the transfer is complete
- ❑ There may be multiple DMA devices in one system
 - Processor and I/O controllers contend for bus cycles and for memory

The DMA Stale Data 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 read (from disk to memory) – the processor will be using **stale** data if that location is also in the cache
 - For a DMA write (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 is solved by
 1. Routing all I/O activity through the cache – expensive and a large negative performance impact
 2. Having the OS selectively invalidate the cache for an I/O read or force write-backs for an I/O write (flushing)

I/O and the Operating System

- ❑ The operating system acts as the interface between the I/O hardware and the program requesting I/O
 - To protect the **shared I/O resources**, the user program is not allowed to communicate directly with the I/O device

- ❑ Thus OS must be able to give commands to I/O devices, handle interrupts generated by I/O devices, provide equitable access to the shared I/O resources, and schedule I/O requests to enhance system throughput
 - I/O interrupts result in a transfer of processor control to the supervisor (OS) process