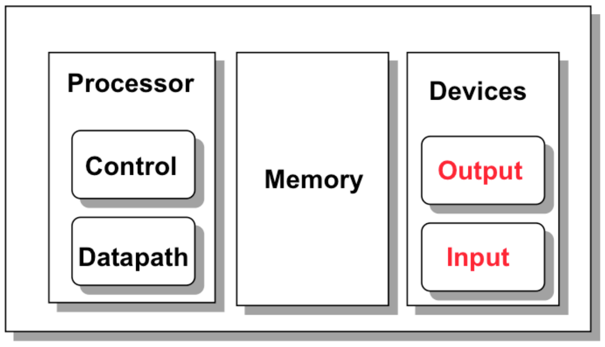
I/O Systems

[Adapted from Mary Jane Irwin for
Computer Organization and Design,
Patterson & Hennessy, © 2005, UCB]

Other handouts
To handout next time

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

Here are some examples of the various I/O devices you are probably familiar with.

Notice that most I/O devices that have human as their partner usually have relatively low peak data rates because humans in general are slow relatively to the computer system.

The exceptions are the laser printer and the graphic displays.

Laser printer requires a high data rate because it takes a lot of bits to describe the high resolution image you like to print by the laser writer.

The graphic display requires a high data rate because all the color objects we see in the real world and take for granted are very hard to replicate on a graphic display.

Data rates span eight orders of magnitude and are always quoted in base 10 so that 10 Mb/sec = 10,000,000 bits/sec

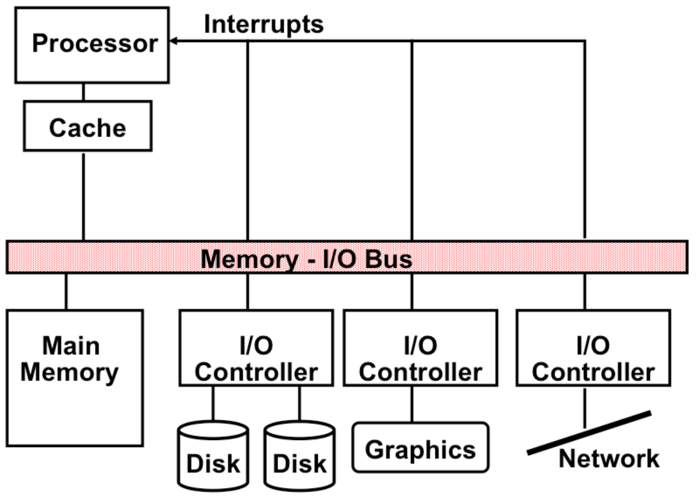
I/O Performance Measures

- 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

A Typical I/O System



This is a more in-depth picture of the I/O system of a typical computer.

The I/O devices are shown here to be connected to the computer via I/O controllers that sit on the Memory I/O busses.

For now, notice that I/O system is inherently more irregular than the Processor and the Memory system because all the different devices (disk, graphics) that one can attach to it.

So when one designs an I/O system, performance is still an important consideration.

But besides raw performance, one also has to think about expandability and resilience in the face of failure.

For example, one has to ask questions such as:

(a)[Expandability]: is there any easy way to connect another disk to the system?

(b) And if this I/O controller (network) fails, is it going to affect the rest of the network?

+2 = 7 min (X:47)

I/O System Performance

- Designing an I/O system to meet a set of bandwidth and/or latency constraints means:

1. Finding the weakest link in the I/O system – the component that constrains the design
   - The processor and memory system ?
   - The underlying interconnection (e.g., bus) ?
   - The I/O controllers ?
   - The I/O devices themselves ?
2. (Re)configuring the weakest link to meet the bandwidth and/or latency requirements
3. Determining requirements for the rest of the components and (re)configuring them to support this latency and/or bandwidth

Look for the bottleneck

I/O System Performance Example

- A disk workload consisting of 64KB reads and writes where the user program executes 200,000 instructions per disk I/O operation and
  - a processor that sustains 3 billion instr/s and averages 100,000 OS instructions to handle a disk I/O operation

The maximum disk I/O rate (# I/O's/s) of the processor is

$$\frac{\text{Instr execution rate}}{\text{Instr per I/O}} = \frac{3 \times 10^9}{(200 + 100) \times 10^3} = \boxed{10{,}000 \text{ I/O's/s}}$$

  - a memory-I/O bus that sustains a transfer rate of 1000 MB/s

Each disk I/O reads/writes 64 KB so the maximum I/O rate of the bus is

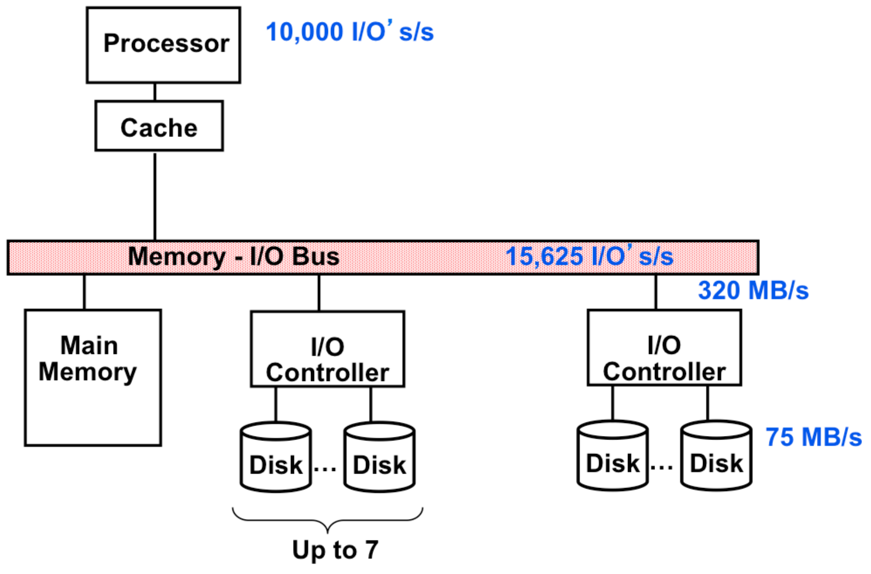
$$\frac{\text{Bus bandwidth}}{\text{Bytes per I/O}} = \frac{1000 \times 10^6}{64 \times 10^3} = \boxed{15{,}625 \text{ I/O's/s}}$$

  - SCSI disk I/O controllers with a DMA transfer rate of 320 MB/s that can accommodate up to 7 disks per controller
  - disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms

what is the maximum sustainable I/O rate and what is the number of disks and SCSI controllers required to achieve that rate?

For lecture

Disk I/O System Example



Start of second hour

I/O System Performance Example, Con’t

So the processor is the bottleneck, not the bus

- disk drives with a read/write bandwidth of 75 MB/s and an average seek plus rotational latency of 6 ms

Disk I/O read/write time = seek + rotational time + transfer time =
6ms + 64KB/(75MB/s) = 6.9ms

Thus each disk can complete 1000ms/6.9ms or 146 I/O’s per second. To saturate the processor requires 10,000 I/O’s per second or
10,000/146 = 69 disks

To calculate the number of SCSI disk controllers, we need to know the average transfer rate per disk to ensure we can put the maximum of 7 disks per SCSI controller and that a disk controller won’t saturate the memory-I/O bus during a DMA transfer

Disk transfer rate = (transfer size)/(transfer time) = 64KB/6.9ms = 9.56 MB/s

Thus 7 disks won’t saturate either the SCSI controller (with a maximum transfer rate of 320 MB/s) or the memory-I/O bus (1000 MB/s). This means we will need 69/7 or 10 SCSI controllers.

Note that only one disk controller will be using the bus at a time, so you only need to be concerned about how much load ONE disk I/O controller is putting on the memory-I/O bus when doing a DMA transfer (i.e., don’t have to worry about the processor bandwidth here).

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

The two major advantages of the bus organization are versatility and low cost.

By versatility, we mean new devices can easily be added.

Furthermore, if a device is designed according to a industry bus standard, it can be move between computer systems that use the same bus standard.

The bus organization is a low cost solution because a single set of wires is shared in multiple ways.

The major disadvantage of the bus organization is that it creates a communication bottleneck. When I/O must pass through a single bus, the bandwidth of that bus can limit the maximum I/O throughput.

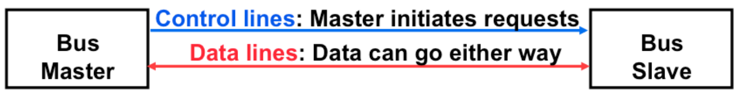
The maximum bus speed is also largely limited by:

(a) The length of the bus.

(b) The number of I/O devices on the bus.

(C) And the need to support a wide range of devices with a widely varying latencies and data transfer rates.

Bus Characteristics


Bus Master
Control lines: Master initiates requests
Data lines: Data can go either way
Bus Slave

- 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

A bus generally contains a set of control lines and a set of data lines. The control lines are used to signal requests and acknowledgments and to indicate what type of information is on the data lines. The data lines carry information between the source and the destination. This information may consists of data, addresses, or complex commands.

A bus transaction includes two parts: (a) sending the address and (b) then receiving or sending the data.

The bus master is the one who starts the bus transaction by sending out the address. The slave is the one who responds to the master by either sending data to the master if the master asks for data. Or the slave may end up receiving data from the master if the master wants to send data.

In the simplest system, the processor is the one and ONLY one bus master and all bus requests must be controlled by the processor. The major drawback of this simple approach is that the processor needs to be involved in every bus transaction and can use up too many processor cycles.

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., SCSI, 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, PCIexpress)
  - The backplane is an interconnection structure within the chassis
  - Used as an intermediary bus connecting I/O busses to the processor-memory bus

Buses are traditionally classified as one of 3 types: processor memory buses, I/O buses, or backplane buses. The processor memory bus is usually design specific while the I/O and backplane buses are often standard buses.

In general processor bus are short and high speed. It tries to match the memory system in order to maximize the memory-to-processor BW and is connected directly to the processor.

I/O bus usually is lengthy and slow because it has to match a wide range of I/O devices and it usually connects to the processor-memory bus or backplane bus.

Backplane bus receives its name because it was often built into the backplane of the computer--it is an interconnection structure within the chassis.

It is designed to allow processors, memory, and I/O devices to coexist on a single bus so it has the cost advantage of having only one single bus for all components.

+2 = 16 min. (X:56)

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)

There are substantial differences between the design requirements for the I/O buses and processor-memory buses and the backplane buses.

Consequently, there are two different schemes for communication on the bus: synchronous and asynchronous.

Synchronous bus includes a clock in the control lines and a fixed protocol for communication that is relative to the clock.

Since the protocol is fixed and everything happens with respect to the clock, it involves little logic and can run very fast. Most processor-memory buses fall into this category.

Synchronous buses have two major disadvantages: (1) every device on the bus must run at the same clock rate. (2) And if they are fast, they must be short to avoid clock skew problem.

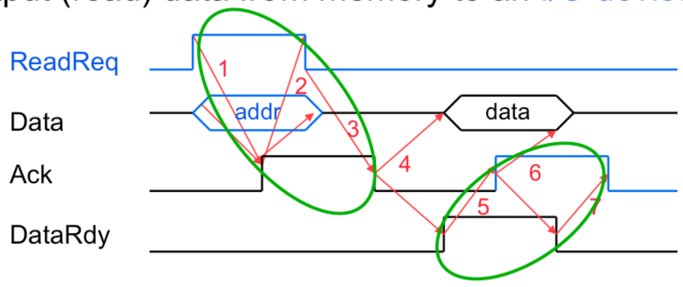
By definition, an asynchronous bus is not clocked so it can accommodate a wide range of devices at different clock rates and can be lengthened without worrying about clock skew.

The draw back is that it can be slow and more complex because a handshaking protocol is needed to coordinate the transmission of data between the sender and receiver.

+2 = 28 min. (Y:08)

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

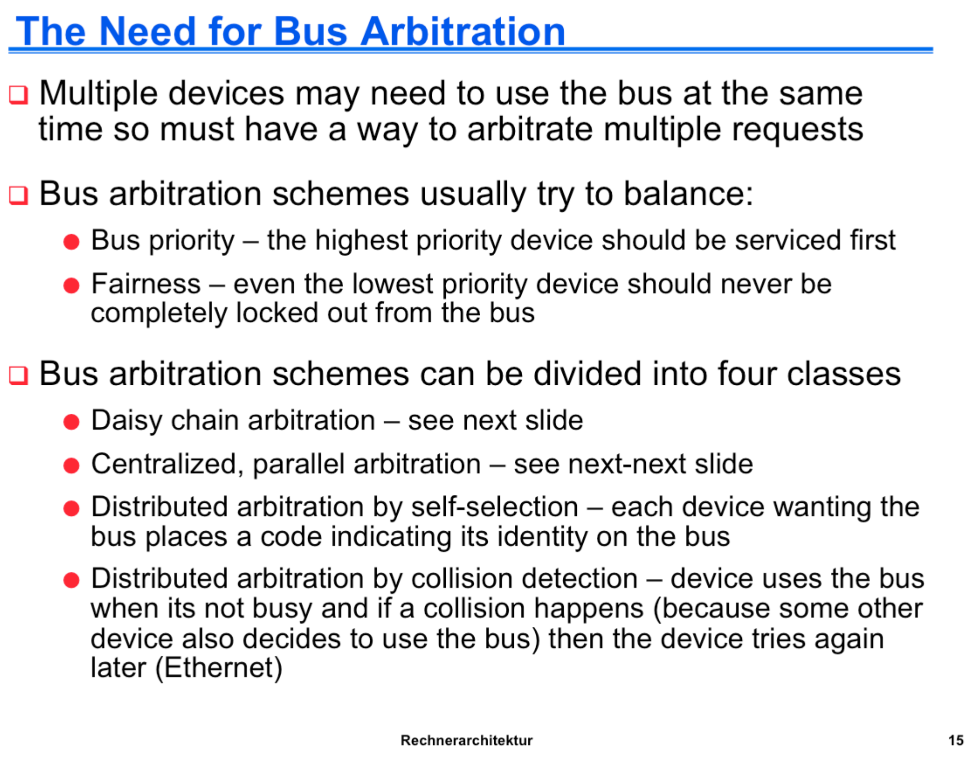
1. Memory sees ReadReq, reads addr from data lines, and raises Ack
2. I/O device sees Ack and releases the ReadReq and data lines
3. Memory sees ReadReq go low and drops Ack
4. When memory has data ready, it places it on data lines and raises DataRdy
5. I/O device sees DataRdy, reads the data from data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

ReadReq: used to indicate a read request for memory. The address is put on the data lines at the same time.

DataRdy: used to indicate that the data word is now ready on the data lines. In an output transaction, the memory will assert this signal. In an input transaction, the I/O device will assert it.

Ack: used to acknowledge the ReadReq or the DataRdy signal of the other party.

The control signals ReadReq and DataRdy are asserted until the other party indicates that the control lines have been seen and the data lines have been read; This indication is made by asserting the Ack line - HANDSHAKING.



The Need for Bus Arbitration

- ❑ Multiple devices may need to use the bus at the same time so must have a way to arbitrate multiple requests
- ❑ Bus arbitration schemes usually try to balance:
  - ● Bus priority – the highest priority device should be serviced first
  - ● Fairness – even the lowest priority device should never be completely locked out from the bus
- ❑ Bus arbitration schemes can be divided into four classes
  - ● Daisy chain arbitration – see next slide
  - ● Centralized, parallel arbitration – see next-next slide
  - ● Distributed arbitration by self-selection – each device wanting the bus places a code indicating its identity on the bus
  - ● Distributed arbitration by collision detection – device uses the bus when its not busy and if a collision happens (because some other device also decides to use the bus) then the device tries again later (Ethernet)

A more aggressive approach is to allow multiple potential bus masters in the system.

With multiple potential bus masters, a mechanism is needed to decide which master gets to use the bus next. This decision process is called bus arbitration and this is how it works.

A potential bus master (which can be a device or the processor) wanting to use the bus first asserts the bus request line and it cannot start using the bus until the request is granted.

Once it finishes using the bus, it must tell the arbiter that it is done so the arbiter can allow other potential bus master to get onto the bus.

All bus arbitration schemes try to balance two factors: bus priority and fairness. Priority is self explanatory. Fairness means even the device with the lowest priority should never be completely locked out from the bus.

Bus arbitration schemes can be divided into four broad classes. In the fist one:

(a) Each device wanting the bus places a code indicating its identity on the bus.

(b) By examining the bus, the device can determine the highest priority device that has made a request and decide whether it can get on.

In the second scheme, each device independently requests the bus and collision will result in garbage on the bus if multiple request occurs simultaneously.

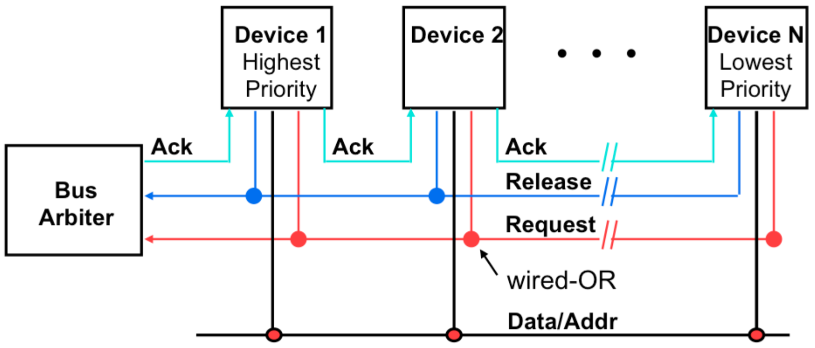
Each device will detect whether its request result in a collision and if it does, it will back off for an random period of time before trying again.

The Ethernet you use for your workstation uses this scheme.

We will talk about the 3rd and 4th schemes in the next two slides.

+3 = 38 min. (Y:18)

Daisy Chain Bus Arbitration



- Advantage: simple
- Disadvantages:
  - Cannot assure fairness – a low-priority device may be locked out indefinitely
  - Slower – the daisy chain grant signal limits the bus speed

The daisy chain arbitration scheme got its name from the structure for the grant line which chains through each device from the highest priority to the lowest priority.

The higher priority device will pass the grant line to the lower priority device ONLY if it does not want it so priority is built into the scheme.

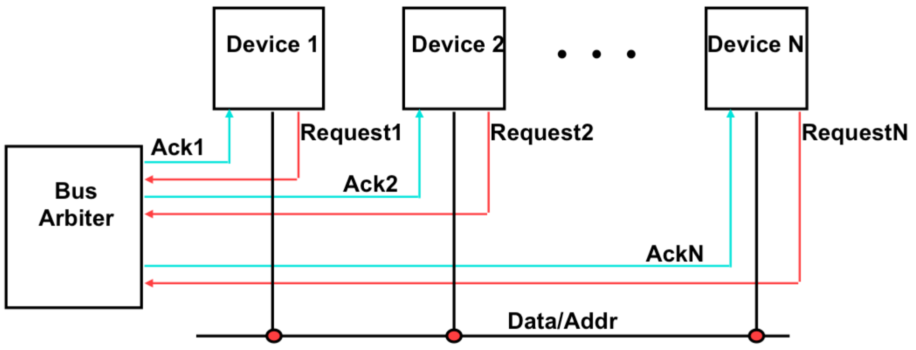
The advantage of this scheme is simple. The disadvantages are:

(a) It cannot assure fairness. A low priority device may be locked out indefinitely.

(b) Also, the daisy chain grant line will limit the bus speed.

+1 = 39 min. (Y:19)

Centralized Parallel Arbitration



- Advantages: flexible, can assure fairness
- Disadvantages: more complicated arbiter hardware
- Used in essentially all processor-memory buses and in high-speed I/O buses

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 interrupts to the processor to indicate that it needs attention

If special I/O instructions are used, the OS will use the I/O instruction to specify both the device number and the command word.

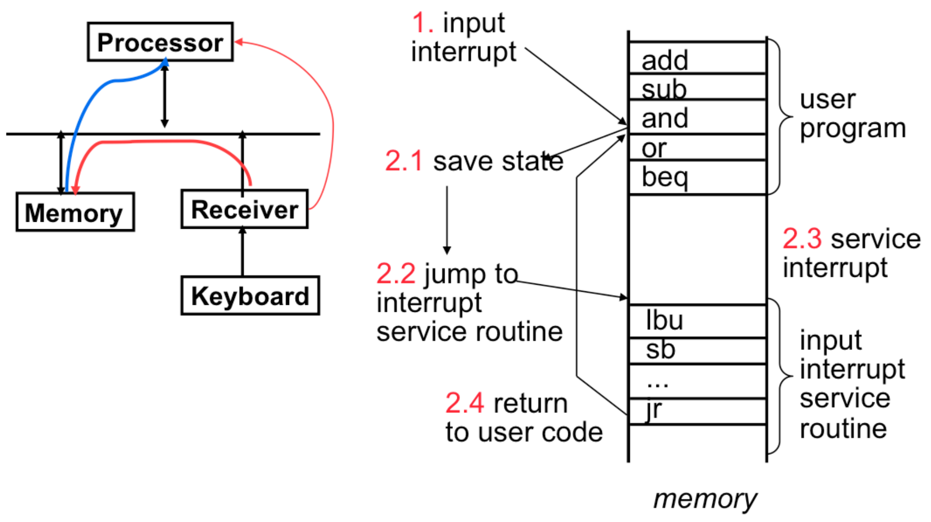
The processor then executes the special I/O instruction by passing the device number to the I/O device (in most cases) via a set of control lines on the bus and at the same time sends the command to the I/O device using the bus's data lines.

Special I/O instructions are not used that widely. Most processors use memory-mapped I/O where portions of the address space are assigned to the I/O device.

Read and write to this special address space are interpreted by the memory controller as I/O commands and the memory controller will do right thing to communicate with the I/O device.

If a polling check takes 100 cycles and the processor has a 500 MHz clock, a mouse running at 30x/s takes only 0.0006% of the processor, a floppy running at 50KB/s takes only 0.5%, but a disk running at 2MB/sec takes 10%.

Interrupt-Driven Input



That is, whenever an I/O device needs attention from the processor, it interrupts the processor from what it is currently doing.

This is how an I/O interrupt looks in the overall scheme of things. The processor is minding its business when one of the I/O device wants its attention and causes an I/O interrupt.

The processor then saves the current PC, branches to the address where the interrupt service routine resides, and starts executing the interrupt service routine.

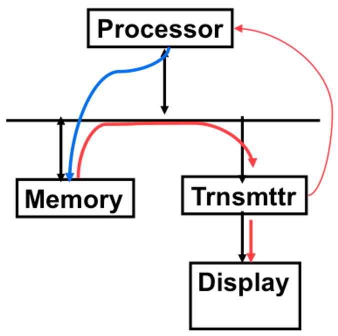
When it finishes executing the interrupt service routine, it branches back to the point of the original program where we stop and continue.

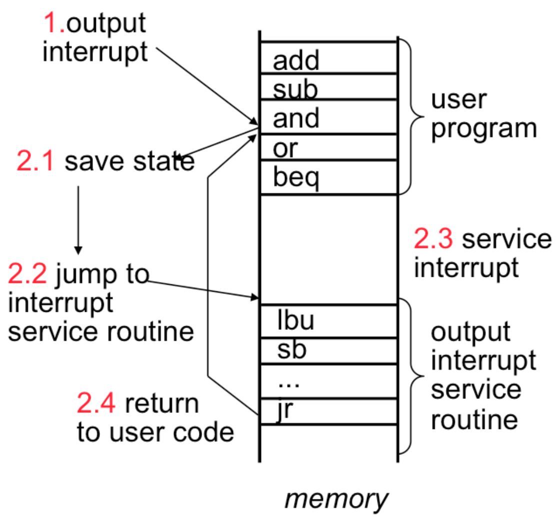
The advantage of this approach is efficiency. The user program's progress is halted only during actual transfer.

The disadvantage is that it requires special hardware in the I/O device to generate the interrupt. And on the processor side, we need special hardware to detect the interrupt and then to save the proper states so we can resume after the interrupt.

+2 = 62 min. (Y:42)

Interrupt-Driven Output





Interrupt-Driven I/O

- ❑ An I/O interrupt is asynchronous wrt instruction execution
  - Is not associated with any instruction so does not 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)

An I/O interrupt is asynchronous with respect to the instruction execution while exception such as overflow or page fault are always associated with a certain instruction.

Also for exception, the only information that needs to be conveyed is the fact that an exceptional condition has occurred but for interrupt, there is more information to be conveyed.

Let me elaborate on each of these two points.

Unlike exception, which is always associated with an instruction, interrupt is not associated with any instruction. The user program is just doing its things when an I/O interrupt occurs.

So I/O interrupt does not prevent any instruction from completing so you can pick your own convenient point to take the interrupt.

As far as conveying more information is concerned, the interrupt detection hardware must somehow let the OS know who is causing the interrupt.

Furthermore, interrupt requests needs to be prioritized.

Vectorized interrupt or interrupt cause register

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 processor will be delayed when the memory is busy doing a DMA transfer. By using caches, the processor can avoid having to access the memory most of the time, leaving most of the memory bandwidth free for use by the I/O devices.

The real issue is cache coherence. What if the I/O device writes data that is currently in the processor cache (the processor may never see the new data) or reads data that is not up-to-date (because of a write-back cache)? Solutions are to flush the cache on every I/O operation (expensive) or have the hardware invalidate the cache lines.

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)
  3. Providing hardware to selectively invalidate or flush the cache – need a hardware snooper