Interfacing to the Outside World

Introduction

There are variety of ways and reasons to talk to outside world
Consider first kinds of devices we may want to interface to
- For embedded systems tasks include
  - Measuring
  - Controlling
  - Interacting with other subsystems
  - Computing
    - Accelerators
    - Graphics
  - Communications

Collectively such devices called I/O system
Devices we may interact with include
- File systems and storage devices
  - Read and write data
- Network
  - Send and receive data
  - Execute remote procedures
- Keyboard and mouse
  - Interact with computer
- Scanning devices
  - Bring in information
- Display and printing devices
  - Display variety of data

We interact with external devices in several ways

Busses

- Comprised of
  - Address
  - Control
  - Data
- Serial
  - Bit and character
  - Character
- Parallel
  - Character
  - Character and Word

Timing

- Asynchronous
  - Transfer independent of sender / receiver timing
- Synchronous
  - Transfer dependent upon sender / receiver timing

Busses a First Look

To understand input and output

We must have some understanding of

- How we communicate with such devices

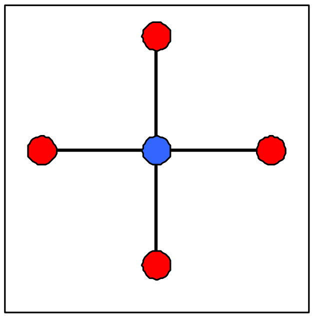
Three general configurations

- Star
- Ring
- Bus

Each has certain advantages and disadvantages

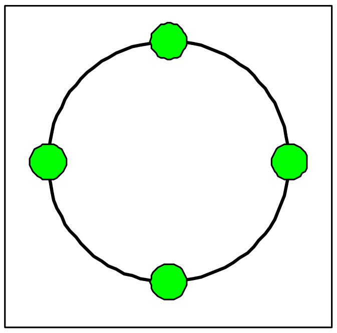
For all configurations

- Two kinds of messages
  - Addressed to single device
  - Multicast



Star

- Master slave kind of arrangement
- Device to device communication must go through master
- *Transmit*
  - Device at center directs activities and message exchange
    - With all other devices
- *Receive*
  - Master transmits to desired destinations
  - Failure
    - If one device or link fails
      - Communication with others continues



Ring

- Variants are common in communication networks
- Basis for token ring networks
  - Embedded processors in an automobile
- There is typically no bus master
- Device accepts all messages circulating in ring

If device receiving message ADDRESSED device

Message accepted

Else

Message passed on o next device

Transmit

Some protocol used to decide who is able to transmit messages

To use the bus

Receive

All devices can listen for message

Failure

Ring usually implemented as two concentric rings

If device fails or ring severed

Can repair itself by reconfiguring

Bus

Variation on the star architecture

May or may not be a bus master

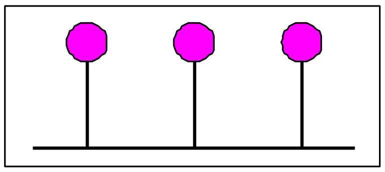
Simple bus probably one of more common architectures

Device interconnection several ways

All can transmit or receive

Some transmit or receive only

Control bus to all or request / grant configuration



Transmit

If bus master

It addresses device

Sends information

Permits another device to transmit

Else

Some protocol used to decide who is able to transmit messages

To use the bus

Messages may be

- Broadcast
- Multicast
- Polled

Receive

All devices can listen for message and act on message

Failure

Individual device failure will not compromise net

Severed net can prevent communication beyond severed point

Whichever scheme we choose

Generally must provide three basic functions

- Data
- Address
- Control

Address

An important question

How do we define and establish an address

Initial address for a device

Subsequent addresses as device added to system

Current schemes

Geographic addressing

Addresses are dynamically assigned during initial configuration

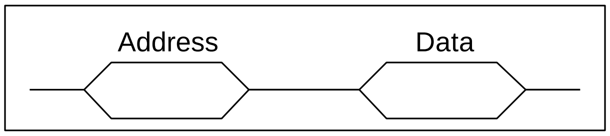
As new device attached to system and detected

Forward capabilities

Serial System

Address sent down same path as data

Appears first to select receiver



Typically address and data

Have some means to distinguish

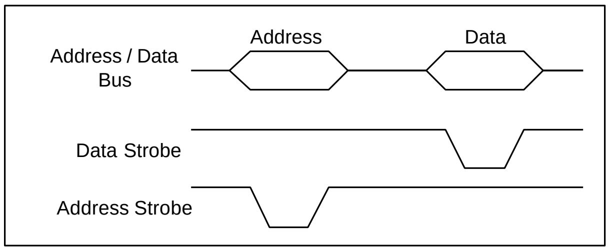
Bit pattern

Parallel System

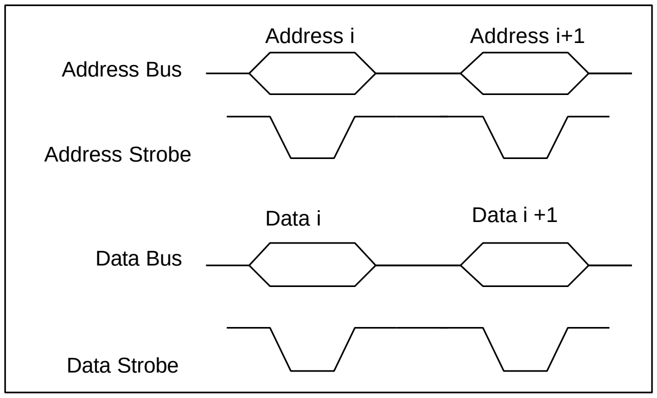
Address may be handled in two main ways

Transmit over same bus as data

- Precedes data
    - Works like serial method only faster
- Tagged to distinguish from data
    - Either as part of transmission or with control lines



- Transmit over separate bus
    - Bus may be same size as data
        - Depending upon architecture
            - May be larger if also used to access memory
            - Smaller is special bus
    - Transmitted simultaneously with data



Control

- Number and structure depend upon nature of system
- Typical signals may include
    - Read / Write
    - Address / Data present or stable
    - Clock
        - Where does the clock come from
            - Separate line
            - Encoded in the data
    - Transmission Direction
    - Ready or Active

- Synchronize
- Reset
- Power
    - May or may not be counted as control signal

Data

- Whether serial or parallel
    - Data lines carry information we wish to transmit or receive
- When in parallel
    - Will typically carry one word
    - For system with 32 bit words
        - Bus would be 32 bits wide
        - Carry DB0 - DB31
    - Important to determine which is the MSB and which the LSB

Moving Outside of the Processor

A First Look

- The procedure serves as basis for look at
    - Interacting with outside world signals
- As we've noted embedded systems
    - Designed to perform some task
- Performing task means interfacing with world outside processor
- Such tasks may include
    - Making measurements
    - Controlling motor
    - Running telecommunication system
- We are dealing with portions of von Neumann machine
    - Designated
        - Input and Output
        - Memory
            - When consider memory mapped I/O

- Our exchange with outside world
    - Synchronous
        - Based upon some timing element
    - Asynchronous
        - Start of and subsequent transactions temporally independent

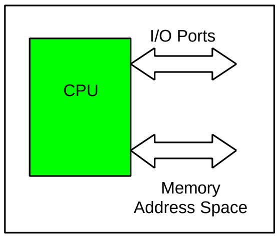
- When studying processor input and output
    - We must look at several things

- Data source or destination
    - I/O ports
    - Memory address space
- Nature of the exchange
    - Three types possible
        - Event
        - Shared variable
        - Message
    - Note
        - Such exchange also occurs between / among
            - Tasks within same processor

- How I/O procedure invoked
    - Associated restrictions
- Where procedure resides
- Protocol for the data exchange
- Timing requirements

Let's look at each of these

Data Source or Destination



Data may be exchanged with external world
  Two ways
    I/O ports
    Memory address space

I/O Ports

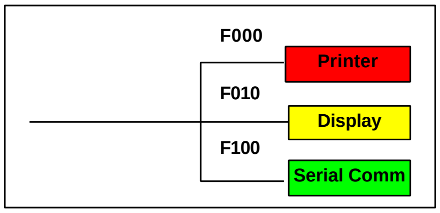
Input / Output Ports
  Number of lines on processor dedicated
    Bringing data into processor
    Sending data out of processor
Direction
  Lines may be
    Input
    Output
    Input and Output
  Unidirectional lines tend to be static
    Direction established
      Hardware

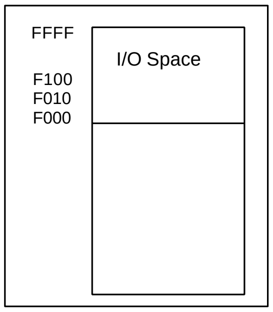
During initialization
Bi-directional lines are dynamic
Direction selected
Based upon nature of transaction
Usually grouped in sets of 4 or 8
Direction for entire group set

Memory Address Space
Read from or written to
Locations within the processors
Primary memory address space
Will elaborate shortly
When discuss memory mapped I/O

Architecture
Let's begin at a high level and examine several I/O structures
Within computer we use two basic approaches
Memory Mapped I/O
Peripheral Processor

Memory Mapped I/O
Basic idea is to for memory and I/O to share the same address space
That is map I/O space into an memory address
I/O reads and writes are done to memory addresses
Traditional scheme used in PC
Such a scheme places the responsibility of external communication
Onto CPU
Simple to implement
Adds extra task
Implementation
Assign address to each input or output device
Some devices may have multiple addresses
Read
Write
Such addresses are typically hard coded
Using switches or jumpers
Early PCs used such a scheme
Expansion led to
Address conflicts
IRQ conflicts





Limited expansion

Advantage

Easy to implement

Low cost

Disadvantage

Extra burden on CPU

Potentially slow

Peripheral Processor

As noted with memory mapped I/O

CPU must be involved in all transactions

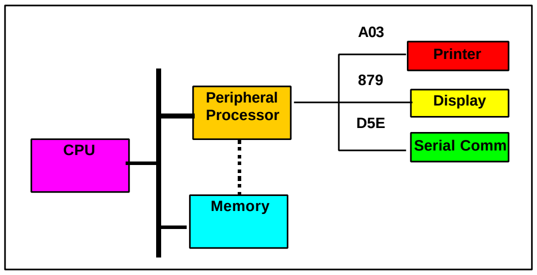
Including detailed timing

Can place significant burden on CPU

Peripheral processor scheme

Dedicates processor to handle all I/O tasks

Basic architecture appears as



Peripheral processor

May or may not be connected to memory

Several interface schemes utilized

Various levels of involvement of

System Bus

CPU

Non peripheral devices

Alternative protocols

Based upon level of CPU involvement

- CPU send / receive from peripheral processor

  PP

  Manages all I/O

  Uses bus in bursts during exchange

  Signals CPU when data available

- CPU tells peripheral processor where to find / put data in memory

  PP

  Exchanges data with memory

  Manages all I/O

  Signals CPU when data available

  Uses bus

In bursts during exchange
For duration of exchange
DMA
Using address passing and shared buffer

- CPU enables non-peripheral device and peripheral processor to communicate
  - PP
    - Exchanges data with device
    - Manages all I/O
    - Signals Device when data available
    - May signal CPU when complete
    - Uses bus
      - In bursts during exchange
      - For duration of exchange

Advantage
- I/O speed independent of CPU
- Unburdens CPU

Disadvantage
- Higher cost
- Complex

Nature of the Exchange

Three types possible
- Event
- Shared Variable
- Message
- Observe these are abstractions
  - Within each type
    - Variety of related signals

Exchange may be used
- Outside world
- Between tasks within processor

Exchange types

Event

Event is change in state signal
Usually assumed to be single signal

Generally asynchronous to executing process
Every occurrence of event
  Simultaneously activates
    Functions procedures to tasks
      Linked to it
Occurrence of event
  May or may not be stored in some way
  Becomes important issue in real time systems
Acquired in several ways
  Sampled
    One example called *polling*
  Arrives in form of asynchronous input
    One example called *interrupt*
    Processor may or may not respond

Generated
  Under software control of
    I/ O port or line
    Write to data line accompanied by control signal

Shared Variable

Variable may be read or written by multiple
  Processors
  I/O devices
Used to exchange data between asynchronous functions
Because no inter process timing constraints
  Integrity of data must be respected
    Such shared data
      Represent critical section
        Protect with
          Semaphore
          Monitor
  As we learned earlier with shared data
    Among tasks and threads

Shared variable(s) may be

- Global
  - Normally discouraged in traditional programming
  - We use such schemes in real time embedded systems

Eliminate cost of passing parameters via stack

Here we may designate

A global buffer area into which to store data

Pointer to buffer area

- Passed

  Usually done as pointer to buffer area

  Passed by task or thread

  Wishing to do I/O

Direction

Unidirectional

Bi-directional

Modification

Read

Read/Write

Nature may be

Complete word

If in memory space

1 to several bits if in I/O port space

Complex data type

Pointer exchanged

Message

May be exchanged via

Proprietary network

One of several standards

Which ever approach used

Typically implemented as hierarchy virtual networks

Above the hardware

Varying number of software layers or levels

At each level

A different language is spoken

Referred to as a *protocol*

The function is to provide services for the level above

Relationship

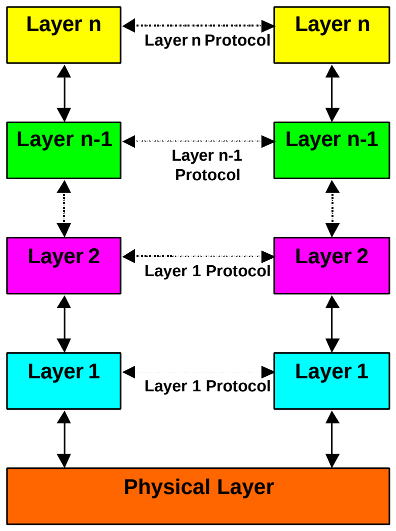
Service provider

Service consumer

Entire collection called

A *network architecture*

Set of protocols used by a machine called



A *protocol stack*

Information sent on each level called

A *message* or *messages*

It is possible that message on higher level

Composed of several lower level messages

Two major protocol schemes or stacks used

OSI

Open Systems Interconnection model

Proposed and developed by

The International Standards Organization

Comprises 7 layer virtual machine

TCP /IP

Transmission Control Protocol / Internet Protocol

Comprises 5 layer virtual machine

Physical and data link layers of OSI

Combined into

Host to network layer

OSI	**TCP/IP**
Physical	Host to Network
Data Link	
Network	Internet
Transport	Transport
Session	Not Present
Presentation	Not Present
Application	Application

When we elect to communicate via either standard

We generally integrate a

Commercially available protocol stack

Messages comprise

Several complete words

Simple

1 - 2 words transferred

Complex

Several blocks of data

Message exchange

Can be viewed as producer - consumer relationship

Usually follows some protocol

Start of message

Header information

Body

Error management

End of message

- Message may be transmitted
  - Serial / Parallel

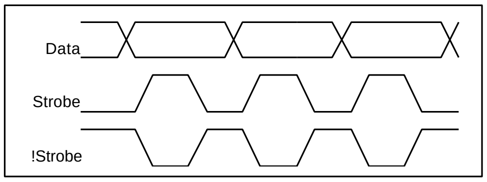
  - Serial
    - Data transferred into out of processor
    - 1 - 2 lines
      - 1 Line
        - May be
          - Unidirectional
            - Input or output only
            - Half duplex
          - Bi-directional
            - One direction at a time
            - Controlled by protocol
            - Duplex
      - 2 Lines
        - Data transferred in both directions
          - Simultaneously
          - Full duplex
    - Perhaps some control lines

  - Parallel
    - Usually width of word
    - Transmission
      - Unidirectional
        - Input or output only
      - Bi-directional
        - One direction at a time

Timing and Data Exchange

- Interface can be designed to be synchronous or asynchronous
- Let’s look first at the asynchronous

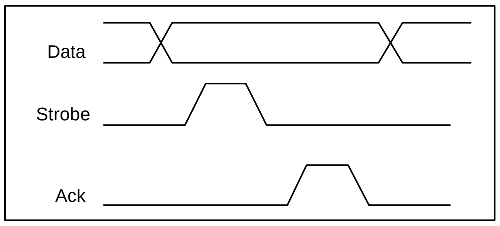
- Asynchronous
  - No clock
  - Exchanges co-ordinated using some form of handshaking protocol
  - Such protocols can be simple or complex
  - Typical examples



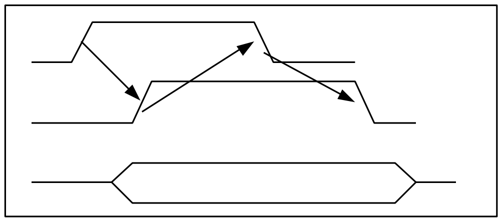
Strobe associated with each data word
- No acknowledgment of acceptance

Observe
- Strobe can be of either polarity

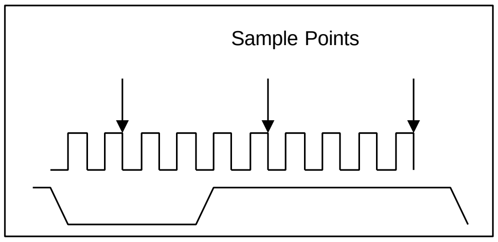
Strobe associated with each data word
- Each data word acknowledged
  - With return strobe



Full handshake
- Ready for data
- Here's data
- I've got it
- OK
- Exchange looks like this



Time from bit transition
- Often used in serial protocols
- Agreed upon
  - Change in state on data line signals
    - Start of transmission
    - End of transmission



Potential problems
- Difficult to test
- Clock noise more difficult to filter out
- Potentially complex protocol
  - To identify start / end of transmission

Potential advantages
- Devices may run at different / differing speed
- No clock skew on long busses

Synchronous

There are several drawbacks of the asynchronous transmission schemes
- Extra overhead of control bits
- Bit clock synchronization scheme less reliable at higher data rates

Problems can be alleviated with synchronous transmission

Still must achieve
- Bit, character, frame synchronization

Frame synchronization usually derived from the former
Generally includes clock in control lines

Exchanges between sender and receiver

Synchronized to the clock

- Directly
- Signals derived from the clock

Manchester phase encoding

Serial Exchange

Clock either separate or encoded in data

Parallel Exchange

Clock one of control lines

Bit Synchronization

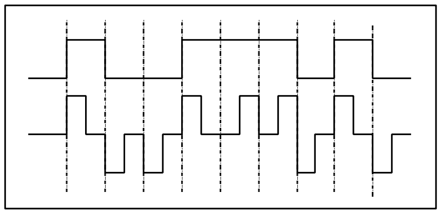
Two schemes typically used

Encode the clock in the data

Re-derive the clock from the data

Encoded Clock

Three different methods generally used



Bipolar Encoding

Binary 0's and 1's are represented by

Different polarity signals

Each bit cell contains clocking information

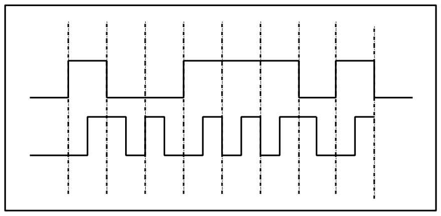
Observe

Signal returns to zero level after each encoded bit

Referred to as *return-to-zero RZ* signal

Scheme requires 3 distinct signal levels

Manchester Phase Encoding



Binary 0 - high to low signal transition

Binary 1 - low to high signal transition

Transition in the center of each bit cell

$0 \rightarrow 1$

$1 \rightarrow 0$

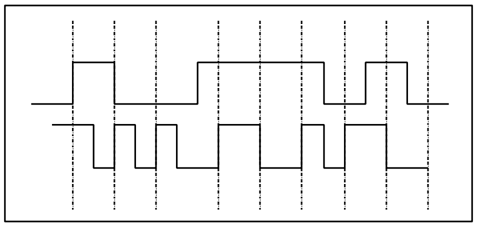
Provides the clock information

Observe

Signal does not return to zero level after each encoded bit

Referred to as *non-return-to-zero NRZ* signal

Manchester Differential Encoding



- Transition in the *center* of each bit cell
  - $0 \rightarrow 1$
  - $1 \rightarrow 0$
  - Provides the clock information
- Transition at the *start* of each bit cell
  - Only if next bit to be encoded is a binary 0

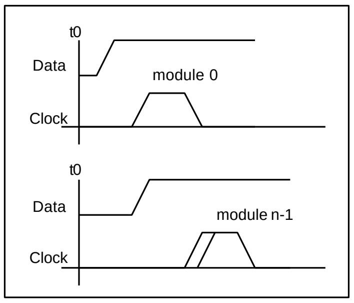
Re-derive the Clock

- Use a preamble including sync sequence
- Done using *phase lock loop- PLL*
- Based upon very stable receiver clock
- PLL used to keep sample clock locked
  - Signal transitions of incoming signal
- Data encoded to ensure a sufficient number of signal transitions
- At each transition sample timing adjusted to ensure
  - Sampling in center of bit
- Scheme will tolerate intervals without transitions
  - Based upon stability of the clock

Bit / Character / Frame Synchronization

- Once individual bits are identified

Potential problems



- All devices must run at same speed
- Clock skew on long busses
  - Because of different loading
    - Have propagation delay along bus
    - Clock arriving at different times with respect to data
      - Along bus
- Clock noise in system

Potential advantages

- Easier to test
- Simpler protocol
- Easier to stay in sync with data

Interrupts and Polling

Once we have decided upon a the physical configuration

Outside world connection

Several aspects of exchange remain

Will often be interacting with several devices

Must determine

How best to share resources

Cannot afford to continually watch device

To determine if it

Has some information

Is ready to receive (more) information

Several methods by which the is done

Polling

Interrupts

Let's examine each

Polling

With polling scheme

Controller

CPU

Peripheral Processor

In loop

We have the accompanying algorithm

1. Send device address
2. Receives response

   The response may consist of a single signal, a word, state information, or status

   a. If goal to transmit or receive data to device

   When device ready received

   Transfer to appropriate routine

   Execute transfer

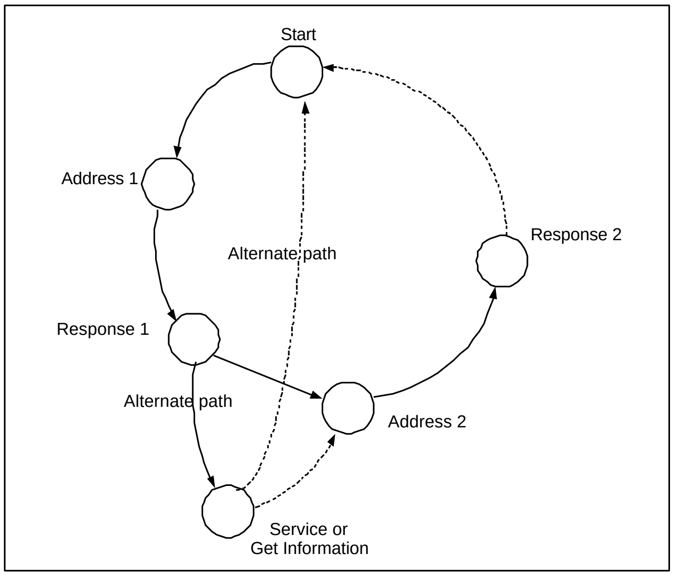
   Return to polling loop

   b. If goal is to collect status

   Can do so and continue polling other devices

   Status may include, self test results, a ready condition after power up, availability for additional transfer, data available.

We express the polling operation in the following state diagram,



In such loop
  Control device knows who it is communicating with
Problem with polling
  Control device can do nothing else while polling
Advantage however
  Process deterministic
  Time to complete predictable

Interrupt

Goal of interrupt scheme same as that of polled scheme
Difference
  Control device not dedicated to monitoring external devices
I/O system may have from 1 to many interrupt lines
Each line may be connected to 1 or several devices

Single Interrupt Line with Single Device

Let's first examine how single interrupt might work
As name suggests
  Interrupt signal interrupts foreground activity

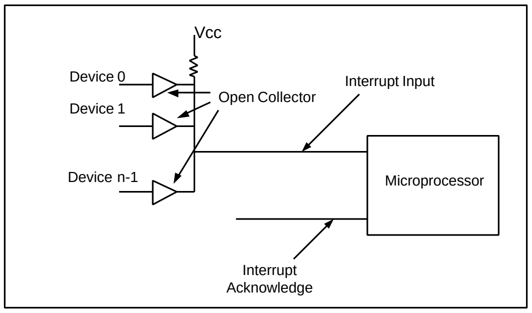
  When interrupt occurs several actions possible
    Ignore
    Respond

  <u>Ignore</u>
    Will discuss shortly

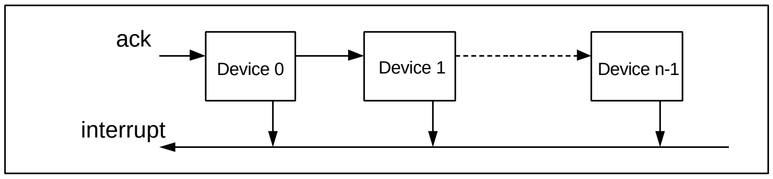
  <u>Respond</u>
    Responding to interrupt much like subroutine call
    Procedure
      Suspend current process
      Branch to routine appropriate to interrupting device
      Execute routine
      Resume former process

Single Interrupt Line with Multiple Devices

Dealing with multiple devices on single interrupt only slightly more complicated
The interrupt is connected to controller as shown,


Vcc
Device 0
Device 1
Device n-1
Open Collector
Interrupt Input
Microprocessor
Interrupt Acknowledge

Dilemma of course
- Identifying which device is requesting service
- Observe device priority determined by
  - Physical proximity to controller
    - Closest device has highest priority.



Several alternatives
- Polling
- Vector

<u>Polling</u>
- Straight forward
- Sequence
  - Interrupt occurs
  - Suspend current task
  - Branch to handler
    - Poll devices to identify
    - Branch to appropriate handler routine
      - Handle
    - Return
  - Resume suspended task

<u>Vector</u>
- With vector scheme
- Device requesting service
  - Interrupts
  - When acknowledged
    - Returns identifier
      - Device name
      - Address of service routine
  - Sequence then proceeds as with polled alternative

- <u>Multiple</u>
  - With several devices on the same line
  - Must now address question
    - How to accommodate multiple
      - Simultaneous interrupts
      - Sequential interrupts

    - *Simultaneous*
      - Must have priority scheme
      - Can be
        - Physical proximity
        - Assigned
      - Higher priority device gets serviced first

    - *Sequential*
      - If second interrupt occurs while handling first
        - Ignore until finished with first
        - Assign each device priority
          - If interrupting device has higher priority
            - Handle in same way
            - Similar to calling subroutine from subroutine

Multiple Interrupt Lines with Single Device on Each

- For each line
  - Same as single line with single device
- With multiple lines
  - Must assign priority to each
  - Same as single line with multiple devices

Multiple Interrupt Lines with Multiple Devices on Each

- For each line
  - Same as single line with multiple devices
- With multiple lines
  - Must assign priority to each
  - Same as single line with multiple devices