University of Glasgow

UESTC1008: Microelectronic Systems

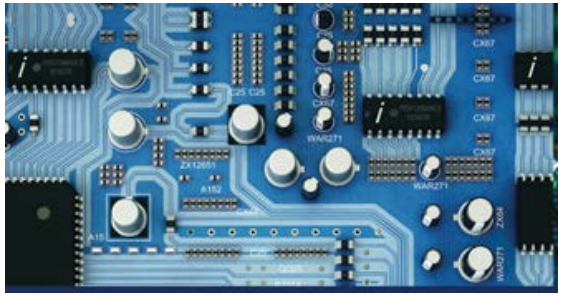
Academic year 2019/2020 – Semester 2 – Presentation 4

Qammer H. Abbasi, Lei Zhang, Sajjad Hussain, Muhammad Ali Imran and Guodong Zhao
{*qammer.abbasi,Lei.zhang,sajjad.Hussain,Muhammad.Imran,guodong.zhao*}*@glasgow.ac.uk*

"A good student never steals or cheats"

Agenda

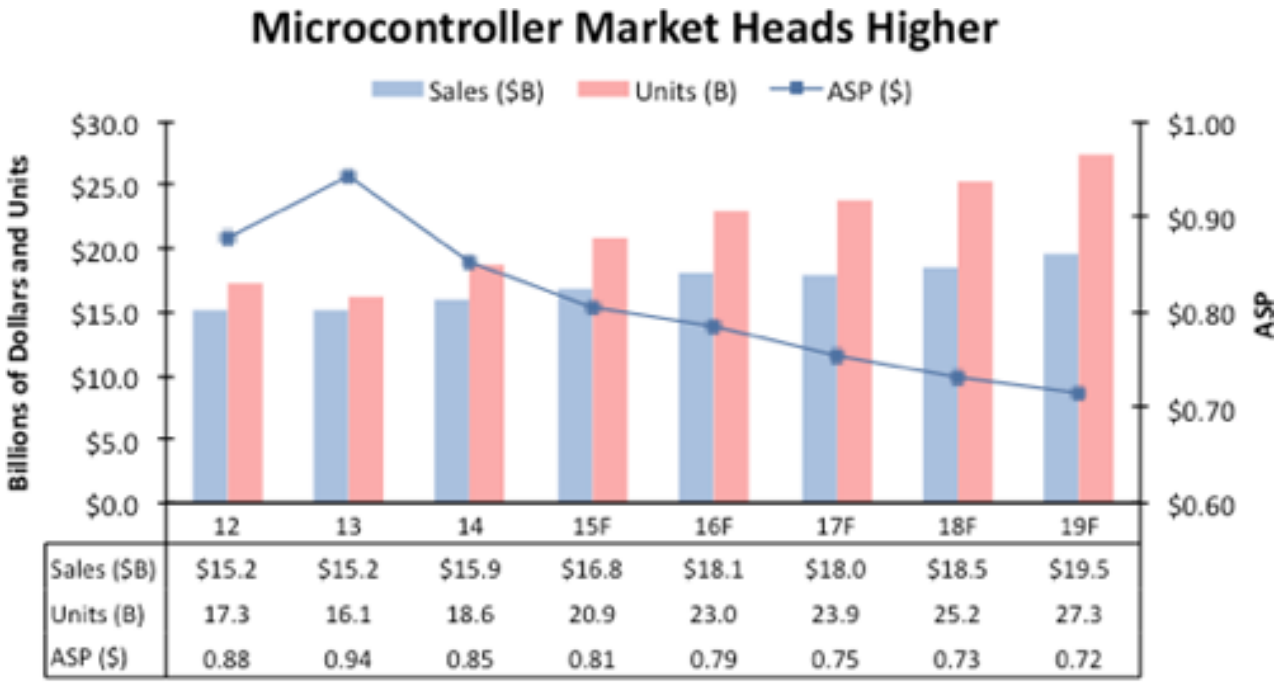
- Review of previous lecture
- Architecture
- Digital I/O
- Summary



Where to find microcontroller

- Personal information products: phone, pager, watch, pocket recorder, calculator
- Laptop components: mouse, keyboard, modem, fax card, sound card, battery charger
- Home appliances: door lock, alarm clock, thermostat, air conditioner, tv remote, hair dryer, VCR, small refrigerator, exercise equipment, washer/dryer, microwave oven
- Toys; video games, cars, dolls, etc.
- Usually anything with a keypad

Microcontroller market



	12	13	14	15F	16F	17F	18F	19F
Sales ($B)	$15.2	$15.2	$15.9	$16.8	$18.1	$18.0	$18.5	$19.5
Units (B)	17.3	16.1	18.6	20.9	23.0	23.9	25.2	27.3
ASP ($)	0.88	0.94	0.85	0.81	0.79	0.75	0.73	0.72

Source: IC Insights

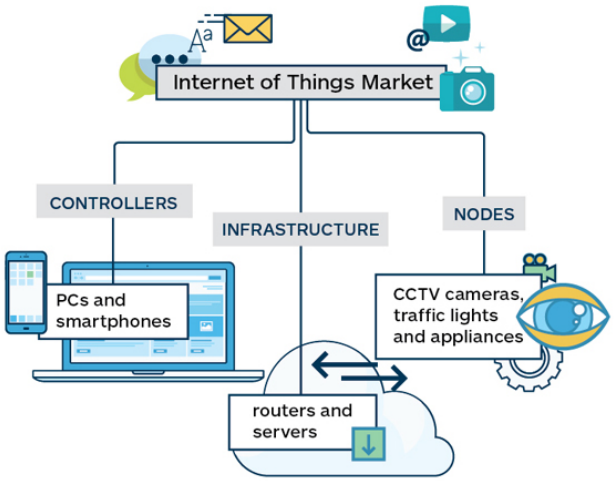
- $15Bn to $20Bn market
- Steady increase over time
- Drop in average selling price over time

- Market share (approx.)
  - 4 bit : 1%,
  - 8 bit 40%,
  - 16bit 20%,
  - 32bit 19%

Microcontroller Market Growth to Rise in Internet of Things Applications

What some still consider to be only a hype surrounding the emerging Internet of things (IoT) trends has already begun disrupting the microcontroller units (MCU) market

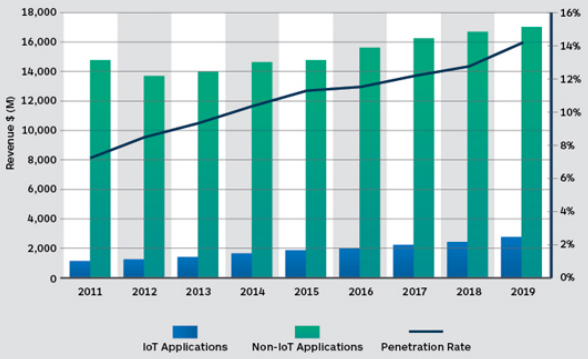
Tom Hackenberg
Senior analyst for IHS Technology



The IoT trend has a strong relationship with the MCU market, as the small nodes used for connectivity, and sensor hubs to collect and log data are primarily based on MCU platforms

The market for MCUs used in connected cars, wearable electronics, building automation and other IoT applications is expected to grow at an overall compound annual growth rate (CAGR) of 11%, from $1.7 billion in 2014 to $2.8 billion in 2019.

MCU market in IoT applications compared to markets outside of IoT



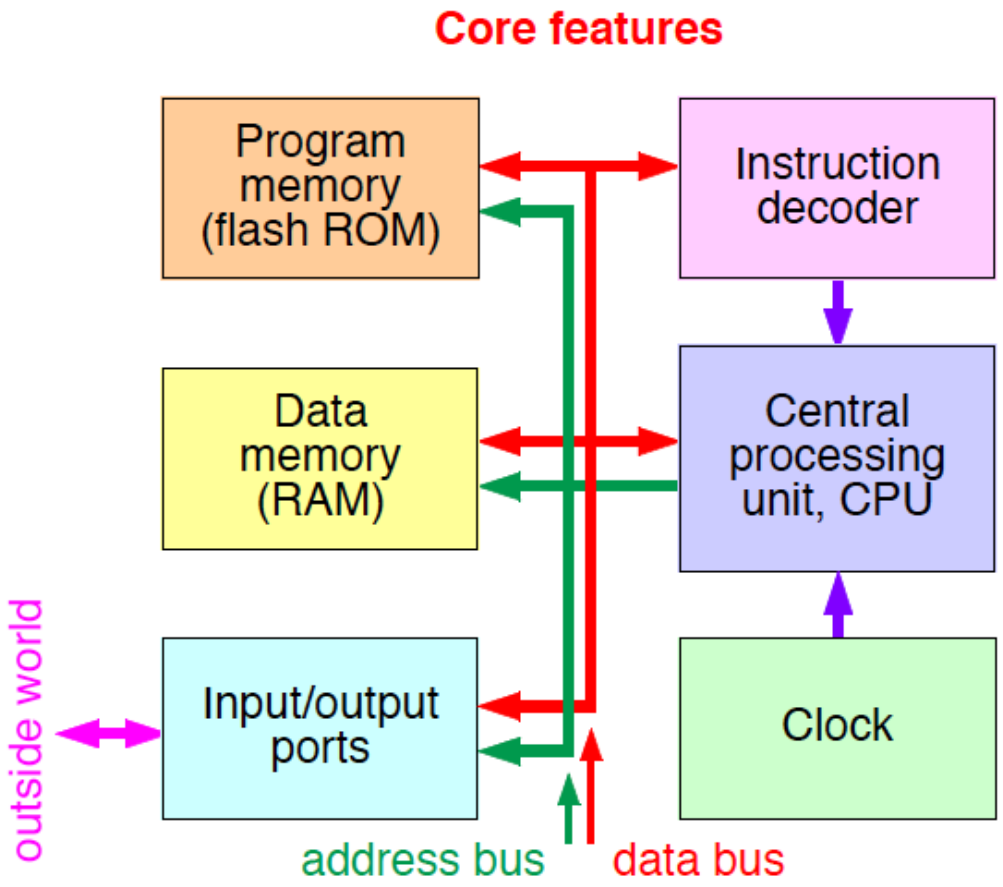
IoT connectivity demands a new consideration of semiconductor features. Many semiconductor companies have begun developing IoT platform solutions and among the semiconductor suppliers adopting IoT-focused strategies are: *Atmel Corporation, Broadcom, Cisco Systems Inc, Freescale Semiconductor, Infineon Technologies, Intel, Microchip Technologies, NXP, Qualcomm, Renesas Electronics Corporation and Texas Instruments.*

From http://electronics360.globalspec.com/article/5780/

The microcontroller – a few more details

- **Clock** – keeps everything synchronised - 100 MHz for the mbed
- **CPU** – the heart of the micro – the ARM Cortex-M core in the mbed
- **Instruction decoder** – controls the chip, carries out actions for each instruction
- **Input/output ports** – serial, digital, analog, PWM etc
- **Memory** – program and data

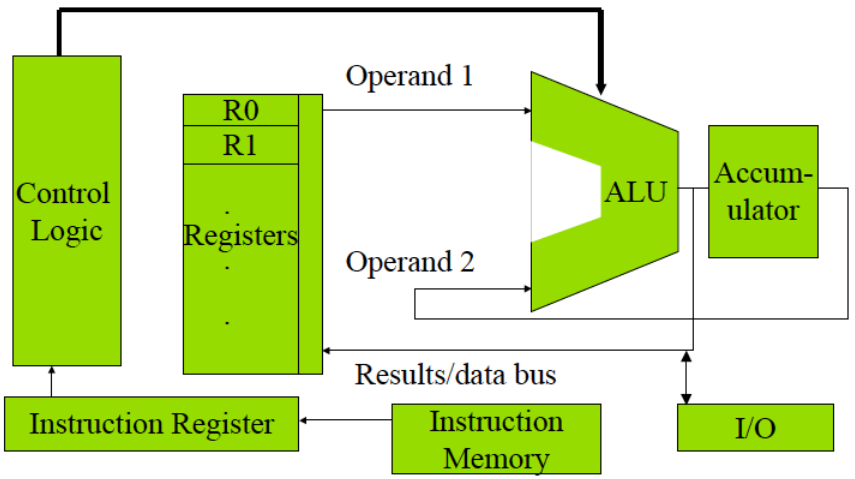
The microcontroller – a few more details



The **clock** keeps all parts synchronized.

The central processing unit

- ALU : Data manipulation
- Registers : Temporary storage of data to be manipulated
  - E.g. load number one in to register R0 from memory location yyy
  - Load number two into register R1 from memory location zzz
  - Add register R0 to R1 and put the result in R0
  - Output register R0 to memory



The program must break down your task into these simple operations

They can be calculated at GHz rates – so the system is powerful

The instruction set

- CSIC
  - Complex all encompassing instruction set
  - Many microprocessors
  - Some instructions are single clock cycle, more complex ones are several cycles (power vs. speed)
- RISC
  - Reduced instruction set of more simple commands
  - Many microcontrollers
  - Most commands operate in fewer clock cycles
  - Pipelined architecture common

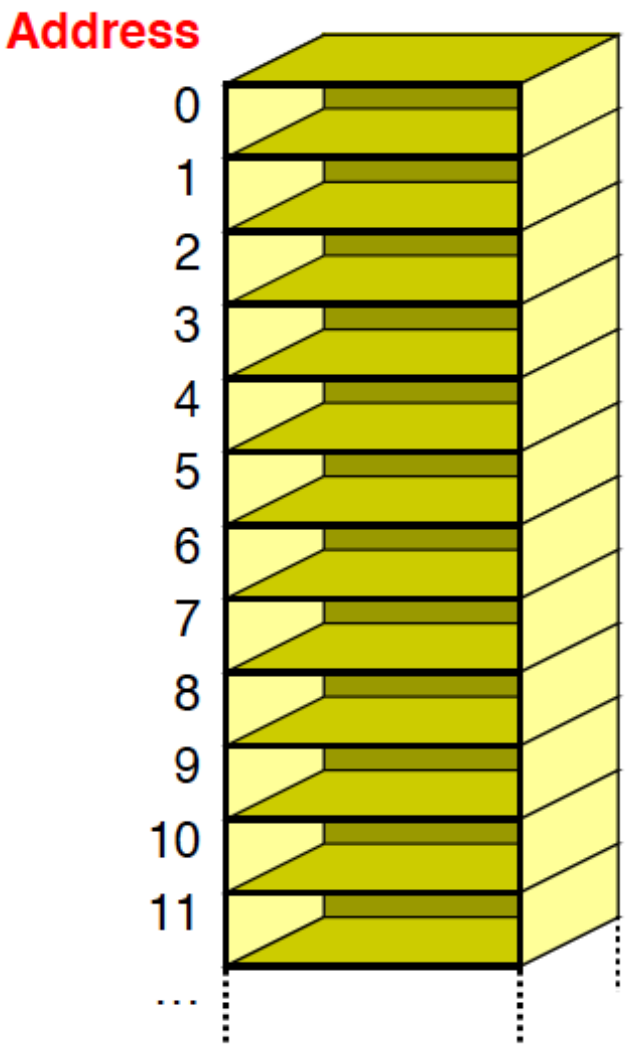
The instructions for the CPU

- Machine language
  - Fundamental basic instructions (instruction code)
  - 10000000 : Or 0x80 : Add register B to accumulator
  - Very easy to make mistakes, can be made optimum
- Assembly language
  - Mnenomics : ADD B
  - Machine specific instructions
- Programming language
  - + : but this also contains a fetch from memory, load into register B, fetch from memory and load into accumulator, add and then output accumulator to memory instructions
  - General, ‘easy’ to write, inefficient

Numbers

- 8 bit unsigned integer number
  - 8 binary bits represent the number
  - E.g. 10011101 0x9D or 9DH
  - To deal with big numbers, simply break into blocks of 4 and convert to hex
- For I/O ports, hex has a bit more meaning than decimal
- Floating point numbers (float, etc.) more complex
  - $1.2345 = +12345 \times 10^{-4} = 12345E\text{-}4$
  - Stored as Sign bit, Exponent : 8 bits, Significand : 23bits= 32bit binary number (in IEEE 754 binary format)
  - Usually don't worry how they are stored, leave the compiler to do that for us
  - Be aware of the limitation i.e. the number of significant figures in the significand

Memory



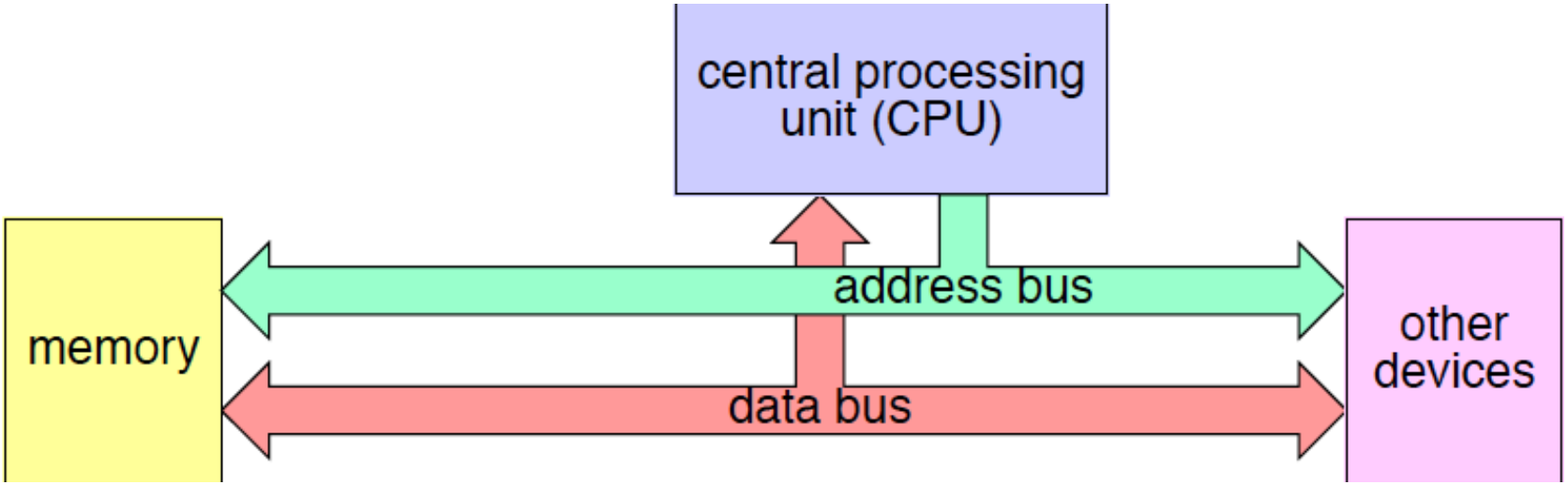
- **Memory** is just like a very tall stack.
- In the case of the mbed, each holds 32 bits.
- Each location is identified by its **address**, a serial number starting from 0. The address is said to **point to** a memory location.
- When a communication is made with memory to read or write a value, both the address and the data stored in the address must be handled.
- There may be distinct “stacks” for different types of memory.
- A memory location may be called a register, and is just like a set of 32 D-type flip-flops.

Communication with memory

Data is transferred between memory and the rest of the system on buses. These are shared sets of wires that join the components, like a multi-lane highway. In the case of the 32 bit μC on the mbed, there are 32 parallel wires.

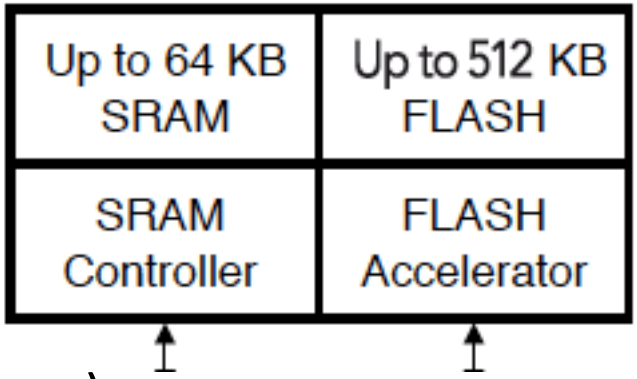
Several sets of these parallel wires are required :

- Address bus – carrier address (serial number) of the "mailtray"
- Data bus – carries the 32 bits either from the memory (read) or to the memory (write)
- Control lines are also needed to synchronise timing, select read/write, ensure that only one device tries to use the bus at once.



Memory Types

Microcontrollers have 2 types of memory



RAM – random access memory
- volatile – i.e., contents are lost when the power is removed
e.g., SRAM (static RAM – 6 transistors)

ROM - read only memory. Non-volatile – contents are retained even when power is removed. Modern ROM is flash memory.

Almost all memory in a PC is RAM. Each program must be read into RAM from non-volatile memory (hard disk) whenever it is needed. Similarly, the operating system is loaded into RAM when the system, is booted.

In contrast, microcontrollers execute only one program, which can be stored in ROM and is therefore available instantly.

Memory Architectures

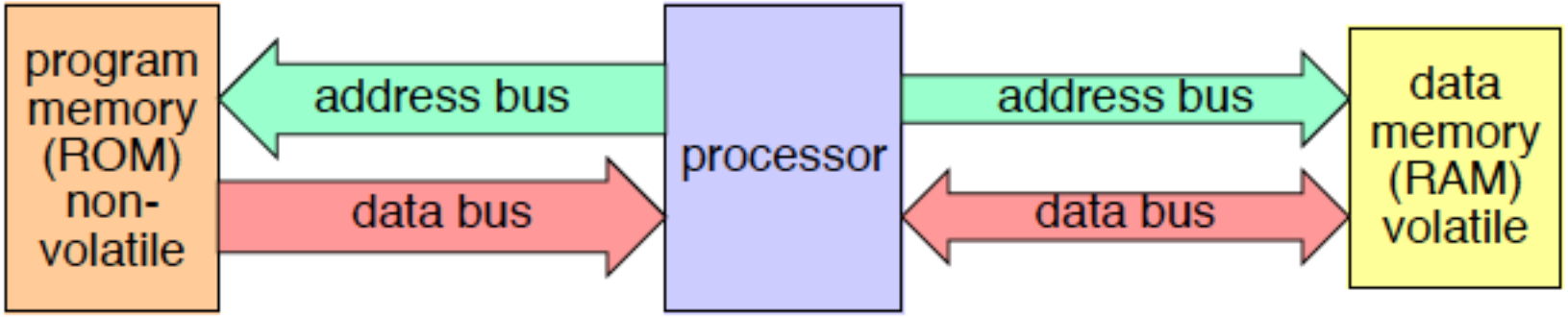
The two types of memory

non-volatile ROM for program

volatile RAM for variables

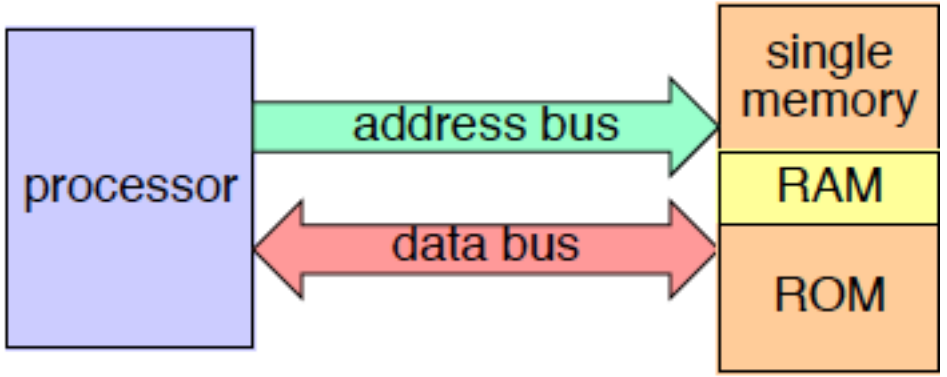
Can be treated in two general ways

two completely separate memory systems, each has its own data and address bus – this is the **Harvard** Architecture



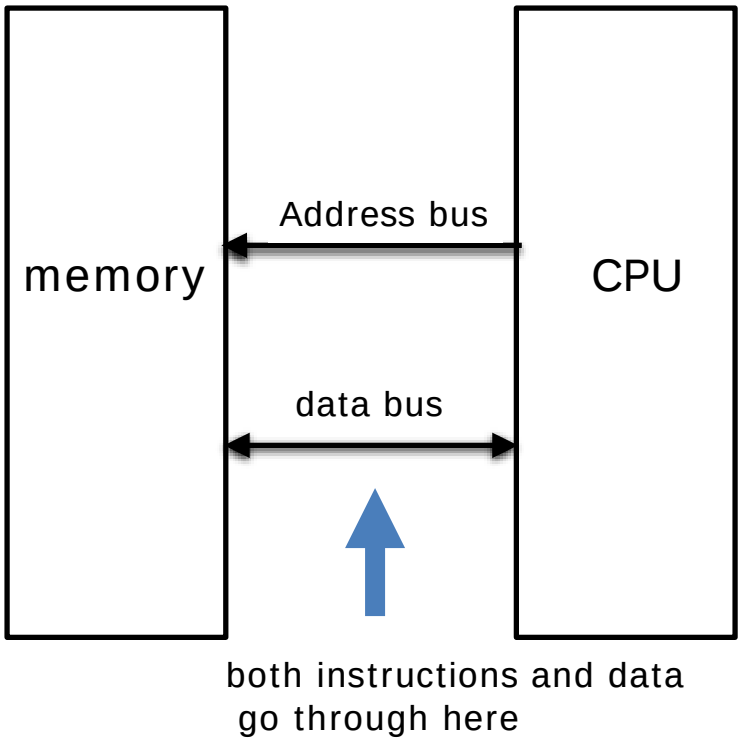
single memory system

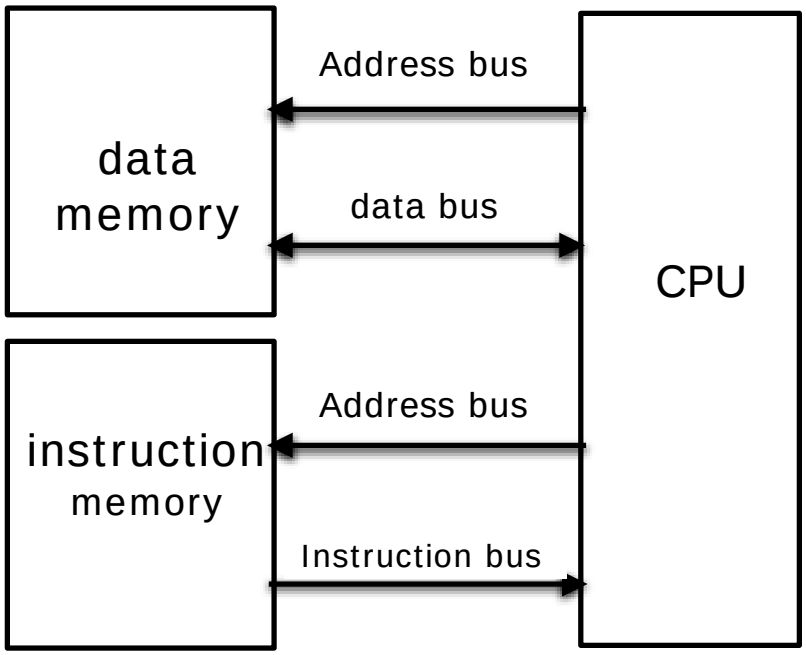
– **von Neumann** (Princeton) architecture



Von Neumann and Harvard Architecture

- Represent two different ways of exchanging data between CPU and memory





Von Neumann Architecture

Harvard Architecture

Microcontroller vs Computer

Microcontroller is a computer but …

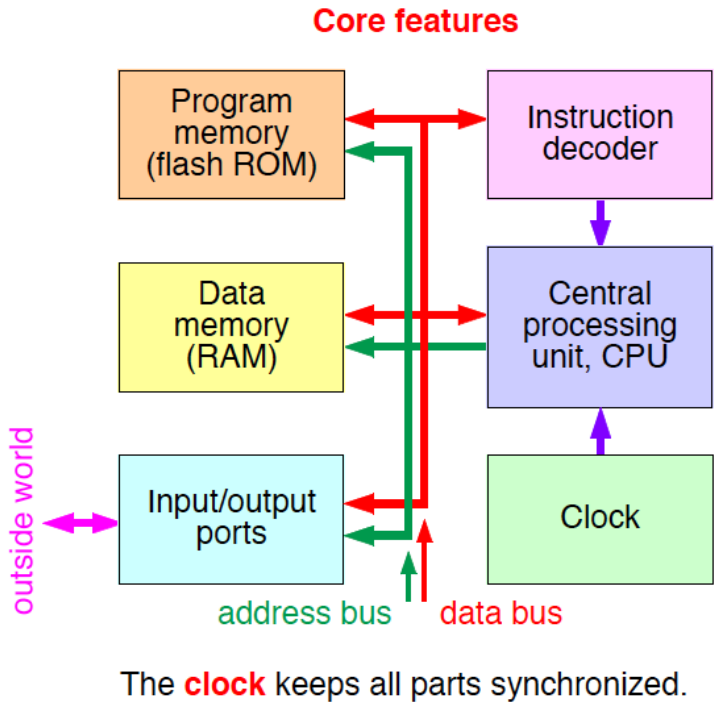
Microcontroller

- Primary purpose is to control a system
- Input = analog/digital signals
- Output = analog/digital signals
- Event or Command driven
  - Response is due to either a change in the input or data entered by person, autonomous systems rely on only sensors
- Reactive
  - physical world in the loop
- As fast as needed

General Purpose Computer

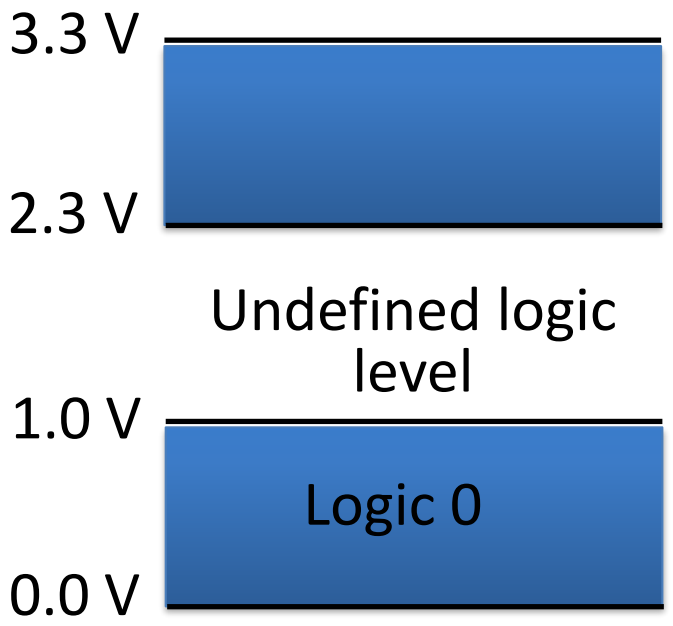
- Primary purpose is to perform calculations and run programs
- Input = keyboard, game console, etc.
- Output = display, monitor
- Command driven
  - Response is due to data entered by someone
- Reactive
  - human-in-the-loop
- As fast as possible

DIGITAL IN/OUT



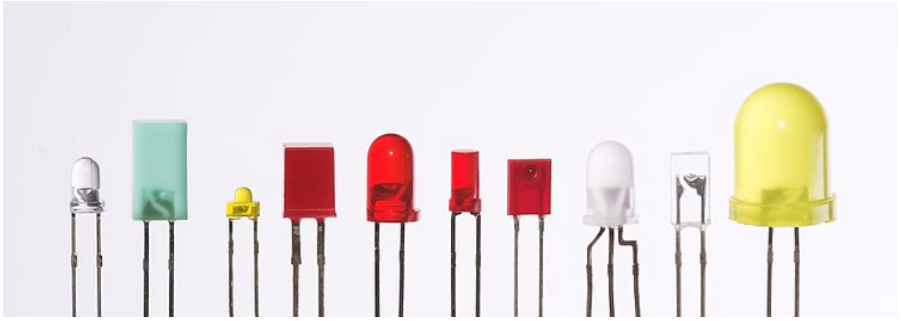
Digital Input and Output

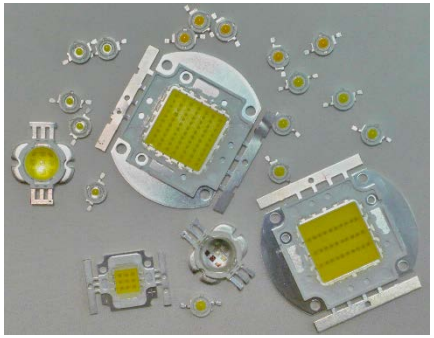
0V	3.3V
Open	Closed
Off	On
Low	High
Clear	Set
logic 0	logic 1
False	True



•

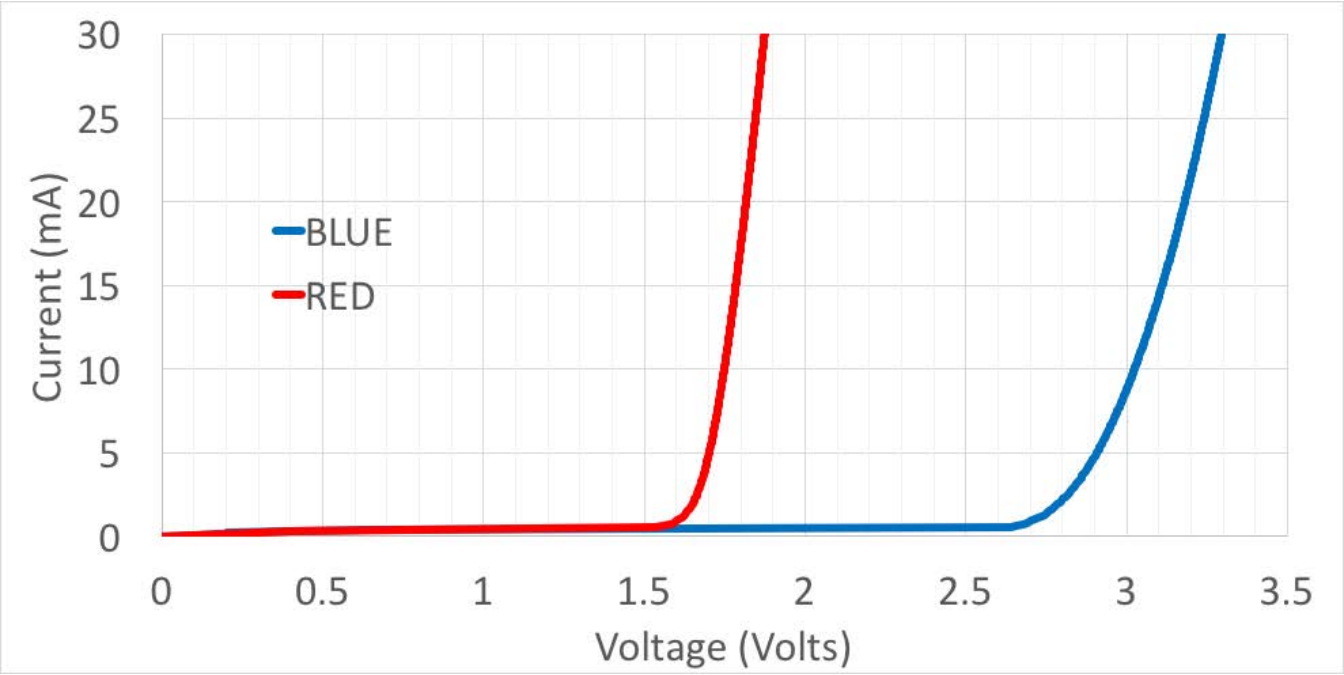
LEDs





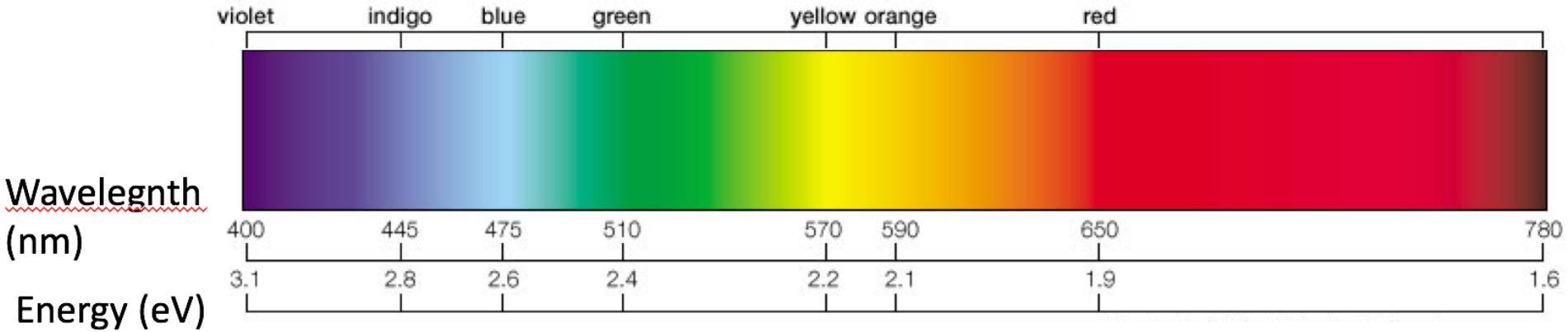
- Range from low power : Ideal as an indicator i.e. On/Off, or Run/Fault, or PANIC : fridge empty
- To high brightness : Ideal for lighting, displays
- The semiconductor material it's made from determines the color
- Most of the LED is in packaging, and the colored done is just to filter out unwanted light and focus the emission.

I-V characteristic

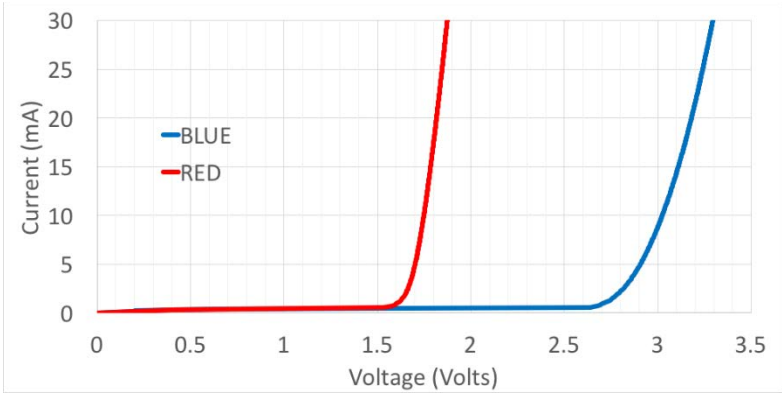


- Since it's a diode, we expect a sharp turn on at a 'turn-on' voltage
- Above this voltage, significant current flows
- The diode is not ideal, there is still a slope – why?

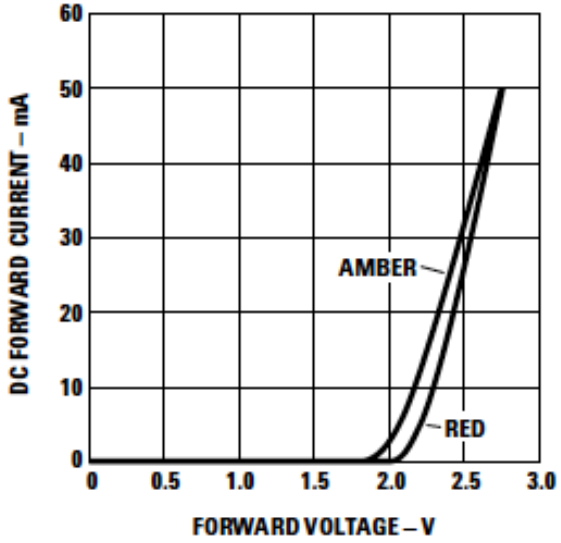
Wavelength, Energy, Voltage



- Your eye can see colors ranging from 400nm (dark violet) to 750nm dark red
- Blue light is higher energy than red
  - needs more voltage



Connecting LEDs



LEDs are diodes – two terminals

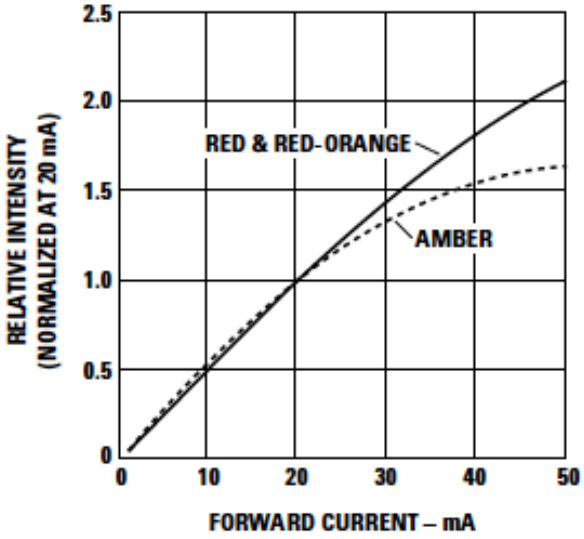
anode and cathode.

conduct current in one direction only.

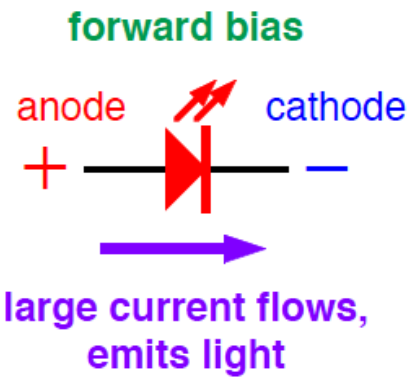
When conducting current

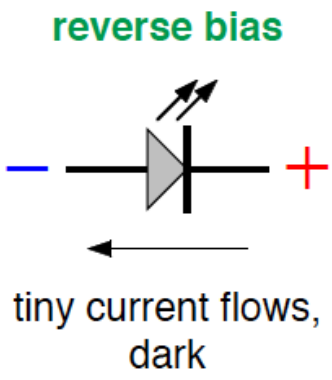
they emit light.

the diode is “forward” biased.



Increasing “forward” current increases light output





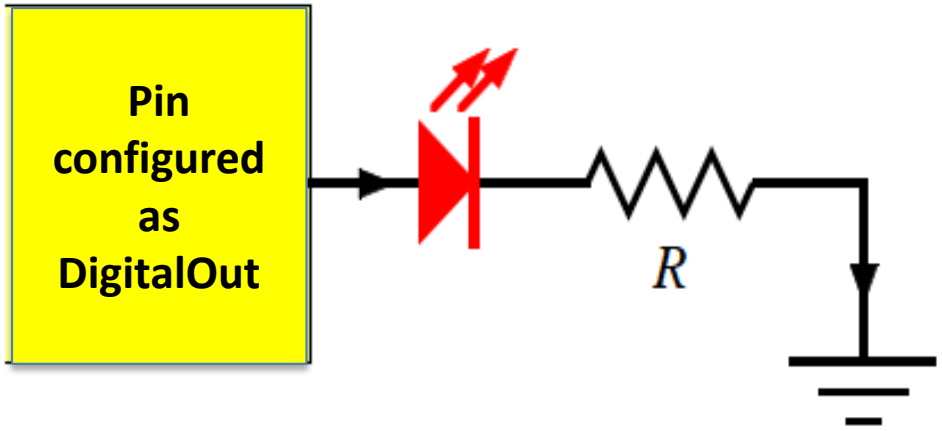
Remember that LEDs are **diodes**, so we must connect them the correct way round! This can be done in two ways.

1. Suppose that we want the LED to illuminate when the logical value of the output is high (1) — **active high**.

   Thus the LED should light when the pin is at V_{DD} (supply, positive, logic 1) but not when the pin is at V_{SS} (ground, negative, logic 0).

   We therefore connect the LED between the pin and V_{SS}.

   Conventional current flows from the pin to V_{SS} with this connection.



The resistor R is to limit the current — we'll calculate its value shortly.

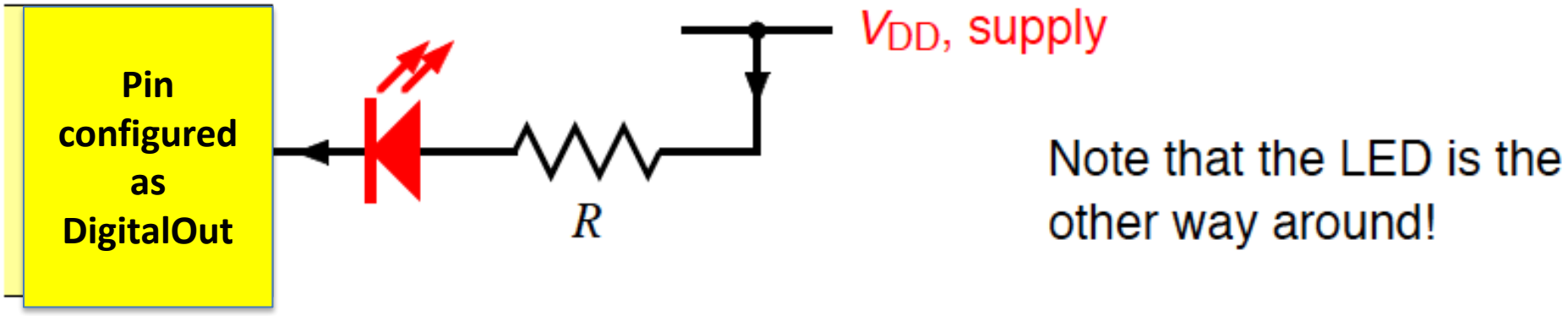
It might seem obvious that you would always want to connect an LED like this but there is a second, complementary way in which this can be done.

2. Suppose that we want the LED to illuminate when the logical value of the output is low (0) — **active low**.

   Thus the LED should light when the pin is at V_{SS} (ground, negative, logic 0) but not when the pin is at V_{DD} (supply, positive, logic 1).

   We therefore connect the LED between the pin and V_{DD}.

   Conventional current flows from V_{DD} to the pin with this connection.

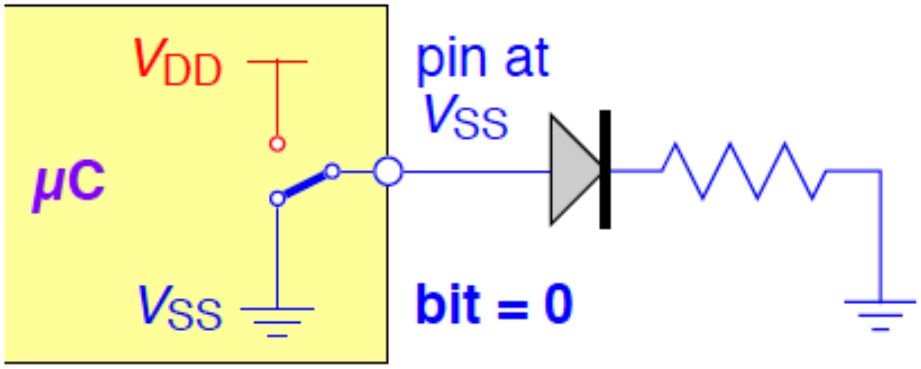


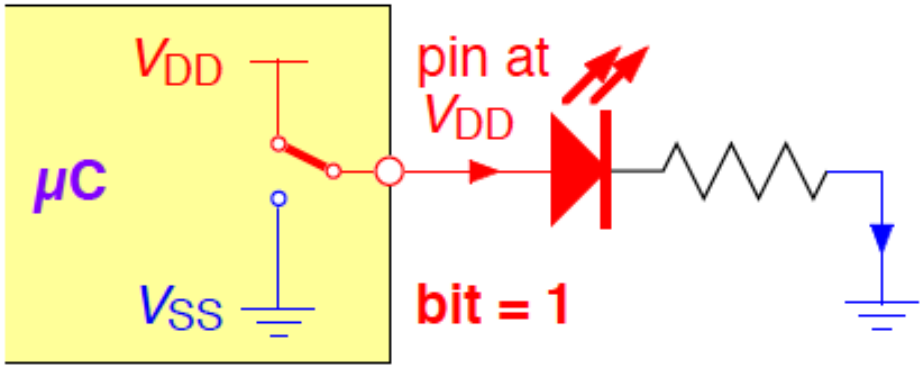
Note that the LED is the other way around!

Remember how the output works for each bit of the port:

logical 0 causes the μC to drive the pin low, to V_{SS}, ground, negative

logical 1 causes the μC to drive the pin high, to V_{DD}, supply, positive





Both ends of the LED are at V_{SS} so no current flows

The anode of the LED is at V_{DD} and the cathode is at V_{SS}.

A forward current can flow (in the direction of the arrows) so the LED illuminates

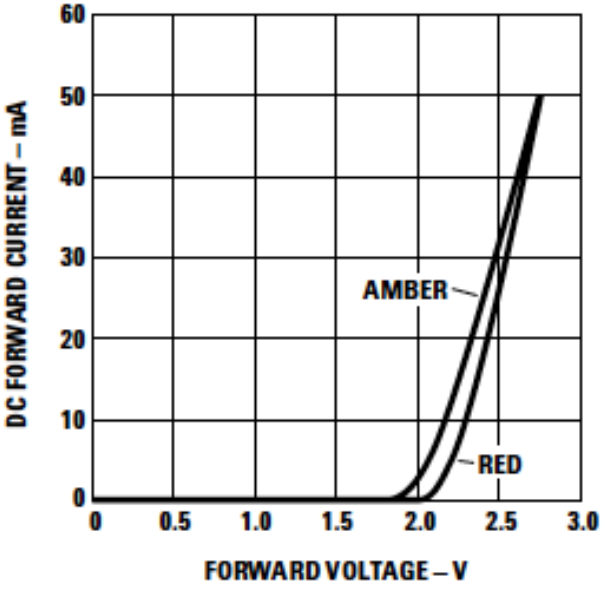
(What would happen if the LED were connected the other way around?)

May wish to control brightness or reduce current

- In this case, add series resistance between LED and mbed.

- Assume that to increase battery lifetime, want to limit current through red LED connected to mbed DigitalOut pin to 5 mA.

What value of resistor should be used ?



Say V from MCU is 3.3V

V drop across diode is 2.2V

V remaining is 3.3-2.2 = 1.1V

V=IR so

1.1 = 0.005 x R

1.1/0.005 = **220 Ohms**

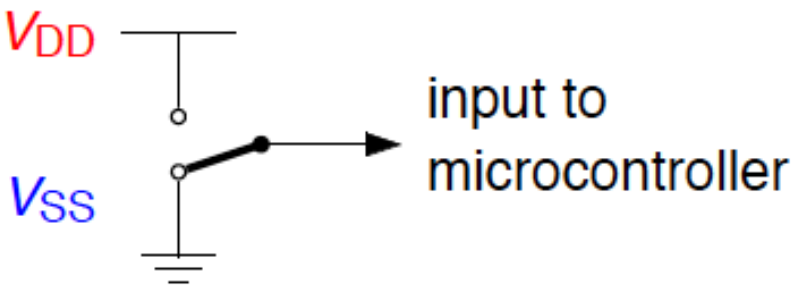
Digital In

- Connecting switches to the mbed

Suppose that we want to connect a simple on–off switch or pushbutton as an input to a μC. How should this be done?

The input should be either:

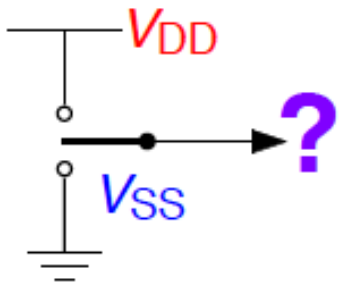
- V_{DD} (supply, positive) for logic 1
- V_{SS} (ground, negative) for logic 0



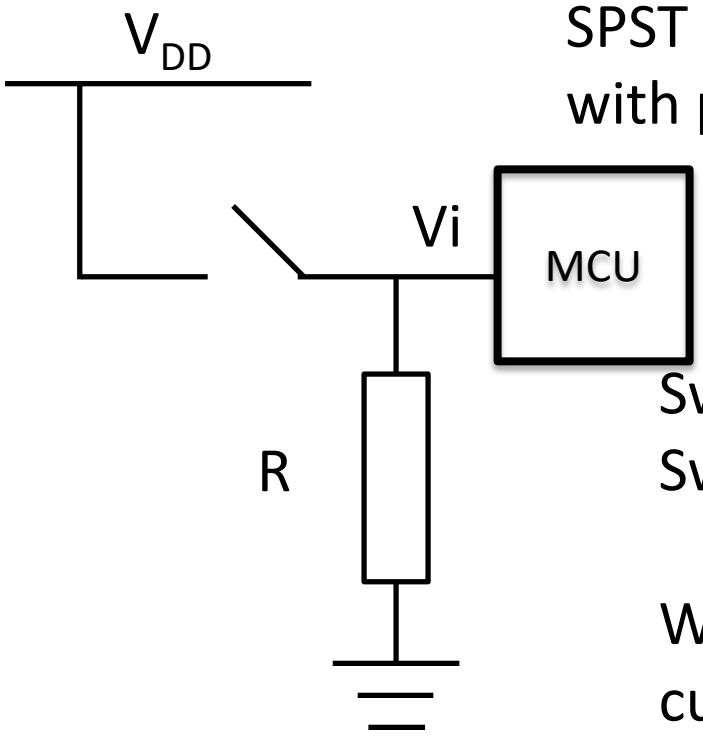
This may be the obvious approach:

However, this is not used in practice, for several reasons:

- it needs an expensive switch (3 terminals instead of 2, not just a simple pushbutton!)
- it needs 2 or 3 wires
- **what happens while the switch is being operated?**



What is actually done in practice



SPST (Single Pole Single Throw) Switch with pull down resistor

Switch open – Vi is connected to ground
Switch closed – Vi is connected to Vss

When switch is closed, there is a wasted current flowing through resistor R, so resistor is usually large

On the mbed, the resistor is available within the microcontroller – the default when a pin is configured as DigitalIn, is the pull-down configuration, so the default of an input pin is that it will be at 0 V. (this can be changed via the "mode" function of the DigitalIn command).

How much current is wasted?

A typical supply voltage is 3 V and pullup resistors are often 10 kΩ. How much current is wasted through the pullup when the button is pressed? For comparison, the microcontroller draws about 4 mA in normal operation.

Ohm's law tells us that current I, voltage V and resistance R are related by

$$I = \frac{V}{R} = \frac{3\,\text{V}}{10\,\text{k}\Omega} = \frac{3\,\text{V}}{10^4\,\Omega} = 3 \times 10^{-4}\,\text{A} = 0.3\,\text{mA} = 300\,\mu\text{A}$$

The normal operating current of the microcontroller is about 4 mA, so the pullup draws nearly a tenth of this. This is why larger resistors are now used.

Of course, the current flows only while the button is pressed.

DEMO – DIGITAL IN / OUT

```
/*Program to flash 1 of 2 LEDS, depending on the state of a 2 way switch

*/
#include "mbed.h"
DigitalOut redled(D3);
DigitalOut greenled(D4);
DigitalIn switchinput(D5);
int main() {
while(1) {
      if(switchinput==1) {     //test value of switch input

      //execute following block of code if switch input is 1
            greenled = 0;      //green LED is off

            redled = 1;        //flash red LED
            wait(1.0);

            redled = 0;
            wait(1.0);

      }                        //end of if

      else {

      //execute this block of code if switchinput is 0
      redled = 0;              //redled is off

      greenled = 1;            //flash green LED

      wait(1.0);
      greenled = 0;
      wait(1.0);

      }                        //end of else

}                              //end of while(1)

}                              //end of main
```

Summary

- Architecture
- Digital I/O
- What will we study in next lecture.