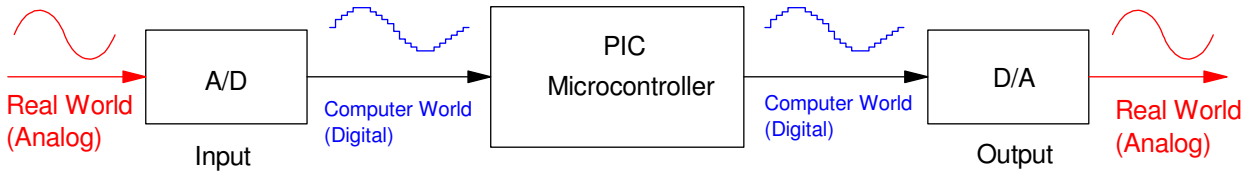
# A/D Conversion (Analog Inputs)

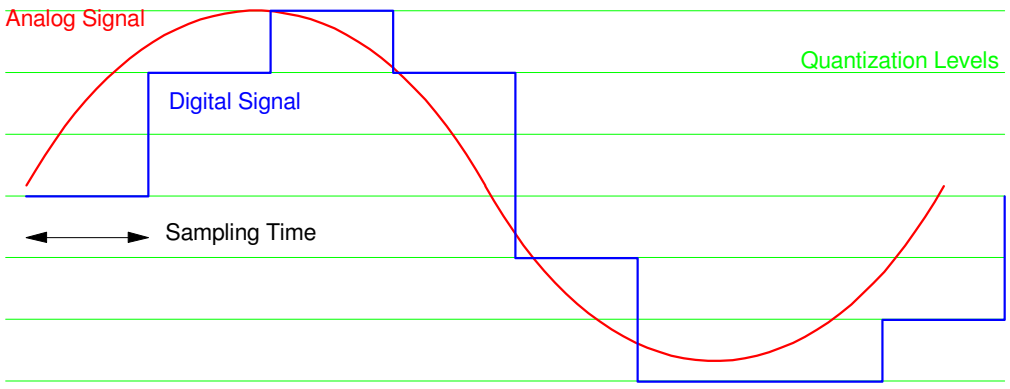
Sadly, the world isn't digital. Temperature, speed, position, flow, time, magnetic field strength are all analog signals: i.e. there are many shades of grey. The A/D converter is a component in your processor that lets you read voltages over the range of (0V .. 5V).




A/D converters convert analog signals to digital (computer input).
D/A converters convert digital signals to analog (computer output)

First, some definitions:

- A/D: Analog to Digital. Converts data from the real world (analog) to the computer world (digital).
- D/A: Digital to Analog. Converts data from the computer world (digital) to the analog world (analog).
- Sampling Rate: The number of samples per second.
- Quantization Level: The resolution of the A/D or D/A converter. How many mV corresponds to one count.
- Quantization Noise: The difference between the real signal (analog) and the digital signal (digital)




Definitions: Due to using integers, the analog signal (red) is slightly different from the digital signal (blue)

A/D converters allow a microcontroller to see what's happening in the real world. The A/D on the PIC processor can read voltages over the range of 0..5V. If you can convert what you want to measure (light, temperature, speed, position, etc.) to a voltage, a PIC can read it. That's actually really powerful: Digikey sells over 10,000 different sensors, many with a resistance or a voltage output. If you build a voltage divider to convert resistance to voltage, a PIC can read many of these sensors with the A/D converter.

This lecture focuses on the A/D converters and how to read analog signals.

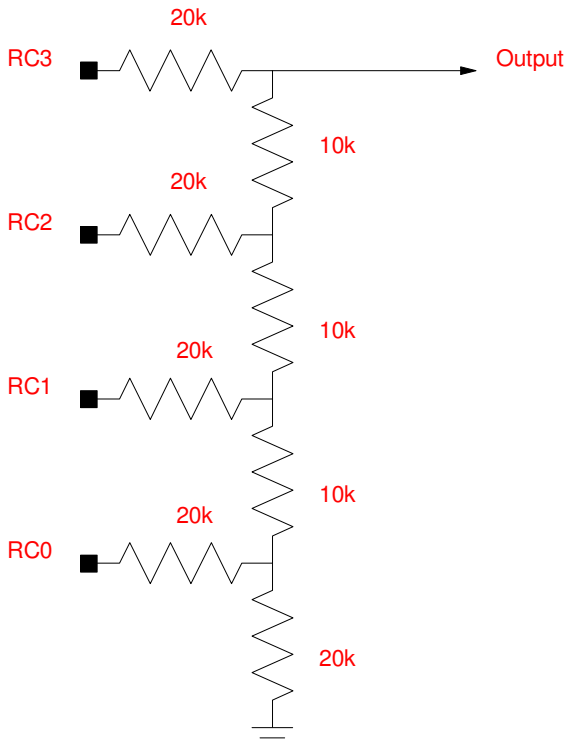
## D/A Converters

The heard of an A/D converter is actually a D/A converter. Likewise, in order to understand A/D converters, you have to understand D/A converters.

The simplest D/A converter is the R-2R ladder. The following circuit, for example, converts a 4-bit binary number to analog signal whose voltage is

$$V_o = \tfrac{1}{2}(RC3) + \tfrac{1}{4}(RC2) + \tfrac{1}{8}(RC1) + \tfrac{1}{16}(RC0)$$

(hint: use superposition and Thevenin equivalents to verify this).




If RC3:RC2:RC1:RC0 represents a binary number from 0..15 and the PIC outputs 0V/5V, the output voltage is

$$V_o = \left(\frac{\text{binary data}}{16}\right) \cdot 5V$$

In general, for an R-2R ladder, the output voltage for an N-state R-2R ladder with 0V/5V inputs is

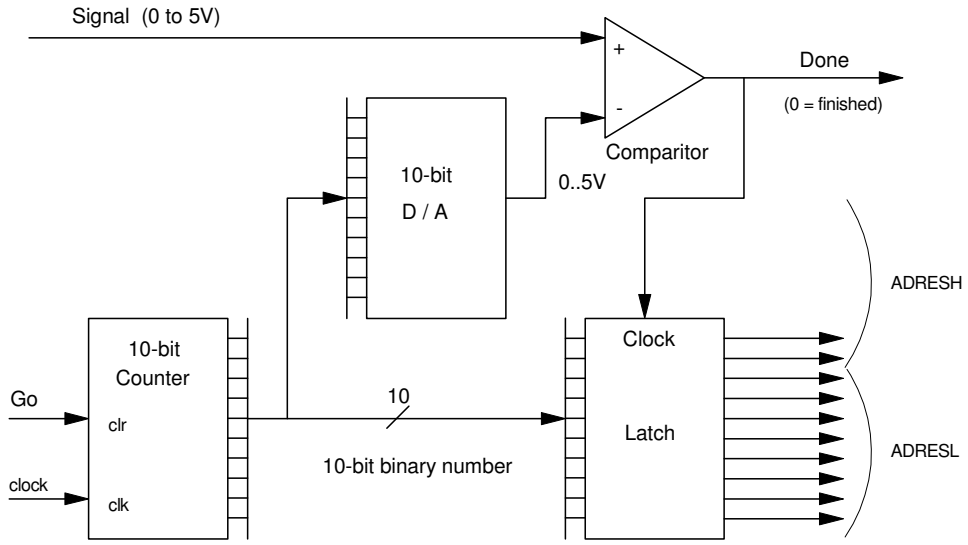
$$.V_o = \left(\frac{\text{binary data}}{2^N}\right) \cdot 5V$$

## A/D Converters

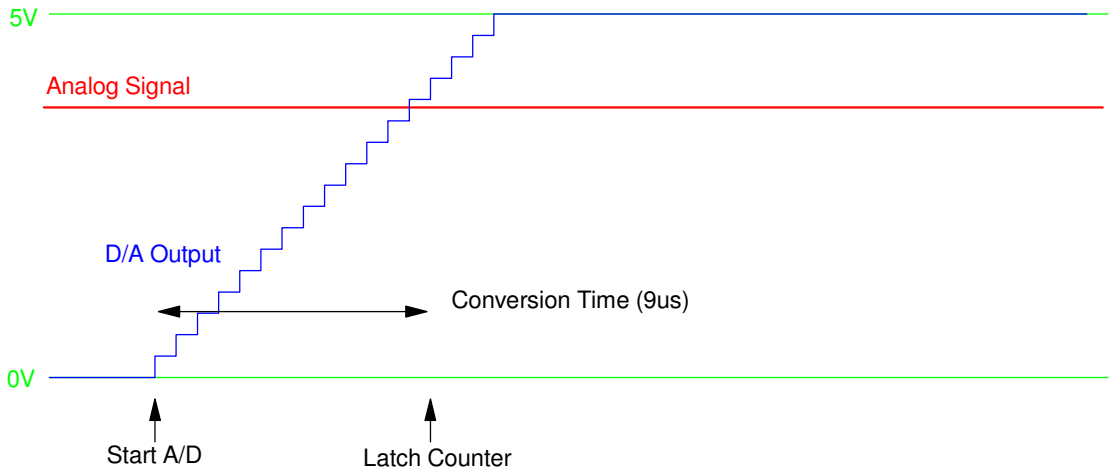
Once you understand D/A converters, you can understand an A/D converter.

The way an A/D converter works is this:

- First, an analog signal is sent (Signal in the figure below).
- Next, an counter starts counting from 0 up to 1023 (0x000 to 0xFFF for a 10-bit A/D like the one on your PIC chip)
- This count goes to a D/A converter, which converts the count to an analog voltage going from 0V (0x000) to 5V (0xFFF).
- A comparitor compares the analog signal to the D/A output. Once the D/A output exceeds the analog signal, a latch is triggered. This saves the present value of the counter.




Hardware for an A/D converter




Timing for an A/D conversion: The value of the counter when the D/A output exceeds the analog signal is the A/D conversion

Note: With this approach, the reading is linear with voltage:

$$Reading = \left(\frac{Voltage}{5}\right) \cdot 2^n$$

For a 10-bit A/D

$$Reading = \left(\frac{V}{409.6}\right)$$

It also takes time to do an A/D conversion. With the A/D on your PIC board, it's about 9us / conversion, giving a maximum sampling frequency of

$$\max(F_{sample}) = \left(\frac{1}{9us}\right) = 111kHz \quad .$$

# A/D Conversion on the PIC18F4626:

PORTA is can be used for analog or digital inputs. If you want to use PORTA, ADCON0, ADCON1, and TRISA need to be set up to tell the PIC chip how to use PORTA. The pins and bit assignments for an analog input follow:

| Address | Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0xFC0 | ADCON2 | ADFM | — | ACQT2 | ACQT1 | ACQT0 | ADCS2 | ADCS1 | ADCS0 |
| 0xFC1 | ADCON1 | — | — | VCFG1 | VCFG0 | PCFG3 | PCFG2 | PCFG1 | PCFG0 |
| 0xFC2 | ADCON0 | — | — | CHS3 | CHS2 | CHS1 | CHS0 | GODONE | ADON |
| 0xFC3 | ADRES | 16 bit register (0..65535) | | | | | | | |

**ADCON0:**

- CHS: Channel to convert: You must wait 14us if you change channels
  - 0000 = Channel 0 (RA0/AN0)
  - 0001 = Channel 1 (RA1/AN1)
  - 0010 = Channel 2 (RA2/AN2)
  - 0011 = Channel 3 (RA3/AN3)
  - 0100 = Channel 4 (RA5/AN4)
  - 0101 = Channel 5 (RE0/AN5)
  - 0110 = Channel 6 (RE1/AN6)
  - 0111 = Channel 7 (RE2/AN7)
  - 1000 = Channel 8 (RB2/AN8)
  - 1001 = Channel 9 (RB3/AN9)
  - 1010 = Channel 10 (RB1/AN10)
  - 1011 = Channel 11 (RB4/AN11)
  - 1100 = Channel 12 (RB0/AN12)
- ADON: 1 = turn on the A/D (and draw an additional 180uA)
- GODONE: Start the A/D conversion. Conversion is complete when bit GODONE = 0 (about 9us later)

**ADCON1**

bit 5 VCFG1: Voltage Reference Configuration bit (VREF- source)

- 1 = VREF- (AN2)
- 0 = VSS

bit 4 VCFG0: Voltage Reference Configuration bit (VREF+ source)

- 1 = VREF+ (AN3)
- 0 = VDD

PCFG3:PCFG0 determine whether certain pins are analog inputs (A) or binary I/O (D)

| PCFG3: PCFG0 | RB0 AN12 | RB4 AN11 | RB1 AN10 | RB3 AN9 | RB2 AN8 | RE2 AN7 | RE1 AN6 | RE0 AN5 | RA5 AN4 | RA3 AN3 | RA2 AN2 | RA1 AN1 | RA0 AN0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | A | A | A | A | A | A | A | A | A | A | A | A | A |
| 0001 | A | A | A | A | A | A | A | A | A | A | A | A | A |
| 0010 | A | A | A | A | A | A | A | A | A | A | A | A | A |
| 0011 | D | A | A | A | A | A | A | A | A | A | A | A | A |
| 0100 | D | D | A | A | A | A | A | A | A | A | A | A | A |
| 0101 | D | D | D | A | A | A | A | A | A | A | A | A | A |
| 0110 | D | D | D | D | A | A | A | A | A | A | A | A | A |
| 0111 | D | D | D | D | D | A | A | A | A | A | A | A | A |
| 1000 | D | D | D | D | D | D | A | A | A | A | A | A | A |
| 1001 | D | D | D | D | D | D | D | A | A | A | A | A | A |
| 1010 | D | D | D | D | D | D | D | D | A | A | A | A | A |
| 1011 | D | D | D | D | D | D | D | D | D | A | A | A | A |
| 1100 | D | D | D | D | D | D | D | D | D | D | A | A | A |
| 1101 | D | D | D | D | D | D | D | D | D | D | D | A | A |
| 1110 | D | D | D | D | D | D | D | D | D | D | D | D | A |
| 1111 | D | D | D | D | D | D | D | D | D | D | D | D | D |

**ADCON2**

ADFM: A/D Result Format Select bit

- 1 = Right justified
- 0 = Left justified

| Result from an A/D Conversion<br>10-bit Result = abcdefghij | | |
|---|---|---|
| ADFM | ADRESH | ADRESL |
| 0 | abcd efgh | ij00 0000 |
| 1 | 0000 00ab | cdef ghij |

bit 6 Unimplemented: Read as ‘0’

bit 5-3 ACQT2:ACQT0: A/D Acquisition Time Select bits

- 110: Automatically restart the A/D conversion every 16th clock
- 000: Manual operation of the A/D (user must set GODONE to start conversions)

bit 2-0 ADCS2:ADCS0: A/D Conversion Clock Select bits

- 101 = FOSC/16 (use with a 20MHz crystal)

**Example:** Set up the A/D so that

- PORTA/E are analog inputs, PORTB/C/D are binary
- The conversion will be right justified (ADFM = 1)
- A 20MHz crystal is used. (ADCS = 10: $F_{OSC}$ / 32)

**Solution:**

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ADCON2 | ADFM | — | ACQT2 | ACQT1 | ACQT0 | ADCS2 | ADCS1 | ADCS0 |
| | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ADCON1 | — | — | VCFG1 | VCFG0 | PCFG3 | PCFG2 | PCFG1 | PCFG0 |
| | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| ADCON0 | — | — | CHS3 | CHS2 | CHS1 | CHS0 | GODONE | ADON |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The code you need to include to initialize the A/D converter is thus:

```
// Turn on the A/D converter
    TRISA  = 0xFF;
    TRISE  = 0x0F;
    ADCON2 = 0x85;
    ADCON1 = 0x07;
    ADCON0 = 0x01;
```

The code you need to read the analog input on pin c is

```
unsigned int A2D_Read(unsigned char c)
{
   unsigned int result;
   unsigned char i;
   c = c & 0x0F;
   ADCON0 = (c << 2) + 0x01;              // set Channel Select
   for (i=0; i<3; i++);                   // wait 2.4us (approx)
   GODONE = 1;                            // start the A/D conversion
   while(GODONE);                         // wait until complete (approx 8us)
   return(ADRES);
}
```

Note:

- A2D_Read(0) Read the voltage on RA0, return 0 (0V) to 1023 (5V)
- A2D_Read(1) Read the voltage on RA1, return 0 (0V) to 1023 (5V)
- A2D_Read(2) Read the voltage on RA2, return 0 (0V) to 1023 (5V)
- A2D_Read(3) Read the voltage on RA3, return 0 (0V) to 1023 (5V)
- etc Read the voltage on RA0

# Fun with A/D Converters

The A/D input allows you to input numbers (0 to 1023) into the PIC processor with a potentiometer. This illustrates some of the things this allows you to do:

- Electronic Trombone: Set the frequency using the analog input. Play a note when you press RB0.
- LED Flashlight: Vary the brightness of a NeoPixel using the potentiometer from 0% to 100% on in 255 steps.
- LED Flashlight (take 2): Vary the color of the NeoPixel using the potentiometer
- Stepper Motor Position Control (Telerobotics): Have a stepper motor follow the position of the potentiometer from 0 steps (A/D = 0) to 200 steps (A/D = 1023).
  - This also makes the stepper motor a temperature indicator if the input voltage is temperature
  - Or a light indicator
- Stepper Motor Speed Controller: Have the speed of the stepper motor set by the analog input
- Multi-Meter. Turn your PIC into a volt / ohm / light / temperature meter.

## Electronic Trombone:

Requirement: Play notes ranging from 100Hz to 200Hz on pin RC0 as you press RB0. The frequency is continuously adjustable using the analog input (potentiometer on your PIC board).

From previous code, the following routine plays notes C2 to C2

```
while(1) {
 if (PORTB) RC0 = !RC0;
 if (RB0) for(i=0; i<4771; i++);
 if (RB1) for(i=0; i<4250; i++);
 if (RB2) for(i=0; i<3786; i++);
 if (RB3) for(i=0; i<3574; i++);
 if (RB4) for(i=0; i<3184; i++);
 if (RB5) for(i=0; i<2837; i++);
 if (RB6) for(i=0; i<2527; i++);
 if (RB7) for(i=0; i<2385; i++);
 }
```

If you replace the hard-coded numbers with a number based upon the A/D reading, you can vary the frequency on the fly. In C, the executing time depends upon the code. To get an accurate measure, start with code close to what we'll need in the end:

```
while(1) {
   A2D = A2D_Read(0);
   N = 2000 - 0.5678*A2D;
   if(PORTB)  RC0 = !RC0;
   for(i=0; i<N; i++);
   }
```

Experimentally, the extremes produce:

- A/D = 0 N = 2000 f = 116.04Hz
- A/D = 1023 N = 1419 f = 154.39Hz

From this,

Hz = 248 - 0.0660N

or

- 100Hz N = 2243 A/D = 0
- 200Hz N = 728 A/D = 1023

giving the function

This results in

- 0V A/D = 0 f = 103.77 Hz
- 5V A/D = 1023 f = 278.82 Hz

A little more adjustment would get you from 100Hz to 200Hz as you adjust the potentiometer.

## LED Flashlight: Brightness Control

Previous code allowed us to drive a NeoPixel using a PIC processor. The global variables RED, GREEN, and BLUE set the brightness of the NeoPixel from 000 (off or 0mA) to 255 (100% on or 20mA).

To change the brightness of the NeoPixels using the A/D converter, use the following code.

```
while(1) {
   A2D   =  A2D_Read(0);
   X       =  A2D/4;
   LCD_Move(1,0);  LCD_Out(X, 0, 3);

   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);
   NeoPixel_Display(X, X, X);

   Wait(1);
}
```

## LED Flashlight: Hue Control

Instead of making all colors the same intensity, producing white light, update each color one at a time. As you hold down one of the buttons, the brightness of that color changes according do the A/D input:

- RB2 Blue
- RB1: Green
- RB0: Red

One version of the main routine to do this:

```
while(1) {

   A2D   =  A2D_Read(0);

   X = A2D / 4;
   if (RB0) RED   = X;
   if (RB1) GREEN = X;
   if (RB2) BLUE  = X;

   LCD_Move(0,10);  LCD_Out(X, 0, 3);
   LCD_Move(1, 0);  LCD_Out(RED, 0, 3);
   LCD_Move(1, 5);  LCD_Out(GREEN, 0, 3);
   LCD_Move(1,10);  LCD_Out(BLUE, 0, 3);

   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);
   NeoPixel_Display(RED, GREEN, BLUE);

   Wait(5);
}
```

## Stepper Motor: Position Control (Telerobotics)

Connect the potentiometer to your arm so that as you move, the potentiometer voltages changes with you. Have the stepper motor follow the potentiometer as

- 0V = 0 steps
- 5V = 200 steps
- Proportional in-between

```
while(1) {
   A2D = A2D_Read(0);
   REF = A2D * 0.1955;

   if (STEP < REF) STEP = STEP + 1;
   if (STEP > REF) STEP = STEP - 1;

   PORTC = TABLE[STEP % 4];

   LCD_Move(0,8);  LCD_Out(REF, 0);
   LCD_Move(1,8);  LCD_Out(STEP, 0);

   Wait_ms(20);

   }
```

## Stepper Motor: Light Sensor

Make the stepper motor indicate the light level as

- 1 Lux 0 steps
- 100 Lux 200 steps

This is the same as the previous solution:

- First, convert light to voltage.
- Once it's a voltage, read the voltage with the A/D input and use that to control the stepper position.

# Multi-Meter

Turn your PIC board into

- A volt meter
- An Ohm meter
- A light sensor
- A temperature sensor

## Volt Meter:

The A/D reading is proportional to voltage

- 0 = 0.00V
- 1023 = 5.00V

The calibration function is then

Volt = 0.0047776 * A2D

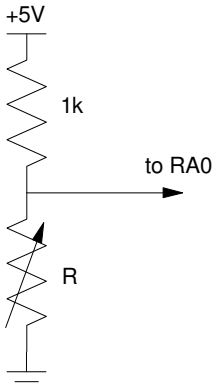
If you want to display this to 2 decimal places, scale this by 100 (so 100 means 1.00 Volts)

Code:

```
while(1) {

   A2D = A2D_Read(0);

   VOLT = 0.488 * A2D;

   LCD_Move(1,8);  LCD_Out(VOLT, 5, 2);

   }
```

## Ohm Meter:

You can convert resistance to voltage using a voltage divider. Assuming a 1k resistor



$$V = \left(\frac{R}{R+1000}\right)5$$

or the A/D reading will be

$$A/D = \left(\frac{R}{R+1000}\right)1023$$

Solving backwards, you can compute the resistance given the A/D reading

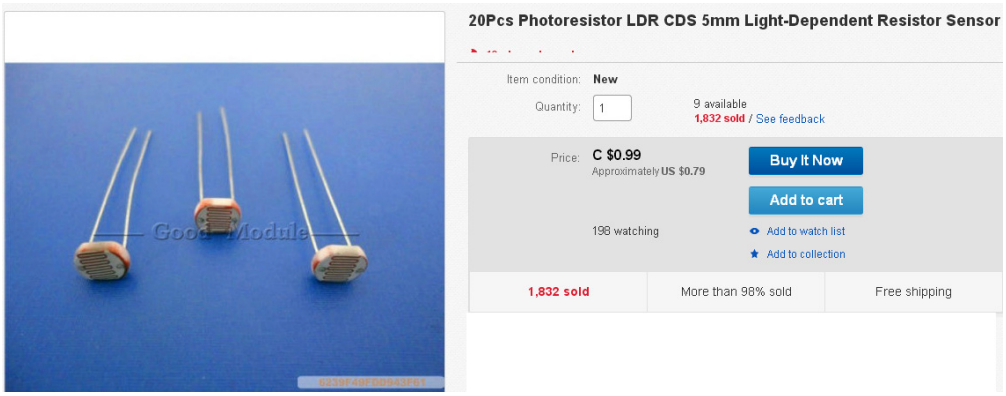
$$R = \left(\frac{A/D}{1023-A/D}\right)1000\Omega$$

Code:

```
while(1) {

   A2D = A2D_Read(0);
   OHM = 1000.0 * (A2D / (1023.0 - A2D) );

   LCD_Move(1,8);  LCD_Out(OHM, 5, 0);

   Wait_ms(10);

 }
```

## Light Meter:

The light sensor in your lab kit has a light-dependent resistor:




with a light - resistance relationship of

$$R \approx \frac{100{,}000}{Lux}$$

Then

$$Lux = \frac{100{,}000}{R}$$

Substituting for R from the previous sensor

$$Lux = \frac{100{,}000}{\left(\frac{A/D}{1023-A/D}\right)1000}$$

$$Lux = \frac{(1023-A/D)}{A/D} \cdot 100$$

Code:

```
while(1) {

   A2D = A2D_Read(0);

   LUX = ( (1023.0 - A2D) / A2D ) * 100;

   LCD_Move(1,0);  LCD_Out(A2D, 5, 0);
   LCD_Move(1,8);  LCD_Out(VOLT, 5, 2);

   Wait_ms(10);

 }
```

## Temperature Sensor

Also in your lab kits is a temperature sensor:



The resistance / voltage relationship is

$$R = 1000 \cdot \exp\left(\frac{3930}{T+273} - \frac{3930}{298}\right)\ \Omega$$

where T is the temperature in degrees C. Solving for T

$$T = \left(\frac{3930}{\ln\left(\frac{R}{1000}\right)+\frac{3930}{298}}\right) - 273 \quad \text{Celsius}$$

Substituting for R

$$T = \left(\frac{3930}{\ln\left(\frac{\left(\frac{A/D}{1023-A/D}\right)1000}{1000}\right)+\frac{3930}{298}}\right) - 273 = \left(\frac{3930}{\ln\left(\frac{A/D}{1023-A/D}\right)+\frac{3930}{298}}\right) - 273$$

Code: Include Math.h for the log function (note: in C, log() is base e, log10() is base 10)

```
#include   <pic18.h>
#include    <math.h>
```

The main routine is then (multiplying T by 10x so you can display temperature to one decimal point)

```
while(1) {

   A2D = A2D_Read(0);
   CELSIUS = 39300. / ( log( A2D / (1023. - A2D) ) + 13.1879 ) - 2730;

   LCD_Move(1,8);  LCD_Out(CELSIUS, 5, 1);
   }
```