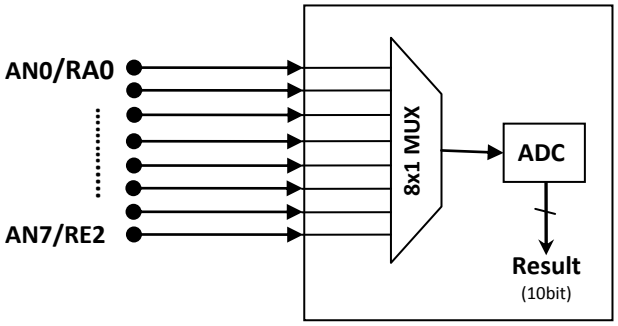
ADC module

ADC is stands for **A**nalog to **D**igital **C**onverter. It is used to convert any analog signal to digital data so that it can be stored and manipulated digitally. Digital voltmeter is a good example it simply takes analog reading (Voltage) and converts it to digital using ADC then by making simple calculations on this digitized value we will have a digital reading corresponds to the original analog, now we can store it, display it on a 7-segment or LCD, send it to PC as we will see later and the most important we can modify and process this digital data. And same as digital voltmeter procedure we can handle any analog input such as temperature, pressure and so on…

PIC16F877A has 8×10bit multiplexed ADC channels (AN0-AN7) mapped to port E and port **A** except RA4. Next figure shows an abstraction view for ADC module.



Analog input pins:
RA0 RA1 RA2 RA3 RA5 RE0 RE1 RE2

As you note from previous figure the result of conversion is 10 bit width and this means that the result of conversion is a value between 0 and 2^{10} -1 or [0,1023]. For example if we use 5 volt as reference voltage then analog input should be in TTL level (ranges from 0 and 5 volt) and in this case **0** volt analog corresponds to **0** digital in result and **5** volt analog corresponds to **1023** digital in result. In real application analog input is unknown and at the same time it is our target. The procedure for calculating this value is straightforward. Firstly, make a conversion to get a digital value that corresponds to analog input. Secondly, make a reverse calculation for the unknown analog voltage as follows (**note that the default value for V_{ref} is V_{DD} voltage at pin 11 or 32, in general it is 5v**):

$$\begin{array}{ccc} V_{ref} & \rightarrow & 1023 \\ Analog_{unknown} & \rightarrow & Digital_{Known} \end{array}$$

From above expression:

$$Analog_{unknown} = \frac{V_{ref}}{1023} \times Digital_{Known}$$

NOTE: PIC C provides us with two choices for ADC manipulation either 8-bit (ADC=8) or 10-bit (ADC=10). Above equation uses 10-bit, if we use 8-bit then we must divide by 255 (2^8-1) rather than 1023 (2^{10}-1).

To use ADC module the following steps must be applied:

Steps:	Corresponding PIC C code (Examples):
1. **Configure pins (analog or digital).**	setup_adc_ports(ALL_ANALOG);
2. **Configure conversion clock.**	setup_adc(ADC_CLOCK_INTERNAL);
3. **Select channel.**	set_adc_channel(0);
4. **Read conversion result.**	DigX = read_adc();//now multiply DigX with Vref/1023 or Vref/255.

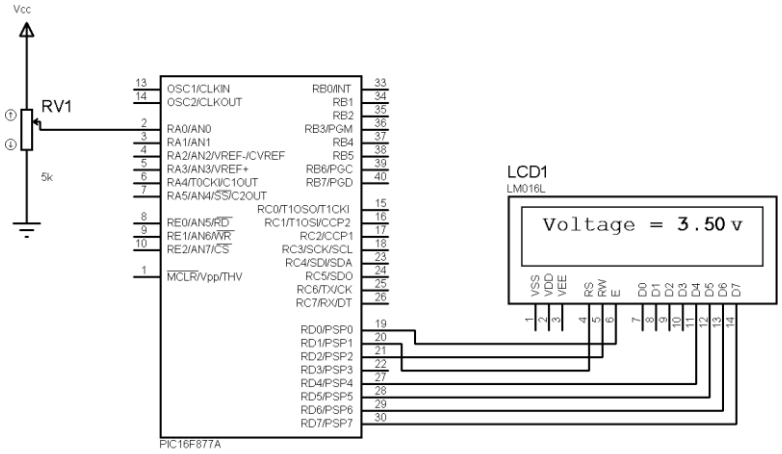
Once you configure the ADC (first 3 steps) you can read the converted value.
The first 3 steps can be written one time in the main() and before while(1).
You can pass other parameters for the PIC C code shown above, we will take some of them using examples.

Two examples will be taken: first, building a digital voltmeter. Second, measuring temperature and displaying it on LCD.

Example 1: TTL digital voltmeter**.**

Main goal: How to deal with ADC module (10bit resolution)**.**

DESCRIPTION: reads a voltage ranges from 0 to 5 volt and display it on LCD.



As shown above a variable resistor is used to choose a voltage from 0 to 5v. Code is shown below.

```
#include <16F877A.h>
#DEVICE ADC=10        //  return 10 bit(FULL RESOLUTION) from ADC, Will be explained later.
#FUSES XT             //  Crystal osc <= 4mhz.
#FUSES NOWDT          //  No Watch Dog Timer.
#FUSES NOPROTECT      //  Code not protected from reading.
#USE DELAY(CLOCK = 4000000)

#include <lcd.c>

void main()
{
   int16 digitalValue;  //16 bit integer, to store the ADC result (10bit).
   float voltage;

   setup_adc_ports( ALL_ANALOG ); //All 8 pins are analog input. Vref is 5v.
   setup_adc( ADC_CLOCK_INTERNAL );//Use internal clock (TAD between 2u-6u Second)
   set_adc_channel( 0 ); //AN0 is the used here pin.

   lcd_init();
   lcd_gotoxy(1,1);
   printf(lcd_putc, "Digital voltmeter");
   delay_ms(1500);;


  while(TRUE)
  {
      digitalValue = read_adc();
      voltage = (float)digitalValue/204;  //ADC equation: digitalValue*5/1023

      printf(lcd_putc,"\fVoltage = %1.2f v", voltage);//(%1.2f): display a float with 1
                                                      //integer digit and 2 fractional.
      delay_ms(500);   //can be ignored
  }
}
```

Because ADC ports are configured to **ALL_ANALOG** we can use any pin from <AN0:AN7> as analog input, for other parameters show 16F877A.h header file. In above example we use AN0 and according to that we must read from channel 0 this is done using **set_adc_channel(0)** line. Suppose that **AN5** is used then instead of passing **0** we must pass **5** i.e.) **set_adc_channel(5)**. **T_{AD}** is the time required from ADC to digitize one bit (at minimum, it must be 1.6uS). By choosing **ADC_CLOCK_INTERNAL** T_{AD} automatically set between 2uS and 6uS.

Example 2: Temperature control system**.**

Main goals: More about ADC module, dealing with LM35 temperature sensor.

DESCRIPTION: Reads a temperature from 0 ℃ and above. If it is less than 22 ℃ send a high signal from RE0 to operate a heater. If the temperature is more than 27 ℃ send a high signal from RE1 to operate an air conditioner. Use ADC=8 and V_{ref} = 1v.

We will use LM35DZ temperature sensor, it is a 3 terminal sensor (V_{CC}, GND and O/P) and it can measure a wide temperature range (from 0 ℃ up to 100 ℃). The most important feature (for more features refer to datasheet) that its output is linear with 10mV / ℃. This means that if temperature is 1 ℃ then LM35's output is 10mV, so and simply:

$$\frac{1℃}{10mV} = \frac{Temperature}{sensor\ output\ voltage}$$

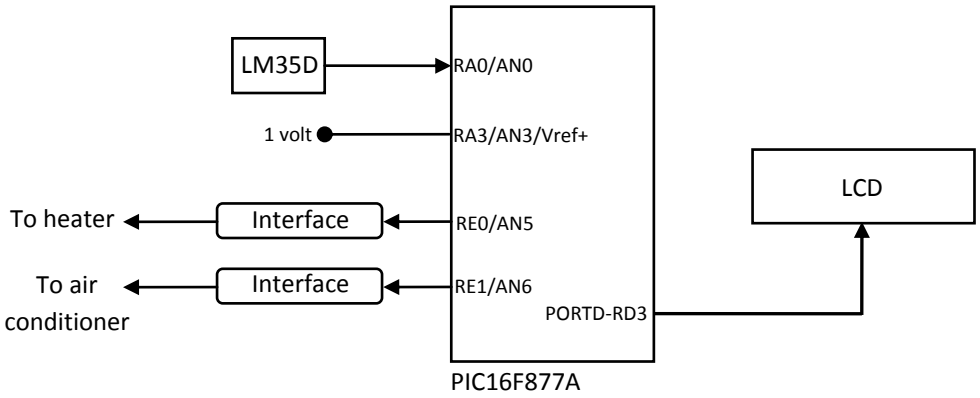
And this implies that:

$$Temperature = \frac{sensor\ output\ voltage}{10mV} = (sensor\ output\ voltage) \times 100$$

For example suppose that the output voltage is equal to 250mV then according to above equations the current temperature is the result of ($250 \times 10^{-3} \times 100$) and this equal to 25 ℃.

Honestly, there are many types to LM35 like LM35A, LM35C and our sensor LM35DZ, each one differ from other in temperature range (e.g. LM25C can measure from -40 ℃ to 110 ℃) and accuracy.

As you note from description RE0 and RE1 must set to digital output, LM35 is connected to AN0 and the reference voltage (pin A3) is set to 1v. Schematic is shown below.



Tip 1: Control system like that is called a **regulator system**. It is automatically maintains a parameter at (or near) a specified value; in our example we maintain temperature.

The interface between PIC (low voltage devices) and heater or air conditioner (high voltage devices) can be done using any device that makes isolation between them like relays (certainly with other elements).

Code is shown in the next page.

```
#include <16F877A.h>
#DEVICE ADC=8      //return 8-bit width. Don't forget to divide digitalValue over 255.
#FUSES XT, NOWDT, NOPROTECT
#USE DELAY(CLOCK = 4000000)

#include <LCD.c>

#DEFINE heater   PIN_E0
#DEFINE air_c    PIN_E1

void main()
{
   int8 digitlValue;   //Store the result of A/D conversion. 8bit is enough.
   float temperature;

   set_tris_d(0);
   output_d(0);

   //Initialize ADC module
   setup_adc_ports(AN0_AN1_VSS_VREF); //AN0 and AN1 are analog input pins.Vref at AN3.
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);

   lcd_init();
   lcd_gotoxy(1,1);
   lcd_putc("Temperature\nControl System");
   delay_ms(1500);

   while(1)
   {
      digitlValue = read_adc();
      temperature = (float)digitlValue / 255; //Apply ADC equ.: digitalValue*Vref/255
      Temperature = temperature * 100;         //Apply LM35 equation.
      printf(lcd_putc, "\fT = %2.2f", temperature);

      if(temperature > 27.0)
      {
        printf(lcd_putc, "\nHigh temperature!");
        output_high(air_c);   //turn ON air conditioner.
      }
      else if(temperature < 22.0)
      {
         printf(lcd_putc, "\nLow temperature!");
         output_high(heater); //turn ON heater.
      }
      else
      {
         printf(lcd_putc, "\nModerate T..re!");
         output_low(air_c);   //turn OFF air conditioner.
         output_low(heater);  //turn OFF heater.
      }
      delay_ms(500);
   }
}//end main
```

In (#**DEVICE**) directive we use **ADC=8** instead of **ADC=10** this means that **read_adc()** function will return 8-bit only from the converted result and a variable with int8 type is enough to store this value also instead of dividing by 1023(2^{10}-1) in ADC equation ($(V_{ref} \times digitalValue)/1023$) we must divide by 255 (2^8-1), the overall result is a light calculation but less accuracy.

Pin **RA3/AN3** can be used as analog input or reference voltage input. This can be determine according to the argument that passed to **setup_adc_ports()** function. In the above code it is used as V_{ref} . By setting V_{ref} to 1v the final result will be somewhat more accurate (in examples like this only). To show the difference you can convert V_{ref} to default V_{DD} (in general 5v) as ***example 1*** and change the ADC equation to ($(digitalValue \times 5)/255$). Next table shows some **setup_adc_ports()** parameters.

Some **setup_adc_ports**() parameters.

Parameter	Description
NO_ANALOG	All pins are digital.
ALL_ANALOG	All pins are analog. $V_{ref} = V_{DD}$.
AN0	AN0 is the only analog pin. $V_{ref} = V_{DD}$.
AN0_AN1_AN3	All of these pins are analog input. $V_{ref} = V_{DD}$.
AN0_AN1_AN2_AN4_VSS_VREF	All of these pins are analog. V_{ref} is set at AN3.

NOTE: V_{DD} or V_{CC} means the voltage that fed to PIC at pin 11 or 32; it is normally 5v but you can choose from 2v – 5.5v (according to PIC specification). So if you use V_{DD} as V_{ref} for example **ALL_ANALOG** or **AN0_AN1_AN3** then you must measure the voltage that supplied to PIC at pin 11 or 32 and change ADC equation according to it.

PIC16F877A hasn't a float point circuitry so all float calculations handled by PIC C using software and this implies to long execution time and more memory usage. In project section we introduced a method called integer coding scheme it can handle any float calculations using simple integer calculations.