Knowledge of baseline

The ADC Module

The purpose of the tutorial is to understand how to use the Analog-to-Digital converter module.

The acquisition of analogue signals by a digital device requires the conversion of a continuous quantity into a discontinuous one. This function is required because most sensors of physical quantities (temperature, humidity, weight, acceleration, light intensity, etc.) render voltage values that vary in certain ranges, while the processor has to work on digital data.
The conversion operation is carried out by circuits called **Analog-to-Digital converters** (ADCs) - *Analog to Digital Converter*), which can belong to many types of construction.

In microcontrollers, ADCs of the kind are commonly found with successive approximations, with a resolution ranging from 8 to 14 bits. By "resolution" we mean that the input voltage, which varies between a minimum and a maximum continuous, will be output by the conversion in binary form, with a certain number of digits; The higher this number, the more accurately the digital value approximates the analogue value.
A 10-bit ADC means that it has the ability to detect 1,024 (2^{10}) distinct analogue values; an 8-bit ADC yields 256 (2^8) values.

The input voltage range can vary between Vss and Vdd, so as not to damage the pin input circuit; if voltages outside this range (voltages below Vss, i.e. negative voltages or voltages above Vdd) are to be converted, external circuits (dividers, air conditioners, amplifiers) must be used to adjust the input voltage to the limits of the microcontroller.

1. **Pins used as ADC module inputs can NOT be used for other functions**.

2. In the PICs, the construction philosophy is the one that tends to the minimum consumption and this is given by the pins configured as analog. This means that **in the default of the POR the analog setting is the one pre-set for all the pins that support it**.

   If you need other functions shared on the pin, such as digital I/O, you need to disable the analog function from the program.

The conversion technique requires the charging of an internal capacitor, of which the discharge time is detected, with a comparator and a binary counter, evaluated with a clock that can be derived from the main oscillator. The content of the counter is the number that is returned when the conversion is complete. This moment is indicated by a special bit in the

ADC control.
The AD converter has a certain power consumption and, due to the policy of minimizing the energy absorbed by the chip, it is turned off by default and requires the setting of a bit to come into operation. Usually there are multiple input channels of the ADC module, which can be selected one at a time with the control register.
Only a few Baselines integrate the converter:

PIC	Pin	ADC Channels
10F220	**6**	**2**
10F222	**6**	**2**
12F510	**8**	**3**
16F506	**14**	**3**
16F526	**14**	**3**
16F527	**20**	**8**
16F570	**20**	**8**

As you can see, the more pins the device has, the more analog inputs can be made available.

The ADC module of the Baselines is 8-bit and has a simplified structure compared to the other families. In particular, there is no possibility of using an external reference voltage for the conversion, which **has as references the supply voltage Vss and Vdd**. It follows that the measurement made by the converter varies with the variation of the Vdd, so this, to ensure a correct result, must be as stable and noise-free as possible.
The module has an internal 0.6V generator that serves as an absolute reference and whose use we see in the Vdd tutorial.

The conversion yields a hexadecimal value that depends on the measured voltage, the reference voltage, and the definition of the converter:

ADC Resolution / Reference Voltage = ADC Reading / Measured Voltage

For an 8-bit conversion using the 5V Vdd as a reference, if the input voltage is 2V:

256 / 5 = ADC reading / 2.5

ADC reading = (256 /5) * 2.5 = 128d -> 7Fh

Given the digital nature of the value output from the conversion, a change in the analog input value will only be detected if it exceeds the resolution of the converter, i.e. the value between one step and the next. This means that input voltage variations are less than :

Reference Voltage / ADC Resolution

will not be detectable. In the case of an 8-bit conversion that uses a 5V Vdd as a reference, it is :

5 / 256 = 0.01953 V

Below this limit there will be no difference in the result of the conversion; To be able to evaluate variations smaller than 19mV, an AD converter with a higher resolution will be required.

From the above it should be clear that the result of the conversion is not an "absolutely" precise value, but is an approximation, due to the very nature of the conversion operation. Added to this are the errors due to the converter and fluctuations in the reference voltage and the voltage to be measured. As a result, the least significant bits of the result can be affected by error, the correction of which is usually carried out either without taking them into account or by using averaging algorithms (e.g. arithmetic mean, olympic mean, etc.) on several successive samples.

The Baseline ADC Module

Let's consider **Baselines with 2 or 3 analog channels**. Chips with more than 14 pins are Baseline, as already mentioned, a bit anomalous, more similar to Midrange.

The control of the ADC module for 10F220/2, 12F510 and 16F506/526 is delegated to an SFR called **`ADCON0`** (***ADc CONtrol register 0***):

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-0	R/W-0
ANS1	ANS0	ADCS1	ADCS0	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

The function of the individual bits is as follows:

bit7:6 **ANS1:0** Selecting Enabled Analog Input Channels:

ANS1:0	Channel configured as analog
00	nobody
01	AN2
10	AN0 and AN2
11	AN0, AN1, AN2

This points to two important things:

1. If you want no pin to be connected to an analog function, you have to programmatically set the two 7:6 bits of ADCON0 to level 0. As mentioned before, by default at the POR the value of these bits is 11, i.e. all pins available as analog are set as such.
   It should also be noted that the ANS1:0 bits at level 1 activate the analog function on the pins regardless of any other settings that have been defined before. The only exception is the comparator inputs, with which the ADC module shares the function.
   So, if we want to use AN0, AN1, AN2 pins as digital I/O, we will definitely have to disable the analog ADC function:

```
; Disable ADC
      bcf       ADCON0,ANS1
      bcf       ADCON0,ANS2
```

2. Some combinations of inputs can be selected: in the case indicated (16F506), AN1 or AN2 alone cannot be active, as well as any other combination not present in the table. This means that when assigning functions to pins, you have to check the situation of the microcontroller you want to use. For example,:

```
; ability AN0 e AN2
      bsf       ADCON0,ANS1
      bcf       ADCON0,ANS2
```

Of course, only two channels can be selected for the PIC10F220/222.

bit5:4 **ADCS1:0** Selecting the Clock for Conversion:

ADCS1:0	Clock
00	*FOSC/16*
01	*FOSC/8*
10	*FOSC/4*
11	*INTOSC/4*

The speed at which the conversion is performed depends on the clock that is applied to the ADC module and this is selected from the indicated bits. It is a matter of using either a clock generated with external components (LP,XT,HS,EC ExtRC) or the internal oscillator. In case the internal oscillator is also the main oscillator, the choice **`Fosc/4`** and **`INTOSC/4`** are the same thing.

The ADC module, as we have said, is of the successive approximation type and requires, in order to complete the conversion, a time of **13** ***Tad***, where ***Tad*** is equal to the cycle of instruction. So, for 4MHz clock, Tad=1us. This means that the conversion is completed in **13us**.
If the clock is **8MHz**, ***Tad*=500ns** and the conversion takes up **6.5us**.
Where a higher external clock can be used, as ***Tad*** must be between 500ns

and 50us, you need to use a division factor other than /4. So, with an external clock of 20MHz, you need a /16 split:

***Fosc*= 20MHz *FOSC/4* = 5MHz -> *Tad* = 200ns**

which is too low. Then we resort to:

***FOSC/16* = 1.25MHz -> *Tad* = 800ns**

which is within the limits provided. Here, the conversion is completed in 10.4us.
It should be noted that the increase in the main clock may not reduce the conversion time: the ADC module is a "module", i.e. an element that works separately from the processor, to which it is synchronized by the clock, but which requires its own time to complete its tasks.
In any case, it is not advisable to go below the limits indicated by the data folio for ***Tad*** in an attempt to shorten the conversion time, because the result may be incorrect and unsystematic.

bit3:2 **CHS1:0** Input Channel Selection:

ADCS1:0	Conversion Clock
00	AN0
01	AN1
10	AN2
11	0.6V Internal Reference

We mentioned that the ADC module has multiple input channels; these correspond to the AN0,1,2 pins and are multiplexed to the module input. Since, of course, only one signal can be converted at a time, the choice of input channel is made with the 3:2 bits of the control register.

Choice 11 does not correspond to any channel, but connects the input of the ADC module with the internal reference that is used for conversion. What is the point of measuring the value of a reference voltage, which is therefore very stable and of a well-defined value? This becomes important in battery-powered equipment without intermediate stabilizers (we know that PICs can work without problems between 2 and 5.5V); if the voltage drops as the battery runs out, the Vdd that is used as the top reference for the conversion varies, which will be incorrect.
By sampling the reference voltage, it becomes possible to correct the conversion results. It is also used to indirectly evaluate the supply voltage Vdd, as we will see later.

bit1 **GO/DONE** Conversion Start Bits/End of Conversion Flags:

GO/DONE	Writing	Reading
1	Start Conversion	

0		Conversion Complete

This bit has a somewhat peculiar function, as it is both a command bit and a signal flag:

- in write, if set to 1, it starts the conversion
- in read, becomes 0 when the conversion is complete

Since Baselines don't handle interrupts, you need to poll the flag to see if the conversion is complete:

```
; Start Conversion
      bsf       ADCON0,GO
; Waiting for the end of the conversion
eclp  btfsc     ADCON0,NOT_DONE
       goto     eclp
....
```

Notice that, apparently, the **bsf** and the **btfsc** refer to two different bits. However, the bit/flag for initiating the conversion is defined in the file ***procesorename.inc*** with several aliases:

```
;----- ADCON0 Bits --------------------------------------------------
ADON          EQU H'0000'
GO_NOT_DONE EQU H'0001'

GO            EQU H'0001'
CHS0          EQU H'0002'
CHS1          EQU H'0003'
ADCS0         EQU H'0004'
ADCS1         EQU H'0005'
ANS0          EQU H'0006'
ANS1          EQU H'0007'

NOT_DONE      EQU H'0001'
```

It may, therefore, be more significant (although certainly not mandatory) to use one label or the other depending on whether it is the operation of the command bit or the analysis of the end-of-conversion flag.

bit0 **ADON** ADC Module Power On Bits:

GO/DONE	ADC
0	disabled
1	qualified

The ADC module can be "turned on" and "off" by program. This is useful for reducing power consumption by enabling the module only when needed.

 **Warning** : Turning off the ADC module with the ADON bit is different from disabling the analog functions associated with the pins.
By changing the **ADON** bit to 0, the ADC module is disabled, but the settings fixed with the other bits of the **ADCON0** register remain valid.
So, to eliminate the unrequired analog functions from the pins, you have to act on the **ADC1:0** bits, as mentioned above.

To recap, managing AD conversion involves the following steps:

1. define which pins are to be allocated to the analogy input with **ADC1:0**
2. Define the conversion clock with **ADCS1:0**
3. select the channel you want to read with **CHS1:0**
4. enable the module with **ADON**
5. We also add a certain delay to allow stabilization and sample of the input voltage by the converter
6. start conversion with **GO/DONE**
7. wait for the conversion to finish by testing the

bit In instructions:

```
; setup ADC
      movlw b'01110010'
;             01 -------AN2
;             --11 ------NTOSC/4
;             ----10 ---CH2
;             ------1-  Abilita ADC
      movwf   ADCON0
; Waiting for stabilization
      delay10us          ;at least 10u
; Start Conversion
      bsf     ADCON0,GO
; Waiting for the end of the
conversion
eclp  btfsc   ADCON0,NOT_DONE
```

Alternatively, the setup can be formulated as a chain of ORs:

```
; setup ADC
      movlw   b'01'<<ANS0|b'11'<<ADCS0|b'10'<<CHS0|1<<ADON
      movwf   ADCON0
```

which, however, is a bit laborious.

The result obtained from the conversion can be found in a special **ADRES** register. The low value indicates the measured voltage:

*Vin = ADRES * Vref /256*

In our case, Vref is the Vdd. You can clearly see the need to have a stabilized supply voltage the more you want a correct result of the conversion; It is not so much the absolute value that matters, since, knowing it, it will be possible to evaluate the result correctly, but the absence of oscillations and ripples that would distort the measurement in a way that would be difficult to adjust.

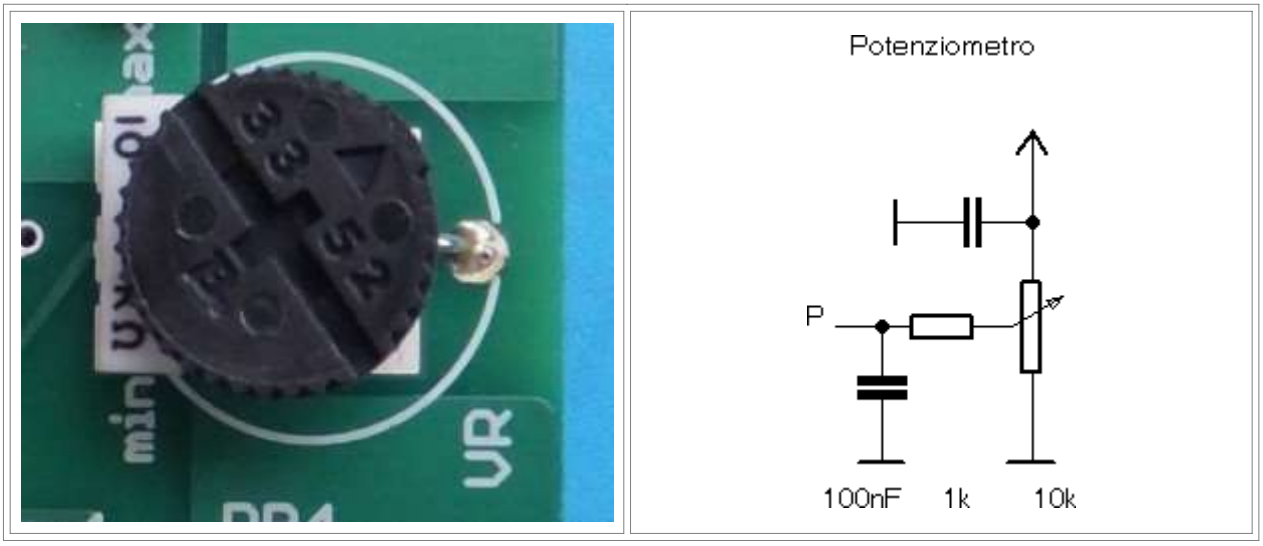
At the end of the conversion, if you need to reduce power consumption, you will disable the ADC module, zeroing the **`ADON bit`**.

Using the ADC to evaluate a voltage.

Let's start by verifying the operation of the ADC on a minimal PIC. In the 10F2xx family, we find the 10F222 and 10F224 which have the converter with two input channels.
We can make a "gauge" that indicates if a voltage is outside a certain radius. Due to the low amount of I/O available, we use two LEDs for the indication:

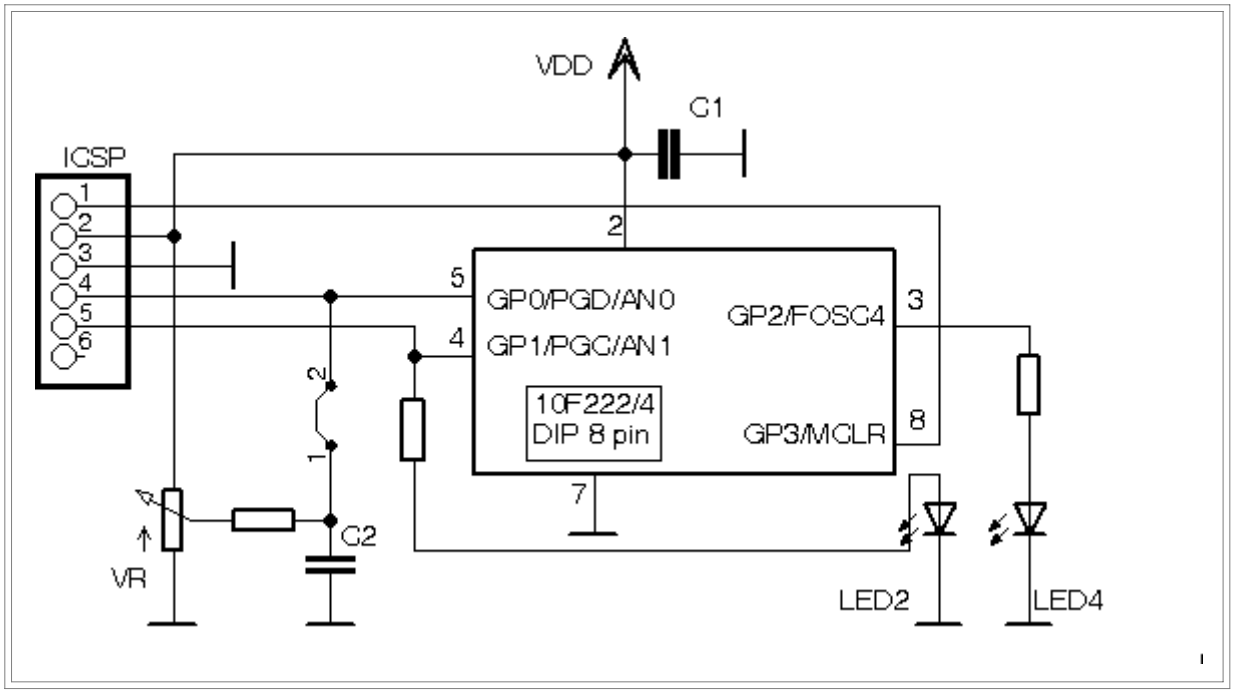
LED1	LED4	Tension
On	**Off**	low
On	**On**	Just
Off	**On**	high

As a signal source we use the potentiometer that the **LPCuB** board provides. This is connected between Vdd and Vss and therefore the cursor slides over all values within this range.



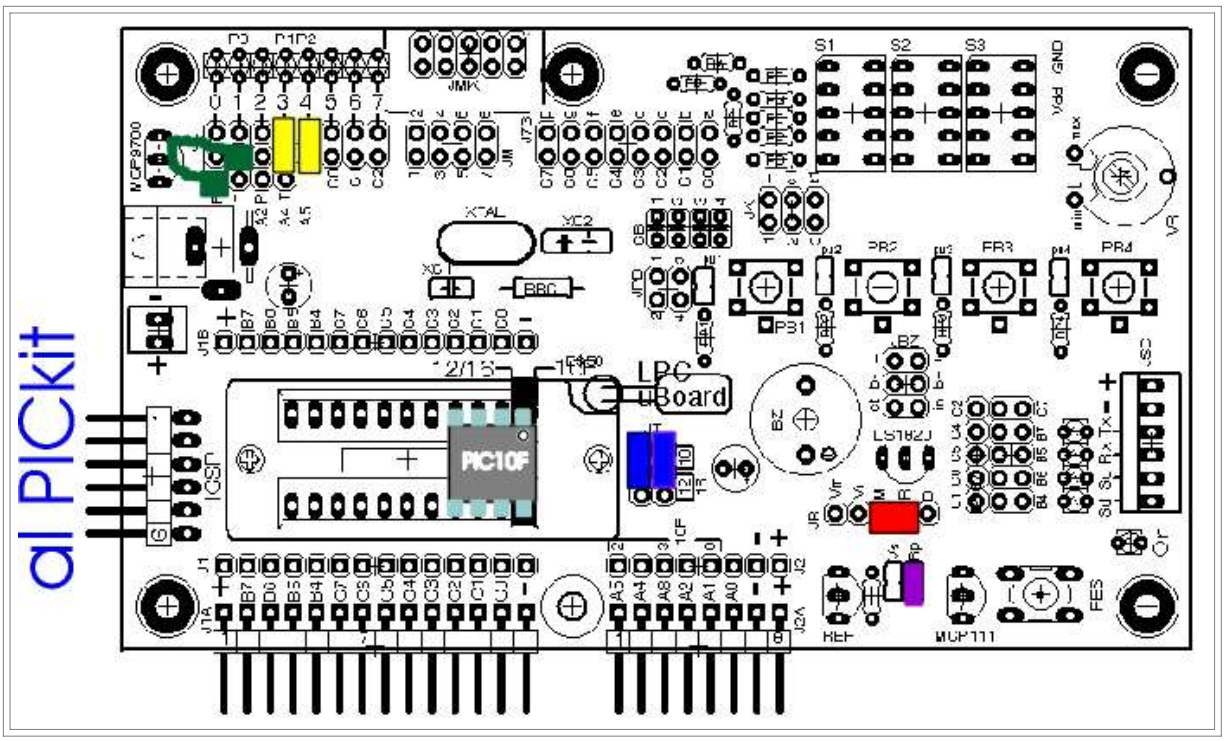
The slider has a limited value resistor in series (from 100ohm to 1k) with the function of protecting the input pin of the microcontroller from any electrostatic phenomena and which, together with a 100nf capacitor, constitutes a low pass filter to limit any impulsive disturbances.

The electrical circuit is simple:



The input voltage to the **`AN0 channel`** is derived from the VR

potentiometer. On the LPCuB, you only need a few jumpers:



The "yellow" jumpers connect the LEDs.
The "green" jumper connects the potentiometer slider.

The "green" jumper must be removed during chip programming, since **AN0** is shared on **GP0** along with the **PGD** data exchange line that is used by the ICSP connection.

In addition, **the Pickit must be disconnected during operation** to avoid having problems due to its internal circuits that would be in parallel to the analog input.

The Reset button is not used in the tutorial and can remain plugged in or unplugged as desired.
If it is connected, always remember not to press it during the programming phase, as the MCLR input shares the Vpp programming voltage.

The program.

We can set the voltage limits for which the LEDs should turn on or off, using a percentage form of the reference voltage value:

```
;*********************************************************************
; CONSTANTS
; Voltage Limits
min EQU (255/3)      ; limite minimo 33% di Vdd (1.66V circa per Vdd=5V)
max EQU ((255/3)*2)  ; limite massimo 66% di Vdd (3.33V circa per Vdd=5V)
```

However, we can express the limits in any other valid way and vary them depending on the application.
The PIC10F220/222 have an ADC module with a more simplified control than described above. In particular, since they can only operate with the internal clock, there is no need to choose the conversion clock. So ADCON0 looks like this:

R/W-1	R/W-1	U-0	U-0	R/W-1	R/W-1	R/W-0	R/W-0
ANS1	ANS0	—	—	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

Due to the low number of pins available, there are only two analog channels and they can be set individually:

ANS1:0	Function
00	Digital GP1 and GP0
01	Analog GP0, Digital GP1
10	GP0 Digital, GP1 Analog
11	Analog GP1 and GP0

Please note that, by default to the POR, both inputs are activated (ANS1:0 = 11) and that to access the digital function you must set them to 0.
The 5:4 bits, which are used for the selection of the conversion clock, have no function here (bits not implemented).
Regarding the selection of the channel to be measured:

CHS1:0	Function
00	AN0/GP0
01	AN1/GP1
10	Vref 0.6V
11	Vref 0.6V

The enable bits and the conversion start/end bit/flag are identical to the above.

Based on this, we configure the ADC module's control register to have the AN0 input:

```
;configure ADC for AN0 enable
    movlw   b'01000001'
    movwf   ADCON0
```

Let's add a wait of at least 10us to allow the stabilization of the settings and the charging of the internal sample&hold capacitor on the chosen analog channel and start the conversion:

```
; Expected stabilization
     DLY10US

; Start Conversion
     bsf   ADCON0,GO
adlp btfsc ADCON0,NOT_DONE
      goto adlp
```

When the **NOT_DONE flag** goes to level 1, the conversion is complete.
We use the result by comparing it with the limits to properly manage the LEDs The

source can be compiled for 10F220 and 10F222 (default).

By turning the potentiometer, only LED3 will be lit for the first third of the stroke; in the central part, LED3 and LED4 will be lit. For the last third, only LED4 will be lit.

If the voltage to be measured is higher than the supply voltage of the microcontroller, a **divider must be inserted** so that the value at the end of the stroke of the potentiometer does not exceed Vdd.

T&T

When writing this source, in the cut-and-paste, the **radix dec line** that changes the indication of the numeric base has escaped. As a result, the Assembler used the hex base, which is the default.
Subsequently, in the definition of the values of the maximum and minimum limits, trusting in the ubiquitous insertion of the decimal root request, the thing was expressed as follows:

```
; Voltage Limits
min EQU (255/3)      ; Minimum limit 33% of Vdd
max EQU ((255/3)*2)  ; maximum limit 66% of Vdd
```

The result of the build was an expected **BUILD SUCCEEDED**, with the executable file being generated, but the usual attention to the rest of the message detected an unexpected line:

```
Warning[202] C:10A_10F. ASM 143 : Argument out of range. Least significant
bits used.
```

What happened?
the value 255, in the absence of a decimal root, was understood as 255h and the result of the formula for **max** was 18Eh.

*(255h /3) *2 = C7h * 2 = 18Eh*

This exceeds the 8-bit insertable number (data width) and the compiler used only the low part (8Eh), producing the executable, but warning of the problem. The program would not have worked properly.
In the ***.lst*** file we find the counterpart of the report:

```
Warning[202]: Argument out of range. Least significant bits used.
0015    0C8E     00141    movlw max
```

By entering the desired root definition, the problem is corrected, since 255 would be understood as a decimal:

*(255 / 3) *2 = * 2 = 170 -> AAh*

In the ***11A_10F.asm file***, the definition line of the numeric root is commented out, so that the compilation recreates the error situation. All you have to do is remove the **;** to correct the situation (remember once again that the content of the comment lines, which begin with the ; is not considered when compiling).
The case reminds us that, even if a root has been specified (or is thought to have been specified...) it is always a good idea to indicate the numbers correctly:

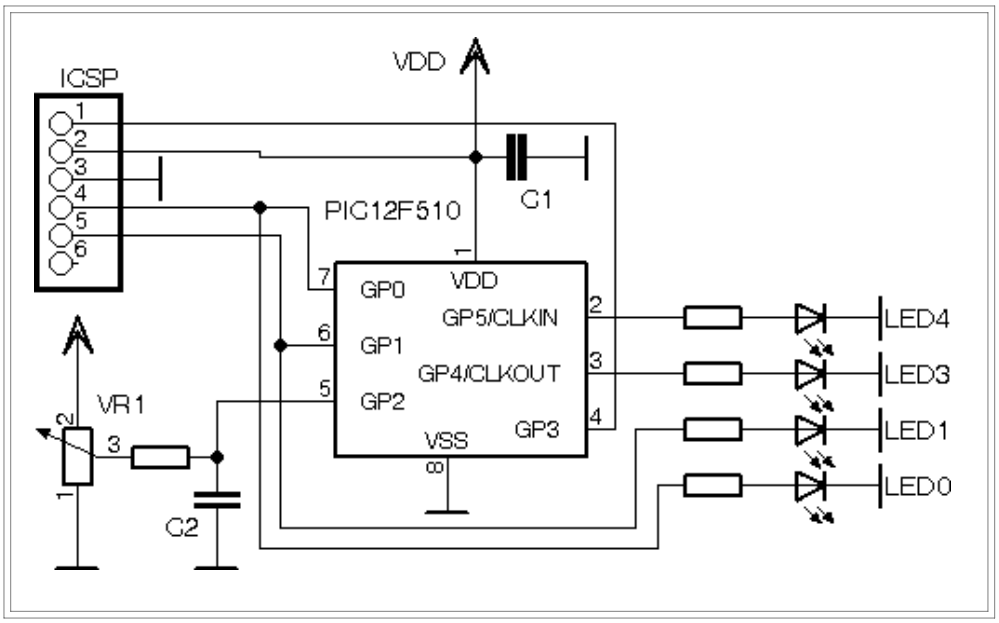
```
; Voltage Limits
min EQU (.255/3)      ; Minimum limit 33% of Vdd
max EQU ((.255/3)*2)  ; maximum limit 66% of Vdd
```

and, above all, it is absolutely necessary to read ALL the message that MPASM sends at the end of the compilation, not being satisfied with the BUILD **SUCCEEDED** alone, which may not be enough to confirm the correctness of the result.

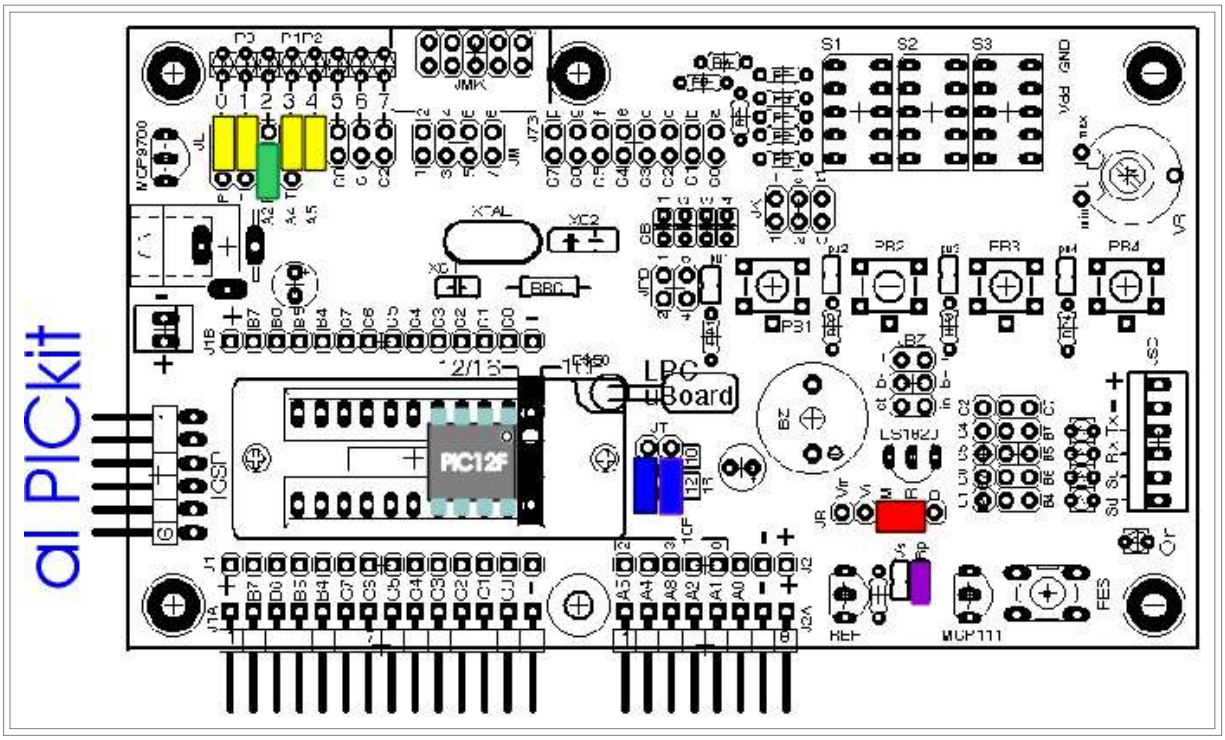
Display the data resulting from the conversion on 4 LEDs

In a few minutes, we can create a simple example relevant to the topic: **we want to visualize the result of the AD conversion, in binary form, on some LEDs**.

As a signal source we still use the potentiometer that the **LPCuB** board provides and as a processor we use a 12F510, with 6 I/O, which allows you to control 4 LEDs, which will be lit in proportion to the result of the conversion.
To have 4 I/Os available as outputs, you only need to use one analog. The chip allows you to select a single input to be dedicated to the ADC, which is **AN2**/**GP2**, as a single input to be dedicated to the ADC:



About the **LPCuB**:



The "yellow" jumpers connect the LEDs to **GP0/1/4/5**.
The "green" jumper connects the potentiometer slider to **GP2**.

The program.

We have two problems: the first is the one we've already seen, which is aligning the LEDs by jumping **GP3**, which cannot be used as an output, and also **GP2** which is used by the analog input.
The second concerns the representation of the result of the AD conversion, which is on 8 bits, on only 4 LEDs.
This is easily overcome with a few considerations; first of all, we observe that with 4 LEDs we can represent binary numbers from 0 to F (from 0000 to 1111), while the number represented on 8 bits varies between 0 and FF (from 00000000 to 11111111).
We can then use only the highest 4 bits, discarding the low nibble; We will have a binary presentation of the result equal to:

display = result / 16

Dealing with binary numbers, a division by 2 is easily achievable with a shift to the right. Dividing by 16 corresponds to 4 successive shifts:

```
; divide result per 16
      movf        result, w
      andlw       0xF0                ; Delete Low Nibble
      movwf       result
      CLRC                            ; Reset Carry for Shift
      RRF         result,f
      RRF         result,f
      RRF         result,f
```

```
      rrf         risultato,f
```

However, this can be solved much more efficiently with a simple swap:

```
; divide result per 16
      swapf       result, w           ; Swap High Nibble<->Nibble    low
      andlw       0x0F                ; Delete Nibble High
      movwf       result
```

You then have to exclude **GP2** and **GP3** **and command** GP4 **and** GP5 **in their place**:

```
; Move Bit 2-3 to 4-5
        btfsc    temp,2
         bsf     temp,4
        btfsc    temp,3
         bsf     temp,5
        movf     temp,w
        movwf    GPIO
```

The data is passed directly to the GPIO without using shadows as all bits are written in a single operation.
The Timer0 is used to cadence an operation approximately every 130ms.

The fact that the lower bits of the conversion result are not taken into account makes the LED display very stable, even if a certain angle of rotation is required to move the binary counter forward or backward.

SWAPF

The instruction operates on a byte by swapping the high nibble with the low nibble. For example,:

```
movlw b'00001111'
movwf target ; target = 00001111
swapf target,f ; exchange nibble high <->nibble
basso

 ; ora target is 11110000
```

The actual conversion mirrors what was said above:

```
; setup ADC
      movlw b'01110010'
;             01 -------AN2
;             --11 ------NTOSC/4
;             ----10 ---CH2
```

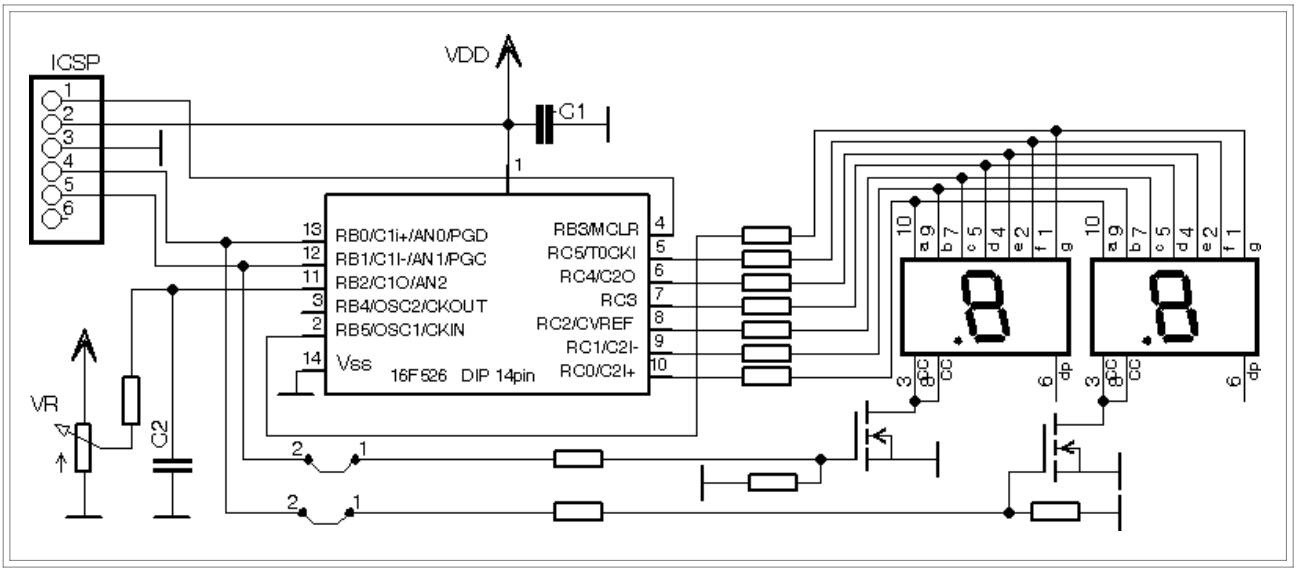
```
;               ------1-  Enable ADC
      movwf     ADCON0
; Start Conversion
      bsf       ADCON0,GO
; Waiting for the end of the conversion
w lp  btfsc     ADCON0,NOT_DONE
goto    wlp
; Save Conversion Result
movf  ADRES,w
      movwf     risultato
```

With the usual few modifications it will be possible to bring the source to 16F506/526. The 10F220/2 also has the ADC module, but you would only have two free I/O.

By turning the potentiometer, the binary digit presented by the LEDs will change, with all off if the cursor is turned towards Vss (min.) and all lit if it is rotated towards Vdd (max).

Analog data reading on 2 7-segment displays

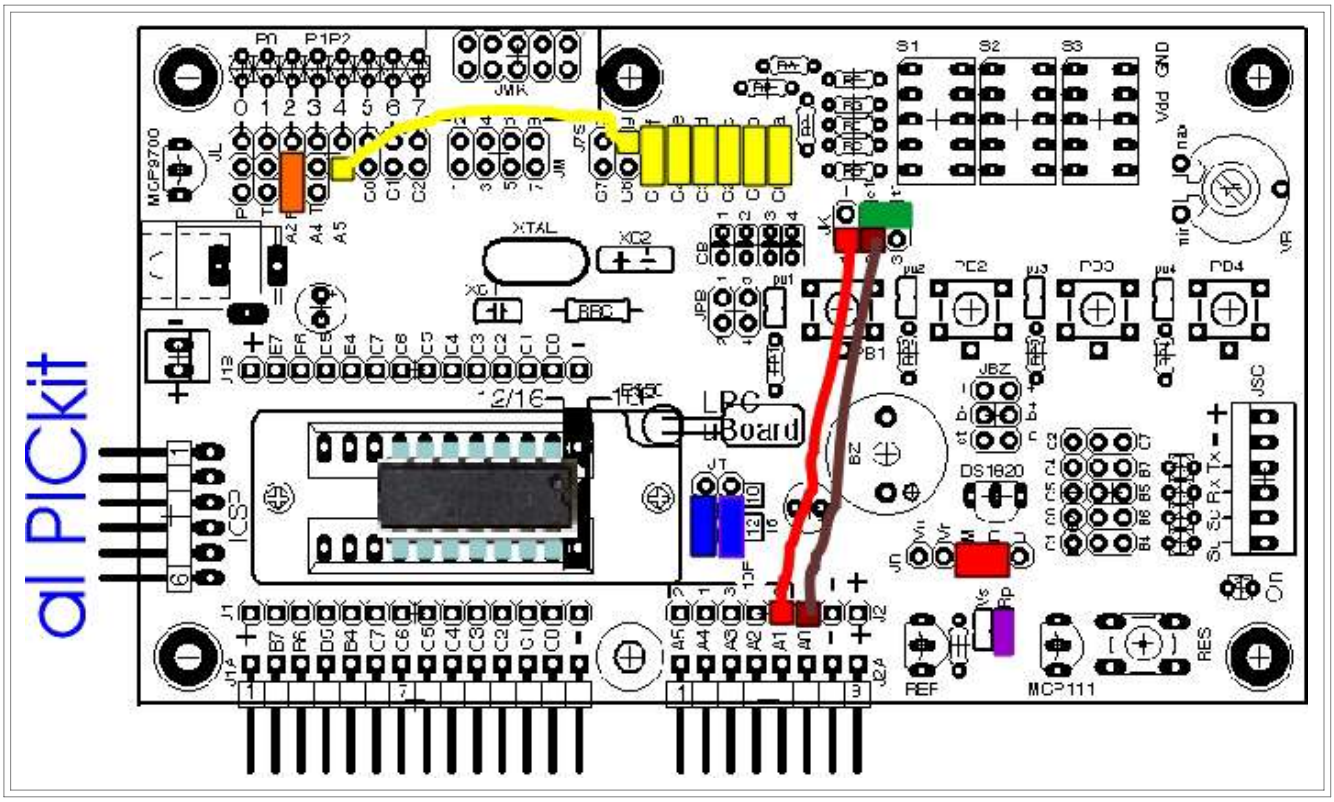
Using a 14-pin chip (16F506/526) we can have the reading of the converted data on 7-segment displays.



Having 2 displays we will be able to represent the entire number of the conversion result, which varies between 00 and FFh.

Jumpers in series to the lines that derive from **RB0** and **RB1** indicate the need to disconnect them during on-board programming.

About the **LPCuB**:



The "yellow" jumpers connect the segments.
The "orange" jumper connects the potentiometer slider to **RB2**.
The "brown" and "red" flying jumpers connect the command I/O of the MOSFET gates. These jumpers will be disconnected when programming the chip.
the "green" jumper connects the gate of the first MOSFET.
The reset jumpers ("red" and "purple") are not necessary, as the program does not provide the function.

The program.

Very simple, it uses various elements seen above and in particular:

- The lookup table system to control segments
- Multiplex mode to manage the two digits

The one-bit state of Timer0 that counts free run is used for the cadence of the multiplex, in which the AD conversion is inserted.
The data obtained from the conversion, in the form of a hexadecimal byte, is passed, divided into two nibbles, directly to the lookup table to obtain the corresponding digits.
By turning the potentiometer, the voltage applied to the input of the ADC module changes and therefore the display indication which varies between 00 and FFh.

A flickering of the unit digit is normal, since no correction of the units is entered in the program: the conversion is affected by any slight variation both of the Vdd, used as a reference, and of the input voltage, afflicted by disturbances given by the induced voltages of the nearby circuits in which the impulse voltages of the multiplex flow, in addition to the normal precision that is ±1/2LSB.
To avoid flickering, slower sampling or multi-value averaging algorithms can be used.

11A_510F.asm

```
;*****************************************************************
; 11A_510.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 11A
;                     ADC result on 4 LEDs
;
;    PIC            : 12F510
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-----------------------------------------------------------------
;
; Pin use :
;   ________________
;       12F510 @ 8 pin
;
;                  |‾‾\/‾‾|
;              Vdd -|1    8|- Vss
;              GP5 -|2    7|- GP0
;              GP4 -|3    6|- GP1
;         GP3/MCLR -|4    5|- GP2
;                  |______|
;
;    Vdd                      1: ++
;    GP5/OSC1/CLKIN           2: Out LED4
;    GP4/OSC2                 3: Out LED3
;    GP3/! MCLR/VPP           4: MCLR
;    GP2/T0CKI/AN2/C1OUT      5: Analog Input
;    GP1/ICSPCLK/AN1/C1IN-6: Out LED1
;    GP0/ICSPDAT/AN0/C1IN+ 7: Out LED0
;    Vss                      8: --
;
; ###############################################################
;
        LIST      p=12F510
         #include <p12F510.inc>

        radix     DEC

; ###############################################################
;                           CONFIGURATION
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
  __config _IntRC_OSC & _IOSCFS_OFF & _WDT_OFF & _CP_OFF & _MCLRE_ON

; ###############################################################
;                            RAM
; general purpose RAM
        UDATA           ; Start Area RAM
Temp    RES 1           ; temporary
```

```
; ###################################################################
;                            MACRO

  #include C:\PIC\Library\Baseline\basemacro.asm

;*******************************************************************
;===================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;|LED4 |LED3 |MCLR | AN2 |LED1 |LED0 |
;
;#define      GPIO,GP5    ; LED4
;#define      GPIO GP4    ; LED3
;#define      GPIO,GP3    ; MCLR
;#define      GPIO,GP2    ; AN2
;#define      GPIO,GP1    ; LED1
;#define      GPIO,GP0    ; LED0

; ###################################################################
;                          RESET ENTRY
;
; Reset Vector
RESVEC     TAILS    0x00

; MOWF Internal Oscillator
        Calibration OSCCAL
        pagesel init
        goto    Init

; ###################################################################
;             TABLES AND SUBROUTINES ON PAGE 0

; ADConv Voltage
Measurement:
; Waiting for noise reduction
        DLY10US
; Start Conversion
        Bsf     ADCON0,GO
ADLP    BTFSC   ADCON0,NOT_DONE
         goto   ADLP
        retlw   0

;
; ###################################################################
;                         MAIN PROGRAM
Main    TAILS

Init:
; Disable comparators to free the BCF digital function
                CM1CON0, C1ON

; disable T0CKI from RB5, prescaler 1:256 to Timer0
         ;      b'11111111'
         ;        1-------     GPWU Enabled
```

```
          ;         -1------     GPPU Enabled
          ;         --0-----     Internal Clock
          ;         ---1----     Falling
          ;         ----0---     prescaler to Timer0
          ;         -----111     1:256
         movlw   b'11010111'
         OPTION

         CLRF    GPIO             ; Clear Port Latch

; All useful ports come out
         movlw   0
         Tris    GPIO

; configure ADC for AN2, INTOSC/4 and
         enable movlw     b'01111011'
         movwf    ADCON0

         CLRF     TMR0       ; Initialize Timer

;-----------------------------------------------------------------
; Read/Display Cycle
; Digit Unit
displp movf      TMR0,w      ; 65ms ?
         skpz                ; yes -
         ADC
          Goto   displp      ; No - Waiting

; AD Conversion
         pagesel ADConv
         call    ADConv
         pagesel $

; Conversion Result Display
; align result bits with available GPIOs swapf
                 ADRES,w     ; swap nibbles in W
         andlw   b'00001111' ; bit only3:0
         movwf   Temp        ; save

 ; Move Bit 2-3 to 4-5
         BTFSC   temp,2
          Bsf    temp,4
         BTFSC   temp,3
          Bsf    temp,5
         movf    temp,w
         movwf   GPIO

; waiting
dlp1     movf    TMR0,w      ; end of state TMR0=0
                            ?
         BNZ     displp      ; Yes - Loop
          Goto   dlp1        ; No - Waiting

;*****************************************************************
;                            THE END
         END
```

11A_10F.asm

```
; 11A_10F.asm
;-------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 11A
;                     ADC Usage
;
;    PIC            : 10F220/2
;    Support        : MPASM
;    Version       A: V.20x-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-------------------------------------------------------------------
;
;  Pin use :
;  ________________
;     10F220/2 @ 8-pin DIP          10F220/2 @ 6 pin SOT-23
;
;                 |__\/__|                    *______|
;            NC -|1    8|- GP3          GP0 -|1    6|- GP3
;           Vdd -|2    7|- Vss          Vss -|2    5|- Vdd
;           GP2 -|3    6|- NC           GP1 -|3    4|- GP2
;           GP1 -|4    5|- GP0                |______|
;                |______|
;
;                              DIP  SOT
;    NC                         1:  Nc
;    Vdd                        2:  5: ++
;    GP2/T0CKI/FOSC4/           3:  4: Out  LED
;    GP1/ICSPCLK/AN1            4:  3: Out  LED
;    GP0/ICSPDAT/AN0            5:  1: AN0
;    NC                         6:  NC
;    Vss                        7:  2: --
;    GP3/MCLR/VPP               8:  6:
;
;
;*******************************************************************
;===================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |---|---|---|---|
; |   | LED4| LED3| AN0 |
;
;#define  GPIO,GP0  ;  #define
LED3 GPIO,GP1 ; #define LED4
GPIO,GP2 ;
;#define GPIO,GP3 ;

;
;*******************************************************************
```

```
; ###################################################################
; Choice of #ifdef
 processor____10F222
        LIST      p=10F222
        #include <p10F222.inc>
 #endif
 #ifdef___10F224
        LIST      p=10F224
        #include <p10F224.inc>
 #endif

        ; radix dec  ; <<<--- to be corrected !!

; ###################################################################
;                         CONFIGURATION
;
 ; No WDT, no CP, pin4=GP3
  __config _IOFSCS_4MHZ & _MCPU_OFF & _WDT_OFF & _CP_OFF & _MCLRE_OFF

; ###################################################################
;                         RAM
;
; ###################################################################
;                         CONSTANTS
; Voltage Limits
min EQU (255/3)         ; Minimum limit 33% of Vdd
max EQU ((255/3)*2)     ; maximum limit 66% of Vdd

; ###################################################################
;                         MACRO
; DLY10US - Waiting 10us
; PIC Baseline with FOSC =
4MHz DLY10US MACRO
    goto $+1            ; 5 x 2US
    Goto $+1
    goto $+1
    goto $+1
    goto $+1
         ENDM

; ###################################################################
;===================================================================
;                          RESET ENTRY
;
; Reset Vector
        ORG     0x00

 ; Internal Oscillator Calibration
        andlw   0xFE        ; ensures no FOSC4
        movwf   OSCCAL

; Pre Clear Output
        CLRF    GPIO

; disable T0CS to free GP2
; OPTION default 11111111
;                 1-------     ! GPWU disable
```

```
;                -1------     ! GPPU disable
;                --0-----     disable T0CS
;                ---1----     Falling Edge
;                ----1---     prescaler at WDT
;                -----111     Rate 1:256
        movlw   b'11011111'
        option

; GP2:1 come out
        movlw   b'00001001'
        TRIS    GPIO

; configure ADC for AN0 and
enable mainloop:
        movlw    b'01000001'
        movwf    ADCON0

; Expected stabilization
        DLY10US

; Start Conversion
        Bsf      ADCON0,GO
ADLP    BTFSC    ADCON0,NOT_DONE
         goto    ADLP

; Compare Result with Limits
; Minimum Limit
        movlw    Min           ; w=ADRES-min
        subwf    ADRES,w
        Bnc      Minexit       ; if C=0, ADRES<=min
; Maximum
        movlw    Max           ; w=ADRES-max subwf
                 ADRES,w       ; if C=1 max=>ADRES
        bc       MaxExit
; ok - in the desired range - LED management
Inrange BSF      LED3          ; in range - both LEDs lit bsf
                 LED4
        Goto     Mainloop
; Below idle - LED management
Minexit Bcf      LED4             ; <= minimum - LED3 on
        Bsf      LED3
        Goto     Mainloop
;       Above the maximum - LED management
Beyond
MaxExit Bsf      LED4             ; >= maximum - LED4 on
        Bcf      LED3
        Goto     Mainloop

;*****************************************************************E
       ND
```

11A_526.asm

```
;*****************************************************************
; 11A_526.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 11A
;                     Two-digit hex ADC result
;
;    PIC            : 16F506/526
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-----------------------------------------------------------------
;
;  Pin use :
;  ________________
;       16F506/526 @ 14 pin
;
;                   |‾‾\/‾‾|
;              Vdd -|1   14|- Vss
;              RB5 -|2   13|- RB0
;              RB4 -|3   12|- RB1
;         RB3/MCLR -|4   11|- RB22
;              RC5 -|5   10|- RC0
;              RC4 -|6    9|- RC1
;              RC3 -|7    8|- RC2
;                   |______|
;
;    Vdd                     1: ++
;    RB5/OSC1/CLKIN          2: Out  segm g
;    RB4/OSC2/CLKOUT         3:
;    RB3/! MCLR/VPP          4: MCLR
;    RC5/T0CKI               5: Out  segm f
;    RC4/C2OUT               6: Out  segm and
;    RC3                     7: Out  segm d
;    RC2/CVref               8: Out  segm c
;    RC1/C2IN-               9: Out  segm b
;    RC0/C2IN+              10: Out  segm at
;    RB2/C1OUT/AN2          11:AN2
;    RB1/C1IN-/AN1/ICSPC    12: Out  Gate unit
;    RB0/C1IN+/AN0/ICSPD    13: Out  Gate Dozens
;    Vss                    14: --
;
; ###############################################################
; Choice of #ifdef
  processor_16F526
         LIST      p=16F526
         #include <p16F526.inc>
  #endif
  #ifdef____16F506
        LIST      p=16F506
         #include <p16F506.inc>
```

```
  #endif
        Radix     DEC

; ###################################################################
;                            CONFIGURATION
  #ifdef____16F526
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_CPDF_OFF & _MCLRE_ON
  #endif
  #ifdef____16F506
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
  __config _IntRC_OSC_RB4EN & _IOSCFS_OFF & _WDT_OFF & _CP_OFF &
_MCLRE_ON
  #endif

; ###################################################################
;                             RAM
; general purpose RAM
        CBLOCK 0x10          ; Start Area RAM
      Temp                   ; temporary
        ENDC

 ;*****************************************************************
;=================================================================
;            DEFINITION OF PORT USE
;
;P ORTC map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;|segmf|segme|segmd|segmc|segmb|segma|
;
#define     Segma  PORTC,0    ; F:A Segments
#define     segmb  PORTC,1    ;
#define     SEGMC  PORTC,2    ;
#define     SEGMD  PORTC,3    ;
#define     Segme  PORTC,4    ;
#define     SEGMF  PORTC,5    ;

;P ORTB map
;| 5 | 4 | 3       | 2 |       1 | 0 |
;|-----|-----|---- |-----|-- --|-----|
;|segmg|     |     | AN2 |GATE1|GATE2|
;
#define     GATE2  PORTB,0    ; Gate MOSFET Digit unit
#define     GATE1  PORTB,1    ; gate MOSFET digit Dozens
;#define            PORTB,2    ; AN2
;#define            PORTB,3    ; MCLR
;#define            PORTB,4    ;
#define     segmg  PORTB,5    ; Segment G
;
; ###################################################################
;                          CONSTANT
                             S
;
#define  bit05k  TMR0,1 ; TMR0 bit for 512us
#define  bit1k   TMR0,2 ;             1024us
#define  bit2k   TMR0,3 ;             2048us
```

```
#define bit4k    TMR0.4 ;              4096us
#define bit8k    TMR0.5 ;              8192us

#define bittmr bit4k ; 4096us

; ###################################################################
;                          LOCAL MACROS
; UNITon MOSFET gate
command  MACRO
        bsf GATE2
          ENDM
UNIToff MACRO
        bcf GATE2
          ENDM
DECon   MACRO
        bsf GATE1
          ENDM
DECoff MACRO
        bcf GATE1
          ENDM

; ###################################################################
;                          RESET ENTRY
;
; Reset Vector
        ORG 0x00

; MOWF Internal Oscillator
        Calibration OSCCAL
        goto Main

; ###################################################################
;             TABLES AND SUBROUTINES ON PAGE 0

;  Segment Data Table - Display Common Cathode
SEGTBL ANDLW 0x0F ; Low nibble only
        addwf PCL,f ; PC tip
        retlw  b'00111111'  ;  "0"  -|-|F|E|D|C|B|A
        retlw  b'00000110'  ;  "1"  -|-|-|-|-|C|B|-
        retlw  b'01011011'  ;  "2"  -|G|-|E|D|-|B|A
        retlw  b'01001111'  ;  "3"  -|G|-|-|D|C|B|A
        retlw  b'01100110'  ;  "4"  -|G|F|-|-|C|B|-
        retlw  b'01101101'  ;  "5"  -|G|F|-|D|C|-|A
        retlw  b'01111101'  ;  "6"  -|G|F|E|D|C|-|A
        retlw  b'00000111'  ;  "7"  -|-|-|-|-|C|B|A
        retlw  b'01111111'  ;  "8"  -|G|F|E|D|C|B|A
        retlw  b'01101111'  ;  "9"  -|G|F|-|D|C|B|A
        retlw  b'01110111'  ;  "A"  -|G|F|E|-|C|B|A
        retlw  b'01111100'  ;  "b"  -|G|F|E|D|C|-|-
        retlw  b'00111001'  ;  "C"  -|-|F|E|D|-|-|A
        retlw  b'01011110'  ;  "d"  -|G|-|E|D|C|B|-
        retlw  b'01111001'  ;  "E"  -|G|F|E|D|-|-|A
        retlw b'01110001' ; "F" -|G|F|E|-|-|-|At

; ADConv Voltage
Measurement:
; Waiting for noise reduction
```

```
        DLY10US
; Start Conversion
        Bsf     ADCON0,GO
ADLP    BTFSC   ADCON0,NOT_DONE
         goto   ADLP
        retlw   0

;
; ######################################################################
;                       MAIN PROGRAM
Main:
; Disable comparators to free the BCF digital function
                CM1CON0, C1ON
        Bcf     CM2CON0, C2ON

; disable T0CKI from RB5, prescaler 1:256 to Timer0
         ;      b'11111111'
         ;        1-------     GPWU Enabled
         ;        -1------     GPPU Enabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----0---     prescaler to Timer0
         ;        -----111     1:256
        movlw   b'11010111'
        OPTION

        CLRF    PORTB      ; Clear Port Latch
        CLRF    PORTC

; All useful ports come out
        movlw   0
        Tris    PORTB
        Tris    PORTC

; configure ADC for AN2, INTOSC/4 and
        enable movlw     b'01111011'
        movwf    ADCON0

        CLRF     TMR0       ; Initialize Timer

;---------------------------------------------------------------------
; cycle display
; Digit unit
displp  BTFSS   bittmr      ; 4ms ?
         Goto   displp      ; No - Waiting
        movf    ADRES,w     ; Yes - Load Conversion Result
        andlw   0x0F        ; Low nibble only
        Call    segtbl      ; Lookup table
        movwf   Temp        ; Save to Temporary
        movwf   PORTC       ; Write on the port segm f:a
        BTFSC   temp,6      ; Command G-Segment
         Bsf    segmg
        BTFSS   temp,6
         Bcf    segmg
        UNITon              ; Unit digit lit
dslp_1  BTFSC   bittmr      ; 4ms ?
         Goto   dslp_1      ; No - Waiting
```

```
UNIToff dslp_2              ; Yes - Turn off units
; digit tens
; ADRES swap in W, low nibble<->high swapf
                ADRES,w
        andlw   0x0f        ; Nibble only low
        call    segtbl
        movwf   Temp
        movwf   PORTC
        BTFSC   temp,6
         Bsf     segmg
        BTFSS   temp,6
         Bcf     segmg
        DECon               ; Turn on dozens
        of calls            ADConv    ; AD
        Conversion
dslp_3 btfss    bittmr      ; 4ms ?
         Goto   dslp_3      ; No - Waiting
        DECoff              ; Yes - Turn off tens
        Goto    displp      ; Loop

;*****************************************************************
;                            THE END
        END
```