# Knowledge of baseline

## Reading the status of the inputs

**The purpose of the tutorial is to read the logical state of the inputs of the microcontroller.**

We have seen so far how it is possible to control digital outputs; However, pins can also function as digital inputs.
The ability to define the pins as outputs or inputs is one of the strengths of microcontrollers: by reading the status of the inputs we can introduce into the program information derived from the situation of contacts, buttons, switches, keyboards, limit switches, encoders and sensors of many kinds, as well as the possibility of receiving signals from serial lines, data converters and peripherals of all kinds. The control of the pins as outputs will then make it possible to govern the controlled processes, operating LEDs, relays, motors, displays, etc.

The input-output conditions of a pin are variable by program and, where necessary by the application, can be dynamically changed during execution: it is not mandatory to maintain the same function at an I/O, but it is possible to change the input/output direction without problems where this is needed to properly manage the hardware.
Nor is it mandatory to assign the direction of the pins to the beginning of the source: this is commonly done only because in most applications the direction remains stable during the execution of the entire program. Here it may be useful to make a comparison with the settings set in the initial config, which, on the other hand, are not alterable by program (at least not in the Baselines).

We have seen how a digital output has two levels, 1 and 0: the first corresponds to a voltage close to Vdd, the other to a voltage close to 0 (Vss).

The following problem can be asked: what voltage do I need to apply to the input for the processor to read a value of 1 rather than 0?
Generally speaking, we can say that the Vdd and the Vss are the absolute references:

| Input connected to the | Logic Level |
|---|---|
| Vdd | 1 |
| Vss | 0 |

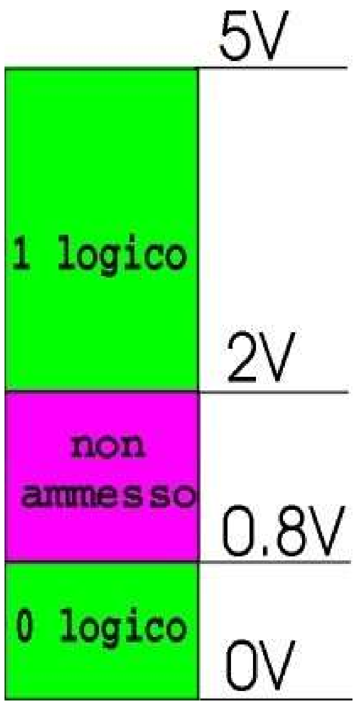
since the voltage applied to a pin cannot positively exceed the Vdd or go below the Vss.

In fact, however, a direct connection to the Vdd or Vss is not required, but it is sufficient that the voltage at the input pin is within the limits of logic circuit recognition.
What is meant by this?

A "logic level" is the voltage at which a signal is recognized by the digital circuit as 1 or 0.
We know that binary logic treats data whose digits have only two values:

| **HIGH level** | **High** | **Set** | **1** |
|---|---|---|---|
| **LOW level** | **Low** | **Clear** | **0** |

How do we define a level 0 (LOW) or 1 (HIGH) with respect to the supply voltage?



Chip manufacturers generally refer to the most common standard which is **TTL** (*Transistor-Transistor Logic*).
This establishes a maximum value for the voltage that is interpreted as 0 and a minimum for the voltage corresponding to 1.

Given the TTL supply voltage which is 5V, three bands are formed:

- From 0 to 0.8V all voltages applied to the input are interpreted as logical 0
- From 2 to 5V all voltages applied to the input are interpreted as 1 logic
- Voltages between 0.8V and 2V should never be sent to the TTL digital input as they would not give rise to uncertainty in the attribution to one or the other level.

For PICs, which are built in MOSFET technology, these levels depend on the supply voltage Vdd, which can range from 2 to 5.5V, and on the type of gate that constitutes the input circuit: by this we mean that the circuits "behind" the pins, configured as input, can be of three different types, depending on the use for which the pin has been designed.
For a 5V "TTL" power supply:

| **Digital input pin logic levels for 4.5V < Vdd < 5.5V** | | | |
|---|---|---|---|
| Type | **Vil** (min) | **Vih** (max) | **Parameter** |
| **TTL** | 0.8 V | 2.0 V | D030A/D040A |
| **Schmitt trigger** | 0.3 Vdd | 0.7 Vdd | D034/D044 |
| **SMBUS** | 0.6 V | 1.4 V | D034A/D044A |

For a lower power supply, the parameters shift slightly. For the entire

Vdd power supply range, you can use this table:

| **Digital Input Pin Logic Levels** | | | |
|---|---|---|---|
| Type | **Vil** (min) | **Vih** (max) | **Parameter** |
| **TTL** | 0.15 Vdd | 0.25 Vdd + 0.8 V | D030/D040 |

So, for example, if we power the micro to 5V, when we bring an input pin to a voltage below 0.75V the processor will consider that pin to be at a low level (0). If we apply a voltage higher than 2.05V, the processor will consider that pin to be at a high level (1).

What about intermediate voltages? These are to be avoided absolutely, because in the section between the two limit values the electronic circuit may not be able to attribute a valid logical value to the voltage; You have an uncertainty band that should never appear at the input pin.
Let's remember that we are talking about digital logic, which works on defined values and cannot admit other states than 1 and 0.
To simplify, we can say that by connecting the pin to the Vdd we definitely have a value of 1 and by connecting it to the Vss we definitely have a value of 0. For the connection we can use mechanical contacts (switches, buttons, switches, etc.) or electronic switches (transistors). The important thing is that at the entrance you only have values in the 1 or 0 range.

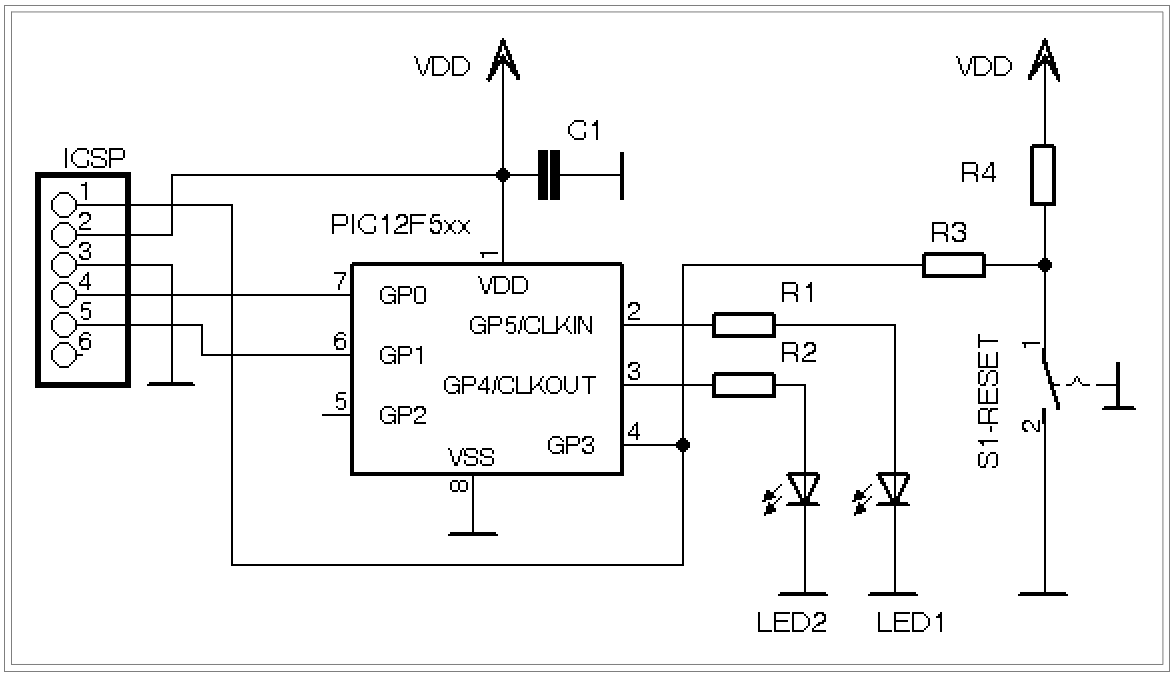
For the sake of completeness, it should be said that the microcontroller accepts variable values in the entire range between Vss and Vdd, but does so exclusively through non-digital, but "analogue" inputs, such as those of comparators and AD (Analog-Digital) converters, whose characteristics and uses we will see later. In these cases, however, the data collected in this way must be converted into binary values, defined by states 1 and 0.

So let's start by simply interfacing with buttons, present on each demoboard or easily implementable on the breadboard. Here you can find **some information pages** on the subject.

## Basically.

**We want to turn on one LED when one button is open and another LED when the button is closed.**

We use the 12F5xx, as well as **the 12F519** and 12F508/509 .



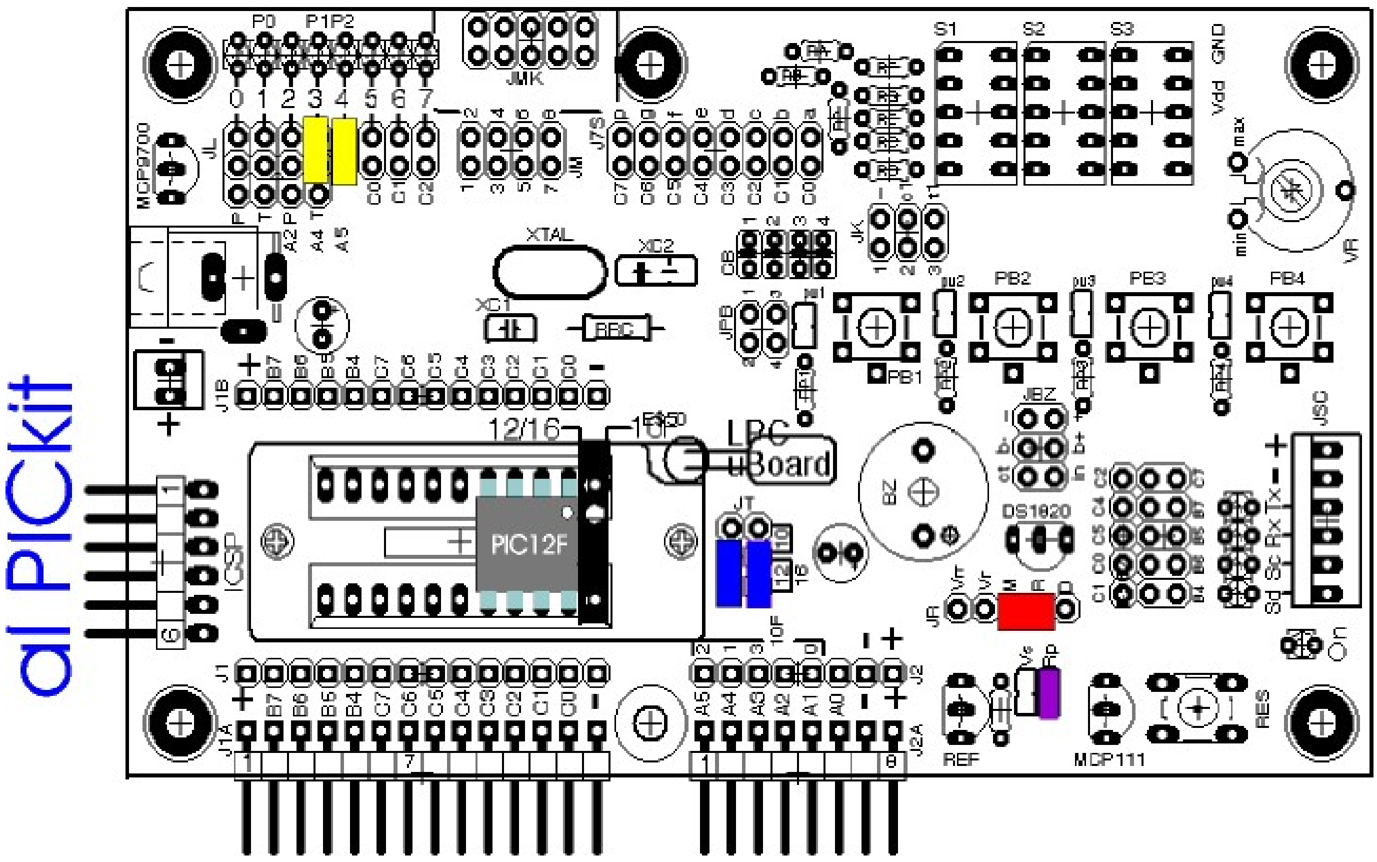
The wiring diagram is that of exercise 3A.
It may seem odd that there are no other buttons besides Reset. The reason is simple: this is used because the **GP3** pin is programmed not as **an MCLR**, but as a **digital input**.

In Baselines, as in most PICs, the pin with **MCLR** function can alternatively be configured as a digital input. This is possible because the Reset function when the supply voltage arrives is carried out completely by an internal circuitry, without any need for MCLR and what is connected externally to the pin, assuming, in this case, the function of **GP3**. Thus, the R4 keeps the digital input pin at a high level, ensuring a logic level of 1; the button, when closed, connects **GP3** to the Vss (low level, i.e. 0).
The R4 is essential because if it were missing, no voltage would be applied to the pin except that due to the disturbances picked up by the connections. This would be precisely the condition of uncertainty that we must absolutely avoid: even with open contact, a precise voltage must be applied to the pin. In this case, the resistance can vary from 1 to 100k: the lower its value, the greater the voltage applied to the pin and therefore the greater what is called "noise rejection", but the greater the current circulating when the contact is closed. On the other hand, by increasing the resistance, the current is reduced when the contact is closed, but also the rejection of disturbances; the 10k typical of the external Reset circuit is fine.

The R3, in the external reset circuit, has the function of limiting the current lost when the Vpp is applied in on-circuit programming (since MCLR also shares the input function of the programming voltage); When used as an input, it is still necessary to reduce the current due to the possible presence of static electricity, impressed upon contact of the circuit with the user. Its value is typically between 100ohm and 4k7. The value 1k usually used is adequate. This resistor is not mandatory, but it should be inserted in any case to increase the safety of the circuit.

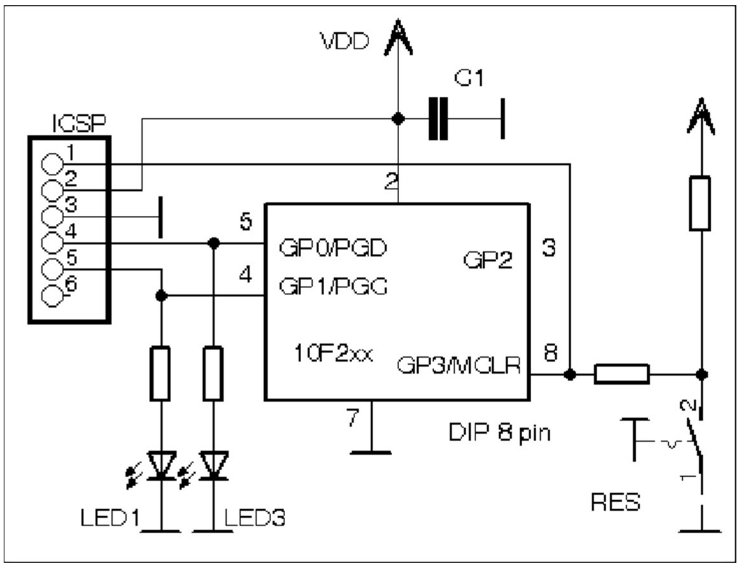
These links are already present on demo boards, while if we use a breadboard they must be done once and for all, since this basic scheme will be used in most of the other tutorials.

The same connections as in Tutorial 3A are valid for LPCuB

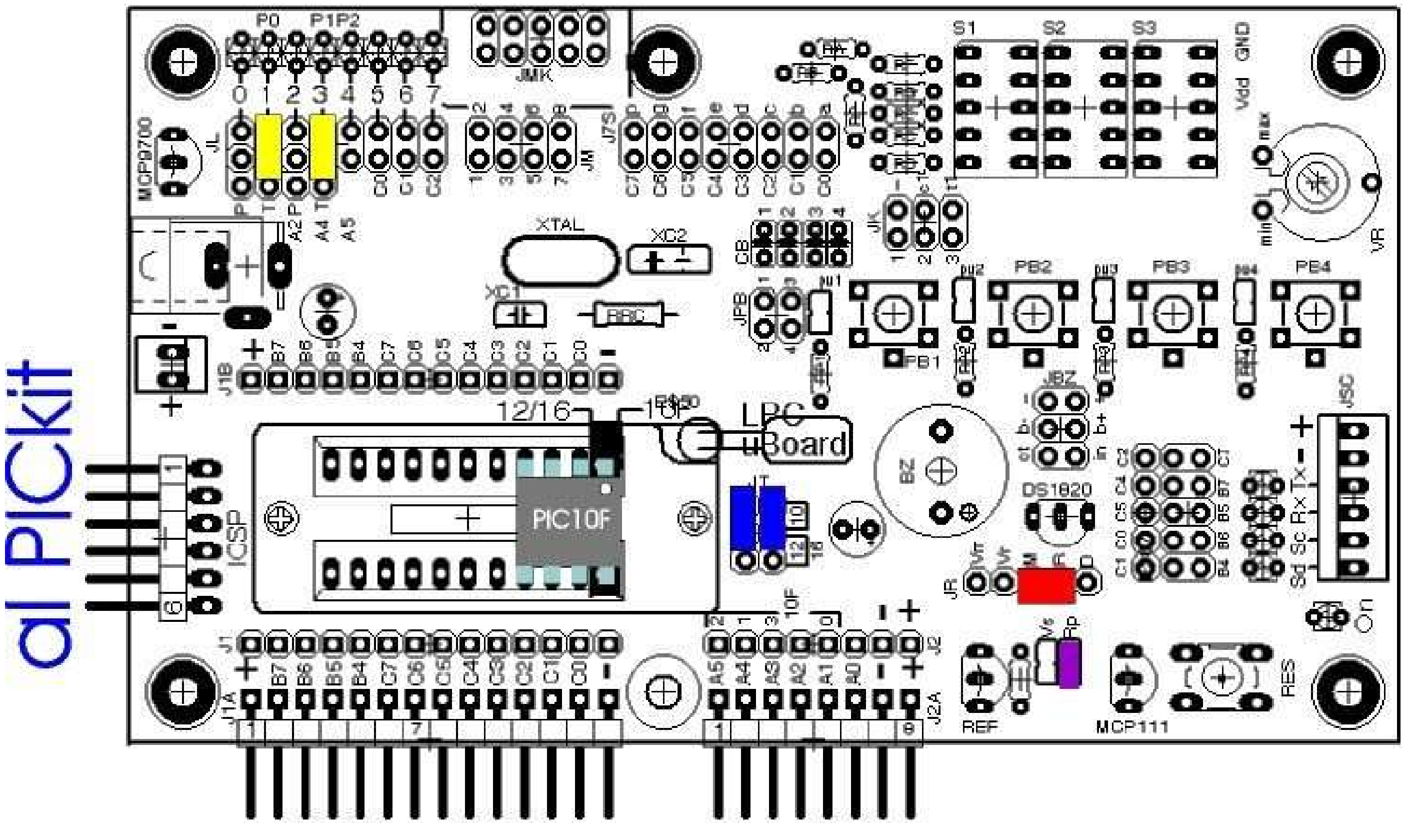


Let's look at the position of the chip in the socket.
The "blue" jumpers choose the ICSP connections for the installed micro.
The "yellow" jumpers connect the LEDs to the **`GP4`** and **`GP5 output pins`**.
The "red" and "purple" jumpers connect the **RES** button and its pull-up.

We can repeat the experience for **PIC10F20x** as well, where we will use **GP0/GP1** for the LEDs.



For the connections on the board, remember to observe the position of the chip on the socket and that of the "blue" jumpers for choosing the right chip.



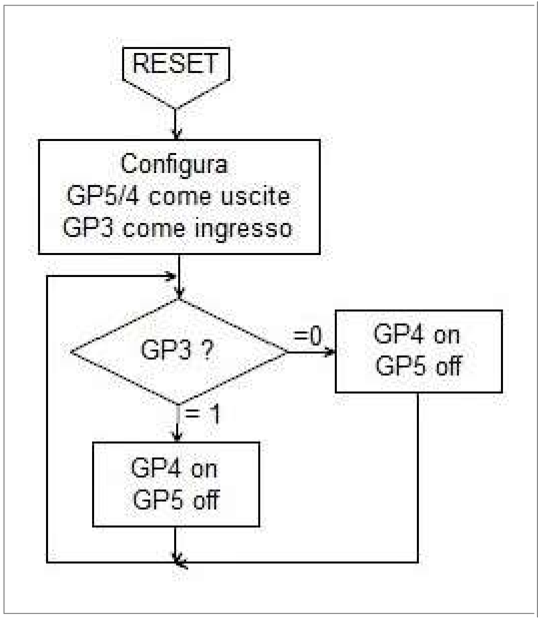
## The source

As always, the first thing is to be clear about what you want to achieve:

- **make one LED light up when the button is pressed (closed) and another when the button is released (open), turning off the first LED; and vice versa.
  The LEDs are connected to GP4 and GP5 respectively, the button to GP3.**

And how:

- **programming pins GP4 and 5 as digital outputs and GP3 as digital input, then analyzing the status of GP3 and acting on the LEDs accordingly.**



Let's draw the usual and indispensable flowchart that shows in graphic form what will have to be executed by the program.

After configuring the digital I/O pins appropriately, we check the status of the GP3 input pin.

Depending on its logic level, we control one or the other of the two LEDs when switched on.

Also in this example the source is made available already fully written, in the **7A_12F5xx.asm file** for 12F519/508/509 and **7A_10F20x.asm** for 10F200/2/4/6.

Let's make a detailed analysis, in order to be clear about all its components and the reason for the various choices.
But first, a necessary clarification.

## GP3 or MCLR ?

Let's take a look at the diagram of the connections seen above and further comment on the choice to use the RES button.

For those who might be perplexed, it should be remembered that in practice there is no foreclosure, since:

- the reset of the microcontroller when the power supply voltage arrives (**POR** - ***Power On Reset***) does not depend, as already mentioned, on **`the MCLR`** pin, but on the internal circuitry that verifies the level of the Vdd without the need for external inputs,
- so much so that the **`MCLR`** pin can be dedicated through the config, to the **`GP3 digital function`**.

The option is established in the initial config with the label:

```
    __config _MCLRE_OFF
```

whose function is to set the **GP3/MCLR** pin as **GP3** and not as **MCLR**.

It should always be kept in mind that the MCLR function may well not be necessary for the application and therefore the corresponding pin is used as digital I/O. This is very interesting in chips with a low pin count.

Keep in mind that:

1. the pin that shares the MCLR function can be used as a digital input, **but not as an output**
2. The choice made with the configuration cannot be changed by program.

So, having already connected the button to the **GP3** from the previous exercises, it becomes convenient to use it as an input. When more inputs are needed, more pins and buttons will be involved.

---

Let's configure the **TRISGPIO direction register** in such a way that we have the following situation

| GPIO | GP5 | GP4 | GP3 | GP2 | GP1 | GP0 |
|---|---|---|---|---|---|---|
| Digital Function | **Out** | **Out** | **in** | **-** | **-** | **-** |
| **TRIS** | **0** | **0** | **1** | **-** | **-** | **-** |

This is possible since GPs **are logically associated in a single GPIO register**, but **they can be used independently of each other**. The general rule for PCIs is this:

| Direction | Bits in Tic-Tac-Toe |
|---|---|
| **Entrance** | **1** |
| **Exit** | **0** |

That is, if I bring a bit of the TRIS register from the program to level 1, the corresponding pin will be configured as an input; if it is at 0 it will be an output.

Note that **at the POR the bits of the TRIS register are configured as inputs (bit logic level = 1).**
**So you only need to change them if you want to use them as outputs.**

We don't use **GP0/1/2** pins in this application, so they can be left in the default state, since they are not involved in any program operations (<u>although this is not the best choice</u>).

We remind you that the **TRIS** register in the Baselines is not accessible in direct read and write and must be loaded only through the **TRIS** statement. So, for a chip in a 14-pin package, we have two PORTs of 6 bits each:

```
    movlw    b'00110000'       ; 5:4 bits = 1 bit3:0 = 0
    TRIS     PORTB             ; pin RB5:4 = in RB3:0 = out
    movlw    b'00110111'       ; bit 3 = 0 at atria = 1
    TRIS     DOOR              ; pin RA3 = out, all others = in
```

We also note that bits 6 and 7 of the register are not implemented, as the chip only makes 6 I/O pins available; Then, any value can be assigned to these bits without any effect. So, the above could also be written like this:

```
    movlw    b'11110000'       ; bit 5:4 = 1 bit3:0 = 0
    TRIS     PORTB             ; pin RB5:4 = in RB3:0 = out all
    movlw    b'11110111'       ; bit 3 = 0  atria = 1
    TRIS     DOOR              ; pin RA3 = out, all others = in
```

or like this:

```
    movlw    b'110000'         ; bit 5:4 = 1 bit3:0 = 0
    TRIS     PORTB             ; pin RB5:4 = in RB3:0 = out
    movlw    b'110111'         ; bit 3 = 0  all atria = 1
    TRIS     DOOR              ; pin RA3 = out, all others = in
```

Although it is recommended that you always use 8-bit

binary forms. A few clarifications on the matter:
The GPIO logic group belongs to the RAM area, which is accessible from the CPU with an 8-bit bus; so the GPIO register has the ability to be 8 bits wide, but, since there are only 6 I/O, it only handles 6 bits.
The two highest bits (bit7:6) have no function: they are **unimplemented bits**. We

have, in our case as an example, this situation:

| GPIO | GP7 | GP6 | GP5 | GP4 | GP3 | GP2 | GP1 | GP0 |
|---|---|---|---|---|---|---|---|---|
| Digital Function | x | **x** | **Out** | **Out** | **in** | **-** | **-** | **-** |
| **TRIS** | **x** | **x** | **0** | **0** | **1** | **-** | **-** | **-** |

x = bit not implemented in the chip
- = bit not used by the program

Let's open a parenthesis to clarify the difference between unimplemented bits and unused bits.

## Unimplemented bits vs. unused bits.

In this case:

- **bits 7 and 6 are not implemented**
- **bits 0, 1, and 2 are not used**

The difference is this:

- **Not implemented** means that there are no physical hardware counterparts dependent on these bits: reads and writes of these bits have no effect.
  Therefore, these are of no importance in the operation of the microcontroller and can simply be overlooked or written with any value (in reading they normally yield 0).

- **Not used** means that they are physically there, i.e. some hardware function depends on them, but they are not used in the specific program we are writing.

In the latter case, it may not matter how the bits that correspond to unused I/O pins (or functions) are configured and/or connected. This requires some knowledge of the purpose of the register bits and their default value, which are described in the component datasheet.

Until now we have not considered the problem of unused bits, leaving them at the default, but it is necessary to know that this may not be the best choice to adopt in a realization destined to come out of the walls of the Laboratory. This is because we consider only one entrance and the exercise probably takes place in a sufficiently safe environment and free from possible induced disturbances.
In general, remembering what has been said about the Microchip philosophy that assigns the conditions of least power consumption to defaults, the I/O at reset are typically set as inputs, we can simply confirm 1 in the corresponding bit of the TRIS register.
Even if we neglect it for now, it is good to be aware of this situation.
In practice, having to take a microcontroller circuit "out", it is advisable to read the information on these pages.

**Warning**: The pins of the microcontroller can share many functions, but only one can be used at a time.
Therefore, a pin can be configured either as an input or as an output, but certainly not at the same time for both functions.

In the direction register, bits at 1 represent an input function, while bits at 0 represent a digital output.

| GPIO | GP7 | GP6 | GP5 | GP4 | GP3 | GP2 | GP1 | GP0 |
|---|---|---|---|---|---|---|---|---|
| Digital Function | x | x | **Out** | **Out** | **in** | **Nu** | **Nu** | **Nu** |
| **TRISGPIO** | 0 | 0 | **0** | **0** | **1** | 1 | 1 | 1 |

Bits 6 and 7 of the TRIS register are not implemented (x), as they do not correspond to any pins. If read, they yield 0 and if written, the data has no effect. We can set them to 0:

```
; Assign the required function to the pins
            movlw     b'00001111'
            Tris      GPIO
```

but even at 1, their writing has no consequence. And we may not even consider them completely:

```
; Assign the required function to the pins
            movlw     b'001111'
            Tris      GPIO
```

It may seem problematic at first glance, but even writing `b'001111'` with only 6 bits, again, we don't create any problems: **since the data bus is 8 bits, the compiler will fill in the missing bits with 0** so, in effect, it's as if we had written `b'00001111'`.

Faced with registers that have unused bits, it is advisable, at least at the beginning, to fill in all the positions, if only to keep the awareness of what you are doing, leaving the "shortcuts" at the moment when you begin to become masters of the subject.

If we want to practice with numerical roots, we can say that `b'00001111'` is equal to 15 decimal or 0F hexadecimal. So the shapes would also be fine:

```
            movlw     .15      ; 15 decimal
; or
            movlw     0x0F     ; 0Fh
```

However, it should be noted that **expressing the number in binary form allows us to have an immediate image of the relationship between the value of the bit and its position in the byte**, as indicated in the previous table.

Solving the program would seem to be a very simple action, using the statements to test the state of a bit in a register (`btfsc`, `btfss`):

```
; Home position: LED4 on, LED5 off
```

```
ioinit       movlw    1<<GP4 | 0<<GP5       ; < -----|
             movwf    GPIO                  ;        |
; Test Button Status BTNCHECK                        |
             BTFSC    GPIO, GP3             ; -> ----+--|
; If open, do not change the situation                 | |
             Goto     ioinit                ;  ->-----| |
; if closed, swap the status of the LEDs                 |
             movlw    1<<GP5 | 0<<GP4       ; < --------|
             movwf    GPIO
; and wait for the button status change Goto
                      BTNCHECK
```

Once again we highlight the two loops on which the program is based.

The form : `movlw 1<<GP4 | 0<<GP5` indicates that we want to load in W a number in which the bit corresponding to GP4 goes to 1 and the bit corresponding to GP5 goes to 0; the correspondence between the labels and the position in the byte is given by the usual file ***processorname.inc*** included at the beginning,   Where we find:

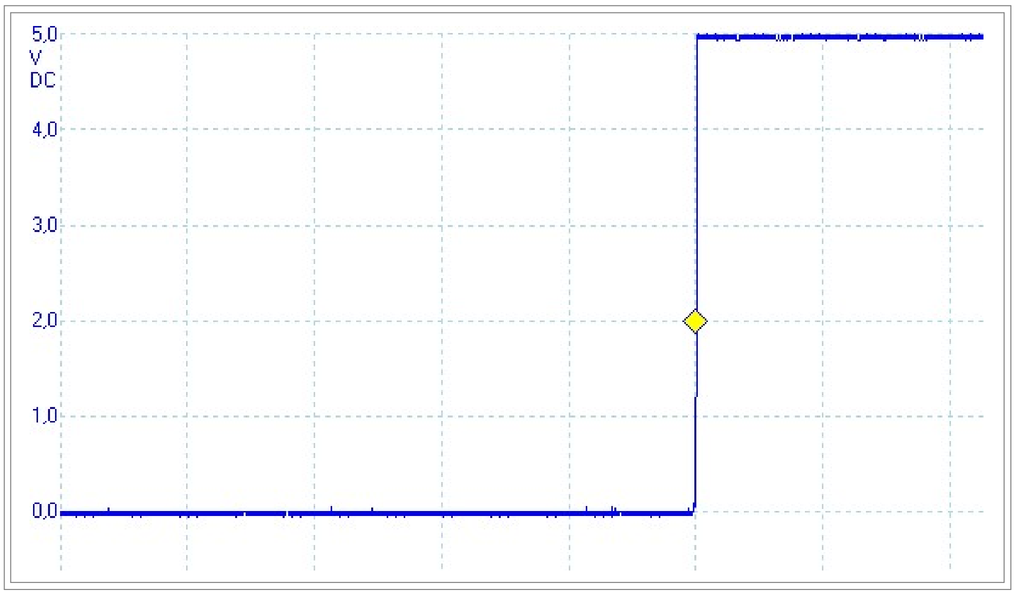
```
;----- GPIO Bits ~~EQU~~-----------------------------------------
GP0       H'0000' EQU
GP1       H'0001' EQU
GP2       H'0002' EQU
GP3       H'0003' EQU
GP4       H'0004' EQU
GP5       H'0005'
```

Which makes **`GP4`** match the number 4 and **`GP5`** the number 5.  Everything looks flawless. But does it work?

## Bounces!

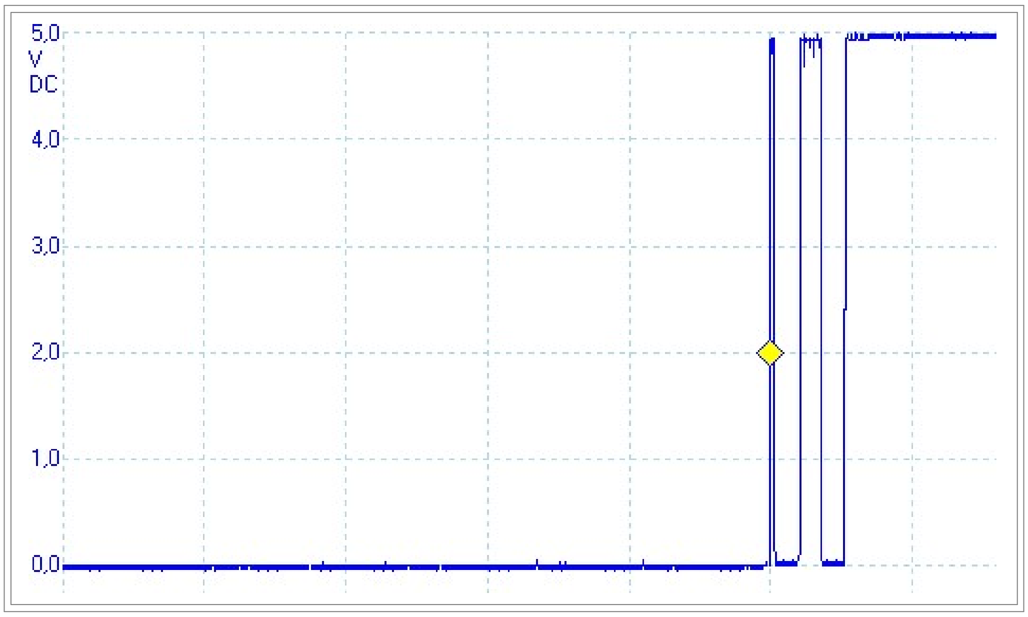
In fact, the program would be correct if it weren't for a mischievous side effect that plagues mechanical contacts: bounces.

The problem is simple: when we make a mechanical contact, the two metal parts close together and, theoretically, we should have an oscillogram like this:



with a direct transition from one level of tension to another in a "clean" front.

In reality, the metal slats of the contact, when strongly impacted, are elastic enough to give rise to a short series of micro openings and closures: bounces. Here we see two bounces before the final closure, for a time of less than 5ms:



In the opening, the phenomenon may be less pronounced, but it still has a good chance of existing.

NOTE Bounces depend on the type of contact: buttons and switches intended for use with electronic circuits will have less "dramatic" rebounds than those generated by switches intended for domestic lighting or industrial applications, which are not the case to be used directly with a microcontroller.
In addition, the rebounds will be accentuated by the state of wear of the contacts.

The component data sheet usually shows the bounce time evaluated by the manufacturer and which can vary from a few milliseconds to a few tens of ms: these are short times, but they are enormously large when compared to the micro seconds or nano seconds of the cycle of a microprocessor instruction. At a clock of 4MHz an instruction is executed in 1-2us; with a clock of 20MHz, the time is reduced to 200-400ns: in 10ms hundreds or thousands of instructions are executed! If the program is checking the status of an input and it is subject to bounces, it is very likely that one or more of them will be intercepted by the instruction flow and thus give rise to false results.

If the definition of contacts includes any kind of electromechanical device such as keyboards, switches, mechanical encoders, etc., it must be said that there are special systems without rebounds, such as optical encoders, mercury contacts (reed), Hall effect switches (magnetic) and others. Thus, for example, it is possible to replace a mechanical limit switch (microswitch) with an electronic one:

These are bounce-free, but have a higher cost than mechanical equivalents. Their use is common in the professional field, especially where they are also used as counting sensors. In the consumer environment, the choice, determined by cost, means that mechanical contacts are the most widespread; So, we are dealing with these bounces and we have to take them into account and, in general, where you are not sure what the entrance will command, it is always the case to provide a debounce software system.

In the event that the contacts are particularly "despicable" from this point of view, a hardware debounce system may be necessary, which can also be implemented when it is not possible to insert a software system. It goes without saying that the addition of external components is ill-suited to the design with microcontrollers, which tend to be implemented with a minimum of additional elements. However, the choice of one or the other anti-rebound system must always be seen depending on the application.

## Debounce.

If we follow the program written before and imagine the presence of bounces, it behaves like this:

1. The button is pressed and contacts close. The instruction `btfsc GPIO,GP3` Locate the PIN at the low level and do not skip the following instruction.
2. the movlw and following `are then executed`, which set the LEDs, and then return in loop to the button test.
3. if the microcontroller is running at 4MHz, the loop instructions are carried out in just 6 microseconds, after which it returns to the input pin check.

4. If the pin remained closed, there would be no problem until the next time it was opened, but , while keeping the button pressed, the contacts reopen and close several times before stabilizing.
5. For each opening the test switches to the `goto ioinit statement`, which changes the state of the LEDs.
   A new closure leads to the reversal of the LEDs and so on until the rebounds are exhausted

The same thing is repeated when opened: the LEDs are turned on and off several times, following the bounces.

One thing should be clear: by eye we are absolutely not aware of this turning on and off of the LEDs, since the bounces are contained in such a short time that the eye does not notice what is happening. But if the outputs were connected to a meter, to relays that control loads, to electronic circuits, disastrous situations would occur for the operating logic of the machine. For example, a counter would advance with each closing-opening of the input contact, counting the bounces as pulses and making the count incorrect.

It is therefore necessary to adopt a debounce action that allows the microcontroller to interpret a close and an open as a single event, even if affected by bounces.

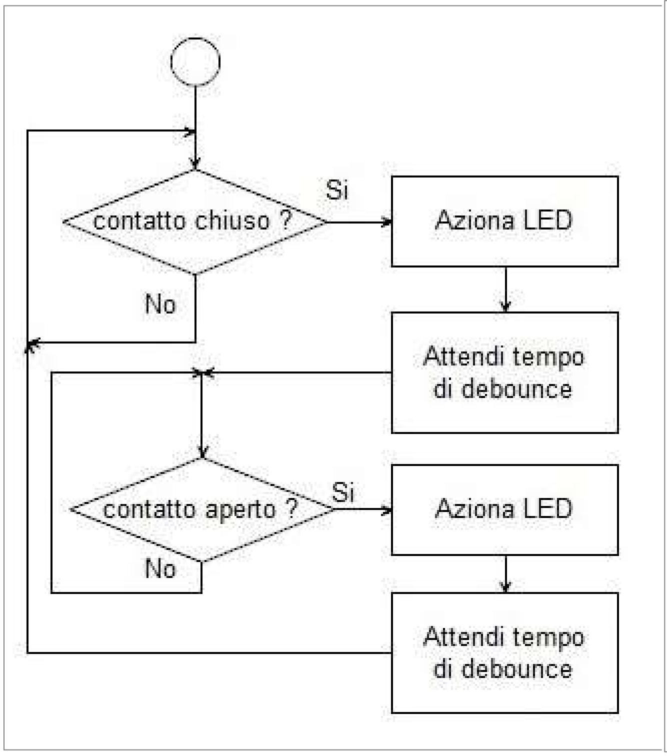
There are many hardware solutions, but all of them involve the addition of components and this increases the complication and cost of the circuit, while, if not well applied, they are useless.
We can usually do without it, by making a suitable software. The
simplest logical structure you can implement is this:

- Once a change in the state of the input has been detected, wait for a time to exclude the capture of bounces.

Obviously, the program, in itself simple, is complicated by the insertion of the necessary instructions, even if the use of subroutines and/or macros does not make this need weigh too much on the programmer. In any case, from the point of view of execution time, whether there is an external debouncer or the software one, it is in any case necessary to consider a waiting time around the contact switches.

Let's suppose, for example, that the datasheet of the switch we want to use indicates a maximum bounce time of less than 40ms.
This means that, once the contact has been activated, it will be necessary to wait 40ms for the mechanical oscillations to switch off before declaring the contact open or closed.

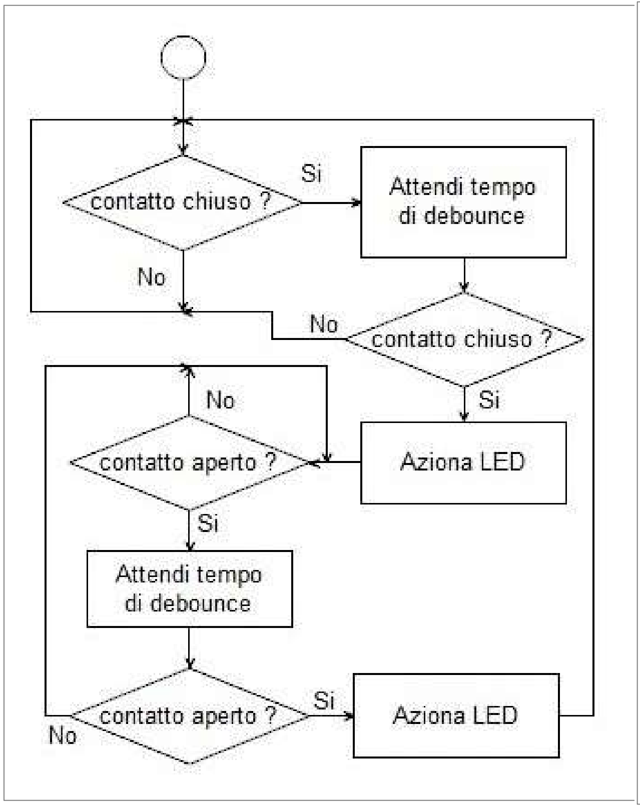
So let's proceed according to this flow chart:



1. We are waiting for the contact to be closed.
2. As soon as it is closed, we activate the LEds accordingly
3. Now we wait for a time to run out of bounces
4. We now test the opening of the contact
5. As soon as it has opened, we activate the LEDs accordingly
6. And we wait for the time of exhaustion of rebounds
7. Let's close the loop by going back to step 1

The anti-bounce waiting time will depend on the characteristics of the contact.

We observe that the action on the LEDs takes place as soon as the change in the state of the input pin is detected, even if there are supposed to be bounces; but you don't have to wait until the end of these to switch the LEDs.
Instead, you must wait for the debounce time before checking for a new change in the contact's status. In other situations, where it is possible that disturbances arrive on the input such that the integrated logic perceives a switch, the debounce procedure may take on a different appearance.
Let's look at a possible flowchart:



1. We are waiting for the contact to be closed.
2. As soon as it closes, we start a debounce time count
3. At the end, we check if the contact is really closed.
   If it isn't, it was a disorder and let's go back to point 1
4. If, at the end of the debounce, the contact is closed, we activate the LEDs
5. We now test the opening of the contact
6. As soon as it opens, we start a debounce time count
7. At the end, we check if the contact is really open.
   If it isn't, it was a disorder and let's go back to point 5
8. If the contact is open at the end of the debounce, we activate the LEDs

9. So let's go back to point 1

Again, the anti-bounce waiting time will depend on the characteristics of the contact.
Other anti-bounce algorithms can be implemented depending on the need to save time or improve security.

## Let's enter the software debounce

Let's change the source accordingly:

```
; Home Position (Button Open)
            LEDOpen

; Button Hold Closed
buttonloop BTFSC    GPIO,GP3
             Goto    buttonloop
; button closed, LED actuates
            LEDClose
; debounce
            Call     debouncetime

; Hold open button BTLA
            BTFSS    GPIO,GP3
             Goto    BTLA

; button open, LED actuates
            LEDOpen
; debounce
            Call     debouncetime
; Another loop
            Goto     buttontloop
```

The time routine is of the kind seen above. Since bounce times vary between a couple and a few tens of milliseconds, we can write a routine modulated by W x 500us, which can then time up to 125ms:

```
; Delay = 500us
; Clock frequency = 4 MHz 500 cycles
Delay05ms                     ; 496
        movlw    0xA5
        movwf    D1
Dly05ms
        decfsz   d1, f
         goto    Dly05ms
        retlw    0        ; 4 cycles (including
```

As far as the operation of the two LEDs is concerned, we always use an R-M-W proof structure: for convenience, we create two simple macros that set the LEDs the way you want in the shadow and then copy it into the **`GPIO.`**

```
; LED status with open button
LEDOpen    MACRO
           Bsf    sGPIO, GP4
           Bcf    sGPIO, GP5
           movf sGPIO, W
           movwf GPIO
           ENDM

; LED status with button closed
LEDClose    MACRO
           Bcf    sGPIO, GP4
           Bsf    sGPIO, GP5
           movf sGPIO, W
           movwf GPIO
           ENDM
```

The proposed approach is very simple; It is possible to resort to more sophisticated software debugging systems, with the use of vertical counters or interrupts, but for now the solution adopted is adequate.

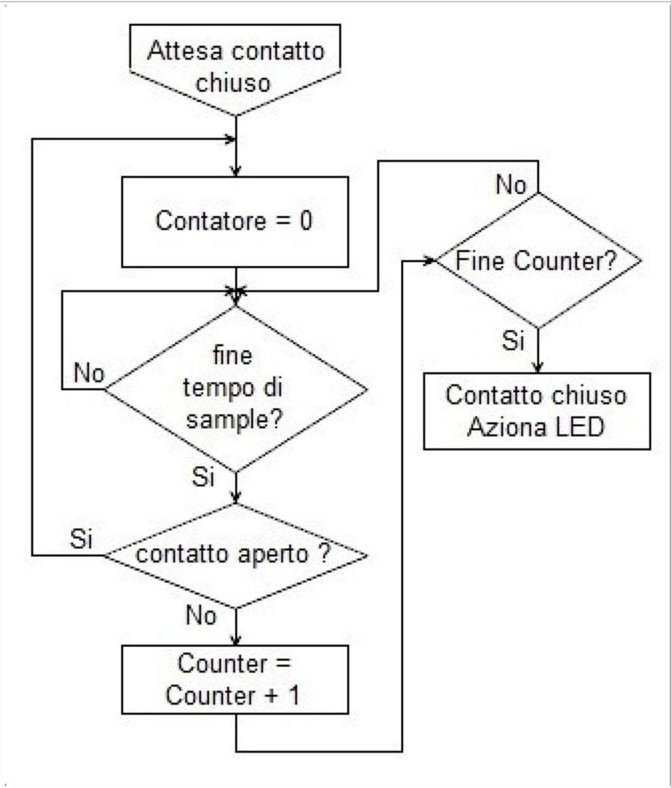
Find the usual versions for 12F519/508/509 and 10F200/202.

## Debounce with counting algorithm

There can be problems in using a fixed debounce time: for example, you have calibrated the time to a certain contact, but then it is not available and you use one with a different bounce time.
It may also happen that you do not have information on the bounce time of the contact you are using and therefore you have to resort to setting debounce waits much longer than necessary, with a penalty for the execution of the program or, on the contrary, risk finding yourself with too short a time.

A different approach may be to repeatedly sample the state of the contact and define it as stable when it has been stable for some time. If the state is not maintained for a sufficient time, it is considered to originate from an impulsive disturbance (glitch) generated by EMI or induced electrical noise.

We can sample the state of the input at a fairly fast rate, on the order of milliseconds, and accept its state only if it persists for at least a dozen or more samples.



For example, here's the flowchart for closing the contact.

1. When we are waiting for the closed contact, we reset a counter and set a sample time
2. At the end of the sampling, we check the status of the contact:
   - If it's still open, we repeat the sampling
   - If it's closed, we increment the counter
3. Repeat this loop until the counter is exhausted with the contact closed. We can then operate the LEDs

If the contact is open before the meter runs out, we consider the closure "accidental" and start sampling again from 0.

The algorithm becomes much more complex than the previous one, but it is not difficult to implement.

For simplicity, we use a sampling with the use of a RAM location **sample_cnt** . Being 8 bits wide, it will be able to count up to 255d, which, at a clock of 4MHz, generates a time of about 768us. As far as the loop of the main counter is concerned, we can define a number of repetitions of the sampling such as to guarantee the exclusion of bounces. Since the sample time is 769us, to get a total debounce time of 20ms we need to make 25 repetitions, a parameter that we define at the beginning of the programme with the directive **Constant** .

## CONSTANT and EQU.

When you want to define a numeric constant, the MPASM Assembler offers a few possibilities:

- Use the **Constant directive**
- Using the **Equitable Directive**

**constant** , as the name implies, is intended to define a constant. It has the following shape:

| | | | | | |
|---|---|---|---|---|---|
| sp | Constant | sp | Label value | sp | [; comment] |

The label in question is defined and the indicated value is associated with it. So, in our case, we will write:

```
Constant    mainstep = .25        ; Number of sampling repetitions
```

In the case of the **equi directive**, which we have already encountered, the

| | | | | | | |
|---|---|---|---|---|---|---|
| Label | sp | Equ | sp | Value | sp | [; comment] |

So we should To

```
mainstep Equ .25           ; Number of sampling repetitions
```

It should be noted that in the first case the directive cannot start in the first column, while in the second the label **must** start in the first column to define itself.

The two notations are completely equivalent and can be used according to your taste.
`Equ` is more commonly used and found in more examples.
`constant` is less commonly used, but may be more suitable for defining labels associated with numeric values rather than in-memory addresses.

Note for any perplexed:

NOTE As mentioned above, **MPASM** accepts both uppercase and lowercase directive names, so writing `equ` rather than `EQU` or `CONSTANT` rather than `constant` is completely equivalent, and the two forms can be used depending on your taste.

We can define the input pin with a label:

```
#define     Button     GPIO,3    ; GP3 as Push Button Input
```

We remind you once again that an intensive use of labels allows two essential advantages:

- makes the source more comprehensible, therefore easier to reread, correct and maintain
- It allows you to define a specific element in a single point of the source which, having to be changed later, will be changed in only one point of the text, avoiding errors and saving time and work.

Let's insert the two debounce loops into the source:

```
; Closed Button Test (Low Level)
; Reset the main counter and the sample counter
CloseCheck Crlf     main_cnt
            CLRF     sample_cnt
; Wait for the contact to close
Closeloop   incfsz sample_cnt,f ; Sampling time
```

```
             Goto     Closeloop
            BTFSC     Button          ; Contact closed?
             Goto     CloseCheck      ; No - Recycle
            incf      main_cnt,f      ; Yes - Increase Main Counter
            movlw     mainstep        ; End Counter ?
            XORWF     main_cnt,w
            skpz                      ; Yes - Action
             Goto     Closeloop       ; No - Other Sampling

; LED actuation for closed contact
            LEDClose
```

Let's look at the use of two unseen statements: **incfsz** and **xorwf** .

---

## INCFSZ

We've already seen opcode, of which it's an extension. The syntax is as usual:

| [label] | sp | incfsz | sp | Object | ,f/w | sp | [; comment] |
|---|---|---|---|---|---|---|---|

The opcode stands for ***increment*** *file and* ***skip*** *nex istruction if result is* ***zero;*** The file in question is incremented by one unit each time the instruction is repeated, and when the increment brings the contents of the file to 0, the following operation is skipped.
In the binary count, once the contents of the file reach **FFh** (all bits to 1), i.e. the maximum value that can be counted in 8 bits, they are reset by the next increment (**FFh + 1 -> 00h**) and the internal logic indicates the overflow of the count with the jump of the next opcode.
This instruction finds a typical place in loops like this:

```
Counterloop    incfsz counter,f         ; Increment Counter
                Goto    Counterloop      ; If non-zero, repeat
increment
End of Count ....                         ; If = 0 continue
```

As we have seen for other instructions, this also requires a "tail" that indicates where the result of the increase should be stored, with the same characteristics and problems already described.

In our case:

```
; Wait for the contact to close
Closeloop    incfsz sample_cnt,f ; Sampling time Goto
                    Closeloop
             BTFSC    Button          ; Contact closed?
```

The **sample_cnt** register is incremented at each step (the result of the increment is stored in the file itself); when it reaches zero, the **goto** is skipped and the next **btfsc statement is passed**.
Since the counter is reset before the loop (and therefore contains 0), it takes 256 increments to reset it to zero.
In counting the loops that make up the loop, it must be considered that each step involves the **incfsz instruction** and the subsequent **goto,** until 0 is reached. At this point the **incfsz** skips the **goto** and this requires an extra cycle:

```
Closeloop   incfsz  sample_cnt,f    ; 1 cycle (+1)
             Goto   Closeloop       ; Two cycles
```

So the total is: *(256 x 3) +1 = 769us*

## XORWF

This instruction performs an **exclusive OR** (XOR ) between the contents of the WREG and that of a register referred to in question. The syntax is as usual:

| [label] | sp | XORWF | sp | Object | ,f/w | sp | [; comment] |
|---|---|---|---|---|---|---|---|

**XOR** has the following truth table:

| B | AT | A^B |
|---|---|---|
| **0** | **0** | **0** |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| **1** | **1** | **0** |

From a verbal point of view, it could be expressed as "the result is True if A or B are different". Basically, if you XOR a bit with 1, the result is the opposite value of the bit. By XORing a register with a numeric value, the result is 0 if the register contains the same value. This Boolean operation is critical in many situations; You can find a more in-depth discussion here.

In our case:

```
  movlw    mainstep        ; loads in W the desired value for the counter
  XORWF    main_cnt,w      ; Compare with the contents of the counter
  skpz                     ; are the same - Skip
   Goto    Closeloop       ; No - Other Sampling
```

Note that we perform the XOR with the result in W ( **,w** ) and not in the counter file so as not to change the value; in fact, if we had indicated the "tail" **,f** the result of the XOR would be replaced the value contained in the counter, modifying it and preventing the correct counting.

## SKPZ

Once the XOR has **been executed**, we verify the equality by testing the Z (Zero) bit of the STATUS. This bit goes to level 1 when the result of an operation is zero. Therefore:

```
       movlw    mainstep       ; loads in W the desired value for
the counter
       XORWF    main_cnt,w     ; Compare with the contents of the
                               counter
       BTFSS    STATUS,Z       ; are the same - Skip
        Goto    Closeloop      ; No - Other Sampling
```

It is the most formal writing. However, MPASM provides a series of pseudo-opcodes to simplify the work of the programmer, especially if it comes from assemblers of other processors (Z80, Motorola, etc.).
One of them is **skpz** - ***skip*** *next instruction if result is* ***zero***, which is equivalent to

```
btfss STATUS,Z
```

However, pseudo opcodes are easy to read, since they briefly indicate the operation performed.
The use of one form or the other is a personal matter, although it is possible that other assemblers other than MPASM do not support these pseudo-opcodes.
The important thing is not to resort to the use of absolutes, which, we keep repeating, is the worst thing a programmer can do. So, even if at the compilation the line:

```
BTFSS 03.2
```

It provides the same binary code, it is to be avoided absolutely. What is Register 02? What is the function of bit 2? The line is not self-explanatory, it is not portable, it does not benefit compilation at all; It only creates a point of doubt that needs a clear comment to be understood. Use the "normal" form or pseudo-opcodes, as you prefer, but avoid the use of absolutes.

For the contact opening phase, we repeat the same procedure, but refer it to level 1 of the **GP3**:

```
; Test for open button (high level)
; Reset the main counter and the sample counter
opencheck    Crlf     main_cnt
             CLRF     sample_cnt
; Wait for the contact to open
openloop     incfsz   sample_cnt,f   ; Sampling time
              Goto    openloop
             BTFSS    Button         ; Open contact?
              Goto    opencheck      ; No - Recycle
```

```
            incf     main_cnt,f      ; Yes - Increase Main Counter
            movlw    mainstep        ; End Counter ?
            XORWF    main_cnt,w
            skpz                     ; Yes - Action
             Goto    openloop        ; No - Other Sampling

; LED actuation for open contact
            LEDOpen
```

Apart from the label renewal, the "highlight" is in the `btfss GPIO,3` line, where we check the high level of the bit. We can combine the two routines:

```
; Closed Button Test (Low Level)
; Reset the main counter and the sample counter
CloseCheck Crlf      main_cnt
            CLRF     sample_cnt
; Wait for the contact to close
Closeloop   incfsz sample_cnt,f ; Sampling time Goto
                     Closeloop
            BTFSC    Button          ; Contact closed?
             Goto    CloseCheck      ; No - Recycle
            incf     main_cnt,f      ; Yes - Increase Main Counter
            movlw    mainstep        ; End Counter ?
            XORWF    main_cnt,w
            skpz                     ; Yes - Action
             Goto     Closeloop      ; No - Other Sampling

; LED actuation for closed
            contact LEDClose
; Test for open button (high level)
; Reset the main counter and the sample counter
opencheck   Crlf     main_cnt
            CLRF     sample_cnt
; Wait for the contact to open
openloop    incfsz sample_cnt,f ; Sampling time Goto
                     openloop
            BTFSS    Button          ; Open contact?
             Goto    opencheck       ; No - Recycle
            incf     main_cnt,f      ; Yes - Increase Main Counter
            movlw    mainstep        ; End Counter ?
            XORWF    main_cnt,w
            skpz                     ; Yes - Action
             Goto    openloop        ; No - Other Sampling

; LED actuation for open contact
            LEDOpen
```

You can find this version in the 12F5xx_2 and 10F20x_2 folders

## The advantage of labels

We would like to reiterate once again the use of labels.
It may be that the advantage has not escaped, but for those who have not noticed, let's try to write more absolutes: let's assume that we have not defined either the limit value of the main **mainstep** counter, or the input pin **Button**:

```
; Closed Button Test (Low Level)
; Reset the main counter and the sample counter
CloseCheck Crlf      main_cnt
            CLRF     sample_cnt
; Wait for the contact to close
Closeloop   incfsz sample_cnt,f ; Sampling time Goto
                     Closeloop
            BTFSC    GPIO,3         ; Contact closed?
             Goto    CloseCheck     ; No - Recycle
            incf     main_cnt,f     ; Yes - Increase Main Counter
            movlw    .25            ; End Counter ?
            XORWF    main_cnt,w
            skpz                    ; Yes - Action
             Goto    Closeloop      ; No - Other Sampling
; LED actuation for closed
            contact LEDClose

; Test for open button (high level)
; Reset the main counter and the sample counter
opencheck   Crlf     main_cnt
            CLRF     sample_cnt
; Wait for the contact to open
openloop    incfsz sample_cnt,f ; Sampling time Goto
                     openloop
            BTFSS    GPIO,3         ; Open contact?
             Goto    opencheck      ; No - Recycle
            incf     main_cnt,f     ; Yes - Increase Main Counter
            movlw    .25            ; End Counter ?
            XORWF    main_cnt,w
            skpz                    ; Yes - Action
             Goto    openloop       ; No - Other Sampling

; LED actuation for open contact
            LEDOpen
```

This code, although completely identical in result to the previous one, is decidedly less effective.

1. In the event that you have to change the number of cycles counted in the **main_cnt** , you must manually change the value every time it occurs in the source (in this case 2), with the risk of errors or forgetfulness
2. If you want to apply the button to another pin, you must manually change the value every time it appears in the source (in this case 2), with the risk of errors or forgetfulness

On the other hand, if you have declared the two values as labels, in case you want to change them, you will only need one action in the declaration line. For example, if I want to change the number of cycles to 10, it will be enough:

```
 Constant    mainstep = .10          ; Number of sampling repetitions
```

and wanting to use GP2 as input:

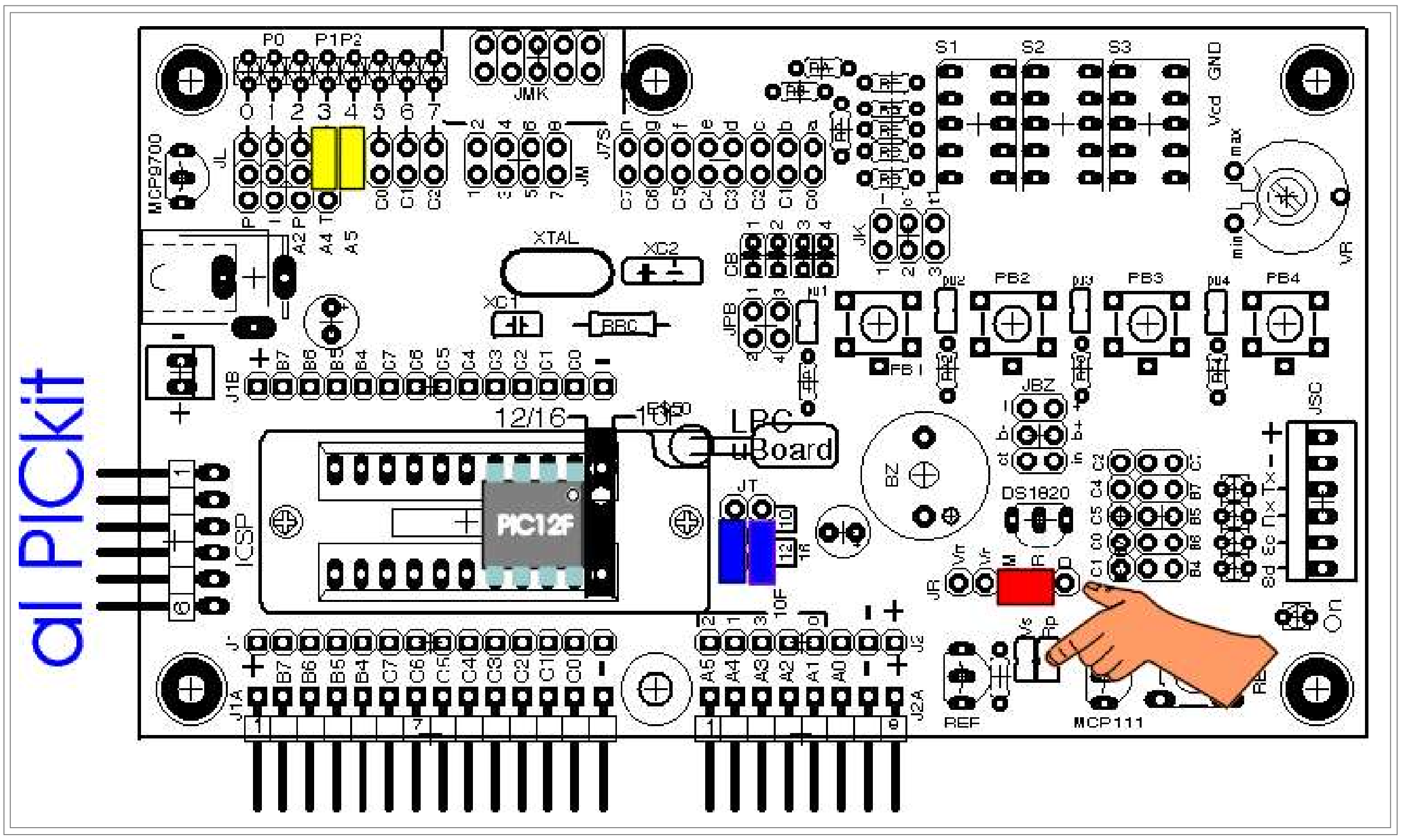
```
#define     Button GPIO,2            ; GP2 as pusant input
```

The Assembler will automatically update the labels with the right values, without errors or forgetfulness.

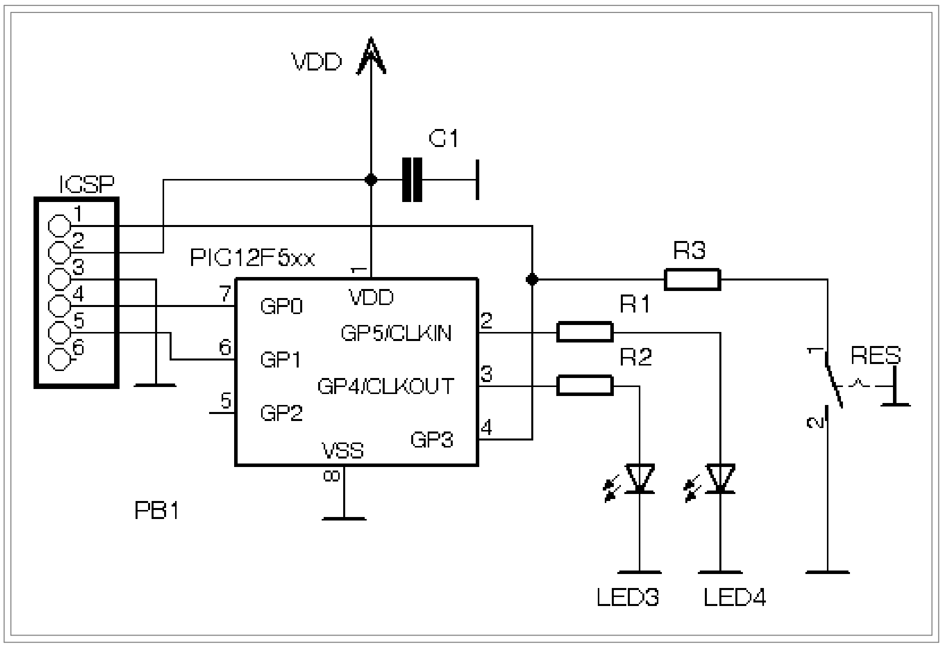
So, we repeat again that the use of labels must be pre-eminent over the use of absolute values to allow the writing of effective, readable and easily maintainable sources.

## PULL-UP of the inputs.

We can carry out an experiment: from the **LPCuB** we eliminate the 10k pull-up on the button, removing the Rp jumper:



and getting this circuit:



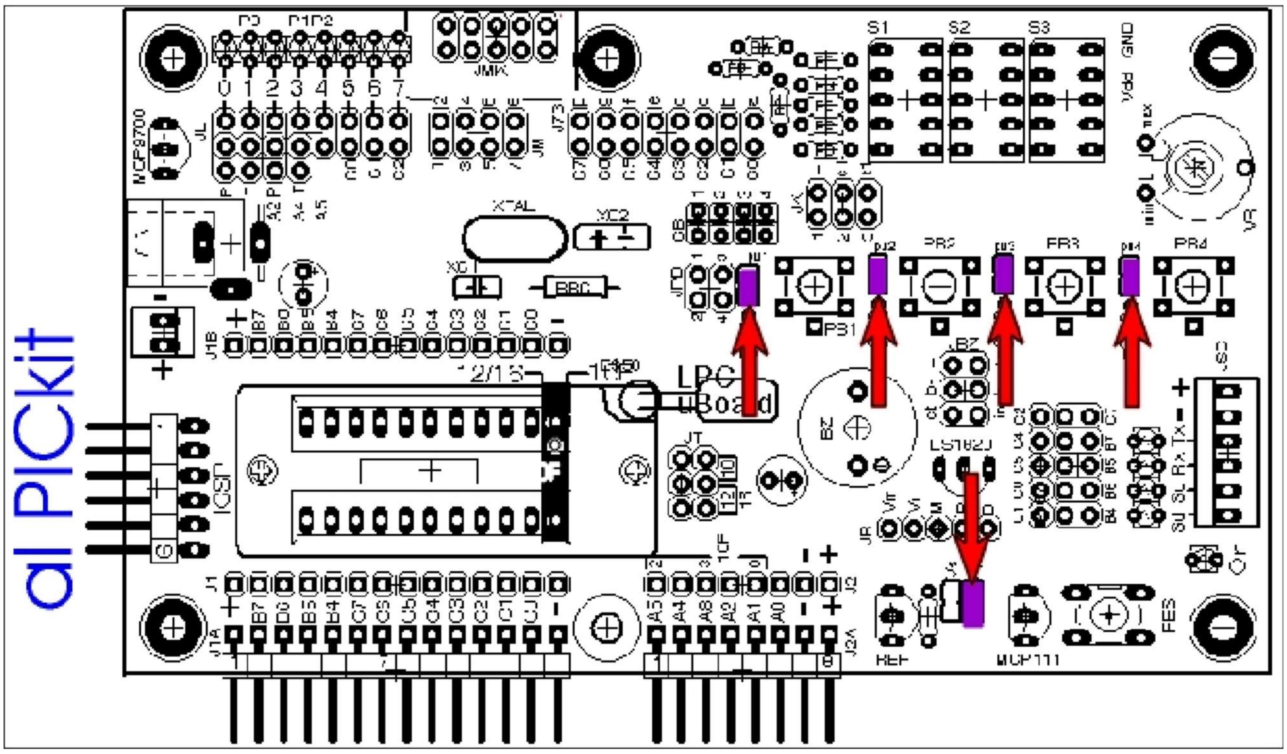
If you're working on a breadboard, just pull out the pull-up resistance on the reset button.

When we repeat the exercise under these conditions, **we observe that the button does not seem to have action or at most has random actions**.

This should be obvious from what I said at the beginning: the input pin has no bias and is high impedance. If we do not apply a pull-up (or a pull-dow if the button is connected to the Vdd) we find ourselves, when the contact is open, in a *floating condition*, where the voltage at the pin is completely dependent on the induced disturbances. Certainly when the contact is closed, the electrical level is 0, but while it is open, its value is indefinite.

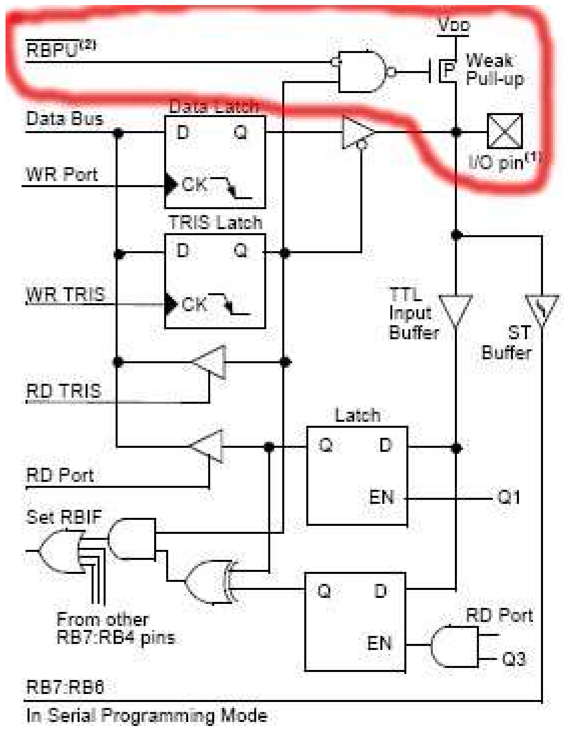
By restoring the resistance (Rp jumper), the circuit resumes normal operation. So is the pull-up mandatory? Yes, of course, but there is an alternative to external resistance.

For the **PB1,2,3,4** buttons of the **LPCuB** the pull ups can be disabled by extracting the **pu1,2,3,4 jumpers**:



# Weak pull-ups

Adding a pull-up resistance, no matter how cheap and small it is, is always an addition; Since having input contacts is a very common situation (buttons, keypads, limit switches, etc.), manufacturers have included in the chip the possibility of activating integrated pull-ups.



In **PICs,** usually PORTB **pins or** GPIOs have this option. An **`RBPU`** bit (or **`GPPU`** or similar) allows you to enable or disable the pull-up.

The built-in pull-ups give rise to a current between 50 and 400 microamperes, with Vdd of 5V and the pin grounded.
This means that the resistive value of the integrated pull-ups ranges from 12.5 to 100 kohms.

Note that these weak pull-ups are not resistors, but are made with **high RDON MOSFETs.** The dispersion of the construction parameters (the resistance varies with the Vds, the manufacturing process and the temperature) means that their real value is defined in a fairly wide range.

The built-in pull-ups are mainly designed to support, without external components, keys or keyboards; since the chip also has a function of automatic identification of the pin level change and that this condition allows the exit from the sleep state (which we will describe in another tutorial), they are called ***Weak Pull-up.*** You can find more information about pull-ups here.

There are a few limitations of these built-in pull-ups to consider:

1. they are present only on some pins and this characteristic is not uniform in all chips (for example, 12F508/509 on GP0, 1, 3, but not on GP2, 4, 5. So, if you need pull-ups on these pins, you need to add them externally.
2. In general, in Baselines they are all enabled or disabled at the same time, since there is no possibility of a command on the single bit.
3. They have a high value that may not be suitable to act as a load for open collectors/open drains that control the input and in case you need a higher immunity to disturbances.
4. They can (rarely) be incompatible with signals from other integrated circuits. Due to their

high value, they are well suited for low-power applications.

It should also be noted that, by enabling pull-ups, they will only be pulled up on pins configured in TRIS as inputs, while they are excluded from pins that are configured as outputs (where, of course, they would have no function).

In Baselines, the control bit of the built-in pull-ups can be found in the **`OPTION register`**:

| W-1 | W-1 | W-1 | W-1 | W-1 | W-1 | W-1 | W-1 |
|---|---|---|---|---|---|---|---|
| $\overline{\text{GPWU}}$ | $\overline{\text{GPPU}}$ | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |
| bit 7 | | | | | | | bit 0 |

**`GPPU`** (***GPio** PullUp*), bit 6, is the one that commands the enabling/disabling of Weak Pull-ups.

The sign of negation (**`! GPPU`** or **`NOT_GPPU`** or **`GPPU)`** indicates that the bit acts in the opposite way to what you expect, i.e.:

| GPPU | Weak ull-up |
|---|---|
| 0 | Activated |
| 1 | Disabled |

This choice derives from the fact that at **the POR,** the tendency of memory locations is to assume the value 1 and that disabled pull-ups are the condition of least consumption.
So, to make the pull-ups active, you will need to program the **`GPPU bit to 0`**:

```
    movlw    b'10011111'    ; enable pull-ups and internal
    OPTION
```

Please note that for Baselines the register is OPITION_REG accessible ONLY in write mode and ONLY with the special **`OPTION`** statement, which is not used by other PIC families. So **you can't write** something like this:

```
            Bcf      OPTION_REG, NOT_GPPU
```

And the correct form

```
    movlw    b'10111111'    ; Enable Pull-Ups
    OPTION
```

If, for example, during a debug, you need to know the state of the log bits, you need to resort to a shadow copy, since the contents can NOT be read in any way:

```
    bcf      sOPTION, NOT_GPPU     ; Enable Pull-Up in Shadow
    movf     sOPTION,w             ; Write the shadow to the
    OPTION
```

.

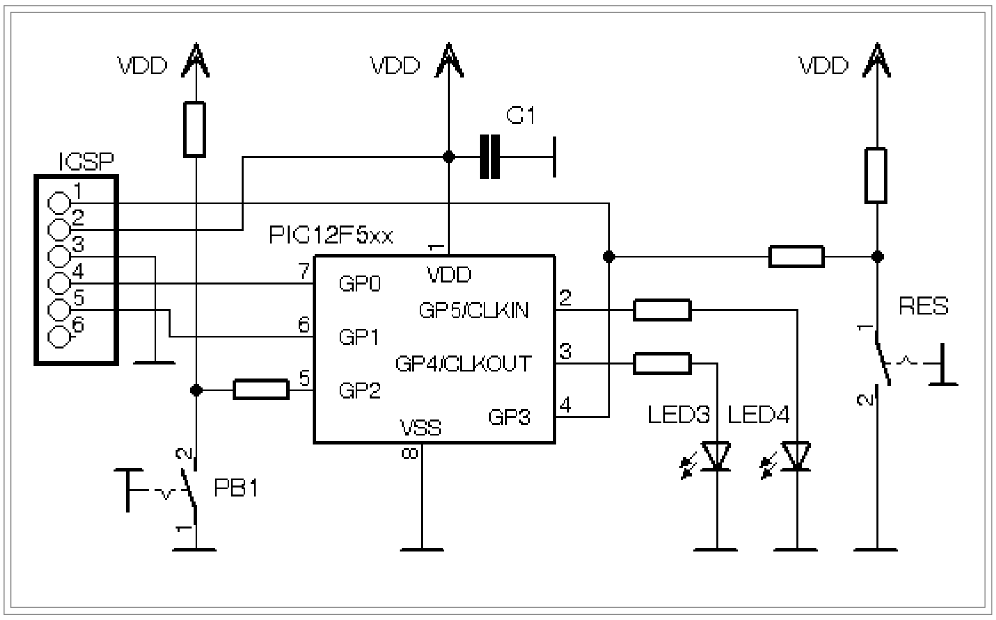
Then, after entering the instructions for enabling weak pull-ups, we can verify that the circuit is working normally.

## Variation.

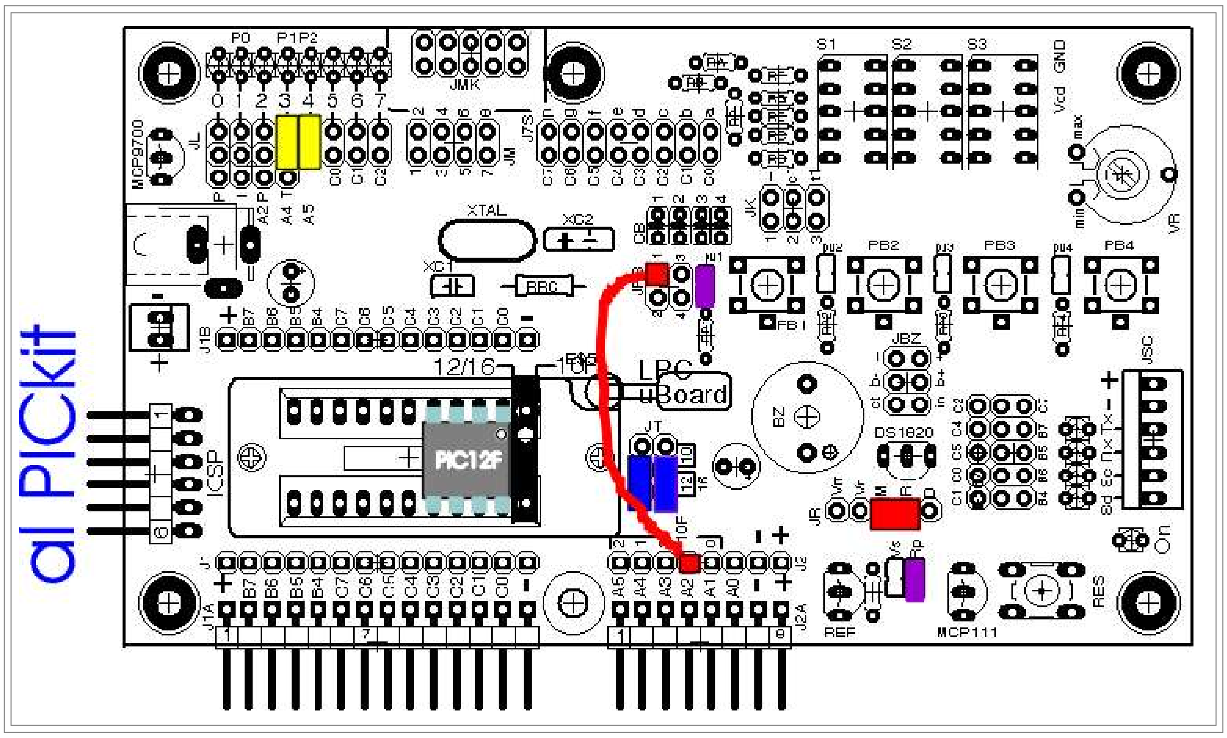
We can make a combination of what we saw in the previous exercise and the current one, creating a system with two LEDs and two buttons:

- By pressing RES we control the LED3
- by pressing PB1 we control the LED4

The LEDs work in "toggle" mode, i.e., when the LED button is pressed, the LED changes state. The circuit is similar to the previous one, to which we add a button on **`GP2`**:



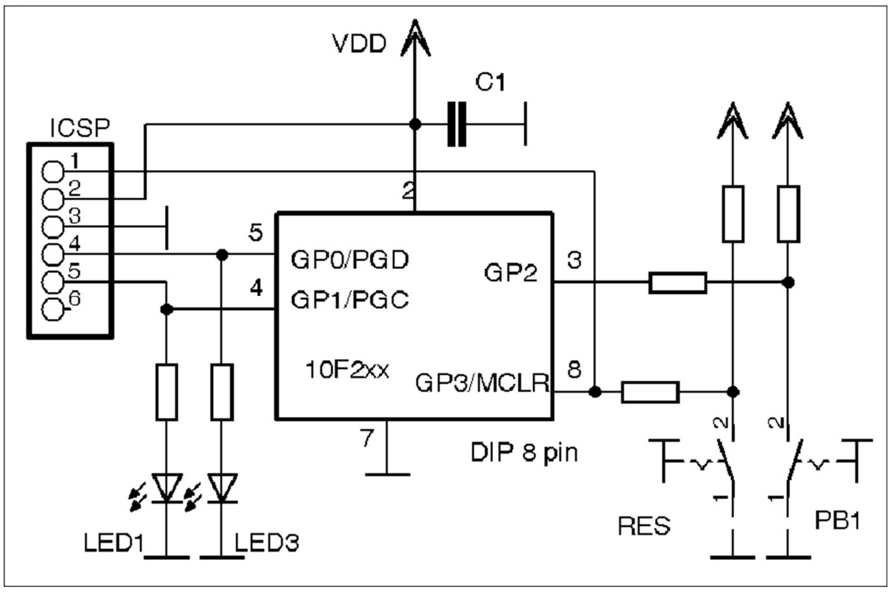
and connections on the **LPCuB**, where a jumper cable is required to connect the button to the **`GP2`**:



The "purple" jumpers are the ones that put the pull-ups in. The "red" flying jumper connects **`GP2`** to the **`PB1 button`**.

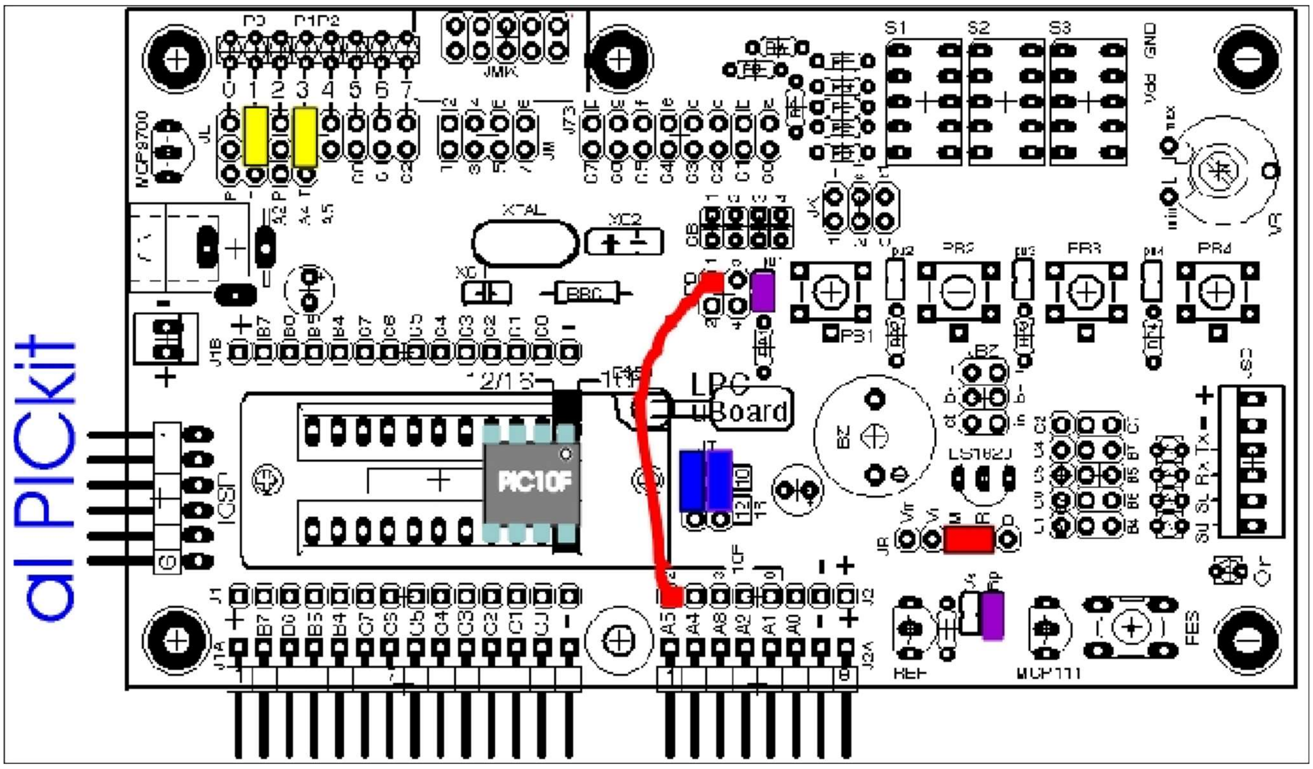
It should be noted that there are some confusing points in the Microchip documentation, including the different designation of some pins, which become part of PORTB or PORTA depending on the chip, despite having the same location and function.
In this case, the GP2 pin of the 12Fs is named RA2 in the 16Fs and 18Fs. The image, however, indicates the correct connection of the flying cable

For the **10F20x**, the scheme is as follows:



The RES button remains connected to GP3/MCLr, while the PB1 button is connected to the GP2 pin. The LEDs are connected to GP0 and GP1. As usual, they are not a problem if they are very low current elements, such as those on the **LPCuB**, which do not affect the signals on the ICSP connection with the Pickit; if they are other circuits, it is advisable to disconnect them during programming.

The connections on the **LPCuB** are as follows:

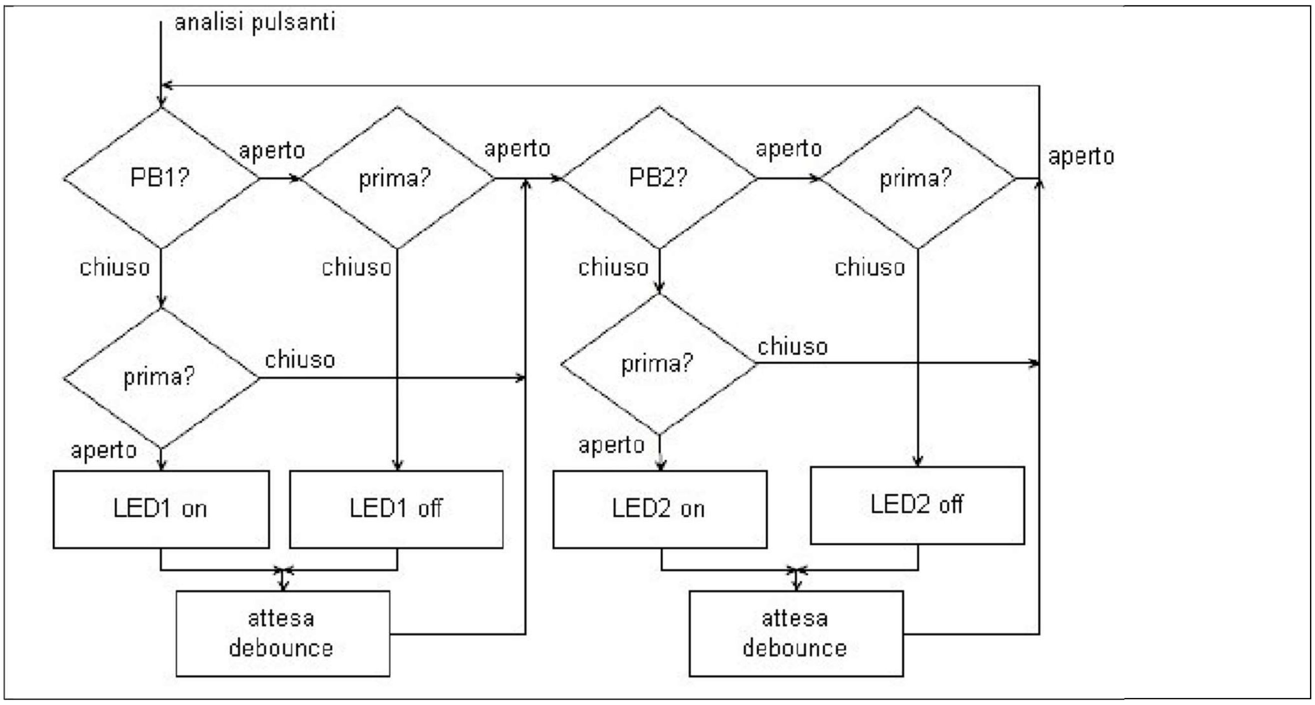


The "red" flying jumper connects GP2 with the PB1 button. The "purple" jumpers are the links of the pull-ups.

## The program.

Baselines do not have interrupts and it is not possible to follow the status of the buttons except through polling, i.e. a sequential structure where the processor analyzes the level of the pins connected to the buttons directly on the port and makes the appropriate decisions. To be exact, however, there is the function of wakeup from sleep following the change of styato of an I/O, which we will see later, but it is not a method that is always applicable.

Since there are more than one buttons, you need to create a sequence. This is one of the classic situations that if you don't deal with them by starting with a block diagram, you can hardly solve them effectively. Let's take a look at a first possibility of analyzing the state of the two buttons:



The diagram is self-explanatory: the previous state of the buttons is saved in a RAM location and checked against the current state. If this has changed, it turns the LED on or off accordingly. After each actuation, a debounce time is waited before moving on to the next step. If the status has not changed, no action is taken.
It is evident that the change in the state of the buttons does not give rise to an instantaneous response, but requires a certain amount of time (response time) from the microcontroller. In general, this is of little importance since the user's button operation has high execution times and therefore the few milliseconds taken by the analysis are not critical. This may be more sensitive in the case of event-driven mechanical contacts, such as limit switches or encoders; In this case, a different structure is required, and if response times are critical or the processor has other actions to wrap around, more complex interrupts and debounce forms are required.

However, starting from the flow chart just seen, the transposition into instructions is completely immediate. At the bottom of the document you will find the source ***7A_12F5xx3.asm*** for 12F508/9/10/19 and ***7A_10F20x3.asm*** for 10F200/2/4/6.

In the version for the PIC10F you can see the addition of a safety mechanism:

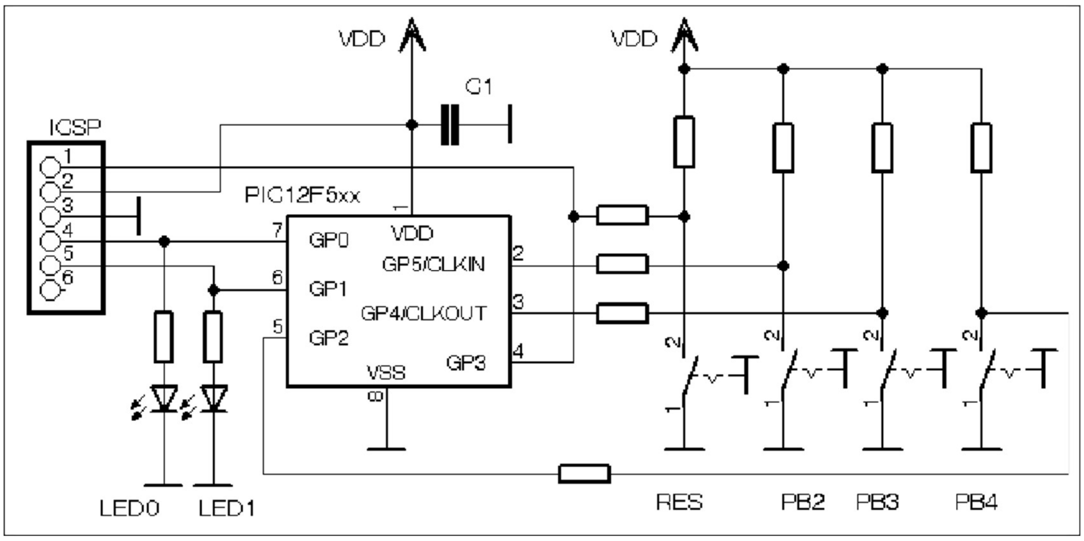
```
; Internal Oscillator Calibration
; Force disabling the Fosc/4 output even in the case of a
; Incorrect calibration value.
       andlw    0xFE     ; reset bit 0 and disable Fosc/4
       movwf    OSCCAL
```

Since the GP2 pin is used, which also has the function of Fosc/4, calibration with an incorrect value could bring the bit 0 to 1, enabling the output and thus preventing the correct functioning of the pin in a digital way. The `'andlw` with `B'11111110' prevents` Fosc/4 ***from being enabled*** no matter what the calibration value is.
It should be noted that the possibility is quite remote, since the programming devices warn if the calibration value is incorrect and if it has been deleted. This can happen by not lending

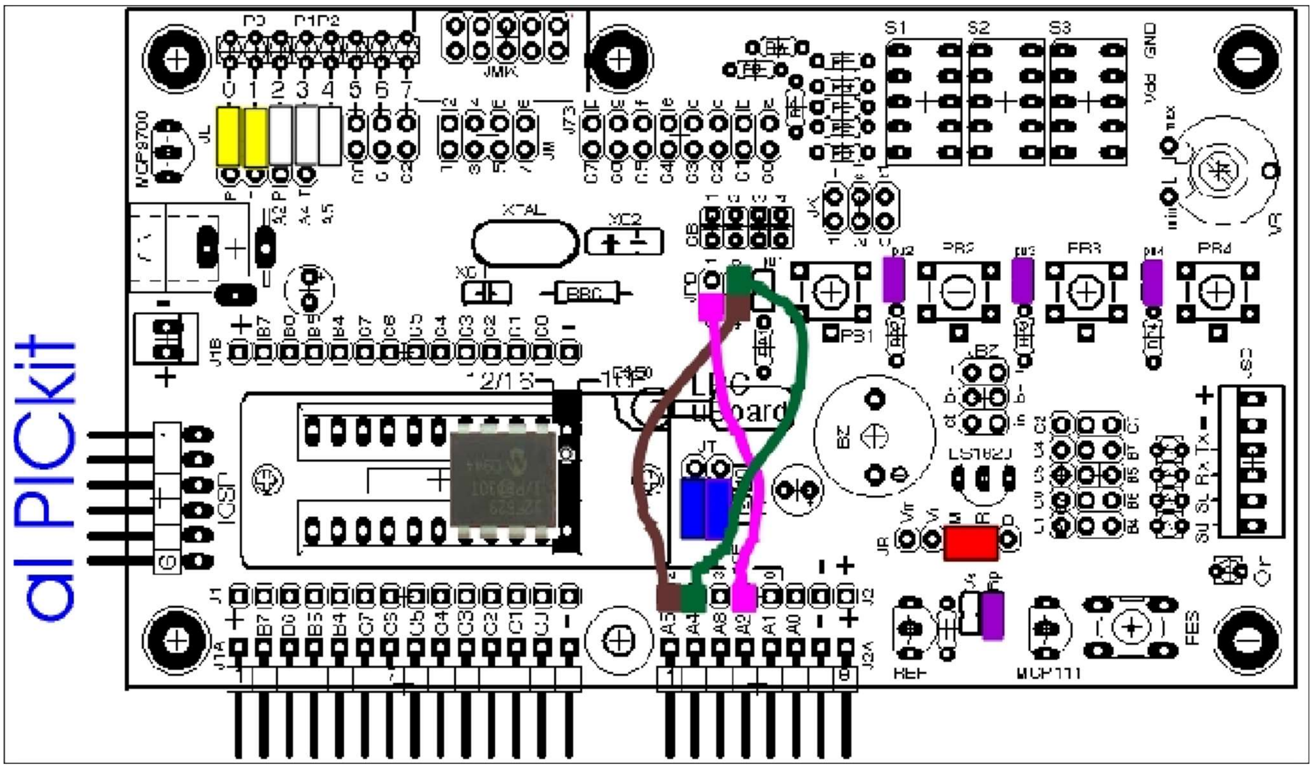
Be careful during experiments. So, it's a security addition that only results in 1us (@4MHz) in starting the program.
If, on the other hand, the calibration value is correct, it always has the bit0 = 0 and therefore the problem does not arise.
As for the correct calibration and restoration of the value in case it has been deleted, we refer to an article on the subject where the method of recovering the correct value using Pickit is explained, without the need to write any specific program.

## Other variation

We want to make it possible for 4 buttons to perform different actions on an LED. Let's use this scheme:



We engage all the I/O of the small 8-pin PIC, using two of them as the output and the other 4 as the input for the buttons.
There are 4 buttons available **on the** LPCuB (**PB1,2,3,4**), but **`GP3`** is already connected to the **RES** button, so we use **PB2,3** and **4** connected to **`GP2,4,5 respectively`**.
For the LED outputs we use **`GP0`** and
**`GP1`**. The connections on the board are as
follows:



Flying jumpers connect the buttons to the I/O:

- Pink **PB2** to **GP2**
- Green **PB3** to **GP4**
- Brown **PB4** to **GP5**

You could also use **PB1** connected to **GP3**, but since there is already the **RES** button, you save a connection.
The "yellow" jumpers connect the two LEDs. If we also insert the "white" jumpers, we will have the status of the PB2,3,4 buttons on the relative LEDs (LED on, button pressed).
"Purple" jumpers are related to pull-ups. They can be excluded to test the effect of the absence of the pull-up or to use the built-in weak pull-ups, controlled by the **OPTION register**.

## The program

The source code is written to be able to use 12F508, 509, 510 and 519.
It uses the time routine, already seen above, which performs a 10ms delay dependent on W. The actual program is as trivial as ever. Start with a button test loop:

```
; Test Buttons
BTNTEST BTFSS   PB2          ; button pressed ?
         Goto   XPB2         ; Yes - Execute button 1
        BTFSS   PB3          ; button pressed ?
         Goto   XPB3         ; Yes - Execute button 2
        BTFSS   PB4          ; button pressed ?
         Goto   XPB4         ; Yes - Execute button 3
        BTFSS   BtnR         ; button pressed ?
         Goto   xpbr         ; Yes - Execute button R
        Goto    BTNTEST     ; Loop
```

If a button is pressed (logical level 0 or clear) you jump to the specific managements. These are micro programs that each perform a different action.

**PB2 Button**:

```
; Executes button 2
XPB2    BSF     sLED0       ; LED0 on
        BSF     sLED1       ; LED1 on
        Movf    sGPIO,w     ; con shadow
        Movwf   GPIO
        Movlw   s1          ; delay 1s
        Call    Delay10msW
        BCF     LED1        ; LED1 off
XPB20   BTFSS   PB2         ; button released ?
         Goto   XPB20       ; No - Waiting
        movlw   debtime     ; debounce
        call    Delay10msW
        clrf    GPIO        ; LED0 off
        clrf    sGPIO
        Goto    BTNTEST     ; Back to Test Buttons
```

1. as soon as pressed, both LEDs light up, using the shadow
2. LED1 is turned off after 1s, regardless of the status of the button. No debounce time is added because it is useless
3. When the button is released, a debounce time is added to ensure release
4. LED0 also turns off
5. Back to button testing

**PB3 Button**:

```
; Executes button 3
XPB3    BSF     LED0        ; LED0 on
        Movlw   debtime     ; debounce
        Call    Delay10msW

XPB30   BTFSS   PB3         ; button released ? No -
         GOTO   XPB30       ; Waiting
        Bsf     LED1        ; LED1 on
        movlw   s1          ; Delay 1s
        call    Delay10msW
        CLRF    GPIO        ; LED off
        Goto    BTNTEST     ; Back to Test Buttons
```

1. as soon as the button is pressed, LED0 lights up
2. A debounce time is added to ensure closure
3. You wait for the button to open
4. LED1 is lit for 1s
5. then turn off both LEDs and return to the button test. Shutdown is achieved by simply resetting the GPIO latches.

**PB4 Button**:

```
; Executes button 4
XPB4     BSF      sLED0         ; LED0 on
         BSF      sLED1         ; LED1 on
         Movf     sGPIO,w       ; con shadow
         Movwf    GPIO
         Movlw    s05           ; Delay 0.5s
         Call     Delay10msW
         BCF      LED1          ; LED1 off
         Movlw    s05           ; delay 0.5s
         Call     Delay10msW
         BSF      LED1          ; LED1 on
         Movlw    s05           ; delay 0.5s
         Call     Delay10msW
         BCF      LED1          ; LED1 off

XPB40    BTFSS    PB4           ; Button Released No  ?
          Goto    XPB40         ; - Waiting Debounce
         Movlw    Debtime       ;
         Call     Delay10msW
         Bsf      LED1          ; LED1 on
         movlw    S05           ; Delay 0.5s
         Call     Delay10msW
         Bcf      LED1          ; LED1 off
         movlw    S05           ; Delay 0.5s
         Call     Delay10msW
         Bsf      LED1          ; LED1 on
         movlw    S05           ; Delay 0.5s
         Call     Delay10msW
         CLRF     GPIO          ; LED0 off
         CLRF     sGPIO
         Goto     BTNTEST       ; Back to Test Buttons
```

1. as soon as the button is pressed, the LED lights up
2. LED1 flashes twice (0.5s)
3. You wait for the button to open
4. Debounce time is added to ensure opening
5. LED1 flashes two more times
6. then turn off both LEDs and return to the button test

**RES Button**:

```
; Executes R button
xpbr     Bsf      sLED0         ; LED0 on
         Bsf      sLED1         ; LED1 on
         movf     sGPIO,w       ; with shadow
         movwf    GPIO
         movlw    S02           ; Delay 0.2s
         Call     Delay10msW
         Bcf      LED1          ; LED off
xpbr0    BTFSS    BtnR          ; button released ?
          Goto    xpbr0         ; No - Waiting
         Bsf      LED1          ; yes - LED on
         movlw    S02           ; Delay 0.2s
```

```
        Call      Delay10msW
        CLRF      GPIO          ; LED0 off
        CLRF      sGPIO
        Goto      BTNTEST       ; Back to Test Buttons
```

1. as soon as the button is pressed, the LEDs light up through the shadow
2. LED1 flashes (0.2s)
3. You wait for the button to open
4. LED1 flashes (0.2s)
5. Debounce time is added to ensure opening
6. then turn off both LEDs and return to the button test

## Possibility

You can try other variations, with more LEDs or more buttons, by linking the state of the buttons with the output combinations you want. Or using 14-pin chips like 16F505/506/526, which allows you to control more I/O.

The way to get to these results is nothing more than the application of what we have described so far, starting from the block diagram, the source from templates with the necessary comments, the use of subs and macros, the absence of absolutes, the adaptation of the config to the processor and the initialization of the I/O.

## Conclusions

From the exercises carried out so far, we should have noticed that the management of digital inputs and outputs is one of the fundamental operations for microcontrollers: the connection between real devices, microcontroller pins and values of the corresponding bits in binary bytes is the basis of electronic control of processes, measurements, automation.

In addition, it should become clear how, with the same component or circuit, it is possible, through the modification of the program, to perform different actions; This is the huge advantage of programmable devices over wired logics.

Obviously, to control external interfaces, especially if they are power, you need to know EXACTLY what you are dealing with and evaluate not superficially the real behavior of what is connected to the micro pins: the logic implemented could be very correct, if the peripherals behaved in an ideal way; For example, contacts without bounce, but it is quite possible that in reality this is not the case and then the logic of the program must be adapted to meet the real needs.

In particular, in the control of contacts, switches, buttons, mechanical encoders, limit switches and, in general, inputs on whose detection an action depends, as well as outputs that control relays, motors, etc., it is necessary to correctly evaluate the intervention times of the electromechanical components, which are usually much slower than the instruction cycle of the micro. It would be a mistake to think that we are going to have to string the instructions one after the other, and then end up with problems such as that of the rebounds, or other worse; Where necessary, the microcontroller must be slowed down and its operations adapted to the response of the input and output peripherals.

## One more note

The debounce structures now seen are such as to capture the change in the input pin and act immediately on the output, then adding the debounce time. Of course, many other approaches are possible (vertical counter, interrupts, use of timers, etc.).

However, it is possible that the change in the state of the input, in special cases, is due to a strong disturbance and not to the beginning of the closing or opening of the contact. This situation, where it exists, requires that the status of the entry be confirmed by sufficient time.
In this case, you should proceed as follows:

1. Locate the change in the status of the input
2. Turn on debug time
3. Check that, after the time has elapsed, the input has not changed status. This eliminates impulses caused by disturbances.
4. only if the status is confirmed, do what follows the switching of the input Turning this

into instructions is simple.

Moreover, it must be said that it is mandatory for any electronic equipment to be made in such a way (wiring, filters, power supply, etc.) as to be insensitive to disturbances that may create false input signals or make outgoing actions unsafe.
This is a priority condition, to which, then, in extreme cases, additional software filtering will be added.

## 7A_10F20x1.asm

```
;**********************************************************************
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 7A_10F20x
;                     Switches two LEDs depending on the status of a
;                     Input: An LED is lit when the input
;                     is at level 0 and the other LEd is off and you
;                     They trade when the input is at level 1.
;                     Software debunce management.
;    PIC            : 10F200/2/4/6
;    Support        : MPASM
;    Version       A: V.20x-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;----------------------------------------------------------------------
;
;  Pin use :
;  ----------------
;     10F200/2/4/6 @ 8 pin DIP      10F200/2/4/6 @ 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/[COUT]  3:4:
;    GP1/ICSPCLK/[CIN-]         4:  3: Out  LED
;    GP0/ICSPDAT/[CIN+]         5:  1: Out  LED
;    NC                         6:  Nc
;    Vss                        7:  2: --
;    GP3/MCLR/VPP               8:  6: In   Button
;
;  [] only 10F204/6
;
;**********************************************************************
;======================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|--------|
; | BTN |     |LED3 |LED1 |
;
#define   LED1 GPIO,GP0     ; LED1
#define   LED3 GPIO,GP1     ; LED3
;#define       GPIO,GP2     ; Not used
#define   btn GPIO,GP3      ; Button
```

```
;
;*******************************************************************
; ##################################################################
; Choice of #ifdef
 processor  10F200
        LIST      p=10F200      ; Processor Definition
        #include <p10F200.inc>
 #endif
 #ifdef __10F202
        LIST      p=10F202      ; Processor Definition
        #include <p10F202.inc>
 #endif
 #ifdef __10F204
        LIST      p=10F204      ; Processor Definition
        #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef __10F206
        LIST      p=10F206      ; Processor Definition
        #include <p10F206.inc>
#define proccomp
 #endif
             radix dec

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


; ##################################################################
;                        RAM
;
; general purpose RAM
        CBLOCK 0x10
   sGPIO                      ; Shadow GPIO
   d1,d2;   ENDC Delay Counter

; ##################################################################
;                        LOCAL MACRO
;
; LED status with open button
LEDOpen   MACRO
        Bsf   sGPIO, GP0
        bcf   sGPIO, GP1
        movf sGPIO, W
        movwf GPIO
          ENDM
; LED status with button closed
LEDClose MACRO
        Bcf   sGPIO, GP0
        bsf   sGPIO, GP1
        movf sGPIO, W
        movwf GPIO
          ENDM
```

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

; Internal Oscillator Calibration
; Force disabling the Fosc/4 output even in the case of a
; Incorrect calibration value.
        andlw   0xFE    ; reset bit 0 and disable Fosc/4
        movwf   OSCCAL
        Goto    Main

; ####################################################################
;==================================================================
;                         SUBROUTINES
; Delay for debounce in W x 500us
Debounce
       movwf d2       ; save number of
iterations Dly050 movlw 0xA5 ; Delay 500us
       movwf d1
Dly051 decfsz d1, f
        goto Dly051
       decfsz d2      ; End of
        iterations? goto Dly050 ; No -
        Other Cycle
       retlw 0        ; yes - Indent

Debouncetime EQU .20  ; 20 x 500us = 10ms

; ####################################################################
;==================================================================
Main:
  #ifdef proccomp       ; 10F204/6
; Disable comparators to free the digital MOVLW function
             0xF7              ; No Comparator
     movf    CMCON0
 #endif

; GP5:4 come out
        movlw   B'1100'
        Tris    GPIO

; Home position (button open) LEDOpen
; Waiting button closed
BTNLC   BTFSC   Btn
         Goto   BTNLC
; button closed, LED
        operatesLose
; Waiting for Debounce
        movlw   Debouncetime
        call    Debounce
; Hold open button BTNLA
        BTFSS   Btn
         Goto   BTNLA
; button open, LED actuates
```

```
        LEDOpen
; Atesa for Debounce
        movlw   Debouncetime
        call    Debounce
; Another loop
        Goto    BTNLC

;********************************************************************EN
       D
```

## 7A_12F5xx1.asm

```
;    Version       : V.519-1.
;*******************************************************************
;-------------------------------------------------------------------
;
;    Title         : Assembly & C Course - Tutorial 7A_12F5xx
;                    Switches two LEDs depending on the status of a
;                    Input: An LED is lit when the input
;                    is at level 0 and the other LEd is off and you
;                    They trade when the input is at level 1.
;                    Software debunce management.
;    PIC           : 12F508/509/510/519
;    Support       : MPASM
;    Date          : 01-05-2013
;    Hardware ref. :
;    Author        :Afg
;
;-------------------------------------------------------------------
;
; Pin use :
;  ----------------
;       12F5xx @ 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 LED
;    GP4/OSC2             3: Out LED
;    GP3/! MCLR/VPP       4: In   Button
;    GP2/T0CKI            5:
;    GP1/ICSPCLK          6:
;    GP0/ICSPDAT          7:
;    Vss                  8: --
;
;*******************************************************************
;===================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|LED5 |LED4 | BTN |     |     |     |
;
;#define    GPIO,GP0    ; Not used
;#define    GPIO,GP1    ; Not used
;#define    GPIO,GP2    ; Not used
#define   Btn   GPIO,GP3 ; #define
button    LED4 GPIO,GP4 ; LED control
#define   LED5 GPIO,GP5 ; LED control
;
```

```
 ;*********************************************************************
 ; ####################################################################

 ; Choice of #ifdef
   processor 12F509
          LIST      p=12F509
          #include <p12F509.inc>
 #define procdig                      ; Processor without analog
   #endif
   #ifdef____ 12F508
         LIST      p=12F508
         #include <p12F508.inc>
 #define procdig                      ; Processor without analog
   #endif
   #ifdef____ 12F519
          LIST      p=12F519          ; Processor definition
          #include <p12F519.inc>
 #define procdig                      ; Processor without analog
   #endif
   #ifdef____ 12F510
         LIST      p=12F510
          #include <p12F510.inc>
 #define procanalog                   ; Processor with analog
   #endif
            radix dec
 ; ####################################################################
 ;                        CONFIGURATION
#ifdef procdig          ; Code 12F508/509/519
 ; Internal Oscillator, No WDT, No CP, GP3
  __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
  #endif

  #ifdef procanalog      ; 12F510
 ; Internal Oscillator, 4MHz, No WDT, No CP, GP3
  __config  _IntRC_OSC & _IOSCFS_OFF & _WDT_OFF & _MCLRE_OFF
  #endif


 ; ####################################################################
 ;                        RAM
 ;
 ; general purpose RAM
         CBLOCK 0x10
   sGPIO                       ; Shadow GPIO
   d1,d2;   ENDC Delay Counter

 ; ####################################################################
 ;                        LOCAL MACRO
 ;
 ; LED status with open button
 LEDOpen   MACRO
         Bsf   sGPIO, GP4
         bcf   sGPIO, GP5
         movf sGPIO, W
         movwf GPIO
           ENDM
 ; LED status with button closed
 LEDClose MACRO
```

```
        Bcf   sGPIO, GP4
        bsf   sGPIO, GP5
        movf sGPIO, W
        movwf GPIO
          ENDM

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

 ; MOWF Internal Oscillator
        Calibration    OSCCAL
        Goto    Main

; ####################################################################
;====================================================================
;                          SUBROUTINES
; Delay for debounce in W x 500us
Debounce
       movwf d2       ; save number of
iterations Dly050 movlw 0xA5 ; Delay 500us
       movwf d1
Dly051 decfsz d1, f
        goto Dly051
       decfsz d2      ; End of
        iterations? goto Dly050 ; No -
        Other Cycle
       retlw 0        ; yes - indent

Debouncetime EQU .20  ; 20 x 500us = 10ms

; ####################################################################
;====================================================================
Main:
  #ifdef procanalog       ; 12F510
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the BCF digital function
                CM1CON0, C1ON
 #endif

; GP5:4 come out
        movlw   B'001111'
        Tris    GPIO

; Home position (button open) LEDOpen
; Waiting button closed
BTNLC   BTFSC   Btn
         Goto   BTNLC
; button closed, LED
        operatesLose
; Waiting for Debounce
        movlw   Debouncetime
        call    Debounce
```

```
; Hold open button BTNLA
       BTFSS   Btn
        Goto   BTNLA
; button open, LED actuates
       LEDOpen
; Atesa for Debounce
       movlw   Debouncetime
       call    Debounce
; Another loop
       Goto    BTNLC

;*****************************************************************EN

      D
```

## 7A_12F50x2.asm

```
;**********************************************************************
; 7A_12F50x2.asm
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 7A_12F5xx
;                     Switches two LEDs depending on the status of a
;                     Input: An LED is lit when the input
;                     is at level 0 and the other LEd is off and you
;                     They trade when the input is at level 1.
;                     Software debunce management.
;    PIC            : 12F508/509/510/519
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;----------------------------------------------------------------------
;
; Pin use :
;  ---------------
;       12F5xx @ 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  LED
;    GP4/OSC2               3: Out  LED
;    GP3/! MCLR/VPP         4: In   Button
;    GP2/T0CKI              5:
;    GP1/ICSPCLK            6:
;    GP0/ICSPDAT            7:
;    Vss                    8: --
;
;**********************************************************************
;======================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|LED5 |LED4 | BTN |     |     |     |
;
;#define    GPIO,GP0    ; Not used
;#define    GPIO,GP1    ; Not used
;#define    GPIO,GP2    ; Not used
#define   Button GPIO,GP3 ; #define
button    LED4   GPIO,GP4 ; LED control
#define   LED5   GPIO,GP5 ; LED control
```

```
  ;
  ;*********************************************************************
  ; ####################################################################

  ; Choice of #ifdef
    processor 12F509
           LIST      p=12F509
           #include <p12F509.inc>
  #define procdig                     ; Processor without analog
    #endif
    #ifdef __12F508
          LIST      p=12F508
          #include <p12F508.inc>
  #define procdig                     ; Processor without analog
    #endif
    #ifdef __12F519
           LIST      p=12F519         ; Processor definition
           #include <p12F519.inc>
  #define procdig                     ; Processor without analog
    #endif
    #ifdef __12F510
          LIST      p=12F510
           #include <p12F510.inc>
  #define procanalog                  ; Processor with analog
    #endif
             radix dec

  ; ####################################################################
  ;                         CONFIGURATION
 #ifdef procdig          ; Code 12F508/509/519
  ; Internal Oscillator, No WDT, No CP, GP3
   __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
   #endif

   #ifdef procanalog      ; 12F510
  ; Internal Oscillator, 4MHz, No WDT, No CP, GP3
   __config  _IntRC_OSC & _IOSCFS_OFF & _WDT_OFF & _MCLRE_OFF
   #endif


  ; ####################################################################
  ;                         RAM
  ;
  ; general purpose RAM
          CBLOCK 0x10
     sGPIO                       ; Shadow GPIO
          ENDC

  ; ####################################################################
  ;                         LOCAL MACRO
  ;
  ; LED status with open button
  LEDOpen   MACRO
          Bsf   sGPIO, GP4
          bcf   sGPIO, GP5
          movf sGPIO, W
          movwf GPIO
            ENDM
```

```
; LED status with button closed
```

```
LEDClose MACRO
        Bcf   sGPIO, GP4
        bsf   sGPIO, GP5
        movf sGPIO, W
        movwf GPIO
          ENDM

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

 ; MOWF Internal Oscillator
        Calibration    OSCCAL

; ####################################################################
;====================================================================
Main:
  #ifdef procanalog       ; 12F510
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the BCF digital function
                CM1CON0, C1ON
 #endif

; GP5:4 come out
        movlw   B'001111'
        Tris    GPIO

; Home position (button open) LEDOpen
; Closed Button Test (Low Level)
; Reset the main counter and the sample closecheck
counter:
        CLRF    main_cnt
        CLRF    sample_cnt
; Wait for the Closeloop contact to close:
        incfsz  sample_cnt,f   ; Sampling time
         Goto   Closeloop
        BTFSC   Button         ; Contact closed?
         Goto   CloseCheck     ; No - Recycle
        incf    main_cnt,f     ; Yes - Increase Main Counter
        movlw   mainstep       ; End Counter ?
        XORWF   main_cnt,w
        skpz                   ; Yes - Action
         Goto   Closeloop      ; No - Other Sampling
; closed-contact LED drive
        LEDClose
; Test for open button (high level)
; Reset the main counter and the sample opencheck
counter:
        CLRF    main_cnt
        CLRF    sample_cnt
; Wait for the contact to open
```

```
OpenLoop:
        incfsz  sample_cnt,f   ; Sampling time
         Goto   openloop
        BTFSS   Button         ; Open contact?
         Goto   opencheck      ; No - Recycle
        incf    main_cnt,f     ; Yes - Increase Main Counter
        movlw   mainstep       ; End Counter ?
        XORWF   main_cnt,w
        skpz                   ; Yes - Action
         Goto   openloop       ; No - Other Sampling
; LED open-contact actuator
        LEDOpen
; and close cycle
        Goto    CloseCheck
;*******************************************************************EN
       D
```

## 7A_10F20x2.asm

```
;**********************************************************************
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 7A_10F20x
;                     Switches two LEDs depending on the status of a
;                     Input: An LED is lit when the input
;                     is at level 0 and the other LEd is off and you
;                     They trade when the input is at level 1.
;                     Software debunce management.
;    PIC            : 10F200/2/4/6
;    Support        : MPASM
;    Version       A: V.20x-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;----------------------------------------------------------------------
;
;  Pin use :
;  ----------------
;     10F200/2/4/6 @ 8 pin DIP      10F200/2/4/6 @ 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/[COUT]  3:4:
;    GP1/ICSPCLK/[CIN-]          4:  3: Out   LED
;    GP0/ICSPDAT/[CIN+]          5:  1: Out   LED
;    NC                          6:  Nc
;    Vss                         7:  2: --
;    GP3/MCLR/VPP                8:  6: In    Button
;
;  [] only 10F204/6
;
;**********************************************************************
;======================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|--------|
; | BTN |     |LED3 |LED1 |
;
#define   LED1    GPIO,GP0     ; LED1
#define   LED3    GPIO,GP1     ; LED3
;#define          GPIO,GP2     ; Not used
#define   Button  GPIO,GP3     ; Button
```

```
;
;*******************************************************************
; ###################################################################
; Choice of #ifdef
 processor  10F200
        LIST      p=10F200     ; Processor Definition
        #include <p10F200.inc>
 #endif
 #ifdef __10F202
        LIST      p=10F202     ; Processor Definition
        #include <p10F202.inc>
 #endif
 #ifdef __10F204
        LIST      p=10F204     ; Processor Definition
        #include <p10F204.inc>
#define proccomp
 #endif
 #ifdef __10F206
        LIST      p=10F206     ; Processor Definition
        #include <p10F206.inc>
#define proccomp
 #endif
           radix dec

; ###################################################################
;                          CONFIGURATION
;
 ; No WDT, no CP, GP3
  __config  _WDT_OFF & _CP_OFF & _MCLRE_OFF


; ###################################################################
;                          RAM
;
; general purpose RAM
        CBLOCK 0x10
   sGPIO                     ; Shadow GPIO
        ENDC

; ###################################################################
;                          LOCAL MACRO
;
; LED status with open button
LEDOpen   MACRO
        Bsf   sGPIO, GP0
        bcf   sGPIO, GP1
        movf sGPIO, W
        movwf GPIO
          ENDM
; LED status with button closed
LEDClose MACRO
        Bcf   sGPIO, GP0
        bsf   sGPIO, GP1
        movf sGPIO, W
        movwf GPIO
          ENDM

; ###################################################################
```

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

; Internal Oscillator Calibration
; Force disabling the Fosc/4 output even in the case of a
; Incorrect calibration value.
        andlw   0xFE    ; reset bit 0 and disable Fosc/4
        movwf   OSCCAL
        Goto    Main

; ####################################################################
;==================================================================
;                         SUBROUTINES
; Delay for debounce in W x 500us
Debounce
       movwf d2       ; save number of
iterations Dly050 movlw 0xA5 ; Delay 500us
       movwf d1
Dly051 decfsz d1, f
        goto Dly051
       decfsz d2      ; End of
        iterations? goto Dly050 ; No -
        Other Cycle
       retlw 0        ; yes - indent

Debouncetime EQU .20  ; 20 x 500us = 10ms

; ####################################################################
;==================================================================
Main:
  #ifdef proccomp             ; 10F204/6
; Disable Comparator to Free Up Movlw Digital Function
             0xF7             ; No Comparator
     movf    CMCON0
 #endif

; GP5:4 come out
        movlw   B'1100'
        Tris    GPIO

; Home position (button open) LEDOpen
; Closed Button Test (Low Level)
; Reset the main counter and the sample closecheck
counter:
        CLRF    main_cnt
        CLRF    sample_cnt
; Wait for the Closeloop contact to close:
        incfsz  sample_cnt,f  ; Sampling time
         Goto   Closeloop
        BTFSC   Button        ; Contact closed?
         Goto   CloseCheck    ; No - Recycle
        incf    main_cnt,f    ; Yes - Increase Main Counter
        movlw   mainstep      ; End Counter ?
        XORWF   main_cnt,w
        skpz                    ; Yes - Action
         Goto   Closeloop       ; No - Other Sampling
; closed-contact LED drive
        LEDClose
; Test for open button (high level)
; Reset the main counter and the sample opencheck
counter:
        CLRF    main_cnt
        CLRF    sample_cnt
; Wait for the OpenLoop contact to open:
        incfsz  sample_cnt,f    ; Sampling time
         Goto   openloop
        BTFSS   Button          ; Open contact?
         Goto   opencheck       ; No - Recycle
        incf    main_cnt,f      ; Yes - Increase Main Counter
        movlw   mainstep        ; End Counter ?
        XORWF   main_cnt,w
        skpz                    ; Yes - Action
         Goto   openloop        ; No - Other Sampling
; LED open-contact actuator
        LEDOpen
; and close cycle
        Goto    CloseCheck
;*********************************************************************EN
       D
```

## 7A_12F50x3.asm

```
;*******************************************************************
;-------------------------------------------------------------------
;
;    Title         : Assembly & C Course - Tutorial 7A_12F5xx
;                    Switches two LEDs depending on the status of two
;                    Inputs.
;                    Software debounce management.
;    PIC           : 12F508/509/510/519
;    Support       : MPASM
;    Version       : V.519-1.0
;    Date          : 01-05-2013
;    Hardware ref. :
;    Author        :Afg
;
;-------------------------------------------------------------------
;
; Pin use :
;  ---------------
;       12F5xx @ 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  LED
;    GP4/OSC2             3: Out  LED
;    GP3/! MCLR/VPP       4: In   RES button
;    GP2/T0CKI            5: In   PB1 button
;    GP1/ICSPCLK          6:
;    GP0/ICSPDAT          7:
;    Vss                  8: --
;
;*******************************************************************
;===================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|LED5 |LED4 | res | PB1 |     |     |

PB1    EQU  2    ; PB1 button on GP2
BtnR   EQU  3    ; RESET button on GP3
LED4   EQU  4    ; LED4 control from GP4
LED5   EQU  5    ; LED5 control from GP5

;
;*******************************************************************
; ##################################################################
; Choice of #ifdef
  processor 12F509
         LIST      p=12F509
         #include <p12F509.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef___ 12F508
        LIST      p=12F508
        #include <p12F508.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef___ 12F519
         LIST      p=12F519         ; Processor definition
         #include <p12F519.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef___ 12F510
        LIST      p=12F510
        #include <p12F510.inc>
#define procanalog                  ; Processor with analog
  #endif
          radix dec

; #####################################################################
;                         CONFIGURATION
#ifdef procdig          ; Code 12F508/509/519
 ; Internal Oscillator, No WDT, No CP, GP3
  __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
  #endif

  #ifdef procanalog      ; 12F510
 ; Internal Oscillator, 4MHz, No WDT, No CP, GP3
  __config  _IntRC_OSC & _IOSCFS_OFF & _WDT_OFF & _MCLRE_OFF
  #endif


; #####################################################################
;                         RAM
;
; general purpose RAM
        CBLOCK 0x10
   lastGPIO                   ; GPIO image
   d1,d2;   ENDC Delay Counter

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

 ; MOWF Internal Oscillator
       Calibration     OSCCAL
        Goto    Main

; #####################################################################
;====================================================================
;                          SUBROUTINES
; Delay for debounce in W x 500us
Debounce
       movwf d2       ; save number of
iterations Dly050 movlw 0xA5 ; Delay 500us
       movwf d1
Dly051 decfsz d1, f
        goto Dly051
       decfsz d2      ; End of
        iterations? goto Dly050 ; No -
        Other Cycle
       retlw 0        ; yes - Indent

Debouncetime EQU .20  ; 20 x 500us = 10ms

; ####################################################################
;==================================================================
Main:
  #ifdef procanalog       ; 12F510
; Disable CLRF Analog Inputs
              ADCON0
; Disable comparators to free the BCF digital function
              CM1CON0, C1ON
  #endif

; disable T0CKI to have GP2 as movlw digital I/O
              b'11011111'
        OPTION

; GPIO preset latch to 0
        CLRF    GPIO

; GP5:4 out, others in
        movlw   b'001111'
        TRIS    GPIO

; Copy Initial Input Status
        MOVF    GPIO,w
        movwf   lastGPIO

; Button Analysis Loop
; Test for closed PB1 button (low level)
PB1cchk btfsc   GPIO,PB1       ; closed?
         Goto   pb1_0          ; No - Previous Status?
        BTFSS   lastGPIO,PB1 ; Yes - Previous state?
         Goto   BtRcchk        ; Was closed - check other BSF
        button  GPIO,LED4      ; was open - turn on LED
; Waiting for Debounce
        movlw   Debouncetime
        Call    Debounce
        Goto    BtRcchk        ; Check Other Button

; open button
pb1_0   BTFSC   lastGPIO,PB1   ; previous?
         GOTO   BtRcchk        ; It was open - check more Button
        Bcf     GPIO,LED4      ; was closed - turns off
                               LED

; Test for closed RES button (low level)
```

```
BtRcchk btfsc   GPIO,BtnR      ; closed?
```

```
          Goto    rb_0            ; No - Previous Status?
         BTFSS    lastGPIO,RB     ; Yes - Previous state?
          Goto    PB1ccchk        ; Was Closed - Loop
         Bsf      GPIO,LED3       ; was open - turn on LED
; Waiting for Debounce
         movlw    Debouncetime
         Call     Debounce
         Goto     PB1cchk         ; Loop

; open button
rb_0     BTFSC    lastGPIO,PB1    ; previous?
          goto    PB1cchk         ; It Was Open - Loop
         Bcf      GPIO,LED4       ; was closed - turns off
                                    LED
; Waiting for Debounce
         movlw    Debouncetime
         Call     Debounce
         Goto     PB1cchk         ; Loop

;********************************************************************EN
        D
```

## 7A_10F20x3.asm

```
;*******************************************************************;---
-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 7A_10F20x3
;                     Switches two LEDs depending on the status of a
;                     Input: An LED is lit when the input
;                     is at level 0 and the other LEd is off and you
;                     They trade when the input is at level 1.
;                     Software debunce management.
;    PIC            : 10F200/2/4/6
;    Support        : MPASM
;    Version       A: V.20x-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-------------------------------------------------------------------
;
;  Pin use :
;  ---------------
;     10F200/2/4/6 @ 8 pin DIP      10F200/2/4/6 @ 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/[COUT] 3:4:
;    GP1/ICSPCLK/[CIN-]        4:  3: Out  LED
;    GP0/ICSPDAT/[CIN+]        5:  1: Out  LED
;    NC                        6:  Nc
;    Vss                       7:  2: --
;    GP3/MCLR/VPP              8:  6: In   Button
;
;  [] only 10F204/6
;
;*******************************************************************
;===================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|--------|
; | BTN | PB1 |LED3 |LED1 |
;
PB1    EQU  2   ; PB1 button on GP2
BtnR   EQU  3   ; RESET button on GP3
LED1   EQU  0   ; LED1 control from GP0
LED3   EQU  1   ; LED5 control from GP1
```

```
;
;*********************************************************************
; ####################################################################
; Choice of #ifdef
 processor  10F200
        LIST      p=10F200     ; Processor Definition
        #include <p10F200.inc>
 #endif
 #ifdef___10F202
        LIST      p=10F202     ; Processor Definition
        #include <p10F202.inc>
 #endif
 #ifdef___10F204
        LIST      p=10F204     ; Processor Definition
        #include <p10F204.inc>
define proccomp
 #endif
 #ifdef___10F206
        LIST      p=10F206     ; Processor Definition
        #include <p10F206.inc>
#define proccomp
 #endif
          Radix   DEC

; ####################################################################
;                         CONFIGURATION
;
 ; No WDT, no CP, GP3
  __config  _WDT_OFF & _CP_OFF & _MCLRE_OFF


; ####################################################################
;                         RAM
;
; general purpose RAM
        CBLOCK 0x10
   sGPIO                    ; Shadow GPIO
        ENDC

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

; Internal Oscillator Calibration
; Force disabling the Fosc/4 output even in the case of a
; Incorrect calibration value.
        andlw   0xFE    ; reset bit 0 and disable Fosc/4
        movwf   OSCCAL
        Goto    Main

; ####################################################################
;===================================================================
;                          SUBROUTINES
; Debounce delay in W x 500us
```

```
Debounce
       movwf d2       ; save number of
iterations Dly050 movlw 0xA5 ; Delay 500us
       movwf d1
Dly051 decfsz d1, f
        goto Dly051
       decfsz d2      ; End of
        iterations? goto Dly050 ; No -
        Other Cycle
       retlw 0        ; yes - Indent

Debouncetime EQU .20  ; 20 x 500us = 10ms

; ####################################################################
;====================================================================
Main:
  #ifdef proccomp             ; 10F204/6
; Disable Comparator to Free Up Movlw Digital Function
                0xF7              ; No Comparator
        movf    CMCON0
 #endif

; disable T0CKI to have GP2 as movlw digital I/O
                b'11011111'
        OPTION

; GPIO preset latch to 0
        CLRF    GPIO

; GP1:0 as out, others as in
        movlw   b'1100'
        TRIS    GPIO

; Copy Initial Input Status
        MOVF    GPIO,w
        movwf   lastGPIO

; Button Analysis Loop
; Test for closed PB1 button (low level)
PB1cchk btfsc   GPIO,PB1       ; closed?
         Goto   pb1_0          ; No - Previous Status?
        BTFSS   lastGPIO,PB1   ; Yes - Previous state?

         Goto   BtRcchk        ; Was closed - check other button
        Bsf     GPIO,LED1      ; was open - turn on LED
; Waiting for Debounce
        movlw   Debouncetime
        Call    Debounce
        Goto    BtRcchk        ; Check Other Button

; open button
b1_0    BTFSC   lastGPIO,PB1  ; previous?
         GOTO   BtRcchk        ; It was open - check more Button
        Bcf     GPIO,LED1      ; was closed - turns off
                              LED

; Test for closed RES button (low level)
BtRcchk btfsc   GPIO,BtnR      ; closed?
```

```
        Goto    rb_0            ; No - Previous Status?
```

```
         BTFSS    lastGPIO,RB     ; Yes - Previous state?
          Goto    PB1ccchk        ; Was Closed - Loop
         Bsf      GPIO,LED3       ; was open - turn on LED
; Waiting for Debounce
         movlw    Debouncetime
         Call     Debounce
         Goto     PB1cchk         ; Loop

; open button
rb_0     BTFSC    lastGPIO,PB1    ; previous?
          GOTO    PB1cchk         ; It Was Open - Loop
         Bcf      GPIO,LED3       ; was closed - turns off
                                    LED
; Waiting for Debounce
         movlw    Debouncetime
         Call     Debounce
         Goto     PB1cchk         ; Loop

;*********************************************************************EN
        D
```

## 7A_12F5xx4.asm

```
;********************************************************************
;--------------------------------------------------------------------
;
; Title          : Assembly & C Course - Tutorial 7A_4
;                  Buttons and LEDs.
; Pressing PB1 will light LED1 for 1s
; Pressing PB2 will cause LED1 to be lit for 1s when released
; Pressing PB3 causes LED1 to flash 2 times
; Pressing PB4 will turn LED1 on 0.2s when closing and releasing
; LED0 is lit when the button is pressed, and
; off when the task is finished executing.
; Tasks are performed individually: only one
; button at a time.
;
; PIC            : 12F508/509/510/519
; Support        : MPASM
; Version        : V.519-1.0
; Date           : 01-05-2013
; Hardware ref. :
; Author         :Afg
;
;--------------------------------------------------------------------
;
; Pin use :
;  ----------------
;       12F5xx @ 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: In   PB4 button
;    GP4/OSC2            3: In   PB3 button
;    GP3/! MCLR/VPP      4: In   RES button
;    GP2/T0CKI           5: In   PB2 button
;    GP1/ICSPCLK         6: Out  LED1
;    GP0/ICSPDAT         7: Out  LED0
;    Vss                 8: --
;
;********************************************************************
;====================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|PB4 |PB3 | res | PB2 | LED1| LED0|
;
#define PB2    GPIO,GP2    ; PB2 button on GP2
#define BtnR   GPIO,GP2    ; RES button on GP3
```

```
#define PB3    GPIO,GP4     ; PB3 button on GP4
#define PB4    GPIO,GP5     ; PB4 button on GP5
#define LED0   GPIO,GP0     ; LED0 control from
                            GP0
#define LED1   GPIO,GP1     ; LED1 control from
                            GP1
#define sLED0  sGPIO,GP0    ; Aliases for Shadow
#define sLED1  sGPIO,GP1
;
;*********************************************************************
; ####################################################################

; Choice of #ifdef
  processor 12F509
         LIST      p=12F509
         #include <p12F509.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef___ 12F508
        LIST      p=12F508
        #include <p12F508.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef___ 12F519
         LIST      p=12F519         ; Processor definition
         #include <p12F519.inc>
#define procdig                     ; Processor without analog
  #endif
  #ifdef___ 12F510
        LIST      p=12F510
         #include <p12F510.inc>
#define procanalog                  ; Processor with analog
  #endif
          radix dec

; ####################################################################
;                        CONFIGURATION
#ifdef procdig          ; Code 12F508/509/519
; Internal Oscillator, No WDT, No CP, GP3
 __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif

 #ifdef procanalog       ; 12F510
; Internal Oscillator, 4MHz, No WDT, No CP, GP3
 __config  _IntRC_OSC & _IOSCFS_OFF & _WDT_OFF & _MCLRE_OFF
 #endif


; ####################################################################
;                        RAM
;
; general purpose RAM
        CBLOCK 0x10
   sGPIO                    ; Shadow
   d1,d2,d3                 ; ENDC Delay Counter

; ####################################################################
;                        CONSTANTS
```

```
;
 CONSTANT debtime = .2 0    ; Debounce time 20 x 10ms = 200ms
 CONSTANT  s05 = .50          ;   50 x 10ms = 500ms
 CONSTANT  s1 = .100          ;  100 x 10ms = 1s
 CONSTANT  s02 = .20          ;   20 x 10ms = 0.2s


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

 ; MOWF Internal Oscillator
        Calibration    OSCCAL
        Goto    Main

; ####################################################################
;                         SUBROUTINES
; 10ms x W delay subroutine
 #include C:\PIC\Library\Baseline\Delay10msW.asm

; ####################################################################
;                         MAIN PROGRAM
Main:
; reset initializations depending on the PIC used

  #ifdef procanalog            ; 12F510
; Disable CLRF Analog Inputs
                ADCON0

; Disable comparators to free the BCF digital function
                CM1CON0, C1ON
        Bcf     CM2CON0, C2ON
 #endif

; disable T0CKI from GP2
        movlw   b'11011111'
        OPTION

        CLRF    GPIO        ; Clear Latch of CLRF
        Ports   GPIO        ; Clear Shadow for
        Debugging

; GP1:0 out, others in
        movlw   B'111100'
        Tris    GPIO


; Test Buttons
BTNTEST BTFSS   PB2          ; button pressed ?
         Goto   XPB2         ; Yes - Execute button 2
        BTFSS   PB3          ; button pressed ?
         Goto   XPB3         ; Yes - Execute button 3
        BTFSS   PB4          ; button pressed ?
         Goto   XPB4         ; Yes - Execute button 4
        BTFSS   BtnR         ; button pressed ?
         Goto   xpbr         ; Yes - Execute button R
        Goto     BTNTEST      ; Loop

; executes PB2 button
XPB2    BSF      sLED0        ; LED0 on
        BSF      sLED1        ; LED1 on
        Movf     sGPIO,w      ; con shadow
        Movwf    GPIO
        Movlw    s1           ; delay 1s
        Call     Delay10msW
        BCF      LED1         ; LED1 off
XPB20   BTFSS    PB2          ; button released ?
         Goto    XPB20        ; No - Waiting
        movlw    debtime      ; debounce
        call     Delay10msW
        clrf     GPIO         ; All LEDs off
        clrf     sGPIO
        Goto     BTNTEST      ; Back to Test Buttons

; Executes PB3 button
XPB3    BSF      LED0         ; LED0 on
        Movlw    debtime      ; debounce
        Call     Delay10msW
XPB30   BTFSS    PB3          ; button released ? No -
         Goto    XPB30        ; Waiting
        Bsf      LED1         ; LED1 on
        movlw    s1           ; Delay 1s
        call     Delay10msW
        CLRF     GPIO         ; All LEDs off
        Goto     BTNTEST      ; Back to Test Buttons

; Executes button 4
XPB4    BSF      sLED0        ; LED0 on
        BSF      sLED1        ; LED1 on
        MOVF     sGPIO, w     ; con shadow
        movwf    GPIO
        movlw    s05          ; Delay 0.5s
        call     Delay10msW
        Bcf      LED1         ; LED1 off
        movlw    s05          ; Delay 0.5s
        call     Delay10msW
        Bsf      LED1         ; LED1 on
        movlw    s05          ; Delay 0.5s
        call     Delay10msW
        Bcf      LED1         ; LED1 off
XPB40   BTFSS    PB4          ; Button Released No  ?
         Goto    xpb40        ; - Waiting Debounce
        Movlw    debtime      ;
        Call     Delay10msW     LED1 on
        BSF      LED1         ; delay 0.5s
        Movlw    s05          ;
        Call     Delay10msW     LED1 off
        BCF      LED1         ; delay 0.5s
        Movlw    s05          ;
        Call     Delay10msW     LED1 on
        BSF      LED1         ; delay 0.5s
        Movlw    S05          ;
        Call     Delay10msW
        CLRF    GPIO        ; LED off
        CLRF    sGPIO
        Goto    BTNTEST     ; Back to Test Buttons

; Executes R button
xpbr    BSF     sLED0       ; LED0 on
        BSF     sLED1       ; LED1 on
        Movf    sGPIO,w     ; con shadow
        Movwf   GPIO
        Movlw   s02         ; delay 0.2s
        Call    Delay10ms_w
        BCF     LED1        ; LED1 off
xpbr0   BTFSS   BtnR        ; button released ?
         Goto   xpbr0       ; No - Waiting
        Bsf     LED1        ; yes - LED1 on
        movlw   S02         ; Delay 0.2s
        Call    Delay10ms_w
        CLRF    GPIO        ; All LEDs Off
        CLRF    sGPIO
        Goto    BTNTEST     ; Back to Test Buttons

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