Knowledge of baseline

Sleep, Wakeup, WDT, Pin Change

The purpose of the tutorial is to learn about the characteristics of sleep state and wakeup modes, including WDT Watchdog and level change on pins.

These functions are common to all PCIs.

Sleep

The need to minimize energy consumption is a fundamental element of many areas of design. In particular, equipment powered by batteries or solar systems requires low consumption, in order to have a long autonomy. This contrasts with the performance expected from the microprocessor: the speed of instruction execution is directly proportional to the clock frequency, but so is the power consumption.

One solution is to use very low working frequencies, which is not always possible for the needs of the application, since reducing the clock frequency increases the execution time of the instructions. However, we have seen that in a generic program, part of the time is spent waiting which, in some applications, can be the predominant part; As a result, another way to save power is to reduce the clock during periods when its activity is not important, resuming it at the time of a particular event, thus obtaining a low or very low average power consumption.

In PICs of higher families it is possible to change the clock during the execution of the program, but in Baselines this possibility is not possible. Speaking of the clock, we have mentioned that these microcontrollers do not have a minimum frequency limit for the main oscillator, since they do not contain dynamic memory (which would require a periodic refresh, and therefore the continuous presence of a clock), but only static RAM and Flash (which do not require it). Thus, it is possible to suspend the clock completely, freezing the situation at a precise moment, and then restart it after some time, following a desired or expected event, resuming the flow of instructions.

This feature is present in all PICs, as it offers the advantage of virtually zero power consumption during periods of clock suspension, while maintaining performance during periods of normal clock work. This mode is called **sleep**.

Sleep

It consists **of the suspension of the main clock**, which corresponds to the stopping of the Program Counter and the instruction execution cycle when the sleep state is induced by the sleep instruction.
The syntax of the opcode is simple, having no object:

[label]	sp	Sleep	sp	[; comment]

When the processor is "sent to sleep" (*to sleep* = sleep) the current consumption is reduced by values of the order of microamperes, allowing a battery-powered device to be able to operate for very long times, in an "active" sleep condition, and resume operation following a specific event.

We say operate, because **the sleep state is not equivalent to the shutdown that results from the power off**: if the power is cut off from the processor, the only way to restart it is to reconnect this power supply, which generates a reset (POR). In sleep the processor is not "turned off", and the power supply voltage must always be present, but it is only in a "sleep" state and can "wake ***up***" as a result of particular events. In the Baselines these are:

- **Watchdog Counter Overflow** (WDT)
- **I/O** pin status change
- Change in comparator status
- **MCLR**

It should be noted that in the Baselines the sleep function is much simplified compared to the others

[illegible]

Wakeup from the sleep condition acts by bringing the position of the Program Counter to the reset vector and restarting the clock.

So, **putting the processor to sleep and then "waking it up" is equivalent to a reset**.

The usefulness of sleep and wakeup can be indicated with an example: suppose we have a system that needs to transmit a signal every time a button is pressed. It is useless to keep the processor active with a loop to check the status of the input, consuming energy, since it is possible to proceed in this way: wakeup is enabled for level variation of the I/O to which the button is connected, the program measures and sends the data; Then put the processor back to sleep. The power consumption is drastically reduced, while the internal wakeup circuit, as soon as the pin level changes, wakes up the processor again. The result is a prompt response to the event, while the average consumption is almost zero.

We can see in a practical example the sleep function.

How do I stop the program.

We are dealing with a program that is not a continuous loop, but a linear series of instructions. We have already dealt with the necessity and the way to stop execution, and we have used it in a number of examples:

```
; stop al programma
        goto    $
```

Now let's look at another possible method to achieve the same thing. Let's replace the last line with a more refined one:

```
; stop al programma
        sleep
```

Again, the execution of subsequent instructions is blocked, since `sleep` stops the clock. The Program Counter can no longer proceed further.
In both cases, the result is identical: the program stops, but the operation of the two solutions is quite different:

- `Goto $` keeps the processor running, perpetually looping on itself. The Program Counter does not advance and the execution of other subsequent instructions cannot continue. However, the line that refers to itself is continuously executed and the micro operates as in ordinary operation.
- `Sleep` stops the clock, and without this the instruction advance mechanism is blocked. However, the processor doesn't work, it doesn't do anything, and the power consumption drops to a minimum.

So, if we apply the second method, we have a stop of the program and, at the same time, a minimum current consumption.

You can take one of the examples covered in the previous chapters and recompile it by replacing the stop line and see the effect.

In both cases, to exit the blockage it is necessary to either press the RES button, generating a reset for MCLR, or remove and reconnect the power supply, generating a POR reset.
However, in the second, it is also possible to get out of the sleep condition through other ways, including WDT overflow.

WDT

WDT stands for ***WatchDog Timer*** ; the meaning of the English word is, literally, "watchdog". This function is present in all PICs and, in the Baselines, it is found, like others, in the most basic form, which allows us to better understand its characteristics and application logic.

It is a timer, similar to the Timer0, that:

- counts to grow from 00 to FFh
- It is not stoppable (free-running timer)
- **It is independent of any other processor function, including the presence of the main clock, since it has its own clock**

The timer counts until it reaches overflow, after which the Program Counter is forced to the reset vector, restarting the program at that point.
Once enabled, WDT overflow acts at any time, regardless of instruction flow, to reset the processor.

It should be noted that the reset generated by the WDT is different from the one obtained by the POR module for the arrival of the supply voltage or the one obtained by sending the MCLR pin to a low level.

The purpose of WDT is to provide an element of security to the devices and processes that the microcontroller manages. It is analogous, in principle, to the "dead man's" button that exists on railway locomotives: if the conductor does not press this button at regular intervals, an automatic system stops the train, because it assumes that the driver is no longer able to do his job.

Similarly, if the program does not reset the counter to zero before the overflow, thus restarting the count cyclically, but, for some reason, allows it to run out, the overflow resets the system, believing that the program is no longer able to function properly. This situation can happen in the case of programs that are not well made, with, for example, stack overflows or undefined loops, or that do not take into account the input variables; or even for an unexpected general failure. A reset is an attempt to restore the functions of the program.

The cyclic "button press" is carried out by the program with the **`clrwdt instruction`**.

CLRWDT

This opcode has only one purpose: to reset the WDT counter (***CLeaR WDT***) and prevent it from overflowing, forcing the count to be resumed from 0. The syntax is simple:

[label]	sp	**clrwdt**	sp	**[; comment]**

It has no object and must be inserted into the program loop with a cadence that anticipates overflow. For example, if the WDT "times out" after 18ms, you need to arrange the instruction in the program so that it repeats less than 18ms apart (to be safe, much less, since the WDT clock is not an accurate element).

If we constantly reset the counter, it will never overflow and therefore the watchdog will not come into operation. It is therefore necessary to insert this instruction into the program in a cyclical way, at a frequency that constantly precedes the moment of the overflow of the counter and brings the count back to zero. The opcode zeroes both the counter and the contents of the prescaler, thus ensuring the nominal duration of the overflow.

In fact, while TMR0 is read-and-write accessible and allows you to finely act on the count, the WDT register is not accessible to the user; The only things he can do are:

- **enable WDT**, **through the config**. This action does not depend on the program's instructions. Once you've decided whether or not to enable WDT, the choice can't be changed except with a new config.
- **determine whether WDT has a prescaler** or not, through the **PSA** bit of the **OPTION** register. This is done by the program, which means that you can vary the prescaler while working the micro.
- **Reset the log count, using the** CLRWDT **(***Clear **WA**tchdog Timer***)** specific instruction

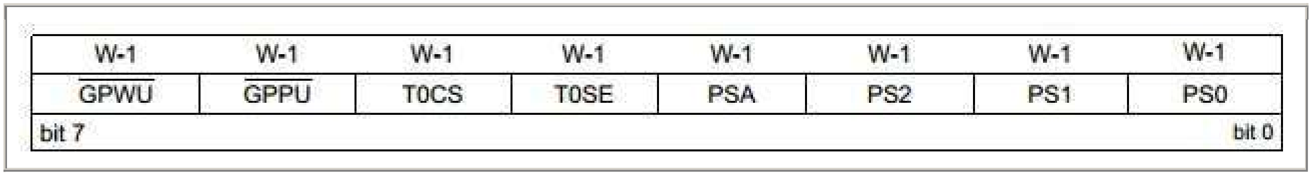
We can highlight this with a practical example.

Check the WDT

From a config point of view, enabling WDT is simple:

```
 _config  _WDTE_ON
```

Its management is also simple, since, in Baselines, it is only necessary to act on the register **OPTION:**



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

We have seen, talking about the Timer0, that it has a pre-scaler that allows you to expand the range of times that can be obtained. However, this prescaler, controlled by **the PSA** bit, is shared with the WDT, or, in essence, **it can only be assigned to one or the other of the timers.**

When the prescaler is made available to Timer0 (**PSA** = 0), the pre-division factor for the WDT is 1:1 and therefore the maximum time before overflow is about 18ms (depending on the temperature and the construction parameters of the chip, i.e., as mentioned, it is an inaccurate time). However, if we associate the prescaler with the WDT (**PSA** = 1, which is the default condition for the POR), it will be the Timer0 that will have, with respect to its input clock, a ratio of 1:1. You can use Timer0 and WDT at the same time, but **you can't have prescalers for both**. That is why it is necessary to carefully evaluate the programmes that need these resources.
The table already seen for Timer0 comes in handy:

PS2-0	TIMER0	WDT
`000`	1:2	1:1
`001`	1:4	1:2
`010`	1:8	1:4
`011`	1:16	1:8
`100`	1:32	1:16
`101`	1:64	1:32
`110`	1:128	1:64
`111`	1:256	1:128

PS2-0 **`bits`**, as with Timer0, change the pre-division ratio: so, for example, if you enter 1:128 (**`PS2-0`** = 111), the WDT overflow will be delayed by 18ms * 128.

On the other hand, it is possible, during the execution of the program, to switch the prescaler from one of the timers to the other, thus having a considerable dynamic possibility. Examples of how this should be done can be found on the chip datasheets in the WDT section.

In Baselines, there is no way to enable or disable WDT except by programming the *Configuration Word*; So, if you want to use this function, the choice must be made at the time of writing the chip and it cannot be changed except by another writing. Therefore:

Caution:

1. **It doesn't make any sense to insert the `clrwdt` opcode into the program if WDT has not been enabled** in the initial config: you would only get the no-load consumption of 1 instruction-loop with no other effect.

2. On the other hand, **once WDT has been enabled in the config, it is obligatory for the program to manage it appropriately, making sure that the instruction to cancel the count is implemented with the right cadence.**
Otherwise, the timer, when it reaches overflow, will reset the micro, creating abnormal operating situations.

But this alone is not enough: if we have activated the WDT it is because we need it to solve an unexpected problem; Your speech shows that something has not worked as it should: either the program is defective or the analysis of the hardware situation is insufficient or there has been a serious problem which has prevented the program from working properly.
The intervention of the WDT alone can also momentarily restore the control functions, restarting the program, but it certainly does not solve the causes. Then, when the WDT is activated,

It must be associated with the management of the event within the programme, so that it can determine what happened and allow it to be corrected.

It is clear that if a microcontroller system is built to manage some automated function, measurement or control of a process, this purpose must be served without the possibility of error. Where a malfunction occurs, it is obvious that you must intervene to solve the problem and not only by "pressing" the reset button and hoping that the thing will happen again as little as possible.

WDT Time

If with **Timer0** it is possible to obtain precise times by varying both the prescaler and the loading of **`TMR0`**, for **WDT** it is only possible to vary the prescaler. Incidentally, it is indeed a postscaler, but for simplicity's sake, we continue to call it a prescaler.
The minimum overflow time of the timer, without prescaler, is on average 18ms, but it can range in a wider range, from 9 to 30ms (param. 31 of electrical characteristics, data sheet). These values can vary from one type of chip to another (for example, for 10F2xx the average value is 16ms with a range from 10 to 31ms): basically, it is a fairly random time, depending on chip to chip and with the variation of voltage Vdd and temperature.
Considering the average value of 18ms and minimum 10ms, we therefore have the following situation:

PS2-0	WDT	Minimum timeout	Average timeout
000	1:1	10ms	18ms
001	1:2	20	36
010	1:4	40	72
011	1:8	80	144
100	1:16	160	288
101	1:32	320	576
110	1:64	640	1.152s
111	1:128	1.28s	2.304s

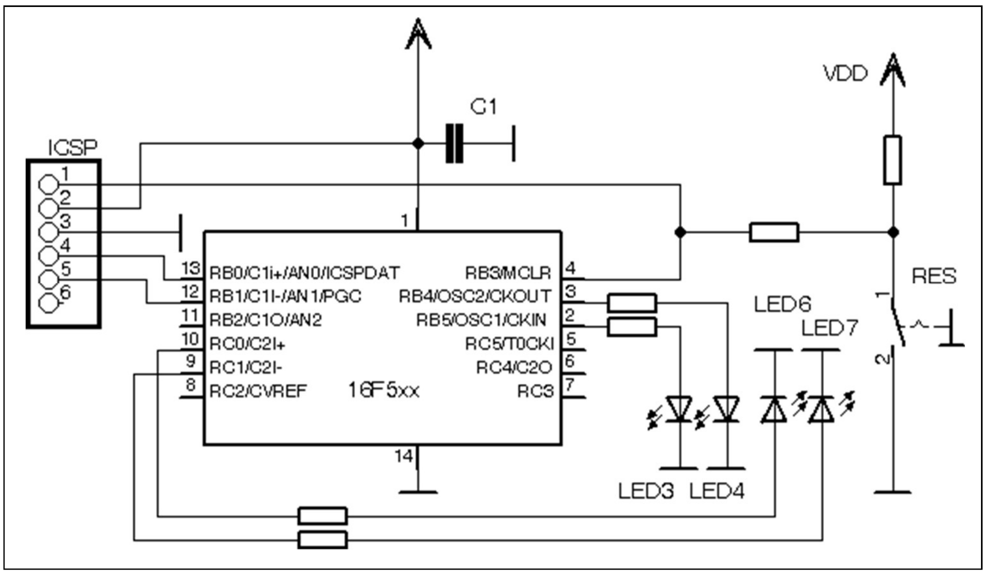
The clrwdt **`opcode`** resets both the counter and the contents of the

prescaler to zero.
The indicated times vary significantly with the temperature, proportionally (low temperature, shorter time), so much so that Microchip can write the application AN720 - Measuring Temperature Using the Watchdog Timer, to which we refer for more details. This must be taken into account if you plan to use the microcontroller in a wide range of ambient temperatures, setting the clear at a cadence appropriate to the minimum time that can be assessed. In addition, very important:

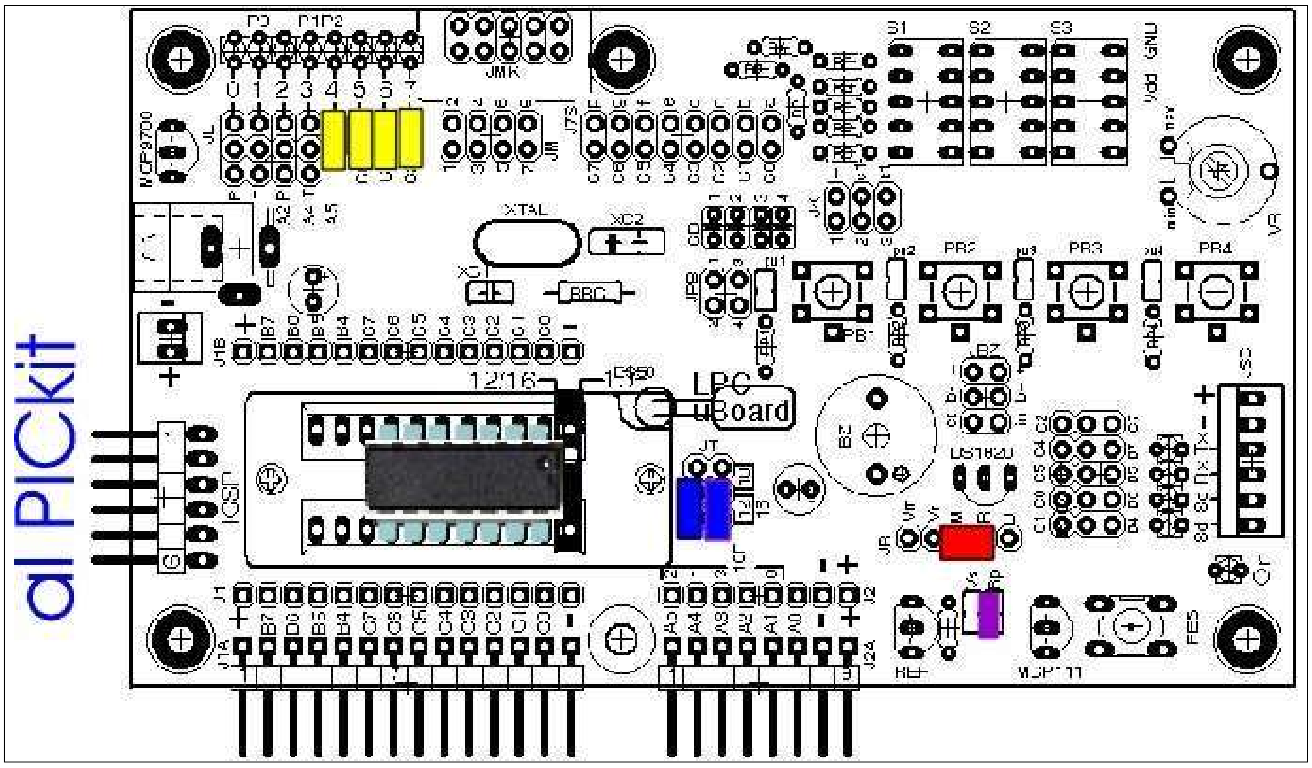
The WDT overflow time, not depending on the main clock, does not depend on its frequency and the timeout occurs for the same time whether Fosc is 4MHz or 20MHz.

Checking How WDT Resets the Microcontroller

We can pretty much see how WDT creates a reset situation.
Let's set up a program like the one in the tutorial that turns on the LEDs in a row one after the other.



And on the **LPCuB**: .



The program is simple: every 1/2 second you add a lit LED to the row, until you have them all lit up and then the cycle starts all over again. In fact, the last one is turned on initially, then turned off after 50ms and switched to the first, then the second, and so on. We've set up the config like this:

```
 _config   _IntRC_OSC  &  _WDT_OFF  &  _CP_OFF  &  _MCLRE_ON
```

The listing can be found in the ***14A_5xx_1.asm*** file, compiled for **16F505/506/526**. If we run it, we see the LED flux described.

Now, let's make a single variation in the config (source ***14A_5xx_2.asm***):

```
 _config   _IntRC_OSC  &  _WDT_ON  &  _CP_OFF  &  _MCLRE_ON
```

and we compile the program: no LED will light up, except the last one! Because? Because we have activated the WDT, which, after 18ms from the start of the program, intervenes by continuously returning the execution to the reset vector and preventing the instructions beyond the first time loop. In fact, the LED7 looks like it's on, but it's flashing very fast. By checking the RC1 pin with a logic probe or oscilloscope , we can see the continuous level switching of the pin. The precise sequence is this:

1. at the POR the PC points to the reset vector and starts executing the instructions
2. having WDT enabled, this timer also starts counting up. We reset it to make sure we have at least the minimum time expected
3. the program turns on the LED and tries to keep it on for 1/2 second
4. the instructions continue, but are stopped by the WDT overflow (while the tempo routine is running)
5. the processor is reset by the intervention of the WDT and the cycle resumes from the reset vector, without ever being able to affect the rest of the instructions

... and how not to have it reset.

So, if we enable WDT, to prevent it from constantly coming into action, we also have to manage it; The function of the watchdog is to block the running program in case of emergency, i.e. when the "conductor" (program) can no longer press the "dead man" button (i.e. perform the clear WDT to prevent it from overflowing). If the program runs smoothly and you don't need to use the watchdog's intervention for a wakeup from the sleep condition, it is imperative to prevent it from overflowing at unsolicited times.

How can we intervene on the previous program so that the LED sequence continues to unfold, despite the presence of the enabled watchdog?
Simply by inserting the delete instruction at a rate that allows you to keep the WDT counter zeroed.

This can be done simply by inserting **`clrwdt`** in the tempo routine so that it repeats with a period lower than the 18ms typical of WDT (better a much lower one to parry overflow limit conditions towards 9ms determined, as said by tolerances or temperature)

The time routine uses Timer0. This needs to be initialized to use the internal clock, with a 1:256 predisvisor.

```
; clock internal (no T0CKI), prescaler 1:256 a
       Timer0 movlw b'11000111'
       option
```

This is possible because the WDT does not employ prescalers and therefore this can be assigned to the timer:

```
 ; W x 50ms delay with clrwdt using Timer0
Dly50msTWDW movwf D1                ; Save number of
                                      repetitions
T0W00 CLRF      TMR0                ; initialize TMR0 <---- |
T0W01 CLRWDT                        ; reset WDT <-------| |
       movf     TMR0,w              ;                   | |
       xorlw    .195                ; Approx. 50ms @4MHz | |
       skpz                         ;                   | |
        Goto    T0W01               ; >-----------------| |
       decfsz   D1,F                ; End of repetitions?  |
        Goto    tow00               ; No - another cycle > - |
       retlw    0                   ; si - return
```

As you can see, it is possible to use Timer0 and WDT at the same time, as long as only one of them needs the prescaler.

With this change, the WDT starts counting when turned on, but after a few micro seconds the time routine kicks in and periodically clears the count, preventing overflow. This would happen on average after 18ms, but `clrwdt` happens much more frequently (every few micro seconds, which certainly keeps the watchdog counter reset even at low temperatures, where the overflow time is reduced) and WDT, despite being operational, never gets to reset the processor.

If we recompile the modified program (***14A_5xx_3.asm***), the sequence of LED lights will repeat without interruption, since we prevent the WDT from reaching overflow.

WDT and sleep

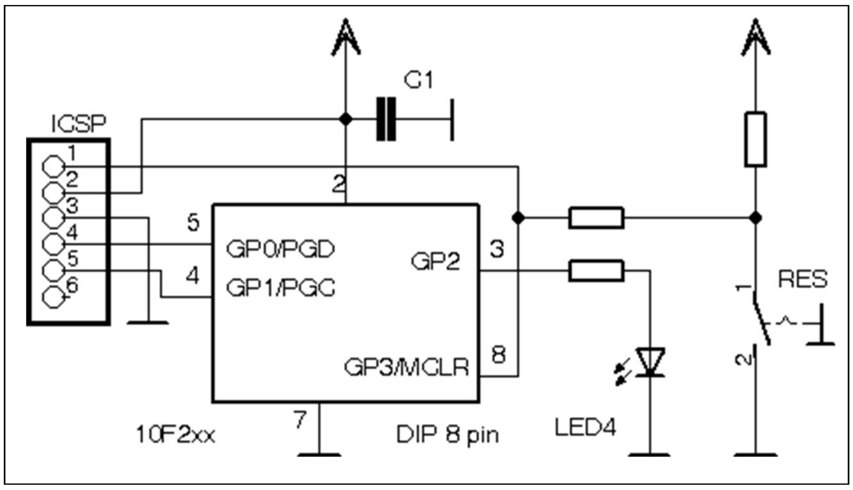
We have said that the WDT also operates in sleep mode, precisely because its operation does not depend on the main clock and therefore it is operational even in sleep conditions: it is possible to exit sleep with the WDT overflow, which triggers a reset.

Thus, WDT can also be used as a coarse wake-up timer of the processor to reduce power consumption: once the necessary operation has been performed, the program sends the micro to sleep, reducing the absorbed current from milliamperes to microamperes. When the overflow time expires, the watchdog restarts the processor, which resumes execution from memory address 0000, performs another operation, and then falls back to sleep. Such operation is suitable, for example, in a battery-powered data logger application, where a measurement is periodically made. The quantity to be measured does not have to be kept under control continuously, but it is enough to sample it periodically and this takes a handful of microseconds; In the rest of the time, the micro does not need to be operational and the sleep condition drastically reduces consumption, prolonging

the life of the batteries.
A sleep-WDT combination could be used, for example, for a data logger or a system that needs to perform an operation at a certain cadence: configure the WDT for the desired time interval, then return the micro to sleep mode. At the end of the set time, the WDT will wake up the system.

We can see a simple application to make the usual LED flash using the WDT and sleep combination.

Let's use a 10F2xx in the diagram of Tutorial 2A:



The program turns on an LED for 1 second, then turns it off and puts the processor to sleep:

```
; pre clear output
          clrf       GPIO
; configure GP2 as out
          movlw      b'00111011'
          TRIS       GPIO

; disable TOCS to free GP2
; OPTION default  11111111
;                 1-------     !GPWU disable
;                 -1------     !GPPU disable
;                 --0-----     disabilita TOCS
;                 ---1----     falling edge
;                 ----1---     prescaler al WDT
;                 -----111     rate 1:128
        movlw   b'11011111'
        option
; on LED per 1s
        bsf       LED
        movlw     .200
        call      Delay10msW
; off LED
        bcf       LED
; stop the program
        sleep              ; Processor Stuck
```

The LED will be off and the processor will be locked: power consumption is reduced to uA.
As in the previous example, when the timeout expires, the processor is reset. However, we took care to set up a prescaler for the WDT in order to bring its overflow to the maximum (prescaler 1:128). We will end up with the LED flashing for 1s with a pause of about 1s. The reason is simple: we have enabled WDT!

- The **POR** starts the processor. The instructions and also the WDT count start
- the **WDT** is set for about 2 seconds
- the LED is lit for 1 second, but at the same time the WDT also counts about 1 second
- the processor goes to sleep and remains in this condition for another second or so, until the WDT overflows. The micro is reset and the cycle starts again: the user sees the LED flashing.

The program (***14A_10F20x.asm***) is applicable to any Baseline PIC with the only modification of the bit/port used for the LED command and the usual variations of config and I/O initialization already widely seen.
In a practical application, instead of flashing an LED, we will insert the instructions necessary for the activity we have to perform; In this way, we will have a cyclical cadence of work and sleep that reduces consumption.

In this regard, a microammeter can be inserted in series with the circuit power supply to check the reduction of current due to the sleep condition.
Such a measure, however, must be made on a suitable circuit in which there are no other components than those interested in the experience. For example, on the LPCuB there are some elements that draw current even though they are not part of the circuit being tested: we mainly have the green LED indicating the presence of the Vdd and the VR potentiometer between Vdd and Vss.
If we intend to make a preliminary evaluation of the circuit current here, it will first be necessary to make a measurement of how much the board absorbs without an on-board processor, a value to

The measurement thus made indicates the average current; the spikes that occur when the LED is turned on should be evaluated differently.

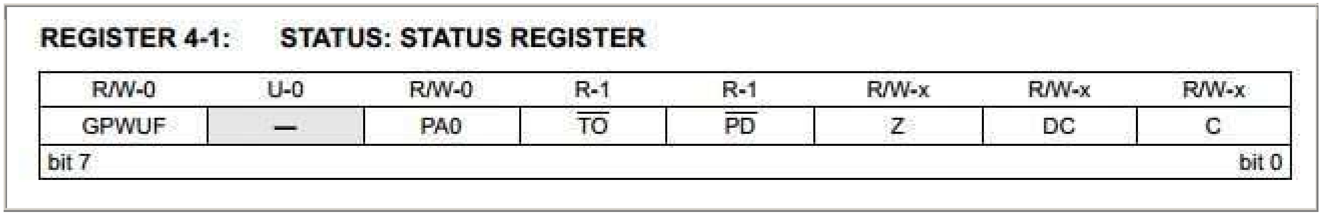
We can see an application of what has been said now in Exercise 18A.

Wake-up for Baselines

We have mentioned that in Baselines the exit from sleep mode (wake-up) is possible through the reset generated by various events:

- **`MCLR :`** with the function enabled by the config, a low level on the pin generates a reset
- **`WDT :`** By enabling WDT from the config, the count overflow generates a reset
- **`Pin Level Change:`** With the function enabled by the STATUS bit, a level change on the pin generates a reset
- **`Comparator:`** with the function enabled by its control register, a change in the state of the output generates a reset

We've seen examples with the watchdog, but the other causes also have the same effect: since they all bring the Program Counter to the reset vector anyway, some **method of distinguishing between them** is desirable.This is possible with the analysis of some flags that are modified depending on the situation: these are the **`TO, PD`** and **`GPWUF/RBWUF bits`** of the **`STATUS.`**
This is the **`STATUS`** situation for 8-pin Baselines:



REGISTER 4-1: STATUS: STATUS REGISTER

R/W-0	U-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
GPWUF	—	PA0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

while for 14-pin ones, the **`PORTB is referred to`** with **`RBWUF`** instead of **`GPIO`** and

`GPWUF.` It should be noted that:

- **the reset generated by the arrival of the supply voltage (POR) and the one produced by the wakeup conditions originate from different internal circuits and produce different effects on SFR defaults**.

For example, for 16F505:

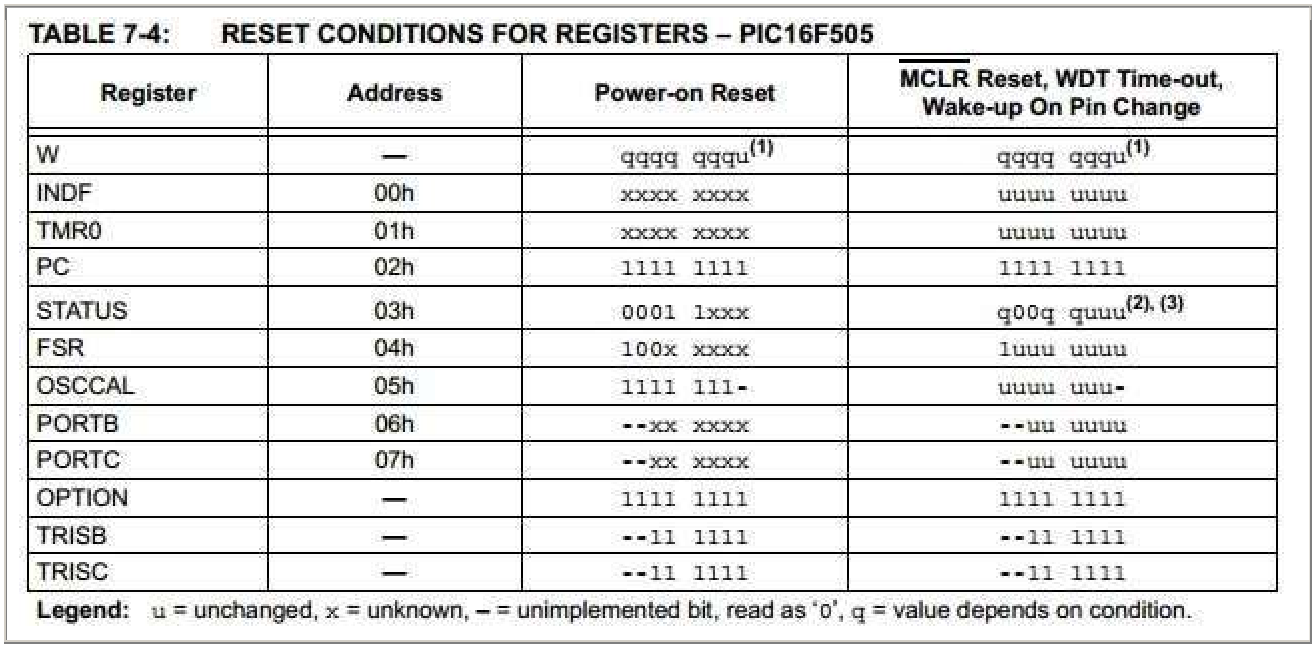


TABLE 7-4: RESET CONDITIONS FOR REGISTERS – PIC16F505

Register	Address	Power-on Reset	$\overline{\text{MCLR}}$ Reset, WDT Time-out, Wake-up On Pin Change
W	—	qqqq qqqu(1)	qqqq qqqu(1)
INDF	00h	xxxx xxxx	uuuu uuuu
TMR0	01h	xxxx xxxx	uuuu uuuu
PC	02h	1111 1111	1111 1111
STATUS	03h	0001 1xxx	q00q quuu(2), (3)
FSR	04h	100x xxxx	1uuu uuuu
OSCCAL	05h	1111 111-	uuuu uuu-
PORTB	06h	--xx xxxx	--uu uuuu
PORTC	07h	--xx xxxx	--uu uuuu
OPTION	—	1111 1111	1111 1111
TRISB	—	--11 1111	--11 1111
TRISC	—	--11 1111	--11 1111

Legend: u = unchanged, x = unknown, – = unimplemented bit, read as '0', q = value depends on condition.

Note that, **at wakeup, the following are not modified**:

- The contents of **TMR0** (since the clock had been suspended and therefore the timer count had stopped).
- The value contained in the calibration register of the internal **OSCCAL** oscillator, which, therefore, would not require an additional one, but the W register still contains the factory calibration value (remember that the real reset vector of the Baseline is not 00, but the last location of the program memory where the calibration value is located).
- The indirect addressing registers (which we will see later) **INDF** and **FSR**

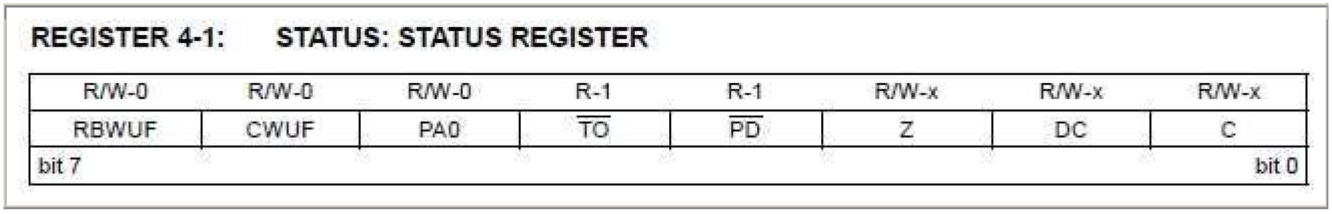
On the other hand, **the following are subject to alterations**:

- The STATUS flags are modified depending on the conditions. This is intended to provide the means to determine the cause of the reset. Note that, as with the **POR,** the selection bit of the **PA0** page is zeroed, pointing to page 0.

- The values of the PORT latches are unchanged, but all bits are set as inputs in the **TRIS registers**. You need to reinitialize the registry every time you wake up.

- The **OPTION** register is reset to FFh and then requires to be initialized at each wakeup.

The situation is similar for all Baselines, given the uniformity we have seen in the family, but it can change depending on the peripherals present. For example, in 16F526, which has comparators, the **STATUS** takes them into account with the **CWUF flag**:



REGISTER 4-1: STATUS: STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
RBWUF	CWUF	PA0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C
bit 7							bit 0

In practice, the indicated flags are modified depending on whether the reset was produced by one or the other cause:

Reset Event		TO	PD	GPWUF RBWUF	CWUF
Reset Not in sleep	POR	1	1	0	0
	WDT	0	u	0	0
	MCLR	u	u	0	0
Reset with WakeUp from sleep	WDT	0	0	0	0
	MCLR	1	0	0	0
	Pin Change	1	0	1	0
	comparator	1	0	0	1

where **u** indicates that the flag retains its previous value. An analysis of these conditions, carried out immediately at the reset vector, allows the program to recognize the motivation of the reset itself and thus adjust the decisions.

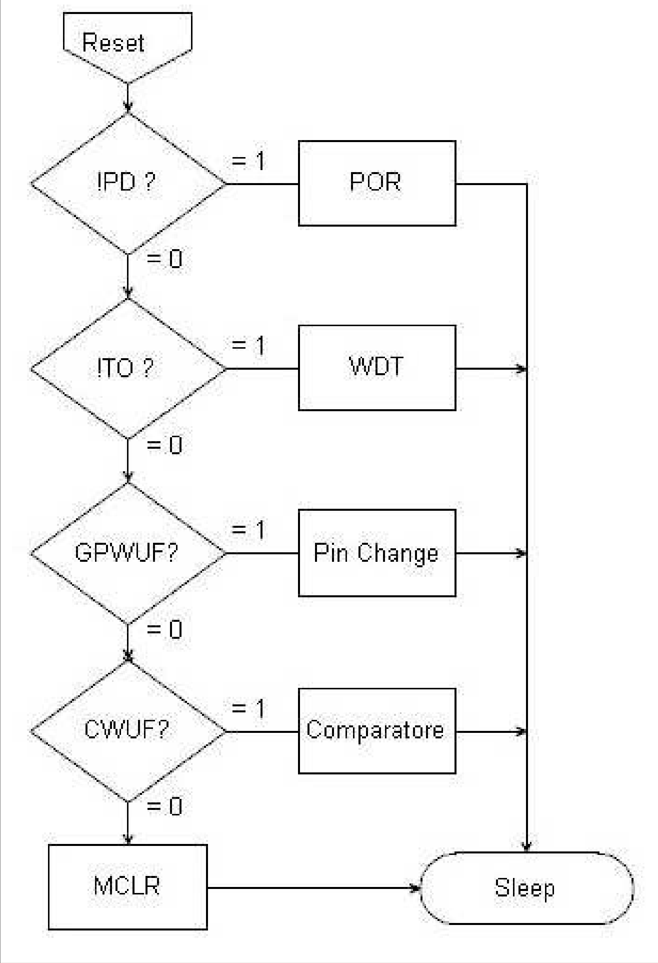
Let's start by verifying in practice how it is possible to diagnose the cause of the reset.

Identify the cause of a reset

We have often said that the **Baseline PICs** are the "simplest" components and are not equipped with particular sophistication; among these there is a complete lack of interrupt management, so it is not possible to treat in this way specific events that in the other families of PICs, through the management of the interrupt, allow the presence of more tasks within the program. In Baselines, a different (and cumbersome) way is used to determine a specific event:

- You send the processor to sleep mode
- Wakes up the processor from a wakeup source

Essentially, sleep/wakeup is a way to respond directly to external signals on I/O.
You can see that Timer0 is not a wakeup source because, since the clock is suspended, it stops counting. For the same reason, the ADC conversion module, where it is present, is also not a wakeup source and is idle during the sleep condition, even if it is enabled.



Basically, there are three pin-dependent events:

1. the state change on an I/O pin configured as input
2. the status of the comparator (where present)
3. MCLR

In addition to two dependent on internal functions:

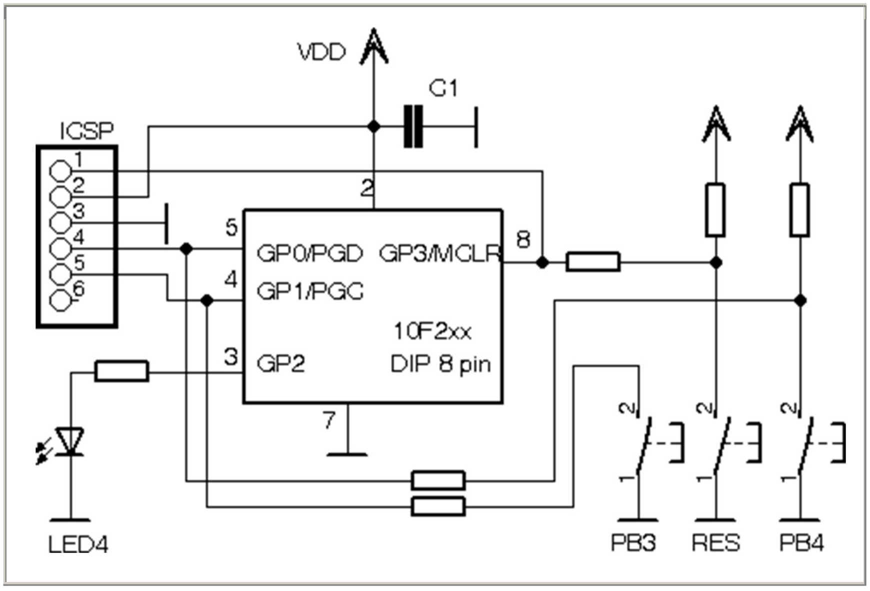
4. POR or WDT

The way to determine what the cause is, in order to initiate the necessary actions, is the one indicated above, i.e. checking the flags in the **`STATUS register`**.

So let's create a simple system that tells us, by lighting up a couple of LEDs, what was the cause of the reset.

The flow chart on the side clearly describes the action.

From a practical point of view, we use this scheme:



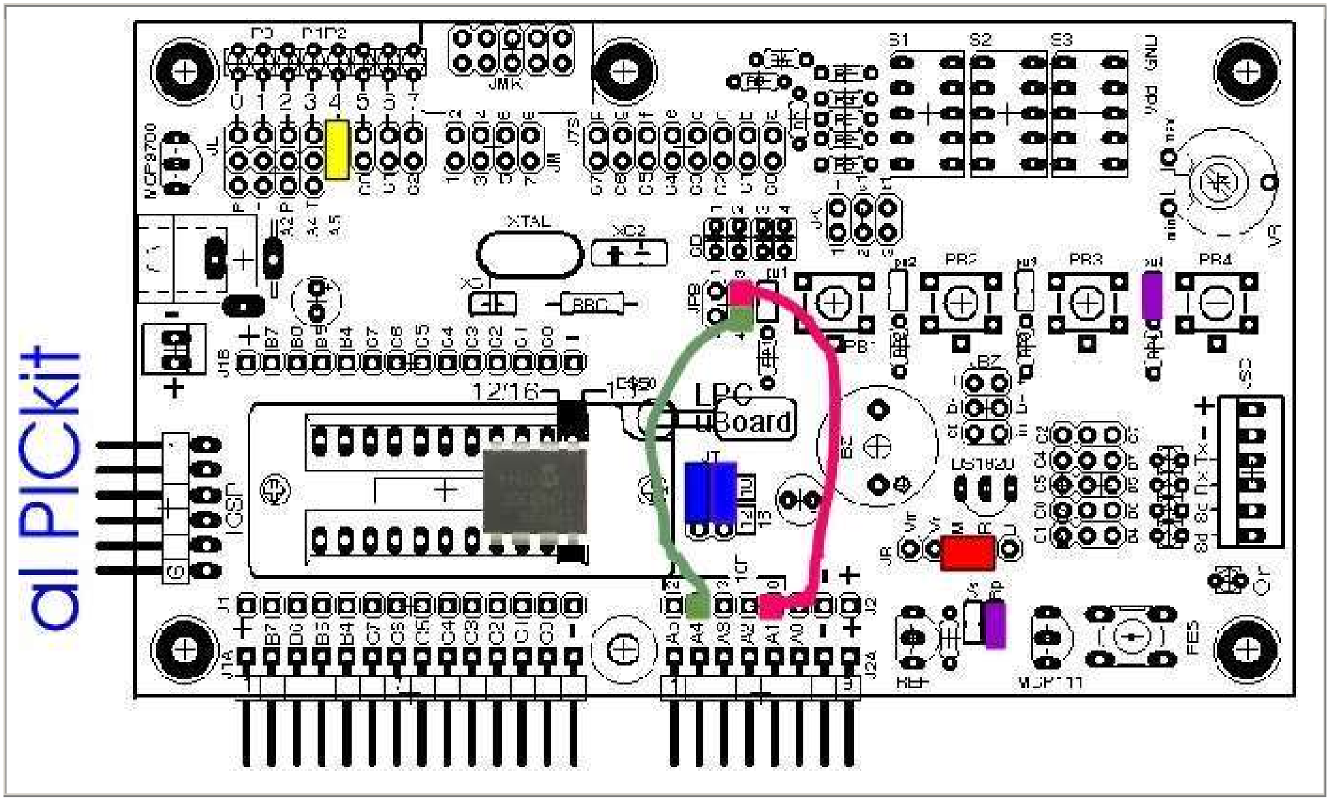
The **RES button** acts on the **`GP3/MCLR`**.
The **PB3** button acts on **`GP1`**/***Pin on change***; it does not require pull-ups because the built-in weak pull-ups are used.
The **PB4** button works by bringing the **`GP0/CIN+ input`** of the comparator to a low level. Requires pull ups. The LED is controlled by **`GP2`**.

How it works is as follows:

- the first time we apply the voltage to the circuit, the program will identify the reset as generated by the POR and the consequent action will be carried out. Then the processor will be put to sleep.
- If we don't press any buttons, the WDT will wake up the processor after about 18ms, making the relative wakeup recognize.
- If, during the sleep time, we act on the buttons, we produce an awakening due to the switching of the comparator (**PB3**), whose `CIN+` input is brought to a low level, while the other input is connected to the internal reference voltage
- or to the change of state of the `GP1` pin, digital input, which is kept high by the integrated weak pull-up and is set to low level by pressing the button (**PB4**), with different LED responses.
- At any time, by pressing the **RES** button, we reset the micro for **MCLR**, with the relative response of the LED.

The cycle repeats until we remove the tension.

On the LPCuB development board:



The "yellow" jumper connects `GP2` to LED4.
The "green" and "red" flying jumpers connect **PB3** to `GP0` and **PB4** to `GP1` **`respectively`**. The "purple" jumpers add the necessary pull-ups.
The "blue" jumpers select the type of chip (also note the arrangement in the socket).

Once the chip is programmed, the Pickit must be detached from the ICSP connection for proper operation.

This is due to the fact that **`GP0`**, **`GP1`** and **`GP3`** are used for the buttons, but at the same time they are the ICSP signal lines, to which the Pickit adds its internal circuitry.

The program

The source (***14A_10ftst.asm***) is compileable for **10F200/2/4/6**: it's nothing complex, once the mechanisms related to sleep and wakeup are clear.

The abundant initial commentary makes the source text self-documenting for the entire project.

After the reset, the first operations are to search for the cause, analyzing the flags of the **`STATUS`**:

```
    BTFSC   STATUS,NOT_PD   ; Tests for the ROP
     Goto   POR_res         ; Yes - Event Management

    BTFSS   STATUS,NOT_TO   ; No - Testing for Watchdog
                            Timeout
     Goto   WDT_res         ; Yes - Event Management
; no - initializes I/O for subsequent tests
    call    IO_Init
    btfsc STATUS,GPWUF      ; Test per pin change
     goto POC_res           ; Yes - Event

; Only for chips with dial
 gauge
#ifdef procanalog           ; test comparator
    btfsc STATUS,CWUF       ; event management
 #endif
; if it's none of the above it's MCLR
    goto    MCLR_res
```

The bits of the STATUS are tested according to the table above. If
! PD = 1 is reset for Power ON (**POR)**
If **! PD=0** and **! TO=0** reset is caused by **WDT**.
Otherwise, it may be a pin change reset, identified by the relative **`GPWUF flag`**.
If the chip has a comparator, the relevant **`CWUF`** flag is checked. The **`C2IN-`** input is connected to ***the internal Vref*** and the output is not enabled.
If none of the above conditions exist, it is a reset caused by **MCLR**.
I/O initialization, which is done after some tests and before others so as not to change flags, is inserted as a subroutine.
Internal weak pull-ups are also enabled which makes an external pull-up useless on **`GP1`** (digital input, used for pin on change), but not on **`GP0`** which is analog input of the

comparator and should be kept at a high level.
For the tempo routine we use the version that employs Timer0, with the `clrwdt` instruction inserted into the loop to keep the watchdog "in bed" while the LEDs are actuated.

The following actions have been chosen on the LED:

- **POR** : LED flashes 2 times
- **MCLR** : LED off for 2 seconds
- **Pin change** : LED flashes 3 times
- **Comparator** : LED flashes 4 times
- **WDT** : LED Flashes Fast

Obviously, it is possible to obtain any other action by changing the relative lines of the program.
The management of flashes is delegated to a Macro, whose purpose is to make the source more readable, since its content will be compiled only in the final .hex and appears in the .lst file only if we have enabled the expansion of macros.
It is a Macro with a parameter that allows you to enter the number of flashes you want.

We could also write the thing in the form of a subroutine; this is possible since you do not exceed the two stack levels available in the Baselines.
In that case we would have the source ***15A_10fts1t.asm***. You can compare the two texts to see the differences. Essentially, the subroutine version will take up less program memory (75 vs. 96); On the other hand, it will require more execution time due to calls and related returns (but in this case it is of little importance, given the non-critical nature of the timings and the activity that the micro must perform).

It should be noted that it is necessary to clear the intervention flags of the device that produced the reset, once the event has been served. If this is not done, immediately after entering sleep the system finds the flag still active and generates an unwanted reset:

- The pin change flag is obtained by simply re-reading the port to which the pin belongs and the **`GPWUF flag`** of the **`STATUS`**
- The **`! TO`** and **`! PD`** can be deleted by acting directly on the **`STATUS`**
- The same goes for the comparator, acting on **`CWUF`**
- The WDT flag is cleared by the `clrwdt statement`

In addition, it is necessary to delete the flags even in the initial phase after the POR, since usually the event sets them, as in the case of pin change.

The example is not limited to **10F20x**, but is applicable to any other Baseline, with the usual modifications.

Pin Change

In order to be able to perform some actions, it is essential to have the ability to interrupt, with external signals, the action of the program to manage asynchronous events. This task is carried out by a mechanism inside the ports for which an intervention request can be generated when a pin, set as input, changes logical level (Pin Level Change).
In the Baselines, as we know, there is no interrupt management, but there is the Pin Change function which has a slightly different action: once the Pin Ch'ange request is enabled and the processor is placed in sleep mode, at the time of the level change on the pin there will be a wake up, similar to the one seen for the WDT, and that leads the action of the processor back to the reset vector.

This mechanism is essential in applications that use contact-controlled inputs (buttons, keypads, limit switches, etc.): the processor is normally idle, in low-power sleep and the contact actuation wakes it up to perform a certain action, then returns to sleep to save energy once the operations are completed.

Depending on the number of pins, pin level change wake up is available on some, but not all:

pin	Pin Change
6	**GP0, 1, 3**
8	**GP0, 1, 3**
14	**RB0, 1, 3, 4**

We see that:

- The GP2 `or` RB2 `pin` does not have this function
- The RB5 `pin` `and` the PORTC `pins` do not have

this function.

- Since the **GP3**/*RB3* pin is shared by **MCLR**, in order to use it as an input you need to disable the reset with **_MCLRE_OFF** in the config.
- All pins that are intended to be used as level change sensors must be set as digital inputs
- It is not possible to assign the Pin Change function to individual pins, but, once enabled, all available pins (which support the event and are programmed as digital inputs) are source for wake up.

The function is enabled by the **!RBWU** of the **OPTION Registry**:

```
; disabilita TOCKI da RB5,abilita Pin change e weak pull up
        ;      b'00010111'
        ;        0-------      GPWU Disabled
        ;        -0------      GPPU disabled
        ;        --0-----      Internal Clock
        ;        ---1----      Falling
        ;        ----0---      prescaler to TMR0
        ;        -----111      1:256
        movlw  b'00010111'
        OPTION
```

We remind you that, depending on the number of pins, some labels are different:

pin	I/O	Pin Change	Weak

			Pull-ups
6	**GPIO**	**! GPWU**	**! GPPU**
8	**GPIO**	**! GPWU**	**! GPPU**
14	**PORTB**	**! RBWU**	**! RBPU**

Essentially, mnemonics refer to **GPIO (GP)** or **PORTB (RB).**

If we also enable built-in weak pull-ups, no external pull-ups are required, but, of course, if there are external pull-ups there is no need to enable internal pull-ups. Built-in weak pull-ups are intended for two reasons: to avoid external components and to have low-power pull-ups.

Again, weak pull-ups and pin changes are enabled or disabled on all possible pins at the same time, but only if these are configured as digital inputs. If they are configured as outputs or are dedicated to other functions, neither weak pull-ups, nor wake ups for pin level change will be activated.

The cause of exiting sleep mode is recognized through the **`RBWUF flag`** of the **`STATUS`**, which goes to level 1.

.

You need to pay attention to these two points:

1. Similar to what happens in an interrupt procedure, once the event occurs, the **flags must be** cleared. In particular, it is mandatory to read the PORT or GPIO: this cancels the wakeup condition.

2. On the other hand, **it is necessary to read the PORT or the GPIO <u>even BEFORE</u> sending the processor to sleep**, since if the awakening flag is, for any reason, set, as soon as it goes to sleep the processor will be immediately awakened.

In the event that you use the function on multiple pins, there may be the problem of recognizing which was the one that caused the awakening event. This can be done by using the PORT reading before sleep to save a copy, which can then be compared with the port reading after wakeup: the difference will indicate which pin has changed level.

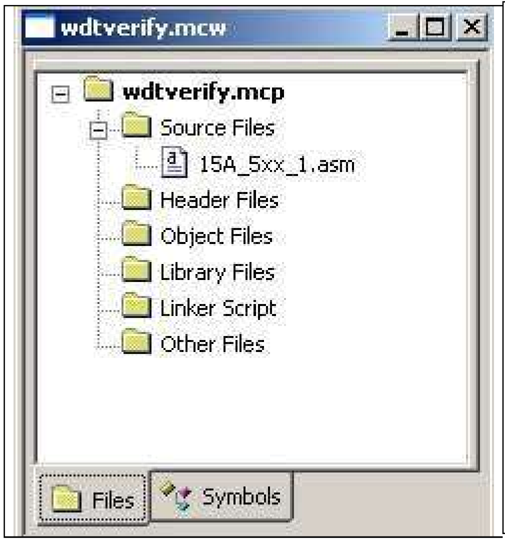
An application for wakeup from pin change is presented in Tutorial 17A.

Appendix.
MPLAB – try different sources without changing projects.

In this tutorial, we have several versions of the same source available, with minor modifications:

- ***`15A_5xx_1.asm, 15A_5xx_1.asm, 15A_5xx_1.asm`***
- ***`15A_10ftst.asm`*** and ***`15A_10ftst1.asm`***

One way is to create a project for each source. The other, more functional, is to try each source in the same project.

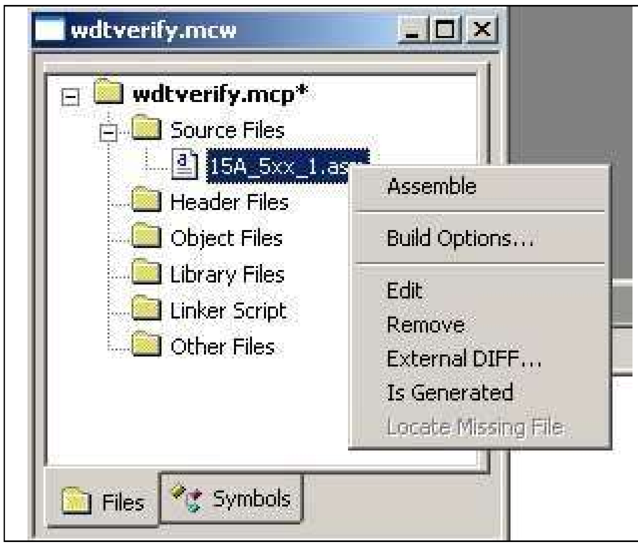


The project provided with the tutorial integrates the first source, so in the resources window we have the situation visible on the side.

In the virtual folder "***Source Files***" the source is ***`15A_5xx_1.asm`***
And this will be compiled.

Once the program has been run and the operation has been verified, we can keep the project environment unchanged (PIC, Programmer, etc.) and change only the source.

If the compilation involves more than one item, there may be multiple files in the folder. The following steps should be done for all those you want to change.

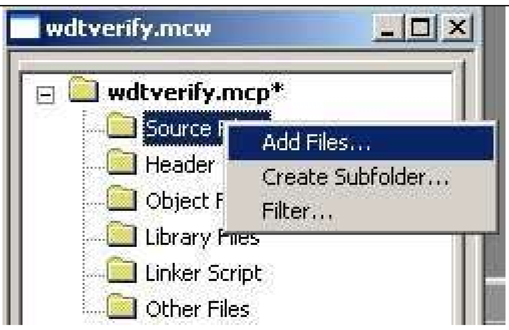


By clicking on the file name with the estro button, open the menu and select "***Remove***": the file will be deleted from the project.

Attention: it is not deleted from the hard disk, but only excluded from the virtual folder "***Source Files***".

The project is now sourceless, but none of the other settings have been changed.

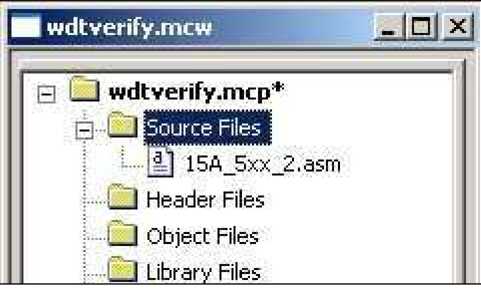
A project also exists without a source, but, of course, it doesn't make sense. All you have to do is add the new source.



Still with the right hand, click on the virtual folder "***Source Files***" for the options menu.

Of course, we choose "Add Files": the classic Windows selection window appears.
We search for the file we want to add and select it.

The added files can be anywhere: MPLAB will insert the necessary path to reach them.



Now there's a new source we can work on.

It should be clear that this is functional if all the sources are dealing with the same processor and if you are using the same environment resources.
If not, it makes more sense to make separate plans.

It can also be commented that a way to reduce the number of files that you have to deal with during the development of a project is to use the statements of the `#if group` (`#if`, `#ifdef`, `#endif`) to condition the compilation, inserting only the lines that are needed, depending on the type of chip or other parameters:

```
 #ifdef __10F202
        LIST      p=10F202
        #include <p10F202.inc>
 #endif
 #ifdef __10F204
        LIST      p=10F204
        #include <p10F204.inc>
#define  proccomp
 #endif

.. . .

 #ifdef  proccomp   ; only 10F204/6
; Disable Comparator
     movlw b'11110111'
     movwf  CMCON0
 #endif
```

A similar system is also to insert alternatives in the source text that will be enabled using the symbol ;, that is, by turning a statement line into a comment line, and vice versa:

```
; no TOCKI, si Pin change e weak pull up, prescaler 1:256 a TMR0
        movlw   b'00010111'
; no TOCKI, no Pin change, no weak pull up, prescaler 1:64 a WDT
;       movlw   b'11011101'
        OPTION
        btfsc   port, pin
         goto   noexec
;       goto    exec        ; Alternative Testing
        bcf     LED1
;       bsf     LED6        ; altern. For debug
```

Here the third line is enabled for compilation while the sixth is not; However, by swapping the comment mark, we select the second solution. So we can add (last line) an action that is not used for execution, but only to verify that it proceeds as intended.
In the final version, it will be possible to eliminate the extra lines, if they limit the reading of the source.
Even at the template level, it is useful to start from the most extensive forms, using the same method:

```
;*******************************************************************
;===================================================================
;            DEFINITION OF PORT USE
;P ORTC map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|--------|--------|--------|--------|--------|--------|
;|    |    |    |    | LED7| LED6|
;
#define      LED6 PORTC,0    ;
#define      LED7 PORTC,1    ;
;#define          PORTC,2    ;
;#define          PORTC,3    ;
;#define          PORTC,4    ;
;#define          PORTC,5    ;

;P ORTB map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|--------|--------|--------|--------|--------|--------|
;| LED5| LED4|    |    |    |    |
;
;#define           PORTB,0 ;
;#define           PORTB,1 ;
;#define           PORTB,2 ;
;#define           PORTB,3   ;  MCLR
#define LED4 PORTB,4 ; LED4 #define
LED5 PORTB,5 ; LED5
```

Even if only a few pins are used, it makes sense to use the full template. This allows us to immediately have two advantages: we have an eye on the complete situation and we can change it in an instant, simply commenting and uncommenting the lines and adding as needed, without rewriting anything.

So, if we have to vary the output LEDs, add a button and a buzzer, there is no need to rewrite everything, but it will be enough to vary what is already available.
We remind you that the lines of comment, whatever their content and however many they may be, do not

They become part of the build, but are only printed in the ***.lst file***.

```
;**********************************************************************
;======================================================================
;             DEFINITION OF PORT USE
;PORTC map
;|  5  |  4  |  3  |  2  |  1  |  0  |
;|-----|-----|-----|-----|-----|-----|
;| bzz | PB2 |     | LED8| LED7| LED6|
;
#define      LED6  PORTC,0    ;
#define      LED7  PORTC,1    ;
#define      LED8  PORTC,2    ; Additional LED
;#define           PORTC,3    ;
#define      PB2   PORTC,4    ; PB2 button
#define      BZZ   PORTC,5    ; Buzzer Command
```

This way of operating is not an obligation, but only a tip to save time and always have the situation under control.

A further caveat regarding compilation: **if the source file is not in the real folder of the project and the defaults are not changed, the files due to the compilation do not end up in the project folder, but in the source folder of the source.** This does not create any problems for the MPLAB environment, nor for the operations that are done through it, since the concept is to be able to have the files anywhere and still find them in the virtual folders of the MPLAB project.
However, it can create some havoc when you go outside the environment.
The advice, until you are familiar with MPLAB, is to physically collect the files necessary for a project in the same folder, in order to easily have the various products of the compilation and not find them scattered everywhere.

Dly5msWDW.asm

```
;*******************************************************************
; Dly5msWDW.asm
;-------------------------------------------------------------------
;
;    Title          : Assembly&C Course -
;                     Delay subroutines for 4MHz clocks.
;                     Performs n loops of 5ms according to N times indicated
;                     in W with clrwdt every 5ms.
;                     Absolute version.
;    PIC            :Baseline
;    Support        : MPASM
;    Version        : V.509-1.0
;    Date           : 01-05-2013
;    Hardware ref. :
;    Author         :Afg
;
;*******************************************************************
; ##################################################################
;===================================================================
;                         RESOURCES
; you need to declare in the main d1,d2,d3 temporary RAM for the counters
;
; ##################################################################

Dly5msWDW:
        movwf   D3      ; Save number of repetitions
                        1us
dlwdlp0 movlw   0xE5    ; 4988 US
        movwf   D1
        movlw   0x04
        movwf   D2
dlwlp1  decfsz  D1,F
         Goto   $+2
        decfsz  d2,f
         Goto   dlwlp1
        clrwdt          ; clear WDT every 5ms
        decfsz  D3,F
         Goto   dlwlp0
        retlw   0

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

Dly5msWD_W.asm

```
;********************************************************************
; Dly5msWD_W.asm
;--------------------------------------------------------------------
;
;    Title         : Assembly&C Course -
;                    Delay subroutines for 4MHz clocks.
;                    Performs n loops of 5ms according to N times indicated
;                    in W with clrwdt every 5ms.
;                    RELOCATABLE version.
;    PIC           :Baseline
;    Support       : MPASM
;    Version       : V.509-1.0
;    Date          : 01-05-2013
;    Hardware ref. :
;    Author        :Afg
;
;********************************************************************
; ####################################################################
;====================================================================
       #include <p12F519.inc>     ; any GLOBAL

        Baseline Dly5msWD_W

; ####################################################################
;                            RAM

       UDATA
D1    Res   1        ; Delay counters
D2    Res   1
D3    Res   1
;
; ####################################################################

Dly5msWD_W    TAILS
        movwf   D3      ; Save Reps 1US DLWDLP0 Movlw
                0xE5    ; 4988 US
        movwf   D1
        movlw   0x04
        movwf   D2
DLWLP1 decfsz D1,F
         Goto   $+2
        decfsz d2,f
         Goto   dlwlp1
        clrwdt          ; clear WDT every
        5ms decfsz d3,f
         Goto   dlwlp0
        retlw   0

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

14A_5xx_1.asm

```
;**********************************************************************
; 14A_5xx_1.asm
;----------------------------------------------------------------------
;
;    Title         : Assembly & C Course - Tutorial 14A
;                    Basic program.
;
;    PIC           : 16F505/506/526
;    Support       : MPASM
;    Version       : V.519-1.0
;    Date          : 01-05-2013
;    Hardware ref. :
;    Author        :Afg
;
;----------------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;       16F505/506/526 @ 14 pin
;
;                     |¯¯\/¯¯|
;              Vdd -|1    14|- Vss
;              RB5 -|2    13|- RB0
;              RB4 -|3    12|- RB1
;         RB3/MCLR -|4    11|- RB22
;              RC5 -|5    10|- RC0
;              RC4 -|6     9|- RC1
;              RC3 -|7     8|- RC2
;                     |______|
;
;    Vdd                        1: ++
;    RB5/OSC1/CLKIN             2: Out  LED5
;    RB4/OSC2/CLKOUT            3: Out  LED4
;    RB3/!MCLR/VPP              4: MCLR
;    RC5/T0CKI                  5:
;    RC4[/C2OUT]                6:
;    RC3                        7:
;    RC2[/CVref]                8:
;    RC1[/C2IN-]                9: Out  LED7
;    RC0[/C2IN+]               10: Out  LED6
;    RB2[/C1OUT/AN2]           11:
;    RB1[/C1IN-/AN1]/ICSPC 12:
;    RB0[/C1IN+/AN0]/ICSPD 13:
;    Vss                       14: --
;
;   []solo 16F526
; ######################################################################
; Choice of #ifdef
  16F526 processor
         LIST      p=16F526
         #include <p16F526.inc>
  #endif
  #ifdef 16F505
        LIST      p=16F505
```

```
         #include <p16F505.inc>
  #endif
  #ifdef 16F506
        LIST      p=16F506
         #include <p16F506.inc>
  #endif
        Radix     DEC

; ######################################################################
;                          CONFIGURATION
  #ifdef 16F526
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_CPDF_OFF & _MCLRE_ON
  #endif
  #ifdef 16F505
; Internal oscillator, no WDT, no CP, MCLR
  __config _IntRC_OSC_RB4EN & _WDT_OFF & _CP_OFF & _MCLRE_ON
  #endif
  #ifdef 16F506
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
  __config _IntRC_OSC_RB4EN & _IOSCFS_OFF & _WDT_OFF & _CP_OFF &
_MCLRE_ON
  #endif

; ######################################################################
;                           RAM
; general purpose RAM
        CBLOCK 0x10          ; start of RAM area
      d1,d2,d3               ; Delay counters
        ENDC

 ;*********************************************************************
;=====================================================================
;            DEFINITION OF PORT USE
;
;P ORTC map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;|     |     |     |     | LED7| LED6|
;
#define      LED6 PORTC,0    ;
#define      LED7 PORTC,1    ;
;#define          PORTC,2    ;
;#define          PORTC,3    ;
;#define          PORTC,4    ;
;#define          PORTC,5    ;

;P ORTB map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;| LED5| LED4|     |     |     |     |
;
;#define           PORTB,0    ;
;#define           PORTB,1    ;
;#define           PORTB,2    ;
;#define           PORTB,3    ; MCLR
```

```
#define     LED4   PORTB,4    ; LED4
#define     LED5   PORTB,5    ; LED5
;
; ####################################################################
;                         CONSTANTS
;
LEDtime EQU .50           ; LED switch-on time 50 x 10ms = 500ms

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

; MOWF Internal Oscillator
        Calibration OSCCAL

; ####################################################################
;                        MAIN PROGRAM
Main:
; reset initializations
  #ifdef 16F526
; Disable CLRF Analog Inputs
                ADCON0

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

; disable T0CKI from RB5
         ;      b'11010111'
         ;        1-------     GPWU Disabled
         ;        -1------     GPPU disabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----1---     prescaler at WDT
         ;        -----000     1:1
        movlw   b'11011000'
        OPTION

        CLRF    PORTB        ; CLRF Port Clear
        Latch   PORTC

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

MainLP BSF      LED7         ; LED7 on
        movlw   LEDtime      ; Call Hold
                Delay10msW

        clrf PORTC ; LED7:6 off clrf PORTB
        ; LED5:4 off movlw LEDtime ; waiting
        call Delay10msW
        Bsf     LED4        ; LED5 on
        movlw   LEDtime     ; waiting
        call    Delay10msW

        BSF     LED5
        Movlw   LEDtime     ; waiting
        Call    Delay10msW

        Bsf     LED6        ; LED6 on
        movlw   LEDtime     ; waiting
        call    Delay10msW

        Goto    mainlp

; ######################################################################
;                       SUBROUTINES

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

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

14A_5xx_2.asm

```
;**********************************************************************
; 14A_5xx_2.asm
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 14A
;                     Check reset from WDT.
;    PIC            : 16F505/506/526
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref. :
;    Author         :Afg
;
;----------------------------------------------------------------------
;
; Pin use :
;  -------------------------
;       16F505/506/526 @ 14 pin
;
;                    |¯¯\/¯¯|
;              Vdd -|1   14|- Vss
;              RB5 -|2   13|- RB0
;              RB4 -|3   12|- RB1
;         RB3/MCLR -|4   11|- RB22
;              RC5 -|5   10|- RC0
;              RC4 -|6    9|- RC1
;              RC3 -|7    8|- RC2
;                    |______|
;
;    Vdd                        1: ++
;    RB5/OSC1/CLKIN             2: Out LED5
;    RB4/OSC2/CLKOUT            3: Out LED4
;    RB3/! MCLR/VPP             4: MCLR
;    RC5/T0CKI                  5:
;    RC4[/C2OUT]                6:
;    RC3                        7:
;    RC2[/CVref]                8:
;    RC1[/C2IN-]                9: Out LED7
;    RC0[/C2IN+]               10: Out LED6
;    RB2[/C1OUT/AN2]           11:
;    RB1[/C1IN-/AN1]/ICSPC 12:
;    RB0[/C1IN+/AN0]/ICSPD 13:
;    Vss                       14: --
;
;   []16F526 only
; #####################################################################
; Choice of #ifdef
  16F526 processor
         LIST      p=16F526
         #include <p16F526.inc>
  #endif
  #ifdef 16F505
        LIST      p=16F505
         #include <p16F505.inc>
  #endif
  #ifdef 16F506
        LIST      p=16F506
         #include <p16F506.inc>
  #endif
        Radix     DEC

; #####################################################################
;                          CONFIGURATION
  #ifdef 16F526
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_ON & _CP_OFF & _CPDF_OFF
& _MCLRE_ON
  #endif
  #ifdef 16F505
; Internal oscillator, no WDT, no CP, MCLR
  __config _IntRC_OSC_RB4EN & _WDT_ON & _CP_OFF & _MCLRE_ON
  #endif
  #ifdef 16F506
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
  __config _IntRC_OSC_RB4EN & _IOSCFS_OFF & _WDT_ON & _CP_OFF &
_MCLRE_ON
  #endif

; #####################################################################
;                           RAM
; general purpose RAM
        CBLOCK 0x10         ; Start RAM
      counter area          ; Counter
      d1,d2,d3              ; ENDC Delay Counters

 ;********************************************************************
;====================================================================
;            DEFINITION OF PORT USE
;
;P ORTC map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;|     |     |     |     | LED7| LED6|
;
#define       LED6 PORTC,0    ;
#define       LED7 PORTC,1    ;
;#define           PORTC,2    ;
;#define           PORTC,3    ;
;#define           PORTC,4    ;
;#define           PORTC,5    ;

;P ORTB map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;| LED5| LED4|     |     |     |     |
;
;#define            PORTB,0    ;
;#define            PORTB,1    ;
;#define            PORTB,2    ;
;#define            PORTB,3    ; MCLR
```

```
#define     LED4   PORTB,4    ; LED4
#define     LED5   PORTB,5    ; LED5
;
; ####################################################################
;                         CONSTANTS
;
LEDtime EQU .50           ; LED switch-on time 50 x 10ms = 500ms

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

; MOWF Internal Oscillator
        Calibration OSCCAL

; ####################################################################
;                        MAIN PROGRAM
Main:
; reset initializations
  #ifdef 16F526
; Disable CLRF Analog Inputs
                ADCON0

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

; disable T0CKI from RB5
         ;      b'11010111'
         ;        1-------     GPWU Disabled
         ;        -1------     GPPU disabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----1---     prescaler at WDT
         ;        -----000     1:1
        movlw   b'11011000'
        OPTION

        CLRF    PORTB       ; CLRF Port Clear
        Latch   PORTC

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

MainLP BSF      LED7        ; LED7 on
        movlw   LEDtime     ; Call Hold
                Delay10msW

        clrf PORTC ; LED7:6 off clrf PORTB
        ; LED5:4 off movlw LEDtime ; waiting
        call Delay10msW
        Bsf     LED4        ; LED5 on
        movlw   LEDtime     ; waiting
        call    Delay10msW

        BSF     LED5
        Movlw   LEDtime     ; waiting
        Call    Delay10msW

        Bsf     LED6        ; LED6 on
        movlw   LEDtime     ; waiting
        call    Delay10msW

        Goto    mainlp

; ####################################################################
;                       SUBROUTINES

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

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

14A_5xx_3.asm

```
;**********************************************************************
; 14A_5xx_3.asm
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 14A
;                     WDT Management
;
;    PIC            : 16F505/506/526
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;----------------------------------------------------------------------
;
; Pin use :
;  -------------------------
;       16F505/506/526 @ 14 pin
;
;                    |¯¯\/¯¯|
;             Vdd -|1   14|- Vss
;             RB5 -|2   13|- RB0
;             RB4 -|3   12|- RB1
;        RB3/MCLR -|4   11|- RB22
;             RC5 -|5   10|- RC0
;             RC4 -|6    9|- RC1
;             RC3 -|7    8|- RC2
;                    |______|
;
;    Vdd                      1: ++
;    RB5/OSC1/CLKIN           2: Out LED5
;    RB4/OSC2/CLKOUT          3: Out LED4
;    RB3/! MCLR/VPP           4: MCLR
;    RC5/T0CKI                5:
;    RC4[/C2OUT]              6:
;    RC3                      7:
;    RC2[/CVref]              8:
;    RC1[/C2IN-]              9: Out LED7
;    RC0[/C2IN+]             10: Out LED6
;    RB2[/C1OUT/AN2]         11:
;    RB1[/C1IN-/AN1]/ICSPC 12:
;    RB0[/C1IN+/AN0]/ICSPD 13:
;    Vss                     14: --
;
;   []16F526 only
; #####################################################################
; Choice of #ifdef
  16F526 processor
         LIST      p=16F526
         #include <p16F526.inc>
  #endif
  #ifdef 16F505
        LIST      p=16F505
```

```
         #include <p16F505.inc>
  #endif
  #ifdef 16F506
        LIST      p=16F506
         #include <p16F506.inc>
  #endif
        Radix     DEC

; ####################################################################
;                          CONFIGURATION
  #ifdef 16F526
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
 __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_ON & _CP_OFF & _CPDF_OFF
& _MCLRE_ON
  #endif
  #ifdef 16F505
; Internal oscillator, no WDT, no CP, MCLR
  __config _IntRC_OSC_RB4EN & _WDT_ON & _CP_OFF & _MCLRE_ON
  #endif
  #ifdef 16F506
; Internal Oscillator, 4MHz, No WDT, No CP, MCLR
  __config _IntRC_OSC_RB4EN & _IOSCFS_OFF & _WDT_ON & _CP_OFF &
_MCLRE_ON
  #endif

; ####################################################################
;                           RAM
; general purpose RAM
        CBLOCK 0x10         ; Start RAM
      counter area          ; Counter
      d1,d2,d3              ; ENDC Delay Counters

 ;*******************************************************************
;===================================================================
;            DEFINITION OF PORT USE
;
;P ORTC map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;|     |     |     |     | LED7| LED6|
;
#define      LED6 PORTC,0    ;
#define      LED7 PORTC,1    ;
;#define          PORTC,2    ;
;#define          PORTC,3    ;
;#define          PORTC,4    ;
;#define          PORTC,5    ;

;P ORTB map
;| 5 |  4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|-----|
;| LED5| LED4|     |     |     |     |
;
;#define           PORTB,0    ;
;#define           PORTB,1    ;
;#define           PORTB,2    ;
```

```
;#define             PORTB,3    ; MCLR
#define     LED4   PORTB,4    ; LED4
#define     LED5   PORTB,5    ; LED5
;
; ######################################################################
;                          CONSTANTS
;
LEDtime EQU .100         ; Switch-on time 100 x 5ms = 500ms

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

; MOWF Internal Oscillator
        Calibration OSCCAL

; ######################################################################
;                         MAIN PROGRAM
Main:
; reset initializations
  #ifdef 16F526
; Disable CLRF Analog Inputs
                ADCON0

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

; disable T0CKI from RB5
         ;      b'11010111'
         ;        1-------     GPWU Disabled
         ;        -1------     GPPU disabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----1---     prescaler at WDT
         ;        -----000     1:1
        movlw   b'11011000'
        OPTION

        CLRF    PORTB       ; CLRF Port Clear
        Latch   PORTC

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

MainLP BSF      LED7        ; LED7 on
        movlw   LEDtime     ; Call Hold
                Dly5msWDW

        clrf PORTC ; LED7:6 off clrf PORTB
        ; LED5:4 off movlw LEDtime ; waiting
        Call    Dly5msWDW

        Bsf     LED4        ; LED5 on
        movlw   LEDtime     ; waiting
        call    Dly5msWDW

        BSF     LED5
        Movlw   LEDtime     ; waiting
        Call    Dly5msWDW

        Bsf     LED6        ; LED6 on
        movlw   LEDtime     ; waiting
        call    Dly5msWDW

        Goto    mainlp


; ####################################################################
;                          SUBROUTINES

; 5ms x W delay subroutine with clrwdt #include
 C:\PIC\Library\Baseline\Dly5msWDW.asm

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

14A_10F20x

```
;**********************************************************************
; 14A_10F20x
;----------------------------------------------------------------------
;
;    Title         : Assembly & C Course - Tutorial 14A
;                    Testing for WDT
;
;    PIC           : 10F200/2/4/6
;    Support       : MPASM
;    Version       A: V.20x-1.0
;    Date          : 01-05-2013
;    Hardware ref. :
;    Author        :Afg
;
;----------------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;     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: Out  LED
;    GP1/ICSPCLK/[CIN-]        4:  3:
;    GP0/ICSPDAT/[CIN+]        5:  1:
;    NC                        6:  nc
;    Vss                       7:  2: --
;    GP3/MCLR/VPP              8:  6: MCLR
;
;  [] solo 10F204/6
;
;**********************************************************************
;======================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-------|-------|-------|-------|
; | BTN | LED4|     |     |
;
;#define        GPIO,GP0    ;
;#define        GPIO,GP1    ;
#define    LED4 GPIO,GP2    ; LED4
;#define        GPIO,GP3    ; MCLR

;
```

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

; ####################################################################
;                       CONFIGURATION
;
 ; No WDT, no CP, pin4=GP3
  __config _WDT_ON & _CP_OFF & _MCLRE_ON

; ####################################################################
;                       RAM
;
; general purpose RAM
        CBLOCK 0x10
   d1,d2,d3                      ; counters for ENDC
        delay

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

 ; MOWF Internal Oscillator
        Calibration OSCCAL

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

; Pre Clear Output
        CLRF    GPIO

; disable T0CS to free GP2
;  OPTION default 11111111
;                 1-------     ! GPWU disable
;                 -1------     ! GPPU disable
;                 --0-----     disable T0CS
;                 ---1----     Falling Edge
;                 ----1---     prescaler at WDT
;                 -----000     Rate 1:128
       movlw   B'11011111
       Option

; GP2 come out
       movlw    b'00111011'
       TRIS     GPIO

; LED light up for 1s
       Bsf       LED
       movlw     .100
       Call      Delay10msW

; turn off LEDs
       Bcf       LED

; Stop the program
       Sleep           ; Locked Processor

 #include C:\PIC\Library\Baseline\Dealy10msW.asm

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

14A_10ftst.asm

```
;**********************************************************************
; 14A_10ftst.asm
;----------------------------------------------------------------------
;
;    Title         : Assembly & C Course - Tutorial 15A
;                    Analysis of the causes of the reset.
;
;    PIC           : 10F200/2/4/6
;    Support       : MPASM
;    Version       : V.519-1.0
;    Date          : 01-05-2013
;    Hardware ref. :
;    Author        :Afg
;----------------------------------------------------------------------
; This program analyzes the cause of the reset in PIC Baseline.
; The reset produced by POR, WDT, MCLR, Port change is identified
; and Comparator.
; The program is compilable for 10F200/2/4/6, but is extensible
; to any other baseline.
; The outputs are assigned as follows:
;     GP0 comparator input
;     GP1 port change input
;     GP2 LED Control Output
;     GP3 MCLR
;
; The LED indicates different situations:
;     POR         2 x 200ms flashes, pause 1/2s
;     MCLR        Off for 2s
;     WDT         Rapid flashing
;     Pin change 3 flashes of 200ms, pause 1/2s
;     Comparator 4 flashes of 200ms, pause 1/2s
;
; When the supply voltage is applied, it will be
; recognized a POR reset. The LED will blink twice, then
; a pause of 1/2s and then flashing fast due to the intervention
; of the WDT.
; If you send the comparator input to a low level, the LED will
; 4 flashes, and then flashes fast again due to WDT.
; If you send the GP1 pin (pin change) to low the LED will make
; 3 flashes, and then flashes fast again due to WDT.
; At any time, pressing the RES button will result in a reset
; from MCLR and the LED will be off for 2s, and then flash again
; velodce because of WDT.
;
; The comparator test is performed only for chips that
; Have (10F204/10F206)
;
;----------------------------------------------------------------------
;
;  Impiego pin :
;  ----------------
;     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: Out  LED
;    GP1/ICSPCLK/[CIN-]        4:  3: In   Pin Change button
;    GP0/ICSPDAT/[CIN+]        5:  1: In   CIN+ button
;    NC                        6:  Nc
;    Vss                       7:  2: --
;    GP3/MCLR/VPP              8:  6: MCLR
;
;  [] only 10F204/6
;
;********************************************************************
;====================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|-----|
; |MCLR | LED | BTN | BTN |
;
;#define        GPIO,GP0    ; Button Input
;#define        GPIO,GP1    ; Button Input
#define   LED   GPIO,GP2    ; LED
;#define        GPIO,GP3    ; MCLR

#define   LEDon   b'00001011'

;********************************************************************
; ####################################################################
; Choice of #ifdef
 10F200 processor
        LIST      p=10F200
        #include <p10F200.inc>
 #endif
 #ifdef 10F202
        LIST      p=10F202
        #include <p10F202.inc>
 #endif
 #ifdef 10F204
        LIST      p=10F204     ; processor with comparator
        #include <p10F204.inc>
#define procanalog
 #endif
 #ifdef 10F206
        LIST      p=10F206     ; processor with comparator
        #include <p10F206.inc>
#define procanalog
 #endif
          radix dec
```

```
; ######################################################################
;                       CONFIGURATION
;
; WDT, no CP, MCLR
  __config _WDT_ON & _CP_OFF & _MCLRE_ON

; ######################################################################
;                       RAM
;
; general purpose RAM
        CBLOCK 0x10
       D1                  ; Counter for Delay
       D2                  ; counter for blink
        ENDC

; ######################################################################
;                       LOCAL MACRO
;
; Macro for LED flashing. 200ms flashes, then pause 1/2s
LEDBLNK   blink MACRO
        movlw   Blink          ; Number of Movwf
        Flashes D2
blnklp bcf      LED            ; Movlw
        off     .4
        Call    Dly50msTWDW    ; for 200ms
        BSF     LED            ; On Movlw
                .4
        Call    Dly50msTWDW    ; for 200ms
        decfsz d2,f            ; other blink ?
         Goto   blnklp         ; Yes
        Bcf     LED            ; no - LED off
        movlw   .10
        Call    Dly50msTWDW    ; for 1/2s
          ENDM

;********************************************************************
;                       PROGRAM START

     ORG     0x0

Main
        BTFSC   STATUS,NOT_PD  ; Tests for POR
        Goto    POR_res        ; Yes - Manage Event

        BTFSS   STATUS,NOT_TO  ; Testing for WDT
        GOTO    WDT_res        ; Yes - Manage Event

        Call    IOinit         ; Initialize I/O before other tests

        BTFSC   STATUS,GPWUF   ; Test for pin change
        GOTO    POC_res        ; yes - manage event

 #ifdef procanalog             ; only for 10F204/206
        btfsc   STATUS,CWUF    ; Tests for goto
        comparator            CMP_res  ; Yes -
        Manage Event
 #endif
```

```
; if it's none of the above, it's MCLR
        goto    MCLR_res

; -----------------------------------------------------------------------------------------------------
;                    Event Handling
; Reset from power
on POR_res
        clrwdt
        Call    IOinit           ; Initialize I/O

; LED Actuation : 2 flashes + off 1/2s
        LEDBLNK 2

; Clear Flags
        movf    GPIO,w           ; Clear Pin Flags on Change
        movf    CMCON0,w         ; Clear Comparator Flag
        CLRF    STATUS           ; Clear other flags
; Sleep
        Sleep


; Reset from
WDT WDT_res
        clrwdt
        Call    IOinit           ; Initialize I/O

; LED Actuation : Fast Flashing
        Bsf     LED              ; On
        Movlw   .1
        Call    Delay
        Bcf     LED              ; off
; Sleep
        Sleep

; Reset for pin change
POC_res
        clrwdt

; LED Actuation : 3 flashes + off 1/2s
        LEDBLNK 3

; Clear Flags
        movf    GPIO,w
        Bcf     STATUS,GPWUF
; Sleep
        Sleep

 #ifdef procanalog
  ; Reset for the CMP_res
comparator
        clrwdt

; LED Operate: 4 Flashes + Off 1/2s
        LEDBLNK 4

; Clear Flags
        movf    CMCON0,w         ; Clear Comparator Flag
        Bcf     STATUS,CWUF      ; Clear Sleep Comparator
        Flag                     ; Sleep
  #endif

 ; Reset for MCLR
MCLR_res
        clrwdt

; LED Actuation : Off 2s
        Bcf     LED
        movlw   .40
        Call    Dly50msTWDW
; Sleep
        Sleep

;*********************************************************************
;                        SUBROUTINES
; initialize
IOinit I/O
        movlw   LEDon           ; LED command pin as out, others in
        trio    GPIO

; clear flag IOC set by movf power up
                GPIO,f

 #ifdef procanalog               ; only for 10F204/206
; comparator on, wakeup reset, GP0 channel, internal reference
; No Output, Normal Polarity
        MOVLW   B'11111000'
        MovWF   CMCON0
 #endif

; enable pin change, internal clock (no T0CKI)
; prescaler 1:256 to Timer0
        movlw   B'00000111'
        Option
        retlw   0

;--------------------------------------------------------------------------------
; 1s delay with clear WDT using Timer0 #include
 C:\PIC\Library\Baseline\Dly50msTWDW.asm

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

14A_10ftst1.asm

```
;**********************************************************************
; 14A_10ftst1.asm
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 15A
;                     Analysis of the causes of the reset.
;                     With subroutines.
;
;    PIC            : 10F200/2/4/6
;    Support        : MPASM
;    Version        : V.519-1.0
;    Date           : 01-05-2013
;    Hardware ref. :
;    Author         :Afg
;----------------------------------------------------------------------
; This program analyzes the cause of the reset in PIC Baseline.
; The reset produced by POR, WDT, MCLR, Port change is identified
; and Comparator.
; The program is compilable for 10F200/2/4/6, but is extensible
; to any other baseline.
; The outputs are assigned as follows:
; GP0 comparator input
; GP1 port change input
; GP2 LED Control Output
; GP3 MCLR
;
; The LED indicates different situations:
; POR         2 x 200ms flashes, pause 1/2s
; MCLR        Off for 2s
; WDT         Rapid flashing
; Pin change 3 flashes of 200ms, pause 1/2s
; Comparator 4 flashes of 200ms, pause 1/2s
;
; When the supply voltage is applied, it will be
; recognized a POR reset. The LED will blink twice, then
; a pause of 1/2s and then flashing fast due to the intervention
; of the WDT.
; If you send the comparator input to a low level, the LED will
; 4 flashes, and then flashes fast again due to WDT.
; If you send the GP1 pin (pin change) to low the LED will make
; 3 flashes, and then flashes fast again due to WDT.
; At any time, pressing the RES button will result in a reset
; from MCLR and the LED will be off for 2s, and then flash again
; velodce because of WDT.
;
; The comparator test is performed only for chips that
; Have (10F204/10F206)
;
;----------------------------------------------------------------------
;
; 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: Out  LED
;    GP1/ICSPCLK/[CIN-]      4:  3: In   Pin Change button
;    GP0/ICSPDAT/[CIN+]      5:  1: In   CIN+ button
;    NC                      6:  NC
;    Vss                     7:  2: --
;    GP3/MCLR/VPP            8:  6: MCLR
;
;  [] only 10F204/6
;
;*********************************************************************
;=====================================================================
;            DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |--------|--------|--------|--------|
; |MCLR | LED | BTN | BTN |
;
;#define        GPIO,GP0    ; Button Input
;#define        GPIO,GP1    ; Button Input
#define   LED   GPIO,GP2    ; LED
;#define        GPIO,GP3    ; MCLR

#define   LEDon   b'00001011'

;*********************************************************************
; ####################################################################
; Choice of #ifdef
 10F200 processor
        LIST      p=10F200
        #include <p10F200.inc>
 #endif
 #ifdef 10F202
        LIST      p=10F202
        #include <p10F202.inc>
 #endif
 #ifdef 10F204
        LIST      p=10F204     ; processor with comparator
        #include <p10F204.inc>
#define procanalog
 #endif
 #ifdef 10F206
        LIST      p=10F206     ; processor with comparator
        #include <p10F206.inc>
#define procanalog
 #endif
```

```
        radix dec
; ###################################################################
;                       CONFIGURATION
;
; WDT, no CP, MCLR
  __config _WDT_ON & _CP_OFF & _MCLRE_ON

; ###################################################################
;                       RAM
;
; general purpose RAM
        CBLOCK 0x10
       D1                  ; Counter for Delay
       D2                  ; counter for blink
        ENDC

;*******************************************************************
;                       PROGRAM START

     ORG     0x0

Main
        BTFSC   STATUS,NOT_PD    ; Tests for POR
         Goto   POR_res          ; Yes - Manage Event

        BTFSS   STATUS,NOT_TO    ; Testing for WDT
         Goto   WDT_res          ; Yes - Manage Event

        Call    IOinit           ; Initialize I/O before other tests

        BTFSC   STATUS,GPWUF     ; Test for pin change
         Goto   POC_res          ; Yes- Manage Event

 #ifdef procanalog               ; only for 10F204/206
        btfsc   STATUS,CWUF      ; Comparator Tests
         Goto   CMP_res          ; Yes - Manage Event
 #endif

; if it's none of the above, it's MCLR
        goto    MCLR_res

; ------------------------------------------------------------------------------------------------
;                   Event Handling
; Reset from power
on POR_res
        clrwdt
        Call    IOinit           ; Initialize I/O

; LED Operation : 2 Flashes + Off 1/2s
        movlw   .2
        Call     Ledblnk

; Clear Flags
        movf    GPIO,w           ; Clear Pin Flags on Change
        movf    CMCON0,w         ; Clear Comparator Flag
        CLRF    STATUS           ; Clear other flags
; Sleep
```

```
        Sleep


; Reset from
WDT WDT_res
        clrwdt
        Call    IOinit           ; Initialize I/O

; LED Actuation : Fast Flashing
        Bsf     LED              ; On
        Movlw   .1
        Call    Delay
        Bcf     LED              ; off
; Sleep
        Sleep

; Reset for pin change
POC_res
        clrwdt

; LED Actuation : 3 Flashes + Off 1/2s
        movlw   .3
        Call     Ledblnk

; Clear Flags
        movf    GPIO,w
        Bcf     STATUS,GPWUF
; Sleep
        Sleep

 #ifdef procanalog
  ; Reset for the CMP_res
comparator
        clrwdt

; LED Operate: 4 Flashes + Off 1/2s
        movlw   .4
        Call     Ledblnk

; Clear Flags
        movf    CMCON0,w         ; Clear    Comparator Flag
        Bcf     STATUS,CWUF      ; Clear    Comparator Flag
        Sleep                    ; Sleep
  #endif

 ; Reset for MCLR
MCLR_res
        clrwdt

; LED Actuation : Off 2s
        Bcf     LED
        movlw   .40
        Call    Dly50msTWDW
; Sleep
        Sleep

 ;*******************************************************************
 ;                        SUBROUTINES
IOinit
        movlw   LEDon           ; LED command pin as out, others in
        trio    GPIO

; clear flag IOC set by movf power up
                GPIO,f

 #ifdef procanalog               ; only for 10F204/206
; comparator on, wakeup reset, GP0 channel, internal reference
; No Output, Normal Polarity
        MOVLW   B'11111000'
        MovWF   CMCON0
 #endif

IOinit1
; enable pin change, internal clock (no T0CKI)
; prescaler 1:256 to Timer0
        movlw   B'00000111'
        Option
        retlw   0

;--------------------------------------------------------------------------------
; 1s delay with clear WDT using Timer0 #include
 C:\PIC\Library\Baseline\Dly50msTWDW.asm
;--------------------------------------------------------------------------------
; LED flashing
Ledblnk
        movwf   D2               ; Save number of
                                   flashes
        Bcf     LED              ; off
        movlw   .4
        Call    Dly50msTWDW      ; for 200ms
        Bsf     LED              ; on
        movlw   .4
        Call    Dly50msTWDW      ; for 200ms
        decfsz  d2,f             ; other blink ?
         Goto   S-9              ; Yes
        Bcf     LED              ; no - LED off
        movlw   .10              ; for 1/2s
        Goto    Dly50msTWDW      ; with Dirty Return

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

Dly50msTWDW.asm

```
;********************************************************************
; Dly50msTWDW.asm
;--------------------------------------------------------------------
;
;    Title          : Assembly & C Course
;                     Delay subroutines for 4MHz clocks.
;                     50ms delay with clear WDT using Timer0
;    PIC            :Baseline
;    Support        : MPASM
;    Version        : V.509-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;********************************************************************
; ####################################################################
;====================================================================
;                            RESOURCES
;
; * declare d1 in RAM, temporary for step counter
; * Timer0 must have a prescaler of 1:256
;
; ####################################################################
;--------------------------------------------------------------
 ; W x 50ms delay with clrwdt using Timer0
Dly50msTWDW
        movwf    D1                ; Save number of
                                     repetitions
T0W00   CLRF     TMR0              ; initialize TMR0 <-----|
T0W01   clrwdt                     ; reset WDT <------| |
        movf     TMR0,w            ;                  | |
        xorlw    .195              ; Approx. 50ms @4MHz |  |
        skpz                       ;                  | |
         Goto    T0W01             ; >------------------ |
        decfsz   D1,F              ; End of repetitions?   |
         Goto    tow00             ; No - other cycle >--- |
        retlw    0                 ; si - return

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