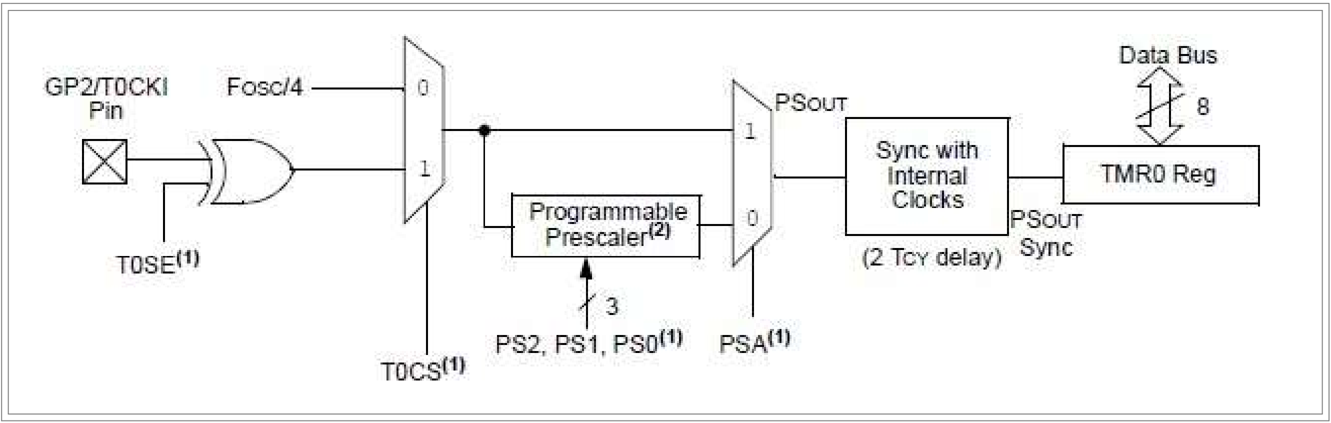
Knowledge of baseline

TIMER0

The purpose of the tutorial is to use a built-in device: the TIMER0.

So far we have seen how it is possible to control digital outputs and inputs, using the so-called digital I/O, the primary function of the microcontroller pins.
However, the PICs also integrate some modules that perform particular tasks, among which one, present on all chips, is a counter-timer: **TIMER0**.

TIMER0, which was originally called RTCC (16C54 and similar), is the only timer available in baselines. As typical of these simplified chips, the timer also comes in the simplest form, and its functional diagram looks like this:



The accessible part is the 8-bit TMR0 count register, readable and writable like a normal RAM location, which is automatically incremented with each clock cycle: at 4MHz there will be an increase of one unit every microseconds (i.e. ***Fosc/4***).
It is a counter that grows (*count up*) and not scales (*count down*) and is of the *free run* type, i.e. it cannot be stopped, as long as the clock and the supply voltage are present.

If we reset the TMR0 count register, after 255 clock pulses it will have reached the **FFh** value and the next pulse will bring it back to 0, to start the count again; this step is called ***overflow***.

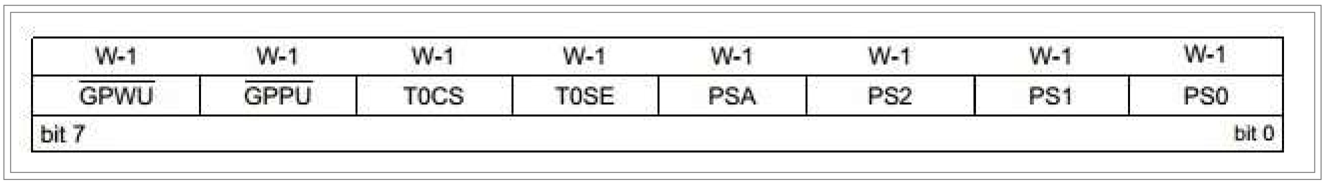
The possibility of writing to the counting register opens up several possibilities: for example, if we want to count a passage of 50 clock pulses (i.e. a time of 50us with the clock at 4MHz) just load the **`TMR0`** with 255-50 = 205 (CDh) and wait for zero to be reached.

The counter ***overflow*** event, in the Baselines, does not have any reporting flags and must be verified by cyclically re-reading the count log (polling). This is a limitation that makes it not always easy to use the timer.

As you can see in the functional diagram above, the **TIMER0** has:

- a *prescaler* that allows you to divide the incoming pulses up to 1:256. This means that, with this value, for example, the timer increases by one unit for every 256 input pulses, greatly expanding the counting capacity and the manageable time.
- In addition, it is given the option to use an **external clock**, which is input from the pin **`T0CKI`**. In this case, the timer can operate as a pulse counter.

TIMER0 does not have its own control register, but its options are managed through a few bits of 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

The functions are:

- bit 5 **`T0CS`** : chooses which clock to send to the timer input. We see it schematized in the functional diagram.
  - **`T0CS`** = 1 input is **`T0CKI`** **(default to POR)**
  - **`T0CS`** = 0 input is the internal clock ***Fosc/4***

- bit 4 **`T0SE`** : Selects the edge of the clock signal on which the timer counts when it comes from the input. The function is schematized with an exclusive OR gate.
  - **`T0SE`** = 1 falling edge **(default to POR)**
  - **`T0SE`** = 0 rising edge

- bit3 **`PSA:`** Select whether the prescaler should be assigned to the ***WatchDog Timer*** (**WDT**) or to **the TIMER0**:
  - **`PSA`** = 1 prescaler at **WDT** **(default at POR)**
  - **`PSA`** = 0 prescaler at **TIMER0**

Let's start with a few observations about these control bits:

- **`T0CS`** offers the possibility of inserting **`T0CKIs`** as an input signal; in this way the **TIMER0** becomes a counter of pulses that come from an external process and that expand the possibilities of applying the timer. Obviously, the input signal will have to meet appropriate characteristics, the first of which is the compatibility of the 0-1 levels with the logic of the microcontroller, while the maximum counting frequency is a little lower than the maximum frequency for the internal clock (data sheet, param.40-42).
  `T0CKI` is a function shared with others on a specific I/O pin (**`GP2`** for PICs with **GPIO** and **`RC5`** for those with **`PORTB/PORTC`**). Since only one of the functions shared by a pin can be used at a time, it is necessary to pay attention to the defaults.

Caution:

- **at the default of the POR, the `T0CS` bit is preset at level 1.**
  This specifies that the `GP2/RC5` pin is set as the `T0CKI` input and cannot be used otherwise until you change `T0CS` from program to level 0.
  So, **if you want to use the pin as digital I/O, you have to program the control bit**.

The setting of `T0CS` = 1 takes precedence over any other and also overrides the eventual assignment of the pin as output made through the TRIS register.
The only exception is the `FOSC4 function` for the 10F220/2, which is done by acting on bit 0 of the `OSCCAL.` Let's mention again the table in the datasheet, where the priority of pin assignments is present:

Priority	GP0	GP1	GP2	GP3
1	AN0	AN1	FOSC4	I/$\overline{\text{MCLR}}$
2	TRIS GPIO	TRIS GPIO	T0CKI	—
3	—	—	TRIS GPIO	—

Attention, therefore:

- By default, when the power supply arrives, the pin automatically acquires the `T0CKI function`
- if we set the 0 bit of `OSCCAL to 1`, the pin takes on the function of FOSC/4
- To have `GP2` it is necessary that the 0 bit of `OSCCAL` is 0, and that the `T0CS bit` of `Option is zeroed by the program`. For example,:

```
; Disable FOSC/4
        Bcf      OSCCAL,FOSC4

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

- `T0SE` allows you to define the front of action of the input signal, in such a way as to be able to discriminate the two situations; this further expands the application possibilities of the timer-counter in relation to particular situations where it is necessary to synchronize on one or the other of the fronts of the input signal.

- `PSA:` The function controls the prescaler and a clarification is needed here:

Baselines have a unique prescaler, common to TIMER0 and WDT.

By default at the POR the bit is at level 1 and the prescaler is assigned to the WDT. If we want to use it for the timer, we need to set the control bit to 0 for example:

```
; disable T0CKI, prescaler to Timer0, 1:256
        movlw    b'11010111'
        OPTION
```

This is a limitation, as the prescaler can only be used by one of the two counters, but this is one of the consequences of the simplified structure of the chips in this family. In any case, it is possible to switch the prescaler to one or the other during operation, switching the control bit. The datasheet provides examples of how to do this:

```
; switch the prescaler from Timer0 to WDT
        clrwdt              ; reset WDT counter
        clrf    TMR0        ; reset Timer0 and
        movlw   b'11011111' ; no T0CKI, presc. at WDT
        OPTION
        clrwdt
        movlw   b'11011101' ; no T0CKI, presc. at WDT, 1:64
        OPTION
```

and:

```
; switch prescaler from WDT to Timer0
        clrwdt              ; reset WDT counter
        movlw   b'11010111' ; no T0CKI, presc. at Timer0,1:256
        OPTION
```

Note that the latter sequence must be executed even if WDT is disabled.

It remains to be seen what the PS2-0 bits function of the **OPTION** bits will be. Their purpose is to set the division factor of the prescaler, according to this table:

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

Note that the predivisor assumes different values in the case of **WDT** or **TIMER0**. In all cases, the values are powers of 2, since the division is done with a flip-flop chain

The general concept is that, if you don't want the pre-divisor for Timer0, just leave the default (or set the **PSA bit to 1)**: the count is unitary (1:1), i.e. an input pulse, an increase in the count. If, on the other hand, the input signal needs to be split, the program will need to set the PSA bit to 0. For example,:

```
; disable T0CKI, presc. 1:256 to Timer0
        ;      b'11010111'
        ;        1-------     GPWU Disabled
        ;        -1------     GPPU disabled
        ;        --0-----     Internal Clock
        ;        ---1----     Falling
        ;        ----0---     prescaler to Timer0
        ;        -----111     1:256
       movlw   b'11010111'
       OPTION
```

For the **TIMER0**, the pre-divider, when engaged, acts both on the internal **Fosc/4** clock and on the signal coming from **T0CKI.** So, if we program set the control bits to binary value 100, the counter will advance by one unit for every 32 input pulses, and so on.

An effect on the time management possibilities can be easily determined with a simple calculation:

@ 4MHz clock, Fosc/4 is worth 1MHz, with period 1us.

With the 8-bit register and the 1:1 predisvisor, the count reaches overflow after 256 pulses, i.e. 256us.
If we insert the 1:256 prescaler, we will get a much longer time, 256 x 256 = 65.536 us, or 65.5ms.

For the sake of completeness, it should be noted that the timer requires, once loaded, two cycles for synchronism, cycles that must be added once to the time count. This must be taken into account in extremely precise applications, where an equally precise clock frequency is required.

You can find **more information about the TIMER0** here.

Using the TIMER0 - Delays

The **TIMER0** lends itself, of course, to creating elements of timing and delays, instead of the waste time routines that we have seen in the previous exercises.
We can write a 10ms tempo routine based not on instruction loops, but on the action of the timer:

```
; TIMER0 Fosc/4 input, 1:64 prescaler
        movlw   b'11010101'
        OPTION
; Wait 10ms
        movlw   .100        ; Precharge Counter
        movwf   TMR0
t01     movf    TMR0,w      ; Overflow Wait
;       btfss   STATUS,Z    ; if 0, skip
        skz                 ; MPASM's pseudo opcode
        Goto    t01
timeout .....
```

Note that there is no need for RAM memory locations, since the time loop no longer includes the decrement of counters, but is performed entirely by the timer.

To be precise, clocked at 4MHz, you get a time of 9.984 ms. The pre-charge value of the counter determines, together with the pre-divisor, the time obtained: by loading 100 into `the TMR0`, the overflow will occur after 256-100=156us. And, given the prescaler, 156*64=9984us.

With a general formulation, you can manually calculate the value to be attributed to the register depending on the desired time. Here's the equation to determine the period:

$$\textit{period} = (256 - TMR0) * (4 / \textit{fosc}) * (\textit{Prescaler})$$

The **TIMER0**, as we have said, as soon as the microcontroller is powered and the clock is active, counts to grow and this happens starting from the value contained in the `TMR0` register. At ROP this value is random. If we want to get a defined time, for example a period of 1 ms clocked at 4MHz, we can use the 1:4 prescaler

$$1ms = (256 - TMR0) * (4/4MHz) * (4) -> TMR0 = 6$$

This is the value that you need to load into the `TMR0`. If you want a timing of 15ms, you can use a 1:64 prescaler

$$1ms = (256 - TMR0) * (4/4MHz) * (64) -> TMR0 = 100$$

However, rather than doing manual calculations, it is much more practical to use one of the many "***Timer0 calculators***" available on the net. For example,:

- PIC timer calculator
- Timer0 calculator
- Timer calculator by Mikroelektronika

This choice is dictated by the fact that it is much better to focus on the program rather than committing energy and time to make calculations that an applet can do better than us.

We can revise the programs of the various tutorials, without changing anything in the hardware, but using time routines based on **Timer0**. For example, to have 10ms, we can use what is written above. Based on the above, this should be without difficulty.

Alternative

Instead, we can consider several methods to use the Timer:

1. What we have seen, that is to preload the count register with a certain value and wait for the overflow, which will occur after a number of clock pulses equal to:

Pulse overflow = (256 - valore_di_ pre-charge) * prescaler

and the time generated is:

period = (256 - alore_di_ pre-charge) * (4 / fosc) * (Prescaler)

With this method, as we saw in the initial example, we wait for the value 00 that identifies the overflow of the counter.

2. A second method is to reset the **`TMR0`** register and wait for the desired number of pulses:

```
; TIMER0 Fosc/4 input, 1:64 prescaler
        movlw   b'11010101'
        OPTION
; Wait 10ms
        CLRF    TMR0
t01     Movf    TMR0,w      ; Expected
;       Xorlw   value
        BTFSC   .156
        Goto    STATUS,Z
Time-   .....
```

In this case, you can use the opcode of the **exclusive OR** to compare the value contained in the register with the desired value. In this case, the overflow happens after:

Overflow pulses = valore_di_ comparison * prescaler

and the time generated is

period = valore_di_ comparison * (4 / fosc) * (Prescaler)

There is no particular difference between the two methods and the choice is personal.

3. Again, since the timer is not stoppable, but continues to count as long as the clock is present, it is possible to use it as **free running**: you set a suitable pre-scaler and simply wait for the counter to go to zero. In this case, the time generated is:

$$period = 256 * (4 / fosc) * (Prescaler)$$

```
; TIMER0 for 1ms cadence (1024us precisely)
; Fosc 4MHz, 1:4 prescaler
          movlw    b'11010001'
          OPTION
; Hold 1ms
          clrf     TMR0         ; initialize Timer0
          goto     $+1          ;
          goto     $+1
          goto     $+1
t01       movf     TMR0,w       ; Overflow Wait
          skpz                  ; = 0, skip
           Goto    t01          ; not 0, waiting
; Timeout: There are 1024US before the next
Time-     .....                 ; Execute Timeout
          .....                 Statements
          .....
Loop      Goto     t01          ; Re-entry to the main loop before 1ms
```

4. or, again, using free running, the content of the `TMR0` is tested for a certain value. Individual bits can also be tested, as the content of the binary counter varies cyclically depending on the clock and prescaler.
   For example, we need to handle an action that takes less than 1ms. Also for a 4MHz clock, if you set a 1:128 divisor, a 1-bit increment of the counter will occur every 128us. Then, bit 2 will change state every 512us:

```
; TIMER0 for 1ms cadence (1024us precisely)
; Fosc 4MHz, prescaler 1:128
          movlw    b'11010101'
          OPTION
; Initial Sync Wait CLRF    TMR0
                                ; initialize Timer0
t01       BTFSC    TMR0,3       ; Wait bit 3=1
           Goto    t01          ; Not 1, Hold
; Yes - Start Management
time-out.....                   ; Execute Management Instructions
          .....                 ; for a <1ms time
          .....
; If handling takes less than 512us, wait for the next 0 bit T02  BTFSS
          TMR0,3                ; Wait bit 3=0
           Goto    T02          ; not 0, waiting
Loop      Goto     t01          ; Return to the main loop
```

All systems are valid, whether you use the internal clock, which is equal to that of the instruction cycle, i.e. *Fosc/4*, or whether you use an external clock that comes from the **`T0CKI pin`**.

In this case, you can use a particular frequency, for example 32768kHz to make RTCC or other timebases.

We'll look at applications of these principles in the following tutorials.

Advantages of using the Timer

We observe that, using a waste time waiting routine, such as those seen so far in the tutorials, the processor is forced to decrement registers until they are zero; no **other operations can be performed during this operation**.
If, on the other hand, we use the timer for counting, it operates independently of the processor, which thus remains free to perform other tasks.

This allows us, for example, to create time cadences on which to synchronize the various operations that the program has to perform.

A second consideration is that the same time can be obtained with different prescalers, while changing the pre-charge value of the timer.
For example, to get 16ms:

Period	Divisor	Pre Charge
16ms	1:64	6
	1:128	131

In general, the choice is equivalent, but you have to consider that a larger prescaler is more appropriate when you can't strictly wait for the overflow, but you need to do other things while waiting.
Let's explain: if the prescaler is 1:1, each instruction-cycle the timer advances by one unit. Condition 0 will only be valid for 1us. If I have inserted a pre-divider, e.g. 64, the **`TMR0 register`** increments by one every 64us and remains in the overflow condition for the same time. Then I can abandon polling and execute other instructions for less than 64us, returning to the timer test for the end of the count with the certainty that I will still intercept the overflow situation. The larger the prescaler, the greater the number of instructions that can be executed while waiting. With a prescaler at 256 I can execute instructions for 256us, i.e., at least 120, considering an equal amount of opcodes from 1 and 2us; Which is no small thing.

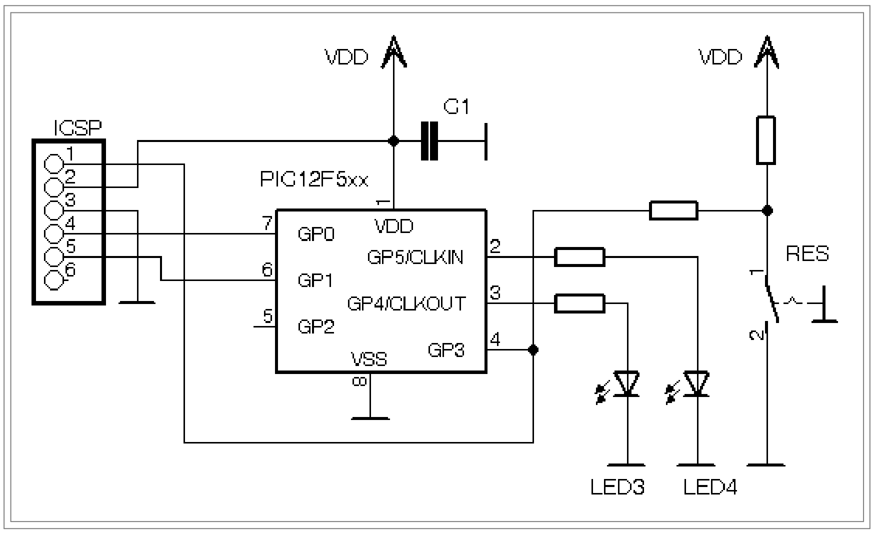
Therefore, it is possible to delegate to the timer the creation of a timing used for one process (task), while another process can be carried out, since the microcontroller is not engaged in the time loop except for the periodic verification of the overflow. This is not a true multitask, since, in the absence of an interrupt, the various tasks cannot be completely asynchronous (i.e., they can be executed on demand with unpredictable times), but can only be concatenated in an execution loop. However, the situation allows for a "simulation" of multi-tasking, which is more than sufficient in many cases. On the other hand, we have seen that Baselines are the simplest ICPs, the which results in very low costs, but also some limitations.

Using the Timer0 – time cadences

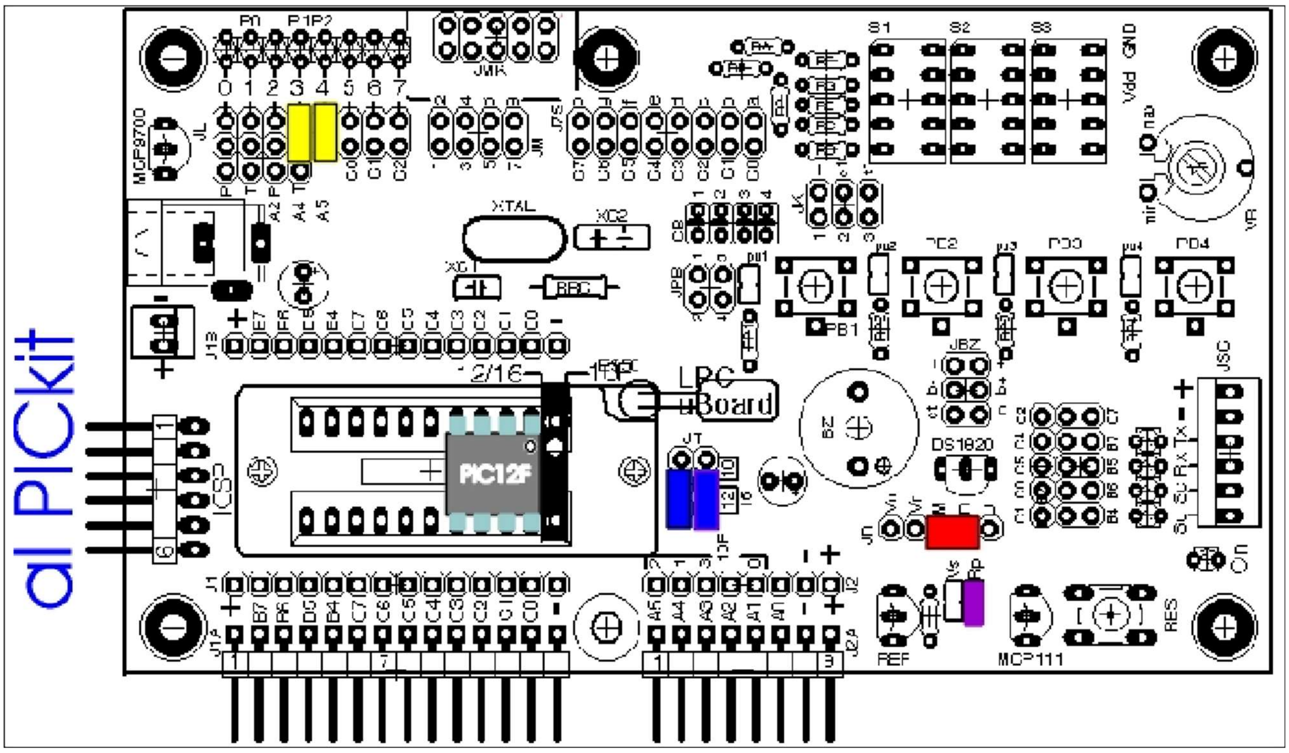
We can check out a simple example of two tasks, consecutively linked in the same time loop.

We want to flash one LED and control another at the touch of a button.

We use the 12F5xx :



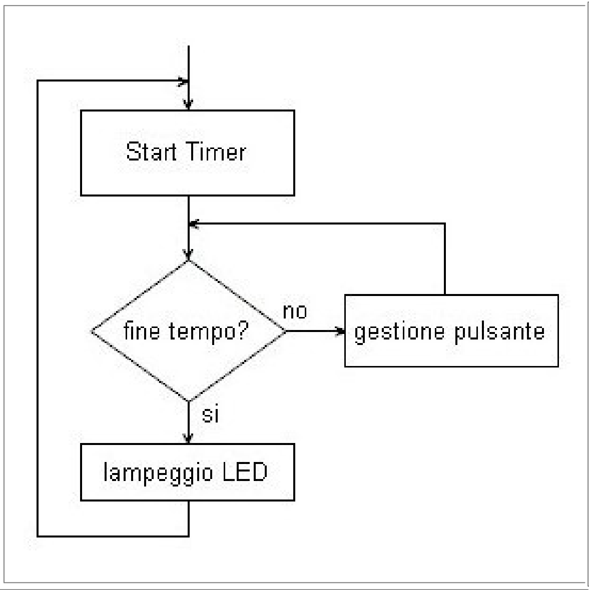
GP4 is the flashing LED and **GP5** for the LED controlled by the RESET button which is used as the input device of the **GP3 pin**.
The provision on the LPCuB:



The "yellow" jumpers connect the LEDs.
The "red" jumper connects the RES button to GP3. The "purple" jumper inserts the pull-up onto the button.

The program

The LED flashing cycle is a constant loop, while the button press can take place at any time.



We then define the flashing as the main task, the one regulated by the timer, and manage the second action in the waiting interval of the main timing.

It is necessary to study a loop that integrates the two actions and determine a timing that is adequate for both.

In particular, it is necessary that the main loop has a time that allows the eye to distinguish the flashing of the LED (and this implies times of hundreds of ms), while it must be considered that the analysis of the state of the button must be completed by a debounce action (which requires times of tens of ms).

We can start looking at the two actions separately. We have already seen the flashing of the LED in various tutorials and now we adapt it to the use of the timer as a time element.

For a flash at the cadence of 0.5s, we can set a timer cycle of 20ms, which we use as a base. Pre load the timer with 100 and set a pre divisor to 1:128 you have:

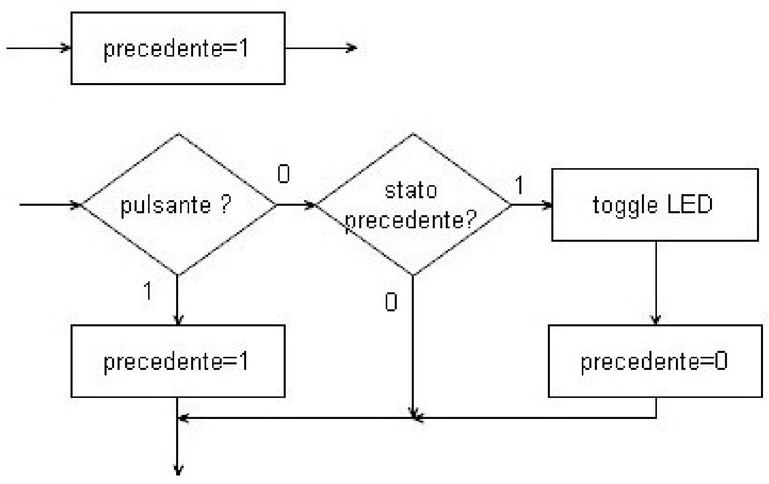
period = (256 - TMR0) * (4 / fosc) * (Prescaler) = (256-100) * 1 * 128 = 19.968 ms

```
; timer set Fosc/4 with prescaler
        movlw    b'11010110'
        OPTION
        movlw    .100       ; Pre Charge Timer
        movwf    TMR0
```

Counting 25 times the 20ms cycles will result in the required 500ms.

```
; LED flashing 1/2s
 Ledloop Movlw    .25        ; Number of
         movwf    stepcnt    repetitions
dllp     movlw    .100       ; in the counter
         movwf    TMR0
; Waiting for the
dllp1    movf     TMR0,W     ; timer = 0 ?
         skz                 ; Yes - Jump
          goto    dllp1      ; No -
         decfsz   stepcnt,f  ; counter -1 = 0?
          goto    dllp       ; No - Other Loop
; LED toggle using the shadow
         movf     sGPIO,w
         xorlw    LEDblink
         movwf    sGPIO
         movwf    GPIO
; another cycle by recharging the
         counter Goto LedLoop
```

For button testing and LED control in on/off mode (*toggle*):



We use a flag that indicates the previous state of the button and that is initialized to 1 (button not pressed).

The button test verifies the current status:

- if the previous one is the same as the current one, no operations are carried out on the LED
- If the previous state was not pressed and is now detected as pressed, the LED status is switched
- if the button is not pressed, no operation is performed on the LED

Basically, we detect the GP3 state change from 1 to 0 and only on this condition do we switch the LED.

In Instructions:

```
; Button test for LEDbtn control
btnchck BTFSS     Button          ; open button ? Goto
                  BCA             ; No - Checking
         Flags
        BTFSC     lastflg         ; Yes - Was it closed
         before? Goto             Blink    ; No - Exit
; Yes - Open-Closed Transition
        Bsf       lastflg         ; Update flag=open

BCA     BTFSS     LastFLG         ; Previous state = open?
         GOTO     Blink           ; No - Exit
Bac     Bcf       lastflg         ; Update flag=Closed
; Yes - Open-Closed Transition

tglbtn  movlw     LEDbtn          ; toggle LEDbtn
        xorwf     sGPIO,w
        movwf     sGPIO
        movwf     GPIO
        movwf
        movwf
```

Since the timer, with the pre scaler engaged, requires 128us for each update and, once it reaches 00, it remains for 128us before moving to 01, we can easily make a mix of the two programs, as we can see in the source ***8A_12F5xx.asm***, fillable for 12F508/509/510/519.
Apparently, when executed, these are two separate tasks, but, in fact, they are part of a single loop and are sequential. This is the solution applicable to small PICs that do not have interrupt handling.

Note that there is no need for a debounce: the button analysis is carried out at each overflow of the timer, i.e. every 20ms, enough time to turn off the bounces of a component of decent quality.

We need a memory location for counting **`stepcnt`** cycles and I/O shadow, as well as symbolizing the various constants:

```
                        DEFINITION OF PORT USE
#define     Button    GPIO,GP3 ; Button Input
;
; Shadow definitions to avoid RMW
#define     lastflg sGPIO,7
LEDblnk     = 0x10      ; Command number of Flashing LED
                                    the

LEDbtn      = 0x20        ; GP5 Command Bit of the LED by push
                                         button
LEDblink    = 0x10        ; GP4 Command Bit of the flashing LED
```

The use of labels instead of absolute values makes it possible to change parameters with a single action.

For example, if you want to swap the function of the LEDs and the flashing time to 250ms, you just need to act on the definitions, without changing anything in the rest of the source and in the hardware connections:

```
LEDbtn     = 0x10   ; GP4 bit LED control by push button
LEDblink   = 0x20   ; GP5 Bit LED Control Flashing

; CONSTANTS
T0preload = .61     ; preload value for TMR0 Step=
.10         ; number of repetitions
```

Before the loop starts, you need to have defined a bit for the button's status flag and initialized it to 1 (button not pressed). In order not to declare a RAM location that would be used for only one bit, we can use the 7th bit of the shadow **sGPIO**, whose state has no reflection on the I/O: the shadow is derived from an 8-bit register, but only 6 are used at the bits connected to the I/O pins, and the two highest bits (bits7-6) have no effect, since the corresponding bits in **GPIO** The following are not implemented:

```
;*********************************************************************
; shadow sGPIO
;| 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
;|----|----|----|----|----|----|----| ---|
;|flag|    |sGP5|sGP4|sGP3|sGP2|sGP1|sGP0|
;
#define  lastflg  sGPIO,7    ; Button Status Flag
```

The accuracy of the timing

It should be noted that in these exercises we did not look for absolute precision in timing, as it is not necessary and in any case sometimes not possible or extremely laborious to obtain, especially in Baselines that do not have interrupts.
In fact, two factors oppose absolute accuracy: the characteristics of the timer, with respect to the clock frequency; the need to add statements outside of loops for decisions and the aforementioned lack of interrupts and their flags.

The first factor depends on the fact that the counting takes place on 8 bits and with prescalers that are multiples of 2, which makes counting affected by a certain tolerance where values that are not precise multiples are required; For example, 25ms is approximated to 24.96ms or up to 25.088 (using 62 as the pre-charge value), while 16.00ms can be achieved with precision, using a pre-charge of 6 and a prescaler factor of 1:64.

Then, in general, the higher the value of the pre-division, the less accurate the times obtained, especially if we interlace other instructions as in the example just seen. This can be remedied, where necessary, by changing the clock frequency with different values: for example, 25.00ms is achievable with a clock of 4.096000MHz.

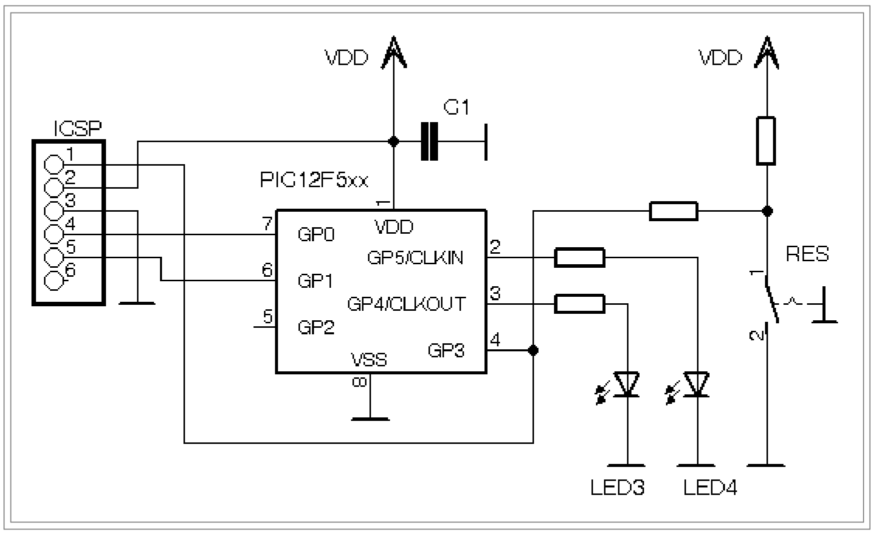
It should be clear that, by using crystals of well-defined values, it will be possible to obtain specific times and frequencies, such as those of baud rate for serial communications (1.8432, 3.6864, 4.9152 MHz, etc.) or for Real Time Clock (32768 Hz) or 3.595295 and 14.31818MHz for NTSC TV transmissions, 25.000MHz for Ethernet, etc. (see also a list of values and uses at http://en.wikipedia.org/wiki/Crystal_oscillator_frequencies).

The second point, i.e. the need to add instructions for decisions or jumps and the like, can be adjusted with a calculation of the duration of the same, which will be subtracted from the time produced by the loop. Thus, in order to produce precision times for RTCC (*Real Time Clock Calendar*) it is necessary, with Baselines, to work on the software and the use of crystals of adequate frequency.
In any case, it is rarely necessary to timings of absolute precision, which, however, are dependent on the accuracy of the clock generator, which must be of the same order.
For most applications, a certain tolerance is allowed: for example, a +/-1% baud rate is accepted. In LED flashing, of course, the acceptable tolerances are much greater.

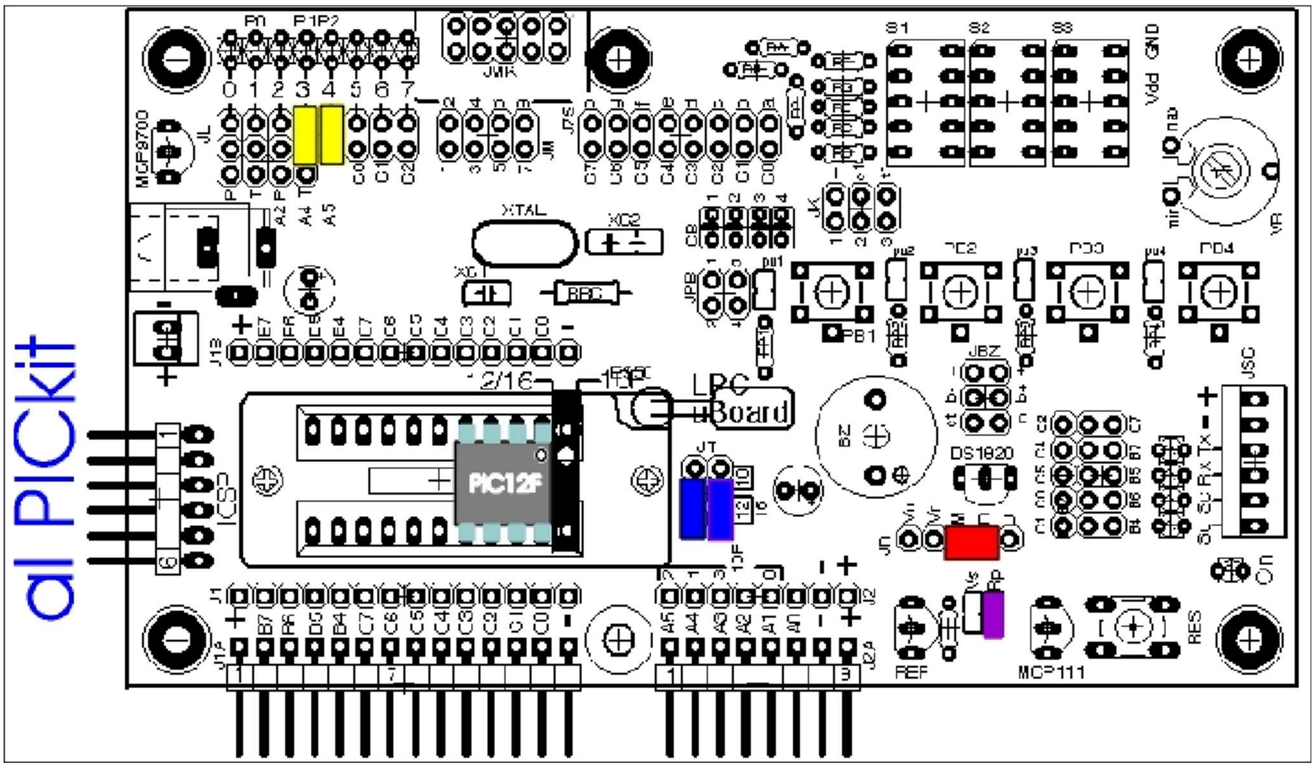
These considerations, however, do not free you from the need to have time algorithms that are as accurate as possible, so that you don't have to work hard when they become necessary.

A variation

Let's see a variation, using a PIC10F20x.
The scheme is similar to the previous one, with the limitation of the smallest number of pins:



And on the LPCuB:



Let's use the timer in a different way: a 4MHz clock, we know, generates an instruction cycle of 1us. We can insert a 1:256 prescaler.
Under these conditions, the TMR0 counter bits advance according to their position in the binary number:

Prescaler pulses 1:256	bit			
	3	2	1	0
0	0	0	0	0
256	0	0	0	1
512	0	0	1	0
768	0	0	1	1
1024	0	1	0	0
1280	0	1	0	1
1536	0	1	1	0
1792	0	1	1	1
2048	1	0	0	0

Bit 0 will go to level 1 with a period of 256us. Bit 1 will go to level 1 with a period of 512us. Bit 2 will go to level 1 with a period of 1024us. Bit 3 will go to level 1 with a period of 2048us. Etc.
This means that, by letting the timer run freely (free run), I can have a time reference simply by checking the status of the counter bits.
Bit 5 will go to level 1 with a period of 8192us, which is about 8ms, which is adequate for the previous application.
We can define some constants:

```
; ####################################################################
;                                CONSTANTS
stepnum    = .31   ; Number of repetitions for 8192*31=253ms

; bit of TMR0
#define bit4096 TMR0.4     ; bit 4 -> 16*256=4096us
#define bit8192 TMR0.5     ; bit 5 -> 32*256=8192us
#define bit16384 TMR0.6    ; bit 6 -> 64*256=16384us
#define bit32768 TMR0.7    ; bit 7 -> 128*256=32768us
```

Next, let's look at the chosen time bit:

```
dllp     BTFSS     bit8192        ; Timer Bit = 1 ?
          Goto     dllp
```

In this way, the program advances each time the indicated bit goes to 1, i.e. 8192us. The core of the program is identical to the previous one.
Checking for flashing requires you to wait for the end of level 1 of the counter bit before re-entering the loop:

```
Waitlp   BTFSC     bit8192        ; Waiting for the end of time
          Goto     Waitlp
         Goto      dllp           ; New Loop
```

This causes the LED to have a period of about 500ms. The line:

```
        CLRF     TMR0           ; initialize Timer0 counter
```

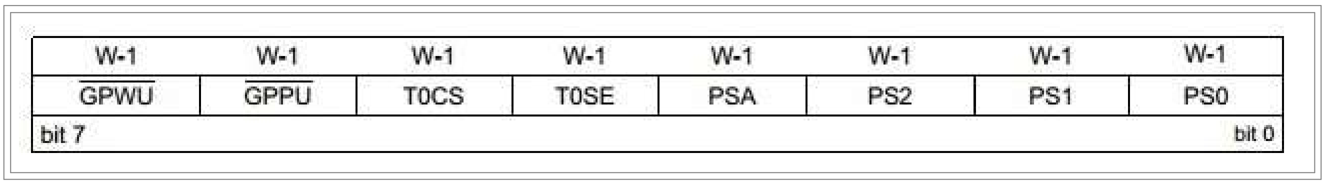
can be omitted: simply, the first time loop will start with the random value present at that moment in **`TMR0`**.
The source in the ***8A_10F20x.asm file*** is compileable for **10F200/202/204/206**. For the latter two, the comparator is disabled in order to eventually access **`GP2`**.

T0CKI

By exploiting the external **`T0CKI`** input, we can input a frequency or pulses coming from the outside into the microcontroller.

In the first case, we can use the external frequency as a clock to generate times, with the precision of the same frequency. For example, we could use a 32768kHz crystal oscillator to make a clock or similar. Another application could be the evaluation of the period of an impulse coming from the outside, for comparison with the internal clock. Unfortunately, since the Baselines do not have interrupt handling, the possibilities in this sense are not at their maximum, but it is still possible to create interesting applications, even without resorting to PICs of higher families.

In this sense, the **`OPTION register`** allows you to set a T0CS parameter that determines whether the count takes place on the rising or falling edge of the input signal, allowing precise synchronism.



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

The affected bits are:

- bit 5 **`T0CS`** : its function is to choose which clock to send to the timer input. We see it schematized in the functional diagram.
  - **`T0CS`** = 1 input is **`T0CKI`** **(default to POR)**
  - **`T0CS`** = 0 input is the internal clock ***Fosc/4***

- bit 4 **`T0SE`** : Selects the edge of the clock signal on which the timer counts when it comes from the input. The function is schematized with an exclusive OR gate.
  - **`T0SE`** = 1 falling edge **(default to POR)**
  - **`T0SE`** = 0 rising edge

Therefore, in order to use an external signal, on the falling edge, after the POR you will not need any change in the register , which sets this configuration as the default.
If, on the other hand, we want to synchronize the rising edge:

```
        movlw   b'11101111'   ; T0CKI, rising, no prescaler
    OPTION
```

The prescaler handling is the same as seen above.

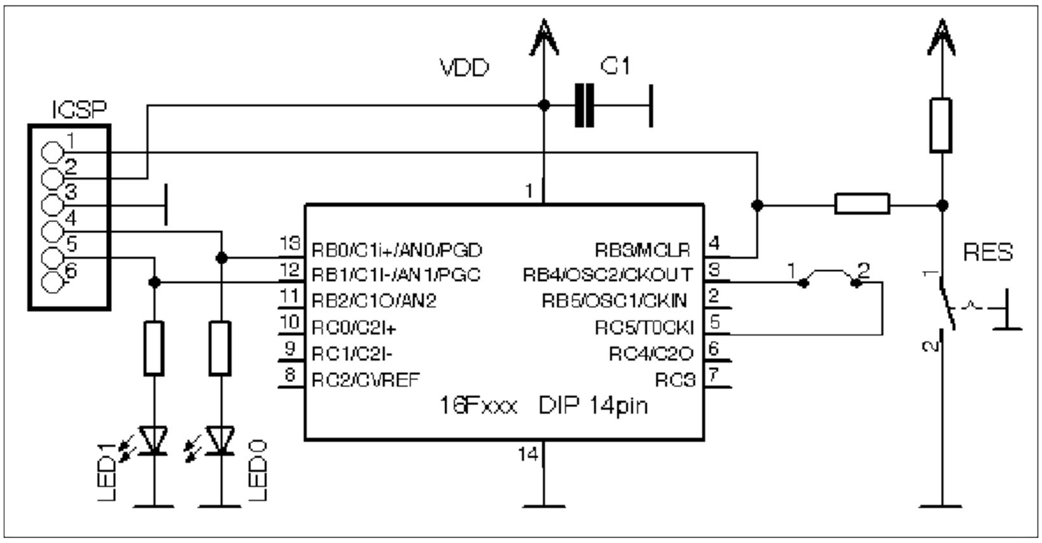
If you have a signal generator that provides a square wave, you can apply it to the **`T0CKI`** pin and test any of the above programs with the new clock, adjusting the tempo routines for the frequency entered.

PIC	T0CKI	FOSC/4
10F2xx DIP 8pin	GP2	GP2
12F5xx DIP 8pin	GP2	-
16F5xx DIP 14pin	RC5	RB4

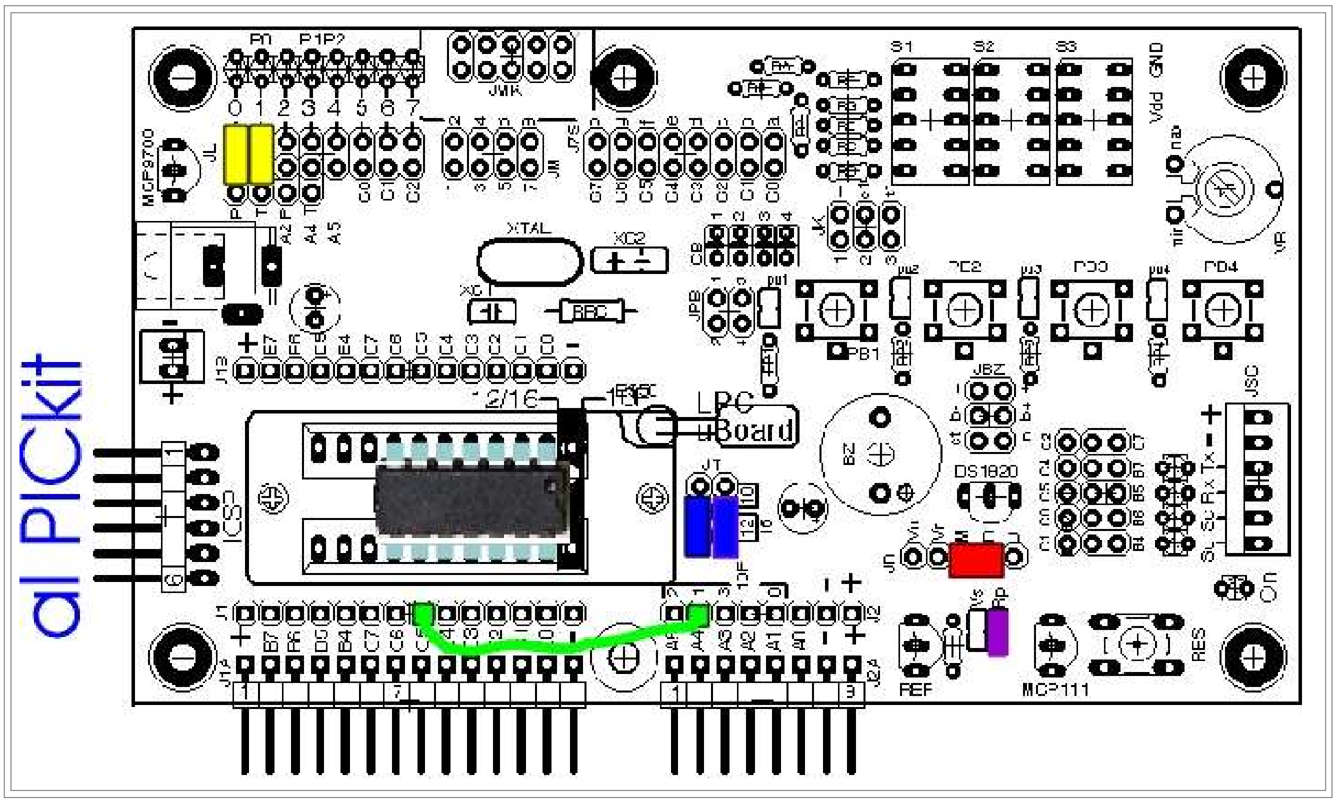
It should be noted that it must be a square wave signal at TTL level, i.e. between 0 and, at most, the Vdd power supply of the microcontroller and not a bipolar signal (symmetrical with respect to 0) such as those used for audio amplifier tests.
For those who do it themselves, we can recommend making any oscillator with CMOS or TTL gate, or a 555, on a breadboard, powering it directly from the same voltage as the microcontroller.

With the LPCuB you can simply test the function with this circuit by connecting the Fosc*/4* internal clock output with the **`T0CKI`** input. As you can see from the table above:

- for 10F there is a ***Fosc/4*** output, but this is shared on **`GP2`** with **`T0CKI,`** so only one of the two can be used.
- for the 12F5xx there is no ***Fosc/4 output***.
- We can, instead, use a 16F5xx that has both **`T0CKI,`** which can be enabled on **`RC5`** and ***Fosc/4*** can be enabled on RB4.



From the point of view of connections on the LPCuB:



The "yellow" jumpers connect the LEDs.
The "green" flying jumper connects T0CKI (RC5) and Fosc/4 (RB4).
The RESET button is not used, so it doesn't matter whether to connect it (as in the figure) or not.

The Program

With this hardware configuration, we flash the two LEDs, one timed by a waste time loop, the other by Timer0.

Notice that the assignment of the clock output function on the **`CLOCKOUT pin`** is determined in the initial config:

```
    _IntRC_OSC_CLKOUTEN
```

while the enabling of T0CKI is controlled by the OPTION register, bit 5 T0CS, on which, however, we do not have to perform any action because the default to the POR provides that T0CKI is enabled.
It should be noted that, once again, the lack of uniformity of the labels of the ***processorname.inc*** files makes the configuration phase unnecessarily laborious for processors with very similar characteristics:

Option	16F526	16F506	16F505
Watchdog	`WDTE_OFF`	`WDT_OFF`	
Dual clocked for 4MHz	`_IOSCFS_4MHz`	`_IOSCFS_OFF`	-
Clock with Fosc/4	`_IntRC_OSC_CLKOUT`	`_IntRC_OSC_CLKOUTEN`	

The flashing loops of the two LEDs are chained together: while waiting for the timer counter to overflow, an instruction loop is executed to wait for the other LED. Let's look at the block diagram:

```
    _IntRC_OSC_CLKOUTEN
```

The waste time loop takes about 8ms, while the test bit of the TMR0 is tested on bit 7 which corresponds to a time of 32,768ms.

The result is that the two LEDs flash at different intervals. When we remove the flying jumper that connects CLOCKOUT with T0CKI, LED1 continues to flash, while LED0 locks. This is due to the fact that, by suspending the clock at the timer, its counter does not advance and consequently the check for the bit is never checked and the time loop for the flashing of LED0 is never completed. By repositioning the jumper, LED0 will also start flashing again.

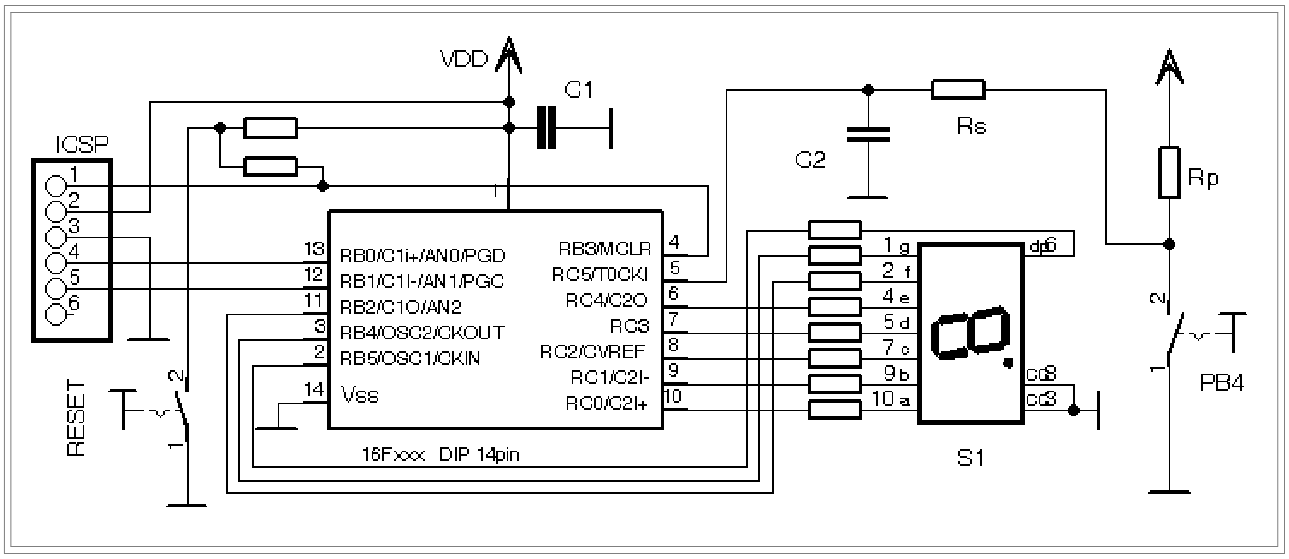
The Timer0 as a counter

There is nothing to prevent the timer from being used as a register for counting individual pulses that come from the outside, thus realizing functions such as piece counting, people counting, etc.

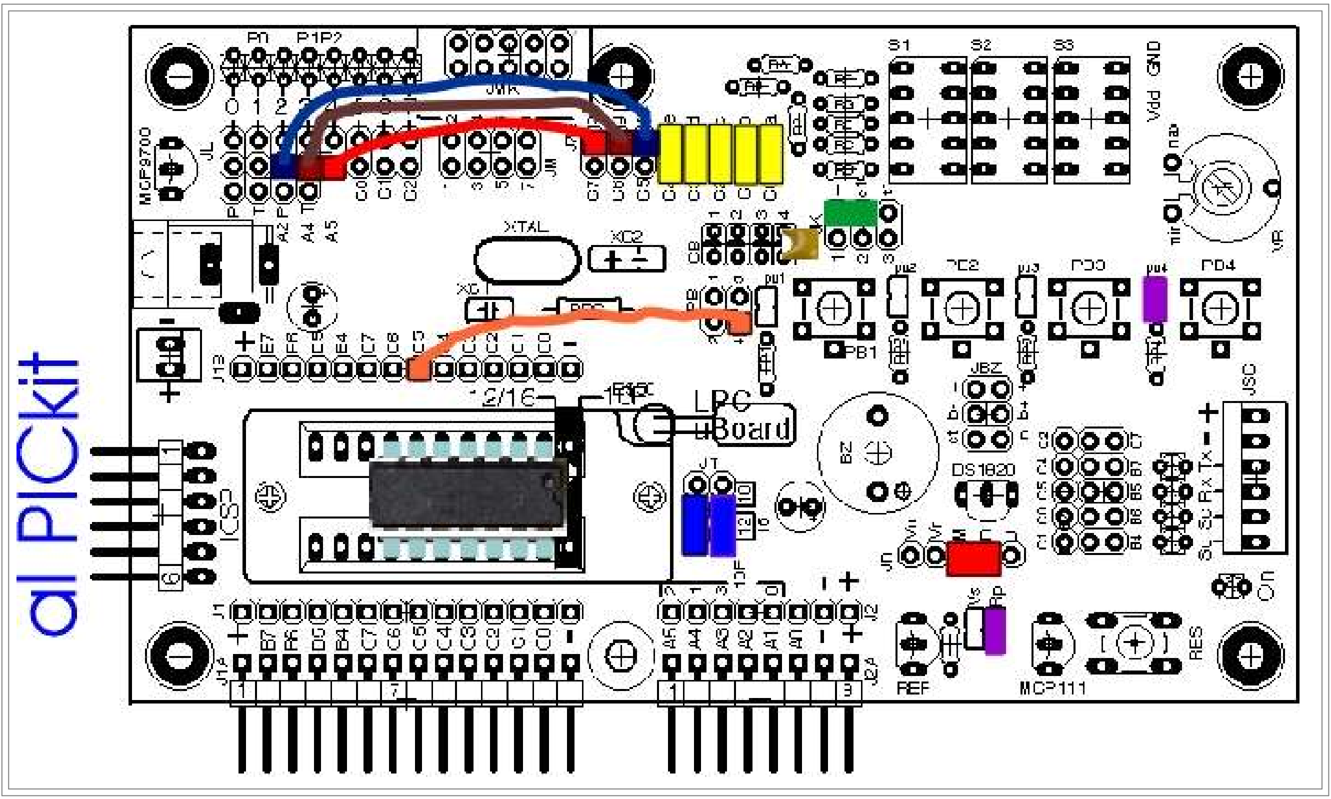
It goes without saying that the input pulse must be free of disturbances and bounces; therefore it will not be possible to use a mechanical contact, unless it is equipped with its own debounce circuit. On the other hand, pulses coming from electronic circuits, optical sensors, reed contacts without bounces, inductive and magnetic proximity sensors, etc., will be suitable.

It should be clearly noted the difference between what we have previously seen on the subject of debounce: if we enter a signal plagued by bounces in any I/O, we can introduce a debounce action through the program. In the case of pulses sent to the timer input, this is NOT possible, as this pin, in its function of **`T0CKI,`** is not a digital I/O that can be managed by the program, but is the direct counting input. Therefore, we cannot apply a program anti-rebound system, but the impulse must be "clean" in itself; otherwise, each bounce of sufficient amplitude and duration will advance the counter by one, making the count incorrect.

That said, we can make the hardware:



We use the **16F526** which controls the 7-segment display; this will indicate the number of pulses received by the `T0CKI` input. To obtain the pulses, we use one button on the **LPCuB** board, to which we add a C2 capacitor, thus realizing a very simple hardware debounce.



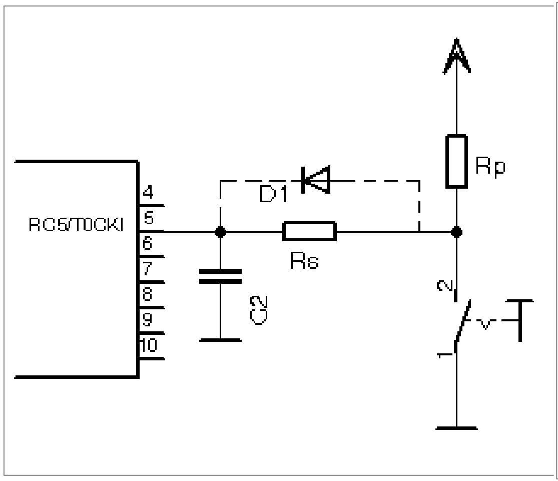
It involves installing:

- the "yellow" jumpers that connect the segments from ***a*** to ***and*** respectively with pins `C0` to `C4` and the three wires to connect the ***g segment*** with **RB4** (the "brown" cable), the ***dp segment*** with **RB5** ("red" cable) and the ***`f segment`*** with **RB2** ("blue" cable).
- In addition, the **`RC5 pin`** (**`T0CKI`**) must be connected with the ***PB4*** button ("orange" cable), inserting the ***capacitor*** and the "purple" jumper for the relative pull-up.

- The "green" jumper, as seen in ex. 7, connects the cathode of the S1 display to the Vss.

As usual, the arrangement of the chip on the socket, the position of the jumpers are to be observed ***JT*** and those related to the RESET button.

Regarding how the hardware debouncer works:



C2 is loaded through ***Rp*** and ***Rs***. When we press the button, ***C2*** is downloaded through ***Rs***. By releasing the button, ***C2*** recharges through ***Rp*** and ***Rs***, taking a time dependent on the RC network to get to the Vdd voltage. If this time is longer than the time of the bounces, they are neutralized.

It is not possible to use long times, as ***Rp***+***Rs*** must be a valid pull-up value for the **T0CKI** input, while ***C2*** cannot take too high a value that would cause inadequate slope fronts.

In fact, the point at which the **T0CKI** input detects the change in the logic state, and therefore accepts the pulse, is equal to about 85% of the Vdd and should be adjusted as a slope through a Schmitt trigger. Fortunately, the entrance has just this type of gate, which allows the whole thing to function with sufficient security.

As for the values, we can use 47k for ***Rp*** and 1K for ***Rs***, while ***C2*** will be the usual ceramic 100nf; diode ***D1*** is optional: if it is added, the function is as follows:

- in open contact the capacitor is charged through ***Rp*** and ***D1***, practically excluding Rs.
- at closed contact, the capacitor discharges through ***Rs***

In this way it is possible to use a higher ***value Rs***, since it is not part of the pull-up and thus increase the time constant for the discharge.

Excluding the diode, the values indicated result in a debounce time that is usually sufficient to absorb the bounces of the buttons used on the LPCuB.

The program

The operation of the system is simple: the timer is used for a counting input from the **T0CKI** and without a prescaler; since this is the default configuration at the POR, no action is required on the **OPTION register**.
Cyclically, the program detects the value contained in **TMR0** and transfers it to the display:

```
DLP movlw   Time          ; 200ms cycle
    Call    Delayms
    movf    TMR0, w
    Call    Display       ; Copy to the
                          display
```

```
      Goto     Dlp            ; New Cycle
```

Where is a constant such that you get 200ms of delay.
This time is not critical in the slightest and only limits the refresh rate of the display. Be careful, therefore, that this has nothing to do with a button debounce, which is delegated exclusively to the hardware.

Therefore, the display increases by one with each press of the button: once it reaches 9, the display advances with the hexadecimal digits from A to F. On the next step, it resets, while the decimal point lights up to indicate display overflow. The transfer from the **TMR0** to the display is complicated by the fact that the segments are controlled partly by **PORTB** and partly by **PORTC**. To do this, we use two lookup tables, one for segments a to ***e*** and one for ***f*** and ***g***:

```
       movlw  0xF          ; greater than Fh ?
       subwf  TMR0,w       ;
     ; btfsc  STATUS,C     ; Yes - Jump
       skpnc               ; MPASM's pseudo opcode
        goto  clearT       ; No - Display

MOVF Display    TMR0,w     ; TMR0 value at the
          movwf  savew
          call   segtblc
          movwf  PORTC
          movf   savew,w
          call   segmtblb
          movwf  PORTB
```

The first table matches the input value with the PORTC-driven segments, while the second is for the PORTB-driven segments.
The output value from the tables is passed directly to the ports, with the usual principle that writing the non-implemented bits has no effect and that of the bits not set as outputs is not effective (remember that you only write to the latch of the port, which is not transferred to the output pin since the bit is configured as input).

We don't use shadows, but we do it knowing full well that in this case they are not necessary, since we write the entire contents of the port in a single operation, while the decimal point bit is updated individually.
In any other case where it is not absolutely certain that it is the result of direct writing, the use of shadows is mandatory.

The verification of the exceeding of the limit value of the counter is carried out by subtraction, using a macro:

```
; if count exceeds Fh, turn on dp CFLSA
                  Savew, 0x0F
         retlw 0
        Bsf      SEGDP
        retlw    0
```

where **savew** is the temporary location where the **TMR0** counter that was passed to **WREG.** The decimal point is lit if the count exceeds 16 (Fh).

With MCLR **enabled**, pressing the **RESET** button restarts the program, resetting the count.

If the counter advances several steps when the button is pressed, this indicates the presence of bounces: the solution will be to use a capacitor of a higher value (220nF-1uF, always multilayer ceramic).

The input of the timer/counter could also be connected to a different sensor, such as an inductive proximity detector or Hall sensor or any other source of pulses without bounces.

The ***source 8A_T0CNTR.asm*** and the related project can be found in the attached files.

The **CFLSA** macro is part of a set of other macros created to easily replace multiple lines of statement in comparison operations. Since these are very common in every program and the use of addition or subtraction operations may not be immediately comprehensible, here are the macros that can be a valuable help.
They are included in the source before their use, referencing a ***compmacro.asm library*** which contains several. In particular, the one we use here implies the following instructions:

```
;*******************************************************************
; CFLSA Compare File to Literal and Skip if Above
; Compare the contents of a log with a number and skip
; If > literal file
CFLSA   MACRO    file,lit
        movlw     (255-lit)  ; W = (-lit)+1
        addwf     file,W     ; W = file - W = file - (255-lit)
        skpc                 ; C=0 if file>lit, skip next line
        ENDM                 ; C=1 if file<=lit, run next line
```

The macro uses the **addwf opcode** with the complemented literal **(255-lit),** which is equivalent to using the subtraction opcode, since :

a-b = a +(-b)

ADDWF

The addwf (ADD W *to file) opcode* has the following function: **it sums the value contained in W with the value contained in the file in question**

Symbolically: **f + W -> d**

where **d** is the destination; in fact, it is a "tail" statement, so the result of the subtraction can be sent to both W and the file.

[label]	sp	addwf	,f/w	sp	Object	sp	[Comment]

An essential factor of education is that the result is reflected in the STATUS C and Z flags.
The **C flag** is the Carry, or carryover, of the sum. If the sum result exceeds FFh, the carry will go to 1, otherwise it will be to 0. If the sum is 0, the Z flag will be set.

- if ***sum*** > ***FFh***: **Z = 0 C = 1**
- if ***0***< ***sum*** < = ***FFh*** **Z = 0 C = 0**
- if ***sum*** = ***0*** : **Z = 1** (**C = 0** if the result has no carryover)

In fact, the statement also modifies the **DC** flag, which we will overlook for now as it is only interesting in the area of conversions.

SUBWF

The subwf (Subtract W ***from** file) opcode* has the following function: **it subtracts the value contained in W from the value contained in the file in question.**

Symbolically: **f - W -> d**

where **d** is the destination; in fact, it is a "tail" statement, so the result of the subtraction can be sent to both W and the file.

[label]	sp	subwf	,f/w	sp	Object	sp	[Comment]

Since subtraction is not commutative, you need to be careful about what is subtracted from what is subtracted. **For PICs, it is the value contained in the W register that is subtracted from operand *f*.**

So, again, the result is ***W-file*** and not ***W-file***, as you might think.

An essential factor of education is that the result is reflected in the flags of the STATUS, and precisely in this way:

Condition	C flag	Z flag
`f > W`	1	0
`f = W`	1	1
`f < W`	0	0

As a result, it is possible to use the statement not only for pure arithmetic subtraction, but also to compare two values, since there are only 3 possible outcomes:

- if ***w*** <***f*** : **Z** = **0** **C** = **1**. destination = (**f-w**)
- if ***w*** =***f***: **Z** = **1** **C** = **1**. destination = **0**
- if ***f*** < ***w***: **Z** = **0** **C** = **0**. destination = (**f-w**) + **0x100**.

It becomes possible to combine these possibilities in pairs to obtain a comparison between the two values:

- if ***f*** <= ***w*** : **Z** = **C** , i.e. both are 1 or both are 0
- if ***w*** = ***f*** : **Z** = **0** , **C** undefined
- if ***w*** < = ***f*** : **Z** indefinite, **C** = **1**.

Usually the Assembly programmer tries to apply comparisons for ***w*** <= ***f*** or ***f*** < ***w***, because you only need to check the state of **C**, ignoring **Z**.
In fact, the statement also modifies the **DC** flag, which we will overlook for now as it is only interesting in the area of conversions.

Two notes on the subtraction statement in Baselines:

- In PICs there are other opcodes concerning subtraction, such as **`sublw`** and **`subwfb`**, but in the 12-bit instruction set of Baselines these are not present: here **`subwf`** is the only possible subtraction . This limits the programmer's possibilities a little.

- **Another feature of the subtraction statement in Baselines is that there is no intervention of the Carry**, which is only modified by the result, without becoming part of it. It follows that, if this is convenient because it is not required to delete/set the flag before the subtraction instruction, there is on the other hand the problem that, in the case of operations on numbers composed of several bytes, the **program will be responsible for handling the carryover**.

MPASM's pseudo opcodes

Let's also open a brief digression about the pseudo opcodes that we are using here and there. It's simply this: STATUS flag tests can be done with the following forms:

```
     BTFSS    STATUS flag ; Skip following statement if flag is 1

     BTFSC    STATUS flag ; Skip following statement if flag is 0
```

I
processors, mainly Motorola HC11, 8051, Parallax, etc. In these, there are statements or assembly forms that refer directly to the flags of the **`STATUS`**.
MPASM supports some of these forms.
It should be noted that **these are not opcodes, but macros with a mnemonic label** similar to those of the families of processors mentioned and which, during compilation, are replaced by the corresponding PIC opcode**.** These macros are part of the MPASM compiler's automatisms and may not be present in other compilers. Where you need to use them and they are not available, you can define the relative labels as macros, according to the following table. Here is the list of the main ones:

Mnemonic	Function	Equivalent code
CLRC	*Clear Carry*	`bcf STATUS,C`
CLRDC	*Clear Digit Carry*	`bcf STATUS,DC`
SETDC	*Set Digit Carry*	`bsf STATUS,DC`
CLRZ	*Clear Zero*	`bcf STATUS,Z`
SETZ	*Set Zero*	`bsf STATUS,Z`
SKPC	*Skip on Carry set*	`btfss STATUS,C`
SKPDC	*Skip on DC set*	`btfss STATUS,DC`
SKPNDC	*Skip on no DC set*	`btfsc STATUS,DC`
SKPZ	*Skip on Zero set*	`btfss STATUS,Z`
SKPNZ	*Skip on no Zero set*	`btfsc STATUS,Z`
B k	*Branch to destination*	`Goto k`
BC k	*Branch on Carry*	`btfsc STATUS,C` `goto k`
BNC k	*Branch on No Carry*	`btfss STATUS,C` `goto k`
BDC k	*Branch on Digit Carry*	`btfsc STATUS,DC` `goto k`
BNDC k	*Branch on No Digit Carry*	`btfss STATUS,DC` `goto k`
BZ k	*Branch on Zero*	`btfsc STATUS,Z` `goto k`
BNZ k	*Branch on Non Zero*	`btfss STATUS,Z` `goto k`

The use of these pseudo opcodes is limited to the fact that they are supported by the Assembler used, which is true for the **MPASM**, but it may not be for others.
Where possible to use them, it's a matter of personal style in the writing of the source. We prefer to use them, both because the Microchip Assembler supports them, and because in our opinion they simplify the reading of what is written.
As mentioned, if the compiler does not provide them, it is always possible to resort to writing Macros, for example:

```
BNZ    MACRO Destination
    BTFSS    STATUS, Z
      Goto     Destination
        ENDM
```

More [information on this topic here](#).

It should be noted that some pseudo opcodes are tied to the processor due to the different internal features. These include **banksel** and **pagesel** commands that select banks and pages and are compiled as statements that modify their flags.
Therefore, if you can also write macros like this:

```
Bank0 MACRO                       ; Go to counter
   bcf STATUS, RP0                0
      ENDM
Bank1 MACRO                       ; Go to Counter
   bsf STATUS, RP0                1
      ENDM
```

there is no point in adding them to a source compiled with MPASM, since the simplest writings will suffice:

```
   banksel 0
   Banksel 1
```

The same goes for macros that have the same function as the other pseudo statements.

Troubleshooting

If the counter advances several steps when the button is pressed, this indicates the presence of bounces, which the input of the timer/counter accepts as varizoins of the level of the T0CKI pin: the solution will be to use a capacitor of a higher value (220nF-1uF, always multilayer ceramic).
In practice, the timer input must in any case be a well-defined TTL signal; If you intend to use a mechanical contact, a conditioning will be interposed, for example with a retriggerable monostable.

Other input sensors ?

The input of the timer/counter can also be connected to a sensor other than the button, such as an inductive proximity detector or a Hall sensor or any other source of pulses without bounce.

It should be noted that such an application is only indicative of the operating possibilities of the timers/counters contained in the microcontrollers and is not usual in practice for the realization of counts

pieces or similar, since it is preferable to make the counting loop considering a digital I/O as input, on which a debounce algorithm can be applied as seen above.
On the other hand, the use of the timer as a counter is useful for measuring periods, pulses, frequency.

8A_12F5xx.asm

```
;**********************************************************************
; 8A_12F5xx.asm
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 8A
;                     An LED flashes while a button controls a
;                     other LEDs. Timing with TIMER0.
;                     .
;    PIC            : 12F508/509/510/519
;    Support        : MPASM
;    Version        : 1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;----------------------------------------------------------------------
;
; Pin use :
;  ----------------
;       12F508/509/510/519 @ 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 LED5
;    GP4/OSC2             3: Out LED4
;    GP3/! MCLR/VPP       4:
;    GP2/T0CKI            5:
;    GP1/ICSPCLK          6:
;    GP0/ICSPDAT          7:
;    Vss                  8: --
;
;**********************************************************************
;                        PROCESSOR DEFINITION
 #ifdef __12F519
        LIST p=12F519
        #include <p12F519.inc>
 #endif
 #ifdef __12F510
        LIST p=12F510
        #include <p12F510.inc>
#define procanalog
 #endif
 #ifdef __12F509
        LIST p=12F509
        #include <p12F509.inc>
#define procbase
 #endif
 #ifdef __12F508
```

```
        LIST p=12F508
        #include <p12F508.inc>
#define procbase
 #endif
        Radix     DEC

; ####################################################################
;===================================================================
;                          CONFIGURATION
;
; Internal oscillator, no WDT, no CP,
 pin4=GP3; #ifdef   12F519
 __config  _IntRC_OSC & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF & _CPDF_OFF &
_MCLRE_OFF
 #endif

 #ifdef___12F510
 __config  _IntRC_OSC & _IOSCFS_OFF & _WDTE_OFF & _CP_OFF & _MCLRE_OFF
 #endif

 #ifdef procbase
 __config  _IntRC_OSC & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif


; ####################################################################
;                        RAM
;
       UDATA_SHR            ; Data RAM
sGPIO   RES 1               ; shadow for GPIO
stepcnt res 1               ; Counter for flashing

; ####################################################################
;===================================================================
;                   DEFINITION OF PORT USE
;
;
; GPIO map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|LED4 | LED3| BTN |     |     |     |

#define    Button   GPIO,GP3 ; Button Input
;
; Shadow definitions to avoid RMW
; sGPIO map
;| 7 | 6 |       5 |    4 | 3 | 2 | 1 | 0 |
;|-----|-----|------|------|-----|-----|-----|----------|
;| flag|     |LEDbtn|LEDblk| BTN | sGP2|sGP1 |sGP0 |
;
;#define    sGPIO,GP0    ;
;#define    sGPIO,GP1    ;
;#define    sGPIO,GP2    ;
;#define    sGPIO,GP3    ;
;#define    sGPIO,GP4    ;
;#define    sGPIO,GP5    ;
;#define    sGPIO,GP6    ;
#define    lastflg sGPIO,7
```

```
LEDblnk     = 0x10   ; Flashing LED control number
```

```
LEDbtn      = 0x20    ; LED control number by push button

; ####################################################################
;                             CONSTANTS
;
T0preload  = .100   ; preload value for TMR0 for 19,968ms
stepnum    = .25    ; Number of repetitions for 500ms

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

; Calibrate Internal Oscillator
        MOMOVWF OSCCAL
        Pagesel Init
        Goto    Init

; Subroutines & Tables


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

Init:
; Reset Initializations
        CLRF    GPIO            ; GPIO preset latch to 0
        movlw   0x80            ; Preset Flag (Open Button)
        movwf   sGPIO           ; LEDs off

 #ifdef procanalog              ; only for 12F510
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the digital MOVLW function
                0x77
        movwf   CM1CON0
 #endif

; TRISGPIO        --001011       GP2-4-5 out
        movlw   b'11001011'
        Tris    GPIO

; set Timer0 Fosc/4 with prescaler 1:128
         ;      b'11010111'
         ;        1-------     GPWU Disabled
         ;        -1------     GPPU disabled
         ;        --0-----     Internal Clock
         ;        ---1----     Falling
         ;        ----0---     prescaler to Timer0
         ;        -----110     1:128
        movlw   b'11010110'
        OPTION

; LED flashing 1s
```

```
Mainloop:
; 500ms delay (25 x 20ms)
        movlw    stepnum       ; Number of
        repetitions MOWF       stepcnt     ; in the
        counter
dllp    movlw    T0preload     ; Pre Load Timer
        MOWF     TMR0          ; Waiting for the
        end of time
dllp1   movf     TMR0,W        ; timer = 0 ?
        skpz                   ; Yes - Jump
         Goto    dllp1         ; No - Waiting

; Button test for LEDbtn control
BTNCHCK BTFSS    Button        ; open button ? Goto
                 BCA           ; No - Checking
         Flags
        BTFSC    lastflg       ; Yes - Was it closed
         before? Goto          Blink ; No - Exit
; Yes - Open-Closed Transition
        Bsf      lastflg       ; Update flag=open
        goto     tglbtn        ; operate LEDbtn

BCA     BTFSS    lastflg       ; Previous state = open?
         Goto    Blink         ; No - Exit
; Yes - Open-Closed Transition
Bac     Bcf      lastflg       ; Update Flag=Closed

tglbtn  movlw    LEDbtn        ; toggle LEDbtn
        xorwf    sGPIO,w
        movwf    sGPIO
        movwf    GPIO

; Time check for LEDblk flashing
Blink   decfsz   stepcnt,f     ; counter-1 = 0 ?
         Goto    dllp          ; No - Other Movlw Hold
        Loop     LEDblnk       ; toggle LEDblnk
        XORWF    sGPIO,w
        movwf    sGPIO
        movwf    GPIO
; Other Cycle
        Goto     Mainloop      ; Charging Counter

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

8A_10F20x.asm

```
;*********************************************************************
; 8A_10F20x.asm
;---------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 8A
;                     An LED flashes while a button controls a
;                     other LEDs. Timing with TIMER0.
;
;    PIC            : 10F200/202/204/206
;    Support        : MPASM
;    Version       A: V.20x-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;---------------------------------------------------------------------
;
;  Pin use :
;  ----------------
;     10F200/202/204/206
;              @ 8 pin DIP              @ 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
;
;   [] only 204/206
;
;*********************************************************************
;=====================================================================
;             DEFINITION OF PORT USE
;
; GPIO map
; | 3 | 2 | 1 | 0 |
; |-----|-----|-----|--------|
; | BTN |     |LED3 | LED1|
;
;#define         GPIO,GP0     ; LED1 flashing
;#define         GPIO,GP1     ; LED3 controlled by the button
;#define         GPIO,GP2     ; LED4 flashing
#define button GPIO,GP3      ; RES button
```

```
LEDblnk     = 0x1   ; Flashing LED control number
LEDbtn      = 0x2   ; LED control number by push button

#define lastflg sGPIO,7     ; Button Previous Status Flag

;*******************************************************************
; ###################################################################
; Choice of #ifdef
 processor    10F200
        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 procanalog
 #endif
 #ifdef __10F206
        LIST      p=10F206
        #include <p10F206.inc>
#define procanalog
 #endif
        radix dec

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


; ###################################################################
;                        RAM
;
       CBLOCK 0x10          ; Data RAM
      stepcnt               ; Counter for flashing
      sGPIO                 ; shadow I/O
       ENDC

; ###################################################################
;                           CONSTANTS
;
stepnum     = .31   ; Number of repetitions for 500ms

; bit of TMR0
#define bit4096 TMR0,4 ; bit 4 -> 16*256=4096us
#define bit8192 TMR0,5 ; bit 5 -> 32*256=8192us
#define bit16384 TMR0,6 ; bit 6 -> 64*256=16384us
#define bit32768 TMR0,7 ; bit 7 -> 128*256=32768us

; ###################################################################
;===================================================================
;                         RESET ENTRY
```

```
;
; Reset Vector
        ORG     0x00

; internal oscillator calibration; FOSC4 force disabled in the
; Case of incorrect calibration value
      andlw   0xFE
      movwf   OSCCAL

      CLRF    GPIO        ; latch I/O presets
      movlw   0x80        ; preset sGPIO, flag=open, LED off
      movwf   sGPIO


 #ifdef procanalog        ; only 10F204/206
; Disable Comparator, No Output
; CMCON def '11111111'
;             1------- cmpout
;             -1------ Output disabled
;             --1----- Normal polarity
;             ---1---- T0CS for Timer0 in
;             ----0--- Disable Comparator
;             -----1-- vref- = CIN-
;             ------1- vref+ = CIN+
;             -------1 no wakeup
     movlw b'11110111'
     movwf CMCON0
 #endif

; no T0CKI, prescaler 1:256
; OPTION def '11111111'
;             1------- GPWU disabled
;             -1------ GPPU disabled
;             --0----- internal clock
;             ---1---- done
;             ----0--- Prescaler to Timer
;             -----111 1:256
     movlw b'11010111'
     OPTION

; I/O initializations on reset
      movlw b'11111000'      ; GP2,1,0 as TRIS
      output GPIO
;-----------------------------------------------------------------
         CLRF    TMR0         ; initialize Timer0 counter

; 1s mainloop LED
flashing:
; 500ms delay (25 x 20ms)
        movlw    stepnum      ; Number of
        repetitions MOWF      stepcnt     ; in the
        counter
dllp    BTFSS    bit8192      ; Timer Bit = 1 ?
         Goto    dllp

; yes - test button for LEDbtn control
BTNCHCK BTFSS    Button       ; open button ? Goto
```

```
          BCA           ; No - Checking
Flags
```

```
        BTFSC    lastflg       ; Yes - Was it closed
                                 before?
         Goto    Blink         ; No - Exit
; Yes - Open-Closed Transition
        Bsf      lastflg       ; Update flag=open
        goto     tglbtn        ; operate LEDbtn

BCA     BTFSS    lastflg       ; Previous state = open?
         Goto    Blink         ; No - Exit
; Yes - Open-Closed Transition
Bac     Bcf      lastflg       ; Update Flag=Closed

tglbtn  movlw    LEDbtn        ; toggle LEDbtn
        xorwf    sGPIO,w
        movwf    sGPIO
        movwf    GPIO

; Time check for LEDblk flashing
Blink   decfsz   stepcnt,f     ; counter-1 = 0 ?
         Goto    Waitlp        ; No - Other Movlw Hold
        Loop     LEDblnk       ; toggle LEDblnk
        XORWF    sGPIO,w
        movwf    sGPIO
        movwf    GPIO
; Other Cycle
        Goto     Mainloop      ; Charging Counter

Waitlp  BTFSC    bit8192       ; Waiting for the
                                 end of time
         Goto    Waitlp
        Goto     dllp          ; New Loop

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

8A_T0CKI

```
;**********************************************************************
; 8A_T0CKI
;----------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 8A
;                     Flashes an LED with F=SC/4 connected to T0CKI
;                     .
;    PIC            : 16F505/506/526
;    Support        : MPASM
;    Version        : 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:
;    RB4/OSC2/CLKOUT          3: CLKOUT
;    RB3/! MCLR/VPP           4:
;    RC5/T0CKI                5: T0CKI
;    RC4/[C2OUT]              6:
;    RC3                      7:
;    RC2/[CVref]              8:
;    RC1/[C2IN-]              9:
;    RC0/[C2IN+]              10:
;    RB2/[C1OUT/AN2]          11:
;    RB1/[C1IN-/AN1/]ICSPC 12: Out   LED1
;    RB0/[C1IN+/AN0/]ICSPD 13: OUT   LED0
;    Vss                      14: --
;
;  [] only 506/526
;**********************************************************************
;                       PROCESSOR DEFINITION
 #ifdef ___16F505
        LIST p=16F505
        #include <p16F505.inc>
 #endif
 #ifdef ___16F506
        LIST p=16F506
        #include <p16F506.inc>
#define procanalog
 #endif
 #ifdef___16F526
        LIST p=16F526
        #include <p16F526.inc>
#define procanalog
 #endif

        Radix     DEC

; ######################################################################
;====================================================================
;                          CONFIGURATION
;
; Internal oscillator, no WDT, no CP,
 pin4=GP3; #ifdef   16F505
 __config  _IntRC_OSC_CLKOUTEN & _WDT_OFF & _CP_OFF & _MCLRE_OFF
 #endif

 #ifdef___16F506
 __config  _IntRC_OSC_CLKOUTEN & _IOSCFS_OFF & _WDT_OFF & _CP_OFF &
_MCLRE_OFF
 #endif

 #ifdef___16F526
 __config  _IntRC_OSC_CLKOUT & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_MCLRE_OFF
 #endif


; ######################################################################
;                        RAM
;
       UDATA_SHR            ; Data RAM
sPORTB res 1                ; shadow for PORTB
LED0cnt res 1               ; flashing counter LED0
LED1cnt res 1               ; LED flash counter1
D1      RES 2               ; Waste time counters

; ######################################################################
;====================================================================
;                   DEFINITION OF PORT USE
;
;
; PORTB map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|     |     |     |     | LED1| LED0|

#define     LED0    PORTB, RB0
#define     LED1    PORTB, RB1
;
#define lastflg    sPORTB,7

LED0bit    = 1       ; LED0 control number
LED1bit    = 2       ; LED1 control number
```

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

```
;
LED0step   = .15    ; Number of repetitions approx. 500ms (32ms
                    each)
LED1step   = .31    ; Number of repetitions approx. 250ms (8ms
                    each)

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

; Calibrate Internal Oscillator
        MOMOVWF OSCCAL
        Pagesel Init
        Goto    Init

; Subroutines & Tables


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

Init:
; Reset Initializations
        CLRF    PORTB           ; PORTB preset latch to 0
        CLRF    sPORTB          ; Shadow Presets

 #ifdef procanalog              ; only for 12F510
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the digital MOVLW function
                0x77
        movwf   CM1CON0
        movwf   CM2CON0
 #endif

; TRISPORTB       --111100       RB1:0 out
        movlw   b'11111100'
        Tris    PORTB

; set Timer0 T0CKI with prescaler 1:256
      ; default b'11111111'
      ;           1-------     GPWU Disabled
      ;           -1------     GPPU disabled
      ;           --1-----     External Clock
      ;           ---1----     Falling
      ;           ----0---     prescaler to Timer0
      ;           -----111     1:256
        movlw   b'11110111'
        OPTION

        CLRF     TMR0          ; Preset Counter

; 1s mainloop LED
flashing:
```

```
; 32.7ms delay
```

```
        movlw    LED0step      ; number of
        movwf    LED0cnt       ; repetitions
        movlw    LED1step      ; lampegio LED0
        movwf    LED1cnt       ; number of
                                 repetitions
                                 flashing LED1

dllp    BTFSC    TMR0,7        ; 32,768ms ?
         GOTO    blink0        ; yes - flashing LED0
        Bcf      lastflg       ; no - flag=no
        Goto     blink1        ; LED flashing1

; flashing loop LED0
blink0  BTFSC    lastflg       ; previous state? yes
         GOTO    blink1        ; - flashing LED1 no
        BSF      lastflg       ; - flag=yes counter-
        decfsz   LED0cnt,f     ; 1 = 0 ?
         goto    blink1        ; no - flashing LED1
        movlw    LED0bit       ; toggle LED0
        xorwf    sPORTB,w
        movwf    sPORTB
        movwf    PORTB
        movlw    LED0step      ; Step Meter Recharge
        movwf    LED0cnt
        ; Goto    dllp          ; New Cycle

blink1:                        ; LED Flashing Loop1
; Hold 8 ms        ; 7998
        Cycles MOVLW     0x3F
        movwf   D1
        movlw   0x07
        movwf   d1+1
Delay8ms_0
        decfsz  D1, F
        goto    $+2
        decfsz  d1+1, f
        goto    Delay8ms_0
        ; Goto   $+1            ; 2 cycles

        decfsz   LED1cnt,f     ; counter-1 = 0 ?
         Goto    dllp          ; No - Other Hold Loop
        movlw    LED1bit       ; yes - toggle LED1
        xorwf    sPORTB,w
        movwf    sPORTB
        movwf    PORTB
        movlw    LED1step      ; Step Meter Recharge
        movwf    LED1cnt

        Goto     dllp          ; Other Cycle

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

8A_T0CNTR

```
;*********************************************************************
; 8A_T0CNTR
;---------------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 8A
;                     Timer0 as counter from external signal on
;                     T0CKI. Display on a 7-digit digit.
;
;    PIC            : 16F505/506/526
;    Support        : MPASM
;    Version        : 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    SEG DP
;    RB4/OSC2/CLKOUT            3: Out    sec g
;    RB3/! MCLR/VPP             4:
;    RC5/T0CKI                  5: T0CKI
;    RC4/[C2OUT]                6: Out    seg e
;    RC3                        7: Out    Sec D
;    RC2/[CVref]                8: Out    sec c
;    RC1/[C2IN-]                9: Out    SEG B
;    RC0/[C2IN+]                10: Out   seg a
;    RB2/[C1OUT/AN2]            11: Out   Seg F
;    RB1/[C1IN-/AN1/]ICSPC 12: Out    LED1
;    RB0/[C1IN+/AN0/]ICSPD 13: Out    LED0
;    Vss                        14: --
;
;  [] only 506/526
;*********************************************************************
;                         PROCESSOR DEFINITION
 #ifdef ___16F505
        LIST p=16F505
        #include <p16F505.inc>
 #endif
 #ifdef ___16F506
```

```
        LIST p=16F506
        #include <p16F506.inc>
#define procanalog
 #endif
 #ifdef___16F526
        LIST p=16F526
        #include <p16F526.inc>
#define procanalog
 #endif

        Radix     DEC


; ####################################################################
;===================================================================
;                           CONFIGURATION
;
; Internal oscillator, no WDT, no CP,
 pin4=GP3; #ifdef   16F505
 __config  _IntRC_OSC_RB4EN & _WDT_OFF & _CP_OFF & _MCLRE_ON
 #endif

 #ifdef___16F506
 __config  _IntRC_OSC_RB4EN & _IOSCFS_OFF & _WDT_OFF & _CP_OFF &
_MCLRE_ON
 #endif

 #ifdef___16F526
 __config  _IntRC_OSC_RB4  & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_MCLRE_ON
 #endif


; ####################################################################
;                         RAM
;
       UDATA_SHR             ; Data RAM
savew   RES 1                ; Temporary
D1      RES 1                ; Temporary for Delay
D2      RES 1
D3      RES 1


; ####################################################################
;===================================================================
;                   DEFINITION OF PORT USE
;
;
; PORTC map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;|T0CKI|seg e|seg d|seg c|seg b|seg a|

; PORTB map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;| dp |seg g| MCLR|seg f|      |      |
#define segdp PORTB,5 ; DP Segment
```

```
; ####################################################################
;                            CONSTANTS
;
Time  = .20   ; Number of repetitions approx. 200ms
              (20*10ms)

; ####################################################################
;                            MACRO
;
 #include C:\PIC\Library\Baseline\compmacro.asm

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

; Calibrate Internal Oscillator
        MOMOVWF OSCCAL
        Pagesel Init
        Goto    Init

; Subroutines Vectors
; Display Control:
        pagesel Disp7segm
        goto    Disp7segm

; Wait Milliseconds
Delayms:
        pagesel Delay10msW
        goto    Delay10msW

; ####################################################################
;                         MAIN PROGRAM
;
Init:
; Reset Initializations
        CLRF    PORTB          ; PORT latch preset to
        0 clrf  PORTC

 #ifdef procanalog             ; only for 12F510
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the digital MOVLW function
                0x77
        movwf   CM1CON0
        movwf   CM2CON0
 #endif

; TRISPORTB       --111111
        movlw   b'11000000'    ; All Out
        Threes  PORTB
        Tris    PORTC

; Timer0 TOCKI without prescaler is the default to

        POR clrf TMR0         ; Preset Counter to
```

0

```
Mainloop:
        movlw    Time          ; Number of
                                 repetitions
        Pagesel  Delayms
        Call     Delayms       ; for 200ms
        Pagesel  $
        movf     TMR0,w        ; Read Counter
; W-> display
        Pagesel  Display
        Call     Display       ; Transfer to Display
        Pagesel  $

        Goto     Mainloop


;*******************************************************************
;                  SUBROUTINES & TABLES in PAGINA1
;
TABLES   CODE 0x200

; Segment Data Table - Common Cathode Display
; control for segments a-e on RB4:0
segtbl1 andlw 0x0F ; only low nibble addwf PCL,f
        ; PC tip
        retlw  b'00011111'  ;  "0"  -|-|-|E|D|C|B|A
        retlw  b'00000110'  ;  "1"  -|-|-|-|-|C|B|-
        retlw  b'00011011'  ;  "2"  -|-|-|E|D|-|B|A
        retlw  b'00001111'  ;  "3"  -|-|-|-|D|C|B|A
        retlw  b'00000110'  ;  "4"  -|-|-|-|-|C|B|-
        retlw  b'00001101'  ;  "5"  -|-|-|-|D|C|-|A
        retlw  b'00011101'  ;  "6"  -|-|-|E|D|C|-|A
        retlw  b'00000111'  ;  "7"  -|-|-|-|-|C|B|A
        retlw  b'00011111'  ;  "8"  -|-|-|E|D|C|B|A
        retlw  b'00001111'  ;  "9"  -|-|-|-|D|C|B|A
        retlw  b'00010111'  ;  "A"  -|-|-|E|-|C|B|A
        retlw  b'00011100'  ;  "b"  -|-|-|E|D|C|-|-
        retlw  b'00011001'  ;  "C"  -|-|-|E|D|-|-|A
        retlw  b'00011110'  ;  "d"  -|-|-|E|D|C|B|-
        retlw  b'00011001'  ;  "E"  -|-|-|E|D|-|-|A
        retlw b'00010001' ; "F" -|-|-|E|-|-|-|AT

; Command for F-G Segments     on RB2 and RB4
segtbl2 andlw   0x0F          ; Low nibble only
        addwf   PCL,f         ; PC tip
        retlw   b'00000100'   ; "0" -|-|F|
        retlw b'0000000000    ; "1" -|-|-|
                         '
        retlw   b'00010000'   ; "2" -|G|-|
        retlw   b'00010000'   ; "3" -|G|-|
        retlw   b'00010100'   ; "4" -|G|F|
        retlw   b'00010100'   ; "5" -|G|F|
        retlw   b'00010100'   ; "6" -|G|F|
        retlw b'0000000000    ; "7" -|-|-|
                         '
        retlw   b'00010100'   ; "8" -|G|F|
        retlw   b'00010100'   ; "9" -|G|F|
        retlw   b'00010100'   ; "A" -|G|F|
retlw  b'00010100'   ; "b" -|G|F|
retlw  b'00000100'   ; "C" -|-|F|
        retlw  b'00010000'    ; "d" -|G|-|
        retlw  b'00010100'    ; "E" -|G|F|
        retlw  b'00010100'    ; "F" -|G|F|

Disp7segm
        movwf savew
        call   segtbl1
        movwf PORTC
        movf   savew,w
        Call   segtbl2
        movwf portb

; if count exceeds Fh, turn on dp CFLSA
        savew, 0x0F
         retlw 0
        Bsf     SEGDP
        retlw    0

; waste time delay 10ms*W
 #include C:\PIC\Library\Baseline\Delay10msW

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