Knowledge of baseline

Interrupts and Multitasks

Multitasking refers to the ability of the microcontroller to perform several actions at the same time. The **interrupt** function is the key. It is a fundamental element of microprocessors and microcontrollers: the ability of peripherals to interrupt the main flow of the program to require specific management allows you to perform functions that would not be possible through polling. To handle asynchronous events more flexibly and effectively, the most advanced processors add other functions, such as the possibility of multi-level interrupts, call priority, DMA (Direct Memory Access), etc.

Unfortunately Baselines do NOT have interrupts!

These "elementary" PICs, for reasons of simplicity of construction (read: cost) do not include in their architecture any interrupt handling (with the exception of 16F527 which is a rather anomalous Baseline).
Therefore, the analysis of the peripherals integrated in the chip (which, moreover, are very limited), can only be done through polling. This significantly limits the possibility of using these chips in various applications.

By **interrupt** we mean, in a nutshell, that a boundary condition, for example the overflow of a counter, the arrival of a piece of data received from the serial line, the end of the AD conversion, at any time, calls the attention of the processor through an appropriate mechanism, shifting the flow of instructions to the handling of the event, and then releasing the control to the main stream, as determined by the programmer. This mechanism includes one or more interrupt vectors, enable flags, signaling flags, all managed by the program.

By **polling** , on the other hand, we mean that the boundary condition must be identified exclusively through a constant analysis of the relative register to capture the moment sought.
This means that the processor is mainly engaged in this task and has very little ability to perform other complex tasks.

For any serious programmer, the absence of interrupts is dramatic, as this situation greatly limits the possibilities of the microcontroller and requires excessive care of execution times. We must, however, keep in mind that the speed of execution of instructions is usually much faster than the events that have to be checked, thus giving the way to follow algorithms of a certain complexity nested within each other or shared in a certain time. "Multitasking" of this kind is a ploy, as a complete asynchronicity of events is not allowed and this can limit the number of possible actions, their complexity and also the temporal precision of the same.

However, by restricting the field of action to the possibilities of chips, even with Baselines it is possible to create "multitasking" applications of a certain effect.
The key is time, i.e. the ability to deal with one event while following another, and

This is possible when the flow of instructions is fast enough to be able to "fit in" the various event handlers in order to avoid data loss.

Let us now take some examples of how it is possible to follow a certain number of events in the "same" time.

A simple multitask

We have 8 LEDs available on the LPCub. We want to make them flash at different frequencies. This means that each LED will have its own on/off period different from the others.
This can be achieved by following the tasks consecutively: the main program is essentially a switch that rotates continuously, selecting one task at a time.

The tasks are deliberately very simple: each task controls an LED (**LEDx**) and has its own counter (**cntrx**) that is preset with the time value (**limitx**) where the task reverses the state of the LED. A **syscntr counter** advances with each click of the selector and each task takes into account these clicks: when they are even **cntrx**, the LED is toggled:

```
; Each tick limit increments local counter
Taskx movlw  limith          ; w = limith
      subwf  syscntr,w       ; w = syscntr-w = syscntr-limith
      BTFSS  STATUS,Z        ; if syscntr = limith -> Z=0 ->
       Goto  Main            ; If several go back to Main
      incf   cntrx,f         ; if not, increment local counter
;each limitx Increments  LED drive
      movlw  limitx          ; w = limit0
      subwf  cntrx,w         ; w = cntrx-w = cntrx-limitx
      BTFSS  STATUS,Z        ; if syscntr = limitx -> Z=0 ->
       Goto  Main            ; If several go back to Main
      CLRF   cntrx           ; if not, delete local counter
      movlw  LEDX            ; LED drive
      XORWF  PORTB,f
      Goto   Main            ; Return to Loop
```

Since the comparison between counters and limits is repeated a large number of times, we replace it with a macro, using a mnemonic from the Parallax Assembler:

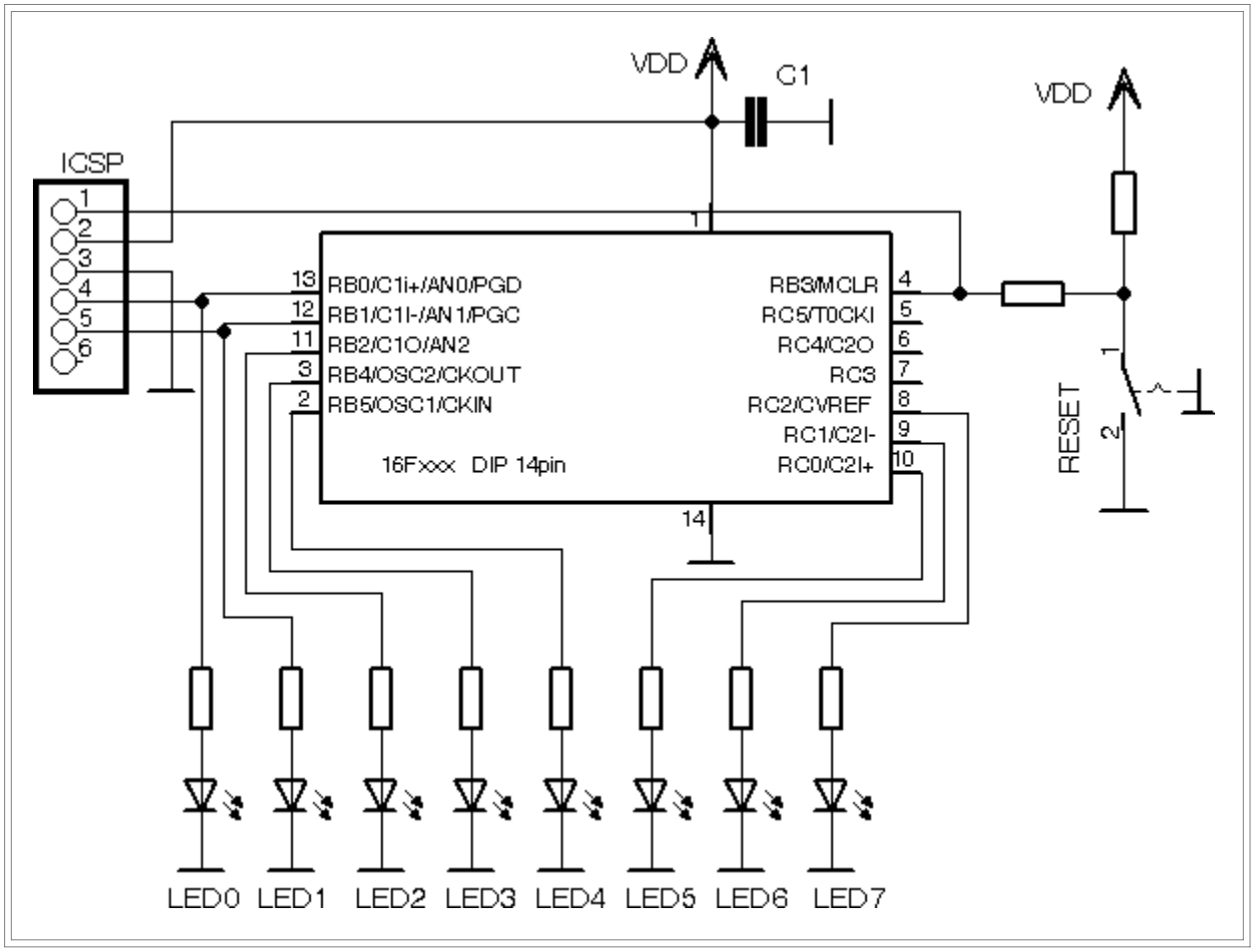
```
; Compare File with Literal, Jump if Non Equal
; Compare files and literals and jump to destination if
different CFLNJE     MACRO  file, lit, dest
    movlw  lit               ; w = lit
    subwf  file,w            ; w = file-w = file-lit
    btfss  STATUS,Z          ; se file = lit -> Z=0 -> esc
     goto  dest              ; if different go to dest
         ENDM
```

It is a Macro with three variable parameters, so the source section seen above is lightened, becoming, for example for Task1:

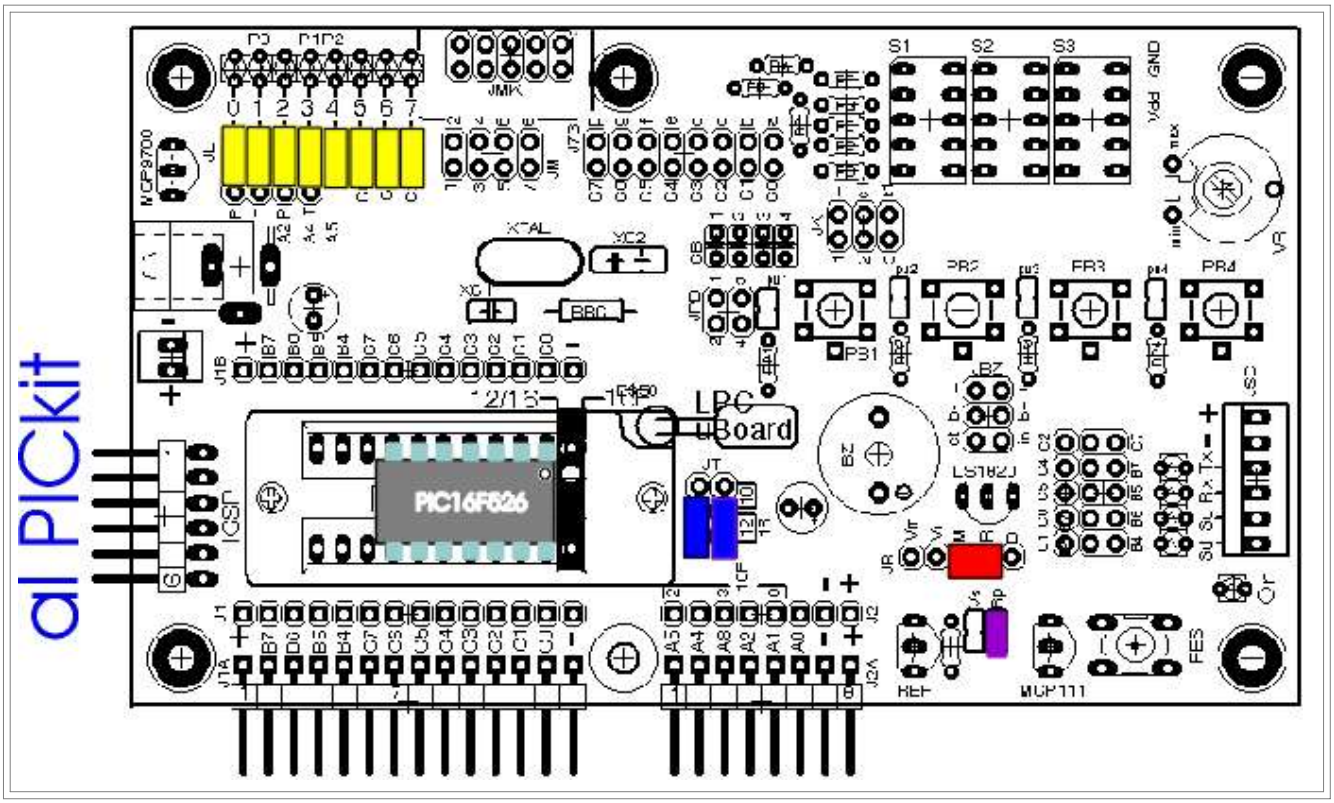
```
; Each tick limit increments local counter
Task1 CFLNJE syscntr, limith, main ; If different it goes back
      to main incf          cntr1,f     ; if no increment local
      counter
;Each limit1 increments of the counter, operates LEDs
      clrf    cntr1         ; if not, delete local counter
      movlw   led1          ; LED drive
      xorwf   PORTB,f
      goto    main          ; Return to Loop
```

The duration time of the ticks of the "selector" must be calculated by adding the execution times of the instructions (1us for all opcodes, except `goto` which is worth 2us).

The wiring diagram will look like this:



We use a 14-pin chip and on the **LPCuB** we associate part of the PORTC with the PORTB to complete the sequence of the 8 LEDs.



The "yellow" jumpers connect the ports to the LEDs.

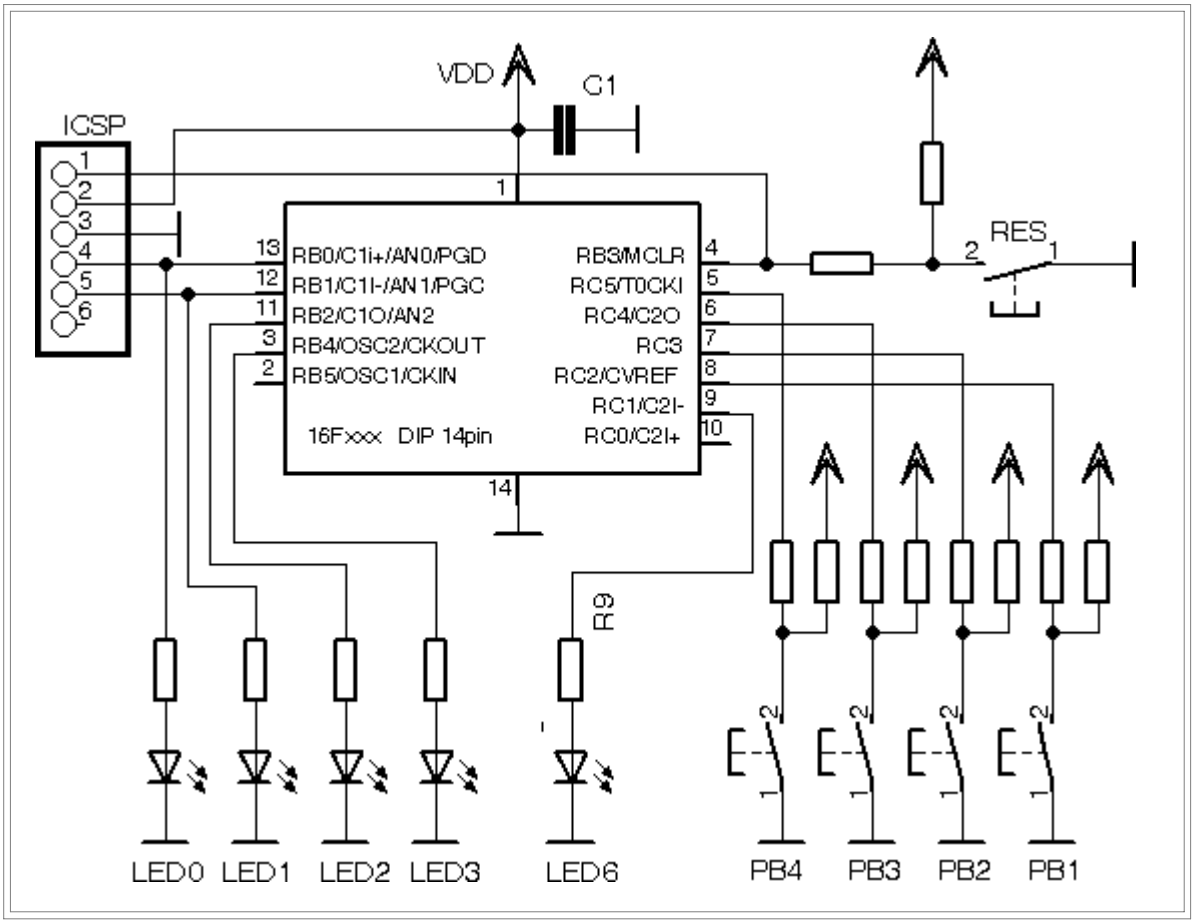
The program (***10A_01.asm***) is provided with its project. Once the source has been compiled and the chip has been programmed, you will see the 8LEDs flashing in a seemingly random way, but in fact each of them responds to the times set in the relative task. By varying the **`limitx`** parameters, we vary the relative LED cycle.

In practice, in addition to the actions on the LEDs, in each task it will be possible to replace different operations, always within the limits imposed by the counting loops.

More tasks with Timer0

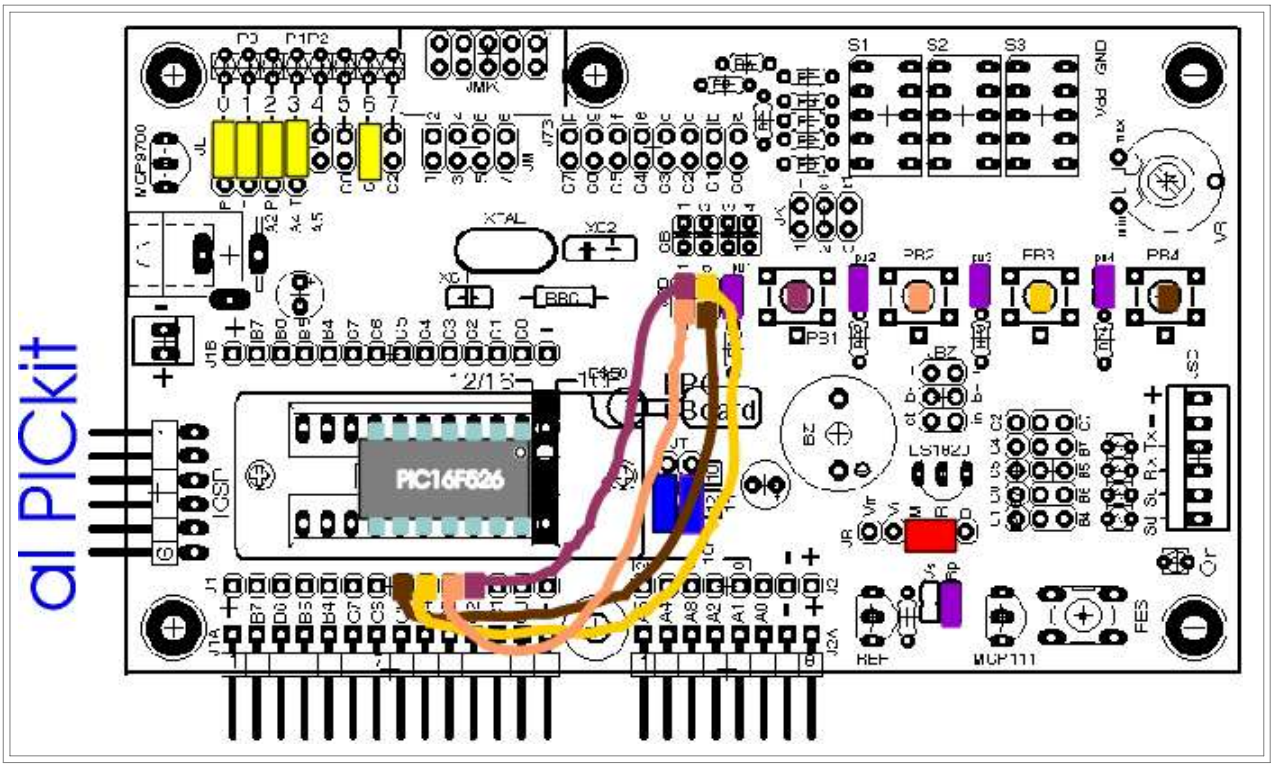
The above example does not lend itself easily to actions where the time element is important, but by using the Timer0, it is possible to cadence with great precision the "switch" that selects the various tasks. Let's look at a possible application.

We have 4 buttons that each have to control an LED. Of course, the buttons can be pressed at any time, even more than one at a time, and there must be the corresponding LED switching. We use a 14-pin PIC:



The LEDs depend on **PORTB**, while the buttons depend on **PORTC**. The Reset is active and pressing it brings the execution of the program back to zero.

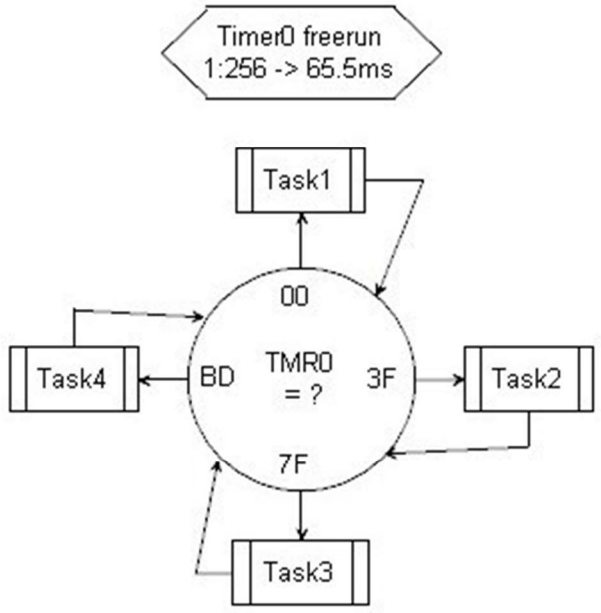
About the **LPCuB**:



The "yellow" jumpers connect the LEDs.
Flying jumpers connect the buttons.
"Purple" jumpers enable button pull-ups. The
RES button is connected and active.

The program

The management of each button is a separate task; So we have 4 tasks, and since they perform similar actions, we can devote the same amount of time to them.
It goes without saying that, having to deal with mechanical contacts, you also have to consider the problem of rebounds.



We set up Timer0 with a 1:256 prescaler and count it from 0 to overflow, which at 4MHz happens after:

t = loan * TMR0 * 1us = 256 *256 * 1 = 65536us

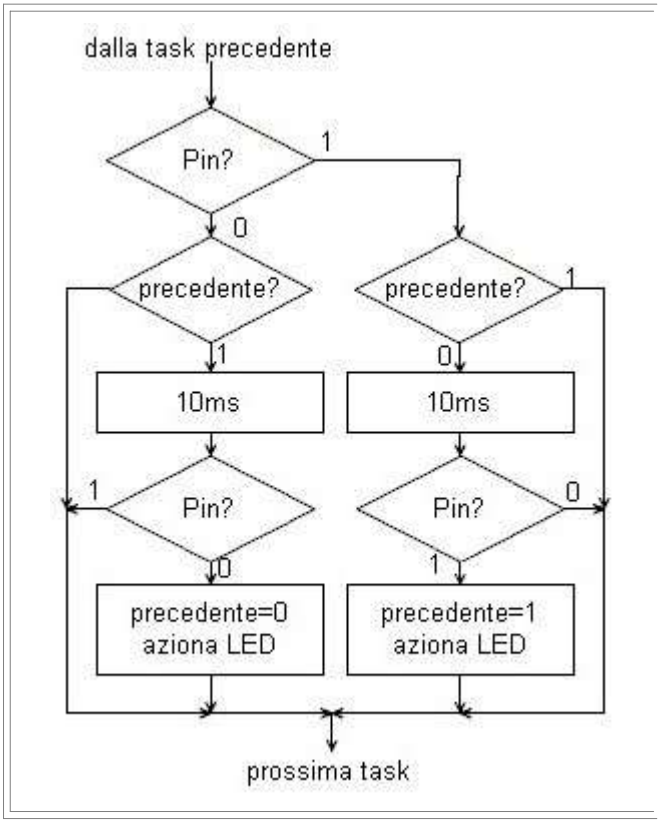
We have 4 tasks and we dedicate 1/4 of the time to each one, or about 16ms: this is the time that a task has available to perform its action.
How do we count these times? Simply by following the progress of the count in the **`TMR0 register`**.

TMR0	Time	Task
00	0	1
3F	16ms	2
7F	32ms	3
BD	48ms	4

The counter starts from 0 and there also starts the action of the first task, which ends after about 16ms, when the counter has reached 3Fh. This is where the second task starts, ending at 7Fh. The third one has time until BDh and the last one has a little more time, until the timer has overflowed and resumes counting from 0.

In our case, the tasks are intended to check the status of the buttons and control the LEDs accordingly. Let's see how to schematize in a block diagram.

There are two mirrored branches, one for the high-low transition (contact closure) and one for the low-high transition (opening):



The logical level of the pin to which the button is connected is checked and compared with the previous state, which is initially set to 1 (the buttons have pull-ups and close to ground, so the pin is at 1 if the button is open, at 0 if it is pressed).

1. If there has been no change, we move on to waiting for the next task.
2. If there has been a change, it waits 10ms to check if it is indeed a change in state, eliminating any bounces.
3. If after the debounce time the contact is confirmed in the new location, the desired action is taken and the previous status flag is updated.

If after the debounce time the status is not validated, no action is taken and the next task is waited.

The corresponding listing (***10A_Alpb.asm***) uses the increasing count of the timer also for the debounce time: the intervention times of the four tasks are fixed and the debounce is carried out by adding the equivalent count of 10ms (27h). The times are declared as labels and therefore can be quickly modified; Let's use the statement **constant**:

```
;*****************************************************************
;                              CONSTANTS
; Task duration (approx. 16ms @4MHz)
  constant tsktime=256*pre(FOSC/4)*2560x3F
; debounce time (circa 10ms @4MHz)
  constant dbtime=0x27
```

Again to have labels in use and not absolutes, we also define the ratio between pins and LEDs:

```
;=================================================================
; DEFINITION OF PORT USE
;
;PORTC map
;| 5    | 4    | 3    | 2    | 1    | 0   |
;|------|------|------|------|------| ----|
;| PB4  | PB3  | PB2  | PB1  | LED6|     |
;
pinport  equ     PORTC
#define pin4in PORTC,5    ; push button PB4
#define pin3in PORTC,4    ; push button PB3
#define pin2in PORTC,3    ; push button PB2
#define pin1in PORTC,2    ; push button PB1
#define LED6   PORTC,1    ; LED
```

```
;#define PORTC,0          ;
pin4 equ 0x10 ; posizione nel port
pin3 equ 8     ;
pin2 equ 4     ;
pin1 equ 2     ;

;P ORTB map
;| 5    | 4    |  3   |  2   |  1   |  0 |
;|------|------|------|------|------| ----|
;|      | LED3| MCLR|LED2 |LED1 |LED0 |
;
#define LED0  PORTB,0   ; LED
#define LED1  PORTB,1   ; LED
#define LED2 PORTB,2    ; LED
;#define   PORTB,3     ;   MCLR
#define LED3 PORTB,4    ; LED
;#define PORTB,5         ;
```

A task, such as the first one:

```
; Task1
; test for pins at 0
Task1   BTFSC PIN1IN          ; pin = 0 ?
         goto  t1_2           ; no-  Other previous
        btfss  pin1lst        ; Yes- test = 1 end
         goto  t1_end         ; no-  task debounce
t1_0    movlw  dbtime         ; Yes  debounce?
        xorwf  TMR0,w         ; -
        Skpz                    End  waiting
         Goto  t1_0           ;      pin always at
        BTFSC  pin1in         ; no-  0? End of Task
         GOTO  t1_end         ; Yes- clear flag
        Bcf    pin1lst        ; no-
                                Yes-
        Bsf    LED0           ; LED Onler
; exit
t1_end movlw   tasktime       ; End of time task?
        xorwf  TMR0,w
        skpz
         Goto  t1_end         ; No - Waiting
        Goto   Task2          ; Yes - Next Task
; test for pins to 1
t1_2    BTFSC PIN1LST         ; previous = 0
         goto  t1_end         ; No - End Task
t1_3    movlw  dbtime         ; Yes - Debounce
        xorwf  TMR0,w         ; End Debounce
        skpz
         Goto    t1_3         ; No - Waiting
        BTFSS Pin1in          ; Yes - Pin always at
         Goto   t1_end        ; 1? No - End of Task
        BSF  PIN1LST          ; Yes - Set Flag
        Bcf    LED0
        Goto   t1_end         ; exit
```

The large number of instructions and the continuous tests and jumps can be easily followed by keeping an eye on the block diagram presented above.
Let us look at one more example of the importance of this instrument, which is essential not only for understanding the programme, but also for its drafting.

All tasks are the same; The only differences are in the time limits, which are increasing. The debounce time is realized by waiting for 27h counts of TMR0 which is equivalent to about 10ms. Thus, the actions with respect to the progress of the timer count are:

TMR0	Time	Task	Action
00	0	1	Task Start
27	0+10ms		End Debounce
3F	16ms		End of Task
40		2	Task Start
66	16+10ms		End Debounce
7F	32ms		End of Task
80		3	Task Start
A6	32+10ms		End Debounce
BD	48ms		End of Task
BE		4	Task Start
E4	48+10ms		End Debounce
FC	64.5ms		End of Task
FD		5	Task Start
FF	65535us		End of Task

after FFh the timer overflows and resumes from 0.

Since there's still time left, let's insert an additional task that flashes an LED at about 500ms:

```
; Task5 - flashes LED
Task5   decfsz cntr5        ; 500ms ?
         goto  Taskloop     ; no - other loop
; ogni 500ms circa
t5_0    movlw task5time     ; Charging counter
        mowf   cntr5
        movlw led6pin       ; toggle led
        xorlw led6port,f
```

```
        goto    Taskloop     ; loop
```

A counter is decremented every 8 passes, equal to about 500ms; when it is reset, it is recharged for the next loop and the state is changed to the connected LED.

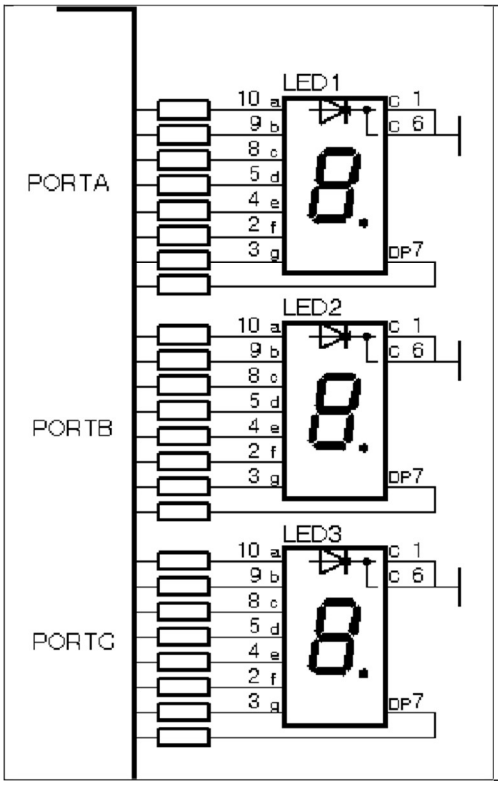
The result will be as follows: by pressing the buttons, the corresponding LED will light up; releasing the button will turn off the LED, while LED6 will continue to flash. Multiple buttons can be pressed at the same time.

The source is compileable for 16F505/506/526.

Such a system can obviously be used for any other kind of multiple action; what is to be considered is that each action has a certain amount of time available, which can be different depending on the needs of the process to be controlled, possibly using counters in RAM where longer times are needed or changing prescalers if shorter times are needed.

Managing a multiplexed display

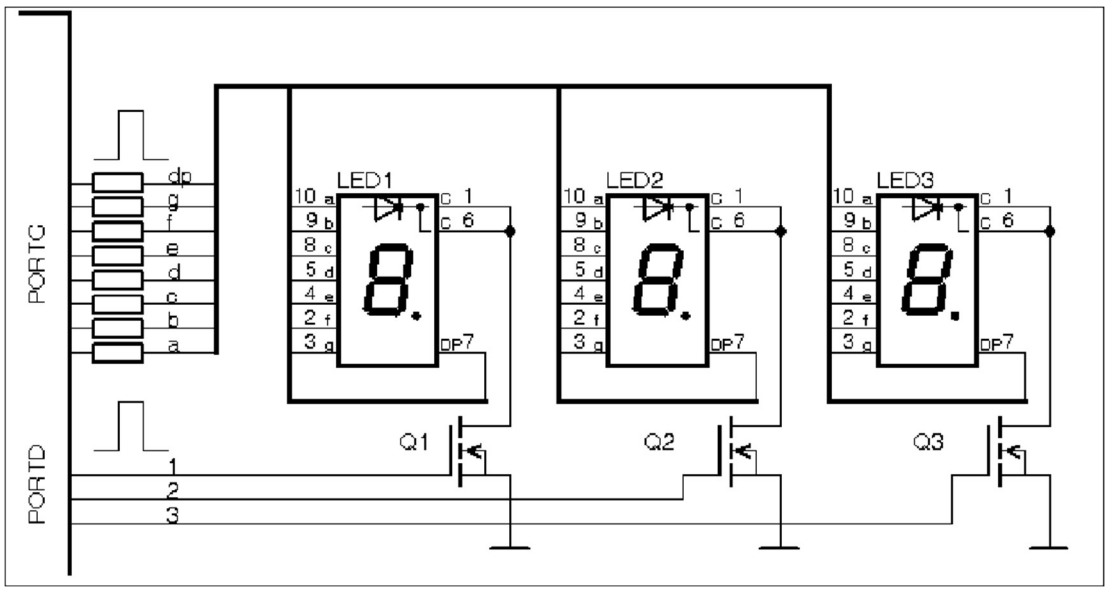
Typically, a multitask situation occurs in the seemingly simple case where we need to display 7-segment multi-digit data while at the same time we have to perform another operation, such as reading a temperature or a pulse or time count. The problem of managing multiple displays is quite easy to understand. If we have only one digit, as we have seen previously, there are no particular difficulties: it is enough to have a chip with a sufficient number of pins to control the segments. This is a "static" command, where the segments are always either on or off depending on the digit to be presented.



If, however, we are dealing with multiple displays, it is necessary to have a greater number of I/O pins, at least 7 for each digit, which means, in the case of two digits, 14 pins engaged in this action; and for three figures it would take as many as 21; with decimal points, it would take 24 I/O, which is three full 8-bit ports.
There is also a need for current-limiting resistors in the segments' LEDs in equal numbers as the segments themselves. This is quite an expensive circuit, also because it requires a chip with 28 or 40 or more pins, as well as involving a rather complex wiring to connect the display and microcontroller.

This is not an unlikely situation, as it is commonly used when it is necessary to minimize the electrical noise produced by the circuit (for example, in the vicinity of radio and TV equipment, where this noise would be a significant disturbance); In fact, the fast on/off switching of the currents in the LEDs due to the multiplex management generates a wide spectrum of interference even with harmonics at high frequencies.

Where this problem does not exist, multiplexing is used, which requires fewer pins:



You only need one I/O for each group of segments identified by the same letter and one additional I/O for each digit. There will be only 8 limiting resistors. For three displays you will have:

Link	Non-mux	mux
Display	3	**3**
I/O	24	**11**
Resistances	24	**8**
Transistor	0	**3**

The advantage in terms of components-cost is considerable. On the other hand, we are going to complicate the program considerably.
In this configuration, we rely on a point that is not always sufficiently clear: the persistence of the image in the eye. It is this: our vision does not seem to be able to distinguish images that follow one another less than 20ms from each other, collecting them, instead, as a single moving image. This effect is the basis of the camera, cinema, television, cartoons, etc.

We can apply this concept to our displays as well. We want to introduce the number 123, but, having only 7 lines that command the segments in common, we use this trick:

- We bring the mask suitable to turn on the digit 1 on the lines of the segments and we ground the common cathode of the first display by turning on the transistor. The number 1 will appear on this one and not on the others, since their cathodes are not connected to the Vss.
- We keep the segments on for some time, then turn off the transistor and disconnect the cathode of the first display.
- We put the mask for the digit 2 on the segment lines and ground the cathode of the second display: the digit will appear on this.
- We hold it for a while, then turn on the digit 3 on the third display and repeat the cycle.

If the repetitions are fast enough, thanks to the persistence of the image, all three displays will appear on, even if they are only one at a time. Here is solved the problem of multiplexing displays.

As you can see, this is a time problem and what better than a timer to manage time? If we approached the problem from the point of view of a microcontroller with interrupts, the action would be very simple:

- We adjust the timer to have an interrupt every few milliseconds
- We manage the interrupt to have the sequence of on/off displays

All this would happen without any intervention of the main flow of instructions, with an absolutely precise cadence.

However, we said that Baselines don't have interrupts and you have to "make do" in a different way. A few considerations are enough to arrive at the solution: essentially it must be noted that the succession of commands to turn on the displays in the right sequence are indeed a function of time, but they are not at all asynchronous, on the contrary, they are strictly synchronous, since the sequence of switching on and off must not be interrupted or slowed down, otherwise the sensation of the still image will be lost, replaced by that of annoying flickering of bright digits. Therefore, all you have to do is set the program to correctly manage the ignitions, which is not so complex, even if you have to follow the timer in polling.
The problem is, if we follow the timer in polling, do we have time left to do something else? Because the presentation of numbers on a display is certainly not the purpose of the program: the numbers will be derived from another action, such as the reading of a temperature or pressure or other quantity, or from the passage of time as happens in a clock.

Let's consider that the refresh time of the displays to ensure that their image remains fixed to the eye is of the order of milliseconds, while the instruction cycle of the controller is of the order of microseconds or less: at 8 MHz clock an instruction is executed in only 500 ns.
Therefore, we can take advantage of this situation to ensure that the "main" task, i.e. the measurement detection, is carried out in the "downtime" between the update of one display and the next.

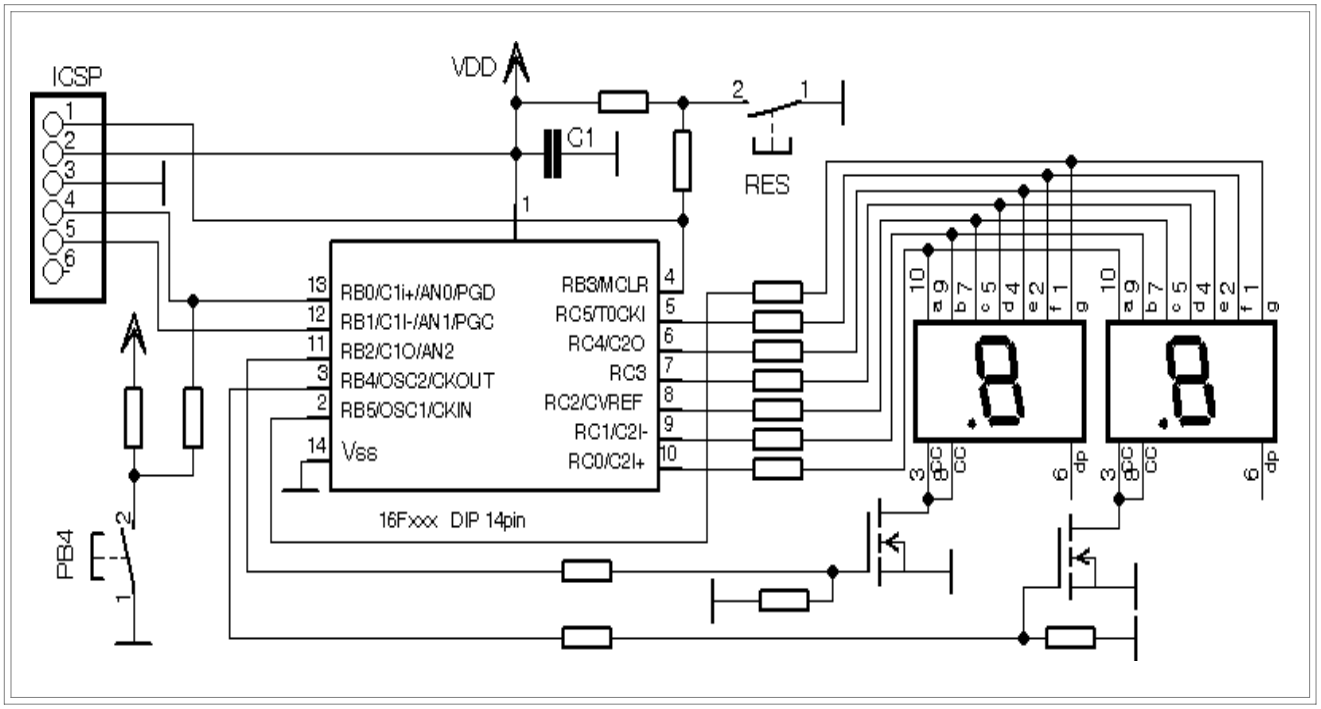
In this way, we can perform two different tasks (the measurement and the display) in a sequence that guarantees their executive safety.

One counter on two displays.

We want to make a device that counts pulses coming from a contact and displays them on a two-digit display.

To control two digits with 7 segments, excluding decimal points, 7+7 I/O pins would be needed. If, on the other hand, we use a multiplex, 9 are enough (7 for segments + 2 for MOSFETs), which allows us to use 14-pin PICs.

From the point of view of connections:



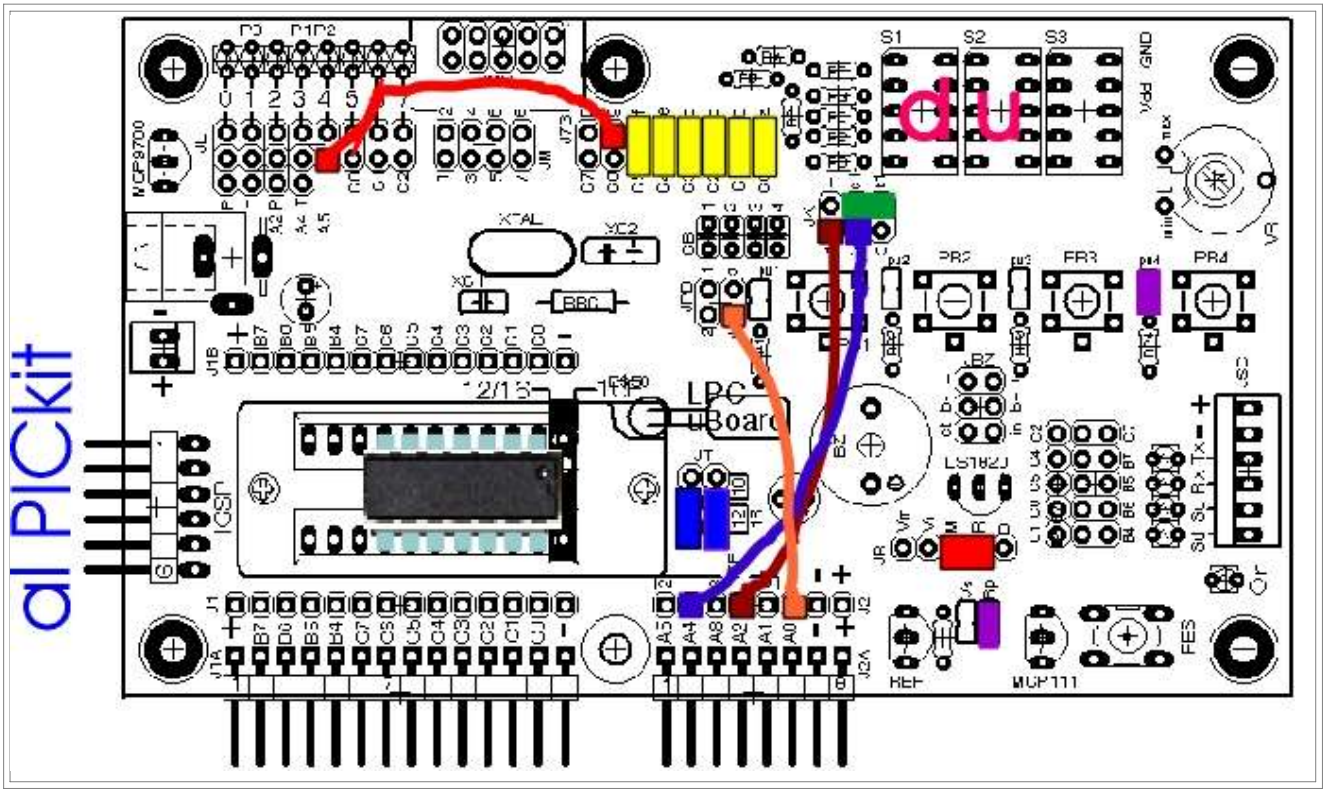
The two displays have the segments in parallel; the usual resistors limit the current in the LEDs ((the value depends on the supply voltage and the characteristics of the display. On the development board these are very low current elements, 2mA, and therefore the resistors have a value of about 1-1.3K).

Common cathodes are driven by two N MOSFETs (type 2N7000) that connect them to ground when the gate is enabled. A series resistor (10-100ohm) limits the gate current; a resistance between gate and ground (50-100k) prevents the MOSFET from conducting unless it is controlled by the microcontroller's I/O. These components are already present on the development board.

If the circuit is made differently and if you don't have a small MOSFET in TO-92 (2N7000, BC170, ZNVL110, etc.), you can use NPN transistors, also in TO92 (type BC547, BC317, 2N2222, 2N3904, etc.). In this case, since the bipolar transistor requires current to enter conduction, the resistance in series at the base will be between 1k and 4k7, while the resistance to ground will be between 10k and 47k (both are not critical).

A button (PB4) is the input that advances the count, while the RES button clears it. About the

LPCuB:



The "yellow" jumpers connect the ***f:a segments*** to **`RC5:0`**.
The "red" flying jumper connects **`RB5`** with the g-segment.
The "orange" flying jumper connects the **PB4 button** with **`RB0`**.
The "brown" flying jumper connects the tens display MOSFET to **`the RB2`**.
The "purple" flying jumper connects the units' display MOSFET to **`the RB4`**.
S1 is the number of tens and S2 is the number of units. S3 is
not used. "Purple" jumpers enable pull-ups.

STOP

1. Since PB4 is connected to RB0 which is also the programming input (PGD), **it is imperative to avoid pressing the button** during this operation or, for safety, disconnecting the flying jumper ("orange").
   Otherwise, the signal between the Pickit and the chip would be grounded, which could cause damage.
2. **Once the chip has been programmed, it is necessary to disconnect the Pickit from the ICSP socket** because its presence prevents proper operation by altering the signal to the

The source

We use Timer0 to produce the command cadence of the multiplex. We have two digits, the ignition of which we can alternate with a frequency of more than a hundred hertz to avoid flickering. There is no precise rule, since several factors contribute, first of all depending on the brightness of the digits and energy consumption:

- The longer the switch-on time, the brighter the segments will be, but the higher the power consumption
- The shorter the switch-on time, the lower the brightness, but the lower the current absorbed

In addition, it is necessary to determine a maximum command frequency that is essentially limited by the operations we have to perform outside of the multiplex management and the microcontroller clock. It should also be borne in mind that the more digits we have to check, the higher the scanning frequency will have to be.
Generally, it will be better to keep the value rather low, so that you can insert as many lines of instruction as possible in the "downtime" of the multiplex. For example, 250Hz is a good compromise: with two displays we can give each one an on time of 2ms, in which theoretically several hundred instructions of 1-2us each can be placed.

This is easily achieved with the Timer0 and we have various ways:

- a first way is to establish a TMR0 precharge value such as to produce overflow after 2ms. With a clock of 4MHz and a 1:16 prescaler, preloading **`TMR0`** with 131, we have overflow after 125 pulses, which gives exact 2ms. The program checks the overflow moment, which remains for a number of us equal to the pre-division value, and reloads **`TMR0`** with the calculated value.
- We can establish a predivisor such that the overflow falls every 2ms. At 4MHz clock we get this with the 1:8 prescaler, which gives 2,048ms. The timer is reset at the beginning of the cycle and then works in free-run; The program verifies the moment of the overflow, which remains for a number of US equal to the pre-division value.
- Let the timer run in free-run with a high-value prescaler: with 1:256, we have 256 clock pulses for each count. At 4MHz this is equivalent to 256us. Since counting is done in binary, a certain number of us have elapsed every time a bit changes state.

The first method is suitable if we want to achieve as high precision as possible, but in our case this is not necessary. The second method is a low prescaler, which leaves little time for overflow verification and directly determines a minimum time that may not be ideal for the rest of the application. The third method applies a high prescaler, which ensures that a condition is maintained for the maximum possible time, thus avoiding the risk of the program losing the moment of the event, while still allowing a wide range of times, verifying the content of the TMR0 register.
In this register, the count is in binary and the change of each bit is a function of the power of two it represents in the number.

Bit	7	6	5	4	3	2	1	0
Value	128	64	32	16	8	4	2	1
Pulses	32768	16384	8192	4096	2048	1024	512	256

So, for example, bit 0 alternates between the value 0 and 1 at every 256us; the two bits every 512us and so on. If

we are not interested in a specific combination, so we will just have to check the single bit In instructions: every time bit 3 changes state 2048us has elapsed, or about 2ms.

We render this in instructions:

```
; cycle display
; first Digit
displp  BTFSS     bittmr          ; 2.048ms ?
         Goto     displp          ; No - Waiting
        movf      pulscntr, w     ; Yes - Charge Meter
        andlw     0x0F            ; Low nibble only
        Call      segtbl          ; Lookup table
        movwf     Temp            ; Save to Temporary
        movwf     PORTC           ; Write on the port segm f:a
        BTFSC     temp,6          ; Command G-Segment
         Bsf      segmg
        BTFSS     temp,6
         Bcf      segmg
        UNITon                    ; Unit digit lit
dslp_1  BTFSC     bittmr          ; 2.048ms ?
         Goto     dslp_1          ; No - Waiting
dslp_2  UNIToff                   ; Yes -Turn off units
; second digit
; swap counter in W, low nibble<->high nibble
        swapf     pulscntr,w
        andlw     0x0f            ; Low nibble only
        Call      segtbl
        movwf     Temp
        movwf     PORTC
        BTFSC     temp,6
         Bsf      segmg
        BTFSS     temp,6
         Bcf      segmg
        DECon                     ; Tens Digit On
dslp_1 Call       Pbtest          ; Verify Contact
        BTFSS     bittmr          ; 2.048ms ?
         Goto     dslp_1          ; No - Waiting
        DECoff                    ; Yes - Turn off tens
        Goto      displp          ; Loop
```

Every 2ms, a digit is activated, presenting the value of the low nibble of the counter and then for another 2ms, on the other digit, the value of the high nibble. The exchange between the two displays takes place by controlling the gates of the relative MOSFETs, which ground the cathodes.

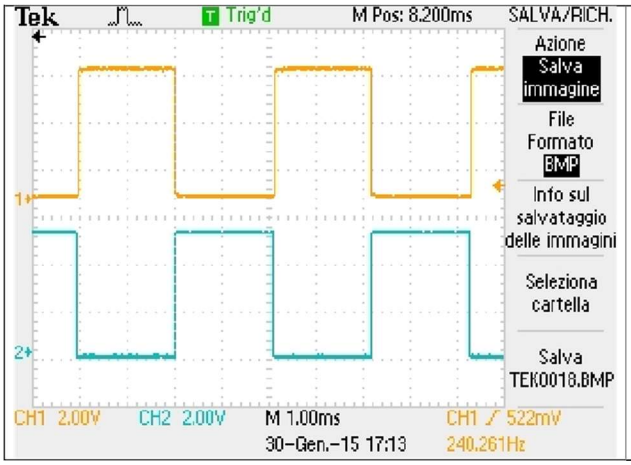
Four simple macros are used to control the MOSFET gates:

```
;#####################################################################
;                          LOCAL MACROS
; MOSFET gate command
UNITon   MACRO           ; Unit Digit
        bsf GATE2
          ENDM
UNIToff  MACRO
        bcf GATE2
          ENDM
```

The relationship between pin and gate is made symbolic and defined in such a way as to allow it to be quickly modified:

```
#define     PB      PORTB,0     ; PB4 contact
#define     LED1    PORTB,1     ; Debug LED
#define     GATE2   PORTB,2     ; Gate MOSFET Units
;#define             PORTB,3     ; MCLR
#define     GATE1   PORTB,4     ; Gate MOSFETs Tens
#define     segmg   PORTB,5     ; Segment G
```

With an oscilloscope, we can see the succession of on/off commands to the two MOSFETs:



these are square waves with a half-period of 2.048ms: the gate at the high level sends the MOSFET that connects the cathodes to the Vss in conduction, turning on the segments.

The low-level gate blocks conduction and the gate display is off.

When one MOSFET is conductive, the other is locked

The count is done **in hexadecimal**, from 00 to FF. The transition from the binary value of the pulse counter to the segments is done with a lookup table, as we have already seen in Tutorial 5A

```
; Segment data table - display Common cathode
segtbl andlw    0x0F            ; Low nibble only
        addwf   PCL,f           ; PC tip
        retlw   b'00111111'     ; "0" -|-|F|E|D|C|B|At
        retlw   b'00000110'     ; "1" -|-|-|-|-|C|B|-
        retlw   b'01011011'     ; "2" -|G|-|E|D|-|B|At
        retlw   b'01001111'     ; "3" -|G|-|-|D|C|B|At
......
```

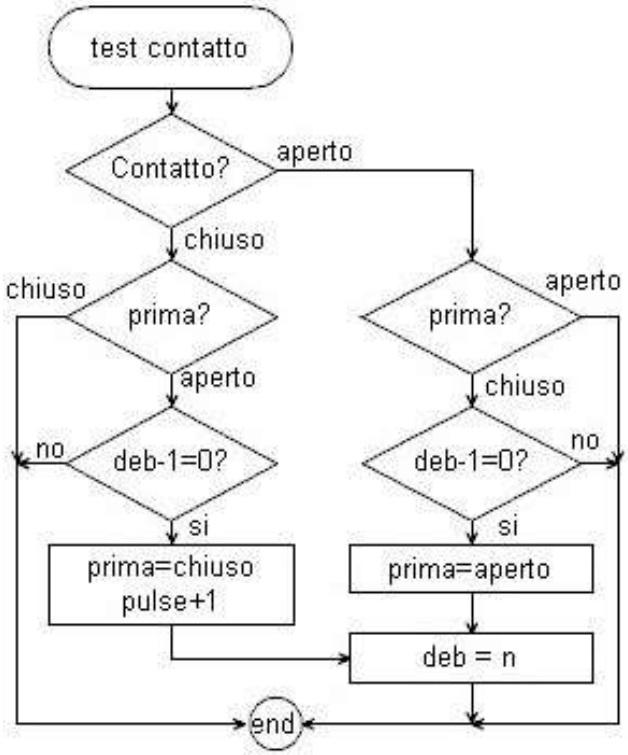
While waiting for the second display to end its power on, the contact status is checked, i.e. once every 4ms.
This means that input pulses of shorter duration will not be considered; In fact, since you also add the debounce time (in the example it is 8ms), the contact must change state more slowly than 20ms. If the incoming contact is free of bounces, such as that coming from an electronic or optical sensor (limit switch, position, proximity, magnetic, etc.) the debounce time could be eliminated; Here, however, we are using a mechanical button and therefore it is necessary to insert anti-bounce waits.

The main cadence also serves as a debounce:

```
; PBtest - Check Button Status
; Closed contact (=0) ?
PBtest  btfsc    PB
         goto    pbt_1          ; No - Verification for
        btfss    pbflg         Open
         goto    pb_e           ;      no - esc
; Yes - Used to be open - Check Debounce
        decfsz   debcntr,f      ; Debounce Counter-1=0?

         goto    pb_e           ; no - esc
        bcf      pbflg          ; yes - flag = close
        incf     Pulsecntr,f    ; contact push button +1
pbt_0   movlw    debtime        ; Debounce Counter Recharge
        movwf    debcntr
pb_e    retlw    0
; contatto aperto
pbt_1   btfsc    pbflg          ; previous closed?
         goto    pb_e           ; no - esc
; It used to be closed - check debounce
        decfsz   pbcntr,f       ; Debounce-1=0 counter?
         goto    pb_e           ; no - esc
        bsf      pbflg          ; yes - flag = open
        goto     pbt_0          ; set debounce counter ed esc
```

The comprehension of the contact algorithm is certainly not immediate, but if we trace the flowchart, the thing becomes different:



The mechanism is simple: every 8ms, in the waiting phase, the status of the contact is checked and compared with the previous one.

If it has changed, a wait constructed with a counter that resets after a certain number of steps (two in the example, equal to 16ms) is inserted into the contact test before confirming the status.

When the transition from high to low level (close) is confirmed, the input pulse counter is incremented.

Once again, we see how the flowchart is an indispensable tool not only for understanding, but also for writing instructions.

All constants and variables are symbolic, so they can only be changed at one point in the source:

```
;*****************************************************************
;                           CONSTANTS
;
debtime   EQU   2          ; Debounce time 2 x 4ms = 8ms

; TMR0 bits for  us
n #define bit1k   TMR0,2   ; 1,024ms
#define bit2k     TMR0,3   ; 2,048ms
#define bit4k     TMR0,4   ; 4,096ms
#define bit8k     TMR0,5   ; 8,192ms
#define bit16k    TMR0,6   ; 16,384ms

#define bittmr     bit2k   ; 2048us
```

Thus, if you want a longer debounce time, the change will be possible by acting on the definition of **debtime**, while wanting to change the cycle time of the multiplex, you will act on **bittmr**.
You can easily vary the multiplex time and see the effect on the displays when using a 4 or 8 or 16ms time.
The same goes for bits/ports and their functions.

The reset button resets the count to zero, restarting the program from the beginning.

Conclusions.

During the display refresh time (2ms) there is room to insert many hundreds of lines of code and thus perform mathematical operations or algorithms much more complex than what is needed to handle the button and counter.
Using a precise time cadence, it is also possible to implement RTCs, clocks, data loggers, serial transmissions, etc., in the same way. Even more "contemporary" actions are possible in the same program, even though the Baselines are devoid of interrupt handling.

It is just a matter of carefully studying the needs of the various tasks and how to "fit" them together in order to be able to execute them without loss of data or control over the process. Evidently, the structure has major limitations, due to the necessary reduction to synchronicity of tasks of an asynchronous nature, which is not always possible. It should also be considered that an increase in the clock reduces the execution time of the instructions and, for example, at 8MHz you can execute twice as many opcodes in the same time as at 4MHz. At 20MHz there is still a further increase. Therefore, by properly calibrating the various components of the program, even with simple Baselines it is possible to perform quite complex tasks, albeit with a fair amount of effort in the drafting of the source.

10A_mtask1.asm

```
;*****************************************************************
; 10A_mtask1.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly&C Course - Tutorial 10A
;                     Example of multitasking: Flashing LEDs at
;                     different cadences.
;                     PIC Baseline. Clock 4MHz.
;    PIC            : 16F505/526
;    Support        : MPASM
;    Version        : V.505-1.0
;    Date           : 01-05-2013
;    Hardware ref.  :
;    Author         :Afg
;
;-----------------------------------------------------------------
;
; Pin use :
;  ----------------
;       16F505/526 @ 14 pin
;
;                   |‾‾\/‾‾|
;             Vdd -|1   14|- Vss
;             RB5 -|2   13|- RB0
;             RB4 -|3   12|- RB1
;        RB3/MCLR -|4   11|- RB2
;             RC5 -|5   10|- RC0
;             RC4 -|6    9|- RC1
;             RC3 -|7    8|- RC2
;                  |______|
;
;    Vdd                      1: ++
;    RB5/OSC1/CLKIN           2: Out LED to Vss
;    RB4/OSC2/CLKOUT          3: Out LED to Vss
;    RB3/! MCLR/VPP           4: MCLR
;    RC5/T0CKI                5:
;    RC4/[C2OUT]              6:
;    RC3                      7:
;    RC2/[CVref]              8: Out LED to Vss
;    RC1/[C2IN-]              9: Out LED to Vss
;    RC0/[C2IN+]             10: Out LED to Vss
;    RB2/[C1OUT/AN2]         11: Out LED at Vss
;    RB1/[C1IN-/AN1]/ICSPC   12: Out LED at Vss
;    RB0/[C1IN+/AN0]/ICSPD   13: Out LED at Vss
;    Vss                     14: --
;
;    [] 16F526 only
;
; ******************************************************************
 #ifdef __16F505
       LIST      p=16F505          ; Processor Definition
       #include <p16F505.inc>
 #endif
 #ifdef __16F526
```

```
        LIST      p=16F526          ; Processor Definition
        #include <p16F526.inc>
 #endif
        Radix     DEC

; ###################################################################
;                           CONFIGURATION
; Internal Oscillator, 4MHz, No WDT, No CP, RB3=MCLR
 #ifdef __16F505
  __config _IntRC_OSC_RB4EN & _WDTE_OFF & _CP_OFF & _MCLRE_ON
 #endif
 #ifdef __16F526
  __config _IntRC_OSC_RB4 & _IOSCFS_4MHz & _WDTE_OFF & _CP_OFF &
_CPDF_OFF & _MCLRE_ON
 #endif

; ###################################################################
;                           CONSTANTS

taskn   Equ   .8             ; Total number of tasks
limith  Equ   .255             ; Ticks Counter Limit

; Limits of the count relative to individual
tasks limit0 equ .188
limit1 equ .77
limit2 equ .3
limit3 equ .20
limit4 equ .100
limit5 equ .45
limit6 equ .12
limit7 equ .125

; LED position in ports
LED0    Equ  1      ; PORTB,0
LED1    Equ  2      ; PORTB,1
LED2    Equ  4      ; PORTB,2
LED3    Equ  0x10   ; PORTB,4
LED4    Equ  0x20   ; PORTB,5
LED5    Equ  1      ; PORTC,0
LED6    Equ  2      ; PORTC,1
LED7    Equ  4      ; PORTC,2


; ###################################################################
;                             RAM
; general purpose RAM
        UDATA
taskcntr    Res 1     ; Task Counter
syscntr     Res 1     ; Ticks Counter
cntr0       Res 1     ; Number of ticks for task0
cntr1       Res 1     ; Number of ticks for task1
cntr2       Res 1     ; Number of ticks for task2
cntr3       Res 1     ; Number of ticks for task3
cntr4       Res 1     ; Number of ticks for task4
cntr5       Res 1     ; Number of ticks for task5
cntr6       Res 1     ; Number of ticks for task6
cntr7       Res 1     ; Number of ticks for task7
```

```
; ###################################################################
;                            LOCAL MACRO
; Compare files and literals and skip to destination if
several CJNEFLMACRO file, lit, dest
      movlw   Lit           ; w = lit
      subwf   file,w        ; w = w-file = lit-file
      BTFSS   STATUS,Z      ; if file = lit -> Z=0 -> exit
       goto   dest          ; if different goes to the
       right
         ENDM

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

; MOWF Internal Oscillator
        Calibration OSCCAL

 #ifdef __16F526
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the BCF digital function
                CM1CON0, C1ON
        Bcf     CM2CON0, C2ON
 #endif

; disable T0CKI from RC5
       movlw   b'11011111'
       OPTION

; Reset Initializations
       CLRF    PORTB            ; GPIO preset latch to
       0 clrf  PORTC
       movlw   0                ; all pins as TRIS
       outputs PORTC
       TRIS    PORTB

; Initialize Counters
       movlw 0xFF
       movwf taskcntr
       clrf   syscntr

; Main Loop
Main   incf    taskcntr,f     ; increment the counter of CFLJNE
       tasks taskcntr,taskn,jtable ; maximum?
       movlw 0                ; Yes - Restart from Task 0
       movwf taskcntr
       incf   syscntr,f       ; Increment Ticks Counter

jtable movf    taskcntr,w     ; jump table
       addwf PCL,f
       Goto    Task0          ; Jump to Goto
       Tasks   Task1
       Goto    Task2
       Goto    Task3
       Goto    Task4
       Goto    Task5
```

```
        Goto   Task6
        Goto   Task7

; executes Task0
; each limith ticks increments local counter
Task0 CFLJNE syscntr, limith, main
      Goto Main           ; If several go back to Main
      incf   cntrx,f      ; if not, increment local counter
;each limit0 increments, activates LED
      CFLJNE time0, limit0, main
      CLRF cntr0          ; If not, delete local meter
      Movlw LED0          ; LED drive
      xorwf PORTB,f
      Goto Main           ; Return to Loop


; executes Task1
Task1 CFLJNE syscntr, limith, main
        incf   cntr1,f
        CFLJNE cntr1, limit1, main
        clrf   cntr1
        movlw led1
        xorwf portb,f
        goto   Main

; executes Task2
Task2 CFLJNE syscntr, limith, main
        incf   cntr2,f
        CFLJNE cntr2, limit2, main
        clrf   cntr2
        movlw led2
        xorwf portb,f
        goto   Main

; runs Task3
Task3 CFLJNE syscntr, limith, main
        incf   cntr3,f
        CFLJNE cntr3, limit3, main
        clrf   cntr3
        movlw led3
        xorwf portb,f
        goto   Main

; executes Task4
Task4 CFLJNE syscntr, limith, main
        incf   cntr4,f
        CFLJNE cntr4, limit4, main
        clrf   cntr4
        movlw led4
        xorwf portb,f
        goto   Main

; runs Task5
Task5 CFLJNE syscntr, limith, main
        incf   cntr5,f
        CFLJNE cntr5, limit5, main
        clrf   cntr5
```

```
        movlw led5
        xorwf portc,f
        goto   Main

; runs Task6
Task6 CFLJNE syscntr, limith, main
        incf   cntr6,f
        CFLJNE cntr6, limit6, main
        clrf   cntr6
        movlw led6
        xorwf PORTC,f
        goto   Main

; runs Task7
Task7 CFLJNE syscntr, limith, main
        incf   cntr7,f
        CFLJNE cntr7, limit7, main
        clrf   cntr7
        movlw led7
        xorwf portc,f
        goto   Main

; ##########################################################################
;                              THE END
     END
```

10A_Aldb.asm

```
;*****************************************************************
; 10A_Aldb.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly&C Course - Tutorial 10A
;                     Multitask for buttons and LEDs with Timer0
;                     4 buttons switch as many LEDs.
;                     1 LED flashes at approx. 1 Hz
;    PIC            : 16F505/506/526
;    Support        : MPASM
;    Version       A: V.526-1.0
;    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: Out  LED3
;    RB3/! MCLR/VPP           4: MCLR
;    RC5/T0CKI                5:
;    RC4[/C2OUT]              6:
;    RC3                      7: In   PB5
;    RC2[/CVref]              8: In   PB3
;    RC1[/C2IN-]              9: In   PB2
;    RC0[/C2IN+]             10: In   PB1
;    RB2[/C1OUT/AN2]         11: Out  LED2
;    RB1[/C1IN-/AN1]/ICSPC   12: Out  LED1
;    RB0[/C1IN+/AN0]/ICSPD   13: Out  LED0
;    Vss                     14: --
;
;   []only 16F506/526
; ################################################################
; Choice of processor
  #ifdef  __16F526
        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
#define procanalog
  #endif
  #ifdef___16F505
; Internal oscillator, no WDT, no CP, MCLR
  __config _IntRC_OSC_RB4EN & _WDT_OFF & _CP_OFF & _MCLRE_ON
#define procanalog
  #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 last              ; Flags
      cntr5                  ; counter for task5
        ENDC

; Previous Status Flag for Input Pins #define
PIN4LST LAST,5            ; Flag Positions Last
#define Pin3LST last,4    ;
#define pin2lst last,3    ;
#define pin1lst last,2     ;

;*********************************************************************
;=====================================================================
;             DEFINITION OF PORT USE
;
;P ORTC map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|------- |
;| PB4 | PB3 | PB2 | PB1 | LED6|     |
;   1     1     1     1     0    0
pinport  Equ PORTC
LED6Port Equ PORTC
#define    pin4in  PORTC,5     ; button
#define    pin3in  PORTC,4     ; button
#define    pin2in  PORTC,3     ; button
#define    pin1in  PORTC,2     ; button
```

```
#define    LED6    PORTC,1    ; LED
;#define        PORTC,0    ;
; Position in Port Pin4
        equ 0x20 ; bit5
pin3    equ 0x10 ; bit4
pin2    EQ 8      ; bit3
pin1    EQ 4      ; bit2
LED6Pin EQqu 2    ; bit1

;P ORTB map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|--------|
;|     | LED3| MCLR|LED2 |LED1 |LED0 |
;   0     0     x    0     0      0
#define LED0 PORTB,0
#define LED1 PORTB,1
#define LED2 PORTB,2
#define LED3 PORTB,4
; for debugging
#define LED4 PORTB,5
#define LED5 PORTC,0
; ###########################################################################
;                                    CONSTANTS
; Task duration (approx. 16ms
  @4MHz) constant tasktime=0x3F
; debounce time (10ms)
  constant dbtime=0x27
; Task5 Duration - 8 x 64,512ms = 516ms
  Constant Task5Time=8
; ###########################################################################
;                          RESET ENTRY
;
; Reset Vector
        ORG     0x00

; MOWF Internal Oscillator
        Calibration OSCCAL

; ###########################################################################
;                         MAIN PROGRAM
Main:
; Reset Initializations
  #ifdef Procanalog
; Disable CLRF Analog Inputs
                ADCON0
; Disable comparators to free the digital MOVLW function
                0x77
        movwf   CM2CON0
        movwf   CM1CON0
 #endif

; disable T0CKI from RB5, 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

         CLRF    PORTB         ; CLRF Port Clear
         Latch   PORTC

; Port Direction Set
         movlw   b'00111100' ; RC5:2 in, RC1:0 out
         tris    PORTC
         movlw   b'000000000' ; RB all out
         three of a kind  PORTB

         movlw   0xff          ; previous flags=1
         movwf   Last
         movlw   task5time     ; Counter Preset for Task5
         MovWF   cntr5
         CLRF    TMR0          ; Start Time Count

Taskloop:                      ; Start of tl_0 task
loops  movf    TMR0,w          ; starting from 0
       skpz
        Goto   tl_0

; Task1
; test per pin to 0
Task1  BTFSC   pin1in          ; pin = 0 ?
        goto   t1_2            ; No - Other Test
       btfss   pin1lst         ; Yes - Previous = 1
        goto   t1_end          ; No - End of Task
t1_0   movlw   dbtime          ; yes - debounce
       xorwf   TMR0,w          ; end debounce?
       Skpz
        Goto   t1_0            ; No - Waiting
       BTFSC   pin1in          ; Yes - Pin always at
        GOTO   t1_end          ; 0? No - End of Task
       bcf     pin1lst         ; yes - previous flag = 0
       bsf     LED0            ; lights LED
t1_end movlw   tasktime        ; End of time task?
       xorwf   TMR0,w
       skpz
        Goto   t1_end          ; No - Waiting
       Goto    Task2           ; Yes - Next Task
; test per pin to 1
t1_2   BTFSC   pin1lst         ; previous = 0
        Goto   t1_end          ; No - End of Task
t1_3   movlw   dbtime          ; Yes  Debounce
       xorwf   TMR0,w          ; -    Debounce
       skpz                      End
        Goto   t1_3            ;      waiting
       BTFSS   pin1in          ; no-  Pin always at 1?
        Goto   t1_end          ; Yes- End of Task
       Bsf     pin1lst         ; no-  Set Flag
                                 Yes-
       Bcf     LED0            ; turns off LEDs
       Goto    t1_end          ; End of action
```

```
; Task2
Task2 btfsc    pin2in
        Goto   t2_2
       BTFSS   pin2lst
        Goto   t2_end
t2_0   movlw   (tasktime+dbtime)
       subwf   TMR0,w
       skpc
        Goto   t2_0
       BTFSC   pin2in
        Goto   t2_end
       Bcf     pin2lst
       Bsf     LED1
t2_end movlw   (tasktime*2)
       subwf   TMR0,w
       skpc
        Goto   t2_end
       Goto    Task3
; 1-t2_2 Pin Test
       BTFSC   pin2lst
        Goto   t2_end
t2_3   movlw   (tasktime+dbtime)
       subwf   TMR0,w
       skpc
        Goto   t2_3
       BTFSS   pin2in
        Goto   t2_end
       Bsf     pin2lst
       Bcf     LED1
       Goto    t2_end

; Task3
Task3 btfsc    pin3in
        Goto   t3_2
       BTFSS   pin3lst
        Goto   t3_end
t3_0   movlw   ((tasktime*2)+dbtime)
       subwf   TMR0,w
       skpc
        Goto   t3_0
       BTFSC   pin3in
        Goto   t3_end
       Bcf     pin3lst
       Bsf     LED2
t3_end movlw   (tasktime*3)
       subwf   TMR0,w
       skpc
        Goto   t3_end
       Goto    Task4
; 1-t3_2 Pin Test
       BTFSC   pin3lst
        Goto   t3_end
t3_3   movlw   ((tasktime*2)+dbtime)
       subwf   TMR0,w
       skpc
        Goto   t3_3
       BTFSS   pin3in
```

```
         Goto    t3_end
        Bsf     pin3lst
        Bcf     LED2
        Goto    t3_end

; Task4
Task4 btfsc    pin4in
         Goto    t4_2
        BTFSS   pin4lst
         Goto    t4_end
t4_0    movlw   ((tasktime*3)+dbtime)
        subwf   TMR0,w
        skpc
         Goto    t4_0
        BTFSC   pin4in
         Goto    t4_end
        Bcf     pin4lst
        Bsf     LED3
t4_end movlw   (tasktime*4)
        subwf   TMR0,w
        skpc
         Goto    t4_end
        Goto    Task5
; 1-t4_2 Pin Test
        BTFSC   pin4lst
         Goto    t4_end
t4_3    movlw   ((tasktime*3)+dbtime)
        subwf   TMR0,w
        skpc
         Goto    t4_3
        BTFSS   pin4in
         Goto    t4_end
        Bsf     pin4lst
        Bcf     LED3
        Goto    t4_end

; Task5 - flashes LED5
task5 decfsz cntr5,f          ; 500ms ?
         Goto    Taskloop       ; No - Other Loop
; every 500ms or so
t5_0    movlw   task5time      ; MovWF Meter
        Recharge               cntr5
        movlw   LED6PIN        ; Toggle LED
        XORWF   LED6port,F
        Goto    Taskloop       ; Loop

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

10A_526.asm

```
;*****************************************************************
; 10A_526.asm
;-----------------------------------------------------------------
;
;    Title          : Assembly & C Course - Tutorial 10A_526
;                     Counter with output on display 2 displays at 7
;                     multiplexed segments.
;                     Counting input by button.
;    PIC            : 16F526 or 16F505
;    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   segment  Dp
;    RB4/OSC2/CLKOUT           3: Out   segment  g
;    RB3/! MCLR/VPP            4: MCLR
;    RC5/T0CKI                 5:
;    RC4[/C2OUT]               6:
;    RC3                       7:
;    RC2[/CVref]               8: Out   segment  f
;    RC1[/C2IN-]               9: Out   segment  an
                                                 d
;    RC0[/C2IN+]              10: Out   segment  d
;    RB2[/C1OUT/AN2]          11: Out   segment  c
;    RB1[/C1IN-/AN1]/ICSPC    12: Out   segment  b
;    RB0[/C1IN+/AN0]/ICSPD    13: Out   segment  at
;    Vss                      14: --
;
;   []only 16F506/526
; ################################################################
; Choice of processor
  #ifdef  __16F526
          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
#define procanalog
  #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
#define procanalog
  #endif

; ************************************************************************
;                           RAM
; general purpose RAM
        CBLOCK 0x10          ; start area RAM
      debcntr, pulscntr      ; counters
      Temp                   ; temporary
      Flags                  ; ENDC input status
        flags

#define PBFLG Flag,7         ; Input Status Flag Bit

 ;*****************************************************************
;=================================================================
;            DEFINITION OF PORT USE
;
;P ORTC map
;| 5 | 4 | 3 | 2 | 1 | 0 |
;|-----|-----|-----|-----|-----|---------|
;| f | and | d | c | b | a |
;
#define     segma PORTC,0      ; Segment A
#define     segmb PORTC,1      ; Segment B
#define     segmc PORTC,2      ; Segment C
#define     segmd PORTC,3      ; Segment D
#define     segme PORTC,4      ; Segment E
#define     segmf PORTC,5      ; Segment F

;P ORTB map
;| 5 | 4 | 3 | 2 | 1 | 0 |
```

```
;|-----|-----|-----|-----|-----|---------|
;| dp | g |        |     |     |     |
;
#define     PB     PORTB,0     ; PB4 contact
#define     LED1   PORTB,1     ; Debug LED #define
            GATE2  PORTB,2     ; Gate MOSFET Units
;#define           PORTB,3     ; MCLR
#define     GATE1  PORTB,4     ; Dozens of MOSFET
gates #define                  segmg PORTB,5    ;
Segment G
;
;*****************************************************************
;
;    Notes:
;
; ###############################################################
;                          CONSTANTS
;
debtime EQU 2            ; Debounce time 2 x 4ms = 8ms
#define bit1k    TMR0,1 ; TMR0 bit for n us
#define bit2k    TMR0.2
#define bit4k    TMR0,3

#define bittmr   bit4k ; 2048us

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

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

 ; MOWF Internal Oscillator
        Calibration OSCCAL
        Goto    Main

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

; Segment Data Table - Display Common Cathode
SEGTBL ANDLW 0x0F            ; Low nibble only
        addwf PCL,f          ; PC tip
        retlw b'00111111'    ; "0" -|-|F|E|D|C|B|At
```

```
        retlw  b'00000110'    ; "1"  -|-|-|-|-|C|B|-
        retlw  b'01011011'    ; "2"      -|G|-|E|D|-
                                              |B|At
        retlw  b'01001111'    ; "3"          -|G|-|-
                                           |D|C|B|At
        retlw  b'01100110'    ; "4"  -|G|F|-|-|C|B|-
        retlw  b'01101101'    ; "5"    -|G|F|-|D|C|-
                                                |At
        retlw  b'01111101'    ; "6"    -|G|F|E|D|C|-
                                                |At
        retlw  b'00000111'    ; "7"        -|-|-|-|-
                                             |C|B|At
        retlw  b'01111111'    ; "8"                -
                                        |G|F|E|D|C|B|At
        retlw  b'01101111'    ; "9"          -|G|F|-
                                           |D|C|B|At
        retlw  b'01110111'    ; "A"        -|G|F|E|-
                                             |C|B|At
        retlw  b'01111100'    ; "b"  -|G|F|E|D|C|-|-
        retlw  b'00111001'    ; "C"    -|-|F|E|D|-|-
                                                |At
        retlw  b'01011110'    ; "D"  -|G|-|E|D|C|B|-
        retlw  b'01111001'    ; "An    -|G|F|E|D|-|-
                                d"              |At
        retlw  b'01110001'    ; "F"    -|G|F|E|-|-|-
                                                |At

; PBtest - Check the status of the
PBtest button:
 ; contact closed ?
        BTFSC    PB
         Goto    pbt_1        ; No - Go to Open Contact
        BTFSS    PBFLG        ; Yes - Previous open?
         Goto    pb_e         ;      No - Closed - Exit
; Yes - Used to be open - Check Debounce
        decfsz   debcntr,f    ; Debounce-1=0 counter?
         Goto    pb_e         ; No - Exit
        BCF      PBFLG        ; Yes - Flag = Closed
        Incf     pulscntr,f   ; Pulse Counter+1 Meter
pbt_0   Movlw    debtime      ; Charging
        MovWF    debcntr
pb_e    retlw    0
; Contact is open
pbt_1   BTFSC    PBFLG        ; previous closed?
         Goto    pb_e         ; No - Exit
    ; Yes - before it was closed - check
                 debounce
         decfsz   debcntr,f    ; Debounce-1=0 counter?
         Goto    pb_e         ; No - Exit
        Bsf      PBFLG        ; Yes - Flag = Open
        Goto     pbt_0        ; Set Debounce Counter and
                              Exit

; ###########################################################################
;                          MAIN PROGRAM
Main:
; Reset Initializations
  #ifdef Procanalog
; Disable CLRF Analog Inputs
               ADCON0
```

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

; Turn off   T0CKI from RB5
          ;       b'11010111'
          ;         1-------         GPWU Disabled
          ;         -1------         GPPU disabled
```

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

         movlw   b'000000000' ; RC All Out
         Tic-Tac-Toe PORTC
         movlw   b'00000001' ; RB0 in, other
         outs three of a kind   PORTB

         CLRF    PORTB        ; Clear Latch of CLRF
         Ports   PORTC
         movlw   0xFF
         movwf   Flags        ; flag as open (=1) movlw
                 debtime      ; Preset Debounce Counter
         Movwf   Debcntr
         CLRF    Pulscntr     ; Clear Pulse Counter

         CLRF    TMR0         ; Initialize Timer

; Display Cycle
; Digit Unit
displp btfss    bittmr       ; 2.048ms ?
          Goto   displp       ; No -
          Waiting
         movf    pulscntr,w ; Yes - Charge Counter
         ANDLW   0x0F         ; Nibble only low call
                 segtbl       ; Lookup table
         movwf   Temp         ; Save to Temporary Movwf
                 PORTC        ; Write on the port segm
         f:a btfsc           temp,6   ; Command G-
         Segment
          Bsf    segmg
         BTFSS   temp,6
          Bcf    segmg
         UNITon               ; Unit digit lit
dslp_1 BTFSC    bittmr       ; 2.048ms ?
          Goto   dslp_1       ; no - waiting
dslp_2 UNIToff               ; Yes - Turn off
units
; digit tens
; swap counter in W, low nibble<->high swapf
                  pulscntr,w
         andlw    0x0f        ; Nibble only low
         call     segtbl
         movwf    Temp
         movwf    PORTC
         BTFSC    temp,6
          Bsf      segmg
         BTFSS    temp,6
          Bcf      segmg
         DECon                ; Turn on dozens
dslp_3 calls     PBtest       ; Verify Contact
         BTFSS   bittmr       ; 2.048ms ?
          Goto    dslp_3      ; No -
          Waiting
```

```
DECoff              ; Yes - Turn off
Dozens of Goto      displp   ; Loop
```

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