

# LABORATORIO N°1

## COMPILACION, SIMULACION Y CREACION DE APLICACIONES BÁSICAS CON MIROCONTROLADORES.

### I. OBJETIVOS

- ✓ Usar la aplicación de Microchip MPLAB en la creación, compilación, simulación y programación de micro controladores.
- ✓ Usar la aplicación PROTEUS como herramienta de depuración y prueba de código realizado en la aplicación MPLAB.
- ✓ Implementar aplicaciones sencillas y verificar su funcionamiento en condiciones reales.

### II. MATERIAL DE APOYO

- Ordenador con las aplicaciones PROTEUS Y MPLAB X.
- Microcontrolador de la serie PIC16F88X.
- Programador de microcontroladores PIC.
- Protoboard
- Resistencias, Diodos, Switch, pulsadores, etc.
- Fuente de alimentación 5V

### III. EQUIPOS NECESARIOS.

- PC y Software de simulación.

### IV. INTRODUCCIÓN

Parte del desarrollo de aplicaciones con micro controladores son de vital importancia las herramientas que permiten llevar a cabo esta labor, el MPLAB y el PROTEUS, son solo una

---

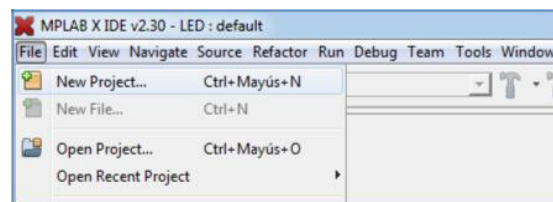
Pequeña muestra de las aplicaciones desarrolladas con este fin. Con el uso de las mismas se puede realizar las operaciones de creación, depuración y terminado final de sistemas de control completo realizadas alrededor del uso de estos versátiles dispositivos.

Con el desarrollo de esta práctica se quiere que el estudiante se familiarice con el entorno de trabajo en el que se realizaran las diversas aplicaciones, haciendo programas sencillos que permitan asimilar el uso de herramientas de uso común en este tipo de labores.

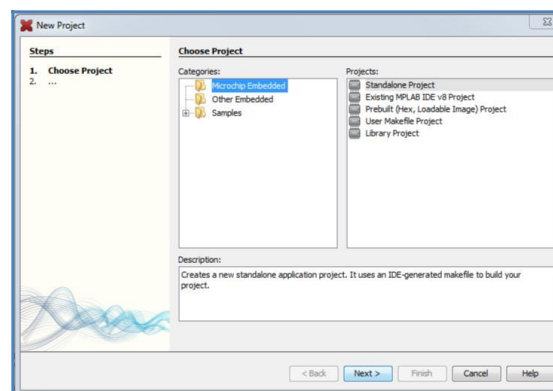
### V. PRIMER EJERCICIO

#### 1) Creación de un proyecto

Para comenzar un nuevo proyecto en MPLAB hacemos click en *File/New Project*.



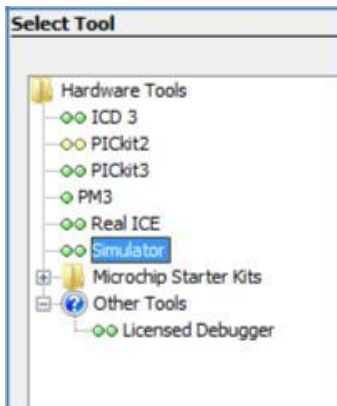
En *Categories* se selecciona *Microchip Embedded*. Click en *Next*.



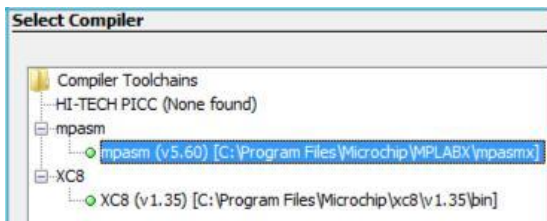
A continuación en *Device* se busca la referencia del microcontrolador a trabajar, en este caso PIC16F887. Click en *Next*.



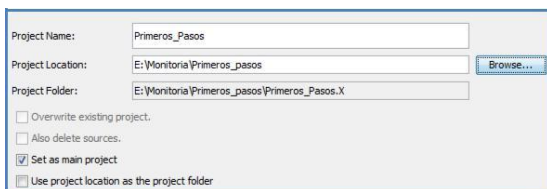
En *Select Tool / Hardware Tools* se selecciona *Simulator*. Click en *Next*.



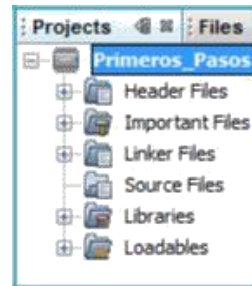
En *Select Compiler* se selecciona el compilador adecuado para ensamblador. Click en *Next*.



Y para finalizar la creación del nuevo proyecto se nombra y se da click en *Finish*.

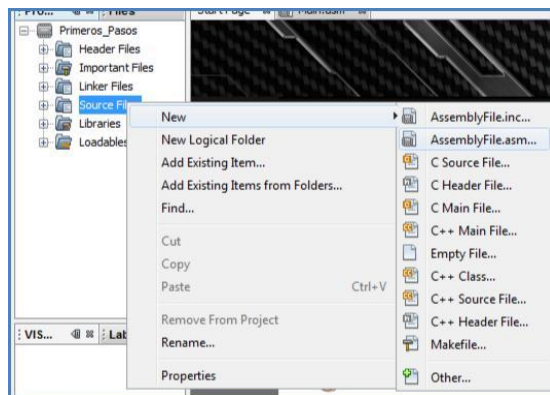


A continuación se podrá ver en la parte izquierda de la ventana del MPLAB el proyecto creado.

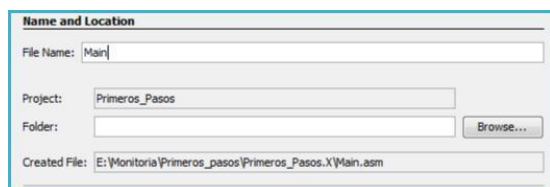


## 2) Configuración básica

Para crear el archivo principal del programa se da click derecho en *Source Files* y se selecciona *New/AssemblyFile.asm...*



A continuación se nombra el archivo y se da click en *Finish*.



La primera línea de código define con que pic se va a trabajar, en esta se enlista el pic.



El paso a seguir es darles lugar y a los registros con los que se trabajara en el programa a desarrollar. En este ejemplo manejaremos los registros *CONFIG1*, *CONFIG2*, *STATUS*, *TRISA*, *PORTA*. Esta inclusión de los registros se da de la siguiente manera:

En el capítulo de *Memory Organization* del datasheet podemos encontrar la dirección de todos los registros. Usando la sentencia *EQU* le asignamos a cada registro su respectiva dirección.

2	<i>_CONFIG1</i>	<i>EQU</i>	<i>H'2007'</i>
3	<i>_CONFIG2</i>	<i>EQU</i>	<i>H'2008'</i>
4	<i>STATUS</i>	<i>EQU</i>	<i>H'0003'</i>
5	<i>TRISA</i>	<i>EQU</i>	<i>H'0085'</i>
6	<i>ANSEL</i>	<i>EQU</i>	<i>H'0188'</i>
7	<i>ANSELH</i>	<i>EQU</i>	<i>H'0189'</i>
8	<i>PORTA</i>	<i>EQU</i>	<i>H'0005'</i>
9	<i>RA0</i>	<i>EQU</i>	<i>H'0000'</i>
10	<i>RP0</i>	<i>EQU</i>	<i>H'0005'</i>
11	<i>RP1</i>	<i>EQU</i>	<i>H'0006'</i>

Se da lugar también a *RA0* que es el bit que se maneja en el *PORTA* y a *RP0* y *RP1* los cuales determinan en que banco se está trabajando.

Para la configuración de bits para el pic es necesario ir al datasheet de éste. En el capítulo llamado *SPECIAL FEATURES OF THE CPU* se encuentra la descripción de los diferentes registros de configuración (*CONFIG1* – *CONFIG2*).

Para el registro *CONFIG1* los bits a modificar serán desde el bit 0 hasta el bit 12 y el resto de bits se dejan en 1:

*FOSC* define el tipo y la velocidad del oscilador que se usará. En este caso será *INTRC\_NOCLKOUT*.

*WDTE* define el estado del watchdog timer. En este caso será OFF.

*PWRTE* define el estado de power-up timer. En este caso será OFF.

*MCLRE* define el estado del master clear. En este caso será ON.

*CP* define el estado de seguridad del código del programa. En este caso será OFF.

*CPD* define el estado de seguridad de la memoria. En este caso será OFF.

*BOREN* define el estado de brown out reset. En este caso será ON.

*IESO* define el estado de switcheo interno/externo. En este caso será ON.

*FCMEN* define el estado de guardacarga del reloj. En este caso será OFF.

*LVP* define el estado de programación a bajo voltaje. En este caso será OFF.

Configurando de esta manera los bits de *CONFIG1* el dato que será cargado es igual a *0XE3F4*

Para el registro *CONFIG2* los bits a modificar serán desde el bit 8 al bit 10 y el resto se deja en 1:

*BOR4V* define el rango del Brown-out reset. En este caso será *BOR21V*.

*WRT* define el estado de la protección de escritura sobre la memoria. En este caso será OFF.

Configurando de esta manera los bits de *CONFIG2* el dato que será cargado es igual a *0XFEFF*

```

;Configuracion de bits
    _CONFIG _CONFIG1, 0XE3F4
    _CONFIG _CONFIG2, 0XFEFF

```

Para los registros de propósito general (variables) se deben definir a partir de la posición *0X20* (notación hexadecimal). Se hace uso de las palabras *CBLOCK* y *ENDC* las cuales ayudan a la definición de variables.

```

10
11    ;Definicion de variables
12    CBLOCK 0X20
13        ;Variable(s) a definir
14    ENDC
15

```

Y por último se define el inicio del programa el cual será a partir de la posición *0X00*.

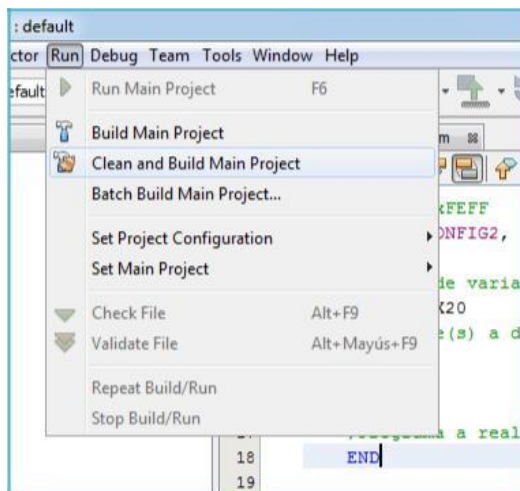
```

14    ENDC
15
16    ORG .0
17        ;Programa a realizar
18    END
19

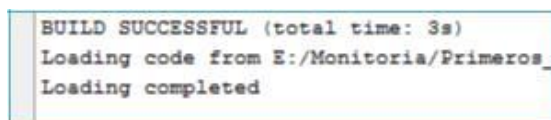
```

Para verificar que todo ha salido bien hasta acá sería conveniente que se compilara y corriera lo

que se lleva de programa. Para esto se accede a *Run/ Clean and Build Main Project*



Si todo está bien el programa debe correr sin ningún error.



### 3) Primeros ejercicios

#### Blinking led

Primero se debe configurar el puerto que va a ser usado, en este caso será el *PORTA* y su configuración será por medio de los registros *TRISA* y *ANSEL*. Como va a ser un dato de salida digital por *RA0* se debe cargar el dato *0X00* en los registros *TRISA* y *ANSEL*.

**Nota:** El registro *TRIS* define el estado de salida o entrada de los *PORT*. Si el bit del registro *TRIS* esta en 1 significa que será de tipo entrada y si es 0 significa que será de tipo salida. El registro *ANSEL* define si *PORTA* o *PORTE* son entradas de tipo analógico o digital, si son entradas analógicas *ANSEL* será puesto en 1 y si son entradas digitales *ANSEL* será puesto en 0.

Como el registro *TRISA* esta en el banco 1 es necesario posicionar el puntero allí y para esto modificamos los 2 bits *RP0* y *RP1* del registro *STATUS*.

**RP<1:0>:** Register Bank Select bits (used for direct addressing)  
 00 = Bank 0 (00h-7Fh)  
 01 = Bank 1 (80h-FFh)  
 10 = Bank 2 (100h-17Fh)  
 11 = Bank 3 (180h-1FFh)

Como vamos a trabajar solamente con *RA0* solo nos importara el valor del bit menos significativo de *TRISA* y *ANSEL*

```
16      ORG .0
17      BSF     STATUS,RP0 ;
18      BCF     STATUS,RP1 ; Bank1
19      BCF     TRISA,0     ;RA0 como salida
20      END
```

Para cambiar al banco donde se encuentra el registro *ANSEL* también se puede usar el comando *BANKSEL*.

```
16      ORG .0
17      BSF     STATUS,RP0 ;
18      BCF     STATUS,RP1 ; Bank1
19      BCF     TRISA,0     ; RA0 como salida
20      BANKSEL ANSEL      ; Bank3
21      CLRF    ANSEL      ; Ansel clareado
22      BANKSEL PORTA     ; Bank0
23      END
```

*CLRF* es una instrucción que sirve para dejar todos los bits de un registro en 0.

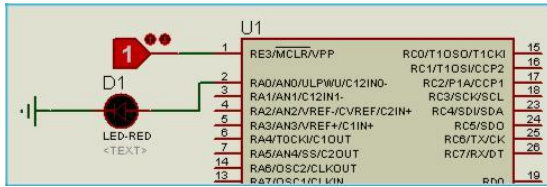
Al terminar la configuración de puertos nos devolvemos al banco 0 y procedemos a empezar el programa.

Como ya tenemos la configuración de los puertos no necesitamos volver a ese código en un futuro por esto es necesario crear una etiqueta la cual nos marcará el punto de retorno. Las etiquetas pueden tener cualquier nombre, en este caso se llamará *LOOP*.

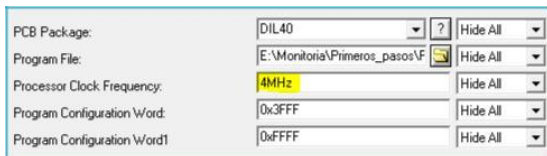
Para poner el bit *RA0* en 1 se utiliza la instrucción *BSF* (*Bit Set f*) y con la instrucción *GOTO* nos devolvemos a la posición de la etiqueta.

```
22      BANKSEL PORTA     ; Bank0
23      LOOP:
24      BSF     PORTA,RA0  ;RA0=1
25      GOTO    LOOP      ;Puntero en loop
26      END
```

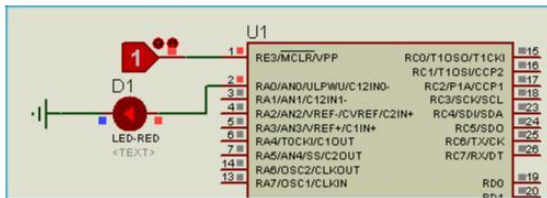
Para verificar el funcionamiento compilamos. Y realizamos un montaje sencillo en *PROTEUS*.



Para cargar el proyecto en el pic de *PROTEUS* se le da doble click a éste, en *Program File* se busca la carpeta donde se guardo el proyecto, en este caso la dirección será *Primeros\_pasos/Primeros\_pasos.X/ dist/ default/ production/ Primeros\_Pasos.X.production.cof*. Se carga el archivo .cof dado que este se puede ejecutar paso a paso y de esta manera trabajar más cómodamente.



Al correr la simulación se debe ver el led encendido solamente. Es necesario activar el master clear (PIN1) de lo contrario el pic no estará activo.



Para hacer que el led parpadee es necesario crear un retardo por software, para esto hay que tener en cuenta el tiempo que tarda una instrucción ( $T_{cy}$ ).

$$T_{cy} = \frac{4}{f_{osc}}$$

Donde  $f_{osc}$  es la frecuencia de oscilador configurada previamente; si  $f_{osc}$  no ha sido definida por el programador ésta tendrá un valor de 4Mhz por defecto.

Ejemplo de retardo por software:

Primero se creara una variable CONT1.

```
11 ;Definicion de variables
12 CBLOCK 0X20
13     CONT1
14 ENDC
```

Después se le carga un valor X, en este caso será de 100

```
32     MOVLW    .100
33     MOVWF    CONT1
```

Haciendo uso de la instrucción *DECFSZ* (Decrementar f, Salto si 0) se comienza a decrementar el valor de CONT1. Siempre y cuando el valor no sea cero el apuntador se devolverá una posición (*GOTO \$-1*).

```
34     DECFSZ   CONT1
35     GOTO     $-1
```

Teniendo en cuenta cuantos ciclos de instrucción tarda desde que  $CONT1 = X$  hasta que  $CONT1 = 0$  se puede estimar el tiempo de retardo.

```
32     MOVLW    .100           ;1 Tcy
33     MOVWF    CONT1         ;1 Tcy
34     DECFSZ   CONT1         ;100 * 1 Tcy
35     GOTO     $-1           ;100 * 2 Tcy
```

El tiempo de retardo para este ejemplo será:

$$1T = \frac{4}{f_{osc}} = \frac{4}{4 \times 10^6} = 1 \times 10^{-6} \text{ seg}$$

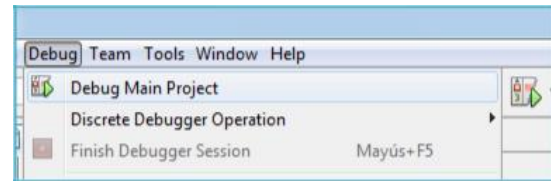
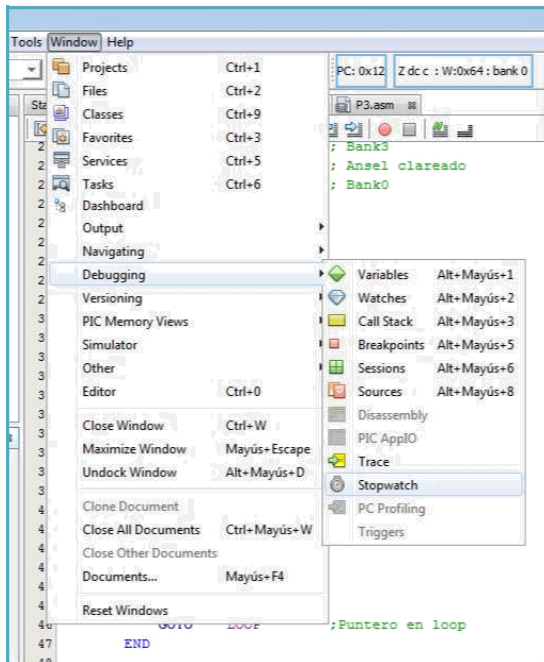
$$T = T_{cy} (2 + 100 (3))$$

$$T = 1 \times 10^{-6} (2 + 100 (3))$$

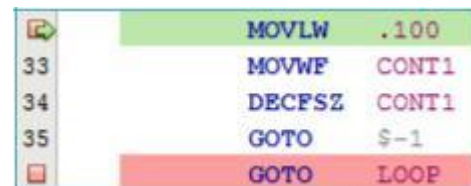
$$T = 302 \times 10^{-6} \text{ seg}$$

Para confirmar que el tiempo de retardo usamos la opción *Stopwatch* la cual se encuentra en *Window/ Debugging/*

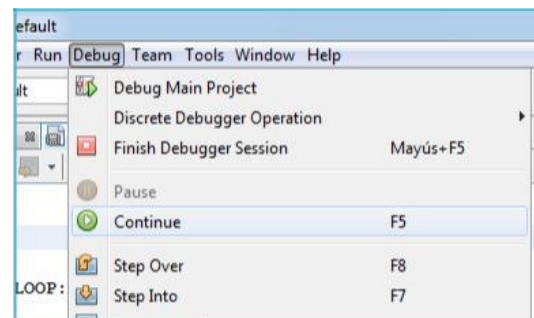




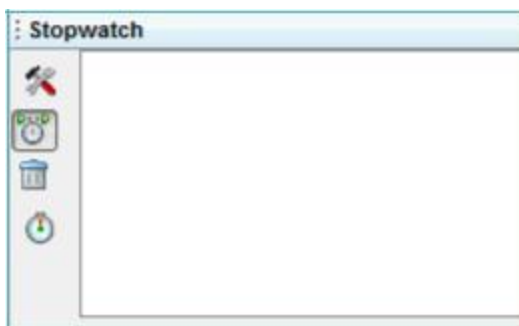
El programa se detendrá en la primera línea resaltada en rojo ahora verde.



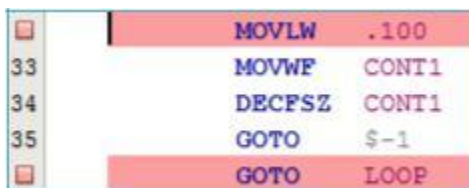
Enseguida se le da click a la opción de *Continue* la cual se encuentra en *Debug*/



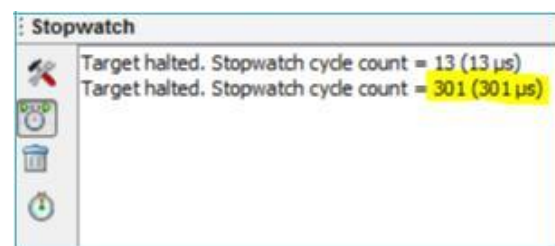
Aparecerá algo así.



Enseguida damos un simple click en los números correspondientes al inicio y fin del retardo recién creado, como se ve a continuación.



De inmediato en la ventana de *Stopwatch* se podrá observar el tiempo transcurrido de limite a limite el cual debe ser de  $T \approx 302 \times 10^{-6} \text{ seg}$



Para observar el tiempo que tarda el programa desde  $\text{CONT1} = 100$  hasta que  $\text{CONT1} = 0$  usamos la opción *Debug Main Project* la cual se encuentra en *Debug*/

Cuando  $\text{CONT1} = 255$  será el máximo tiempo de retardo.  $T_{\text{max}} = 767 \times 10^{-6} \text{ seg}$

Nota: Para crear un retardo por software más grande se debe aumentar el número de variables contadoras.

```

MOVLW    X           ;1 Tcy
MOVWF    CONT1        ;1 Tcy
MOVLW    Y           ;X * 1 Tcy
MOVWF    CONT2        ;X * 1 Tcy
DECFSZ   CONT2        ;X * Y * 1 Tcy
GOTO     $-1          ;X * Y * 2 Tcy
DECFSZ   CONT1        ;X * 1 Tcy
GOTO     $-5          ;X * 2 Tcy

```

Ya con el retardo por software podemos hacer que el led parpadee añadiendo al programa unas líneas de código que cambien el estado de RA0. Si RA0 es igual a 0 entonces que se vuelva 1 y viceversa.

```

BTFSS    PORTA,RA0
GOTO     $+3
BCF      PORTA,RA0
GOTO     $+2
BSF      PORTA,RA0

```

De esta manera el programa completo será:

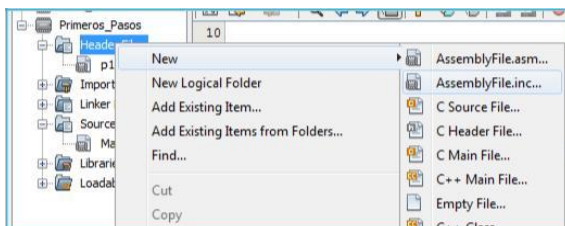
```

25      LOOP:
26          BTFSS    PORTA,RA0
27          GOTO     $+3
28          BCF      PORTA,RA0
29          GOTO     $+2
30          BSF      PORTA,RA0
31
32          MOVLW    .100
33          MOVWF    CONT1
34          DECFSZ   CONT1
35          GOTO     $-1
36          GOTO     LOOP

```

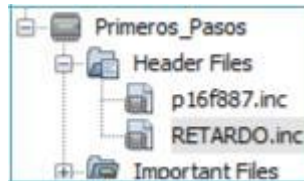
Solo queda compilar y simular para verificar el funcionamiento.

Para dejar el retardo como una rutina primero se crea un archivo .inc en la carpeta de *Header Files*.



Se le nombra a este archivo RETARDO.

File Name: RETARDO



En este archivo se debe darle un nombre a la rutina, de preferencia el mismo nombre (RETARDO).

```

1      RETARDO:
2
3      RETURN

```

Y en este se pega el código correspondiente al retardo creado por software.

```

1      RETARDO:
2          MOVLW    X
3          MOVWF    CONT1
4          MOVLW    Y
5          MOVWF    CONT2
6          DECFSZ   CONT2
7          GOTO     $-1
8          DECFSZ   CONT1
9          GOTO     $-5
10     RETURN

```

Para incluir la rutina en el programa principal es necesario escribir #INCLUDE "RETARDO.INC" antes de END en el archivo Main.asm

```

#include "RETARDO.INC"
END

```

Y en el programa principal en donde se encontraba el código del retardo se reemplaza con CALL RETARDO

30	LOOP:		
31	BTFSS	PORTA, RAO	
32	GOTO	\$+3	
33	BCF	PORTA, RAO	
34	GOTO	\$+2	
35	BSF	PORTA, RAO	
36			
37	CALL	RETARDO	
38			
39	GOTO	LOOP	

[6]. Designing Embedded Systems with PIC Microcontrollers Principles and applications. Tim Wilmshurst. 2007. Elsevier

De esta manera el programa se da por terminado.

#### IV. DESCRIPCIÓN DEL LABORATORIO.

Como actividad de este laboratorio se plante el diseño de un sistema de digital que realice operaciones binarias a 24 bits. Con las operaciones el sistema debe estar en capacidad de resolver operaciones matemáticas como la siguiente.

$$(25_{10} + 15_{10}) \times (625_{10} - 250_{10}) + 191_{10}$$

También debe realizar conteo, incremento, decremento, rotación a izquierda o derecha, negación, AND, OR Y XOR. La ecuación es un ejemplo de las ecuaciones que se pueden resolver usando 24 bits. Cada grupo debe plantear su propia ecuación y resolverla usando el pic16f887.

#### VII. BIBLIOGRAFIA

- [1]. Microcontroladores dsPIC Diseño práctico de aplicaciones. Tercera Edición. J. M<sup>a</sup>. Angulo Usategui y I. Angulo Martínez. Editorial McGraw Hill, 2007
- [2]. Microcontroladores PIC Diseño práctico de aplicaciones. Tercera Edición. J. M<sup>a</sup>. Angulo Usategui y I. Angulo Martínez. Editorial McGraw Hill, 1999
- [3]. Microcontroladores PIC. La clave del diseño. E. Martín Cuenca , J. M<sup>a</sup>. Angulo Usategui y I. Angulo Martínez. Editorial Thomson
- [4]. Microcontroladores PIC, la solución en un chip J. M<sup>a</sup>. Angulo Usategui, E. Martín Cuenca y I. Angulo Martínez. Editorial Paraninfo, 2000
- [5]. Microcontrolador PIC16F84. Desarrollo de proyectos. PALACIOS, E.- REMIRO, F. y LÓPEZ, L.J. Febrero 2004. Rústica y CD-ROM, 648 Págs..