SISTEMAS ELECTRÓNICOS Y AUTOMÁTICOS
PRACTICAS DE MICROCONTROLADORES PIC

PRÁCTICA 3:

Bucles

- ***Bucles:***
  - ***Bucles infinitos***
  - ***Bucles finitos***
  - ***Bucles anidados***
- ***Medir tiempos con MPLAB***
- ***Ejemplos y ejercicios***

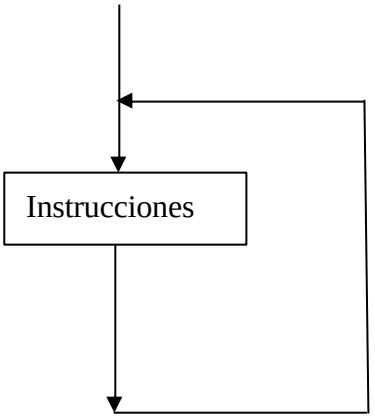
---

1. Bucles

Uno de los conceptos fundamentales de la programación es el concepto del bucle o lazo. Un bucle permite básicamente la repetición del código del programa y puede funcionar de forma indefinida o repetir una parte del código un número determinado de veces. En primer lugar se estudiarán los bucles infinitos.

1.1. Bucles infinitos

Un bucle o lazo infinito es aquel que no tiene fin, es decir, mantiene la siguiente estructura:



Por ejemplo:

```
Principal
          ..........
          ..........
          ..........
          goto Principal
```

La mayoría de los programas de microcontroladores posee una estructura de bucle infinito.

El programa CONTAR.ASM funciona como un bucle infinito, enviando de forma secuencial un número binario comprendido entre 0 y 255 a PORTB.

```
;CONTAR.ASM

               List   p=16F84;Tipo de procesador
               include "P16F84.INC"  ;Definiciones de registros internos

               org    0x00           ;Vector de Reset
               goto   Inicio

               org    0x05           ;Salva el vector de interrupción

Inicio         bsf STATUS,RP0        ;selecciona el Banco1
               clrf   TRISB          ;configura PORTB todo como salidas
               bcf STATUS,RP0        ;selecciona el Banco0
               clrf   PORTB          ;borra PORTB (leds apagados)

Bucle          incf PORTB,f          ;suma 1 al registro PORTB
               goto Bucle            ;y otra vez, y otra vez, …

               end
```

EJERCICIO 1:

Cread un proyecto y simulad el funcionamiento del programa **CONTAR.ASM**.

EJERCICIO 2:

Modificad el programa **CONTAR.ASM** para que la secuencia que se muestre en PORTB sea:

> *(en decimal):* **254, 252, 250, 248, ... 4, 2, 0, 254, 252, 250, 248, ... 4, 2, 0, 254, 252, 250, 248, ... 4, 2, 0, ...**
>
> *(o en hexadecimal):* **FE, FC, FA, ... 04, 02, 00, FE, FC, FA, ... 04, 02, 00, FE, FC, FA, ...04, 02, 00, FE, FC, FA, ... 04, 02, 00,...**

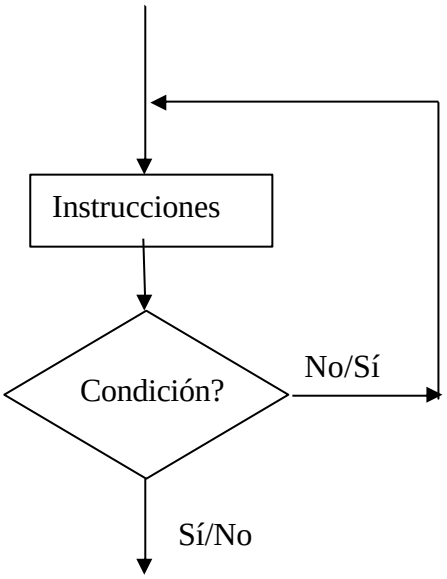
1.2. Bucles finitos

Los bucles finitos se ejecutan un determinado número de veces y pueden tener varios tipos de estructuras, las más utilizadas son:

- Bucle con condición de testeo
- Bucle que se repite un número conocido de veces

1.2.1.Bucle con condición de testeo

Se utiliza una instrucción de testeo para controlar la ejecución del bucle. Para este caso la repetición del lazo es finita, pero no se puede precisar el número de veces que se repite.



Ejemplo1:

```
EsperaUno
          btfss PORTA,4
          goto EsperaUno
```

En el ejemplo1 se lee el pin 4 de PORTA y hasta que no se pone a 1 no sale de este bucle.

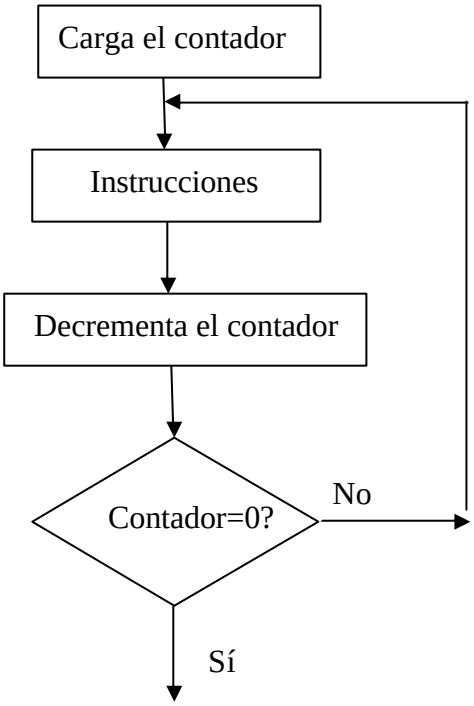
Ejemplo2:

```
NoEsIgual
          movf PORTA,W
          sublw 0x56
          btfss STATUS,Z
          goto NoEsIgual
EsIgual
```

En el ejemplo2 se lee el registro PORTA y hasta que no su valor no es igual al de la constante 0x56 no se sale del bucle.

1.2.2.Bucle que se repite un número conocido de veces

Las instrucciones **decfsz** e **incfsz** se utilizan generalmente en combinación con una instrucción de salto **goto**, para el diseño de bucles de instrucciones que deben repetirse una cantidad determinada de veces. Se hace de manera tal que un registro se decrementa o incrementa hasta que tome un determinado valor. En este caso, sí se conoce el número de veces que se repite el bucle:



Ejemplo:

```
      movlw NumeroVeces; este es el numero de veces que se repite el bucle
      movwf Contador; carga el contador con el numero de veces
Lazo ……
      ……
      ……
      decfsz Contador,F; se decrementa el contador hasta que llega a 0
      goto Lazo; si no llega a cero repite el bucle
      ……
      ……
```

EJERCICIO 3:

Cread un proyecto y simulad el funcionamiento del programa **SALTOS_01.ASM**.

El Puerto B, que actúa como salida es controlado por el bit 0 del Puerto A, que actúa como entrada. De manera tal que:

- *Si el bit 0 del PORTA es "1", se encienden todos los LEDs de salida.*

- *Si el bit 0 del PORTA es "0", se encienden los LEDs del nibble alto y se apagan los bajo.*

EJERCICIO 4:

Cread un proyecto y simulad el funcionamiento del programa **SALTOS_02.ASM**.

Compara el dato del puerto de entrada PORTA y un "Numero" (por ejemplo el 13):

- *Si (PORTA) = Numero, se encienden todos los LEDs de salida.*

- *Si (PORTA) y Numero no son iguales, se activan los LEDs pares de salida y apagan impares.*

EJERCICIO 5:

Cread un proyecto y simulad el funcionamiento del programa **SALTOS_03.ASM**.

Compara el dato introducido por el Puerto A que actúa como entrada, con un "Numero":

- *Si (PORTA) es mayor o igual que "Numero" se encienden todos los LEDs de salida.*

- *Si (PORTA) es menor que "Numero" se activan los LEDs pares de salida.*

EJERCICIO 6:

Cread un proyecto y simulad el funcionamiento del programa **SALTOS_05.ASM.**

Compara el dato introducido por el Puerto A que actúa como entrada, con un "Numero". Pueden darse 3 posibilidades

- *Si (PORTA) = "Numero" se encienden todos los LEDs de salida.*

- *Si (PORTA) > "Numero" se activan los LEDs pares de salida.*

- *Si (PORTA) < "Numero" sólo se encienden los LEDs del nibble alto.*

1.3. Bucles anidados

Un bucle anidado es un bucle dentro de otro bucle. El número total de iteraciones ejecutadas es el producto de las iteraciones del bucle interno por las iteraciones del bucle externo.

Haciendo uso de estos bucles, se tiene la posibilidad de superar las 256 iteraciones, límite de un bucle con contador. El bucle interno puede ejecutarse 256 veces y el externo se ejecuta cada vez que el bucle interno termina, hasta un total como máximo de 256. En el programa PAUSA.ASM aparece un bucle anidado.

```
;PAUSA.ASM
               List   p=16F84        ;Tipo de procesador
               include"P16F84.INC"   ;Definiciones de registros internos
Contador1      equ 0Ch
Contador2      equ 0Dh


               org    0x00           ;Vector de Reset
               goto   Inicio

               org    0x05           ;Salva el vector de interrupción
Inicio         bsf STATUS,RP0
               clrf   TRISB
               bcf STATUS,RP0
               clrf   PORTB
               clrf Contador1
               clrf Contador2

Principal      incf PORTB,f

Bucle          decfsz Contador1,f
               goto Bucle
               decfsz Contador2,f
               goto Bucle

               goto Principal

               end
```

EJERCICIO 7:

Cread un proyecto y simulad el funcionamiento del programa **PAUSA.ASM.**

2. Medir tiempos con MPLAB

Para calcular el tiempo de ejecución de un programa o de una subrutina, se puede contar el número de instrucciones que se realizan y multiplicarlo por 4 veces el período de la señal de reloj o por 8 en el caso de que las instrucciones sean de salto. Esto en algunas ocasiones resulta engorroso.

El MPLAB dispone de una opción de cronómetro denominada ***Stopwatch*** que permite medir el tiempo de ejecución de las instrucciones de los programas.
El cronómetro ***Stopwatch*** calcula el tiempo basándose en la frecuencia del reloj del microcontrolador PIC que se está simulando. Es necesario fijar previamente la frecuencia del oscilador empleado, para eso, se activa desde el menú ***Debugger > Settings > Clock*** tal como se muestra en la Figura 1. Inmediatamente se abre un cuadro de diálogo donde se fija la frecuencia del reloj. En general, se utilizará el valor de 4 MHz.

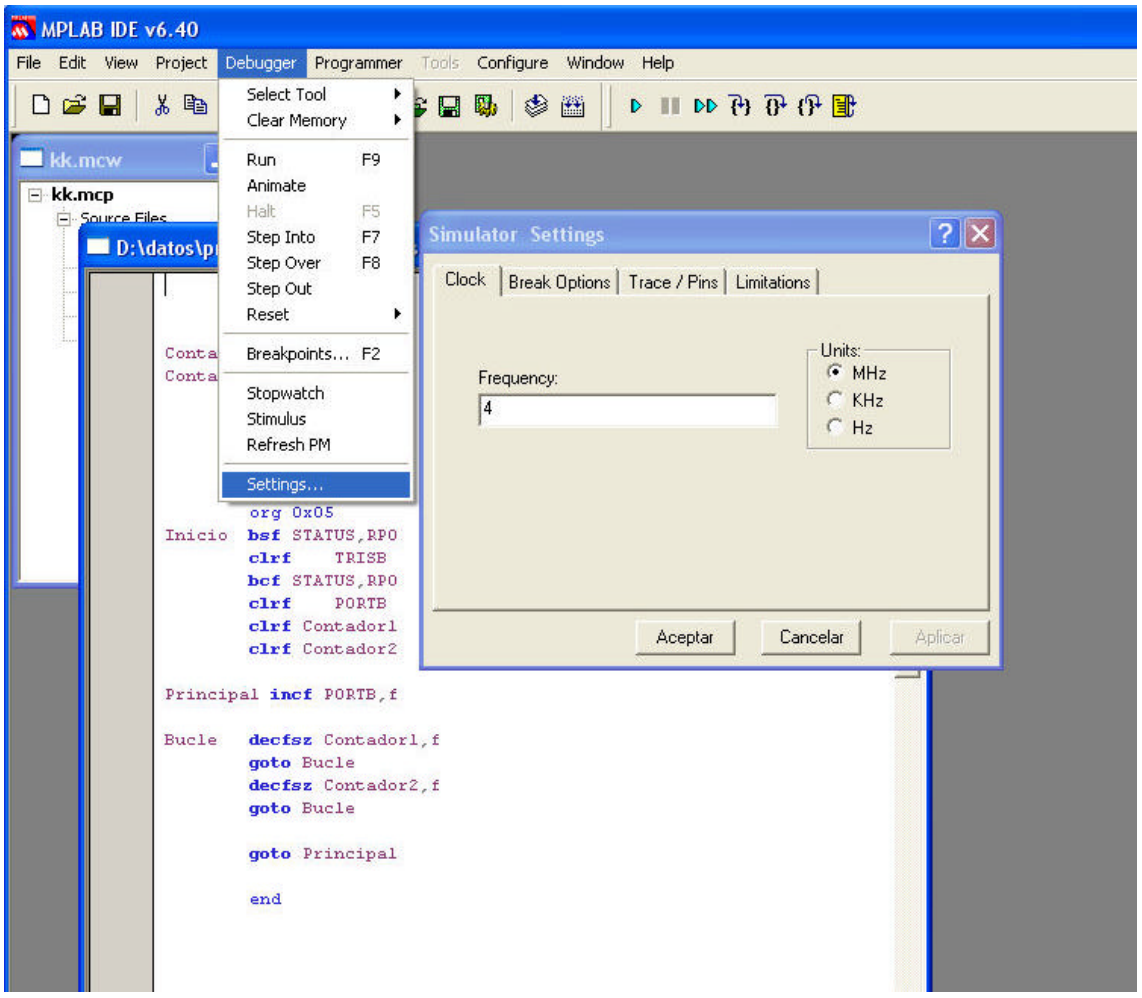


Figura 1. Selección de la frecuencia de simulación en el MPLAB.

Después se activa la opción ***Debugger > Stopwatch***, con esto se consigue abrir la ventana que muestra el tiempo transcurrido y los ciclos máquina empleados en la ejecución de cada instrucción, como puede apreciarse en la Figura 2.

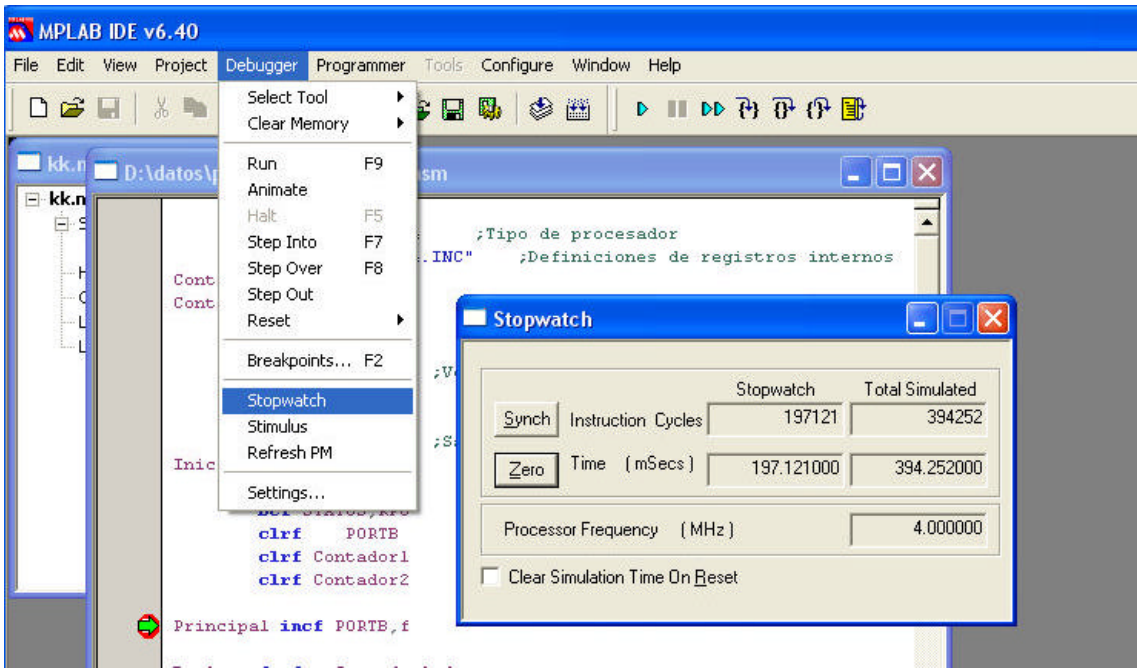


Figura 2. Ventana con el contenido del tiempo transcurrido

EJERCICIO 8:

Mide con MPLAB el tiempo que tarda en ejecutarse el bucle anidado del programa **PAUSA.ASM.**

3. Ejercicios opcionales

EJERCICIO 9:
Escribid un programa al que denominaremos **DESTELLO.ASM** que haga parpadear indefinidamente un LED conectado al bit 0 de la puerta B. Tal y como se ha explicado, para poder ver los destellos debe introducir un retardo entre el encendido y el apagado del LED.

Cread un proyecto en MPLAB y simulad el programa **DESTELLO.ASM.**

EJERCICIO 10:
Escribid un programa denominado **FLIP.ASM** que lea el estado del pulsador conectado al bit 1 de la puerta A y, según su estado, encienda o apague un LED conectado al bit 1 de la puerta B. Es decir, mientras se mantenga apretado el pulsador conectado a la puerta A, se mantendrá encendido el LED del bit 0 de la puerta B y, cuando se suelte el pulsador, lo apagará.

Cread un proyecto en MPLAB y simulad el programa **FLIP.ASM.**

EJERCICIO 11:
Realizad una modificación al programa anterior, a la que se denominará **ONOFF.ASM**, para que tenga lugar una acción de conmutación. El botón ON es conectado al bit 0 de la puerta A y el de OFF, al bit 1 de la puerta A. El LED sigue siendo el del bit 1 del puerta B.

Cread un proyecto en MPLAB y simulad el programa **ONOFF.ASM**.