INTRODUCCIÓN

Llegó el gran momento...!!!. Antes de dar inicio a esta serie de Tutoriales, quiero agradecer a quienes colaboraron en la publicación de estas páginas y a quienes las enriquecerán con sus valiosos aportes, los cuales serán muy bienvenidos, así es que ... GRACIAS...!!!

Desde hace tiempo quería publicar estos documentos en un formato sencillo de descargar, la fama que éstos alcanzaron, me obligaron a conservarlos tal como fueron creados.

Así fueron mis inicios y aquí quedarán, las animaciones fueron quitadas y algunas reemplazadas por imágenes estáticas, tratando siempre de mantener su esencia.

Al final de cada tutorial verán unos enlaces que están relacionados con el tutorial en cuestión, espero no desanimarlos con esto, pero hay cosas que ya quedaron en la historia. Confío en que ustedes sabrán seleccionar de aquí todo aquello que les sea de utilidad.

Sean Bienvenidos y disfruten de todo lo expuesto en estos documentos...!!!

*Tutorial reconocido por el Grupo de ABCdatos.com
y Valorado por sus lectores.
Mi agradecimiento para todos ellos...!!!.*

R-Luis...

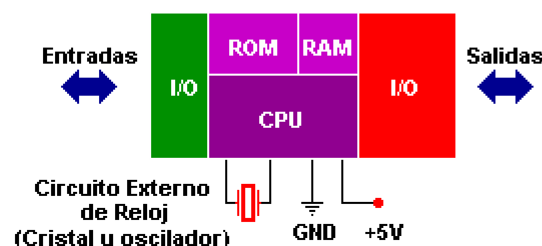
SISTEMAS MICROCONTROLADOS

El diagrama de un sistema microcontrolado sería algo así

Los dispositivos de entrada pueden ser un teclado, un interruptor, un sensor, etc.

Los dispositivos de salida pueden ser LED's, pequeños parlantes, zumbadores, interruptores de potencia (tiristores, optoacopladores), u otros dispositivos como relés, luces, un secador de pelo, en fin.. lo que quieras.

Aquí tienes una representación en bloques del microcontroador, para que te des una idea, y puedes ver que lo adaptamos tal y cual es un ordenador, con su fuente de alimentación, un circuito de reloj y el chip microcontrolador, el cual dispone de su CPU, sus memorias, y por supuesto, sus puertos de comunicación listos para conectarse al mundo exterior.



Definamos entonces al microcontrolador; Es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Sacado de un libro...!!!. En fin estas son básicamente algunas de sus partes...

- **Memoria ROM** (Memoria de sólo lectura)
- **Memoria RAM** (Memoria de acceso aleatorio)
- **Líneas de entrada/salida (I/O)** También llamados puertos
- **Lógica de control** Coordina la interacción entre los demás bloques

Eso no es todo, algunos traen funciones especiales, ya hablaremos de ellas.

Microcontroladores PIC16CXX/FXX de Microchip

Me referiré a estos porque serán los que utilizaré aquí, (al menos por ahora). Estos micros pertenecen a la gama media y disponen de un set de 35 instrucciones, por eso lo llaman de tipo RISC (Reduced Instruction Set Computer) en entendible sería "Computador con Set de Instrucciones Reducido" pocas instrucciones pero muy poderosas, otras son de tipo CISC (Complex Instruction Set Computer - Computador con Set de Instrucciones Complejo), demasiadas instrucciones, y lo peor, difíciles de recordar.

Esta familia de microcontroladores se divide en tres rangos según la capacidad de los microcontroladores. El más bajo lo compone la familia 16C5X. El rango medio lo componen las familias 16C6X/ 7X/ 8X, algunos con conversores A/D, comparadores, interrupciones, etc. La familia de rango superior lo componen los 17CXX.

Estas son las funciones especiales de las cuales disponen algunos micros...

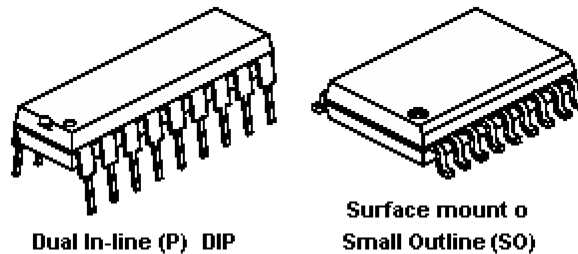
- **Conversores análogo a digital (A/D)** en caso de que se requiera medir señales analógicas, por ejemplo temperatura, voltaje, luminosidad, etc.
- **Temporizadores programables (Timer's)** Si se requiere medir períodos de tiempo entre eventos, generar temporizaciones o salidas con frecuencia específica, etc.
- **Interfaz serial RS-232.** Cuando se necesita establecer comunicación con otro microcontrolador o con un computador.
- **Memoria EEPROM** Para desarrollar una aplicación donde los datos no se alteren a pesar de quitar la alimentación, que es un tipo de memoria ROM que se puede programar o borrar eléctricamente sin necesidad de circuitos especiales.
- **salidas PWM (modulación por ancho de pulso)** Para quienes requieren el control de motores DC o cargas resistivas, existen microcontroladores que pueden ofrecer varias de ellas.
- **Técnica llamada de "Interrupciones"**, (ésta me gustó) Cuando una señal externa activa una línea de interrupción, el microcontrolador deja de lado la tarea que está ejecutando, atiende dicha interrupción, y luego continúa con lo que estaba haciendo.

Todo esto, sólo para tener una idea de lo que son los micros, ahora vamos a un par de ellos en especial

Presentación oficial! - PIC16C84/F84

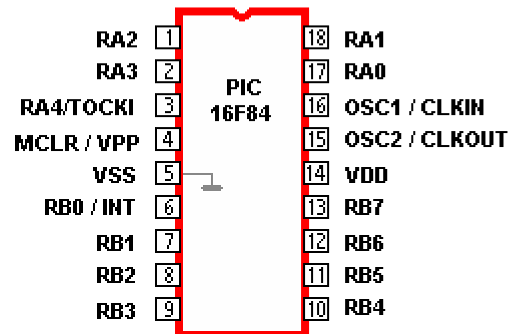
El PIC16C84 está fabricado en tecnología CMOS, consume baja potencia, y es completamente estático (si el reloj se detiene, los datos de la memoria no se pierden). El 16F84 tiene las mismas características pero posee memoria FLASH, esto hace que tenga menor consumo de energía, y como si fuera poco tiene mayor capacidad de almacenamiento.

El encapsulado más común para estos microcontrolador es el DIP (Dual In line Pin) de 18 pines, (el nuestro...), y utiliza un reloj de 4 MHz (cristal de cuarzo). Sin embargo, hay otros tipos de encapsulado, por ejemplo, el encapsulado tipo surface mount (montaje superficial) es mucho + pequeño.



Terminales del microcontrolador y sus respectivas funciones:

Ésta sería la disposición de sus terminales y sus respectivos nombres...



Encapsulado DIP - PIC16C84/F84

Patas 1, 2, 3, 17 y 18 (RA0-RA4/TOCKI): Es el PORT A. Corresponden a 5 líneas bidireccionales de E/S (definidas por programación). Es capaz de entregar niveles TTL cuando la alimentación aplicada en VDD es de 5V ± 5%. El pin **RA4/TOCKI** como entrada puede programarse en funcionamiento normal o como entrada del contador/temporizador TMR0. Cuando este pin se programa como entrada digital, funciona como un disparador de Schmitt (Schmitt trigger), puede reconocer señales un poco distorsionadas y llevarlas a niveles lógicos (cero y cinco voltios). Cuando se usa como salida digital se comporta como colector abierto; por lo tanto se debe poner una resistencia de pull-Up (resistencia externa conectada a un nivel de cinco voltios, ...no te preocupes, mas abajo lo entenderás mejor). Como salida, la lógica es inversa: un "0" escrito al pin del puerto entrega a la salida un "1" lógico. Este pin como salida no puede manejar cargas como fuente, sólo en el modo sumidero.

Pata 4 (MCLR / Vpp): Es una pata de múltiples aplicaciones, es la entrada de Reset (master clear) si está a nivel bajo y también es la habilitación de la tensión de programación cuando se está programando el dispositivo. Cuando su tensión es la de VDD el PIC funciona normalmente.

Patas 5 y 14 (VSS y VDD): Son respectivamente las patas de masa y alimentación. La tensión de alimentación de un PIC está comprendida entre 2V y 6V aunque se recomienda no sobrepasar los 5.5V.

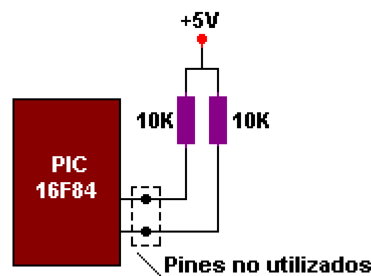
Patas 6, 7, 8, 9, 10, 11, 12, 13 (RB0-RB7): Es el PORT B. Corresponden a ocho líneas bidireccionales de E/S (definidas por programación). Pueden manejar niveles TTL cuando la tensión de alimentación aplicada en VDD es de 5V ± 5%. RB0 puede programarse además como entrada de interrupciones externas INT. Los pines RB4 a RB7 pueden programarse para responder a interrupciones por cambio de estado. Las patas RB6 y RB7 se corresponden con las líneas de entrada de reloj y entrada de datos respectivamente, cuando está en modo programación del integrado.

Patas 15 y 16 (OSC1/CLKIN y OSC2/CLKOUT): Corresponden a los pines de la entrada externa de reloj y salida de oscilador a cristal respectivamente.

Ahora un poco de electrónica:

Esto comienza a ponerse interesante, no crees...?, ok sigamos... Como estos dispositivos son de tecnología CMOS, todos los pines deben estar conectados a alguna parte, nunca dejarlos al aire porque

se puede dañar el integrado. Los pines que no se estén usando se deben conectar a la fuente de alimentación de +5V, como se muestra en la siguiente figura...

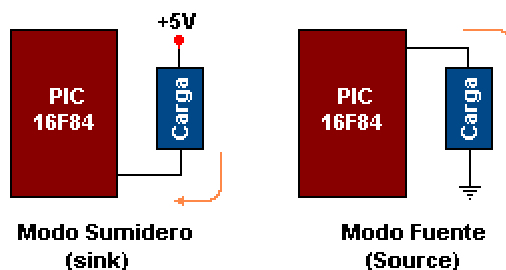


Capacidad de corriente en los puertos

La máxima capacidad de corriente de cada uno de los pines de los puertos en modo sumidero (sink) es de 25 mA y en modo fuente (source) es de 20 mA. La máxima capacidad de corriente total de los puertos es:

	PUERTO A	PUERTO B
Modo Sumidero	80 mA	150 mA
Modo Fuente	50 mA	100 mA

Así se vería la conexión para ambos modos de funcionamiento.



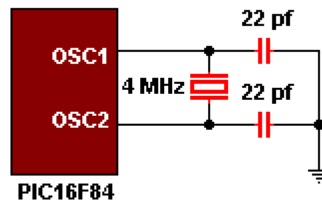
El oscilador externo

Es un circuito externo que le indica al micro la velocidad a la que debe trabajar. Este circuito, que se conoce como oscilador o reloj, es muy simple pero de vital importancia para el buen funcionamiento del sistema. El P1C16C84/F84 puede utilizar cuatro tipos de reloj diferentes. Estos tipos son:

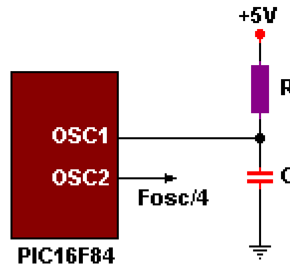
- **RC.** Oscilador con resistencia y condensador.
- **XT.** Cristal.
- **HS.** Cristal de alta velocidad.
- **LP.** Cristal para baja frecuencia y bajo consumo de potencia.

En el momento de programar o "quemar" el microcontrolador se debe especificar qué tipo de oscilador se usa. Esto se hace a través de unos fusibles llamados "fusibles de configuración" o **fuses.**

Aquí utilizaremos el cristal de 4 MHz, porque garantiza mayor precisión y un buen arranque del microcontrolador. Internamente esta frecuencia es dividida por cuatro, lo que hace que la frecuencia efectiva de trabajo sea de 1 MHz, por lo que cada instrucción se ejecuta en un microsegundo. El cristal debe ir acompañado de dos condensadores y el modo de conexión es el siguiente...



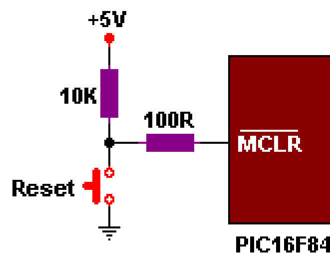
Si no requieres mucha precisión en el oscilador, puedes utilizar una resistencia y un condensador, como se muestra en la figura. donde OSC2 queda libre entregando una señal cuya frecuencia es la del OSC/4.



Según las recomendaciones de Microchip R puede tomar valores entre 5k y 100k, y C superior a 20pf.

Reset

El PIC 16C84/F84 posee internamente un circuito temporizador conectado al pin de reset que funciona cuando se da alimentación al micro, se puede entonces conectar el pin de MCLR a la fuente de alimentación. Esto hace que al encender el sistema el microcontrolador quede en estado de reset por un tiempo mientras se estabilizan todas las señales del circuito (lo cual es bastante bueno, por eso siempre la usaremos...).



Este último circuito, es por si deseas tener control sobre el reset del sistema, sólo le conectas un botón y listo...

Ahora vamos al interior del micro...

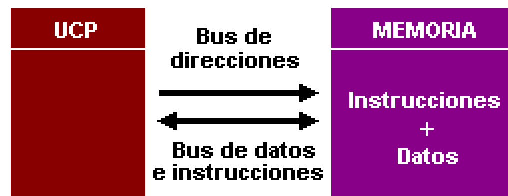
ARQUITECTURA INTERNA DEL MICROCONTROLADOR

Uffff...!!!, Ya sé...!!!, tranquilo que ya comenzaremos con lo que estas esperando, antes debemos saber dónde alojar nuestro programa, como se va a ejecutar, y como configurar sus puertos.

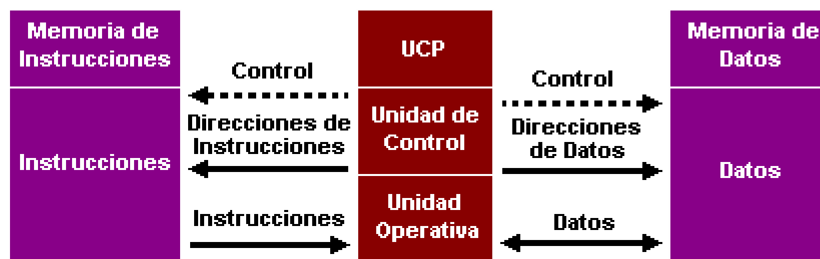
Arquitectura interna del PIC:

Hay dos arquitecturas conocidas; la clásica de von Neumann, y la arquitectura Harvard, veamos como son...

Arquitectura Von Neumann Dispone de una sola memoria principal donde se almacenan datos e instrucciones de forma indistinta. A dicha memoria se accede a través de un sistema de buses único (direcciones, datos y control).



Arquitectura Harvard Dispone de dos memorias independientes, una que contiene sólo instrucciones, y otra que contiene sólo datos. Ambas disponen de sus respectivos sistemas de buses de acceso y es posible realizar operaciones de acceso (lectura o escritura) simultáneamente en ambas memorias, ésta es la estructura para los PIC's.



Ahora vamos por partes, *o creo que me voy a perder...* :oP

El procesador o UCP

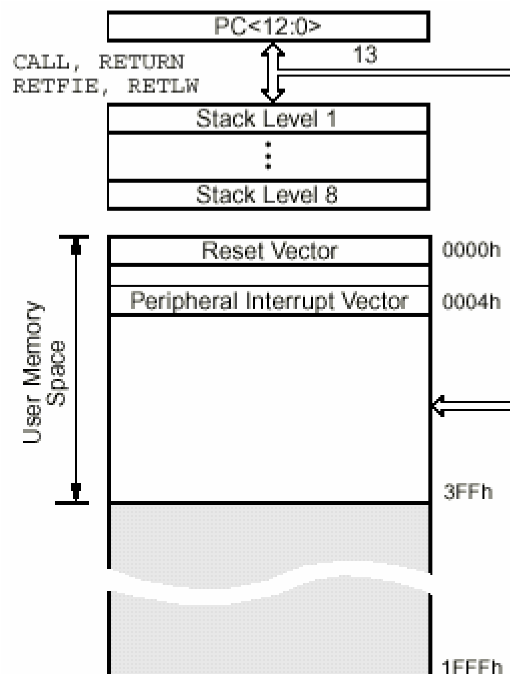
Es el elemento más importante del microcontrolador. Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, decodificarlo y ejecutarlo, también realiza la búsqueda de los operandos y almacena el resultado.

Memoria de programa

Esta vendría a ser la memoria de instrucciones, aquí es donde almacenaremos nuestro programa o código que el micro debe ejecutar. No hay posibilidad de utilizar memorias externas de ampliación. Son 5 los tipos de memoria. pero sólo describiré dos:

- **Memorias EEPROM.** (Electrical Erasable Programmable Read Only Memory - Memoria de sólo lectura Programable y borrable eléctricamente) Común en el PIC 16C84. Ésta tarea se hace a través de un circuito grabador y bajo el control de un PC. El número de veces que puede grabarse y borrarse una memoria EEPROM es finito aproximadamente 1000 veces, *no es acaso suficiente...?*. Este tipo de memoria es relativamente lenta.
- **Memorias FLASH.** Disponible en el PIC16F84. Posee las mismas características que la EEPROM, pero ésta tiene menor consumo de energía y mayor capacidad de almacenamiento, por ello está sustituyendo a la memoria EEPROM.

La memoria de programa se divide en páginas de 2,048 posiciones. El PIC16F84A sólo tiene implementadas 1K posiciones es decir de 0000h a 03FFh y el resto no está implementado. (*es aquello que se ve en gris*)



Cuando ocurre un Reset, el contador de programa (PC) apunta a la dirección 0000h, y el micro se inicia nuevamente. Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la iniciación del mismo (*por ejemplo, la configuración de los puertos...*).

Ahora, si ocurre una interrupción el contador de programa (PC) apunta a la dirección 0004h, entonces ahí escribiremos la programación necesaria para atender dicha interrupción.

Algo que se debe tener en cuenta es la pila o Stack, que consta de 8 posiciones (*o niveles*), esto es como una pila de 8 platos el último en poner es el primero en sacar, si seguimos con este ejemplo, cada plato contiene la dirección y los datos de la instrucción que se está ejecutando, así cuando se efectúa una llamada (CALL) o una interrupción, el PC sabe dónde debe regresar (*mediante la instrucción RETURN, RETLW o RETFIE, según el caso*) para continuar con la ejecución del programa.

Recuerda, sólo 8 llamadas "CALL", ten en cuenta las "INTERRUPCIONES".

Memoria de datos

Tiene dos zonas diferentes:

1. RAM estática ó SRAM: donde residen los Registros Específicos (SFR) con 24 posiciones de tamaño byte, aunque dos de ellas no son operativas y los Registros de Propósito General (GPR) con 68 posiciones. La RAM del PIC16F84A se halla dividida en dos bancos (banco 0 y banco 1) de 128 bytes cada uno (7Fh)

File Address	Bank 0	Bank 1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h			87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch – 4Fh	68 General Purpose registers (SRAM)	Mapped (accesses) in Bank 0	8Ch – CFh
50h – 7Fh			D0h – FFh

2. EEPROM: de 64 bytes donde, opcionalmente, se pueden almacenar datos que no se pierden al desconectar la alimentación.

O.k., ahora unas cuantas palabras finales respecto a cómo configurar los puertos del PIC y comenzamos con lo más emocionante.

CONFIGURACIÓN DE LOS PUERTOS DEL PIC

Llegó el momento de ver como configurar los puertos del PIC. Para poder hacerlo es necesario conocer la tabla de registros de la memoria de datos, la cual como dijimos, está dividida en el **BANCO 0** y **BANCO 1**.

Los registros importantes en la configuración de los puertos son:

STATUS dirección **0x3**
PORTA dirección **0x5**
PORTB dirección **0x6**
TRISA dirección **0x5**
TRISB dirección **0x6**

Por defecto el PIC tendrá todos los I/O port's (es decir los puertos RA y RB), colocados como entrada de datos, y si queremos cambiarlos habrá que configurarlos.

Al configurar los puertos deberás tener en cuenta que:

Si asignas un **CERO (0)** a un pin, éste quedará como **salida** y...
Si le asignas un **UNO (1)**, quedará como **entrada**

Esta asignación se hace en:

TRISA para los pines del **PUERTO A** (5 bits)
TRISB para los pines del **PUERTO B** (8 bits)

Por Ejemplo:

Si **TRISA** es igual a 11110 todos sus pines serán entradas salvo RA0 que esta como salida

Si **TRISB** es igual a 00000001 todos sus pines serán salidas salvo RB0 que esta como entrada

Cuando el PIC arranca se encuentra en el BANCO 0, como TRISA y TRISB están en el **BANCO 1** no queda otra, deberemos cambiar de banco. Esto se logra a través del Registro **STATUS**

STATUS es un Registro de 8 bits u 8 casillas, en el cual la Nº 5 (**RP0**) define la posición del banco en donde nos encontramos

Si pones un **CERO (0)** a RP0 estaremos en el **BANCO 0**
Si le pones un **UNO (1)** ya ves, estaremos en el **BANCO 1**

REGISTRO STATUS							
7	6	5	4	3	2	1	0
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

Listo, ahora ya sabemos como configurar los puertos, pero lo aclararemos con un ejemplo completo.

Vamos a escribir un código que configure todos los pines del puerto A como entrada y todos los del puerto B como salida.

```
1.  ;---------------Encabezado-------------
2.          list     p=16f84       ; usaremos el PIC 16f84
3.          radix    hex           ; y la numeración hexadecimal
4.  ;------------mapa de memoria---------
5.  estado  equ      0x03          ; Aquí le asignamos nombres a los
6.  trisa   equ      0x05          ; registros indicando la posición
7.  trisb   equ      0x06          ; en la que se encuentran
8.  ;-------Configuración de puertos-------
9.  reset   org      0x00          ; origen del programa, aquí comenzaré
10.                                ; siempre que ocurra un reset
11.         goto     inicio         ; salto a "inicio"
12.         org      0x05          ; origen del código de programa
13. Inicio  bsf      estado,5      ; pongo rp0 a 1 y paso al banco1
14.         movlw    b'11111'      ; cargo W con 11111
15.         movwf    trisa         ; y paso el valor a trisa
16.         movlw    b'00000000'   ; cargo W con 00000000
17.         movwf    trisb         ; y paso el valor a trisb
18.         bcf      estado,5      ; pongo rp0 a 0 y regreso al banco0
19. ;------------------------------------------
20.         end                    ; se acabó
21. ;------------------------------------------
```

Descripción del código:

Todo lo que escribas luego de un "**;**" (*punto y coma*) será ignorado por el ensamblador, estos son los famosos comentarios, y sirve para saber qué hace cada línea de código.

Dicho esto no queda más que describir el código, así que vamos por partes.

```
1.  ;---------------Encabezado-------------
2.          list     p=16f84       ; usaremos el PIC 16f84
3.          radix    hex           ; y la numeración hexadecimal
```

Aquí le indicas al ensamblador para que microcontrolador estas codificando (PIC16F84). y cuál será el sistema de numeración que utilizarás (hexadecimal).

Nota que hay tres columnas, en este caso la primera está vacía. Respeta las tabulaciones para no confundir al ensamblador.

```
4.  ;------------mapa de memoria---------
5.  estado  equ     0x03         ; Aquí le asignamos nombres a los
6.  trisa   equ     0x05         ; registros indicando la posición
7.  trisb   equ     0x06         ; en la que se encuentran
```

Recuerdas lo de la memoria de datos...? Bien, al registro STATUS, que está en la posición 0x03 de la memoria de datos le puse la etiqueta **"estado"**. **equ** es algo así comoooo...**igual** . (Es decir, le estoy asignando el nombre estado al registro que está en la posición 0x03 de la memoria de datos).

Luego hice lo mismo con trisa y trisb. Ahora sigamos...

```
8.  ;-------Configuración de puertos-------
9.  reset   org     0x00         ; origen del programa, aquí comenzaré
10.                              ; siempre que ocurra un reset
11.         goto    inicio       ; salto a "inicio"
12.         org     0x05         ; origen del código de programa
13. Inicio  bsf     estado,5     ; pongo rp0 a 1 y paso al banco1
14.         movlw   b'11111'     ; cargo W con 11111
15.         movwf   trisa        ; y paso el valor a trisa
16.         movlw   b'00000000'  ; cargo W con 00000000
17.         movwf   trisb        ; y paso el valor a trisb
18.         bcf     estado,5     ; pongo rp0 a 0 y regreso al banco0
```

La directiva **org** indica el sitio de la memoria en donde se escribe una parte del programa. En este caso el contador de programa apuntará a la dirección 0x00 (*reset*) entonces ejecutará la instrucción que sigue a continuación, (saltar a la etiqueta inicio) y nuestro código de programa comienza en la dirección de memoria 0x05 (aquí salto por encima de la interrupción 0x04)

BSF (*SET FILE REGISTER*), es la instrucción que pone un uno en el bit del registro especificado, en este caso pone a uno el bit 5 del registro STATUS (*el rp0*), para pasar al banco 1.

movlw es algo así como... mueve el siguiente literal al Registro W.

W es el Registro de Trabajo, y lo usamos para almacenar momentáneamente los datos que queremos mover. una vez hecho esto pasamos el dato a trisa, o a trisb, según el caso.

movwf es algo así como... mueve el contenido del registro W al registro f, en este caso f sería trisa o trisb.

BCF (*BIT CLEAR FILE REGISTER*), ésta instrucción limpia el bit del registro especificado, o lo pone a cero, en este caso pone a cero el bit 5 del registro STATUS para regresar al banco 0.

```
19. ;------------------------------------------
20.         end                 ; se acabó
21. ;------------------------------------------
```

Fue suficiente por hoy...

Programando en serio

Debo confesar que el programa anterior aunque parezca una burrada, lo utilizaremos de tiempo completo, y lo único que cambiaremos serán los pines de entrada y salida.

Te recuerdo que lo que hicimos hasta ahora, solo fue configurar los puertos, pero no genera ninguna señal ni nada por el estilo.

Ahora si programaremos en serio. Encenderemos un LED, lo mantendremos encendido por un tiempo, luego lo apagaremos y haremos que se repita todo de nuevo. Recuerda ponerle un nombre, aquí lo llamaré LED1.asm (no olvides el **.asm**)

Comencemos

```
1.  ;---------------Encabezado-------------
2.          LIST    p=16f84
3.          radix   hex
4.  ;------------mapa de memoria---------
5.  estado  equ     0x03            ; Haciendo asignaciones
6.  TRISB   equ     0x06
7.  ptob    equ     0x06
8.  reg1    equ     0x0C            ; Estos 3 registros los utilizaré
9.  reg2    equ     0x0D            ; para hacer el retardo
10. reg3    equ     0x0E
11. ;-------Configuración de puertos-------
12. reset   org     0x00            ; origen del programa, aquí comenzará
13.                                 ; siempre que ocurra un reset
14.         goto    inicio          ; salta a "inicio"
15. inicio  bsf     estado,5        ; pone rp0 a 1 y pasa al banco1
16.         movlw   b'00000000'     ; carga W con 00000000
17.         movwf   TRISB           ; y pasa el valor a trisb
18.         bcf     estado,5        ; pone rp0 a 0 y regresa al banco0
19. ;----Aquí enciende y apaga el LED-----
20. ahora   bsf     ptob,0          ; pone un 1 en RB0 (enciende el LED)
21.         call    retardo         ; llama al retardo
22.         bcf     ptob,0          ; pone a 0 RB0 (apaga el LED)
23.         call    retardo         ; llama al retardo
24.         goto    ahora           ; repite todo de nuevo
25. ;-----------Rutina de Retardo-----------
26. retardo movlw   10              ; Aquí se cargan los registros
27.         movwf   reg1            ; reg1, reg2 y reg3
28.                                 ; con los valores 10, 20 y 30
29. tres    movlw   20              ; respectivamente
30.         movwf   reg2
31. dos     movlw   30
32.         movwf   reg3
33. uno     decfsz  reg3,1          ; Aquí se comienza a decrementar
34.         goto    uno             ; Cuando reg3 llegue a 0
35.         decfsz  reg2,1          ; le quitare 1 a reg2
36.         goto    dos             ; cuando reg2 llegue a 0
37.         decfsz  reg1,1          ; le quitare 1 a reg1
38.         goto    tres            ; cuando reg1 llegue a 0
39.         retlw   00              ; regresare al lugar
40.                                 ; de donde se hizo la llamada
41. ;------------------------------------------
42.         end                     ; se acabó
43. ;------------------------------------------
```

Descripción del código:

No te asustes por el tamaño del código, que aunque parezca difícil todo está igual que el código anterior, por lo que sólo describiré los cambios... (**lo que está en rojo**)

Se agregaron 3 registros mas (reg1, reg2 y reg3), éstos vendrían a ser como variables ubicadas en sus respectivas posiciones (0x0C, 0x0D, 0x0E,) y son registros de propósito general (recuerda que para el PIC16F84 son 68, puedes elegir cualquiera).

A demás se agregó **ptob**, etiqueta que corresponde a la dirección del puerto B

Analicemos lo que sigue..., que éste es el programa en sí:

```
19. ;----Aquí enciende y apaga el LED-----
20. ahora   bsf     ptob,0          ; pone un 1 en RB0 (enciende el LED)
21.         call    retardo         ; llama al retardo
22.         bcf     ptob,0          ; pone a 0 RB0 (apaga el LED)
23.         call    retardo         ; llama al retardo
24.         goto    ahora           ; repite todo de nuevo
```

La etiqueta **"ahora"** es el nombre de todo este procedimiento o **rutina**, de tal modo que cuando quiera repetir el procedimiento solo saltare a **"ahora"**.

bsf es poner a uno un bit, en este caso al primer bit (el bit **0**) del puerto B (**ptob**).

call es una llamada, en este caso llama a la rutina de **retardo**, cuando regrese, continuará con el código.

bcf es poner a cero un bit, en este caso al primer bit (bit **0**) del puerto B (**ptob**). y luego llama al retardo, cuando regrese se encontrará con la instrucción **goto** obligándolo a saltar a la etiqueta **ahora** para que se repita todo de nuevo. Eso es todo...!!!.

Rutina de retardo

Esta es la parte más difícil, pero trataré de hacerlo sencillo así puedes continuar con lo que sigue y no te trabas en esta parte. **Primero veremos cómo se cargan los registros para el retardo.** Veamos el código...

```
25. ;-----------Rutina de Retardo-----------
26. retardo movlw   10              ; Aquí se cargan los registros
27.         movwf   reg1            ; reg1, reg2 y reg3
28.                                 ; con los valores 10, 20 y 30
29. tres    movlw   20              ; respectivamente
30.         movwf   reg2
31. dos     movlw   30
32.         movwf   reg3
```

Recordemos que en el mapa de memoria los registros **0x0C, 0x0D** y **0x0E** fueron nombrados como **reg1, reg2** y **reg3** respectivamente. Ahora simularemos los tres registros para ver cómo se cargan mediante el registro de trabajo W, (utilizamos W por que los valores**10, 20** y **30** son valores constantes). Repito, esto es una simulación bien a lo bruto, así que vamos a suponer que en vez de 10 cargo **1**, en lugar de 20 cargo **2** y en lugar de 30 cargo **3**, hago esto, solo con fines didácticos así podrás comprenderlo mejor, ok?.

(La simulación podrás verla en la web...)

Lo que acabas de ver, fue la carga de los registros reg1, reg2 y reg3. Ahora verás cómo se comienza a decrementar cada uno de esos registros, primero reg3, luego reg2 y finalmente reg1.

```
29. tres     movlw   20            ; respectivamente
30.          movwf   reg2
31. dos      movlw   30
32.          movwf   reg3
33. uno      decfsz  reg3,1        ; Aquí se comienza a decrementar
34.          goto    uno           ; Cuando reg3 llegue a 0
35.          decfsz  reg2,1        ; le quitare 1 a reg2
36.          goto    dos           ; cuando reg2 llegue a 0
37.          decfsz  reg1,1        ; le quitare 1 a reg1
38.          goto    tres          ; cuando reg1 llegue a 0
39.          retlw   00            ; regresare al lugar
40.                                ; de donde se hizo la llamada
```

Veamos, **decfsz reg3,1** esto es, decrementa reg3, si al decrementar te da cero saltéate una línea.
El **1** que sigue a reg3, indica que guarde el valor de reg3 decrementado en el mismo reg3, es comoooo.... contador=contador-1 (**se entiende...?**)

goto, es saltar y **goto uno** es saltar a la etiqueta **uno**. En esta pequeña vuelta estoy decrementando reg3 hasta que se haga cero.

Cuando reg3 llegue a 0 decrementaré reg2 en una unidad, volveré a cargar reg3 y lo decrementaré nuevamente para recién restarle otra unidad a reg2, y así... hasta que reg2 se haga cero. Cuando eso ocurra decrementaré reg1 en una unidad, cargaré nuevamente reg2 y reg3, para luego decrementarlos de nuevo, todo esto ocurrirá hasta que reg1 se haga igual a cero.

(La simulación podrás verla en la web...)

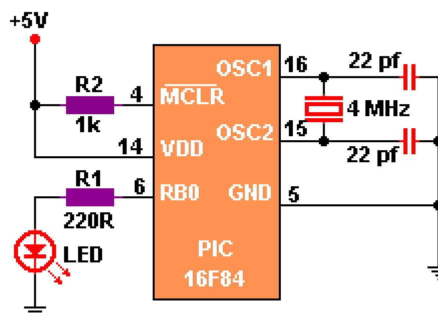
Esta rutina de retardo, aunque parezca absurda y larga nos permite ver como se enciende y se apaga el LED, de lo contrario no podríamos notar la diferencia, o lo veríamos apagado o encendido, ya que la velocidad es demasiado alta si estamos trabajando con un XT de 4 MHz.

Finalmente nos queda la última instrucción:

```
41. ;-----------------------------------------
42.          end                   ; se acabó
43. ;-----------------------------------------
```

Sin palabras.

Una vez cargado el programa en el PIC, necesitarás ver el programa funcionando, por lo que deberás armar este circuito.



El pin 4 (MCLR) está conectado por lo del Reset, para que se estabilicen los niveles de tensión.

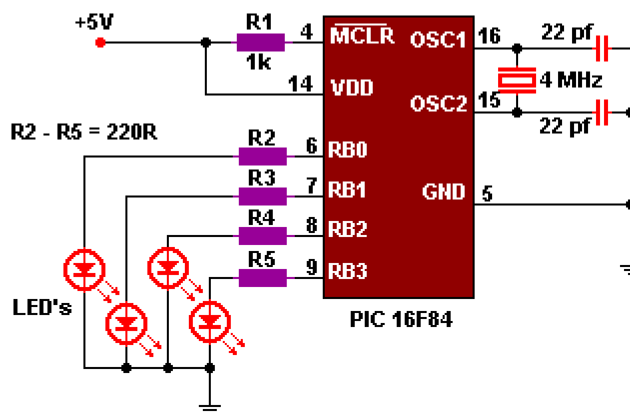
Eso fue todo, buena suerte...!!!

Antes de empezar

Si bien nos vamos a complicar con las siguientes lecciones, sería bueno que consultes el Set de instrucciones, así será más fácil que comprendas cada línea de código que se escribe, ya que cada una está acompañada de su respectivo ejemplo, y una vez lo comprendas puedes quedarte con el Resumen de instrucciones.

Lo que haremos ahora será un programa que encienda 4 LED's en forma secuencial, y para ello recurriremos a la rutina de retardo del programa anterior, que espero lo hayas comprendido, si no es así regresa y no vengas aquí hasta que lo entiendas :o))

Ok, sigamos. El circuito será el siguiente...



Y el código que realiza la secuencia es el que viene a continuación.

```
1.  ;---------------Encabezado-------------
2.          LIST    p=16f84
3.          radix   hex
4.  ;------------mapa de memoria---------
5.  estado  equ     0x03        ; Haciendo asignaciones
6.  TRISB   equ     0x06
7.  ptob    equ     0x06
8.  rotar   equ     0x0A        ; para desplazar el dato
9.  reg1    equ     0x0C        ; para hacer el retardo
10. reg2    equ     0x0D
11. reg3    equ     0x0E
12. ;-------Configuración de puertos-------
13. reset   org     0x00
14.         goto    inicio      ; salta a "inicio"
15.         org     0x05        ; aquí comienza el programa
16. inicio  bsf     estado,5    ; configurando el puerto B
17.         movlw   b'00000000'
18.         movwf   TRISB
19.         bcf     estado,5
20. ;----Realiza la secuencia de LED's-----
21. ahora   movlw   0x01        ; carga W con 00000001
22.         movwf   rotar       ; lo pasa al registro rotar
23. rotando movf    rotar,0     ; pasa el contenido de rotar a W
24.         movwf   ptob        ; y de allí al puerto B
25.                             ; encendiendo el LED correspondiente
26.         call    retardo     ; llama a retardo
27.         rlf     rotar,1     ; desplaza un bit el contenido
28.                             ; de rotar y lo guarda.
29.         btfss   rotar,4     ; prueba si se activa el 5to. bit
30.                             ; si es así saltea una línea
31.         goto    rotando     ; sino, sigue rotando
32.         goto    ahora       ; repite todo de nuevo
33. ;-----------Rutina de Retardo-----------
34. retardo movlw   10          ; Carga los registros
35.         movwf   reg1        ; reg1, reg2 y reg3
```

```
36.                                 ; con los valores 10, 20 y 30
37. tres      movlw    20           ; respectivamente
38.           movwf    reg2
39. dos       movlw    30
40.           movwf    reg3
41. uno       decfsz   reg3,1       ; Comienza a decrementar
42.           goto     uno          ; cuando termine
43.           decfsz   reg2,1       ; regresará a la siguiente
44.           goto     dos          ; línea de código
45.           decfsz   reg1,1       ; de donde fue llamado
46.           goto     tres
47.           retlw    00
48. ;------------------------------------------
49.           end                   ; The End
50. ;------------------------------------------
```

Todo está como antes, salvo lo que está en rojo. Veamos de que se trata eso de rotar y rotando.

```
20. ;----Realiza la secuencia de LED's-----
21. ahora     movlw    0x01         ; carga W con 00000001
22.           movwf    rotar        ; lo pasa al registro rotar
23. rotando   movf     rotar,0      ; pasa el contenido de rotar a W
24.           movwf    ptob         ; y de allí al puerto B
25.                                 ; encendiendo el LED correspondiente
26.           call     retardo      ; llama a retardo
27.           rlf      rotar,1      ; desplaza un bit el contenido
28.                                 ; de rotar y lo guarda.
29.           btfss    rotar,4      ; prueba si se activa el 5to. bit
30.                                 ; si es así saltea una línea
31.           goto     rotando      ; sino, sigue rotando
32.           goto     ahora        ; repite todo de nuevo
```

rotar es el registro en el que almacenaré momentáneamente el valor del desplazamiento de los bit's. Así cuando llegue al 5to. no enviaré ese dato ya que se habrá activado el 4to. bit del puerto B del PIC (sería el 4to. LED), y entonces volveré a comenzar.

Descripción del código:

Ok, voy a poner a 1 el primer bit del registro **rotar** a través de **w**, esto se hace en las dos primeras líneas.

rotando, es una subrrutina: Aquí se pasa el contenido del registro **rotar** o sea (**00000001**) a**W** (por eso el **0**) para luego enviarlo al puerto B (**portb**), y encender el primer LED, luego llama al retardo, cuando regrese se encontrará con la siguiente instrucción.

rlf rotar,1 esto es como decirle, rota el contenido del registro **rotar** un lugar a la izquierda, y guarda el valor en el mismo registro **rotar** (por eso el **1**). Es decir, que si antes**rotar=00000001** luego de esta instrucción **rotar=00000010**. Ahora viene la verificación del 5to. bit, para saber si completó la rotación.

btfss rotar,4 es como si le preguntarías ¿oye, se activó tu 5to. bit?.

Ok...!!! Ya sé lo que estás pensando ¿cómo que el 5to. si aparece el 4?, bueno, es porque no estas siguiendo el tutorial, recuerda que el primer bit está en la posición 0 y por ende, el 5to. está en la posición 4 ¿ahora está bien?. Continuemos, si resulta ser que no, saltara hasta **rotando** y pasará el contenido del registro rotar a W nuevamente (recuerda que ahora **rotar**es **00000010** por aquello del desplazamiento). Luego lo enviará al puerto B, llamará al retardo y rotará nuevamente.

Bien supongamos que ya paso todo eso y que ahora **rotar** tiene el valor **00001000** y estamos ubicados justo en la etiqueta rotando. Entonces pasará el valor a **W** y de allí al puerto B, luego llamará al retardo, cuando regrese, rotará el contenido del registro **rotar** una vez más y entonces su contenido será **00010000**. Ahora viene la pregunta...

btfss rotar,4 ¿está activo el 5to. bit del registro **rotar**?, Siiiii, si Sr. está activo..., Perfecto, entonces saltaré una línea, me encontraré con **goto ahora** y comenzaré de nuevo.

Eso es todo. Ahora veámoslo en forma gráfica para aclarar un poco las ideas.

Codigo

```
ahora    movlw   0x01
         movwf   rotar

rotando  movf    rotar,0
         movwf   ptob
         call    retardo

         rlf     rotar,1

         btfss   rotar,4
         goto    rotando
         goto    ahora
```

Retardo

Registro 0x0A

rotar 00000000

Registro de Trabajo

w 00000000

Puerto B

RB7 ---------- RB0

ptob 00000000

Esa fue la idea, que veas como se hace la rotación, hay varias formas de hacerlo, yo aquí te mostré una, otras utilizan el **carry** del registro **STATUS**, otras no utilizan la rotación para hacer esta secuencia, sino van activando los bit's de a uno, en fin, tienes muchas opciones...

Señales de Entrada

Lo interesante y llamativo de los microcontroladores es que obedecen tus órdenes al pie de los bit's :o)

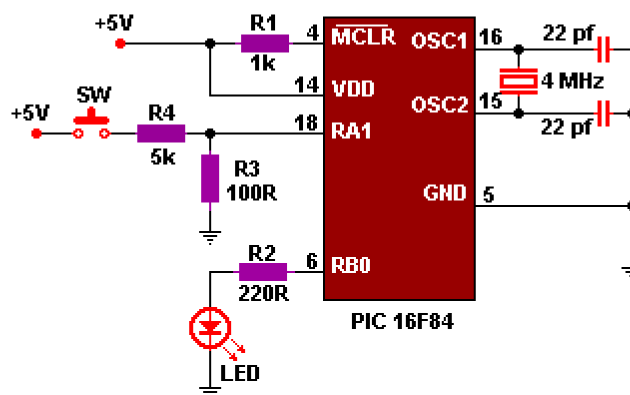
Por ejemplo, si le ordenas que vigile un pulsador, el muy pillo lo hará, y cuando alguien lo active le dará su merecido, jejejeje

Bien..., eso es lo que haremos ahora. Pero esta vez también utilizaremos el puerto A

Ahhhhhhh...!!!, y para complicarlo un poco más lo haremos con un solo pulsador. Si un travieso lo activa encenderemos un LED, y si lo activan nuevamente, lo apagaremos, quedó...?

Mmmmmmmm... Lo que estoy notando es que voy a necesitar un registro que me guarde la información de si el LED está prendido o apagado, (sino... cómo voy a saber qué hacer cuando lo presionen...!!!) bueno, ahora si...

Comencemos... el pulsador lo conectaré a RA1 y el LED a RB0, así se ve en el siguiente esquema



Hay varias formas de hacerlo, y más adelante utilizaremos el método de INTERRUPCIONES. Aquí lo haré a mi modo, porque este tutorial es mío, Ok...? :oP

Vamos a lo nuestro.

```
1.  ;---------------Encabezado-------------
2.          List     p=16F84
3.          radix    hex
4.  ;------------mapa de memoria---------
5.  STATUS  EQU      03          ; status esta en la dirección 03
6.  TRISA   EQU      05
7.  PORTA   EQU      05          ; el puerto a
8.  TRISB   EQU      06
9.  PORTB   EQU      06          ; el puerto b
10. cont    EQU      0A
11. ;-------Configuración de puertos-------
12. reset   ORG      0
13.         GOTO     inicio
14.         ORG      5
15. inicio  BSF      STATUS,5    ; configurando puertos
16.         MOVLW    0x02        ; carga w con 0000 0010
17.         MOVWF    TRISA       ; ahora RA1 es entrada
18.         MOVLW    0x00        ; carga w con 0000 0000
19.         MOVWF    TRISB       ; y el puerto B salida
20.         BCF      STATUS,5
21.         CLRF     PORTB       ; limpio el puerto B
22.         CLRF     cont        ; limpio el contador
23.         BSF      cont,0      ; pongo el contador a 1
24. ;------------------------------------------
25. test    BTFSC    PORTA,1     ; si alguien presiona
26.         CALL     led         ; voy a la etiqueta "led"
27.         GOTO     test        ; sino me quedo esperando
28. led     BTFSC    cont,0      ; si el contador está a 1
29.         GOTO     on_led      ; lo atiendo en "on_led"
30.         BCF      PORTB,0     ; sino, apago el LED
31.         BSF      cont,0      ; pongo el contador a 1
32.         GOTO     libre       ; y espero que suelten el pulsador
33. on_led  BSF      PORTB,0     ; enciendo el LED
34.         CLRF     cont        ; pongo el contador a 0
35. libre   BTFSC    PORTA,1     ; si no sueltan el pulsador
36.         GOTO     libre       ; me quedaré a esperar
37.         RETURN               ; si lo sueltan regreso
38.                              ; a testear nuevamente
39. ;------------------------------------------
40.         END
41. ;------------------------------------------
```

El registro que agregué, como te habrás dado cuenta es **cont** y su dirección respectiva **0x0A**. De la configuración de puertos ni hablar, vamos por lo que está en rojo.

```
21.         CLRF     PORTB       ; limpio el puerto B
22.         CLRF     cont        ; limpio el contador
23.         BSF      cont,0      ; pongo el contador a 1
```

CLRF es borrar, o limpiar, o poner a cero, en este caso pongo a cero todo el puerto B y también el registro **cont**, y luego pongo a 1 el primer bit de este último, es decir el bit **0**.

Con esto me aseguro de que no hay ninguna señal en el puerto B, y que el registro **cont** es igual a **0000001**, (**señal de que el LED está apagado**)

Sigamos...

```
25. test    BTFSC    PORTA,1     ; si alguien presiona
26.         CALL     led         ; voy a la etiqueta "led"
27.         GOTO     test        ; sino me quedo esperando
```

Con **BTFSC** estoy probando el segundo bit (**Bit 1**) del puerto A. Si este bit esta a cero es por que nadie lo presionó, entonces salto una línea, y me encuentro con **GOTO test**, así que aquí estaré dando vueltas un buen rato, hasta que a alguien se le ocurra presionar el dichoso pulsador...

Ohhhhhhhhh...!!!, parece ser que alguien lo presionó. Bueno, esta vez no iré a GOTO, sino a**CALL led**, esto es una llamada a la subrutina **led**, allí vamos...

```
28. led     BTFSC   cont,0       ; si el contador está a 1
29.         GOTO    on_led       ; lo atiendo en "on_led"
30.         BCF     PORTB,0      ; sino, apago el LED
31.         BSF     cont,0       ; pongo el contador a 1
32.         GOTO    libre        ; y espero que suelten el pulsador
33. on_led  BSF     PORTB,0      ; enciendo el LED
34.         CLRF    cont         ; pongo el contador a 0
35. libre   BTFSC   PORTA,1      ; si no sueltan el pulsador
36.         GOTO    libre        ; me quedaré a esperar
37.         RETURN               ; si lo sueltan regreso
38.                              ; a testear nuevamente
```

Antes de hacer algo debo saber si el LED está encendido o apagado. Recuerda que si está apagado **cont=0000001**, de lo contrario **cont=00000000**

Pregunta...(**BTFSC cont,0 ?**) - el primer bit del registro cont es igual a uno?...

Si es así el LED está apagado así que lo atenderé en **"on_led"** ahí pondré a uno el primer bit del puerto B (encenderé el led), luego haré **cont=0000000** para saber que desde este momento el LED está encendido.

El tema es que nunca falta aquellos que presionan un pulsador y luego no lo quieren soltar, así que le daremos para que tengan..., y nos quedaremos en la subrrutina **"libre"** hasta que lo suelten, y cuando lo liberen, saltaremos una línea hasta la instrucción **RETURN**.

Así es que caeremos en (**GOTO test**) y esperamos a que opriman nuevamente el pulsador. y si quieres saber si esto funciona ponle el dedito, y caerás otra vez en la subrrutina **"led"**

Pregunta...(**BTFSC cont,0 ?**) - el primer bit del registro cont es 1?...

Noooooooo...!!! Eso significa que el LED está encendido, entonces lo apagaré (**BCF PORTB,0**), haré **cont=00000001** (de ahora en más LED apagado) y me quedaré en**"libre"** esperando a que sueltes el pulsador, y cuando lo hagas mi 16F84 estará listo para un nuevo ciclo.

Te gustó...?, fue fácil verdad...?

Creo que es un buen punto de partida.

R-Luis...

--- /// ---

Bueno, mil gracias a todos los lectores de esta web, a los que me siguieron durante tanto tiempo por todo hosting por el que anduve hospedando la web.
A quienes hicieron sus aportes, sus críticas y a quienes especialmente se encargaron de publicar la web por cuanto sitio pudieron. Mis Saludos para Todos Uds.
Cordialmente...
R-Luis