:: Microcontroladores PIC - Tabla de Referencias

Conceptos Básicos

Programación

Descripción de algunas Herramientas

Apéndice

En la sección de Software se encuentra la descripción de algunos programas que pueden serte de utilidad para la programación de los microcontroladores, claro que depende del circuito grabador del que dispones para esta tarea.

:: Microcontroladores PIC - Capítulo 1

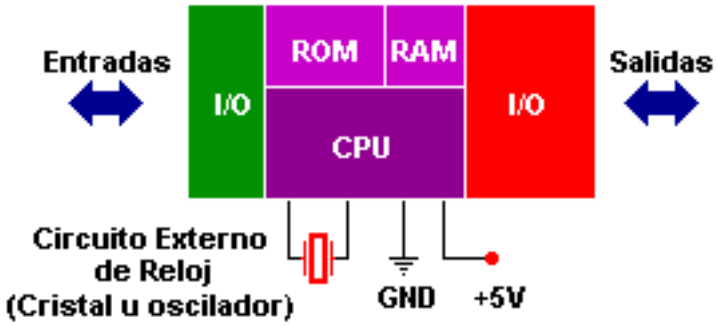
Microcontroladores - Sistemas microcontrolados

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.

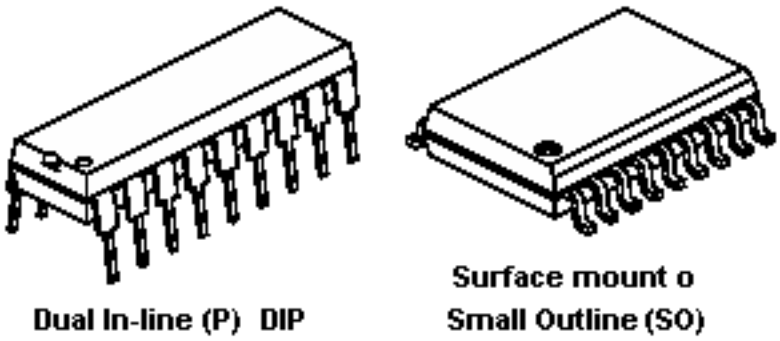
Todo esto, sólo para tener una idea de lo que son los micros, ahora vamos a un par de ellos en especial

:: Microcontroladores PIC - Capítulo 2

Presentación oficial! - PIC16C84/F84

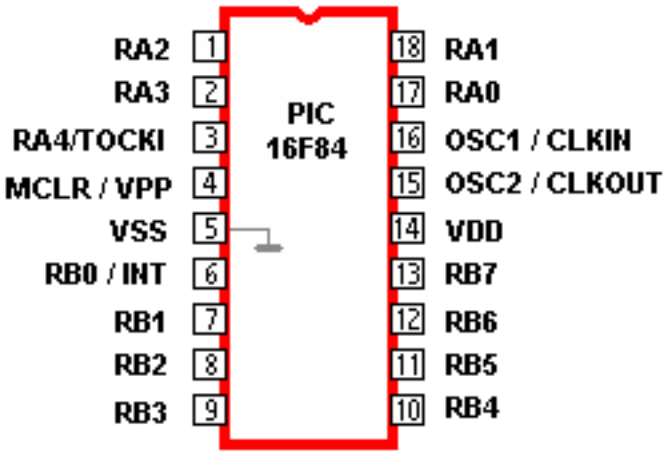
El P1C16C84 está fabricado en tecnología CMOS, consume baja potencia, y es completamente estático (si el reloj se detiene, los datos de la memoria no se pierden). El 16F84 tiene las mismas características pero posee memoria FLASH, esto hace que tenga menor consumo de energía, y como si fuera poco tiene 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.

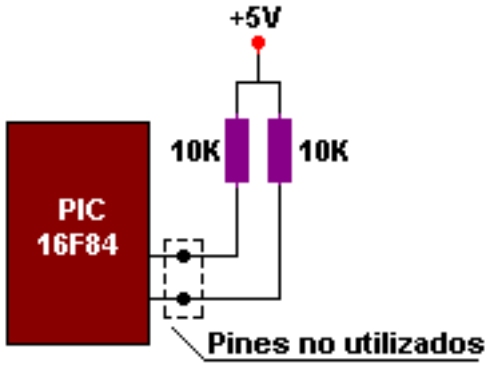
Patas 5 y 14 (VSS y VDD): Son respectivamente las patas de masa y alimentación. La tensión de alimentación de un PIC está comprendida entre 2V y 6V aunque se recomienda no sobrepasar los 5.5V.

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...

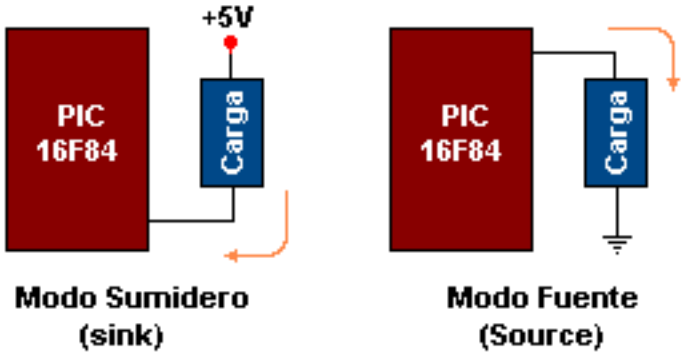


Capacidad de corriente en los puertos

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

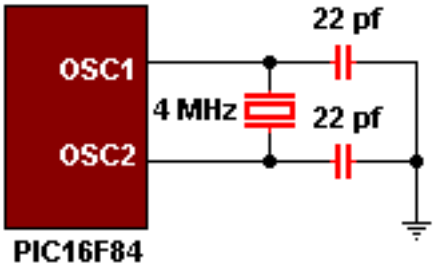
Es un circuito externo que le indica al micro la velocidad a la que debe trabajar. Este circuito, que se conoce como oscilador o reloj, es muy simple pero de vital importancia para el buen funcionamiento del sistema. El P1C16C84/F84 puede utilizar cuatro tipos de reloj diferentes. Estos tipos son:

- **RC.** Oscilador con resistencia y condensador.
- **XT.** Cristal.
- **HS.** Cristal de alta velocidad.

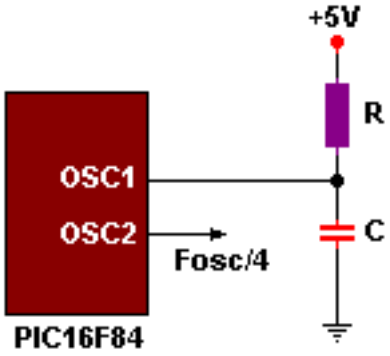
- **LP.** Cristal para baja frecuencia y bajo consumo de potencia.

En el momento de programar o "quemar" el microcontrolador se debe especificar que 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.

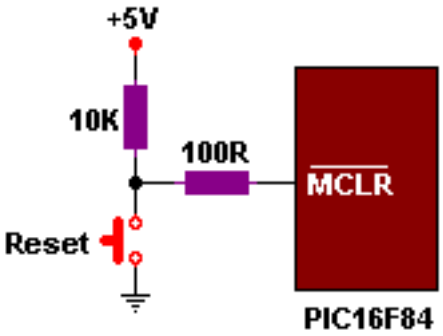


Según las recomendaciones de Microchip R puede tomar valores entre 5k y 100k, y C superior a 20pf.

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

:: Microcontroladores PIC - Capítulo 3

Estructura interna del Microcontrolador

Uffff...!!!, Ya se...!!!, tranquilo que ya comenzaremos con lo que estas esperando, antes debemos saber donde 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

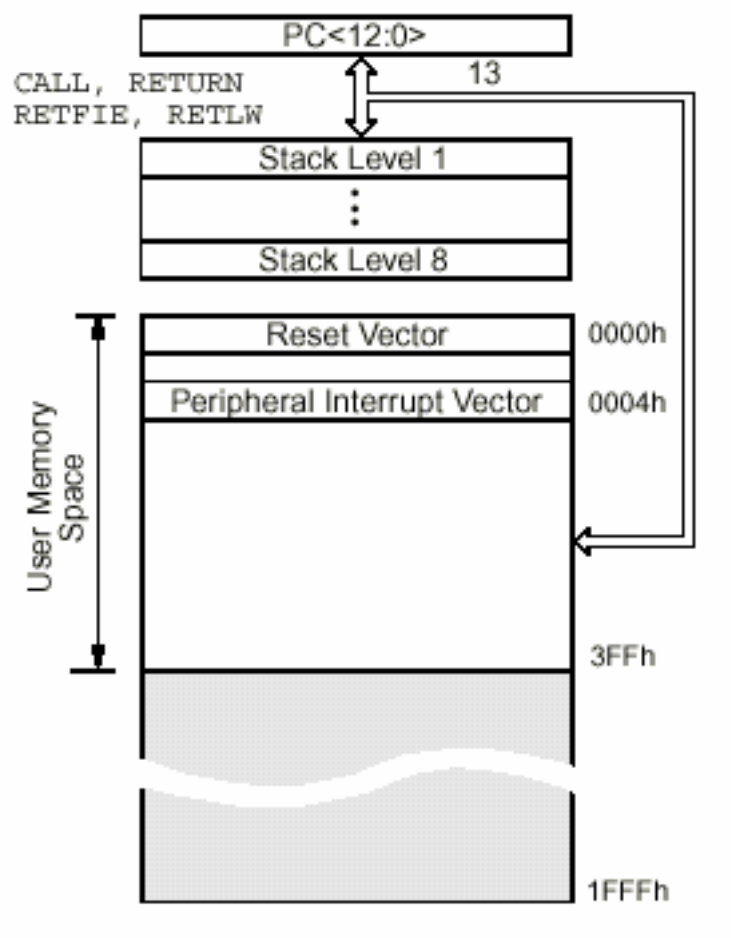
Es el elemento más importante del microcontrolador. Se encarga de direccionar la memoria de instrucciones, recibir el código OP de la instrucción en curso, decodificarlo y ejecutarlo, también realiza la búsqueda de los operandos y almacena el resultado.

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.

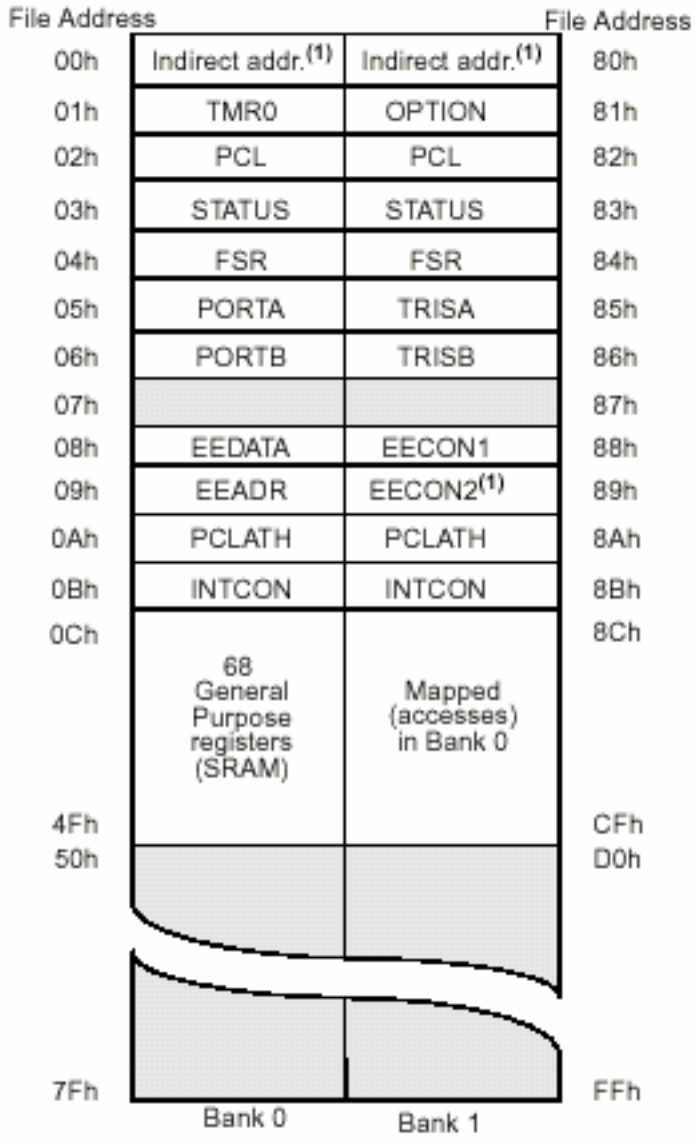
Algo que se debe tener en cuenta es la pila o Stack, que consta de 8 posiciones (*o niveles*), esto es como una pila de 8 platos el último en poner es el primero en sacar, si seguimos con este ejemplo, cada plato contiene la dirección y los datos de la instrucción que se está ejecutando, así cuando se efectúa una llamada (CALL) o una interrupción, el PC sabe donde 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)



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 y comenzamos con lo más emocionante.

:: Microcontroladores PIC - Capítulo 4

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 N° 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.

```
;--------------Encabezado------------

      list    p=16f84     ; usaremos el PIC 16f84
      radix   hex         ; y la numeración hexadecimal

;-----------mapa de memoria---------

estado  equ     0x03        ; Aquí le asignamos nombres a los
trisa   equ     0x05        ; registros indicando la posición
trisb   equ     0x06        ; en la que se encuentran

;-------Configuración de puertos-------

reset   org     0x00        ; origen del programa, aquí comienzaré
                            ; siempre que ocurra un reset
      goto    inicio      ; salto a "inicio"
      org     0x05        ; origen del código de programa
inicio  bsf     estado,5          ; pongo rp0 a 1 y paso al banco1
      movlw   b'11111'          ; cargo W con 11111
      movwf   trisa       ; y paso el valor a trisa
      movlw   b'00000000'   ; cargo W con 00000000
      movwf   trisb       ; y paso el valor a trisb
      bcf     estado,5          ; pongo rp0 a 0 y regreso al banco0

;----------------------------------------
      end             ; se acabó
;----------------------------------------
```

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 que hace cada línea de código.

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

```
;---------------Encabezado-------------

      list    p=16f84      ; usaremos el PIC 16f84
      radix   hex          ; y la numeración hexadecimal
```

Aquí le indicas al ensamblador para que microcontrolador estas codificando (PIC16F84). y cual 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.

```
;------------mapa de memoria---------

estado  equ    0x03        ; Aquí le asignamos nombres a los
trisa   equ    0x05        ; registros indicando la posición
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...

```
;-------Configuración de puertos-------

reset   org     0x00        ; origen del programa, aquí comenzaré
                            ; siempre que ocurra un reset
        goto    inicio      ; salto a "inicio"
        org     0x05        ; origen del código de programa
inicio  bsf     estado,5            ; pongo rp0 a 1 y paso al banco1
        movlw   b'11111'            ; cargo W con 11111
        movwf   trisa       ; y paso el valor a trisa
        movlw   b'00000000'     ; cargo W con 00000000
        movwf   trisb       ; y paso el valor a trisb
        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.

```
;----------------------------------------
        end             ; se acabó
;----------------------------------------
```

Fue suficiente por hoy...

:: Microcontroladores PIC - Capítulo 5

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

```
;---------------Encabezado-------------

      LIST   p=16f84
      radix  hex

;------------mapa de memoria---------

estado  equ    0x03       ; Haciendo asignaciones
TRISB   equ    0x06
ptob    equ    0x06

reg1    equ    0x0C       ; Estos 3 registros los utilizaré
reg2    equ    0x0D       ; para hacer el retardo
reg3    equ    0x0E

;-------Configuración de puertos-------

reset   org    0x00       ; origen del programa, aquí comenzará
                          ; siempre que ocurra un reset
      goto   inicio       ; salta a "inicio"
inicio  bsf    estado,5          ; pone rp0 a 1 y pasa al banco1
      movlw  b'00000000'   ; carga W con 00000000
      movwf  TRISB        ; y pasa el valor a trisb
      bcf    estado,5           ; pone rp0 a 0 y regresa al banco0

;----Aquí enciende y apaga el LED-----

ahora   bsf    ptob,0      ; pone un 1 en RB0 (enciende el LED)
      call   retardo      ; llama al retardo
```

```
      bcf    ptob,0      ; pone a 0 RB0 (apaga el LED)
      call   retardo     ; llama al retardo
      goto   ahora       ; repite todo de nuevo

;----------Rutina de Retardo----------

retardo movlw  10         ; Aquí se cargan los registros
      movwf  reg1         ; reg1, reg2 y reg3
                          ; con los valores 10, 20 y 30
tres   movlw  20          ; respectivamente
      movwf  reg2

dos    movlw  30
      movwf  reg3

uno    decfsz reg3,1      ; Aquí se comienza a decrementar
      goto   uno          ; Cuando reg3 llegue a 0
      decfsz reg2,1       ; le quitare 1 a reg2
      goto   dos          ; cuando reg2 llegue a 0
      decfsz reg1,1       ; le quitare 1 a reg1
      goto   tres         ; cuando reg1 llegue a 0
      retlw  00           ; regresare al lugar
                          ; de donde se hizo la llamada

;----------------------------------------
      end               ; se acabó
;----------------------------------------
```

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:

```
;----Aquí enciende y apaga el LED-----

ahora   bsf     ptob,0      ; pone un 1 en RB0 (enciende el LED)
        call    retardo     ; llama al retardo

        bcf     ptob,0      ; pone a 0 RB0 (apaga el LED)
        call    retardo     ; llama al retardo
        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 como se cargan los registros para el retardo.** Veamos el código...

```
;-----------Rutina de Retardo-----------

retardo movlw   10          ; Aquí se cargan los registros
        movwf   reg1        ; reg1, reg2 y reg3
                            ; con los valores 10, 20 y 30
tres    movlw   20          ; respectivamente
        movwf   reg2

dos     movlw   30
        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 como 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 ésto solo con fines didácticos así podrás comprenderlo mejor, ok?.

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

```
tres   movlw   20        ; respectivamente
       movwf   reg2

dos    movlw   30
       movwf   reg3

uno    decfsz  reg3,1     ; Aquí se comienza a decrementar
       goto    uno       ; Cuando reg3 llegue a 0
       decfsz  reg2,1     ; le quitare 1 a reg2
       goto    dos       ; cuando reg2 llegue a 0
       decfsz  reg1,1     ; le quitare 1 a reg1
       goto    tres      ; cuando reg1 llegue a 0
       retlw   00        ; regresare al lugar
                         ; 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.

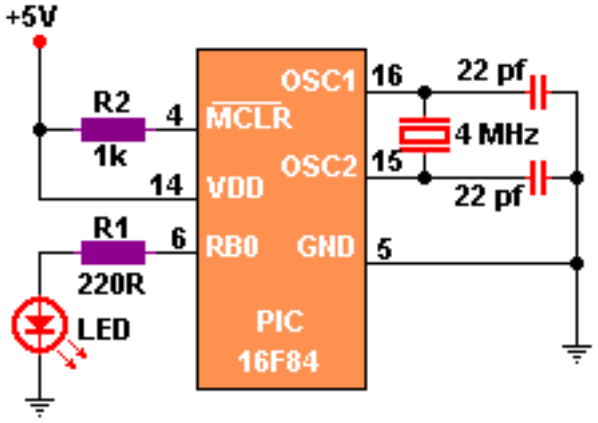
Cuando reg3 llegue a 0 decrementaré reg2 en una unidad, volveré a cargar reg3 y lo decrementaré nuevamente para recién restarle otra unidad a reg2, y así... hasta que reg2 se haga cero. Cuando eso ocurra decrementaré reg1 en una unidad, cargaré nuevamente reg2 y reg3, para luego decrementarlos de nuevo,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

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:

```
;----------------------------------------
     end              ; se acabó
;----------------------------------------
```

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...!!!

:: Microcontroladores PIC - Capítulo 6

Un poco de herramientas

Lo anterior estuvo bárbaro, pero dónde voy a escribir el código?, como se hace para ensamblar?, y cómo cargo mi programa en el PIC?, mmmmm... demasiadas preguntas, comencemos de cero...

Necesitaras:

- Un PC (con una 386 más que suficiente).
- El programa para editar tu código, que bien podría ser el **Edit** del DOS y asi generar tu archivo .asm
- Como ensamblador, yo utilizo es **Mpasm 2.15** y puedes bajarlo de www.microchip.com y con éste me basta para generar los archivos .hex
- El programa para grabar el PIC, Con el **Prog V.1.41** no tuve problemas y va perfecto con mi circuito grabador, puedes buscarlo en www.webelectronica.com.ar, ahora tambien disponible desde aquí
- Por supuesto necesitas el Hardware (circuito electrónico que se conecta al puerto de la PC) en el que insertarás el PIC para cargar el programa, hay muchísimos por la red, pero si quieres el que yo utilizo, **aquí** tienes el esquema eléctrico, el listado de componentes y la conexión al puerto paralelo, si ya lo quieres montado y listo para usar, contacta conmigo, que más puedo decir...

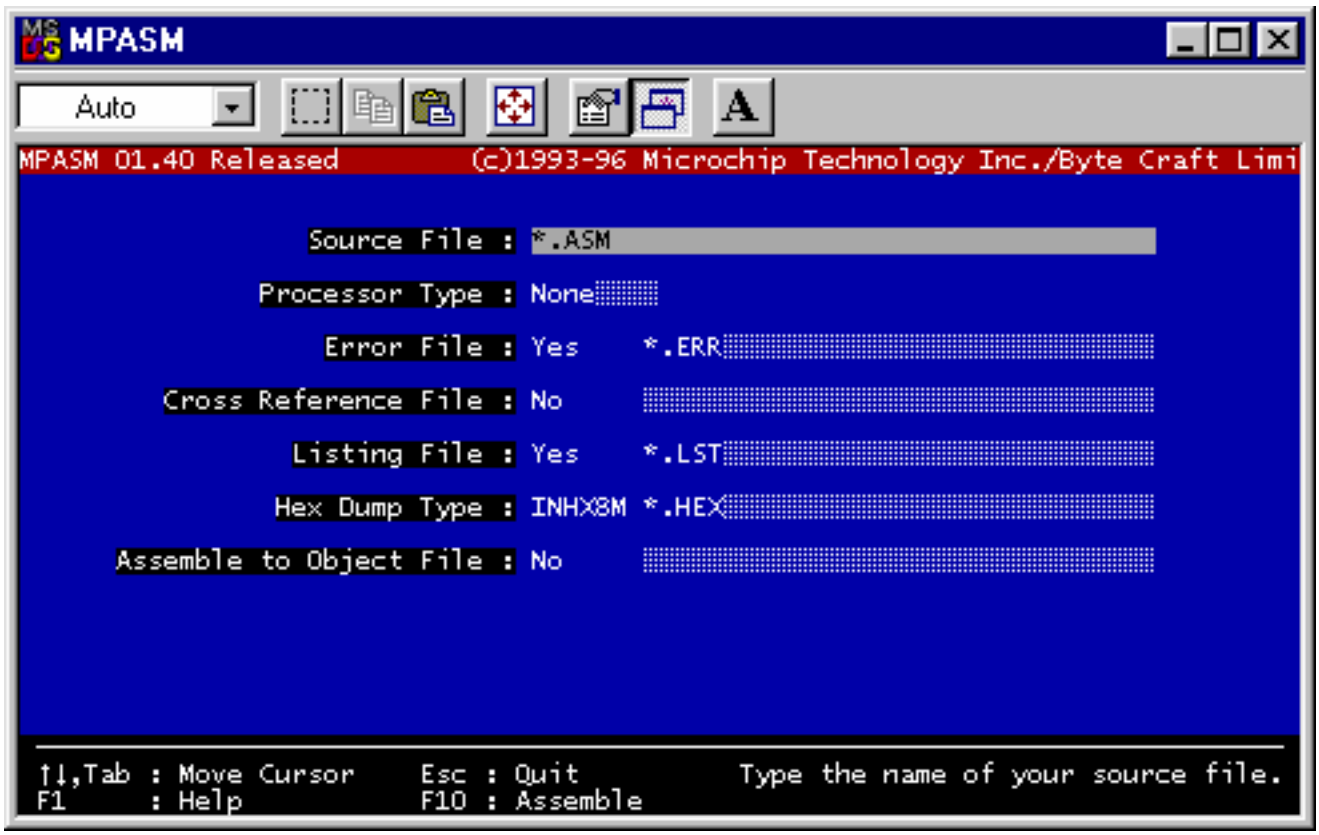
Ahora sí..., con todo esto y los programas instalados en la máquina, podemos comenzar...

Abre una ventana del DOS y apunta al directorio donde tienes todas tus herramientas, yo las puse en una carpeta llamada **tutor**, si haces lo que yo, te quedará algo así...

C:\tutor>edit

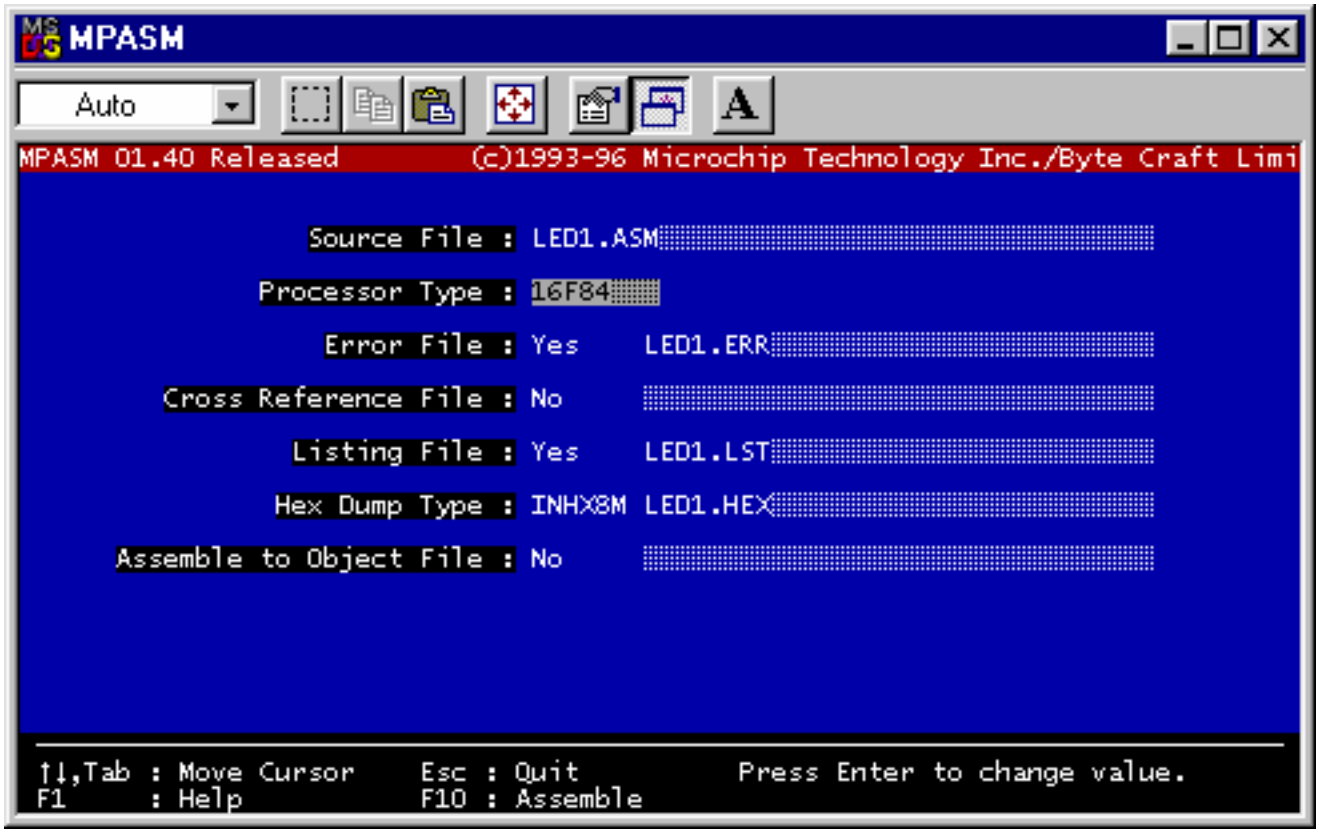
Una vez ahí puedes escribir tu código..., por último lo guardamos seleccionando el menú **Archivo --> Guardar como --> led1.asm** No olvides el .asm

Muy bien, ya tenemos **led1.asm**, sólo resta ensamblarlo. Entonces vamos por Mpasm, lo abres, y veras algo como esto...



En **Sourse File** presiona Enter para seleccionar el archivo a ensamblar

Haz lo mismo en **Processor Type** y busca el PIC16f84, que es el que usaremos, el resto lo dejas como está..., te debería quedar algo así...

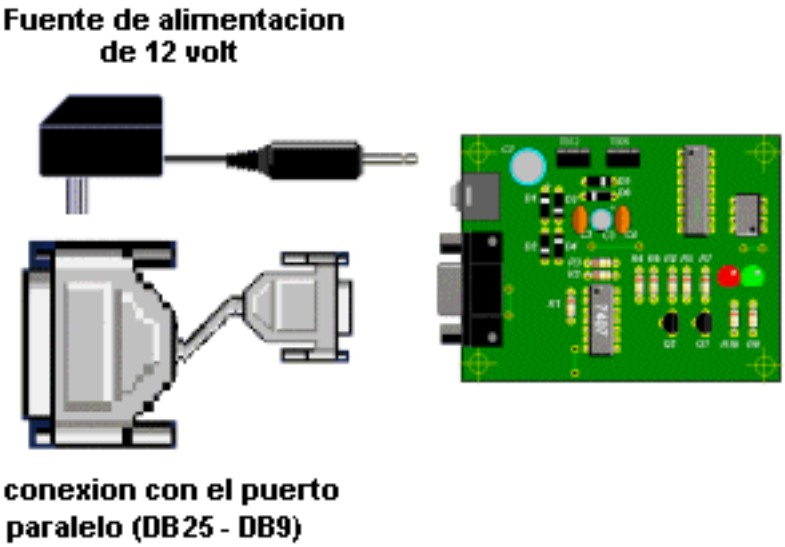


Ésto, generará el archivo **LED1.ERR, LED1.LST**, y **LED1.HEX**, este último es el que estamos buscando, los anteriores sirven para saber si se cometió algún error, si es así debes abrirlos (con el Bloc de Notas es suficiente) corregir los errores y ensamblar nuevamente.

Para ensamblar sólo debes presionar **F10** y verás la ventana de resultados

Que como es lógico... no cometí errores =P. Aunque por ahí apareció un Warning, que en realidad no es causa de falla en el código.

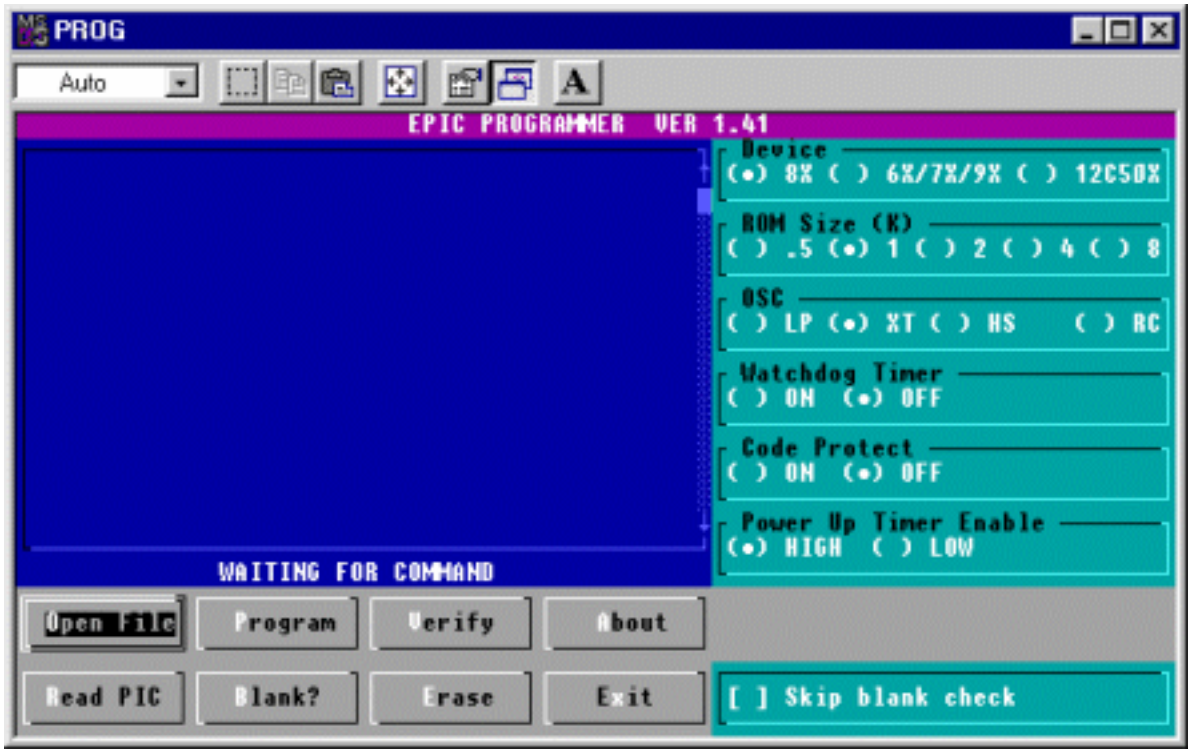
Bien, ya tenemos **LED1.HEX** y es el que cargaremos en el pic. Lo que viene es una imagen representativa del grabador de pic, con el cable que se conectará al puerto paralelo, y la fuente de alimentación externa. No coloques el PIC sin haber hecho las conexiones anteriores.



Ya puedes ejecutar el software de programación, abre **Prog.exe** y seguramente

.

se apagarán los LED's rojo y verde (si estaban encendidos...!). Te aparecerá la siguiente ventana



haciendo click en **Open File** seleccionas **LED1.HEX** y aquí está...

Colocas el PIC en el grabador, luego Seleccionas la opción **Program** y esperas a que te aparezca el mensaje **Programming Complete**

Ejemmmmm, perdón... este mensajito salió porque el pic ya tenía un programa grabado, bueno, no importa, como es regrabable, sólo le daremos a ok y listo, el nuevo programa borrará al anterior.

Ahora siiiiii...!!!, Si todo fue bien, ya puedes quitar el PIC del zócalo y llevarlo a tu circuito para ver como funciona.

Algo que me estaba quedando en el tintero son los fusibles de programación, como:

Device el tipo de chip, en nuestro caso el 8x;
Rom Sice la cantidad de memoria, en nuestro caso de 1k;
OSC el tipo de oscilador que utilizaremos, para nosotros un XT;
Watchdog Timer El perro guardián, que aquí no lo utilizamos;

Code protect para que nadie tenga acceso al código grabado en el PIC;
Power Up Timer temporizador de encendido. En el PIC16f84, funciona de modo invertido, por eso está en **LOW**. Para mayor detalle consulta **aquí**.

Ahora que ya estamos listos y preparados con todo el soft y el hard necesario, lo único que necesitamos es complicarnos un poco mas las cosas, allá vamos.....

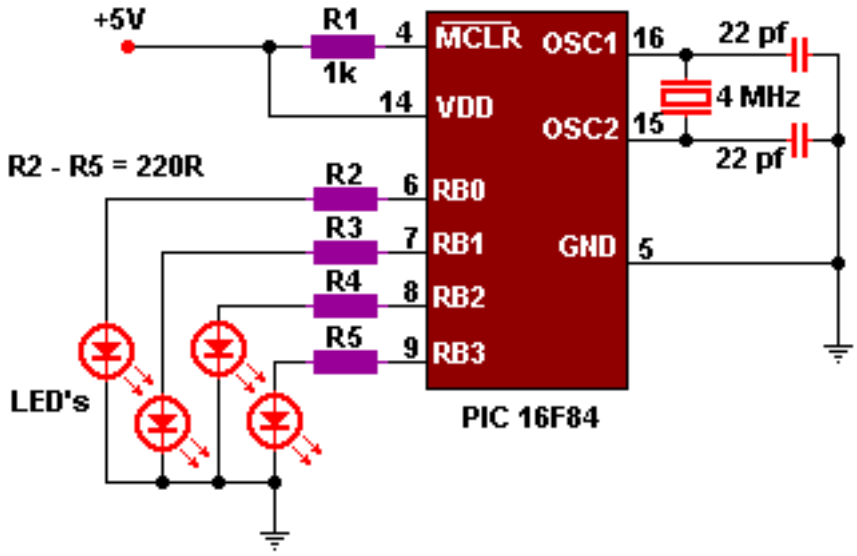
:: Microcontroladores PIC - Capítulo 7

Antes de empezar

Si bien nos vamos a complicar con las siguientes lecciones, sería bueno que consultes el Set de instrucciones, así será mas 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.

```
;--------------Encabezado------------

      LIST   p=16f84
      radix  hex

;-----------mapa de memoria---------

estado  equ    0x03       ; Haciendo asignaciones
TRISB   equ    0x06
ptob    equ    0x06
rotar   equ    0x0A       ; para desplazar el dato

reg1    equ    0x0C       ; para hacer el retardo
reg2    equ    0x0D
reg3    equ    0x0E

;-------Configuración de puertos-------

reset   org    0x00

      goto   inicio      ; salta a "inicio"
      org    0x05        ; aquí comienza el programa
inicio  bsf    estado,5          ; configurando el puerto B
      movlw  b'00000000'
      movwf  TRISB
      bcf    estado,5

;----Realiza la secuencia de LED's-----

ahora   movlw  0x01       ; carga W con 00000001
      movwf  rotar       ; lo pasa al registro rotar

rotando movf   rotar,0     ; pasa el contenido de rotar a W
      movwf  ptob        ; y de allí al puerto B
                       ; encendiendo el LED correspondiente
      call   retardo     ; llama a retardo
      rlf    rotar,1     ; desplaza un bit el contenido
                       ; de rotar y lo guarda.
      btfss  rotar,4     ; prueba si se activa el 5to. bit
                       ; si es así saltea una línea
      goto   rotando     ; sino, sigue rotando
      goto   ahora       ; repite todo de nuevo

;----------Rutina de Retardo----------

retardo movlw  10         ; Carga los registros
      movwf  reg1        ; reg1, reg2 y reg3
                       ; con los valores 10, 20 y 30
tres    movlw  20         ; respectivamente
      movwf  reg2

dos     movlw  30
      movwf  reg3
```

```
uno    decfsz  reg3,1      ; Comienza a decrementar
       goto    uno         ; cuando termine
       decfsz  reg2,1      ; regresará a la siguiente
       goto    dos         ; línea de código
       decfsz  reg1,1      ; de donde fue llamado
       goto    tres
       retlw   00

;----------------------------------------
       end              ; The End
;----------------------------------------
```

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

```
;----Realiza la secuencia de LED's-----

ahora   movlw   0x01        ; carga W con 00000001
       movwf   rotar        ; lo pasa al registro rotar

rotando movf    rotar,0     ; pasa el contenido de rotar a W
       movwf   ptob         ; y de allí al puerto B
                            ; encendiendo el LED correspondiente
       call    retardo      ; llama a retardo
       rlf     rotar,1      ; desplaza un bit el contenido
                            ; de rotar y lo guarda.
       btfss   rotar,4      ; prueba si se activa el 5to. bit
                            ; si es así saltea una línea
       goto    rotando      ; sino, sigue rotando
       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 se lo que estas pensando ¿como que el 5to. si aparece el 4?, bueno, es por que no estas siguiendo el tutorial, recuerda que el primer bit está en la posición 0 y por ende, el 5to. esta en la posición 4 ¿ahora esta 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**?, Siiiiii, 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.

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...

Fue todo por hoy, ahora nos tomamos un descanso y vemos que otras herramientas tenemos...

:: Microcontroladores PIC - Capítulo 8

Más herramientas

Programadores:

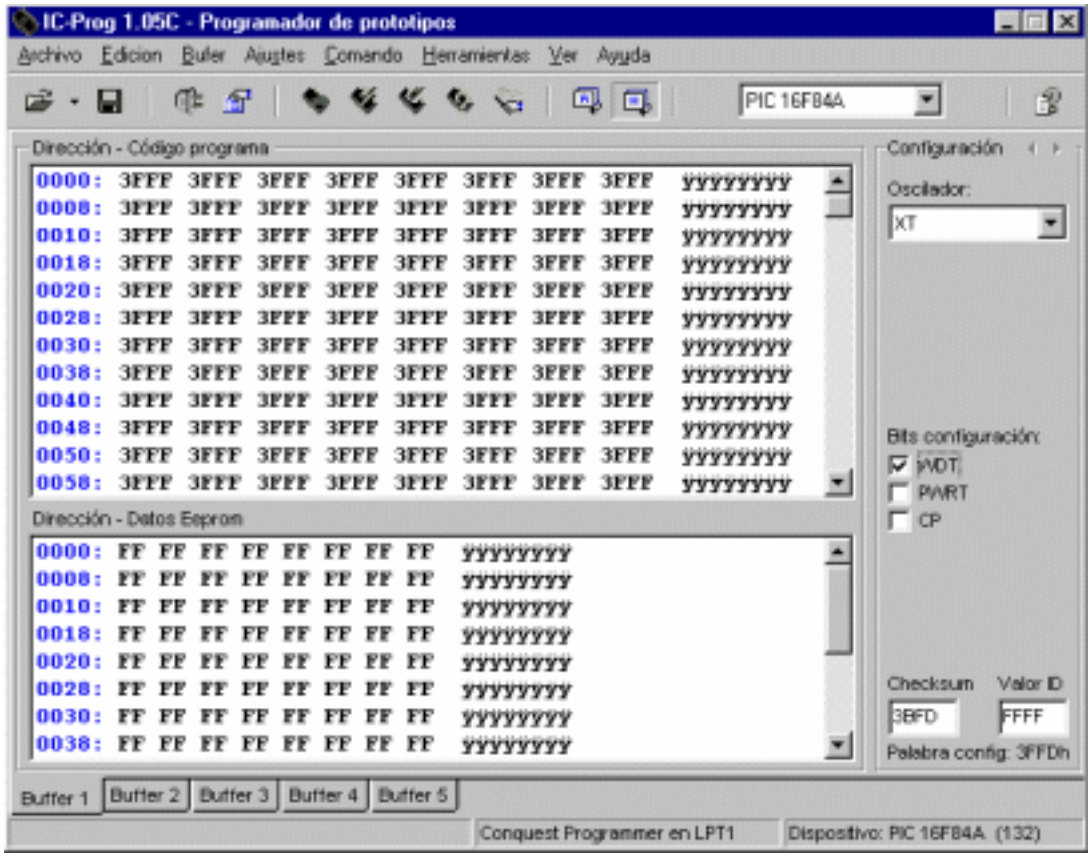
Hay muchos programadores dando vueltas por la Red, uno de los que me llamó la atención fue el Programador PIPO2 de José Manuel García, por su sencilles, por la cantidad de PIC's que soporta, por que no requiere fuente de alimentación, su bajo costo, y por supuesto, por que está listo para armar, jejeje.

El programador anterior no es profesional pero para lo que queremos hacer, se lleva todas las de ganar. Si quieres algo más, como un programador semi-profesional, está su pariente el Programador PP2, tambien de José Manuel García, aunque requiere unos cuantos $$$$ más, que no creo sea la gran diferencia, por cierto...

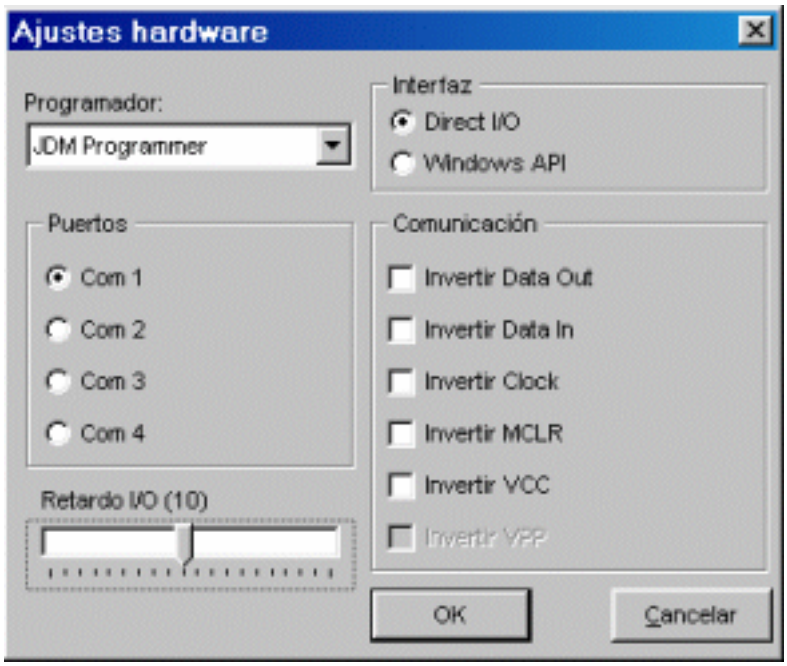
El autor, recomienda su utilización con el IC-Prog, un programa bien a lo windows, es más puedes seleccionar el Idioma, pero para utilizarlo con el programador debes hacerle algunos ajustes. Bueno el mismo autor te indica cómo.

Otra de las características de este programa es que puedes desensamblar un archivo .hex y ver su código fuente, para luego modificarlo o hacer lo que quieras con él. **oye, respeta los derechos de autor, ok?** :-P

Aquí tienes una imagen de su entorno de trabajo



Y aquí el cuadro de diálogo en que debes realizar los ajustes para el hardware que estés utilizando.



Eso por ahora, luego veremos más herramientas...

:: Microcontroladores PIC - Capítulo 9

Señales de Entrada

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

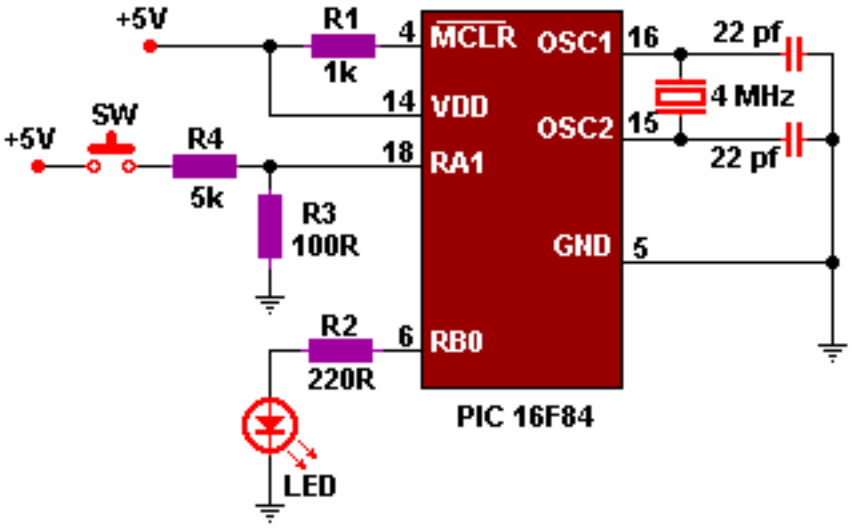
Por ejemplo, si le ordenas que vigile un pulsador, el muy pillo lo hará, y cuando alguien lo active le dará su merecido, jejejeje

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 que 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.

```
;--------------Encabezado------------
      List    p=16F84
      radix   hex
;------------mapa de memoria---------

STATUS  EQU     03          ; status esta en la dirección 03
TRISA   EQU     05
PORTA   EQU     05          ; el puerto a
TRISB   EQU     06
PORTB   EQU     06          ; el puerto b

cont    EQU     0A

;-------Configuración de puertos-------

reset   ORG     0
      GOTO    inicio
      ORG     5

inicio  BSF     STATUS,5        ; configurando puertos
      MOVLW   0x02        ; carga w con 0000 0010
      MOVWF   TRISA       ; ahora RA1 es entrada
      MOVLW   0x00        ; carga w con 0000 0000
      MOVWF   TRISB       ; y el puerto B salida
      BCF     STATUS,5

      CLRF    PORTB       ; limpio el puerto B
      CLRF    cont        ; limpio el contador
      BSF     cont,0      ; pongo el contador a 1

;----------------------------------------

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

led     BTFSC   cont,0      ; si el contador está a 1
      GOTO    on_led      ; lo atiendo en "on_led"
      BCF     PORTB,0     ; sino, apago el LED
      BSF     cont,0      ; pongo el contador a 1
      GOTO    libre       ; y espero que suelten el pulsador

on_led  BSF     PORTB,0     ; enciendo el LED
      CLRF    cont        ; pongo el contador a 0
libre   BTFSC   PORTA,1     ; si no sueltan el pulsador
      GOTO    libre       ; me quedaré a esperar

      RETURN              ; si lo sueltan regreso
                        ; a testear nuevamente

;----------------------------------------
      END
```

```
;----------------------------------------
```

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.

```
      CLRF    PORTB       ; limpio el puerto B
      CLRF    cont        ; limpio el contador
      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...

```
;----------------------------------------
test    BTFSC   PORTA,1     ; si alguien presiona
        CALL    led         ; voy a la etiqueta "led"
        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 subrrutina **led**, allí vamos...

```
led    BTFSC   cont,0      ; si el contador está a 1
       GOTO    on_led      ; lo atiendo en "on_led"
       BCF     PORTB,0     ; sino, apago el LED
       BSF     cont,0      ; pongo el contador a 1
       GOTO    libre       ; y espero que suelten el pulsador

on_led BSF     PORTB,0     ; enciendo el LED
       CLRF    cont        ; pongo el contador a 0
libre  BTFSC   PORTA,1     ; si no sueltan el pulsador
       GOTO    libre       ; me quedaré a esperar

       RETURN             ; si lo sueltan regreso
                          ; 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 esta 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. En breve hablaremos del famoso MPLAB de Microchip, por lo pronto, trata de conseguirlo...

Si llegaste con éxito hasta aquí, ten por seguro que puedes hacer cosas más complejas, a las que nos dedicaremos de ahora en más...

Saludos... y hasta la próxima...!!!

R-Luis...

:: Microcontroladores PIC - Set de Instrucciones

Es importante que tengas en claro las notaciones que deberás tomar en cuenta para poder interpretarlas, no son muchas.

NOTACION PARA NUMEROS

- **Decimal** : D'100' ó .100
- **Hexadecimal** : H'64' ó 0x64 ó 64
- **Octal** : O'144'
- **Binario** : B'01101100'
- **ASCII** : A'C' ó 'C'

NOTACION PARA REGISTROS Y LITERALES

- **w** : Registro W, similar al acumulador, es el registro de trabajo.
- **f** : Campo de 5 bits (fffff), contiene la dirección del banco de registros, que ocupa el banco 0 del área de datos. **Direcciona uno de esos registros.**
- **k** : Representa una constante de 8 bits.
- **d** : Bit del código OP de la instrucción. Selecciona el destino donde se guarda el resultado de una operación. Si d=0, el destino es W, y si d=1 el destino es f.
- **b** : Determina la posición de un bit dentro de un registro de 8 bits, (*o sea, tomará valores entre 0 y 7*)

SIMBOLOS

- **[ ]** : Opciones.
- **( )** : Contenido.
- **=>** : Se asigna a ...
- **< >** : Campo de bits de un registro.
- **E** : Pertenece al conjunto ...
- **Label** : Nombre de la etiqueta.
- **TOS** : Cima de la pila con 8 niveles en la gama media.
- **PC** : Contador de programa que direcciona la memoria de instrucciones.

FLAGS

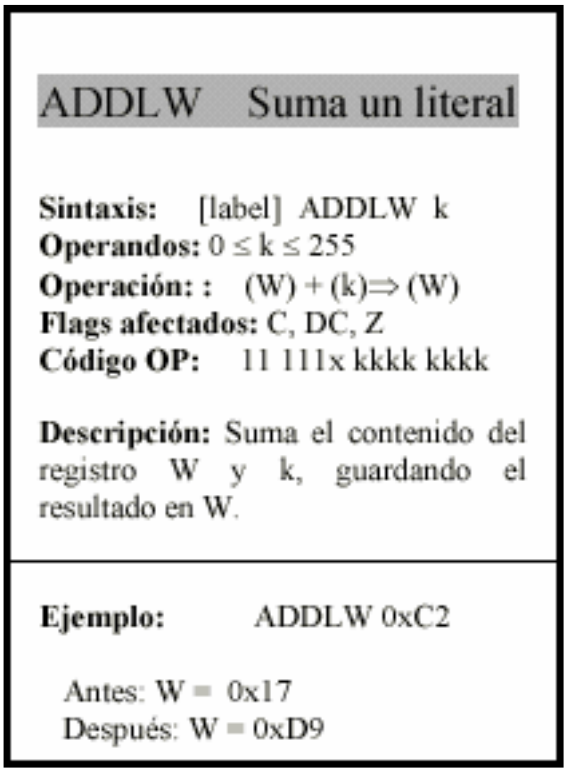
Los Flags o banderas son marcadores, representados por bits dentro del

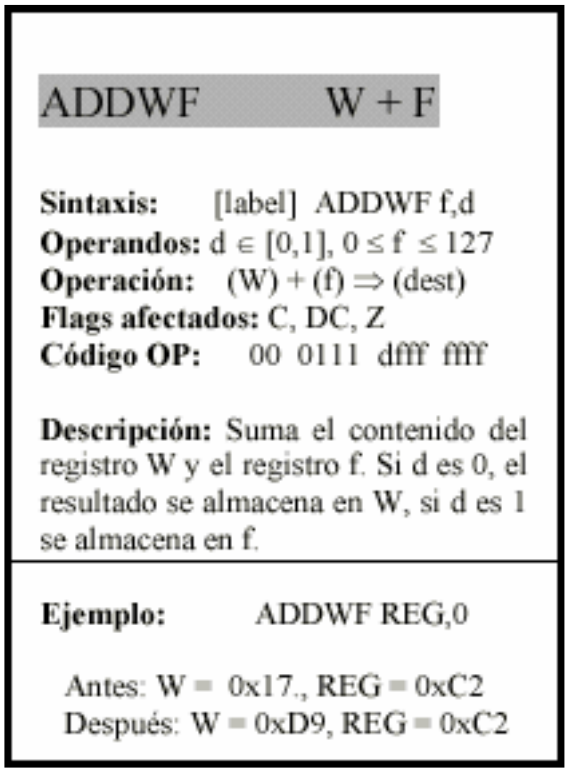
registro STATUS, y son:

- **Z** : Flag de cero, se pone a 1 cuando una operación lógica o aritmética da 0 (cero)como resultado. En cualquier otro caso se pone a 0.
- **C** : Flag de Carry, se pone a 1 cuando la operación que le afecta sobrepasa el nivel de representación del procesador, en nuestro caso es de 8 BIT's , de esta manera si sumamos a 0b11111111 un 0b00000011 el resultado sería 0b00000010 y el BIT de Carry pasaría a 1.
- **DC** : Flag de carry del nibble inferior, este se comporta igual que el BIT de Carry, solo que el límite de representación son los 4 bits inferiores, de esta manera si tenemos 0b00001111 y sumamos 0b00000111, el resultado será 0b00010110 y el BIT de DC se pone a 1, el BIT de Carry estará a 0 al no superarse los 8 bits y el de Z a 0 al ser el número diferente de 0.

No te preocupes si te quedan dudas respecto a los FLAGS, éstas se aclararán a medida que vayas avanzando en el tutorial.

Ahora si, ya podemos empezar con el set de instrucciones:





ANDLW W AND literal

Sintaxis: [label] ANDLW k
Operandos: $0 \le k \le 255$
Operación: : (W) AND (k)⇒ (W)
Flags afectados: Z
Código OP: 11 1001 kkkk kkkk

Descripción: Realiza la operación lógica AND entre el contenido del registro W y k, guardando el resultado en W.

Ejemplo: ADDLW 0xC2

Antes: W = 0x17
Después: W = 0xD9

ANDWF W AND F

Sintaxis: [label] ANDWF f,d
Operandos: $d \in [0,1]$, $0 \le f \le 127$
Operación: (W) AND (f) ⇒ (dest)
Flags afectados: Z
Código OP: 00 0101 dfff ffff

Descripción: Realiza la operación lógica AND entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : ANDWF REG,0

Antes: W = 0x17., REG = 0xC2
Después: W = 0x17, REG = 0x02

BCF Borra un bit

Sintaxis: [label] BCF f,b
Operandos: $0 \le f \le 127$, $0 \le b \le 7$
Operación: : 0 ⇒ (f<b>)
Flags afectados: Ninguno
Código OP: 01 00bb bfff ffff

Descripción: Borra el bit b del registro f

Ejemplo: : BCF REG,7

Antes: REG = 0xC7
Después: REG = 0x47

BSF Activa un bit

Sintaxis: [label] BSF f,b
Operandos: $0 \le f \le 127$, , $0 \le b \le 7$
Operación: 1 ⇒ (f<b>)
Flags afectados: Ninguno
Código OP: 01 01bb bfff ffff

Descripción: Activa el bit b del registro f

Ejemplo: : BSF REG,7

Antes: REG = 0x0A
Después: REG = 0x8A

BTFSC Test de bit y salto

Sintaxis: [label] BTFSC f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: Salto si (f<b>) = 0
Flags afectados: Ninguno
Código OP: 01 10bb bfff ffff

Descripción: Si el bit b del registro f es 0, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj.

Ejemplo:
```
        BTFSC REG,6
        GOTO NO_ES_0
SI_ES_0 Instrucción
NO_ES_0 Instrucción
```

BTFSS Test de bit y salto

Sintaxis: [label] BTFSS f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: Salto si (f<b>) = 1
Flags afectados: Ninguno
Código OP: 01 11bb bfff ffff

Descripción: Si el bit b del registro f es 1, se salta una instrucción y se continúa con la ejecución. En caso de salto, ocupará dos ciclos de reloj.

Ejemplo:
```
        BTFSS REG,6
        GOTO NO_ES_0
SI_ES_0 Instrucción
NO_ES_0 Instrucción
```

CALL Salto a subrutina

Sintaxis: [label] CALL k
Operandos: 0 ≤ k ≤ 2047
Operación: PC ⇒ Pila; k ⇒ PC
Flags afectados: Ninguno
Código OP: 10 0kkk kkkk kkkk

Descripción: Salto a una subrutina. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj.

Ejemplo:
```
ORIGEN CALL DESTINO

Antes:  PC = ORIGEN
Después: PC = DESTINO
```

CLRF Borra un registro

Sintaxis: [label] CLRF f
Operandos: 0 ≤ f ≤ 127
Operación: : 0x00 ⇒ (f), 1 ⇒ Z
Flags afectados: Z
Código OP: 00 0001 1fff ffff

Descripción: El registro f se carga con 0x00. El flag Z se activa.

Ejemplo:
```
        CLRF REG

Antes: REG = 0x5A
Después: REG = 0x00, Z = 1
```

CLRW Borra el registro W

Sintaxis: [label] CLRW
Operandos: Ninguno
Operación: : 0x00 ⇒ W, 1 ⇒ Z
Flags afectados: Z
Código OP: 00 0001 0xxx xxxx

Descripción: El registro de trabajo W se carga con 0x00. El flag Z se activa.

Ejemplo: : CLRW

Antes: W = 0x5A
Después: W = 0x00, Z = 1

CLRWDT Borra el WDT

Sintaxis: [label] CLRWDT
Operandos: Ninguno
Operación: 0x00 ⇒ WDT, 1 ⇒ /TO
1 ⇒ /PD
Flags afectados: /TO, /PD
Código OP: 00 0000 0110 0100
Descripción: Esta instrucción borra tanto el WDT como su preescaler. Los bits /TO y /PD del registro de estado se ponen a 1.

Ejemplo: : CLRWDT
Después: Contador WDT = 0,
Preescales WDT = 0,
/TO = 1, /PD = 1

COMF Complemento de f

Sintaxis: [label] COMF f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: : (/ f), 1 ⇒ (dest)
Flags afectados: Z
Código OP: 00 1001 dfff ffff

Descripción: El registro f es complementado. El flag Z se activa si el resultado es 0. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f..

Ejemplo: : COMF REG,0

Antes: REG = 0x13
Después: REG = 0x13, W = 0XEC

DECF Decremento de f

Sintaxis: [label] DECF f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: : (f) – 1 ⇒ (dest)
Flags afectados: Z
Código OP: 00 0011 dfff ffff

Descripción: Decrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : DECF CONT,1

Antes: CONT = 0x01, Z = 0
Después: CONT = 0x00, Z = 1

DECFSZ Decremento y salto

Sintaxis: [label] DECFSZ f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: (f) -1 ⇒ d; Salto si R=0
Flags afectados: Ninguno
Código OP: 00 1011 dfff ffff

Descripción: Decrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.

Ejemplo:
```
DECFSC REG,0
GOTO NO_ES_0
SI_ES_0 Instrucción
NO_ES_0 Salta instrucción anterior
```

GOTO Salto incondicional

Sintaxis: [label] GOTO k
Operandos: 0 ≤ k ≤ 2047
Operación: k ⇒ PC <8:0>
Flags afectados: Ninguno
Código OP: 10 1kkk kkkk kkkk

Descripción: Se trata de un salto incondicional. La parte baja de k se carga en PCL, y la alta en PCLATCH. Ocupa 2 ciclos de reloj.

Ejemplo: ORIGEN GOTO DESTINO

Antes: PC = ORIGEN
Después: PC = DESTINO

INCF Incremento de f

Sintaxis: [label] INCF f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: : (f) + 1 ⇒ (dest)
Flags afectados: Z
Código OP: 00 1010 dfff ffff

Descripción: Incrementa en 1 el contenido de f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: :
```
INCF CONT,1
```

Antes: CONT = 0xFF, Z = 0
Después: CONT = 0x00, Z = 1

INCFSZ Incremento y salto

Sintaxis: [label] INCFSZ f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: (f) -1 ⇒ d; Salto si R=0
Flags afectados: Ninguno
Código OP: 00 1111 dfff ffff

Descripción: Incrementa el contenido del registro f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Si la resta es 0 salta la siguiente instrucción, en cuyo caso costaría 2 ciclos.

Ejemplo:
```
INCFSC REG,0
GOTO NO_ES_0
SI_ES_0 Instrucción
NO_ES_0 Salta instrucción anterior
```

IORLW W OR literal

Sintaxis: [label] IORLW k
Operandos: $0 \leq k \leq 255$
Operación: : (W) OR (k) $\Rightarrow$ (W)
Flags afectados: Z
Código OP: 11 1000 kkkk kkkk

Descripción: Se realiza la operación lógica OR entre el contenido del registro W y k, guardando el resultado en W.

Ejemplo: : IORLW 0x35

Antes: W = 0x9A
Después: W = 0xBF

IORWF W AND F

Sintaxis: [label] IORWF f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 127$
Operación: (W) OR (f) $\Rightarrow$ (dest)
Flags afectados: Z
Código OP: 00 0100 dfff ffff

Descripción: Realiza la operación lógica OR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : IORWF REG,0

Antes: W = 0x91, REG = 0x13
Después: W = 0x93, REG = 0x13

MOVLW Cargar literal en W

Sintaxis: [label] MOVLW f
Operandos: $0 \leq f \leq 255$
Operación: (k) $\Rightarrow$ (W)
Flags afectados: Ninguno
Código OP: 11 00xx kkkk kkkk

Descripción: El literal k pasa al registro W.

Ejemplo: MOVLW 0x5A

Después: REG = 0x4F, W = 0x5A

MOVF Mover a f

Sintaxis: [label] MOVF f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 127$
Operación: (f) $\Rightarrow$ (dest)
Flags afectados: Z
Código OP: 00 1000 dfff ffff
Descripción: El contenido del registro f se mueve al destino d. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f. Permite verificar el registro, puesto que afecta a Z.

Ejemplo: MOVF REG,0

Después: W = REG

MOVWF Mover a f

Sintaxis: [label] MOVWF f
Operandos: $0 \leq f \leq 127$
Operación: $W \Rightarrow (f)$
Flags afectados: Ninguno
Código OP: 00 0000 1fff ffff

Descripción: El contenido del registro W pasa el registro f.

Ejemplo: MOVWF REG,0

Antes: REG = 0xFF, W = 0x4F
Después: REG = 0x4F, W = 0x4F

NOP No operar

Sintaxis: [label] NOP
Operandos: Ninguno
Operación: No operar
Flags afectados: Ninguno
Código OP: 00 0000 0xx0 0000

Descripción: No realiza operación alguna. En realidad consume un ciclo de instrucción sin hacer nada.

Ejemplo: : CLRWDT

Después: Contador WDT = 0,
Preescales WDT = 0,
/TO = 1, /PD = 1

RETFIE Retorno de interrup.

Sintaxis: [label] RETFIE
Operandos: Ninguno
Operación: : $1 \Rightarrow$ GIE; TOS$\Rightarrow$PC
Flags afectados: Ninguno
Código OP: 00 0000 0000 1001

Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos. Las interrupciones vuelven a ser habilitadas.

Ejemplo: : RETFIE

Después: PC = dirección de retorno
GIE = 1

RETLW Retorno, carga W

Sintaxis: [label] RETLW k
Operandos: $0 \leq k \leq 255$
Operación: : (k)$\Rightarrow$ (W); TOS$\Rightarrow$PC
Flags afectados: Ninguno
Código OP: 11 01xx kkkk kkkk

Descripción: El registro W se carga con la constante k. El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.

Ejemplo: : RETLW 0x37

Después: PC = dirección de retorno
W = 0x37

RETURN Retorno de rutina

Sintaxis: [label] RETURN
Operandos: Ninguno
Operación: : TOS ⇒ PC
Flags afectados: Ninguno
Código OP: 00 0000 0000 1000

Descripción: El PC se carga con el contenido de la cima de la pila (TOS): dirección de retorno. Consume 2 ciclos.

Ejemplo: : RETURN

Después: PC = dirección de retorno

RLF Rota f a la izquierda

Sintaxis: [label] RLF f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: Rotación a la izquierda
Flags afectados: C
Código OP: 00 1101 dfff ffff

Descripción: El contenido de f se rota a la izquierda. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: RRF REG,0

Antes: REG = 1110 0110, C = 0
Después: REG = 1110 0110,
W = 1100 1100, C = 1

RRF Rota f a la derecha

Sintaxis: [label] RRF f,d
Operandos: d ∈ [0,1], 0 ≤ f ≤ 127
Operación: Rotación a la derecha
Flags afectados: C
Código OP: 00 1100 dfff ffff

Descripción: El contenido de f se rota a la derecha. El bit de menos peso de f pasa al carry (C), y el carry se coloca en el de mayor peso. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: RRF REG,0

Antes: REG = 1110 0110, C = 1
Después: REG = 1110 0110,
W = 01110 0011, C = 0

SLEEP Modo bajo consumo

Sintaxis: [label] SLEEP
Operandos: Ninguno
Operación: 0x00⇒WDT, 1 ⇒ / TO
0 ⇒ WDT Preescaler, 0 ⇒ / PD
Flags afectados: / PD, / TO
Código OP: 00 0000 0110 0011

Descripción: El bit de energía se pone a 0, y a 1 el de descanso. El WDT y su preescaler se borran. El micro para el oscilador, llendo al modo "durmiente".

Ejemplo: : SLEEP

Preescales WDT = 0,
/TO = 1, /PD = 1

SUBLW Resta Literal - W

Sintaxis: [label] SUBLW k
Operandos: $0 \le k \le 255$
Operación: (k) - (W) ⇒ (W)
Flags afectados: Z, C, DC
Código OP: 11 110x kkkk kkkk
Descripción: Mediante el método del complemento a dos el contenido de W es restado al literal. El resultado se almacena en W.

Ejemplos: SUBLW 0x02

Antes:W=1,C=?. Después: W=1, C=1
Antes:W=2,C=?. Después: W=0, C=1
Antes:W=3,C=?.Después:W=FF,C=0
(El resultado es negativo)

SUBWF Resta f – W

Sintaxis: [label] SUBWF f,d
Operandos: $d \in [0,1]$, $0 \le f \le 127$
Operación: (f) – (W)⇒ (dest)
Flags afectados: C, DC, Z
Código OP: 00 0010 dfff ffff
Descripción: Mediante el método del complemento a dos el contenido de W es restado al de f. . Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplos: SUBWF REG,1
Antes: REG = 0x03, W = 0x02, C = ?
Después:REG=0x01, W = 0x4F, C=1
Antes: REG = 0x02, W = 0x02, C = ?
Después:REG=0x00, W =0x02, C= 1
Antes: REG= 0x01, W= 0x02, C= ?
Después:REG=0xFF, W=0x02, C= 0
(Resultado negativo)

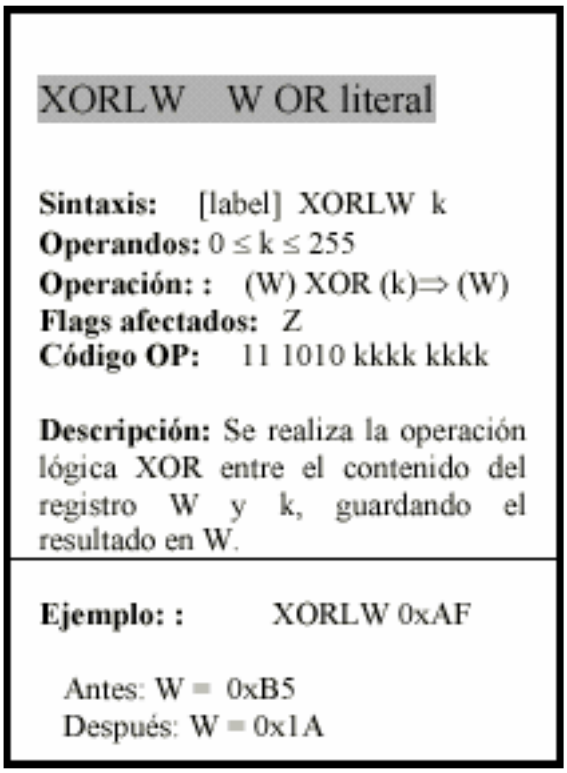
SWAPF Intercambio de f

Sintaxis: [label] SWAPF f,d
Operandos: $d \in [0,1]$, $0 \le f \le 127$
Operación: : (f <3: 0>)⇔ (f <7:4>)
Flags afectados: Ninguno
Código OP: 00 1110 dfff ffff

Descripción: Los 4 bits de más peso y los 4 de menos son intercambiados. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : SWAPF REG,0

Antes: REG = 0xA5
Después: REG = 0xA5, W = 0x5A



XORLW W OR literal

Sintaxis: [label] XORLW k
Operandos: $0 \leq k \leq 255$
Operación: : (W) XOR (k)⇒ (W)
Flags afectados: Z
Código OP: 11 1010 kkkk kkkk

Descripción: Se realiza la operación lógica XOR entre el contenido del registro W y k, guardando el resultado en W.

Ejemplo: : XORLW 0xAF

Antes: W = 0xB5
Después: W = 0x1A

XORWF W AND F

Sintaxis: [label] XORWF f,d
Operandos: $d \in [0,1]$, $0 \leq f \leq 127$
Operación: (W) XOR (f) ⇒ (dest)
Flags afectados: Z
Código OP: 00 0110 dfff ffff

Descripción: Realiza la operación lógica XOR entre los registros W y f. Si d es 0, el resultado se almacena en W, si d es 1 se almacena en f.

Ejemplo: : XORWF REG,0

Antes: W = 0xB5, REG = 0xAF
Después: W = 0xB5, REG = 0x1A

Ya veo que quieres el resumen de instrucciones, bueno **aquí** lo tienes...

:: Microcontroladores PIC - Capítulo 2

En este resumen las instrucciones están clasificadas según su operación sea orientada a registros, a bits, o a literales y de control:

Para que no te confundas...!!!. En esta primera tabla aparecen **W, f** y **d**. Recuerda que...

- **W** : es el registro de trabajo y almacena datos de forma momentánea
- **f** : es la dirección de un registro, si es llamada apunta al contenido de ese registro
- **d** : es el destino donde se guarda el resultado de una operación, si es **1** se guarda en el registro f, y si es **0** en W.

OPERACIONES ORIENTADAS A REGISTROS			
Nemotécnicos		**Operación**	**Estados afectados**
ADDWF	f,d	Sumar W y f	C,DC,Z
ANDWF	f,d	AND entre W y f	Z
CLRF	f	Limpiar f	Z
CLRW		Limpiar W	Z
COMF	f,d	Complementar f	Z
DECF	f,d	Decrementar f	Z
DECFSZ	f,d	Decrementar f, saltar si cero	
INCF	f,d	Incrementar f	Z
INCFSZ	f,d	Incrementar f, saltar si cero	
IORWF	f,d	OR entre W y f	Z
MOVF	f,d	Mover f	Z
MOVWF	f	Mover W a F	
NOP		No Operación	
RLF	f,d	Rotar a la izquierda a través del carry	C
RRF	f,d	Rotar a la derecha a través del carry	C
SUBWF	f,d	Restar W de f	C,DC,Z
SWAPF	f,d	Intercambiar nibbles de f	
XORWF	f,d	OR exclusiva entre W y f	Z

En este otro cuadro, a demás de **f** aparece **b**. que vendría a ser uno de los 8 bits del registro **f**

OPERACIONES ORIENTADAS A BITS			
Nemotécnicos		**Operación**	**Estados afectados**
BCF	f,b	Limpiar bit b de f	
BSF	f,b	Activar bit b de f	
BTFSC	f,b	Probar bit b de f, saltar si cero	
BTFSS	f,b	Probar bit b de f, saltar si uno	

Por último, aparece **k** que viene a ser una constante de 8 bit, es decir que k puede tomar valores entre 0 y 255, éstos inclusive, según la instrucción a utilizar

OPERACIONES ORIENTADAS A LITERALES Y DE CONTROL			
Nemotécnicos		**Operación**	**Estados afectados**
ADDLW	k	Sumar literal k a W	C,DC,Z
ANDLW	k	AND entre k y W	Z
CALL	k	Llamar subrutina	
CLRWDT		Limpiar WDT	-TO,-TD
GOTO	K	Salta a dirección k	
IORLW	K	OR entre k y W	Z
MOVLW	K	Cargar W con literal k	
RETFIE		Retornar de interrupción	
RETLW	K	Retornar y cargar W con k	
RETURN		Retornar de subrutina	
SLEEP		Ir al modo de bajo consumo	-TO,-TD
SUBLW	K	Restarle k a W	C,DC,Z
XORLW	K	OR exclusiva entre k y W	Z

Bueno, eso es todo, y creo que más que suficiente.

:: Microcontroladores PIC - Fuses del PIC 16F84/C84

LOS FUSES DEL PIC

Estas 4 "variables" del Pic16F84 (modelos superiores tienen más), sirven para configurar ciertos aspectos del microcontrolador. Cada FUSE activa o desactiva una opción de funcionamiento.

OSC (Oscilador):

Es el modo de oscilación que va a usar el Pic.

Cada vez que el Pic recibe un pulso eléctrico del oscilador da un paso para ejecutar una instrucción (4 impulsos para completar una), por lo que podemos decir que es una señal que le recuerda al Pic que tiene que seguir avanzando.

Según esto, el pic puede usar 4 tipos de oscilador:

XT: Es un acrónimo que viene de XTAL (o cristal en castellano). Este modo de funcionamiento implica que tendremos que disponer de un cristal de cuarzo externo al Pic y dos condensadores. El valor del cristal generalmente será de 4Mhz o 10Mhz, y los condensadores serán cerámicos de entre 27 y 33 nF. La exactitud de este dispositivo es muy muy alta, por lo que lo hace muy recomendable para casi todas las aplicaciones.

RC: Este es el sistema más sencillo y económico. Se basa en un montaje con una resistencia y un condensador. La velocidad a la que oscile el pic dependerá de los valores del condensador y de la resistencia. En la hoja de características del Pic están los valores.

HS: Para cuando necesitemos aplicaciones de "alta velocidad", entre 8 y 10Mhz. Se basa también en un cristal de cuarzo, como el XT

LP: "Low Power" la velocidad máxima a la que podemos poner el pic con este oscilador es de 200Khz. Al

igual que el XT y el HS, necesitaremos de un cristal de cuarzo y unos condensadores.

WDT (Watchdog Timer):

El famoso "perro" del pic. (perro guardián). Esta es una capacidad del pic de autoresetearse. Es muy útil, por ejemplo si un Pic, por un descuido de programación, se queda en un bucle infinito, esta "utilidad" lo sacará de él. Su funcionamiento es sumamente sencillo. Simplemente es un registro que debemos borrar cada cierto tiempo. Si transcurrido un cierto tiempo el registro no ha sido borrado el pic se resetea. La instrucción para borrar el registro es CLRWDT. Con poner un par de ellos a lo largo de nuestro código es suficiente para tener una garantía de que el pic no se quede "haciendo el bobo" (como dijo alguien por ahí...).

PWRT (Power Up Timer Reset):

Si activamos este FUSE, lo que conseguimos es que se genere un retardo en la inicialización del Pic. Esto se usa para que la tensión se estabilice, por lo que se recomienda su uso.

CP (Code Protect):

El "dichoso" Code Protection. Protección del código. Lo único que hace es impedir que algun curioso se apropie de tu creación no tiene efecto alguno en el correcto funcionamiento del PIC, ni que no se pueda sobreescribir su contenido. Lo único que hace es eso, impedir su lectura. Por cierto, dicen que puedes quitar la protección por medio de hardware, yo nunca lo hice, por que no lo creo necesario, ya que lo entretenido de esto es el desafío, no crees...???