:: PIC - Parte III - Interrupciones y Temporizaciones

Guía rápida

Interrupciones:

Interrupciones Internas y Temporizaciones:

Un interesante proyecto:

Apéndice:

Nota

En la sección **Apéndice** se encuentran los tres registros que venimos utilizando con más frecuencia hasta ahora, los puse solos porque a veces es bueno tenerlos a mano, o mejor... imprimirlos, también se encuentra un vínculo a la tabla de ciclos de instrucción para cuando necesites hacer cálculos en las temporizaciones.

Saludos para todos, y nos vemos en la próxima...!!!

R-Luis

:: Microcontroladores PIC - Parte Tres - Introducción

Breve Introducción

Esta vez traje a la web uno de los temas a los que más les temía cuando me estaba iniciando con los micros, y aunque parecía difícil, en realidad no lo es tanto.

Trabajar con interrupciones tiene sus ventajas y es hora de aprender a aprovecharlas, si las tenemos dentro del PIC, porque dejarlas de lado...???

Veamos de que se tratará todo esto

Comenzaremos con Interrupciones y analizaremos un poco el Registro INTCON y todos sus Bit's luego nos meteremos un poquito más allá y estudiaremos el registro OPTION, y así aprenderemos a hacer temporizaciones, pero lo haremos de dos formas, la que me gusta y también la otra... Finalmente hablaremos de un proyecto que cierta vez me encomendaron y al cual le dí solución con algunas de las cosas que verás en esta actualización, de acuerdo...???

Bien, manos a la obra...!!!

:: PIC - Parte III - Capítulo 1

INTERRUPCIONES:

Una de las características más importante de los microcontroladores y que mencionamos al inicio en nuestro primer tutorial, es que tienen la posibilidad de manejar interrupciones, y qué es esto...???

Muy sencillo, se trata de un acontecimiento que hace que el micro deje de lado lo que se encuentra realizando, atienda ese suceso y luego regrese y continúe con lo suyo.

Pues eso son las interrupciones, pero veamos, hay dos tipos de interrupciones posibles, una es mediante una acción externa (es decir por la activación de uno de sus pines), la otra es interna (por ejemplo cuando ocurre el desbordamiento de uno de sus registros)

En el PIC 16f84 hay 4 fuentes de interrupciones, veamos cuales son...

- Por el pin RB0/INT, que regresa al PIC del modo SLEEP (interrupción externa).
- Por los pines RB4 a RB7, configurados como entrada y en caso de que alguno de ellos cambie de estado (interrupción externa).
- Por desbordamiento del registro TMR0, cuando este registro pasa de 255 a 0 en decimal ó 0xFF a 0x00 en hexa (interrupción interna).
- Al completar la escritura de la EEPROM de datos (interrupción interna).

El tema es que, debe haber algo que nos indique la fuente de interrupción que se ha producido, y estas son las banderas de interrupciones, cada interrupción tiene su propia bandera y es un bit del registro INTCON, que cambia de estado de **0** a **1** cuando se produce la interrupción, salvo la última que se encuentra en el registro EECON1

Ahora veamos cuales son esas banderas...

- Para **RB0 / INT** la bandera es **INTF** (Bit1)
- Parar los pines **RB4** a **RB7**, es **RBIF** (Bit0)
- Para **TMR0**, es **T0IF** (Bit2)
- Para la **EEPROM**, es **EEIF** (Bit4) en el registro **EECON1**.

Si prestas atención, todas estas banderas terminan en **F** es decir **FLAG's**, aplausos para mí...!!! :o))

Bien, ahora veremos todo esto en el registro INTCON, aquí va...

REGISTRO INTCON							
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

El Bit GIE habilita todas las interrupciones, Los Bit's de fondo gris son las banderas, y los BIT's que se corresponden con cada flag son la habilitación de la fuente de interrupción para que esta cambie, recuerda que el flag de EEIE se encuentra en el registro EECON1.

Y ahora los detalles de cada Bit del registro INTCON.

BIT's	L ó E	Reset	Descripción
Bit 7: **GIE** Habilitación General.	L/E	0	1 = Todas las Interrupciones activadas 0 = Todas las Interrupciones desactivadas
Bit 6: **EEIE** Int. de Periféricos	L/E	0	1 = Activada 0 = Desactivada
Bit 5: **T0IE** Int. del TMR0	L/E	0	1 = Activada 0 = Desactivada
Bit 4: **INTE** Int. Externa	L/E	0	1 = Activada 0 = Desactivada
Bit 3: **RBIE** Int. por PORTB	L/E	0	1 = Activada 0 = Desactivada
Bit 2: **T0IF** Bandera del TMR0.	L/E	0	1 = TMR0 desbordado. 0 = No se ha desbordado
Bit 1: **INTF** Bandera - RB0/INT	L/E	0	1 = Ocurrió una interrupción externa 0 = No ha ocurrido interrupción externa
Bit 0: **RBIF** Bandera - RB4:RB7	L/E	x	1 = Al menos un pin cambio de estado 0 = Ningún pin ha cambiado de estado.

En la tabla, los valores de **L** ó **E** son para que sepas si el bit es de lectura o escritura, los valores de **Reset** son el estado de cada Bit después de producirse un reset o cuando se inicia el micro.

Por cierto y antes de que lo olvide, si bien cada flag cambia o se pone a 1 al producirse una interrupción, es tarea tuya borrarlo o ponerlo a cero nuevamente, ya que si no lo haces el micro estará siempre interrumpido o lo que es lo mismo, creerá que la interrupción se está produciendo continuamente.

Recapitulemos un poco... Ya sabemos como y cuando se produce una interrupción, conocemos las banderas que nos indican la fuente de interrupción producida, conocemos el registro donde se encuentran y los valores que toman cada uno de sus BIT's. Pues bien, ahora hay que atenderlas.

Lo primero que debes saber, es que cuando una interrupción se produce, sea cual fuere la fuente de interrupción, el micro deja todo y salta a la dirección 0x04, éste es el vector de interrupción, si recuerdas de nuestro primer tutorial, siempre saltábamos por encima de esta dirección para iniciar nuestro programa, en esta dirección es donde escribiremos la rutina que dé servicio a todas las interrupciones, o bien haremos un salto a donde se encuentre ese trozo de código, el cual se conoce como **ISR** (Rutina de Servicio de Interrupción)

El Tiempo de Procesamiento de la ISR debe ser lo más breve posible, para dar lugar a que se ejecuten las otras interrupciones, ya que puedes haber habilitado más de una de ellas.

Lo más crítico de una interrupción es tener que guardar todos los registros

importantes con sus respectivos valores, para luego restaurarlos, y así el micro pueda continuar con la tarea que estaba realizando cuando fue interrumpido.

Eso es lo que veremos ahora...

Rutina de Servicio de Interrupciones (ISR):

La tarea de guardar todos los registros importantes puede ser mas o menos complicada si el programa que estás realizando es demasiado extenso o principalmente cuando en la ISR modificas alguno de los valores de esos registros, en algunos casos no es necesario ya que por lo general se trata de no modificarlos utilizando registros alternativos, pero veamos como hacerlo.

Primero debes guardar el contenido del registro W, el problema de mover W a otro registro (haciendo uso de **MOVF**) es que esta instrucción corrompe la bandera Z, modificando el registro de Estado. Según la hoja de datos otorgada por Microchip, en uno de sus apartados recomienda una secuencia de código que permite guardar y restaurar los registros sin modificarlos.

Suponte que W=0x0A y ESTADO=0xAF, La forma de guardarlos y recuperar estos registros sería la siguiente:

```
; ========= Inicio - Rutina de Servicio de Interrupción ==========
; ========== Guardando W y el Registro de Estado ==========

MOVWF   Reg_W       ; Guardamos W en Reg_W (Reg_W=0x0A)
SWAPF   ESTADO,W    ; invertimos los nibbles del registro ESTADO
                    ; y lo pasamos a W (ESTADO=0xAF), (W=0xFA)
MOVWF   Reg_S       ; Guardamos el contenido de ESTADO (Reg_S=0xFA)
.       .
.       .           ; Atendemos la interrupción
.       .
; ========= Fin - Rutina de Servicio de Interrupción ===========
; ========= Restaurando W y el Registro de Estado ===========

SWAPF   Reg_S,W     ; invertimos los nibbles de Reg_S
                    ; y lo pasamos a W (Reg_S=0xFA), (W=0xAF)
MOVWF   ESTADO      ; Restauramos ESTADO (ESTADO=0xAF)
SWAPF   Reg_W,f     ; invertimos los nibbles de Reg_W (Reg_W=0xA0)
SWAPF   Reg_W,W     ; y lo pasamos a w invirtiéndoles nuevamente
RETFIE              ; Ahora W=0x0A
```

Reg_W y **Reg_S** son registros alternativos para guardar los valores del registro W y del registro de estado respectivamente.

SWAPF ESTADO,W

Es como decirle "invierte los nibbles del registro ESTADO y guárdalos en W".

La instrucción SWAPF invierte los nibbles del registro, por ejemplo si el registro tenia 0xAF luego de SWAPF quedará 0xFA, si especificas **W** el valor invertido se guarda en W si indicas **f** se guardará en el mismo registro, así...

SWAPF Reg_W,f

Es como decirle "invierte los nibbles de Reg_W y guárdalos en Reg_W".

Creo que se entiende...

Bien, lo bueno de utilizar la instrucción SWAPF en lugar de MOVF es que no afecta la bandera Z del registro de ESTADO, y aunque los nibbles se invierten, al restaurarlos los vuelves a invertir nuevamente para dejarlos como estaban.

Como dije anteriormente, no siempre será necesario, todo depende del programa que estés haciendo.

Algo que no mencioné, es la instrucción...

RETFIE

Veamos y tratemos de resumir un poco lo que vimos hasta ahora...

Si se ha producido una interrupción, obviamente una de las banderas del registro INTCON cambiará de estado y el micro irá a la dirección 0x04 como si se hubiera producido un CALL (una llamada) a esa dirección para ejecutar la ISR, por lo tanto la pila o STACK se carga una posición más, y el mecanismo de las interrupciones se deshabilita (es decir **GIE=0**) para dar lugar a la ISR.

Ahora bien, debes recuperar los registros importantes (lo que acabamos de ver), averiguar la fuente de interrupción, atender la interrupción, luego restaurar aquellos registros importantes, reponer el estado de las banderas del registro INTCON (aquellas que fueron modificadas por la interrupción) y regresar, pero no con un RETURN, ya que no estas regresando de una rutina cualquiera, sino de una interrupción, la cual está deshabilitada, y para habilitarla nuevamente es recomendable utilizar la instrucción **RETFIE**, y yo cumplí...!!!

Y ahora todas las interrupciones están habilitadas nuevamente, es decir **GIE=1**

Nada impide que utilices **RETURN** pero deberás usar una instrucción más para habilitar GIE si deseas continuar usando la interrupción, esto queda a criterio del programador, Microchip recomienda el uso de **RETFIE** y yo como chico bueno la utilizo. ;oP

Como era de esperarse no todo termina aquí, ya que algunos de los parámetros para las interrupciones se encuentran en otro registro, el registro OPTION, veamos de que se trata...

:: PIC - Parte III - Capítulo 3

El Registro OPTION

Este es otro de los registros que tienen mucho que ver con las interrupciones, algunos de sus Bit's deben ser modificados, según la aplicación que estés realizando.

Por ejemplo; dijimos que por el pin RB0/INT, regresas al PIC del modo SLEEP, lo cual podría hacerse mediante un pulsador, suponte que el pulsador está al polo positivo (VCC) y con una resistencia a GND, de tal modo que la interrupción se produzca al enviar un 1 (presionando el pulsador), pero también podría hacerse enviando un 0 (liberando al pulsador). por lo tanto la interrupción debe ser **sensible a** un **1** o bien a un **0**, como sabrá esto el micro...???, pues muy fácil, hay que especificarlo, y esto se hace en el Bit6 (**INTDEG**) del registro OPTION, con un **1** será sensible al flanco ascendente, y en el momento que envíes un 1 por el pulsador se producirá la interrupción, si pones ese Bit a **0** será sensible al flanco descendente y la interrupción se producirá cuando liberes el pulsador, es decir enviando un 0.

Este es el registro OPTION...

REGISTRO OPTION							
RBPU	**INTDEG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**

Y aquí verás como configurar algunos de sus BIT's...

BIT's	L ó E	Reset	Descripción
Bit 7: **RBPU** Pull-up p' PORTB	L/E	1	1 = Cargas Pull-Up Desconectadas 0 = Cargas Pull-Up Conectadas
Bit 6: **INTEDG** Flanco/Interrup.	L/E	1	1 = RB0/INT será sensible a flanco ascendente 0 = RB0/INT será sensible a flanco descendente
Bit 5: **T0CS** Fte./Reloj p' TMR0	L/E	1	1 = Pulsos por el pin RA4/T0CKI (contador) 0 = Pulsos igual Fosc/4 (temporizador)
Bit 4: **T0SE** Flanco/T0CKI	L/E	1	1 = Incremento TMR0 en flanco descendente 0 = Incremento en flanco ascendente
Bit 3: **PSA** Divisor/Frecuencia	L/E	1	1 = Divisor asignado al WDT 0 = Divisor asignado al TMR0

Como puedes ver, en la tabla no figuran los primeros tres Bit's, y es que la combinación de los BIT's; PS2, PS1 y PS0 (2, 1 y 0 respectivamente) determinan el valor del divisor de frecuencia o prescaler (mmmmmmm, no te preocupes que cuando terminemos con este tutorial o mejor dicho cuando hablemos de temporizaciones sabrás de que se trata)...

Basta de teoría, es hora de pasar a la práctica, haremos nuestro primer programa con

interrupciones, que emocionante...!!! ya me estaban picando las manos para codificar...!!!, quería hacer algo complejo e interesante, pero temo que te pierdas y lo que es peor, temo enredarme al tratar de explicarlo, así que haremos algo sencillito, ok.??? luego lo iremos complicando y pondremos a llorar a muchos, jejeje

Bueno, como son la 3 de la madrugada y mis bellos ojos comienzan a cerrarse, lo haremos mañana, ahí nos vemos...!!!

:: PIC - Parte III - Capítulo 4

Codificando interrupciones

Ya estoy de regreso nuevamente, y a ver quién me sigue... que esta vez haré un programa que a muchos les puede resultar bobo, así que... a no criticar, que ya lo advertí...

Comenzamos...???

Bien, el programa consiste en preparar todo para el encendido de un LED que conectaremos en RB1, pero como dije, sólo prepararemos todo, porque luego haremos dormir al micro hasta que interrumpamos su sueño para atender un pulsador conectado a RB0/INT, momento en el cual deberá encender el LED, y regresar a dormir nuevamente, y cuando vuelvas a presionenar el pulsador haremos que lo apague y otra vez lo despacharemos a dormir.

Esto ya lo hicimos anteriormente, sólo que ahora lo haremos con interrupciones, ok...???

Allá vamossss...!!!

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

        LIST    P=16F84
        #include <P16F84.INC>

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

        ORG     0x00
        GOTO    inicio
        ORG     0x04
        GOTO    ISR
        ORG     0X05
inicio  BSF     STATUS,RP0      ; configurando puertos
        MOVLW   0x01            ; carga w con 0000 0001
        MOVWF   TRISB           ; RB0/INT es entrada
        BCF     OPTION_REG,6    ; seleccionamos flanco descendente
        BCF     STATUS,RP0

;-------Habilitación de interrupciones-------

        BSF     INTCON,GIE      ; habilitamos todas las interrupciones
        BSF     INTCON,INTE     ; que sean interrupciones externas
        CLRF    PORTB           ; limpio el puerto B
```

```
sueño  SLEEP
       GOTO    sueño            ; Dulces sueños...!!!

;-------------Rutina de servicio de interrupciones-------------

ISR    BTFSC   PORTB,0          ; verificamos que suelten el pulsador
       GOTO    ISR
       BTFSC   PORTB,1          ; y ahora sí, si el led está a 1
       GOTO    off_led          ; ire a off_led para apagarlo
       BSF     PORTB,1          ; sino, enciendo el LED
       BCF     INTCON,INTF      ; borro bandera de interrupción
       RETFIE
off_led BCF    PORTB,1          ; apago el LED
       BCF     INTCON,INTF      ; borro bandera de interrupción
       RETFIE

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

Desde nuestros primeros tutoriales hemos alcanzado a conocer varias de las instrucciones que se encuentran en este trozo de código, razón por la cual no las describiré, así es que vamos por aquello que está en rojo...

```
ORG     0x04
GOTO    ISR
```

La primera línea es el vector de interrupción, y cuando ésta se produzca, el código de programa apuntará a esta dirección y continuará con la siguiente instrucción, es decir GOTO ISR, la cual es un salto a ISR (Rutina de Servicio de Interrupciones) para atender dicha interrupción.

Configuramos el puerto B, como habrás notado, hemos configurado RB0/INT como entrada y el resto de los bits como salida, luego...

```
BCF     OPTION_REG,6     seleccionamos flanco descendente
```

En la página anterior dijimos que podíamos seleccionar el flanco con el cual se producirá la interrupción, pues eso es lo que estamos haciendo con esta instrucción, entonces vamos al registro OPTION y ponemos el BIT6 a **"0"** de este modo la interrupción se producirá cuando suelten el pulsador.

Ahora pasamos a lo más interesante, la habilitación de las interrupciones...

```
BSF     INTCON,GIE      ; habilitamos todas las interrupciones
BSF     INTCON,INTE     ; que sean interrupciones externas
```

Observa que la habilitación de interrupciones se hace en el banco0 ya que el Registro INTCON se encuentra en este banco. Bien, En la primera línea hacemos una habilitación general de todas las interrupciones, hacemos GIE=1, en la segunda línea, habilitamos interrupciones externas, hacemos INTE=1, recuerda que la bandera para la interrupción por el pin RB0/INT es INTF, no lo olvides, pues esta cambiará cuando la interrupción se produzca y luego de atenderla deberemos volverla a cero.

Lo que viene ahora es simplemente limpiar el puerto B, y luego...

```
sueño   SLEEP
```

```
GOTO    sueño
```

SLEEP es la instrucción que pone al micro en estado de bajo consumo, es como que todo se detiene y éste pasa a modo de reposo, (consulta el set de instrucciones para mayor detalle...) debí haber puesto simplemente SLEEP pero veamos, si se ejecutaría la instrucción SLEEP el micro entraría en reposo hasta que se produce la interrupción, lo cual dijimos anteriormente que es como una llamada (un call), cuando regrese se encontrará con **GOTO sueño** y lo volveremos a dormir.

Te imaginas...??? si no pusiéramos el **GOTO sueño,** cuando regrese de la interrupción pasaría a la **ISR** (Rutina de servicio de interrupción), y lo peor de todo, es que lo haría sin que se produzca la interrupción, gran dolor de cabeza...!!!

Pero bueno, ahora nos quedamos a dormir con el micro hasta que un chico traviezo active el pulsador de RB0/INT ...Felices sueñosssss...!!!!

chico malo...!!!, que siempre me pones a prueba, Ya veo que no pudiste esperar y presionaste el pulsador...

De acuerdo... sabes a donde me llevaste...??? justo a...

```
ORG     0x04
GOTO    ISR
```

Entonces vamos hacia allá, ahora te daré para que tengas, guardes y repartas...

Lo que viene, es la rutina de servicio de interrupción **ISR** y comenzamos con un...

```
ISR   BTFSC   PORTB,0   ; verificamos que suelten el pulsador
      GOTO    ISR
```

ISR no es una instrucción, sino la etiqueta que atiende la interrupción (pude haber puesto rut_serv u otra cosa, en fin...). Con **BTFSC PORTB,0**, prevenimos los rebotes, no se si era necesario ya que seleccionamos flanco descendente para este pin, pero por si las moscas lo puse, en realidad suele pasar que cuando se libera el pulsador se genera una pequeña chispa, la cual ya conocemos como rebote eléctrico, sólo lo puse por prevención.

Ahora si atenderemos la interrupción, comenzando por...

```
BTFSC   PORTB,1
```

BTFSC PORTB,1 es probar si el segundo bit (Bit1 de PORTB) está en **0**, es decir si el LED está apagado y saltar un línea si es así.

*En lecciones anteriores utilizamos un registro llamado **cont** con el cual sabíamos si el LED estaba prendido o apagado, y aquí tienes una forma de optimizar ese código, espero que no te pierdas con esto...!!!*

Bien... como recién iniciamos, el LED está apagado, por lo tanto saltamos una línea y pasamos a...

```
BSF   PORTB,1
```

es decir hacemos **RB1 = 1** (prendemos el LED). Perfecto, la interrupción ya fue atendida, pero ahora debemos habilitarla de nuevo así permitimos que se vuelva a ejecutar, y como tenemos un único pulsador el cual me cambió la bandera INTF, deberemos borrarla nuevamente, así es que...

```
BCF   INTCON,INTF   ; borro bandera de interrupción
```

Obviamente al producirse la interrupción se hizo **GIE = 0** para darnos lugar a atenderla, entonces...

```
RETFIE
```

y ahora **GIE=1**, las interrupciones están nuevamente habilitadas la bandera de RB0/INT está lista para una nueva interrupción y retornamos a ...

```
sueño   SLEEP
        GOTO    sueño
```

y esperaré a que pulses RB0, pues si ya lo hiciste habrás ido por segunda vez a ...

```
ORG     0x04
GOTO    ISR
```

prevenimos los rebotes y luego vamos a...

```
BTFSC   PORTB,1
```

es decir, prueba y salta si el Bit1 de PORTB es cero, y como esta vez el LED está prendido... simplemente harás un...

```
GOTO    off_led   ; ire a off_led para apagarlo
```

un salto a la etiqueta off_led...

```
off_led   BCF   PORTB,1   ; sino, apago el LED
```

no se si requiere explicación pero bueno, pones a cero el Bit1 de PORTB. Finalmente...

```
BCF   INTCON,INTF   ; borro bandera de interrupción
RETFIE
```

Ahora estamos listos para comenzar de nuevo...

te gustó...???, bieeeenn, aplausos para quien logró comprender...

y ahora la pregunta del millón...

que pasó con el mapa de memoria...???, donde se definieron las posiciones de memoria para los registros TRISB, PORTB, el registro OPTION..??? y el INTCON...???, donde demonios se definieron estas posiciones de memoria que supuestamente deben estar en el encabezado...???

No me digas que no te percataste de eso...!!! :-O

La respuesta la tendremos en el próximo tutorial, y por la misma web ...ahí nos vemosssss...!!!

:o))

Simulando la interrupción con MPLAB

Antes de comenzar con la simulación vamos a aclarar un pequeño detalle, o mejor dicho... vamos a dar respuesta a la pregunta del millón... no pensarías que te dejaría así...!!!, noooooooo...!!!

Bueno, antes de que copies el código en MPLAB e intentes ensamblarlo, lo cual seguramente te dará quinientos mil errores, debes saber que Cuando instalaste MPLAB allá en:

C:\Archivos de programa\MPLAB

Se instaló también un pequeño archivito en el mismo directorio, llamado **P16F84.INC**, Bien, búscalo y una vez lo tengas a mano... lo abres, que le haremos un par de modificaciones...

Te encontrarás con algo como esto...

```
;==========================================
;
;       Register Definitions
;
;==========================================
W                               EQU     H'0000'
F                               EQU     H'0001'

;----- Register Files------------------------------------

INDF                            EQU     H'0000'
TMR0                            EQU     H'0001'
PCL                             EQU     H'0002'
STATUS                          EQU     H'0003'
FSR                             EQU     H'0004'
PORTA                           EQU     H'0005'
PORTB                           EQU     H'0006'
EEDATA                          EQU     H'0008'
EEADR                           EQU     H'0009'
PCLATH                          EQU     H'000A'
INTCON                          EQU     H'000B'
OPTION_REG                      EQU     H'0081'
TRISA                           EQU     H'0085'
TRISB                           EQU     H'0086'
```

```
EECON1                  EQU     H'0088'
EECON2                  EQU     H'0089'
```

Bueno, es sólo una parte del archivo P16F84.INC, éste archivo contiene los nombres de los registros con sus respectivas posiciones de memoria, aquello que nosotros veníamos indicando en cada código que íbamos escribiendo, y una que otra cosita más como los nombres de los Bit's de cada uno de los registros, y si recuerdas siempre debíamos indicar la posición 0x05 para TRISA y 0x06 para TRISB, por tanto para OPTION_REG (registro OPTION) sería 0x01, te preguntarás... porque aquí las cosas se ven totalmente distintas...???

Lo que ocurre, es que cuando pasas al banco 1... TRISA está quinto en ese banco, es decir está en la posición 0x05, lo mismo ocurre con TRISB en 0x06, y por ende OPTION_REG está en 0x01, observa ahora los bancos de la RAM de nuestro primer tutorial... y compara con lo que acabamos de ver...

Convencido...???.

El tema es que para evitar tener que definirlos, tomaremos aquello que marqué en rojo y lo cambiaremos por...

```
OPTION_REG          EQU     H'0001'
TRISA               EQU     H'0005'
TRISB               EQU     H'0006'
```

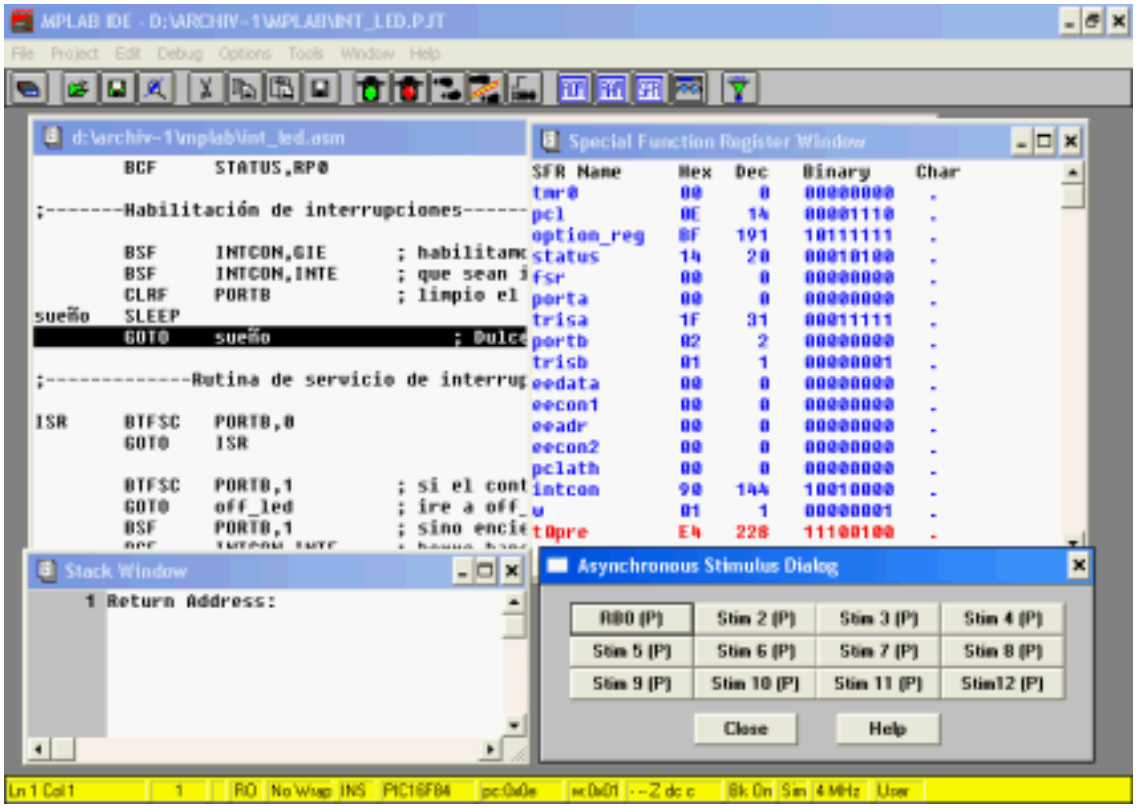
De ahora en más siempre que hagamos un programa será obligación colocar en el encabezado de nuestro código la siguiente línea...

#include <P16F84.INC>

De acuerdo... Ahora guardamos los cambios, cerramos el archivo y comenzamos un nuevo proyecto en MPLAB al cual lo llamaremos INT_LED.pjt, y en él creamos int_led.asm, copias el código, lo pegas y le das a...

Project --> Build All

Y como todo está perfecto...!!! comenzaremos la simulación, te debería quedar algo así...



No te asustes por todas las ventanas abiertas, son sólo 4, y todas accesibles desde el menú Window, la primera es el código, la que está al lado es **Special Function Register** en la que veremos como cambian los registros, la de abajo es la ventana que nos muestra la pila o STACK y la última es la de **Asynchronous Stimulus** esta última se encuentra en el menú **Debug --> Simulator Stimulus**, cuando la abras configura **Stim 1 (P)** como **RB0 (P)**, eso hará que cuando lo presionemos envíe un pulso de nivel alto por el pin RB0, al configurarlo como **(P)** se convierte en un pulsador, ahora sí, ya estamos listos para comenzar...

Reseteamos el Micro o presionamos F6, y te habrás ubicado en **GOTO inicio**, ahora ve a...

Debug --> Run --> Animate

y quedará todo como está en la imagen anterior

Aquí haremos un par de observaciones, fíjate que estas en **GOTO sueño**, ésta es la siguiente instrucción que se debería ejecutar, pero no lo hace ya que el micro está dormido gracias a la instrucción SLEEP, observa también que en la ventana **Special Function Register** todo se pintó de azul por tanto el micro se detuvo y apagó casi todo. El STACK está vacío ya que no se produjo ninguna llamada, PORTB está en 00000000, es decir que el LED está apagado (RB1=0) y no hay ninguna interrupción todavía (RB0=0), finalmente héchale una mirada al registro INTCON que esta en 10010000 es decir GIE=1 e INTE=1 las interrupciones están habilitadas

Ahora viene lo bueno...

Envía un pulso por **RB0 (P)**, y verás que la interrupción hace saltar al micro en la dirección 0x04, (no esperes ver en PORTB que RB0 se ponga a 1 ya que al configurar RB0 (P) sólo envía un pulso momentáneo el cual es difícil notar), el STACK se incrementa en una posición, y en el registro INTCON se deshabilita GIE, la bandera INTF se pone a 1, luego el micro apunta a ISR, atiende la interrupción encendiendo el LED (RB1=1), luego Borra la bandera INTF y con RETFIE vacía la pila habilitando GIE nuevamente para regresar a GOTO sueño

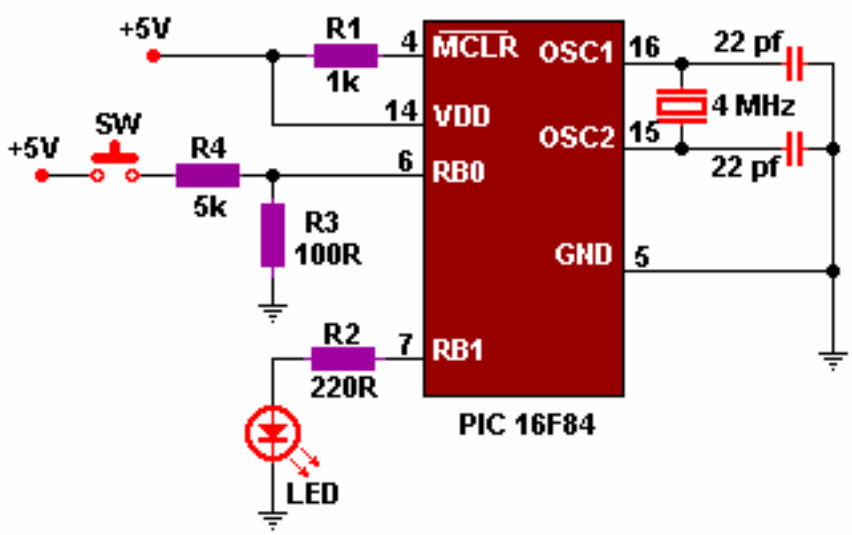
donde ejecutará SLEEP, para dormirse y esperar a que presiones nuevamente el pulsador...

Bravooooo...!!!

Tienes mucho para entretenerte, una jugada que te pondrá de pelos...

Observa lo que harás; configura **RB0 (P)** como **RB0 (T)**, resetea el micro y comienza nuevamente, la mayor sorpresa es que cuando lo presiones, RB0 se pondrá a 1 y no habrá interrupción, esto es así por que seleccionamos flanco de bajada para la interrupción en RB0/INT, aquello que hicimos en el Bit6 del registro OPTION, recuerdas eso...???, ok... entonces debería haber interrupción cuando presiones nuevamente RB0 y ya no diré más...

Bueno... sólo dejarte el circuito para que lo pruebes cuando grabes el programa en el PIC...



Suerte...!!!, y a no bajar los brazos que lo probé y funciona perfectamente.

Por si te quedan dudas de como funciona el programa, realiza la simulación en modo STEP (con el botón de los zapatitos), eso debería responder a todas tus dudas...

:: PIC - Parte III - Capítulo 6

Interrupciones Internas y Temporizaciones

Lo que hicimos anteriormente fue trabajar con interrupciones externas aunque no tocamos aquello de las interrupciones por los pines RB4 a RB7, el procedimiento es muy similar sólo debes tener en cuenta la bandera que informa el estado de las interrupciones y estudiarlas para saber por cual de esos pines se produjo.

Lo que veremos ahora será como trabajar con una interrupción interna, algo no muy distinto a lo anterior pero que lo vale, ya que haremos uso de algo que fue pedido en el foro por César Bonavides, a quien envío mis cordiales saludos allá en méxico.

Antes de entrar en materia, debo confesar que lo fuerte de esta sección serán las **Temporizaciones**, aunque obviamente haremos uso de interrupciones, de acuerdo...???

Desde mis inicios con los microcontroladores, temporizar fue algo que me trajo grandes dolores de cabeza, y aunque me las arreglaba como podía, me quedaba siempre con la duda, *que hago cuando quiero temporizar sólo un segundo...???*, supongo que a muchos les suele pasar, hasta que finalmente no lo pude evitar y comencé a escarbar en cuanto tutorial caía en mis manos, y de ello surgió lo que describiré aquí, y trataré de explicarlo lo mejor posible, espero no meter la pata...!!!

Como siempre... un poco de teoría no viene mal no crees...???

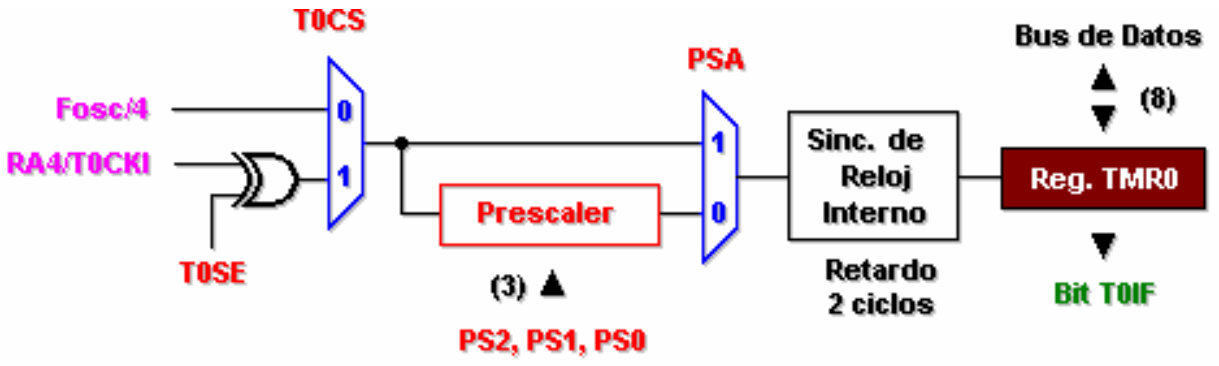
Bien, a prepararse que comenzamos...

Temporizaciones

Algunas de las cosas que deberemos tener en cuenta son...

Por un lado, el oscilador externo, y yo lo haré con un XT de 4 Mhz es decir 4 millones de ciclos por segundo, y por otro lado un registro llamado TMR0, este registro es un Temporizador/Contador de 8 bits, que como es lógico cuenta en binario y en forma ascendente es decir... de 0x00 a 0xFF, lo interesante de este registro es que cuando ocurre un desbordamiento, es decir pasa de 0xFF a 0x00 hecha sus alaridos en el registro INTCON modificando la bandera T0IF, es decir, produce una interrupción.

Veamos ahora como está conformada la estructura del micro para trabajar con este registro, y quienes están involucrados en ello.



Creo que ahora se van a entender un poco mejor las cosas...

Aquello que está en rojo son Bit's configurables en el Registro OPTION, lo que está en verde **Bit T0IF** es el Bit2 del registro INTCON (la bandera que se altera cuando el TMR0 se desborda), y lo que está en rosado... bueno, hablemos de ello...

El registro TMR0 puede trabajar de dos modos distintos, ya sea como temporizador o bien como contador el cual seleccionarás en **T0CS** (Bit5 del registro OPTION)

En modo Temporizador: El TMR0 se incrementa con cada **Ciclo de Instrucción** (**Fosc/4**) y cuando el registro se desborde provocará una interrupción.

En modo Contador: El TMR0 se incrementará con cada pulso que ingrese por el pin **RA4/T0CKI**, y por supuesto... cuando se desborde producirá la interrupción. **T0SE** es el Bit4 del Registro OPTION, en él seleccionas el flanco con el cual se incrementará el TMR0 cuando haya un pulso por RA4.

Algo que quería mencionar es que el micro dispone de dos temporizadores, el TMR0 y WDT (Watchdog). El primero es el que estamos tratando en esta sección, el segundo es el *perro guardián*, lo que hace es vigilar cada cierto tiempo que el micro no se quede colgado, por ejemplo cuando se queda detenido en un bucle infinito o en una larga espera de un acontecimiento que no se produce, entonces actúa reseteando al micro, en otro momento hablaremos más de él...

El **Prescaler** es un predivisor de frecuencia que se utiliza comúnmente para programar tiempos largos y puedes aplicarlo al TMR0 o al WDT, esto lo configuras en **PSA** Bit3 del registro OPTION.

Hablamos tanto del Registro OPTION que decidí traértelo nuevamente pero marcando con color aquello que acabamos de mencionar, y aquí está...

REGISTRO OPTION							
RBPU	**INTDEG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**

Y para refrescar...

Bit 5: **T0CS**: Selecciona la fuente de Reloj para TMR0
1 = Pulsos por el pin RA4/T0CKI (contador)
0 = Ciclo de instrucción interno (temporizador)

Bit 4: **T0SE**: Flanco de incremento para RA4/T0CKI
1 = Incrementa TMR0 en flanco descendente
0 = Incremento en flanco ascendente

Bit 3: **PSA**: Bit de asignación del Prescaler
1 = Divisor asignado al WDT
0 = Divisor asignado al TMR0

Bit 2-0: **PS2, PS1, PS0**: Selección del prescaler (divisor de frecuencia)

El prescaler debe tener algún valor, y si bien no puedes cargarle un valor cualquiera, tienes la posibilidad de seleccionarlo según la combinación de **PS2, PS1** y **PS0**.

En la siguiente tabla puedes ver estos posibles valores...

PS2	PS1	PS0	División del TMR0	División del WDT
0	0	0	1/2	1/1
0	0	1	1/4	1/2
0	1	0	1/8	1/4
0	1	1	1/16	1/8
1	0	0	1/32	1/16
1	0	1	1/64	1/32
1	1	0	1/128	1/64
1	1	1	1/256	1/128

Algo que me estaba quedando pendiente es la **Sincronización de Reloj interno**, por lo visto, según la hoja de datos, el micro genera un pequeño retraso de 2 ciclos de instrucción mientras sincroniza el prescaler con el reloj interno.

Bien, ahora veremos para que nos sirve toda esta información...

:: PIC - Parte III - Capítulo 7

Como hacer una temporización con el registro TMR0

El tiempo empleado en una temporización se puede calcular a partir de **un ciclo de instrucción** (es decir 1 instrucción por cada microsegundo, si estas trabajando con un XT de 4 Mhz), también necesitas el valor del **Divisor de Frecuencia** (el que seleccionabas con los Bit's PS2, PS1 y PS0), y finalmente con **el complemento del valor cargado en TMR0** (es decir 255-TMR0), la ecuación que te permite realizar el cálculo es la que sigue...

Temporización = Ciclo de instrucción * (255-TMR0) * Divisor de Frecuencia

Vemos un ejemplo...???

Suponte que deseas una temporización de 10 ms (10 milisegundos), que estás trabajando con un XT de 4 Mhz, y que a demás seleccionaste como Divisor de frecuencia 256 (es decir PS2,PS1,PS0 = 1,1,1).

Pregunta... (*como en el secundario*...)

Cuál es el valor que se debe cargar en TMR0...???

Lo arreglaremos con un pasaje de términos...

255-TMR0 = Temporización(en microsegundos)/(1 ciclo/us * Div. de Frec.)

y reemplazando tendrás...

255-TMR0 = 10000 us/(1 ciclo/us * 256)
255-TMR0 = 10000 /(256 ciclos)
255-TMR0 = 39,0625 ciclos
255-TMR0 ~ 39 ciclos

Eso significa que en TMR0 deberás cargar 255-39=**216** (0xD8 en hexa) y a partir de allí el TMR0 contará los 39 ciclos que faltan para desbordarse y producir la interrupción, y el tiempo que tardará en hacerlo es aproximadamente 10000 us, o sea 10 ms.

Antes de seguir, despejemos un par de dudas:

1 seg. = 1000 ms = 1000000 us y ...
1 ciclos/us es el tiempo empleado en ejecutarse una instrucción

ok..., sería bueno que me confirmes si la mayor temporización que se puede obtener haciendo uso de este registro es 0,06528 segundos, será...??? ahí queda...!!!

Lo que haremos ahora, será codificar el ejemplo visto anteriormente, pero una vez producida la interrupción encendemos un LED, luego volvemos, temporizamos 10 ms y en la próxima interrupción, lo apagamos, es decir, el LED parpadeará cada 10 ms, como es obvio, no lo vamos a notar, así que sólo lo simularemos en MPLAB, (*en*

realidad si se nota, luego te cuento como).

Bien, el código es el siguiente...

```
;--------------Encabezado------------
        LIST    P=16F84
        #include <P16F84.INC>
;-------Configuración de puertos-------

        ORG     0x00
        GOTO    inicio
        ORG     0x04            ; Atiendo la interrupción
        BTFSS   PORTB,0         ; si el LED está apagado
        GOTO    LED             ; voy a LED y lo enciendo
        BCF     PORTB,0         ; sino apago el LED
        BCF     INTCON,2        ; limpio la bandera T0IF
        RETFIE                  ; regreso habilitando la interrupción

LED     BSF     PORTB,0         ; enciendo el LED
        BCF     INTCON,2        ; borro la bandera T0IF
        RETFIE                  ; regreso habilitando la interrupción

inicio  BSF     STATUS,5        ; configurando puertos
        CLRF    TRISB           ; puerto B es salida
        MOVLW   0x07            ; cargo w con 00000111
        MOVWF   OPTION_REG      ; el Divisor = 256
        BCF     STATUS,5
        MOVLW   0xA0            ; cargo w con 10100000
        MOVWF   INTCON          ; habilitamos GIE y T0IE
        CLRF    PORTB           ; limpiamos PORTB

tiempo  MOVLW   0xD8            ; cargo w con 216
        MOVWF   TMR0            ; lo paso a TMR0

NADA    BTFSC   TMR0,7          ; me quedo haciendo nada
        GOTO    NADA            ; hasta que TMR0 desborde, y entonces
        GOTO    tiempo          ; volveré a cargar TMR0

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

Aquí vamos...

ORG 0X04 ; Atiendo la interrupción

Aquí vendremos cuando se desborde el TMR0, es decir cuando se produzca la interrupción y no haremos una ISR aparte como lo hicimos anteriormente, atenderemos la interrupción directamente aquí.

El código que sigue es como dice el comentario, se trata de verificar si RB0 está a 1 (es decir si el LED esta encendido), y como de comienzo no lo está, irá a GOTO LED, ahí lo enciende, luego...

```
BCF   INTCON,2   ; limpio la bandera T0IF
```

Esto es lo que debemos tener en cuenta para salir de una interrupción, borrar la bandera que indica al micro que hubo una interrupción, o nos quedaremos siempre en la rutina de servicio. Finalmente con...

```
RETFIE
```

habilitamos nuevamente la interrupción.

Pasemos ahora a la etiqueta **inicio**, lo primero que haremos será cambiar de banco y luego configurar el puerto B como salida, y aquí viene lo nuevo...

```
MOVLW   0x07          ; cargo w con 00000111
MOVWF   OPTION_REG    ; el Divisor = 256
```

Veamos que Bit's estamos configurando en **OPTION_REG**

Los Bit's 7 y 6 no los utilizamos por ahora, T0CS=0 (**TMR0 es temporizador**), T0SE=0 (no se usa), PSA=0 (**Prescaler asignado a TMR0**), PS2,PS1,PS0=1,1,1 (**Prescaler es 256**), en conclusión 00000111=0x07 y es lo que cargamos en el registro OPTION.

Ahora cambiamos de banco y habilitamos las interrupciones **GIE**, y en especial **T0IE**, que es la interrupción por desbordamiento del registro TMR0, luego...

```
CLRF   PORTB    ; limpiamos PORTB
```

Lo que viene ahora es preparar la temporización, y de los cálculos que hicimos debíamos cargar 216 en TMR0 y a partir de ahí esperar a que este registro se desborde y produzca la interrupción, entonces hacemos eso justamente...

```
tiempo  MOVLW   0XD8     ; cargo w con 216
        MOVWF   TMR0     ; lo paso a TMR0
```

tiempo es la etiqueta en donde cargaré el registro TMR0 cada vez que quiera hacer una temporización, y 0xD8 es 216 en hexadecimal

```
NADA  BTFSC  TMR0,7  ; me quedo haciendo nada
      GOTO   NADA    ; hasta que TMR0 desborde, y entonces
      GOTO   tiempo  ; volveré a cargar TMR0
```

La verdad es que ya no tengo nada que hacer, sino esperar a que desborde el TMR0, así es que hice un bucle al cuete, con **BTFSC TMR0,7** estas probando si el Bit7 de TMR0 está a 0, y como ya sabemos que estará a 1, pues ahí te quedas dando vueltas en ese bucle mientras el tiempo pasa, hasta que de repente se produce una interrupción, luego vas, la atiendes y cuando regresas caes en...

```
GOTO   tiempo  ; volveré a cargar TMR0
```

para que comiences a temporizar nuevamente, es decir recargar TMR0 con 216 para luego quedarte en el bucle a esperar la interrupción.

Ahora pasemos a lo mejor de todo esto, **La simulación en MPLAB**, allá vamos...

:: PIC - Parte III - Capítulo 8

Simulando interrupciones y temporizaciones con TMR0 en MPLAB

Antes de simular hay que crear un nuevo proyecto, así que eso es lo que haremos, abres MPLAB y si por las dudas te pregunta que si deseas abrir el proyecto anterior seleccionas **No** luego te vas a...

Project --> New project...

y creamos **tmr.pjt** le das a **Ok** y en la ventana **Edit project** seleccionas **tmr[.hex]** y luego a **Node Properties** allí tildas **INHX8M** y **HEX**, confirmamos y nuevamente en **Edit project**, te vas a **Add Node** y escribes **tmr.asm** para ligarlo al proyecto, finalmente le das a **aceptar** y luego **Ok**, bien, ya creamos el proyecto, pero nos falta tmr.asm, así que ve a **File --> New** y ya tienes Untitled1, que mas...???, ah sí... guárdalo como tmr.asm, listo señores, ahora sí...!!!

Vamos al código, lo copias, lo pegas y luego...

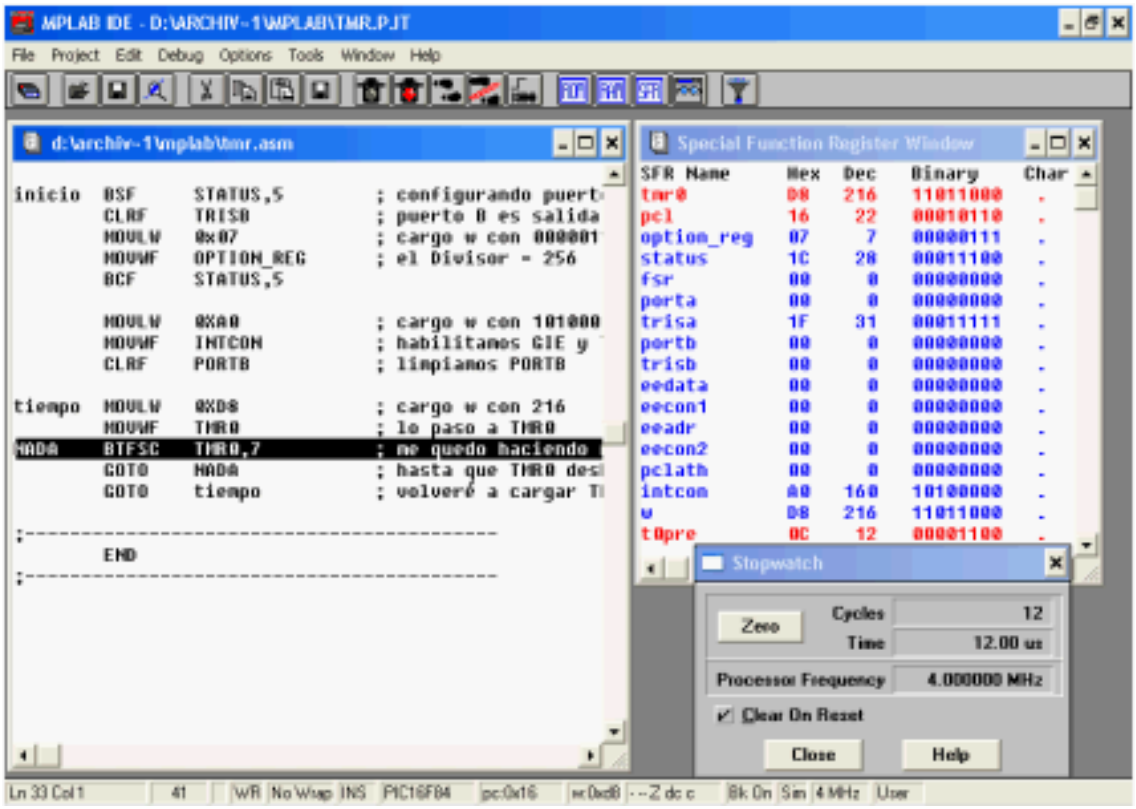
Project --> Build All

Perfecto...!!!

Abriremos dos ventanas que son para chequear el funcionamiento de la temporización y la interrupción, para ello ve a ...

Windows --> Special Function Register
Windows --> Stopwatch...

Resetea el micro y dale a los zapatitos hasta quedar en la siguiente imagen, es decir en la etiqueta **Nada**.



A abrir bien los ojitos que haremos un par de observaciones, primero lo primero...

En Special Function Register

Configuración de puertos, el puerto A no me interesa por que no lo utilizo así que ahí queda, con sus 5 Bit's como entrada (TRISA=00011111), El puerto B está todo como salida (TRISB=00000000), y hemos configurado OPTION_REG como estaba planeado para TMR0 incluso puedes ver el prescaler (los 3 primeros Bit's=111), en el registro INTCON está habilitado **GIE** y **T0IE**, finalmente hemos cargado TMR0 con 216.

En Stopwatch:

Es la primera vez que abrimos esta ventana, y como verás en **Cycles**, tenemos **12**, es decir que hemos ejecutado 12 ciclos de instrucción y estos han consumido 12 microsegundos lo cual puedes ver en **Time**, la frecuencia de procesador utilizada es de 4 Mhz. y está tildado **Clear On Reset**, esto último significa que cuando resetees el micro Stopwatch limpiará todo y lo pondrá a cero, pero como lo que queremos es ver si realmente nuestro programa consume los 10 milisegundos hasta que se desborde TMR0, pues limpiaremos todo a mano, así que dale a **Zero**, y entonces Cycles=0 y Time=0.

Analicemos un poco lo que tiene que ocurrir a partir de ahora, primero que nada, decirte que aquí comienza la temporización de los 10 milisegundo y terminará cuando se produzca la interrupción y salta a ORG 0x04 y como este último es sólo un vector de interrupción pondremos un **Breack Point** en la siguiente línea es decir en...

BTFSS PORTB,0

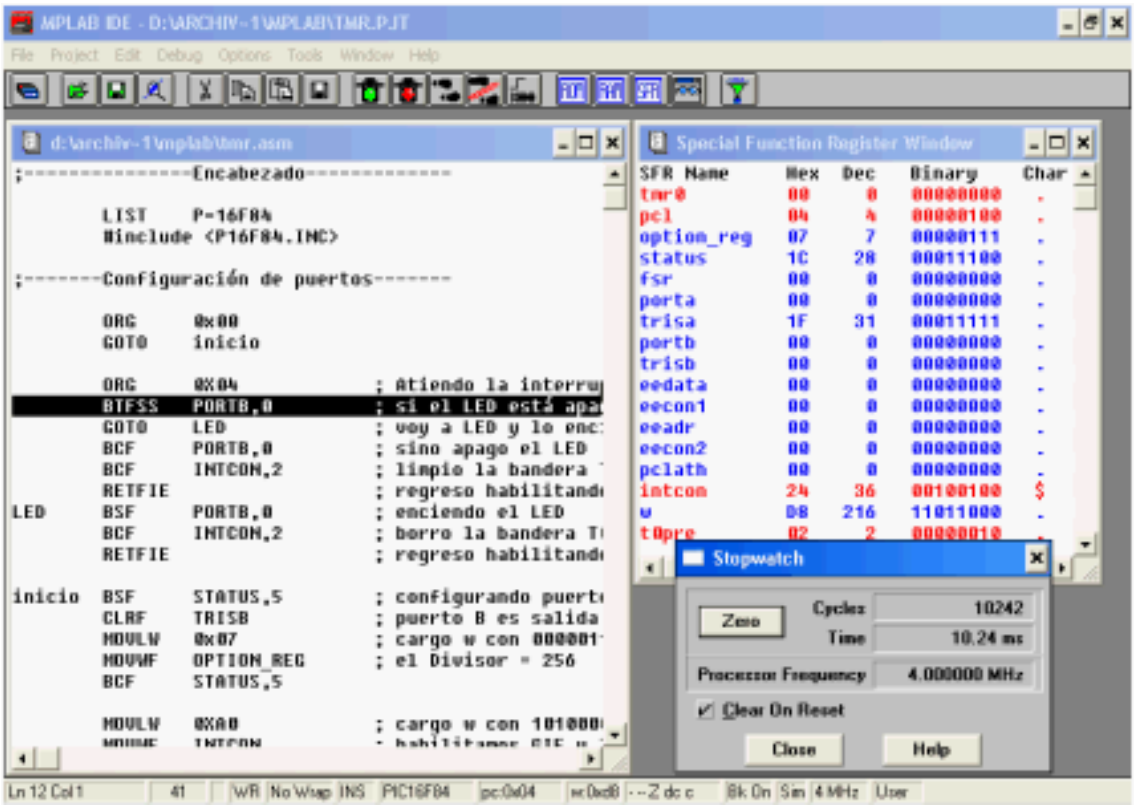
entonces ve a esa dirección y click con el botón derecho, seleccionas **Breack Point(s)** y esta línea se pintará de rojo, lo que hicimos recién es poner un punto de ruptura de tal modo que cuando corramos el programa, comenzará la temporización y cuando se produzca la interrupción, habrán transcurrido los 10 milisegundo y el programa quedará enclavado en **BTFSS PORTB,0**. Si me

seguiste y estás bien hasta aquí, daremos el último paso de la simulación...

Vamos... que esto es mas fácil que cebar mate :o))

Haz click en el semáforo verde de la barra de herramientas o bien, ve a **Debug --> Run --> Run**, y que sea lo que Dios diga...!!!

Por suerte se clavó donde debía, eso significa que estás así...



A observar nuevamente para saber si todo está en su lugar...

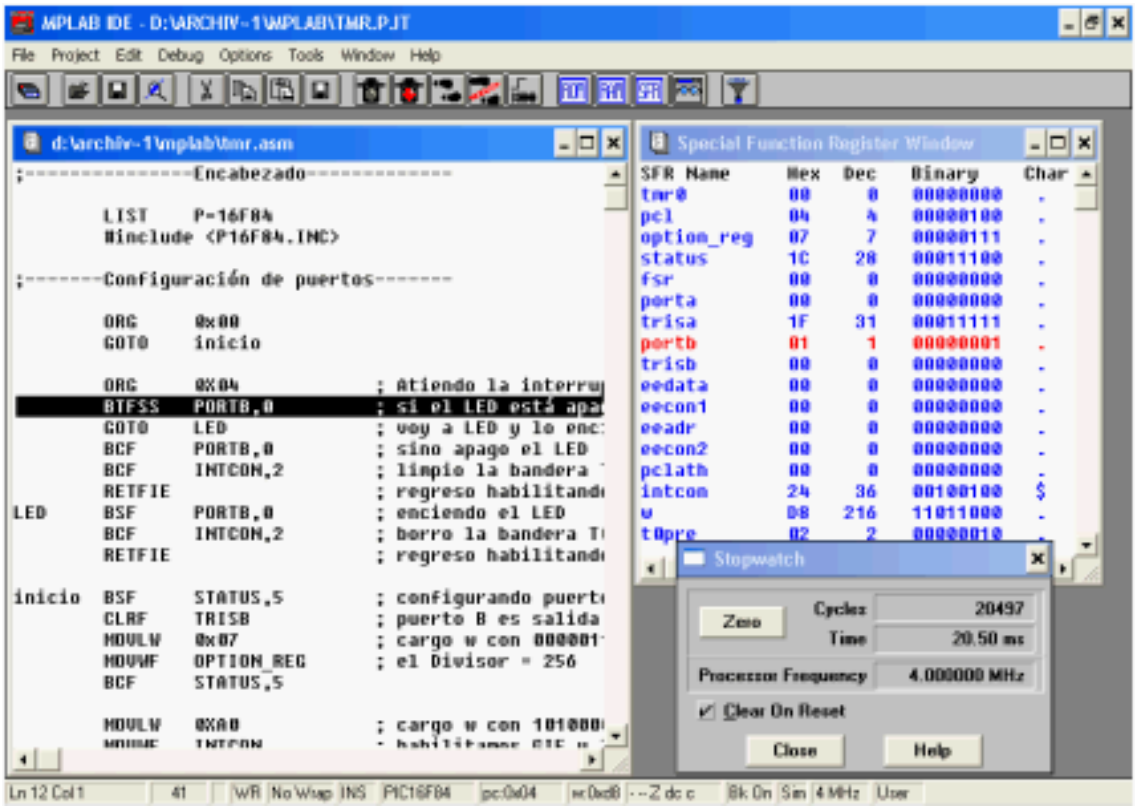
Lo primero que se me ocurre es ver si realmente pasaron los 10 ms

ufaaaaa...!!! me pasé con 242 microsegundos, supongo que debe ser por los decimales, aquellos que aparecieron en los cálculos, pero bueno estamos ahí no crees...???

Sigamos observando... como es lógico el TMR0 se desbordó y está en puros ceros, el registro OPTION_REG no se altera en absoluto y como todavía no encendí el led, PORTB también está en ceros, un último detalle, en el registro INTCON, GIE=0 (interrupciones deshabilitadas), y T0IF=1 (la bandera esta activa)

Te parece si salteamos el Breack Point para dejar que el programa siga corriendo y ver que pasa luego...???

Bien, entonces coloca el cursor donde pusimos el Breack Point y haz click con el botón derecho y luego selecciona **Run To Here** eso hará que el programa continúe, y entonces quedarás nuevamente en el punto de ruptura, así...



Como notarás, pasó lo mismo que hace un momento, sólo que esta vez PORTB tiene encendido el LED o lo que es lo mismo RB0=1 y en Stopwatch, Time=20,50, eso significa que llevamos 2 temporizaciones de 10 ms, yo diría que vamos bien, eso me agrada, si haces **Run To Here** nuevamente, Time se incrementará en 10 ms más y quedará en 30,75 y lo mejor de todo es que el LED se apagará, es decir RB0=0.

Oye...!!!, nos merecemos un aplauso no crees...???, después de tanto despelote con fórmulas, interrupciones y todo eso, estamos listos para temporizar.

En resumidas cuentas... el lío de ecuaciones anterior, si trabajas con un XT de 4 Mhz, se resume en...

Temporización = (255-TMR0) * Divisor de Frecuencia

Esta es la ecuación que utilizaré en todas las temporizaciones ya que siempre trabajo con un cristal de 4 Mhz.

Te imaginas que pasaría si utilizara un registro auxiliar y le cargara el valor 0x64 (100 en decimal) para luego decrementarlo cada vez que se desborda el TMR0...???

Muy sencillo, como el TMR0 se desbordará 100 veces tendrás...

Temporización = 100 * 39 * 256

998400

Siiii...!!!, prácticamente 1000000 de microsegundos, o sea 1 segundo

Teóricamente esto debería cumplirse, pero por una u otra razón, en la realidad no es tan así, y no queda otra que ajustar las cosas a mano.

OK., hasta aquí llegamos...!!!, por cierto y para aquellos que no se convencen con simple teoría, lean...

Les enseñaré un pequeño truco que cierta vez aprendí observando una lámpara de neón, creo que como profesor de ciencias, ser observador, tiene sus ventajas ;oP

Arma el circuito como siempre lo hacemos, y colócale una resistencia de 220R y un LED en RB0, luego grabas el programa en el PIC, lo montas en el circuito y como dije anteriormente, el LED permanecerá encendido, pero si lo mueves de un lado a otro, verás como parpadea ese LED, debes ser un buen observador, yo lo acabo de hacer, y funciona de diez.

Bueno, que puedo decir, creo que fue suficiente por hoy...

Más formas de Temporizar

Puedes hacer la misma temporización sin utilizar interrupciones, es decir sin habilitarlas, y luego sólo chequear la bandera T0IF, ya que ésta cambiará siempre que se desborde el registro TMR0, y para analizar la bandera nuevamente la vuelves a limpiar, veamos un ejemplo

Haz el código que quieras y luego llama (Call) a una rutina de retardo, que la puedes etiquetar **retardo**, y cuando la temporización termine le colocas un RETURN, así...

```
            .
            .
            Call      retardo
            .
            .
retardo     BCF       INTCON,2   ; limpias bandera
            MOVLW     0xD8       ; cargas w con 216
            MOVWF     TMR0       ; lo pasas a TMR0
chequear    BTFSS     INTCON,2   ; chequeas T0IF
            GOTO      chequear   ; hasta que TMR0 desborde, y entonces
            RETURN               ; retornas del retardo
            .
            .
```

En este ejemplo no fue configurado INTCON (para evitar las interrupciones) pero sí, el registro OPTION y del mismo modo que lo hicimos anteriormente, con el mismo prescaler, el resto lo dejo en tus manos, que más...!!!, esto sirve de práctica, y el que lo haga, lo manda y en la próxima actualización lo incluimos en la web.

Para seguir con el tema, vamos a tomar de ejemplo una rutinas similar a la que hemos utilizado en la mayoría de los programas que se encuentran en la web y que cumplen la función de generar un retardo cualquiera sin hacer uso del registro TMR0, sino más bien utilizando dos registros alternativos. La idea es averiguar cual es el tiempo que consume esa rutina...

Y aquí la tenemos en vivo...

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

retardo  MOVLW  0x10    ; Aquí se cargan los registros
         MOVWF  reg1    ; reg1 y reg2 con 0x10 y 0x20
dos      MOVLW  0x20
         MOVWF
uno      DECFSZ reg2,1  ; Aquí se comienza a decrementar
         GOTO   uno     ; Cuando reg2 llegue a 0
         DECFSZ reg1,1  ; le quitare 1 a reg1
         GOTO   dos     ; iré a cargar reg2 nuevamente
         RETURN         ; Si reg1 termino de decrementar
                        ; regreso del retardo

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

Lo que hace este código es lo siguiente; primero carga ambos registros reg1 y reg2, luego comienza a decrementar reg2, cuando éste llega a cero, resta 1 a reg1 y carga nuevamente reg2 y lo decrementa, y cuando llega a cero nuevamente, vuelve a restarle 1 a reg1, en síntesis reg1 se decrementará en 1 cada vez que se vacíe reg2, si te das cuenta reg1 es igual a 0x10 (0x10 = d'16'), por lo tanto reg2 se decrementará 16 veces antes de que reg1 llegue a cero.

Bien, veamos ahora cuanto tiempo consume esta rutina de retardo, y para poder hacerlo deberemos tener en cuenta la cantidad de ciclos de instrucción de cada una de las instrucciones que se encuentran en esta rutina, por lo general la mayoría de las rutinas consumen un ciclo de instrucción salvo cuando realizan un salto, bueno pues en ese caso consume dos ciclos de instrucción. Aquí tienes una tabla con los ciclos de instrucción por si te hace falta.

De las instrucciones que están en rojo; la primera consume un ciclo mientras no se realice el salto y no saltará por lo menos 14 veces (d'14' = 0x20) hasta que reg2 se haga cero, pero la segunda consume dos ciclos y también se repite 14 veces, entonces...

rutina uno = 3 * 14 ciclos = 42 ciclos

De las instrucciones que están en verde; la primera y la segunda instrucción consumen un ciclo y se repiten 16 veces (d'16' = 0x10) es decir reg2 se carga 16 veces antes de que se vacíe reg1, las dos instrucciones restantes consumen 1 y 2 ciclos de instrucción respectivamente y también se repiten 16 veces.

Por tanto tenemos 5 ciclos de instrucción que se repiten 16 veces mas la rutina **uno** que se encuentra incluida dentro de la rutina 2, es decir...

rutina dos y uno = (42 + 5) * 16 = 752 ciclos

Hasta aquí diría que ya estamos, pero el tema es que, cada vez que sales de la rutina que está en rojo e ingresas a la que está en verde, lo haces con un salto es decir 2 ciclos de instrucción y lo haces 16 veces, es decir 32 ciclos más, y por otro lado, desde que la rutina de retardo se inicia, pasaron 4 instrucciones, antes de entrar a la etiqueta uno, eso significa que al total de ciclos de instrucción, hay que agregarle 36 ciclos más, y tendrás en total 788 ciclos.

Si trabajas con un XT de 4 Mhz cada instrucción se ejecuta en un microsegundo, por lo tanto esta rutina de retardo demora **788 microsegundos**

La verdad es que no se cual de las dos temporizaciones es más precisa, si haciendo

uso del TMR0, o del modo que acabamos de ver, lo que si creo, es que debes tener en cuenta todos los detalles posibles al hacer los cálculos, tanto en uno como en otro tipo de temporización, lo mejor es evitar los decimales y trabajar en lo posible con números enteros, esto lo digo por las temporizaciones con el registro TMR0, respecto a lo último que acabamos de ver, debes tener mucha precaución en todas las instrucciones utilizadas, para que así no se te escape ninguna.

A demás, con esta última forma de temporizar puedes utilizar el TMR0 para otra cosa, no se... queda a tu criterio, de todos modos ya lo viste y sabes como utilizarlo

Viste Cesar...??? Tarea cumplida...!!!

Bien mis queridos amigos, este tutorial está a punto de dar sus últimos respiros...

Espero haber dado respuesta a muchas de sus dudas, y recuerden que soy un simple terrícola, y que si en algo la erré... sería bueno saberlo, para así corregir los errores y continuar aprendiendo todos juntos.

:: PIC - Parte III - Capítulo 10

Un interesante proyecto

Cierta vez, me llamaron de una empresa para reparar un circuito que no funcionaba correctamente, por suerte... me pasaron dos circuitos similares, pero con distinta falla, el tema es que ambos estaban comandados por un microcontrolador muy similar al PIC16F84 (feliz yo...!!!), pero este bicho tenía dos patas demás, y lo peor de todo, es que de fábrica le habían borrado los datos del integrado (a llorar...!!!, pero me las pagarán...!!!), de buenas a primeras, la solución fue sencilla, sólo cambié el micro fallado por el del otro circuito, Arreglé unooooooo...!!!.

Pero esto no se quedaría así..., y decidí hacer uno, pero para el PIC16f84 así es que... a prepararse que les comentaré de que se trata...

El proyecto...!!!

Le pondré de nombre porton.asm (sin acento), y consiste en lo siguiente...

Se trata de comandar el portón de una cochera, y con las siguientes funciones. Suponiendo el análisis desde que el portón está cerrado...

Con un pulsador, llamémosle **"abrir"** el portón comienza a elevarse, al mismo tiempo, se enciende un semáforo pegado en la pared de calle, el cual indica la apertura del portón y que en cualquier momento sacaré o guardaré mi convertible :o))

Una vez concluida la apertura comienza a gritar un pequeño timbre intermitente.

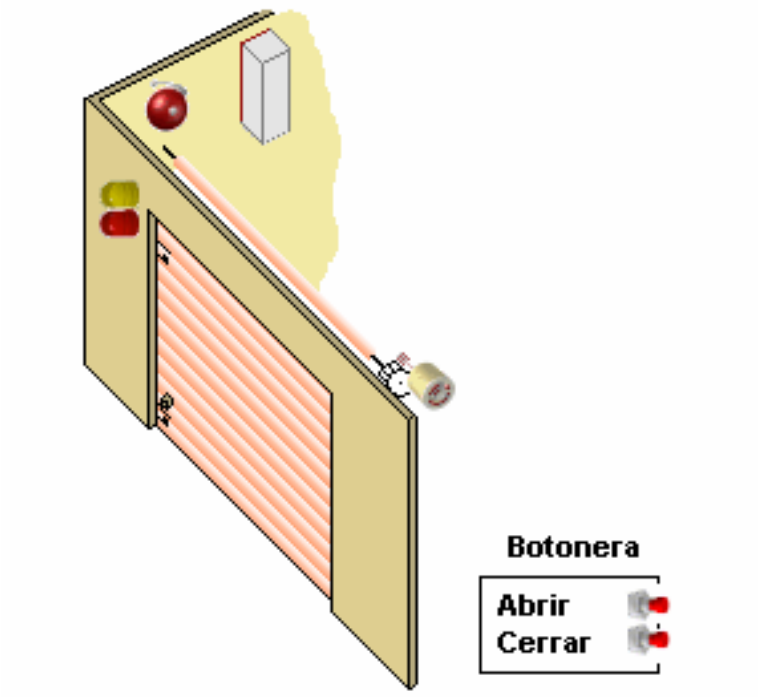
Con un segundo pulsador, llamémosle **"cerrar"** lo primero que ocurre es que el timbre se apaga, el portón comienza a cerrarse y una vez concluido el cierre, el semáforo se apaga y se enciende la luz del interior de la cochera, la cual permanece encendida por un período de 50 segundos, esto último es debido a que cuando guarde mi ferrari la cochera quedará a oscuras.

Está fácil verdad...!!!

Bueno, pero la vamos a complicar un poquito más, cuando el portón termine de abrirse deberá activar un contacto (pulsador) al cual le llamaremos **Dabrir** (Detiene Abrir), por otro lado, también colocaremos otro contacto para cuando termine de cerrarse y lo llamaremos **Dcerrar** (Detiene Cerrar), estos dos pulsadores vendrían a ser los **fines de carrera** del portón. Finalmente le agregaremos un pulsador de Reset (*ese que siempre ponemos en todos los circuitos con pic*)

Como ya veo el despiole que se harán para entender todo lo que debemos programar, me ví obligado a hacer una animación con todas las funciones que el micro deberá realizar, (*lo que si olvide incluir es el pulsador de Reset, pero de*

todos modos lo tendremos en cuenta al codificar). Bien, actualiza la página si lo quieres ver...



Creo que ahora está todo más claro...

Antes de empezar a codificar haremos una revisión del funcionamiento de cada uno de los circuitos que se deben incorporar, los cuales serán comandados por el microcontrolador.

Comencemos por el semáforo, éste puede construirse con un 555, y ya lo vimos en el tutorial de electrónica básica, incluso vimos unos cuantos que pueden ser controlados con una señal lógica (del tipo 1-0), esto último lo vimos en el tutorial de electrónica digital (haciendo uso de compuertas lógicas).

Lo siguiente es el timbre, y como no soy muy amigo del área de audio, pues que más, busca por la web, apuesto que conseguirás muchos circuitos, y si tienes uno que sirva para este fin, puedes enviarlo y así completamos este proyecto. Una posibilidad es hacerlo con un 555 pero con una frecuencia que te permita generar sonidos.

Respecto a la luz interna de la cochera, lo más fácil, podemos hacerlo con un relé y listo...

Ahora lo más complejo, como poner en marcha el motor...?

No se mucho de motores de corriente alterna, pero veamos, el motor con el que hice las pruebas es un motor monofásico obviamente con capacitor y lo más interesante es que posee 4 cables (A, B, C y D), estos se unen de a pares (PAR_{AB} y PAR_{CD}), cuando estos se conectan a la red domiciliaria el motor gira en un sentido, y si intercambias un cable de cada par (suponte PAR_{AC} y PAR_{BD}) obtendrás el giro del motor en sentido opuesto, y ahora me van a matar...!!!, supongo que te estarás preguntando... cual es A, cual B, C y D...???, pues no lo sé, la verdad es que cuando yo lo probé y vi que el motor cambió de giro me puse tan feliz que no busqué más..., de todos modos para mí, esto es a prueba y error...

Por supuesto que si alguien lo sabe sería bueno que lo comente y luego le agregamos en esta página...

Lo que haremos ahora será analizar un poco lo que deberemos codificar

:: PIC - Parte III - Capítulo 11

Analizando entradas y salidas

Entradas

En la animación anterior vimos dos pulsadores ubicados en una botonera a los cuales les llamé **Abrir** y **Cerrar**, el nombre indica la función que cumplen así que no diré nada de ellos, luego tenemos los fines de carrera, que no son otra cosa que contactos (sensores) que se activan cuando el portón termina de abrirse (**Dabrir**) o cerrarse (**Dcerrar**). La configuración para estas entradas será la siguiente...

- Abrir <-- RB0/INT (pin6) Pulsador // 0=no_pulsado 1=pulsado
- Cerrar <-- RA0 (pin17) Pulsador // 0=no_pulsado 1=pulsado
- Dabrir <-- RA1 (pin18) Sensor // 0=no_pulsado 1=pulsado
- Dcerrar <-- RA2 (pin1) Sensor // 0=no_pulsado 1=pulsado

Te preguntarás... por qué no estoy utilizando todos los pines del puerto A..???, y es que la mayor parte del tiempo ésta permanecerá con la cochera cerrada, razón por la cual usaremos la instrucción SLEEP para dormir al micro todo el tiempo que no esté en uso, y lo sacaremos de ese estado con la interrupción por el pin RB0 (lo que vimos anteriormente en este mismo tutorial), luego nos queda RA0, RA1 y RA2 que lógicamente son las típicas entradas.

Salidas

Un detalle a tener en cuenta es que para cambiar el sentido de giro del motor, es preferible hacerlo siempre que éste no se encuentra en movimiento, así que agregué una salida más (Tensión) para quitar la tensión del motor, es más, haré un pequeño retardo (*para evitar desastres...!!!*), Suponte que el motor sube (Tensión=1 y Gmotor=0), para hacer que este baje primero deberás quitar la alimentación del motor (Tensión=0), luego invertir el sentido de giro del motor para que baje (Gmotor=1), esperar un momento y recién entonces ponerlo en marcha (Tensión=1), me explico...???

Ok. ahora sí, uno de los pines (Tensión) activará o desactivará un relé que envía tensión al motor, el segundo utilizará un relé para encender y apagar el Semáforo, el tercero hará lo mismo con el timbre, el cuarto trabajará con dos relés para cambiar el giro del motor (*también podría ser un sólo relé de 8 patas*), la quinta y última salida encenderá la luz de la cochera (*podría ser un relé o un Triac*). La configuración de estos pines será la siguiente...

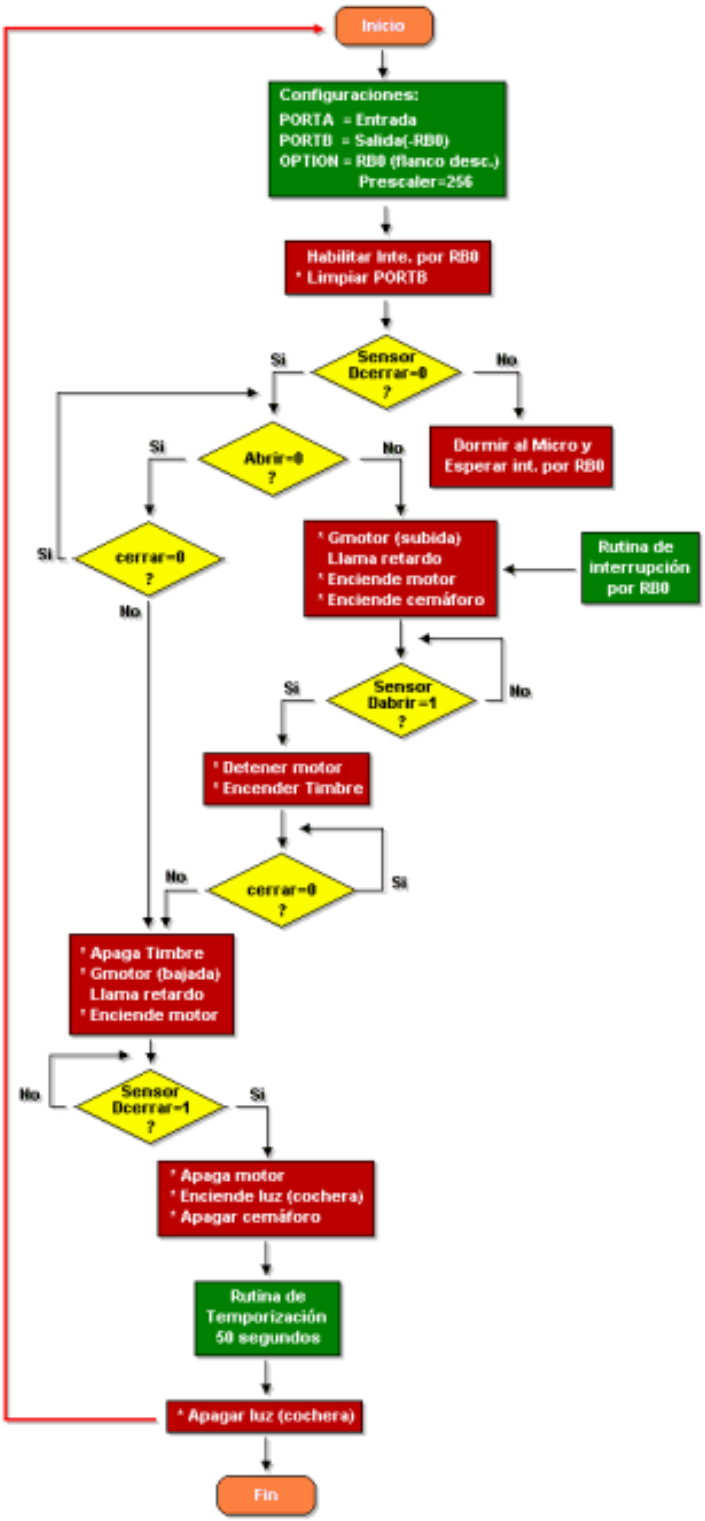
- Tensión <-- RB1 (pin7) Alimentación-Motor // 0=apagado 1=encendido
- Semáforo <-- RB2 (pin8) ---- // 0=apagado 1=encendido
- Timbre <-- RB3 (pin9) ---- // 0=apagado 1=encendido
- Gmotor <-- RB4 (pin10) Sentido de Giro // 0=sube 1=baja
- Luz <-- RB5 (pin11) Luz-Cochera // 0=apagado 1=encendido

Ahora haremos un par de observaciones para el buen funcionamiento de la tarjeta de control. Si bien el funcionamiento del pic será secuencial imagínate que pasaría si hoy decido lavar mi móvil, la verdad es que no soportaría el escándalo del timbre ni al semáforo prendiendo y apagando todo el tiempo, pues para eso está el pulsador de reset, por lo tanto cuando sea activado detendré todo el proceso limpiando el puerto B, lo cual haremos justo cuando el programa arranque, es decir en el inicio del código, Entonces... buscaremos el portón, si éste está cerrado, duermo al micro pero si no lo está deberá esperar una instrucción, ya sea abrir o cerrar.

Pero para comprender mejor lo que debemos codificar lo analizaremos con un par de diagramas de flujo (*como hacen los grandes programadores*...)

Diagrama de flujo del código principal

Creo que esta forma de analizar un código simplifica demasiadas explicaciones ya que todo está a la vista. Veamos, lo que está en amarillo es el análisis de pulsadores y sensores, lo que está en rojo y con un asterisco es la activación o desactivación de los pines del puerto B, y lo que no tiene asterisco son simples instrucciones para el programa o llamadas a subrutinas, aquello que está en verde... bueno, la primera es configuración inicial de los puertos, luego una rutina de interrupciones, que una vez atendida regresa el control al flujo del programa. Finalmente la temporización de los 50 segundos, que una vez concluida apaga la luz de la cochera y regresa al inicio del programa para luego hacer dormir al micro con la instrucción SLEEP. Haz click en la siguiente imagen para ampliar el diagrama...



Lo que no se encuentra en este diagrama de flujo es la pequeña rutina de retardo que se utiliza para encender el motor luego de cambiarle el sentido de giro, Tampoco se incluyó la temporización de los 50 segundos porque sino se haría demasiado extenso el diagrama de flujo, pero no te preocupes que lo analizaremos aparte, y nos falta algo más... La rutina de servicio de interrupciones (ISR), que también la analizaremos por separado

Analiza el diagrama de flujo, creo que no requiere más explicación.

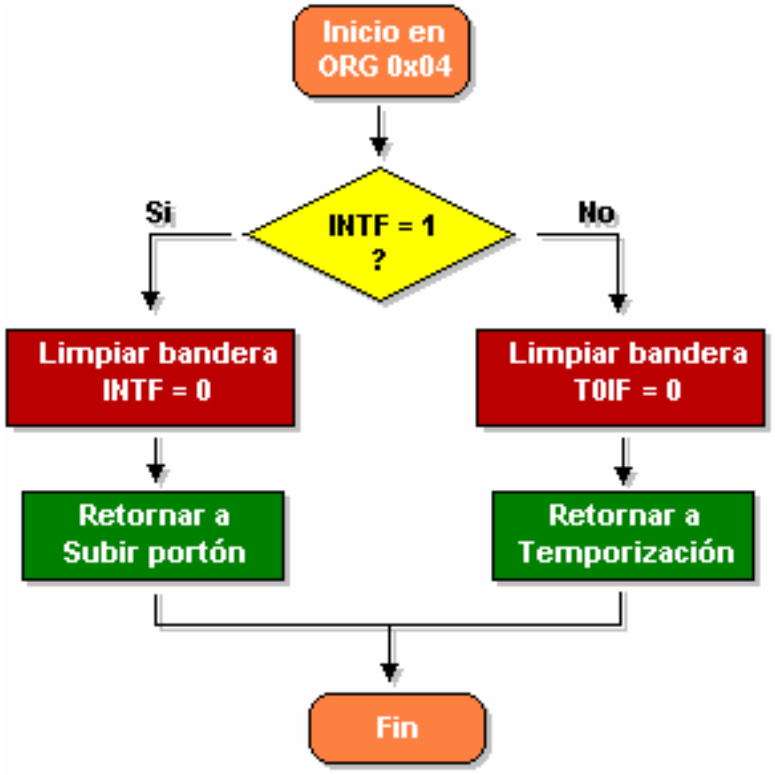
Y ahora sí... nos haremos cargo de lo que nos está faltando, que es justamente a lo que apunta esta actualización...

Comenzaré con la rutina de interrupciones, luego la temporización de los 50 segundos y finalmente el pequeño retardo, de acuerdo...???

:: PIC - Parte III - Capítulo 12

Diagrama de flujo para el control de Interrupciones

Como verás la ISR o rutina de servicio de interrupciones es muy pero muy corta, ya que sólo se limita a limpiar las banderas de interrupción y regresar el control al flujo del programa, pero veamos... son dos las fuentes de interrupción, una externa debido a una señal enviada por RB0 que activa la bandera INTF para sacar al micro del modo SLEEP y así iniciar con el flujo del programa, la otra es interna y debido al desbordamiento del registro TMR0 que activa la bandera T0IF, la cual se limpia y luego regresa a la temporización de los 50 segundos (*ya que de allí es que vino*) para cargar TMR0 nuevamente, siempre que el registro Reg1 no haya llegado a cero.



Te preguntarás ahora... con que finalidad utilicé interrupciones?, si lo único que hago es limpiar las banderas cuando estas se producen?.

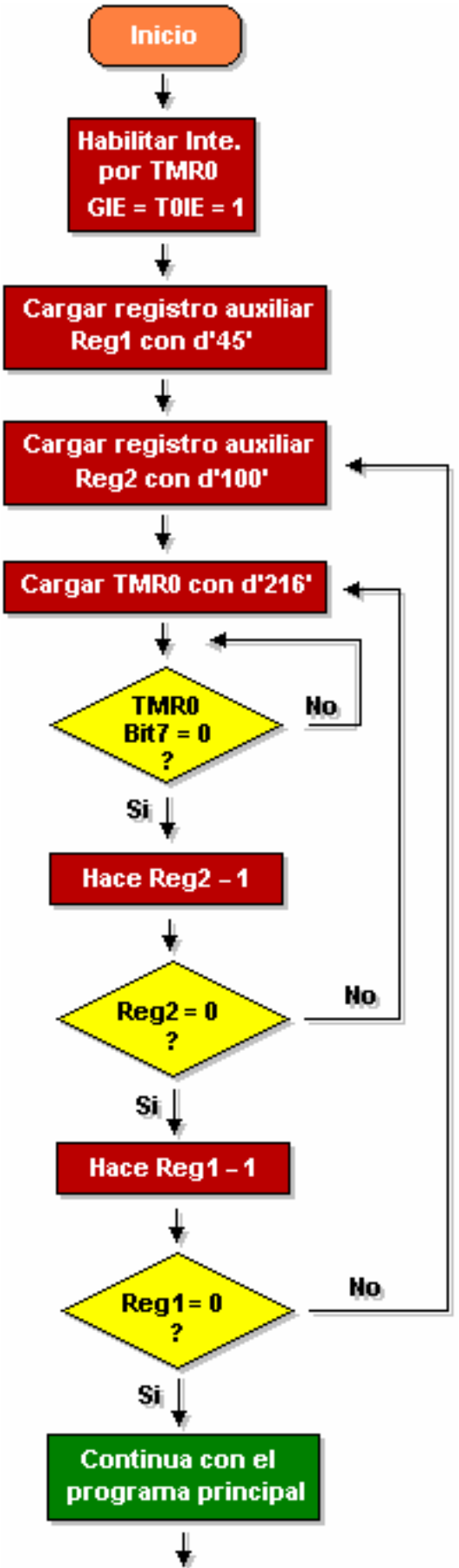
Bueno, no me cuestiones todavía, como dije antes... la mayor parte del tiempo el portón permanecerá cerrado, por lo cual se lo pone en modo de bajo consumo (con la instrucción SLEEP), y se lo saca de ese estado con una interrupción por RB0.

La interrupción por TMR0 es por darle la mayor exactitud posible a los 50 segundos, y a demás por aplicar algo de lo que vimos en este tutorial.

De todos modos lo hice YO...!!!, que tanto...!!!

Diagrama de flujo - Temporización de 50 segundos

Antes que nada voy a aclarar una cosa... En esta rutina voy a pasar un nuevo valor para el registro INTCON habilitando GIE y T0IE y a la vez deshabilitando la interrupción por RB0, ya que lo único que haré ahora será esperar a que se cumplan los 50 segundos...



Sencillo no...??? bueno, si tuviéramos que hacer los cálculos para obtener los 50... lo haríamos aplicando la ecuación que vimos anteriormente, o sea...

Temporización = (255-TMR0) * Divisor de Frecuencia

Aquí... el divisor de frecuencia es 256 ya que hemos configurado en el registro OPTION los valores **111** para PS2, PS1 y PS0 (el prescaler), por otro lado, la diferencia 255-TMR0 es 39, ya que TMR0 = 216, Teniendo todos estos datos, ya podemos obtener la temporización, la cual vale...

9984 us = 39 * 256

Aproximadamente un milisegundo, pero este **se repite 100 veces** debido a Reg2, razón por la cual obtenemos prácticamente un segundo, y como lo que buscamos es obtener un retardo de 50 segundos pues simplemente usamos un registro más (ya que el anterior no nos alcanza) y **repetimos toda esa rutina 50 veces** (y es lo que se debería cargar en Reg1), te preguntarás porque entonces cargué 45...???, la respuesta es muy simple, la cantidad de saltos en toda esta rutina consumen demasiados ciclos de instrucción y al probar este retardo se me atrasaba casi 7 segundos, pues es eso justamente sólo hice un pequeño ajuste.

Bien, veamos como sería el cálculo si cargaría Reg1 con 50...

Temporización = Reg1 * Reg2 * 39 * 256
49.920.000 useg. = 50 * 100 * 39 * 256

Que bueno no...!!!

La otra rutina de retardo es prácticamente lo mismo sólo que el tiempo que consume es de aproximadamente 189.000 useg. es decir casi la quinta parte de un segundo, y como ya lo analizamos anteriormente pues... no la incluiré...

Se viene lo mejor... el código en vivo...!!!

Código para el control de portón, semáforo, timbre y luz de cochera

Esta vez no haré comentarios respecto al código ya que habla por sí sólo y a demás ya lo analizamos bastante, no sea cosa que te me duermas... jaja...

```
;---------------- Encabezado --------------------
        LIST    P=16F84
        #include <P16F84.INC>

;----------- Registros de tiempo ----------------
reg1    EQU     0x0C
reg2    EQU     0x0D

;-------------- Vector de Reset -----------------
        ORG     0x00
        GOTO    inicio

;----------Rutina de interrupción (ISR)----------
        ORG     0x04
        BTFSS   INTCON,1        ; salta si la interrup. es por RB0
        GOTO    tmr             ; sino, es por TMR0 y ahí lo atiende
        BCF     INTCON,1        ; limpia bandera INTF
        RETFIE                  ; retorno de interrupción
tmr     BCF     INTCON,2        ; limpia bandera de T0IF
        RETFIE                  ; retorno de interrupción

;----------- Configuración de puertos -----------
inicio  BSF     STATUS,RP0
        MOVLW   0xFF            ; carga w con 1111 1111
        MOVWF   TRISA           ; PORTA es entradas
        MOVLW   0x01            ; carga w con 0000 0001
        MOVWF   TRISB           ; RB0=entrada, el resto salida
        MOVLW   0X47            ; carga w con 0100 0111
        MOVWF   OPTION_REG      ; RB0=flanco ascendente, prescaler=256
        BCF     STATUS,RP0

;------- Habilitación interrupción por RB0 -------
        MOVLW   0x90            ; habilitamos interrupciones 1001 0000
        MOVWF   INTCON          ; GIE e INTE(para RB0)
```

```
        CLRF    PORTB           ; limpio el puerto B

;------------ El portón está abierto? ------------
        BTFSC   PORTA,2         ; salta si Dcerrar=0 (portón abierto)
        GOTO    dormir          ; sino (portón cerrado) va a dormir

;------- Si está abierto espera instrucción ------
espera  BTFSC   PORTB,0         ; salta si no se pulsa Abrir
        GOTO    abrir           ; sino, fue pulsado y lo atiende en abrir
        BTFSC   PORTA,0         ; salta si no se pulsa cerrar
        GOTO    c_rrar          ; sino, fue pulsado y lo atiende en c_rrar
        GOTO    espera

;-------- micro en estado de bajo consumo --------
dormir  SLEEP                   ; Espera interrupción por RB0

;--------- Rutina para abrir el portón -----------
abrir   BCF     PORTB,4         ; prepara motor para subir
        CALL    retardo         ; hace un pequeño retardo
        BSF     PORTB,1         ; enciende motor (abre)
        BSF     PORTB,2         ; enciende semáforo
d_abrir BTFSS   PORTA,1         ; salta si se pulsa Dabrir (sensor)
        GOTO    d_abrir         ; sino lo atiende en d_abrir
        BCF     PORTB,1         ; Dabrir=1, entonces detiene motor
        BSF     PORTB,3         ; y enciende timbre
espero  BTFSC   PORTA,0         ; salta si no se pulsa cerrar (pulsador)
        GOTO    c_rrar          ; sino, fue activado y lo atiende en c_rrar
        GOTO    espero          ; espero que se pulse cerrar

;--------- Rutina para cerrar el portón -----------
c_rrar  BCF     PORTB,3         ; apaga timbre
        BSF     PORTB,4         ; invierte motor para bajar
        CALL    retardo         ; hace un pequeño retardo
        BSF     PORTB,1         ; enciende motor
dcerrar BTFSS   PORTA,2         ; salta si se activa Dcerrar (sensor)
        GOTO    dcerrar         ; sino, espero que se active
        BCF     PORTB,1         ; Dcerrar=1, entonces detiene motor
        BSF     PORTB,5         ; enciende luz de cochera
        BCF     PORTB,2         ; apaga semáforo

;------- Rutina de temporización 50 segundos -------
        MOVLW   0xA0            ; habilita T0IE interrupción por TMR0
        MOVWF   INTCON
        MOVLW   0x2D            ; cargo w con d'45'=0x2D
        MOVWF   reg1            ; lo pasa a reg1
tiempo1 MOVLW   0x64            ; cargo w con 100
        MOVWF   reg2            ; y lo pasa a reg2
tiempo  MOVLW   0xD8            ; cargo w con 216
        MOVWF   TMR0            ; lo pasa a TMR0
```

```
nada    BTFSC   TMR0,7          ; Salta si Bit7 de TMR0 es cero
        GOTO    nada            ; sino espera interrupción por TMR0

        DECFSZ  reg2,1          ; decremento reg2 y salta si reg2=0
        GOTO    tiempo          ; sino vuelve a cargar TMR0
        DECFSZ  reg1,1          ; decrementa reg1 y salta si reg1=0
        GOTO    tiempo1         ; sino vuelve a cargar reg2
        BCF     PORTB,5         ; 50 seg. cumplidos apago luz de cochera
        CLRF    INTCON          ; deshabilito interrupciones
        GOTO    inicio          ; vuelve a empezar para dormir al micro

; ------ retardo de 189000 us. aproximadamente --------
retardo MOVLW   0xFA            ; Aquí se cargan los registros
        MOVWF   reg1            ; carga reg1 con 250
dos     MOVWF   reg2            ; y reg2 con 250
uno     DECFSZ  reg2,1          ; decrementa y salta si reg2=0
        GOTO    uno
        DECFSZ  reg1,1          ; decrementa y salta si reg1=0
        GOTO    dos             ; irá a cargar reg2 nuevamente
        RETURN                  ; regreso del retardo
;----------------------------------------
        END
;----------------------------------------
```

Observa aquello que está en rojo...

```
CLRF    INTCON    ; deshabilito interrupciones
GOTO    inicio    ; vuelve a empezar para dormir al micro
```

Eso de deshabilitar las interrupciones lo hice por que ya no será necesaria la interrupción por TMR0 por que terminó la temporización, y lo de regresar al inicio del código es por permitir la habilitación de la interrupción para RB0 y finalmente poner al micro en modo SLEEP.

En la temporización de los 189 milisegundos hay algo muy curioso...

```
retardo MOVLW   0xFA        ; Aquí se cargan los registros
     MOVWF   reg1         ; carga reg1 con 250
dos    MOVWF   reg2         ; y reg2 con 250
```

Primero se cargó w con d'250' y luego se pasó el valor a reg1 y a reg2, mientras el decremento en la temporización se ejecuta, hay un salto a la etiqueta **dos**, donde sólo se hace **MOVWF reg2**, Lo curioso es que no se carga nuevamente el registro w con el valor 250 para pasarlo a reg2, y esto es así por que el registro w no fue utilizado para nada mientras se ejecutaba el retardo, por lo tanto w aún retiene los 250 que le cargamos anteriormente. Cuando hagas esto, debes tener mucho cuidado y asegurarte de que no utilizaste w y que todavía tiene el valor anterior, de lo contrario tendrás muchos problemas...

Bueno creo que no queda nada más, o mejor dicho sí, como hacer para conectar los relés, en otras palabras... el circuito...!!!

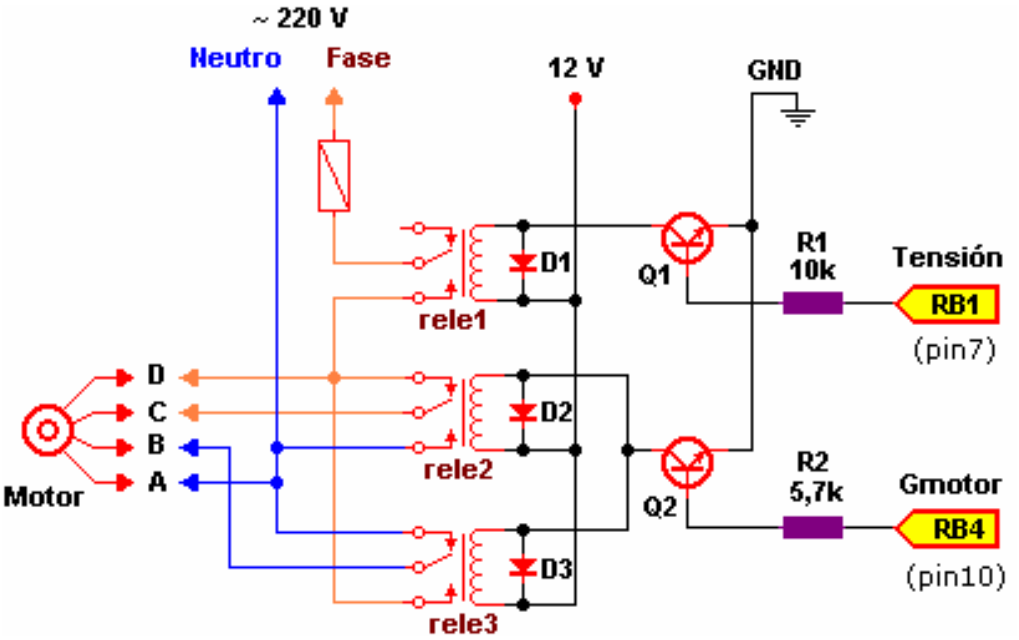
Hablaremos de él en la siguiente página...

:: PIC - Parte III - Capítulo 14

Los esquemas eléctricos

Etapa de Relés para el control del motor

El primer esquema es el relacionado con el encendido y control de giro del motor, en él se incluyen 3 relés, uno para el encendido del motor y los otros dos para invertir los cables de cada par del motor...

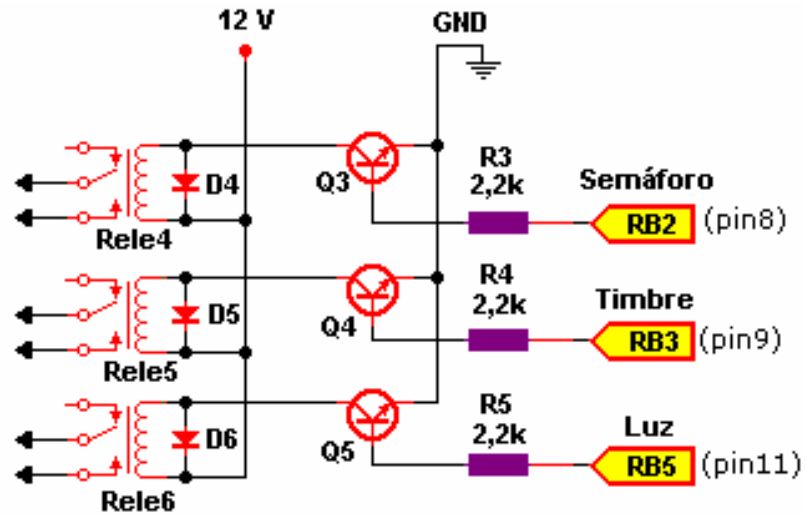


Los pares del motor son AB y CD, observa que los cables A y D permanecerán sin cambio mientras que los únicos que cambian son los cables B y C, cuando uno de ellos es **fase** el otro es **neutro**, y viceversa.

En esta etapa se está trabajando con dos niveles de tensión, la primera es de 220 v y es alterna, justamente lo que requiere el motor para funcionar, la segunda es de 12 v y continua, es lo que requiere el relé para activar sus contactos, a la vez se pueden ver los terminales que deberán ser conectados al micro.

Etapa de Relés para Semáforo, Timbre y Luz.

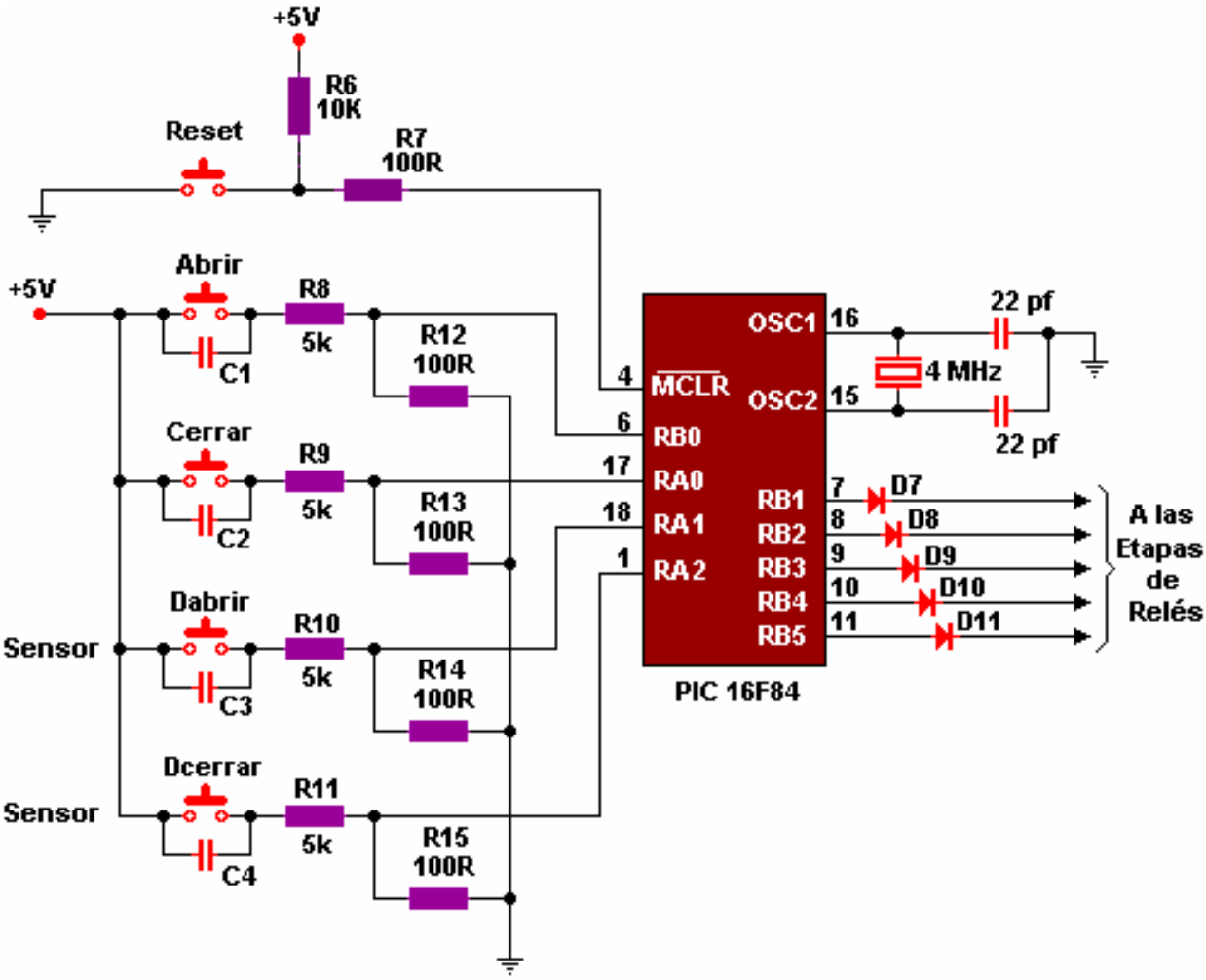
Aquí se encuentran visibles el resto de los relés que controlarán el encendido del Semáforo, Timbre y la Luz interior de la cochera...



Como podrás notar, en este esquema no se encuentra ni el circuito de Semáforo, ni el del timbre ni la lámpara de la cochera, ya que son circuitos externos a la placa de control, nada quita que lo incluyas si lo deseas.

En ambos esquemas se trabaja con transistores NPN del tipo BC337, por lo tanto la señal enviada a los transistores para que estos se activen debe ser positiva. Los Diodos son los mismos que utilizamos en las fuentes de alimentación, los 1N4007.

Te comento que estos esquemas fueron sacados de la misma placa que me enviaron para reparación, y si bien en ellos figuran los pines que deberán ir al microcontrolador deberás prever riesgos, y antes de entrar en las etapas de relés colocar un diodo, es más sería bueno intercalar un Driver entre esta etapa y el micro que bien podría ser el ULN2803 o un Buffer como el CD40106, pero recuerda que estos invierten su señal de salida esto es, si envias un 1 lógico el integrado entrega un 0 lógico en su salida. Yo no incluiré estos integrados en el siguiente esquema pero sí los diodos, así que aclarado esto, veamos como queda el esquema para el micro...



Los capacitores C1 a C4 son de cerámica de 0,1 uf, éstos son para prevenir los rebotes ya que en el código del programa no se hizo ninguna rutina de retardo para evitar rebotes. Los diodos que van a la etapa de relés son del tipo 1N4148, y finalmente habrá que tener en cuenta la fuente de alimentación, que lógicamente debe ser de 5V.

Una cosa más **Dabrir** y **Dcerrar** que figuran como pulsadores en el esquema no son otra cosa que sensores.

Creo no haber olvidado nada, unas palabras finales y damos por concluido este tutorial hasta la próxima actualización...

:: PIC - Parte III - Palabras Finales

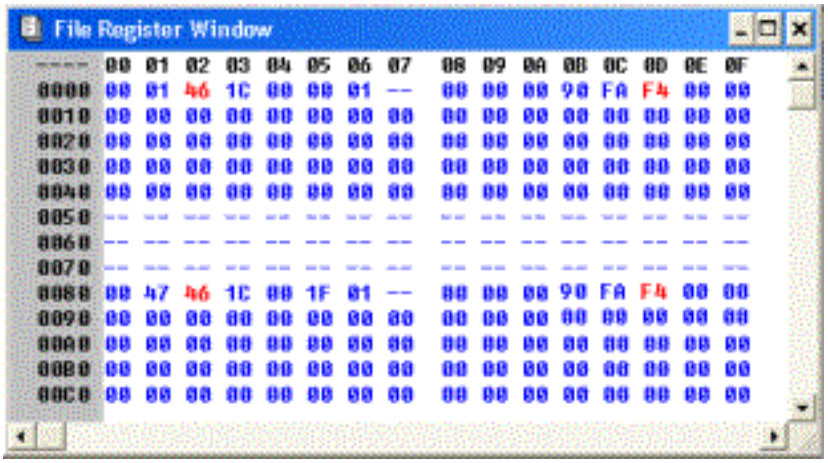
Palabras Finales

Advertí que esto pondría **de pelos...!!!** a muchos, pero si estuviste siguiendo los tutoriales no te debería traer grandes problemas, ***Palabra mía...!!!***, ya que estás en condiciones de comprenderlo fácilmente, a demás es sólo una cuestión de interpretación, y de paso vamos incorporando algunas cositas que todavía no habíamos visto, como ser los diagramas de flujo, los cuales son muy utilizados al momento de codificar, éstos evitan que te pierdas, al menos a mí me ayuda bastante...

Respecto a la simulación de todo este código con MPLAB, haré algunas aclaraciones, primero que nada, decirte que yo lo hice y con buenos resultados, pero oye...!!! no creas que me quedé a esperar que se cumplan los retardos jajaja... nooooo...!!! ni ahí... lo que hice, fue modificar los valores de los registros reg1 y reg2 y ponerlos a 1 para que la rutina termine rápido y continúe con lo que sigue.

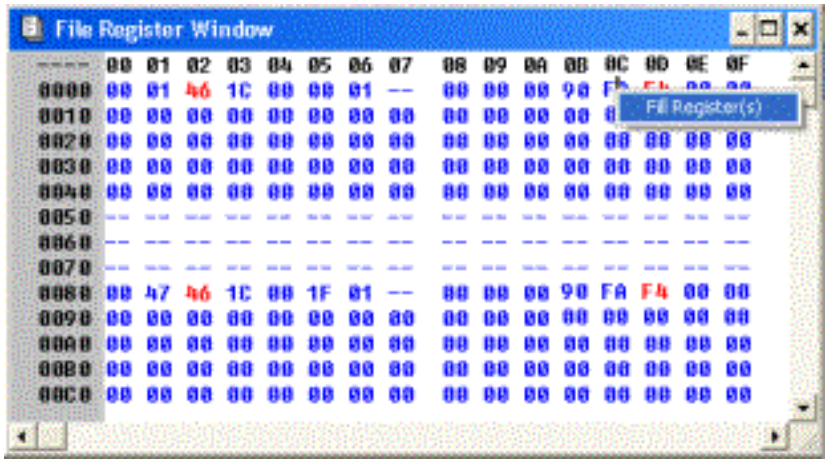
Ésto nunca lo hicimos así que vamos... que te mostraré como hacerlo...

Primero que nada debes tener abierta File Registers Window, que se encuentra en el menú **Window --> File Registers**, y cuando estés corriendo el código y entres a la rutina de retardo lo detienes y verás algo como esto...

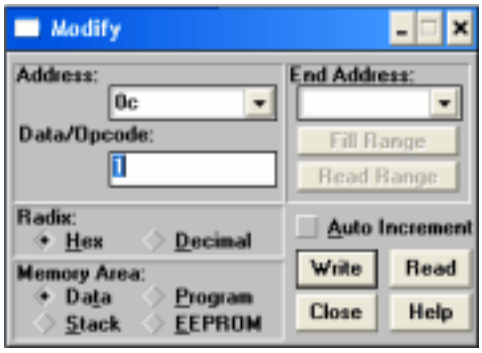


Si recuerdas el código, reg1 es 0C y reg2 es 0D, cuando detuve la ejecución reg2 quedó en **F4**, puesto que se estaba decrementando en la rutina de retardo, y reg1 se mantiene sin cambio y aún retiene el valor que le cargamos inicialmente, es decir **FA**, ahora vamos a modificar los valores de estos registros, así le engañamos a MPLAB para que crea que se decrementó bastante.

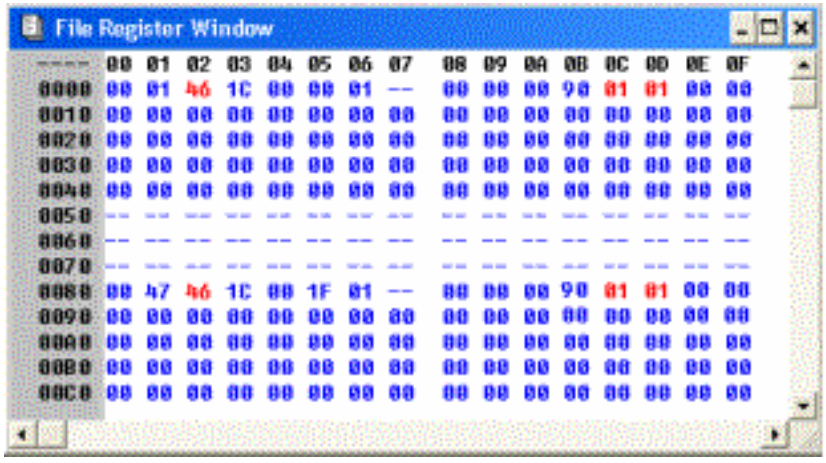
ok, haz un click con el botón derecho del mouse justo en **FA** y luego en **File Register(s)**



y allí te aparecerá un cuadro de diálogo en el que puedes ver justo en **Address** el registro que estás por modificar, en este caso **0c** y en **Data/Opcode** el nuevo valor para ese registro en este caso **1** (este valor es el que estamos ingresando...!!!), tal como se ve en esta imagen...



Luego le das a **Write** (escribir) y habrás modificado el valor del registro **0x0C (reg1)**, para modificar reg2 (0x0D) repites la misma operación y finalmente te quedará así...



Ahora sí, le das a los zapatitos para ver que todo va correctamente y te ahorraste de esperar que llegue el invierno, jeje

Ni que hablar de la temporización de los 50 segundos, lo que hice allí fue verificar que la interrupción por TMR0 se produzca, que Reg2 decremente cuando esto ocurra y finalmente que Reg1 decremente cuando Reg2 se haga cero, observa que he utilizado los mismos registros en ambas temporizaciones, y no te afecta en nada ya que siempre están a cero si no se está ejecutando alguna de ellas.

Bien, el resto lo verifique grabando el programa en el pic y controlando el tiempo

con un cronómetro para saber si realmente se cumplen los 50 segundos.

Por cierto, deberías tener un circuito entrenador si piensas seguir en este tren, ya es hora no crees...???, en la red hay muchos entrenadores de los que puedes disponer, uno más complejo que otro, yo me hice uno que por suerte me sirvió prácticamente para todos los proyectos que hice...

Creo que lo pondré en venta...!!!

:o))

Quizás ese sea el tema para la próxima actualización, escucharé propuestas...

:: PIC - Parte III - Ciclos consumidos por cada Instrucción

Tabla de Instrucciones y Ciclos de Instrucción:

En las siguientes tablas se pueden ver los ciclos de instrucción consumidos por cada una de las instrucciones del PIC16F84...

Ciclos de Instrucc.	
ADDWF	1
ANDWF	1
CLRF	1
CLRW	1
COMF	1
DECF	1
DECFSZ	1 ó 2
INCF	1
INCFSZ	1 ó 2
IORWF	1
MOVF	1
MOVWF	1
NOP	1
RLF	1
RRF	1
SUBWF	1
SWAPF	1
XORWF	1

Ciclos de Instrucc.	
BCF	1
BSF	1
BTFSC	1 ó 2
BTFSS	1 ó 2

Ciclos de Instrucc.	
ADDLW	1
ANDLW	1
CALL	2
CLRWDT	1
GOTO	2
IORLW	1
MOVLW	1
RETFIE	2
RETLW	2
RETURN	2
SLEEP	1
SUBLW	1
XORLW	1

Es fácil de recordarlas ya que sólo los saltos consumen 2 ciclos de instrucción, observa que los saltos condicionales consumen **1 ó 2**, será 1 siempre que no se realice el salto, ahora si el salto se produce serán 2 los ciclos de instrucción.

Por otro lado las instrucciones de llamada salto y retorno consumen **2 ciclos** debido a que son saltos a una determinada posición de memoria, y bueno, el resto sólo consume uno.

:: PIC - Parte III - Registros STATUS, OPTION e INTCON

Registro STATUS:

Contiene el estado aritmético de la ALU, el estado del Reset y los bits para selección de banco.

REGISTRO STATUS							
IRP	**RP1**	**RP0**	**TO**	**PD**	**Z**	**DC**	**C**

Estado de sus Bit's...

BIT's	L ó E	Reset	Descripción
Bit 7-6: **IRP-RP1**	L/E	0	No implementado: '0'
Bit 5: **RP0** Bank Select	L/E	0	1 = Banco 1 0 = Banco 0
Bit 4: **TO** Time-Out	L	1	1 = Recién encendido, tras CLRWDT, o SLEEP. 0 = Ocurrió un time-out en el WDT
Bit 3: **PD** Power Down	L	1	1 = Luego de un Rset, de una instrucción CLRWDT 0 = Tras ejecutar una instrucción SLEEP
Bit 2: **Z** Zero	L/E	x	1 = El resultado de una operación lógica o aritmética es 0. 0 = El resultado es distinto de 0
Bit 1: **DC** Digit Carry	L/E	x	1 = Acarreo en la suma y no en la resta (4º bit) 0 = Acarreo en la resta y no en la suma (4º bit)
Bit 0: **C** Carry	L/E	x	1 = Acarreo en la suma y no en la resta (8º bit) 0 = Acarreo en la resta y no en la suma (8º bit)

Registro OPTION:

Contiene varios bits de control para configurar el divisor de frecuencia o prescaler del TMR0/WDT, la interrupción externa INT, TMR0 y los pull-ups para el PORTB

REGISTRO OPTION							
RBU	**INTDEG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**

Estado de sus Bit's

BIT's	L ó E	Reset	Descripción
Bit 7: **RBPU** Pull-up p' PORTB	L/E	1	1 = Cargas Pull-Up Desconectadas 0 = Cargas Pull-Up Conectadas
Bit 6: **INTEDG** Flanco/Interrup.	L/E	1	1 = RB0/INT será sensible a flanco ascendente 0 = RB0/INT será sensible a flanco descendente
Bit 5: **T0CS** Fte./Reloj p' TMR0	L/E	1	1 = Pulsos por pata T0CKI (contador) 0 = Pulsos igual a reloj interno / 4 (temporizador)
Bit 4: **T0SE** Flanco/T0CKI	L/E	1	1 = Incremento TMR0 en flanco descendente 0 = Incremento en flanco ascendente
Bit 3: **PSA** Divisor/Frecuencia	L/E	1	1 = Divisor asignado al WDT 0 = Divisor asignado al TMR0

La combinación de los BIT's; PS2, PS1 y PS0 (0, 1 y 2) determinan el valor del divisor de frecuencia, el cual se puede ver en la siguiente tabla.

PS2	PS1	PS0	División del TMR0	División del WDT
0	0	0	1/2	1/1
0	0	1	1/4	1/2
0	1	0	1/8	1/4
0	1	1	1/16	1/8
1	0	0	1/32	1/16
1	0	1	1/64	1/32
1	1	0	1/128	1/64
1	1	1	1/256	1/128

Obviamente estos tres bits son de L/E y luego de un reset P2, P1 y P0 se ponen a 1.

Registro INTCON:

Contiene los bits para habilitar cada una de las fuentes de interrupción y las banderas que informan el origen de la interrupción.

REGISTRO INTCON							
GIE	**EEIE**	**T0IE**	**INTE**	**RBIE**	**T0IF**	**INTF**	**RBIF**

Estado de sus Bit's.

BIT's	L ó E	Reset	Descripción
Bit 7: **GIE** Habilitación Gral.	L/E	0	1 = Todas las Interrupciones activadas 0 = Todas las Interrupciones desactivadas
Bit 6: **EEIE** Int. Periféricos	L/E	0	1 = Activada 0 = Desactivada
Bit 5: **T0IE** Int. del TMR0	L/E	0	1 = Activada 0 = Desactivada
Bit 4: **INTE** Int. Externa	L/E	0	1 = Activada 0 = Desactivada
Bit 3: **RBIE** Int. por PORTB	L/E	0	1 = Activada 0 = Desactivada
Bit 2: **T0IF** Bandera del TMR0.	L/E	0	1 = TMR0 desbordado. Borrar por software 0 = No se ha desbordado
Bit 1: **INTF** Bandera - RB0/INT	L/E	0	1 = Ocurrió una interrupción externa 0 = No ha ocurrido interrupción externa
Bit 0: **RBIF** Bandera - RB4:RB7	L/E	x	1 = Al menos un pin cambio de estado 0 = Ningún pin ha cambiado de estado.