

PIC16F84

. Introducción

Con este tutorial solo pretendo introducir al lector en el mundo de la programación de microcontroladores PIC de forma práctica y sencilla. Doy por supuestos unos conocimientos muy básicos sobre electrónica digital. Hablaremos de Instrucciones, registros de memoria RAM, memoria EEPROM (un tipo de ROM), de niveles lógicos "0" o "1" y cosas por el estilo.

El PIC16F84 es un microcontrolador, una especie de "ordenador en miniatura" (con muchas comillas) que podremos programar. En su interior posee un microprocesador, una memoria RAM (volatil) donde guardaremos las variables, una memoria EEPROM (no volatil) donde guardaremos nuestro programa, un Timer o contador que nos facilitará algunas tareas, y alguna cosilla mas...

Algunas características mas representativas son:

- 1Kbyte de memoria EEPROM para nuestro programa
- 68 bytes (de 8 bits) de memoria RAM
- 64 bytes de memoria EEPROM para datos (no vátiles)
- Solo 35 instrucciones
- 13 pines de entrada/salida (un puerto de 8 bits + otro de 5 bits)
- Timer/contador de 8 bits

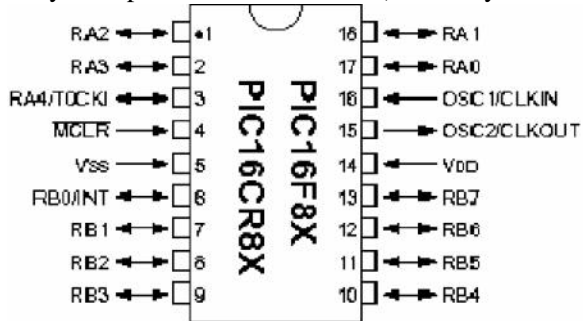
1. Descripción de sus pines

RA0, RA1, RA2, RA3 y RA4: son los pines del puerto A

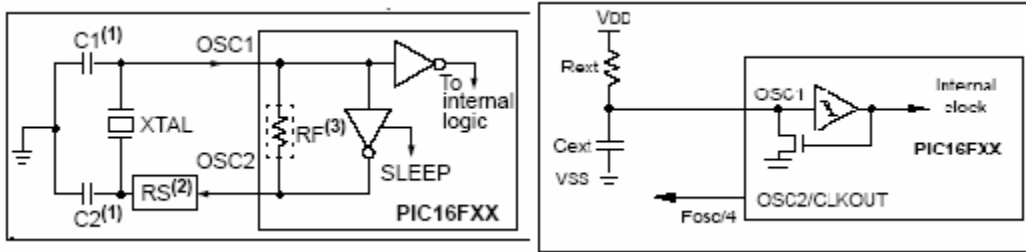
RB0, RB1, RB2, RB3, RB4, RB5, RB6 y RB7: son los pines del puerto B.

MCLR: Pin de reseteo del pic, cuando se pone a "0" el pic se resetea.

Vdd y Vss: pines de alimentación (Vdd 5V y Vss a masa)



OSC1/CLKIN y OSC2/CLKOUT: son para el oscilador. Los tipos de osciladores mas usados son el XT (cristal de cuarzo) y el RC (resistencia y condensador). El modo de conexión es el siguiente:



Oscilador XT

$C1=C2=33\text{pF}$

Crystal = 4MHz

Oscilador RC

$C1$ alrededor de 20pF

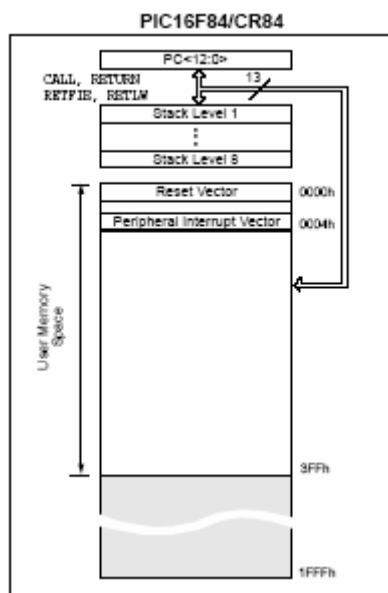
$5K\ \Omega = R1 = 100K\ \Omega$

2. Organización de la memoria

En primer lugar tenemos que distinguir claramente entre tres tipos de memoria:

- Una: la memoria EEPROM donde almacenaremos el programa que haremos, esta memoria solo podrá ser leída por el pic (el pic va leyendo las instrucciones del programa almacenado en esta memoria y las va ejecutando). Al apagar el pic esta memoria no se borra.
- Dos: la memoria RAM en cuyos registros se irán almacenando los valores de las variables que nosotros queramos y cuando nosotros queramos (por programa), al apagar el pic esta memoria se borra.
- Tres: la memoria EEPROM para datos, es un espacio de memoria EEPROM en la que se pueden guardar variables que queremos conservar aunque se apague el pic. No se tratará aquí por ser una memoria mas difícil de emplear.

2.1 La memoria EEPROM o memoria de programa



El espacio marcado como "User memory Space" es el espacio de memoria donde irá nuestro programa, comprende las direcciones de memoria desde la 0000h hasta la 3FFh (3FFh en decimal es 1023, mas la dirección 0000h hacen 1024 direcciones, es decir, 1Kbyte)

"Reset Vector" es la primera dirección a la que se dirige el pic al encenderlo o al resetearlo.

"PC" y los "Stack Level" son empleados por el pic y nosotros no tenemos acceso a ellos.

- INDF (direccionamiento indirecto): Dirección 00h, sirve para ver el dato de la dirección a la que apunta el registro FSR (dir. 04h) que veremos mas adelante
- TMR0 (Timer/contador): Dirección 01h, Aquí se puede ver el valor en tiempo real del Timer/contador. También se puede introducir un valor y alterar así el conteo. Este conteo puede ser interno (cuenta ciclos de reloj) o externo (cuenta impulsos introducidos por RA4).
- PCL (Parte baja del contador de programa): Dirección 02h, Modificando este registro se modifica el contador de programa. este contador de programa es el que señala al

pic en que dirección (de EEPROM) tiene que leer la siguiente instrucción. Esto se utiliza mucho para consultar tablas (ya veremos mas adelante)

- STATUS: Dirección 03h, este es uno de los registros mas importantes y el que mas vas a utilizar. Hay que analizar el funcionamiento de este registro bit a bit:
 - CARRY, Dirección STATUS,0 (bit 0): bit de desbordamiento. Este bit se pone a "1" cuando la operación anterior ha rebasado la capacidad de un byte. Por ejemplo, si sumo dos números y el resultado no cabe en 8 bit el CARRY se pone a "1", Pasa lo mismo cuando resto dos números y el resultado es un número negativo. Se puede usar para saber si un número es mayor que otro (restándolos, si hay acarreo es que el segundo era mayor que el primero). Una vez que este bit se pone a "1" no se baja solo (a "0"), hay que hacerlo por programa si queremos volverlo a utilizar.
 - DC (digit carry), Dirección STATUS,1 (bit 1): lo mismo que el anterior pero esta vez nos avisa si el número no cabe en cuatro bits.
 - Z (zero), Dirección STATUS,2 (bit 2): Se pone a "1" si la operación anterior ha sido cero. Y pasa a "0" si la operación anterior no ha sido cero. Se usa para comprobar la igualdad entre dos números (restándolos, si el resultado es cero ambos números son iguales)
 - PD (Power - Down bit), Dirección STATUS,3 (bit3) se pone a "0" después de ejecutar la instrucción SLEEP*, se pone a "1" después de ejecutar la instrucción CLRWD* o después de un power-up*.
 - TO (Timer Up), Dirección STATUS,4 (bit4) se pone a "0" cuando se acaba el tiempo del WATCHDOG*, Se pone a "1" despues de ejecutar las instrucciones, CLRWD* o SLEEP* o despues de un power-up*.
 - RP0 y RP1 (selección de banco), Dirección STATUS,5 y STATUS,6. Como el PIC16F84 solo tiene dos bancos de memoria el RP1 no se usa para nada, la selección del banco se hace mediante RP0 (STATUS,5), si está a "0" nos encontramos en el banco 0, y si está a "1" nos encontramos en el banco 1.
 - IRP, Dirección STATUS,7, En este PIC no se usa para nada.
- FSR (Puntero), Dirección 04h, se usa para direccionamiento indirecto en combinación con el registro INDF (dir. 00h): se carga la dirección del registro que queremos leer indirectamente en FSR y se lee el contenido de dicho registro en INDF.
- PORTA (Puerto A), Dirección 05h. Con este registro se puede ver o modificar el estado de los pines del puerto A (RA0 - RA4). Si un bit de este registro está a "1" también lo estará el pin correspondiente a ese bit. El que un pin esté a "1" quiere decir que su tensión es de 5V, si está a "0" su tensión es 0V.

Correspondencia:

- RA0 ==> PORTA,0
- RA1 ==> PORTA,1
- RA2 ==> PORTA,2
- RA3 ==> PORTA,3
- RA4 ==> PORTA,4

- PORTB (Puerto B), Dirección 06h igual que PORTA pero con el puerto B

Correspondencia:

- RB0 ==> PORTB,0
- RB1 ==> PORTB,1
- RB2 ==> PORTB,2
- RB3 ==> PORTB,3
- RB4 ==> PORTB,4
- RB5 ==> PORTB,5
- RB6 ==> PORTB,6

- RB7 ==> PORTB,7
- Dirección 07h, No utilizada por este PIC.
- EEDATA, Dirección 08h. En este registro se pone el dato que se quiere grabar en la EEPROM de datos
- EEADR, Dirección 09h. En este registro se pone la dirección de la EEPROM de datos donde queremos almacenar el contenido de EEDATA
- PCLATH, Dirección 0Ah. Modifica la parte alta del contador de programa (PC), el contador de programa se compone de 13 bits, los 8 bits de menor peso se pueden modificar con PCL (dir. 02h) y los 5 bits de mayor peso se pueden modificar con PCLATH
- INTCON (controla las interrupciones), Dirección 0Bh. Se estudia bit a bit:
 - RBIF (Flag de interrupción por cambio de PORTB) Dirección INTCON,0 (bit 0) se pone a "1" cuando alguno de los pines RB4, RB5, RB6, o RB7 cambia su estado. Una vez que está a "1" no pasa a "0" por si mismo: hay que ponerlo a cero por programa.
 - INTF (Flag de interrupción de RB0) Dirección INTCON,1. Si está a "1" es que ha ocurrido una interrupción por RB0, si está a "0" es que dicha interrupción no ha ocurrido. Este bit es una copia de RB0.
 - TOIF (Flag de interrupción por desbordamiento de TMR0) Dirección INTCON,2. Cuando TMR0 se desborda este Flag avisa poniendose a "1". Poner a "0" por programa.
 - RBIE (Habilita la interrupción por cambio de PORTB) Dirección INTCON,3. Si está a "1" las interrupciones por cambio de PORTB son posibles.
 - INTE (Habilita la interrupción por RB0) Dirección INTCON,4. Si lo ponemos a "1" la interrupción por RB0 es posible
 - TOIE (Habilita la interrupción por desbordamiento de TMR0) Dirección INTCON,5. Si este bit esta a "1" la interrupción por desbordamiento de TMR0 es posible.
 - EEIE (Habilita la interrupción por fin de escritura en la EEPROM de datos) Dirección INTCON,6. Cuando este bit está a "1" la interrupción cuando acaba la escritura en la EEPROM de datos es posible.
 - GIE (Habilita las interrupciones globalmente) Dirección INTCON,7. Este bit permite que cualquier interrupción de las anteriores sea posible. Para usar alguna de las interrupciones anteriores hay que habilitarlas globalmente e individualmente.

4. Registros internos (continuación)

Ahora vamos con el banco 1, solo un comentario antes de empezar: ¿recuerdas la tabla de registros internos que veíamos en el punto 2.2? ([ver tabla](#)) ves que los registros del banco 0 y los del banco 1 tienen direcciones distintas, en realidad podemos utilizar las mismas direcciones para referirnos a registros que están en uno u otro banco. El pic las diferenciará sin problemas gracias al bit de selección de banco (RP0). Por ejemplo, la dirección 05h se refiere a PORTA si estamos en el banco 0 y a TRISA si estamos en el banco 2.

Sabiendo esto vamos con los registros del BANCO 1:

- INDF, Dirección 00h, Igual que en el Banco 0
- OPTION, Dirección 01h, (configuración del prescaler, Timer, y alguna cosa mas) Se estudia bit a bit

- PS0, PS1 y PS2 (Bits del 0 al 2) Configuración del preescaler: El preescaler es un divisor de pulsos que está a la entrada del Timer-contador. El preescaler divide el número de pulsos que le entran al timer-contador o al Wachtdog. El factor de división es el siguiente (según los valores de PS2, PS1 y PS0 respectivamente)

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

- PSA, Dirección OPTION,3. Bit de asignación de preescaler: si está a "1" el preescaler se asigna a WDT (Wachtdog), si está a "0" se asigna al TMR0
 - TOSE, Dirección OPTION,4. Bit de selección del tipo de flanco para TMR0. A "1" se incrementa TMR0 por flanco descendente de RA4, a "0" se incrementa TMR0 por flanco ascendente de RA4.
 - TOCS, Dirección OPTION,5. Selecciona la entrada de reloj de TMR0. A "1" la entrada de reloj de TMR0 es por flanco de la patilla RA4, a "0" la entrada de reloj de TMR0 es por ciclo de reloj interno.
 - INTEDG, Dirección OPTION,6. Tipo de flanco para la interrupción por RB0: A "1" la interrupción será por flanco ascendente, a "0" la interrupción será por flanco descendente.
 - RBPU, dirección OPTION,7. Carga Pull-Up en puerto B. A "0" todas las salidas del puerto B tendrán una carga de pull-Up interna.
- PCL, Dirección 02h, igual que en el banco 0
 - STATUS, Dirección 03h, Igual que en el banco 0
 - FSR, Dirección 04h, Igual que en el banco 0
 - TRISA, Dirección 05h, Configura el puerto A como entrada o salida. Si un bit de este registro se pone a "0" el pin correspondiente en el puerto A será una salida, por el contrario, si se pone a "1" el pin correspondiente en el puerto A será una entrada.
 - TRISB, Dirección 06h, Igual que el anterior pero con el puerto B
 - Dirección 07h, No usada en este pic
 - EECON1, Dirección 08h, Controla la lectura y escritura en la EEPROM de datos. Se estudia bit a bit:
 - RD, Dirección EECON1,0 (bit 0) A "1" iniciamos el ciclo de lectura, cuando acaba el ciclo se pone a "0" el solito
 - WR, Dirección EECON1,1 (bit 1) A "1" indicamos que comienza el ciclo de escritura, cuando acaba el ciclo se pone a "0" el solito
 - WREN, Dirección EECON1,2 (bit 2) si lo ponemos a "1" se permite la escritura, a "0" no se permite.
 - WRERR, Dirección EECON1,3 (bit 3) error de escritura, si está a "1" indica que no se ha terminado el ciclo de escritura.
 - EEIF, Dirección EECON1,4 (bit 4) interrupción de ciclo de escritura de la EEPROM, si está a "1" indica que el ciclo de escritura ha terminado, hay que ponerlo a "0" por programa.
 - Bits del 5 al 7 no se utilizan.
 - EECON2, Dirección 09h, Se utiliza para la escritura en la EEPROM de datos como medida de seguridad: para poder guardar algo en la EEPROM hay que cargar el valor 55h en este registro.
 - PCLATH, Dirección 0Ah, Igual que en el banco 0

- INTCON, Dirección 0Bh, Igual que en el banco 1

5. Set de Instrucciones del PIC16F84

Para entender mejor cada instrucción se explica a continuación el significado de algunos parámetros:

- f: Registro al que afecta la instrucción
- W: Acumulador (Working register)
- b: Número de bit (hay instrucciones que afectan a un solo bit)
- k: constante (un número)
- d: selección de destino del resultado de la instrucción, puede ser "0" o "1", si es "0" el resultado se guarda en el acumulador (W) y si es "1" se guarda en el registro f al que afecta la instrucción.

5.1 Instrucciones orientadas a registros:

- ADDWF f,d Suma W y el registro f, el resultado lo guarda según d (si d=0 se guarda en W y si d=1 se guarda en f).
- ANDWF f,d Realiza la operación AND lógica entre W y f, el resultado lo guarda según d.
- CLRF f Borra el registro f (pone todos sus bits a cero).
- CLRW - Borra el acumulador.
- COMF f,d Calcula el complementario del registro f (los bits que están a "0" los pone a "1" y viceversa. Resultado según d.
- DECF f,d Decrementa f en uno (le resta uno). Resultado según d.
- DECFSZ f,d Decrementa f y se salta la siguiente instrucción si el resultado es cero. Resultado según d.
- INCF f,d Incrementa f en uno (le suma uno). Resultado según d.
- INCFSZ f,d Incrementa f y se salta la siguiente instrucción si el resultado es cero (cuando se desborda un registro vuelve al valor 00h). Resultado según d.
- IORWF f,d Realiza la operación lógica OR entre W y f. Resultado según d.
- MOVF f,d Mueve el contenido del registro f a W si d=0 (si d=1 lo vuelve a poner en el mismo registro)
- MOVWF f mueve el valor de W a f. Por ejemplo, si queremos copiar el valor del registro "REG1" al registro "REG2" (ya veremos como ponerles nombres a los registros) escribiremos:

```
MOVF REG1,0 ;mueve el valor de REG1 a W
MOVWF REG2 ;mueve el valor de W a REG2
```

Lo que va después del ; son comentarios

- NOP - No hace nada, solo pierde el tiempo durante un ciclo.
- RLF f,d Rota el registro f hacia la izquierda a través del bit CARRY (todos los bits se mueven un lugar hacia la izquierda, el bit 7 de f pasa al CARRY y el bit CARRY pasa al bit 0 de f). Resultado según d.
- RRF f,d Lo mismo que RLF pero hacia la derecha.
- SUBWF f,d Resta f y W (f - W). Resultado según d.
- SWAPF f,d intercambia los 4 primeros bits de f por los otros cuatro. Resultado según d.
- XORWF f,d Realiza la operación lógica XOR (OR exclusiva) entre W y f. Resultado según d.

5.2 Instrucciones orientadas a bits:

- BCF f,b Pone a "0" el bit b del registro f
- BSF f,d Pone a "1" el bit b del registro f
- BTFSC f,b Se salta la siguiente instrucción si el bit b del registro f es "0"
- BTFSS f,b Se salta la siguiente instrucción si el bit b del registro f es "1"

5.3 Instrucciones orientadas a constantes y de control:

- ADDLW k Le suma el valor k al acumulador (W).
- ANDLW k Operación lógica AND entre W y el valor k (resultado en W).
- CALL k Llamada a subrutina cuyo inicio esta en la dirección k
- CLRWD - Borra el registro Watchdog
- GOTO k Salta a la dirección k de programa.
- IORLW k Operación lógica OR entre W y el valor k (resultado en W)
- MOVLW k carga el acumulador con el valor k. Por ejemplo, si queremos cargar el valor 2Ah en el registro "REG1" escribiremos:

```
MOVLW 2AH ;carga el acumulador con el valor 2Ah
MOVWF REG1 ;mueve el valor de W a "REG1"
```

- RETFIE - Instrucción para volver de la interrupción
- RETLW k carga el valor k en W y vuelve de la interrupción
- RETURN - vuelve de una subrutina.
- SLEEP - El pic pasa a modo de Standby

6. Instrucciones para el ensamblador

Podemos usar para escribir los programas el bloc de notas de Windows, una vez escrito se guarda con extensión .asm y se convierte (ensambla) con un programa ensamblador, el [MPASM](#). El resultado es un archivo con extensión .hex que podemos transferir al PIC16F84. Todo esto se explica mas detalladamente en [Programador del PIC16F84](#).

Existen una serie de instrucciones que son para el ensamblador y nos hacen la tarea de programación mas sencilla y mas legible.

- EQU: Un ejemplo de esto son las etiquetas, podemos poner un nombre a un registro de memoria, esto se hace mediante la instrucción EQU. Por ejemplo:

```
VARIABLE1 EQU 0CH
```

A partir de ahora en lugar de escribir 0CH podemos escribir VARIABLE1. Con EQU también podemos poner nombre a constantes de la misma forma.

- #DEFINE: Otra instrucción para el ensamblador que usaremos será la instrucción #DEFINE. Es parecido a EQU, solo que aquí no ponemos etiquetas a un registro, podemos ponerla a una instrucción entera, Por ejemplo:

```
#DEFINE BANCO1 BSF STATUS,5
#DEFINE BANCO0 BCF STATUS,5
```

A partir de ahora, cuando escribamos BANCO1 se pondrá a "1" el bit de selección de banco y pasaremos al banco 1, al escribir BANCO0 pasaremos al banco 0

- ORG: Indica al ensamblador la dirección (de memoria de programa) donde se guardará la instrucción que vaya a continuación. Por ejemplo:


```
ORG 00H
CLRF VARIABLE1
```

La instrucción CLRF está en la dirección de memoria de programa 00H (será la primera instrucción en ser ejecutada por el pic)

- END: Se escribe al final del programa para indicar que ya ha acabado. (es obligatorio, si no da error).
- Etiquetas a direcciones de Programa: muy útiles para usar con instrucciones CALL (Llamada a subrutina) o GOTO (Salto). Por ejemplo:

```
.....
[Hay programa anterior]
.....
BTFSC VARIABLE1,0 ;Si el bit 0 de VARIABLE1 es "0" se salta
la siguiente instrucción
GOTO ESUNO ;Salta a ESUNO solo si el bit 0 de VARIABLE1 es
"1" BSF VARIABLE1,0 Si el bit 0 de VARIABLE1 es 0 se ejecuta
esta instrucción y el programa sigue por aquí
.....
[Continúa el programa]
.....
ESUNO ;Etiqueta a una dirección de programa
BCF VARIABLE1,0 ;Si el bit 0 de VARIABLE1 es "1" se ejecuta
esta otra instrucción y el programa sigue por aquí
.....
[Cuntinúa el programa]
.....
```

Un poco de orden:

Es importante llevar un poco de orden a la hora de escribir el programa, nos ayudará mucho:

- Al principio van los EQU y los #DEFINE, después comenzamos con el programa.
- El programa se escribe en cuatro columnas separadas por tabuladores:
 - En la primera columna se ponen las etiquetas a direcciones de programa
 - En la segunda columna se ponen las instrucciones (BSF, CLRF, BTFSC... etc.)
 - En la tercera columna se ponen Los registros o parámetros a los que afecta la instrucción.
 - En la cuarta columna se ponen los comentarios que creas pertinentes (cuantos mas mejor) seguidos de un punto y coma.

Ejemplo de programa bien ordenado (se podría mejorar ;):

```
;*****
;*      http://www.electron.es.vg      *
;*****
;* EJEMPLO 1: PROGRAMA BIEN ORDENADO*
;*****
;* El siguiente programa configura  *
;* RA1 como entrada y RA0 como     *
;* salida y hace que la salida (RA0)*
;* sea la inversa de la entrada      *
;* (RA1)                             *
;*****
```

```
;(Conviene poner título y una
;pequeña explicación de lo que
;hace el programa)
```

```
;(Primero los ECU y los #DEFINE)
```

```
STATUS EQU    03H
TRISA   EQU    05H
PORTA   EQU    05H
```

```
#DEFINE BANCO0 BCF    STATUS,5
#DEFINE BANCO1 BSF    STATUS,5
```

```
;(Después empezamos con el programa)
```

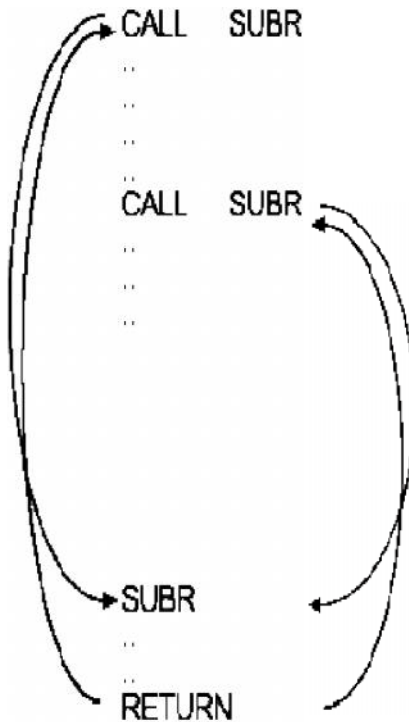
```
      ORG      00H      ;Empezamos siempre a escribir en esta dirección
      BANCO1      ;Pasamos al banco 1 para hacer algunas
                    ;configuraciones
      BCF      TRISA,0 ;Configuramos RA0 como salida
      BSF      TRISA,1 ;Configuramos RA1 como entrada
      BANCO0      ;Volvemos al banco 0

INICIO BTFSC    PORTA,1 ;Comprueba la entrada (RA1), si es "0" se salta la
                    ;siguiente instrucción
      GOTO     ESUNO    ;si la entrada (RA1) es "1" va a ESUNO

      BSF      PORTA,0 ;Pone a "1" la salida RA0. Ejecuta esta
instrucción
                    ;porque la entrada RA1 era "0"
      GOTO     INICIO    ;Vuelve otra vez a comprobar el estado de la
                    ;entrada RA1

ESUNO  BCF      PORTA,0 ;Pone a "0" la salida RA0. Ejecuta esta
instrucción
                    ;porque la entrada RA1 era "1"
      GOTO     INICIO    ;Vuelve otra vez a comprobar el estado de la
                    ;entrada RA1

      END      ;Indica final del programa
.....
```



7. Subrutinas

Una subrutina o subprograma es una parte de programa que hace algo concreto y se repite a menudo, para ahorrar memoria y esfuerzo y para hacer el programa mas comprensible se agrupa en forma de subrutina. Una subrutina se debe ejecutar siempre llamándola con la instrucción CALL y al final de dicha subrutina debe haber siempre un RETURN. El esquema de la derecha muestra como funcionan las subrutinas:

Durante el programa principal se llama varias veces a la subrutina SUBR (el nombre es lo de menos) con la instrucción CALL. Cuando el pic ejecuta una instrucción CALL se guarda en memoria la dirección de código de programa a la que tiene que retornar de tal forma que cuando se encuentra con la instrucción RETURN vuelve al programa principal donde lo dejó.

Una subrutina no solo puede ser llamada desde el programa principal, también puede hacerse desde otra subrutina (una subrutina que llama a otra subrutina) o desde una interrupción (enseguida las veremos).

El siguiente ejemplo muestra un programa que utiliza una subrutina de retardo a la que llama DELAY. Esta subrutina de retardo se hace decrementando el registro CUENTA2 desde FFh hasta 00h 16 veces (las veces que se decrementa CUENTA2 son contadas hacia atrás por CUENTA1) De esta forma se consigue perder tiempo (el tiempo perdido con esta subrutina depende de la frecuencia a la que opere el pic)

```

;*****
;*      http://www.electron.es.vg      *
;*****
;*      EJEMPLO 2 USO DE SUBROUTINAS    *
;*****
;* Este programa configura RB0 como salida*
;* y genera una intermitencia en dicha  *
;* salida                               *
;*****

```

```

STATUS EQU    03H
TRISB  EQU    06H
PORTB  EQU    06H

```

```

CUENTA1 EQU    0CH      ;Las variables que usemos siempre a
                        ;partir de la dirección 0Ch
CUENTA2 EQU    0DH

```

```

F      EQU    1
w      EQU    0

```

```

ORG    00H

```

```

        BSF      STATUS,5          ;banco 1
        BCF      TRISB,0          ;RB0 como salida
        BCF      STATUS,5          ;banco 0

INICIO  BSF      TRISB,0          ;Pone a "1" RB0 (enciende)
        CALL     DELAY           ;Llama a la subrutina de retardo
        BCF      TRISB,0          ;Cuando vuelve del retardo pone
                                   ;a "0" RB0 (apaga)
        CALL     DELAY           ;llama a la subrutina de retardo
        GOTO     INICIO          ;cuando vuelve del retardo
                                   ;ejecuta el GOTO

;=====
;=  DELAY:      Subrutina de retardo      =
;=              Modifica los siguientes registros: =
;=              CUENTA1                  =
;=              CUENTA2                  =
;=              ACUMULADOR               =
;=              STATUS                   =

;(Conviene hacerse un pequeño resumen de lo que
;hace cada subrutina, puede sernos muy útil para
;usarla en otros programas)

DELAY   MOVLW    010H              ;Carga el acumulador con el valor
                                   ;10H (16 en decimal)
        MOVWF    CUENTA1          ;Mueve el contenido del acumulador
                                   ;a CUENTA1
ACA1    MOVLW    0FFH              ;Carga el acumulador con el valor FFH
        MOVWF    CUENTA2          ;Mueve el contenido del acumulador
                                   ;a CUENTA2
ACA     DECFSZ   CUENTA2,F          ;Decrementa CUENTA2, guarda el
resultado
                                   ;en f, y si es cero se salta la siguiente
                                   ;instrucción
        GOTO     ACA              ;vuelve a decrementar mientras
                                   ;CUENTA2 no sea cero
        DECFSZ   CUENTA1,F          ;Se decrementa CUENTA1 cada vez que
                                   ;CUENTA2 llega a cero
        GOTO     ACA1             ;mientras CUENTA1 no llegue a cero recarga
                                   ;CUENTA2 y repite el proceso
        RETURN                    ;retorna al programa principal

;=
;=  FIN DE LA SUBROUTINA DELAY      =
;=====

        END                      ;Fin del programa
*****

```

8. Interrupciones

Cuando se produce una interrupción el pic deja automáticamente lo que esté haciendo, va directo a la dirección 04h de programa y ejecuta lo que encuentre a partir de allí hasta encontrarse con la instrucción

RETFIE que le hará abandonar la interrupción y volver al lugar donde se encontraba antes de producirse dicha interrupción.

Para que se pueda producir una interrupción hay que habilitar las interrupciones globalmente y la interrupción en concreto que queremos utilizar (con el registro [INTCON](#)). Este pic tiene 4 tipos de posibles interrupciones:

1. Por cambio en los bits RB4-RB7
2. Por el estado de RB0
3. Por desbordamiento del Timer-contador
4. Por fin de ciclo de escritura de la EEPROM de datos

Mientras se está ejecutando una interrupción no se puede producir otra interrupción, el pic no lo permite.

Una cosa importante a tener en cuenta al usar interrupciones es que cuando estas se producen podríamos estar trabajando con registros que pueden ser modificados en la propia interrupción, como el acumulador o el [STATUS](#). Para que la interrupción no eche a perder el buen funcionamiento del programa principal conviene guardar los valores de estos registros en otras variables que no vayamos a modificar. Antes de salir de la interrupción volvemos a restaurar los valores guardados y todo solucionado.

El siguiente ejemplo muestra un programa que usa la interrupción por cambio en el puerto B (En pines RB4 a RB7)

```
;*****  
;*      http://www.electron.es.vg      *  
;*****  
;*      EJEMPLO 3: USO DE INTERRUPCIONES      *  
;*****  
;* Este programa invierte el estado del pin*  
;* RA0 cada vez que se modifica el estado *  
;* de alguno de los pines RB4, RB5, RB6 o *  
;* RB7. Para ello habilita la interrupción *  
;* por cambio de RB4-RB7                  *  
;*****
```

```
STATUS EQU    03H  
TRISA  EQU    05H  
PORTA  EQU    05H  
TRISB  EQU    06H  
PORTB  EQU    06H  
INTCON EQU    0BH
```

```
ACUM    EQU    0CH  
STAT    EQU    0DH
```

```
F      EQU    1  
w      EQU    0
```

```
#DEFINE BANCO0 BCF    STATUS, 5  
#DEFINE BANCO1 BSF    STATUS, 5
```

```
ORG      00H  
GOTO     INICIO    ;ponemos este GOTO al principio para poder poner
```

```

;el subprograma de las interrupciones a partir de
;la dirección 04h

;Comienza la interrupción:
;=====

ORG      04H      ;El pic salta a esta dirección cuando se produce
                  ;una interrupción
BCF       INTCON,0 ;bit que indica un cambio en RB4-RB7, recuerda
que
                  ;hay que ponerlo a "0" por programa, este es el
                  ;momento

                  ;comenzamos guardando el contenido del acumulador
                  ;y del STATUS para restaurarlos antes de salir de
                  ;la interrupción (es recomendable hacer esto
                  ;siempre que se usen interrupciones)

MOVWF     ACUM      ;Copia el acumulador al registro ACUM
MOV       STATUS,W  ;Guarda STATUS en el acumulador
BANCO0    ;Por si acaso, nunca se sabe en que parte de
           ;programa principal salta la interrupción
MOVWF     STAT      ;Copia el acumulador al registro STAT

           ;Invertimos el estado de RA0:
           ;=====

BTFSC     PORTA,0   ;si RA0 es "0" salta la siguiente instrucción
GOTO      ESUNO     ;vete a ESUNO
BSF        PORTA,0   ;Pon a "1" RA0 (porque era "0")
GOTO      HECHO     ;ya está invertido RA0, vete a HECHO

ESUNO     BCF        PORTA,0 ;Pon a "0" RA0 (Porque era "1")

           ;Ya se ha invertido el estado de RA0
           ;=====

           ;ahora hay que restaurar los valores del STATUS y
           ;del acumulador antes de salir de la
interrupción:

HECHO     MOVF       STAT,W  ;Guarda el contenido de STAT en el acumulador
          MOVWF      STATUS  ;Restaura el STATUS
          SWAPF      ACUM,F   ;Da la vuelta al registro ACUM
          SWAPF      ACUM,W   ;Vuelve a dar la vuelta al registro ACUM y
restaura
          ;el acumulador (Con la instrucción SWAPF para no
          ;alterar el STATUS, la instrucción MOVF altera el
          ;bit 2 del STATUS)
          RETFIE          ;fin de la interrupción

          ;Fin de la interrupción
          ;=====

INICIO    BANCO1     ;Pasamos al banco 1
          MOVLW      0FFH    ;Todos los bits del acumulador a "1"
          MOVWF      TRISB   ;configuramos todo el puerto B como entradas

```

```

BCF      TRISA,0 ;RA0 como salida
BANCO0   ;Volvemos al banco 0

;Configuración de las interrupciones:
;=====

BSF      INTCON,7 ;Habilita las interrupciones globalmente
BSF      INTCON,3 ;Habilita la interrupción por cambio de puerto B

;=====
;ya están configuradas las interrupciones, a
;partir de ahora cuando haya un cambio en RB4-RB7
;saltará la interrupción (a la dirección 04h de
;programa)

NADA     GOTO   NADA ;En este ejemplo no se hace nada en el programa
;principal, simplemente se espera a que salte la
;interrupción. La verdadera utilidad de las
;interrupciones es que se pueden hacer "cosas"
;mientras sin preocuparse de la interrupción

END      ;FIN DE PROGRAMA

```

9. Timer - Contador TMR0

El registro [TMR0](#) puede contar ciclos de instrucción interna o pulsos de entrada por RA4 según el valor del bit 5 del registro [OPTION](#) ([TOCS](#)). Si este bit está a "1" [TMR0](#) cuenta pulsos por RA4 y se le llama Contador; si el bit está a "0" cuenta ciclos de instrucción interna y se le llama Timer.

Cada ciclo de instrucción dura 4 veces el ciclo del reloj del pic (para un reloj de 4MHz ==> Ciclo reloj=0,25 µSeg ==> Ciclo instrucción = 4 X 0,25 = 1µSeg)

Cuando lo usamos como contador (Por RA4) podemos determinar si el incremento se hará por flanco ascendente o descendente con el bit 4 del registro [OPTION](#) ([TOSE](#))

Podemos leer o escribir el registro [TMR0](#) en cualquier momento. Cuando escribamos en él deja de contar durante dos ciclos, cuando lo leamos no pasa nada.

Podemos asignar el prescaler al [TMR0](#), si hacemos esto podemos elegir el factor en el que se verá dividido el conteo mediante los [bits del 0 al 2](#) del registro [OPTION](#) según la [tabla del factor de división](#) . Por ejemplo, si elegimos un factor de división de 1/2 tienen que entrar 2 pulsos para que [TMR0](#) se incremente en uno, si está a 1/4 tienen que entrar 4... etc.

También podemos utilizar la interrupción que se produce cuando se desborda el [TMR0](#), es decir, cuando pasa de FFh a 00h. (se configura desde el registro [INTCON](#))

El siguiente ejemplo usa la interrupción por desbordamiento de [TMR0](#) para obtener una onda cuadrada a la salida RB0:

```

;*****
;*      http://www.electron.es.vg      *
;*****
;*      EJEMPLO 4: USO DEL TMR0        *

```

```

;*****
;* Este programa crea una señal cuadrada a *
;* la salida RB0, para ello utiliza el TMR0*
;* y la interrupción por desbordamiento del*
;* mismo. Se le asignará el prescaler con *
;* un factor de división de 1/2. De esta *
;* forma las interrupciones saltarán a *
;* intervalos fijos de tiempo. Invirtiendo *
;* el estado de RB0 durante las *
;* interrupciones se conseguirá una onda *
;* cuadrada perfecta *
;*****

```

```

OPTIONR EQU    01H    ;Registro para configuración del TMR0
STATUS  EQU    03H
TRISB   EQU    06H
PORTB   EQU    06H
INTCON  EQU    0BH

```

```

ACUM     EQU    0CH
STAT     EQU    0DH

```

```

F        EQU    1
W        EQU    0

```

```

#DEFINE BANCO0 BCF    STATUS,5
#DEFINE BANCO1 BSF    STATUS,5

```

```

ORG      00H
GOTO     INICIO    ;ponemos este GOTO al principio para poder poner
                   ;el subprograma de las interrupciones a partir de
                   ;la dirección 04h

```

```

                   ;Comienza la interrupción:
                   ;=====

```

```

ORG      04H      ;El pic salta a esta dirección cuando se produce
                   ;una interrupción
BCF      INTCON,2  ;bit que indica desbordamiento de TMR0, recuerda
                   ;que hay que ponerlo a "0" por programa, este es
                   ;el momento

```

```

                   ;comenzamos guardando el contenido del acumulador
                   ;y del STATUS para restaurarlos antes de salir de
                   ;la interrupción (es recomendable hacer esto
                   ;siempre que se usen interrupciones)

```

```

MOVWF    ACUM      ;Copia el acumulador al registro ACUM
MOVF     STATUS,W   ;Guarda STATUS en el acumulador
BANCO0   ;Por si acaso, nunca se sabe en que parte de
           ;programa principal salta la interrupción
MOVWF    STAT      ;Copia el acumulador al registro STAT

```

```

                   ;Para generar una onda cuadrada Invertimos el

```



```

                                ;estado de RB0 cada vez que salta una
interrupción

;=====

        BTFSC    PORTB,0  ;si RB0 es "0" salta la siguiente instrucción
        GOTO     ESUNO    ;vete a ESUNO
        BSF      PORTB,0  ;Pon a "1" RB0 (porque era "0")
        GOTO     HECHO    ;ya está invertido RB0, vete a HECHO

ESUNO    BCF      PORTB,0  ;Pon a "0" RA0 (Porque era "1")

                                ;Ya se ha invertido el estado de RB0
                                ;=====

                                ;ahora hay que restaurar los valores del STATUS y
                                ;del acumulador antes de salir de la
interrupción:

HECHO    MOVF     STAT,W   ;Guarda el contenido de STAT en el acumulador
        MOVWF    STATUS   ;Restaura el STATUS
        SWAPF    ACUM,F   ;Da la vuelta al registro ACUM
        SWAPF    ACUM,W   ;Vuelve a dar la vuelta al registro ACUM y
restaura

                                ;el acumulador (Con la instruccion SWAPF para no
                                ;alterar el STATUS, la instrucción MOVF altera el
                                ;bit 2 del STATUS)
        RETFIE    ;fin de la interrupción

                                ;Fin de la interrupción
                                ;=====

INICIO   BANCO1                ;Pasamos al banco 1
        BCF      TRISB,0  ;RB0 como salida
        BCF      OPTIONR,3 ;Asignamos el preescaler a TMR0
        BCF      OPTIONR,0 ;\
        BCF      OPTIONR,1 ; }Prescaler a 1/2
        BCF      OPTIONR,2 ;/
        BCF      OPTIONR,5 ;Entrada de TMR0 por ciclo de
                                ;instrucción interna (se pone a contar)
        BANCO0                ;Volvemos al banco 0

                                ;Configuración de las interrupciones:
                                ;=====

        BSF      INTCON,7 ;Habilita las interrupciones globalmente
        BSF      INTCON,5 ;Habilita la interrupción por desbordamiento de
TMR0

                                ;=====
                                ;ya están configuradas las interrupciones, a
                                ;partir de ahora cuando cuando se desborde TMR0
                                ;saltará la interrupción (a la dirección 04h de
                                ;programa)

```

```

NADA    GOTO    NADA    ;En este ejemplo no se hace nada en el programa
                                ;principal, simplemente se espera a que salte la
                                ;interrupción. La verdadera utilidad de las
                                ;interrupciones es que se pueden hacer "cosas"
                                ;mientras sin preocuparse de las interrupciones

                                END                                ;FIN DE PROGRAMA

```

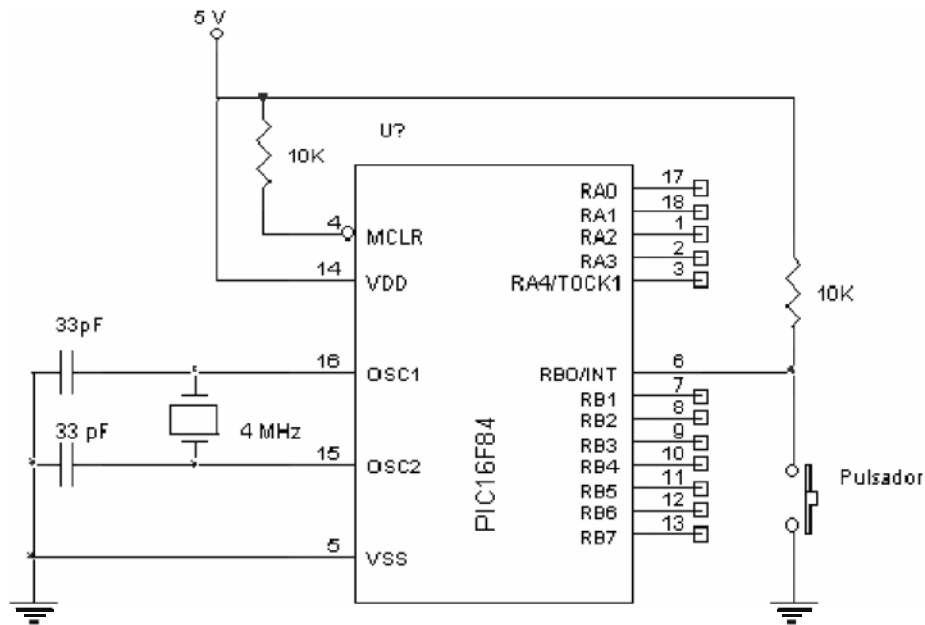
10. Pulsadores e interruptores (rebotes)

Es muy normal usar pulsadores o interruptores en alguna de las entradas del pic para controlarlo. Estos pulsadores no hacen una conexión perfecta e instantánea como podemos pensar: un pulsador se compone de dos partes de metal que entran en contacto (choca una con la otra) al accionarlo. Este choque genera unos pequeñísimos rebotes que suceden tan rápido que son imperceptibles para nosotros, Sin embargo, no lo son para el PIC que trabaja a esas velocidades. Esto es un problema muy común que puede volvernos locos si no lo conocemos pero que resolverás sin problemas cuando acabes de leer esta página. El siguiente diagrama muestra lo que pasaría al accionar un pulsador:



La solución es sencilla, basta con añadir un pequeño retardo en nuestro programa desde que se detecta el primer pulso hasta que se vuelve a leer la entrada del pulsador. Dicho retardo debe ser suficientemente largo para asegurar que cuando finalice ya se hayan extinguido los rebotes, pero también suficientemente corto para que sea imperceptible para nosotros.

En el siguiente ejemplo se cuentan los pulsos introducidos al PIC por RB0. El esquema es el siguiente:



Fíjate que cuando se acciona el pulsador la entrada RB0 se pone a "0". Para evitar contar los rebotes se llama a una subrutina de retardo llamada REBOTE, esta subrutina funciona bien para osciladores de 4MHz.

```

;*****
;*      http://www.electron.es.vg      *
;*****
;*  EJEMPLO 5: EVITANDO REBOTES      *
;*****
;* El siguiente programa cuenta las *
;* veces que se acciona un pulsador *
;* conectado a RB0 y Previene los   *
;* rebotes del mismo mediante la    *
;* subrutina de retardo REBOTE      *
;*****

```

```

STATUS EQU    03H
TRISB   EQU    05H
PORTB   EQU    05H

```

```

CUENTA EQU    0CH      ;Registro donde contaremos
RETARDO EQU    0DH      ;Registro para el retardo

```

```

F      EQU    1
w      EQU    0

```

```

#DEFINE BANCO0 BCF    STATUS,5
#DEFINE BANCO1 BSF    STATUS,5

```

```

ORG      00H
BANCO1      ;Pasamos al banco 1

```

```

        BSF      TRISB,0 ;Configuramos RB0 como Entrada
        BANCO0   ;Volvemos al banco 0

        CLRF     CUENTA ;Pone a cero la cuenta

INICIO  BTFSS    PORTB,0 ;si RB0 es "1" salta la siguiente instrucción
        CALL     SUMA1  ;Llama a la subrutina SUMA1 (porque RB0 es "0" y,
                        ;por lo tanto, se ha accionado el pulsador)
        GOTO     INICIO ;vuelve a INICIO

;=====
;      SUMA1: Subrutina que suma uno al registro
;            CUENTA y espera a que se suelte el
;            pulsador conectado a RB0. Tiene en
;            cuenta los rebotes

SUMA1   INCF     CUENTA,F ;Incrementa el registro CUENTA
        CALL     REBOTE  ;Llama a la subrutina que previene los rebotes
ESPERA  BTFSS    PORTB,0 ;\
        GOTO     ESPERA  ; }Espera a que se suelte el pulsador para
retornar
        RETURN      ;/

;      Fin de SUMA1
;=====

;=====
;      REBOTE: Subrutina que hace una pequeña
;            temporización para prevenir
;            los rebotes

REBOTE  MOVLW    0FFH     ;Carga el acumulador con 0FFh
        MOVWF    RETARDO ;Mueve el contenido del acumulador a RETARDO
REBO    DECFSZ   RETARDO,F ;\
        GOTO     REBO    ; }No retorna hasta que RETARDO llega a cero
        RETURN      ;/

;      Fin de REBOTE
;=====

        END          ;Indica final del programa

```

11. Tablas con la instrucción RETLW

Es muy común utilizar la instrucción RETLW para la creación de tablas de valores. Esta instrucción devuelve un valor en el acumulador al retornar de una subrutina, la usaremos conjuntamente con el registro [PCL](#) (echale un vistazo si no lo recuerdas). La creación de la tabla se hará de la siguiente forma:

TABLA	MOVWF	PCL
VALORES	RETLW	VALOR0
	RETLW	VALOR1

RETLW VALOR2

RETLW VALOR3

....

Donde VALOR0, VALOR1, VALOR2... etc. son los valores que queremos almacenar en la tabla.

La estrategia a seguir para consultar algún valor de la tabla es cargar en el acumulador la dirección de memoria de programa donde se encuentra el valor que quieres leer y después llamar a la subrutina TABLA (con un CALL). Y te estarás preguntando como se en que dirección esta cada valor, pues bien, el VALOR0 está en la dirección VALORES (es la etiqueta que hemos puesto a esa dirección), el VALOR1 está en la dirección VALORES+1, VALOR2 está en la dirección VALORES+2... etc.

Y como no hay como un ejemplo para ver las cosas mas claras ahí va uno: El siguiente ejemplo cuenta las veces que se acciona un pulsador conectado en RA0 y muestra el valor del conteo en un display de 7 segmentos de ánodo común conectado al puerto B. La tabla contiene los valores necesarios para iluminar el display con cada número del 0 al 9. A continuación se muestran el esquema y el programa:

