

3.- Descripción Detallada del PIC16F877

3.1.- Los Puertos Paralelos de Entrada / Salida

Los integrados PIC16F874 y PIC16F877 poseen 5 puertos de entrada / salida denominados PORTA, PORTB,...,PORTE, mientras que el PIC16F873 y PIC16F876. Estos puertos son totalmente programables, es decir, sus líneas pueden ser configuradas para trabajar como entradas o como salidas a selección del programador.

3.1.1.- El Puerto A (PORTA).

El puerto A posee 6 líneas bidireccionales. Los 3 registros asociados a este puerto son: **Registro PORTA (05H)**.- Registro de estado del Puerto A. Cada uno de los 6 bits menos significativos (RA5,...,RA0) de este registro están asociados a la línea física correspondiente del puerto. Al hacer una **lectura** este registro se lee el estado de **todas** las patitas del puerto. Todas las **escrituras** al registro son operaciones del tipo “lee-modifica-escribe”, es decir, toda escritura al puerto implica que el estado de las patitas es leído, luego es modificado y posteriormente se escribe al latch de datos del puerto.

POR, BOR otros Reset	-	-	0	x	0	0	0	0
	-	-	0	u	0	0	0	0
05h	-	-	RA5	RA4	RA3	RA2	RA1	RA0
Bit	7	6	5	4	3	2	1	0

Registro PORTA (05h)

Registro TRISA (85H).- Cada bit de este registro configura la dirección en que fluye la información de la patita correspondiente del puerto A, así, para k=0,1,...,5:}

Bit k de TRISA = 1 configura la patita RAK del puerto A como **Entrada**

Bit k de TRISA = 0 configura la patita RAK del puerto A como **Salida**

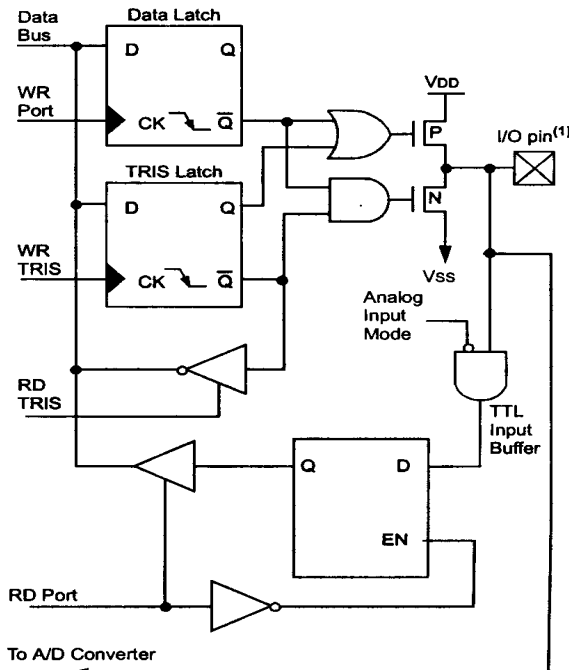
POR, BOR otros Reset	-	-	1	1	1	1	1	1
	-	-	1	1	1	1	1	1
85h	-	-	Registro de dirección de datos del puerto A					
Bit	7	6	5	4	3	2	1	0

Registro TRISA (85h)

Todas las patitas del puerto A poseen diodos de protección conectados a Vdd (contra altos voltajes de entrada) y a Vss (contra voltajes negativos) además, manejan niveles de entrada tipo TTL y como salidas se comportan como drivers tipo CMOS. Excepto la patita RA4, la cual como entrada posee un Disparador Schmitt y como salida es de Drenaje abierto, además RA4 sólo posee un diodo de protección conectado a Vss.

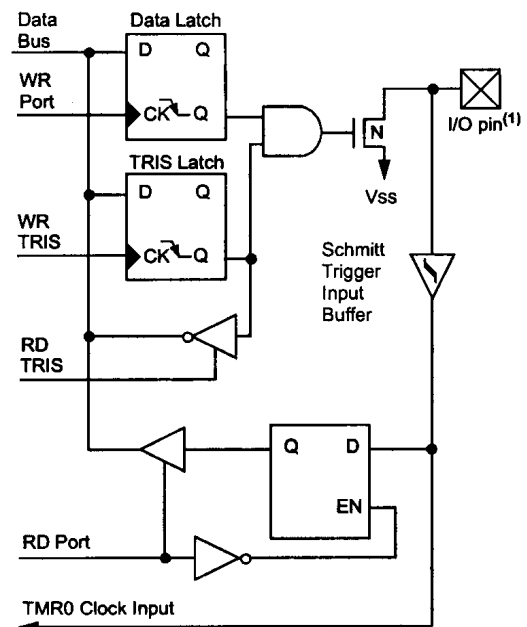
El Registro ADCON1 (9FH).- Las patitas RA0, RA1, RA2, RA3 y RA5 están multiplexadas con las entradas analógicas AN0,...,AN4, de manera que antes de utilizarlas debemos configurar si serán usadas como entradas analógicas o como entradas / salidas digitales. Para seleccionar la segunda opción (entradas / salidas digitales) se debe colocar en la mitad menos significativa de este registro un 0110₂ (es decir, un 06h).

En las siguientes dos figuras se muestra el detalle de implementación interna de las patitas del puerto A, mostrando la diferencia entre las patitas RA4 y las demás patitas del puerto A



Note 1: I/O pins have protection diodes to VDD and VSS.

Patitas RA0,RA1,RA2,RA3 y RA5



Note 1: I/O pin has protection diodes to VSS only.

Patita RA4

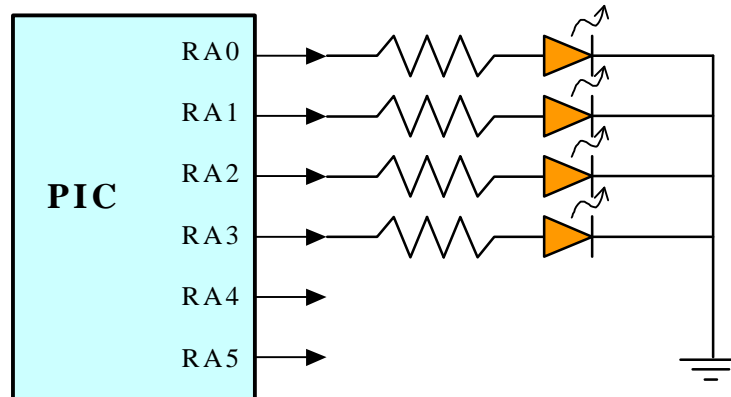
Ejemplo 1: Salidas digitales.- En este ejemplo se configuran las patitas RA0,...,RA3 del puerto A para manejar el encendido y apagado de 4 diodos luminosos conectados a ellos.

```

Include "p16f877.inc"
org 0x0000          ;Inicia en el vector de reset
;Inicialización del puerto A:
CLRF STATUS         ;Selecciona Banco 0
CLRF PORTA          ;Inicializa latches de datos de PORTA
BSF STATUS,RP0      ;Selecciona Banco 1
MOVLW 0x06          ;Configura todas las patitas de A
MOVWF ADCON1        ;como digitales
MOVLW 0x00          ;configura todas las patitas de A
MOVWF TRISA         ;como salidas digitales
;Una vez inicializado el puerto, se procede a controlar los LEDs
BCF STATUS,RP0      ;regresa al banco 0
ciclo CLRF PORTA     ;Apaga todos los LEDs
BSF PORTA,0         ;enciende el LED RA0
BSF PORTA,1         ;enciende el LED RA1
BSF PORTA,2         ;enciende el LED RA2
BSF PORTA,3         ;enciende el LED RA3
GOTO ciclo
end
    
```

Hardware necesario.- Como prácticamente en todos los programas para PIC, si no conectamos el hardware adecuado no podremos ver ningún efecto al ejecutar el programa. En este caso, el hardware es muy simple, además de conectar las señales

de reloj de acuerdo a alguna de las opciones descritas en el capítulo anterior y consiste en 4 LEDs conectados a las patitas RA0,...,RA3 con las respectivas resistencias limitadoras de corriente, como se muestra en la siguiente figura:

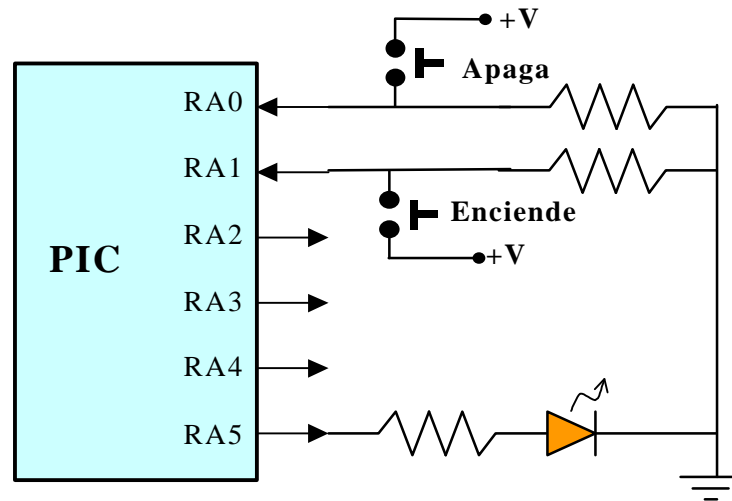


- **Observación.** Al ejecutar el programa en un PIC real con un cristal de 4 Mhz (no simulado) a simple vista no podemos apreciar el parpadeo de los LEDs, ya que éstos se encienden durante tres a seis μ seg y se apagan durante 1 a 3 μ seg.

Ejemplo 2: Entradas digitales.- En este segundo ejemplo se configura la patita RA5 del puerto A como salida conectada a un LED, el cual se controla de acuerdo al estado de las patitas RA0 y RA1 configuradas como entradas

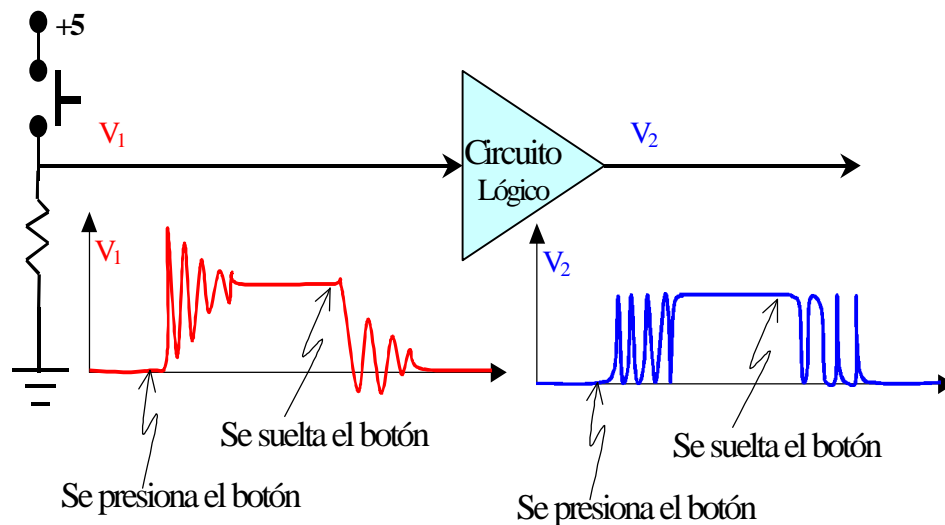
```
;*Este programa Enciende un LED conectado a RA5 cuando se presiona un botón
;*conectado a RA1 y lo apaga cuando se presiona un botón conectado a RA0
;*****
    include "p16f877.inc"
    org 0x0000          ;Inicia en el vector de reset
;Inicialización del puerto A:
    CLRF STATUS         ;Selecciona Banco 0
    CLRF PORTA          ;Inicializa latches de datos de PORTA
    BSF STATUS,RP0      ;Selecciona Banco 1
    MOVLW 0x06          ;Configura todas las patitas de A
    MOVWF ADCON1         ;como digitales
    MOVLW 0x1F          ;configura todas las patitas de A como entradas
    MOVWF TRISA         ;y RA5 como salida
;Una vez inicializado el puerto, se procede leer las patitas RA0 y RA1
    BCF STATUS,RP0      ;regresa al banco 0
chRA0 BTFSS PORTA,0     ;checa si RA0=1
      GOTO chRA1        ;si RA0=0 salta a checar RA1
Apaga BCF PORTA,5       ;si RA0=1 apaga el LED
chRA1 BTFSS PORTA,1     ;checa si RA1=1
      GOTO chRA0        ;si RA1=0 salta a checar RA0
Prend BSF PORTA,5       ;si RA1=1 enciende el LED
      GOTO chRA0
    end
```

Hardware necesario.- En la siguiente figura se muestra la conexión de los dos botones y el LED necesarios para probar el programa, además claro está de la circuitería de reloj necesaria.

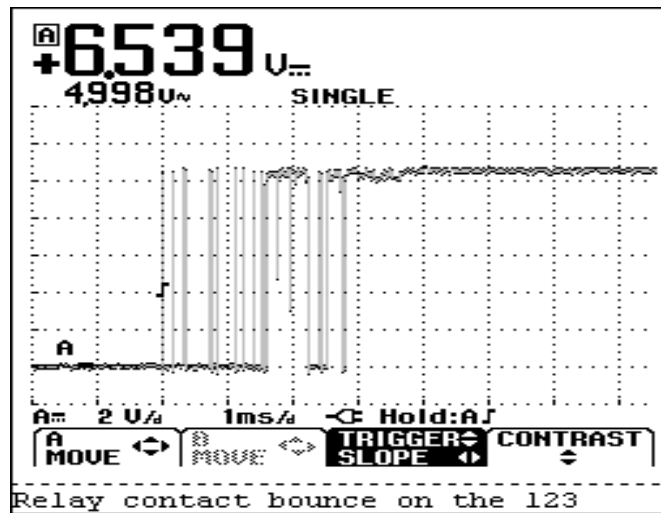


El efecto rebote y su eliminación

En el momento de presionar un botón pulsador o cualquier conmutador electromecánico es inevitable que se produzca un pequeño arco eléctrico durante el breve instante en que las placas del contacto se aproximan o se alejan de sus puntos de conexión, ya que las distancias microscópicas entre los puntos a diferente potencial son suficientes para romper por un instante la resistividad del aire. Este fenómeno se conoce como **rebote (bounce)** y se ilustra en la siguiente figura



Duración del rebote.- La experiencia empírica indica que el periodo transitorio de un rebote depende entre otros factores, de la calidad de los switches y de la rapidez de su accionamiento, pero a lo más puede durar unos **20 milisegundos**, En la siguiente figura se muestra un rebote real en los contactos de un relevador capturado en la pantalla de un osciloscopio digital Fluke 123. (Obsérvese que el rebote mostrado duró solamente 3 milisegundos).



Ejemplo 3: Limpieza del rebote (debouncing).- El rebote generado por un switch no siempre es un problema, por ejemplo el programa del ejemplo anterior funciona correctamente haya o no haya rebote. Sin embargo, ¿qué pasaría si se desea controlar el encendido y apagado del LED con un solo botón? En este caso sí se debe contemplar el **limpiado del rebote**. (Siempre que a una sola tecla le asignemos dos o más funciones diferentes en un programa debemos considerar el efecto del rebote). En este ejemplo se usa sólo el botón conectado a la patita RA0 para controlar el encendido y apagado del LED, y se incluye el limpiado del rebote.

```
;*Este programa Enciende un LED conectado a RA5 cuando se presiona un botón
;*conectado a RA0 y lo apaga cuando se presiona nuevamente el
;*mismo botón y a sí sucesivamente.
;*****
    include "p16f877.inc"
    org 0x0000          ;Inicia en el vector de reset
;Inicialización del puerto A:
    CLRF STATUS         ;Selecciona Banco 0
    CLRF PORTA          ;Inicializa latches de datos de PORTA
    BSF STATUS,RP0      ;Selecciona Banco 1
    MOVLW 0x06          ;Configura todas las patitas de A
    MOVWF ADCON1         ;como digitales
    MOVLW 0x1F          ;configura todas las patitas de A como salidas
    MOVWF TRISA         ;y RA5 como salida
    BCF STATUS,RP0      ;regresa al banco 0
checa BTFSS PORTA,0      ;checa si RA0=1
    GOTO chequea        ;si RA=0 chequea de nuevo
    CALL d20ms          ;si RA0=1 espera 20 miliseg. A que pase el rebote
checl BTFSS PORTA,0      ;checa nuevamente si RA0=1
    GOTO chequea        ;si RA1=0 falsa alarma, vuelve a chequear
    MOVLWF PORTA,0x20   ;si RA1=1 señal válida; carga máscara en W
    XORWF PORTA,1       ;conmuta estado del LED
    GOTO chequea        ;repite
end
```

Hardware necesario.- El hardware es igual al del ejemplo 2, pero usando sólo el botón conectado a RA0.

Ejemplo 4. Pausa de 20 milisegundos.- En el código anterior se resalta la parte correspondiente al limpiado del rebote, la cual incluye un subrutina (**d20ms**) que no se muestra en el recuadro, esta rutina simplemente realiza una pausa de 20 milisegundos. A continuación se muestran dos diferentes implementaciones de esta rutina (**suponiendo un reloj de 100 KHz**):

```
;* Rutina de Pausa que dura 20 milisegundos (usando memoria de datos)
;* (Supone un reloj de 100 KHz)
;*****
;Define constantes
N      EQU 0xA5
cont   EQU 0x20
; inicia rutina
d20ms  MOVLW N           ;Carga dato que controla la duración
      MOVWF cont        ;inicializa contador con el dato
rep    DECFSZ cont,1     ;Decrementa contador y escapa si cero
      GOTO rep          ;si no es cero, repite
esc    RETURN           ;regresa de esta subrutina
end
```

```
;* Rutina de Pausa que dura 20 milisegundos (sin usar memoria de datos)
;* (Supone un reloj de 100 KHz)
;*****
;Define dato que controla la duración
N      EQU ???
;inicia rutina
d20ms  MOVLW N           ;Carga dato que controla la duración
rep    ADDLW 0xFF        ;Le suma -1
      BTFSS STATUS,2    ;Si es cero escapa
      GOTO rep          ;si no es cero, repite
esc    RETURN           ;regresa de esta subrutina
end
```

Cálculo de N.- Para lograr que la duración de la subrutina presentada sea de 20 milisegundos es necesario elegir adecuadamente el valor de la constante N. Para ello ejemplificamos el cálculo en la primera de las dos versiones: A continuación se repite el código fuente de la rutina incluyendo entre paréntesis el número de ciclos que dura cada instrucción:

```
d20ms  MOVLW N           ;(1)
      MOVWF cont        ;(1)
rep    DECFSZ cont,1     ;(1 si no escapa, 2 si escapa)
      GOTO rep          ;(2)
esc    RETURN           ;(2)
```

De manera que el número de ciclos de instrucción T_{sub} consumidos por la rutina, incluyendo los 2 ciclos de la llamada (CALL) serán

$$T_{sub} = [2+1+1+(N-1)*(1+2)+2+2] \text{ ciclos}$$

Lo cual se puede expresar en segundos como

$$T_{sub} = (3N+5) T_{cy}$$

Donde T_{cy} es la duración en segundos de un ciclo de instrucción. **Suponiendo que se está usando un reloj de 100 KHz**, ($T_{cy}=40\mu\text{seg}$), despejando

$$N = (T_{\text{sub}}/T_{cy} - 5)/3$$

Sustituyendo $T_{\text{sub}} = 20 \text{ mseg}$; entonces $N = 165 = A5H$.

- **Observación.** La máxima duración que se puede lograr con esta rutina (Para $N=0$ el ciclo se ejecuta 256 veces) es de 30.92 mseg (con la frecuencia del reloj de 100 KHz supuesta). Así, si la frecuencia del reloj es mayor, por ejemplo, en el modo de oscilador interno se tienen 4 Mhz y $T_{cy}=1\mu\text{seg}$ esta rutina no podrá proporcionar nunca los 20 mseg y habrá que realizar una pausa más larga anidando dos ciclos.

Ejercicio.- Calcula la duración en ciclos de instrucción y en segundos para la segunda versión en términos de N . Despeja N para $T_{\text{sub}} = 20 \text{ mseg}$. ¿Cuál es la máxima duración de esta rutina? ¿con qué valor de N se logra?

Ejemplo 5. Pausa con dos ciclos anidados.- Como ya se mencionó, la pausa del ejemplo 4 es demasiado corta, de manera que para frecuencias altas de reloj no puede alcanzar ni siquiera los 20 mseg. A continuación se muestra una pausa que puede dar tiempos mucho más largos usando ciclos anidados:

```
;* Rutina de Pausa para tiempos largos
;* (Adecuada para un reloj de 4 Mhz)
;*****
;Define constantes Para 20 milisegundos
N      EQU 0x1A
M      EQU 0x0
cont1 EQU 0x20
cont2 EQU 0x21
;inicia rutina
pau    MOVLW N           ;(1) Carga dato N
        MOVWF cont1      ;(1) inicializa contador1 ciclo externo
rep1   MOVLW M           ;(1) Carga dato M
        MOVWF cont2      ;(1) inicializa contador2 ciclo interno
rep2   DECFSZ cont2,1     ;(1,2)Decrementa contador2 y escapa si cero
        GOTO rep2        ;(2) si no es cero, repite ciclo interno
        DECFSZ cont1,1    ;(1,2)Decrementa contador1 y escapa si cero
        GOTO rep1        ;(2) si no es cero repite ciclo externo
esc    RETURN           ;(2) regresa de esta subrutina
end
```

La duración de esta rutina en ciclos de reloj será

$$T_{\text{sub}} = 2+1+1+N*[1+1+(M-1)*(1+2)+2+1+2]+1-2+2+2 \text{ ciclos}$$

Lo cual se puede simplificar como sigue

$$T_{\text{sub}} = [N*(3M+4)+7] T_{cy}$$

La máxima duración de esta rutina se obtiene para $N=M=256$, lo cual en el programa se logra con $N=M=0$; entonces $T_{sub} \text{ máximo} = 197,639 T_{cy}$. Para una frecuencia de 4 Mhz $T_{cy} = 1 \mu\text{seg}$, entonces $T_{sub} \text{ máximo} = 0.197639 \text{ seg}$.

Si deseamos $T_{sub} = 20 \text{ mseg}$, haciendo $M=256$, despejando N

$$N = (T_{sub}/T_{cy} - 7)/(3M+4)$$

Sustituyendo valores $N = 25.89 \approx 1Ah$. (debido a la aproximación la rutina realmente dura 20.08 mseg).

- **Observación.** Hasta aquí se están realizando pausas de duración controlada por software, sin embargo, lo más adecuado para controlar tiempo de manera más exacta es usar los timers que se verán más adelante.

3.1.2.- El Puerto B (PORTB)

El puerto B es un puerto digital de 8 bits, todas sus patitas son bidireccionales y trabaja en forma similar al puerto A. Tiene tres registros asociados: El registro de datos PORTB, el registro de dirección de los datos TRISB y el registro OPTION_REG.

Registro PORTB (06H, 106H) .- Los ocho bits que contiene reflejan directamente el estado de las ocho patitas del puerto B RB0,...,RB7.

Registro TRISB (86H, 186H).- En forma similar a TRISA, al poner un 0 en un bit de TRISB se configura la patita RB correspondiente como salida y al poner un 1 en un bit de TRISB se configura la patita RB correspondiente como entrada.

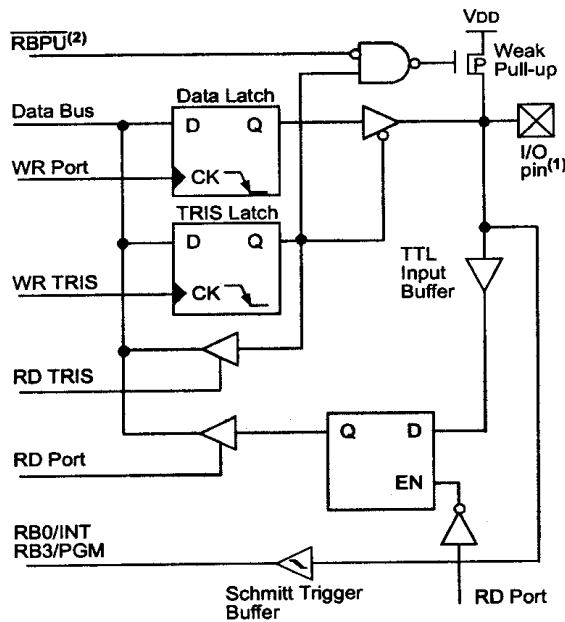
Registro OPTION_REG (81H, 181H).- El bit 7 de este registro, denominado RBPU es usado para conectar/desconectar una resistencia “pull-up” conectada a cada patita RB. Poniendo un 0 en este bit todas las resistencias se conectan. Para desconectar las resistencias “pull-up” se debe poner este bit en 1, también se desconectan automáticamente cuando la patita correspondiente es configurada como salida. Un Reset desconecta todas las resistencias.

Patitas RB4,...,RB7.- Estas cuatro patitas del puerto B tienen la capacidad de generar una solicitud de interrupción a la CPU cuando están configuradas como entradas. El estado de estas patitas es comparado con el último estado que tenían durante la última lectura a PORTB, guardado en un latch. Los bits que indican que hay una diferencia entre estos valores por cada patita están conectados a una puerta OR cuya salida activa el bit RBIF del registro INTCON solicitando con esto una interrupción. Esta interrupción es especialmente útil para despertar al dispositivo de su estado de SLEEP cuando alguna de las cuatro líneas es activada, por ejemplo, en respuesta a la presión de una tecla.

Esta característica de solicitud de interrupción cuando se detecta un cambio junto con las resistencias “pull-up” configurables para estas cuatro patitas, las hacen ideales para

el manejo de teclados en dispositivos portátiles que requieren “dormirse” durante largos ratos para economizar baterías y “despertarse” cuando una tecla es presionada.

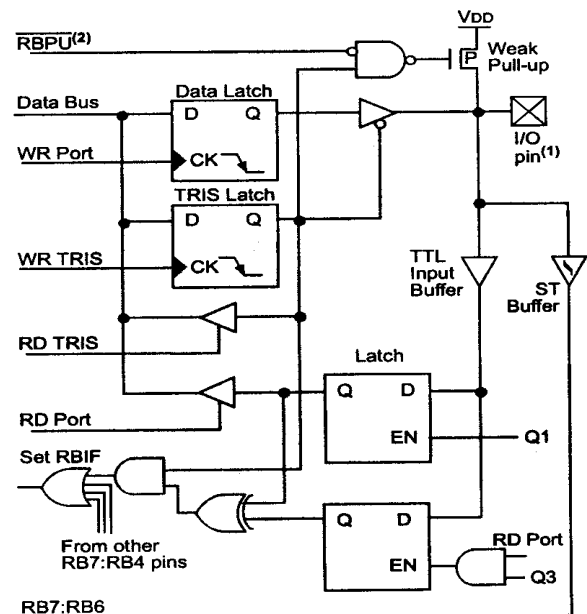
En la siguientes figuras se muestra el alambrado interno de las patitas del puerto B.



Note 1: I/O pins have diode protection to V_{DD} and V_{SS}.

2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Patitas RB0,...RB3



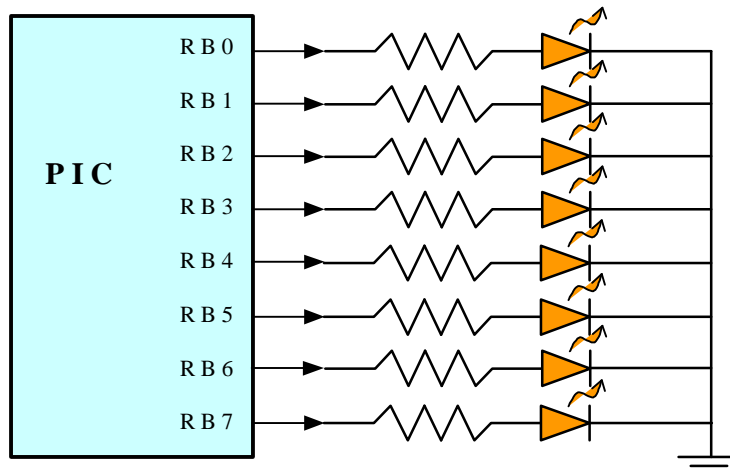
Note 1: I/O pins have diode protection to V_{DD} and V_{SS}.

2: To enable weak pull-ups, set the appropriate TRIS bit(s) and clear the RBPU bit (OPTION_REG<7>).

Patitas RB4,...,RB7

Ejemplo 6. Luces secuenciales.- En este ejemplo se realiza el encendido secuencial de 8 LEDs conectados a RB0,...,RB7, es decir, se enciende RB0 durante un segundo, luego RB1 y así sucesivamente hasta llegar a RB7 para recomenzar después con RB0. (en un segundo dado sólo un LED está prendido).

Hardware necesario.- Sólo se requieren los ocho LEDs conectados a las patitas RBP0,...,RBP7, como se muestra en la siguiente figura.



```

;* Este programa Enciende en secuencia (uno a la vez) 8 LEDs conectados
;* a las patitas del puerto B
;*****
;
;    Include "p16f877.inc"
;    org 0x0000      ;Inicia en el vector de reset
;Inicialización del puerto B:
;    CLRF STATUS      ;Selecciona Banco 0
;    CLRF PORTB        ;Inicializa latches de datos de PORTB
;    BSF STATUS,RP0    ;Selecciona Banco 1
;    CLRF TRISB        ;Configura todas las patitas de B como salidas
;Una vez inicializado el puerto, se procede a controlar los LEDs
;    CLRF STATUS,RP0   ;regresa al banco 0 y limpia acarreo
;    CLRF PORTB        ;Apaga todos los LEDs
;    BSF PORTB,0       ;enciende el LED RB0
ciclo CALL dlseg       ;pausa de un segundo
;    RLF PORTB,1       ;Recorre posición encendida a la izquierda
;    GOTO ciclo        ;repite
;    end

```

- **Observación.** Si se usa el módulo ICD (In Circuit Debugger) para probar este programa se debe tomar en cuenta que este módulo usa las líneas RB3, RB6 y RB7 del puerto B para comunicación y programación del PIC.

Ejemplo 7. Pausa de 1 segundo.- Para realizar la pausa de un segundo que se requiere para el programa anterior podemos hacer uso de la pausa implementada en el ejemplo 5 con su máxima duración $T_{sub} = 0.197639$ seg (con $N=M=0$). Para ello implementamos un tercer ciclo externo que repita esta pausa P veces.

```

;* Subrutina de Pausa de 1 segundo
;*****
P    EQU 0x05
cont3 EQU 0x22
dlseg MOVLW P      ;(1)carga duración del ciclo
      MOVWF cont3  ;(1)inicializa contador3
ciclo CALL pau     ;(196868)pausa de 0.197639 seg
      DECFSZ cont3,1 ;(1,2)Decrementa y escapa si cero
      GOTO ciclo    ;(2)si no es cero repite
      RETURN        ;(2)si es cero retorna

```

La duración de esta rutina incluyendo la llamada será

$$T_{sub} = 2+1+1+(P)*(197639+1+2)+1-2+2 \text{ ciclos}$$

Es decir,

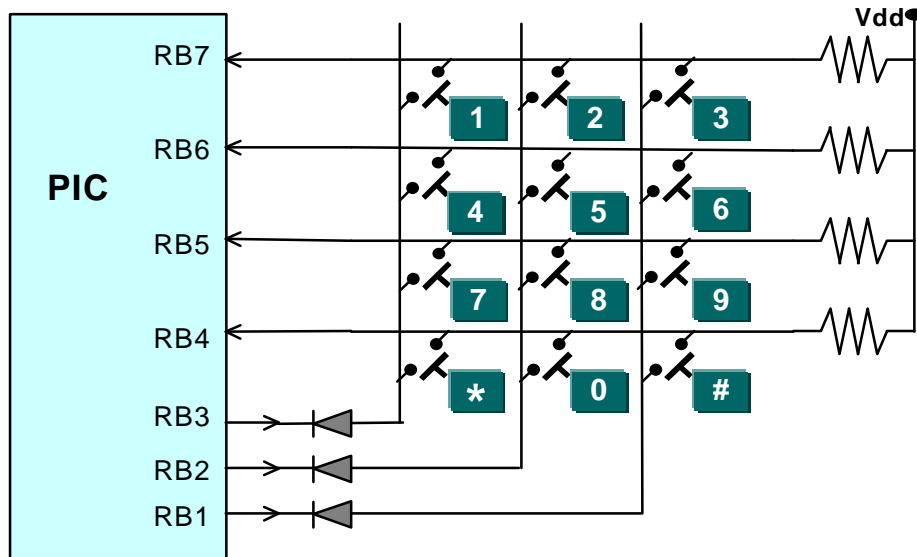
$$T_{sub} = 197642 \cdot P + 5 \text{ Tcy}$$

La máxima duración de esta rutina (para $P=256$), suponiendo un reloj de 4 Mhz, se tiene un T_{sub} máximo = 50.596,357 segundos. Sin embargo, si deseamos un $T_{sub}=1$ seg. Obtenemos, despejando

$$P = (T_{\text{sub}} - 5) / 197642 = 5.059628 \approx 5$$

Ejemplo 8.- Conexión de un teclado matricial

Una de las funciones más comunes de un microcontrolador es la aceptación de datos numéricos o alfanuméricos suministrados por un operador mediante un teclado. La mayoría de los teclados están organizados en forma matricial como un conjunto de switches en las intersecciones de varios renglones y columnas conductoras como se muestra en la siguiente figura que ejemplifica un teclado de tipo telefónico.



En la figura también se ilustra una conexión típica con tres líneas de salida del puerto B que controlan las columnas y cuatro líneas de salida del mismo puerto que recogen la información de los renglones.

- **Observación.** Es recomendable colocar un diodo de protección en cada línea de salida del puerto para evitar que al activar más de una columna a la vez se produzca un corto circuito.

Procedimiento de detección de teclas y codificación.- Para la detección y asignación de código según la tecla presionada se procede como sigue:

1. **Detección.** Se ponen en bajo todos los renglones (cuidando que haya diodos de protección) y se leen las columnas .
2. Si hay alguna columna activa se limpia el rebote, si es tecla válida se pasa al paso 3, si no es tecla válida se asigna un código de “ninguna tecla presionada”.
3. **Codificación.** El puerto activa un renglón a la vez colocando un cero lógico en la línea correspondiente al renglón a activar.

4. Por cada renglón activo se lee la información de columnas y dependiendo del renglón y la columna activada (en bajo) se asigna el código a la tecla de la intersección.

```

; * Estas subrutinas realizan la inicialización y detección de un teclado
; * tipo telefónico controlando las columnas con las salidas RB1,RB2 y RB3
; * y los renglones con las entradas RB4,RB5,RB6 y RB7
; * con resistencias pull-up internas para evitar resistencias externas.
;*****
;Subrutina de inicialización del puerto B:
initB CLRF STATUS      ;Selecciona Banco 0
      CLRF PORTB       ;Inicializa latches de datos de PORTB
      BSF STATUS,RP0    ;Selecciona Banco 1
      MOVLW 0xF0        ;configura RB7,...,RB4 como entradas
      MOVWF TRISB       ;y RB3,RB2,RB1 como salidas
      BCF OPTION_REG,7  ;Conecta todas las resistencias Pull-Up
      BCF STATUS,RP0    ;regresa al Banco 0
      RETURN

;Subrutina de Detección de tecla presionada: regresa w=0 si no hay tecla
;presionada y w=0xFF si hay alguna tecla presionada
detec CLRF PORTB,1      ;activa las cuatro columnas
      BTFSS PORTB,7     ;lee renglón 1,2,3
      GOTO rebo         ;si tecla presionada limpia rebote
      BTFSS PORTB,6     ;lee renglón 4,5,6
      GOTO rebo         ;si tecla presionada limpia rebote
      BTFSS PORTB,5     ;lee renglón 7,8,9
      GOTO rebo         ;si tecla presionada limpia rebote
      BTFSS PORTB,4     ;lee renglón *,0,#
      GOTO rebo         ;si tecla presionada limpia rebote
      RETLW 0x0         ;no hubo tecla presionada retorna con w=0
rebo  CALL d20ms        ;pausa de 20 milisegundos
      BTFSS PORTB,7     ;lee renglón 1,2,3
      RETLW 0xFF        ;tecla presionada, retorna con w=0xFF
      BTFSS PORTB,6     ;lee renglón 4,5,6
      RETLW 0xFF        ;tecla presionada, retorna con w=0xFF
      BTFSS PORTB,5     ;lee renglón 7,8,9
      RETLW 0xFF        ;tecla presionada, retorna con w=0xFF
      BTFSS PORTB,4     ;lee renglón *,0,#
      RETLW 0xFF        ;tecla presionada, retorna con w=0xFF
      RETLW 0x0         ;falsa alarma retorna con w=0

```

A continuación se presenta la rutina de codificación que asigna el código ASCII de la tecla presionada y lo devuelve en el registro W

```


; * Esta subrutina realiza la codificación del teclado tipo telefónico
; * retornando en W el código ASCII de la tecla presionada
; * regresa W=0 si no hubo tecla presionada
;*****
codif MOVLW 0xF7        ;desactiva todas las columnas
      MOVWF PORTB       ;y activa la columna 1,4,7,*
      BTFSS PORTB,7     ;Es la tecla 1?
      RETLW '1'         ;retorna código del '1'
      BTFSS PORTB,6     ;Es la tecla 4?
      RETLW '4'         ;retorna código del '4'

```

```

BTFSS PORTB,5      ;Es la tecla 7?
RETLW '7'          ;retorna código del '7'
BTFSS PORTB,4      ;Es la tecla *?
RETLW '*'          ;Retorna código del '*'
MOVLW 0x0FB        ;desactiva todas las columnas
MOVWF PORTB        ;y activa la columna 2,5,8,0
BTFSS PORTB,7      ;Es la tecla 2?
RETLW '2'          ;retorna código del '2'
BTFSS PORTB,6      ;Es la tecla 5?
RETLW '5'          ;retorna código del '5'
BTFSS PORTB,5      ;Es la tecla 8?
RETLW '8'          ;retorna código del '8'
BTFSS PORTB,4      ;Es la tecla 0?
RETLW '0'          ;Retorna código del '0'
MOVLW 0x0FD        ;desactiva todas las columnas
MOVWF PORTB        ;y activa la columna 3,6,9,#
BTFSS PORTB,7      ;Es la tecla 3?
RETLW '3'          ;retorna código del '3'
BTFSS PORTB,6      ;Es la tecla 6?
RETLW '6'          ;retorna código del '6'
BTFSS PORTB,5      ;Es la tecla 9?
RETLW '9'          ;retorna código del '9'
BTFSS PORTB,4      ;Es la tecla #?
RETLW '#'          ;Retorna código del '#'
RETLW 0x00         ;falsa alarma, no hay tecla presionada

```

 **Observación:** La repetición de código se puede evitar utilizando direccionamiento indirecto dentro de un ciclo, sin embargo, esto sólo representa un ahorro de código y de memoria para teclados de mayor complejidad.

3.1.3.- El puerto C (PORTC).

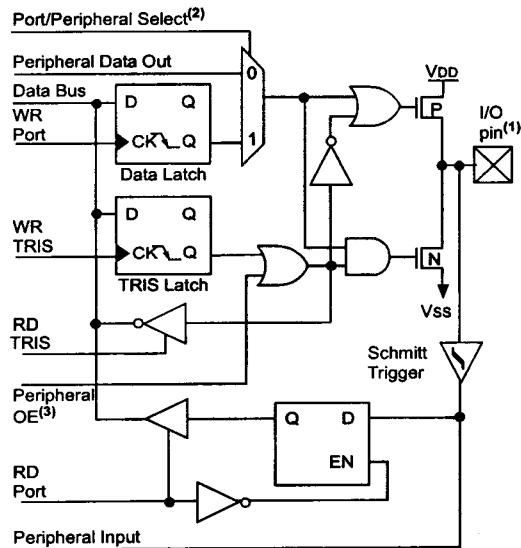
El puerto C consta de 8 líneas bidireccionales. Trabaja en forma similar a los dos puertos anteriores y tiene asociados los registros:

Registro PORTC (07H).- Es el registro de datos cuyos 8 bits RC7,RC6,...,RC0 reflejan directamente el valor lógico de las líneas físicas del puerto C.

Registro TRISC(87H).- Registro de control de dirección de las líneas del puerto C. Poniendo un 1 en un bit del registro TRISC se configura la línea correspondiente como entrada y poniendo un 0 se configura la línea correspondiente como salida.

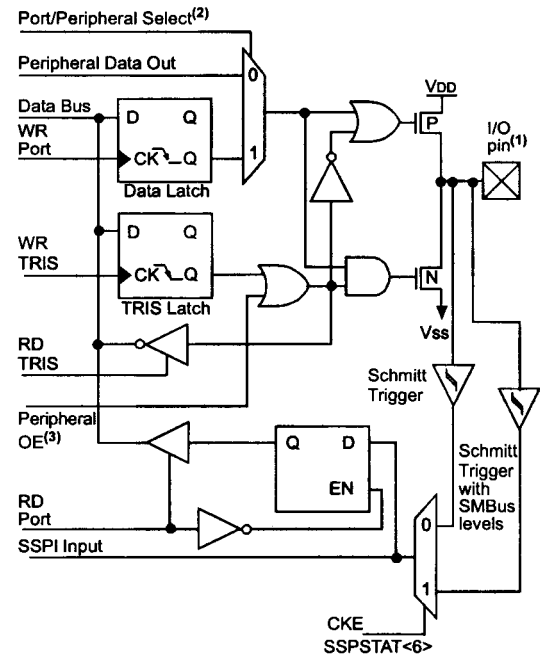
Las líneas del puerto C se encuentran multiplexadas con varias líneas controladas por otros periféricos, cuando se habilita la línea del periférico respectivo puede ser ignorada la configuración de TRISC, de hecho, algunos periféricos configuran la línea como salida mientras que otros la configuran como entrada.

Cada entrada del puerto C posee un buffer con disparador Schmitt. Además, cuando se selecciona la función I²C, las patitas PORTC<4,3> pueden ser configuradas con niveles I²C o con niveles SMBus mediante el bit CKE del registro SSPSTAT<6>, como se muestra en las figuras siguientes.



- Note 1:** I/O pins have diode protection to VDD and VSS.
Note 2: Port/Peripheral select signal selects between port data and peripheral output.
Note 3: Peripheral OE (output enable) is only activated if peripheral select is active.

Patitas 7,6,5,2,1 y 0 del puerto C



- Note 1:** I/O pins have diode protection to VDD and VSS.
Note 2: Port/Peripheral select signal selects between port data and peripheral output.
Note 3: Peripheral OE (output enable) is only activated if peripheral select is active.

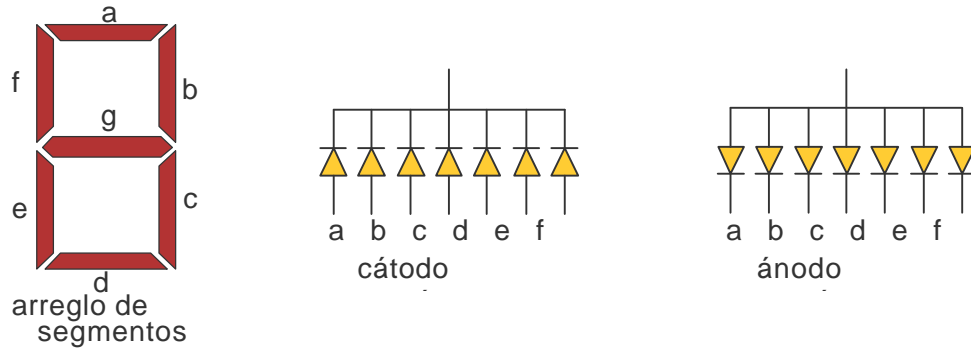
Patitas 4 y 3 del puerto C

En la siguiente tabla se resumen las líneas del puerto C y las de los periféricos que están multiplexadas con ellas.

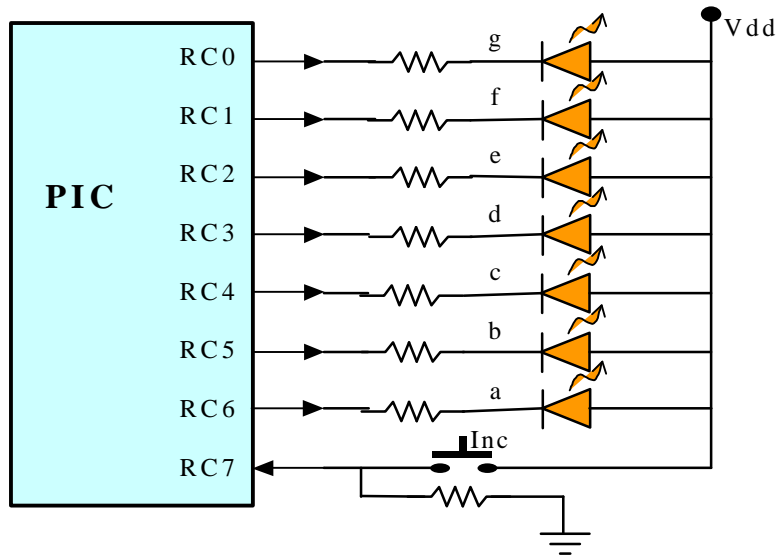
Nombre	Función multiplexada
RC0/T1OSO/T1CKI	Salida oscilatoria del Timer1/reloj de entrada del Timer 1
RC1/T1OSI/CCP2	Entrada oscilatoria del Timer1/entrada de captura2 o salida de comparación2 o salida PWM2
RC2/CCP1	Entrada de captura1 o salida de comparación1 o salida PWM1
RC3/SCK/SCL	Reloj para los modos de comunicación serie síncrona SPI e I ² C
RC4/SDI/SDA	Dato de entrada (en modo SPI)/ Dato de entrada-salida (modo I ² C)
RC5/SDO	Dato de salida (en modo SPI)
RC6/TX/CK	Línea de transmisión asíncrona de la USART/reloj síncrono
RC7/RX/DT	Línea de recepción asíncrona de la USART/dato síncrono

Ejemplo 9.- Manejo de un Display de 7 segmentos

Un display de siete segmentos es un arreglo de 7 LEDs conectados como se muestra en la siguiente figura:



Para este ejemplo se supone un display de ánodo común conectado al puerto C como se muestra a continuación



El programa visualiza dígitos numéricos hexadecimales en el display, de acuerdo a la codificación mostrada en la siguiente tabla

Dato	a	b	c	d	e	f	g	Código
0	0	0	0	0	0	0	1	01
1	1	0	0	1	1	1	1	4F
2	0	0	1	0	0	1	0	12
3	0	0	0	0	1	1	0	06
4	1	0	0	1	1	0	0	4C
5	0	1	0	0	1	0	0	24
6	0	1	1	1	1	1	1	3F
7	0	0	0	1	1	1	1	0F
8	0	0	0	0	0	0	0	00
9	0	0	0	0	1	0	0	04

Dato	a	b	c	d	e	f	g	Código
A	0	0	0	1	0	0	0	08
b	1	1	0	0	0	0	0	60
C	0	1	1	0	0	0	1	31
d	1	0	0	0	0	1	0	42
E	0	1	1	0	0	0	0	30
F	0	1	1	1	0	0	0	38

```

; * Este programa despliega un dígito hexadecimal en un display de ánodo
; * común conectado al puerto C. El dígito se incrementa en 1 cada vez que
; * se presiona un botón conectado al MSB del mismo puerto C
    
```

```

;*****
Include "p16f877.inc"
cont equ 0x20
org 0x0000
inic BSF STATUS,RP0      ;banco 1
    MOVLW 0x80           ;Todos los bits del puerto C como salidas
    MOVWF TRISC          ;y el MSB como entrada
    BCF STATUS,RP0      ;regresa al Banco 0
    MOVLW 0x01          ;
    MOVWF PORTC         ;Despliega un cero
    CLRF cont           ;Inicializa contador en cero
tecla BTFSS PORTC,7      ;checa botón
    GOTO tecla          ;Si no se ha presionado espera
    CALL d20ms          ;pausa de 20 milisegundos
    BTFSS PORTC,7      ;checa nuevamente el botón
    GOTO tecla          ;tecla falsa, espera de nuevo
    INCF cont,1         ;tecla válida, incrementa contador
    CALL codigo         ;obtiene código para desplegar el contador
    MOVWF PORTC         ;despliega contador
suelt BTFSC PORTC,7      ;checa de nuevo el botón
    GOTO suelt          ;si sigue presionado espera
    GOTO tecla          ;si ya se soltó espera nueva presión.
codigo:
    MOVLW 0x0F          ;carga máscara
    ANDWF cont,0        ;enmascara el contador y lo deja en W
    ADDWF PCL,1         ;Salta W instrucciones adelante
    RETLW 0x01          ;código del 0
    RETLW 0x4F          ;código del 1
    RETLW 0x12          ;código del 2
    RETLW 0x06          ;código del 3
    RETLW 0x4C          ;código del 4
    RETLW 0x24          ;código del 5
    RETLW 0x3F          ;código del 6
    RETLW 0x0F          ;código del 7
    RETLW 0x00          ;código del 8
    RETLW 0x04          ;código del 9
    RETLW 0x08          ;código de la A
    RETLW 0x60          ;código de la b
    ...
    RETLW 0x38          ;código de la F
end

```

Observación: Una manera más cómoda de escribir la lista de instrucciones RETLW al final del programa anterior puede lograrse usando la directiva DT (Define Table) del ensamblador, la cual nos permite definir una tabla de datos que será sustituida por una lista de instrucciones RETLW; así, la lista anterior puede quedar como sigue:

```

DT 0x01,0x4F,0x12,0x06,0x4C,0x24,0x3F,0x0F
DT 0x00,0x04,0x08,0x60,0x31,0x42,0x30,0x38

```

3.1.4.- Los Puertos D y E

Estos dos puertos **no** se encuentran disponibles en los circuitos PIC16F873 y PIC16F876.

El puerto D es un puerto de 8 líneas configurables como entradas o salidas mediante el registro **TRISD** (88H) y cuyas líneas pueden ser accesadas mediante el registro **PORTD** (08H). Cuando se configuran como entradas éstas poseen un disparador Schmitt.

El Puerto E sólo posee 3 líneas configurables como entradas o salidas mediante el los 3 bits menos significativos del registro **TRISE** (89H). Sus líneas pueden ser accesadas mediante los 3 bits menos significativos del registro **PORTE** (09H). Las líneas del puerto E están compartidas con el convertidor analógico / digital, por ello, antes de usarlas deberán ser configuradas como entradas / salidas digitales o analógicas, según se desee en forma similar a como se hizo con el puerto A, usando el registro de configuración **ADCON1** (9FH).

El Puerto D puede configurarse para trabajar simultáneamente con sus 8 bits como un puerto esclavo (**Parallel Slave Port**) de comunicación paralela bidireccional con líneas de protocolo proporcionadas por las tres líneas del Puerto E, para ello se deberá activar el bit **PSPMODE** (**TRISE<4>**).

3.2.- El Puerto Serie USART

3.2.1.- Introducción

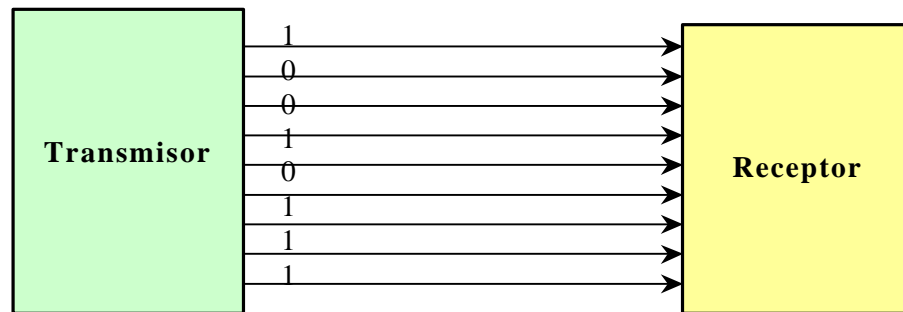
La USART (Universal Synchronous Asynchronous Receiver Transmitter) es uno de los dos periféricos contenidos en el PIC que le permiten realizar comunicación en serie. El otro es el MSSP (Master Synchronous Serial Port), el cual no es tratado en estas notas. La USART, también conocida como SCI (Serial Communications Interface) puede configurarse como una unidad de comunicación en serie para la transmisión de datos asíncrona con dispositivos tales como terminales de computadora o computadoras personales, o bien para comunicación síncrona con dispositivos tales como convertidores A/D o D/A, circuitos integrados o memorias EEPROM con comunicación serie, etc.

La gran mayoría de los sistemas de comunicación de datos digitales actuales utilizan la comunicación en serie, debido a las grandes ventajas que representa esta manera de comunicar los datos:

- **Económica.-** Utiliza pocas líneas de transmisión inclusive puede usar sólo una línea.
- **Confiable.-** Los estándares actuales permiten transmitir datos con bits de paridad y a niveles de voltaje o corriente que los hacen poco sensibles a ruido externo. Además por tratarse de información digital, los cambios en amplitud de la señales (normalmente causados por ruido) afectan muy poco o nada a la información.
- **Versátil.-** No está limitada a usar conductores eléctricos como medio de transmisión, pudiendo usarse también: fibra óptica, aire, vacío, etc. Además el tipo de energía utilizada puede ser diferente: luz visible, infrarroja, ultrasonido, pulsos eléctricos, radio frecuencia, microondas, etc.

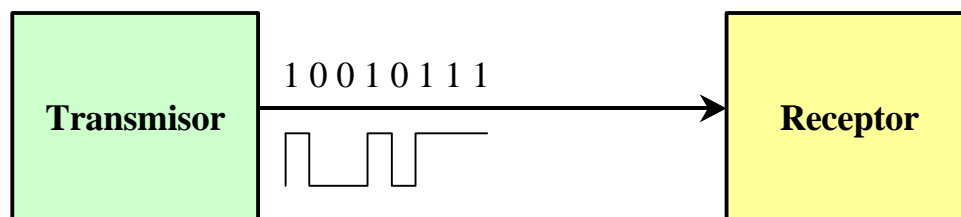
Una gran cantidad de periféricos se comunican actualmente en serie con una micro computadora: líneas telefónicas, terminales remotas, unidades de cassette magnético, el ratón, teclados, etc.

Comunicación en paralelo.- En este caso se utiliza una línea física por cada bit del dato a comunicar además de posibles líneas para protocolo. Esquemáticamente en la siguiente figura se muestra como se transmitiría el dato de 8 bits 1001 0111= 97h. Este tipo de comunicación se puede realizar mediante el PIC usando el puerto D como puerto de datos y las líneas del puerto E como líneas de protocolo.



La principal ventaja de la comunicación paralelo es la alta velocidad de transmisión, ya que se envían simultáneamente todos los bits de un dato. No obstante, si la distancia entre el transmisor y el receptor es grande, puede ser que el costo de las líneas sea tan alto que se vuelva incosteable este método de comunicación.

Comunicación en Serie: En cambio, la comunicación en serie sólo utiliza una línea para la transmisión de datos, y opcionalmente alguna línea o líneas para protocolo. Por ejemplo, en la siguiente figura se muestra como se transmitiría en serie el mismo dato (97h):



La desventaja obvia de la comunicación serie es que los bits de un dato se envían de a uno por uno, de manera que mientras que la comunicación en paralelo envía en un ciclo un dato de 8 bits, a la comunicación serie le toma más de 8 ciclos (ya que además del dato en la comunicación serie se requiere agregar algunos bits de sincronización).

Sin embargo, debido a que la comunicación serie requiere sólo una línea para la transmisión esto abarata los costos en líneas de transmisión y no sólo esto, ya que este hecho también hace posible que los datos puedan ser enviados no necesariamente por un conductor eléctrico, sino inclusive por aire o por el vacío si en lugar de pulsos eléctricos se usan impulsos electromagnéticos, tales como: ondas de radio, microondas, pulsos luminosos, infrarrojo, ultrasonido, láser (a través de fibra óptica), etc.

3.2.2.- Protocolo de comunicación serie.

A diferencia de la comunicación en paralelo, en la comunicación en serie se hace necesario establecer métodos de sincronización para evitar la interpretación errónea de los datos transmitidos. Para ilustrar esto consideremos la siguiente información en serie:

...0100110001001100100...

Esta información puede interpretarse de diversas maneras, (aún si se recibe a la velocidad adecuada) dependiendo del punto de inicio de separación de datos, por ejemplo, una posible interpretación sería como sigue:

... 01	00110001	00110010	0 ...
--------	----------	----------	-------

Que interpretado como códigos ASCII corresponde a los caracteres '1' y '2'. Sin embargo, otra posible interpretación es:

... 010	01100010	01100100
---------	----------	----------

Que corresponde a los caracteres 'b' y 'd' .

Sincronización de bit.- Una manera de resolver el problema anterior es la sincronización de bits que puede realizarse por varios métodos:

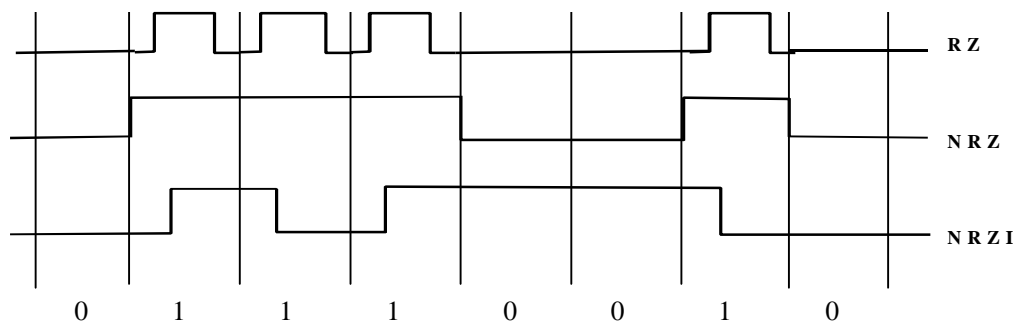
- 1) Enviar por una línea adicional una señal de reloj que indique el centro de las celdas de bits en la línea de datos (**datos no - auto reloj**).
- 2) Enviar con cada bit y por la misma línea de datos información que permita extraer la señal de reloj (**datos auto reloj**).
- 3) Lograr mediante alguna estrategia que los relojes de transmisión y de recepción se mantengan en fase continuamente.

Codificación no auto reloj.- En la figura siguiente se muestran las tres codificaciones de una línea de datos:

RZ.- Una celda de bit es 1 si contiene un impulso positivo y un 0 si no lo contiene.

NRZ.- La celda contiene un 1 o 0 de acuerdo al nivel de la señal (constante) en la celda.

NRZI.- La celda de bit contiene un 1 si hay una transición y un 0 si no la hay.



Como puede verse, en estos sistemas (RZ, NRZ y NRZI) las secuencias de ceros no contienen ninguna transición que permita ubicar la situación de las celdas de bit. De hecho, el formato NRZ no la contiene ni en los unos.

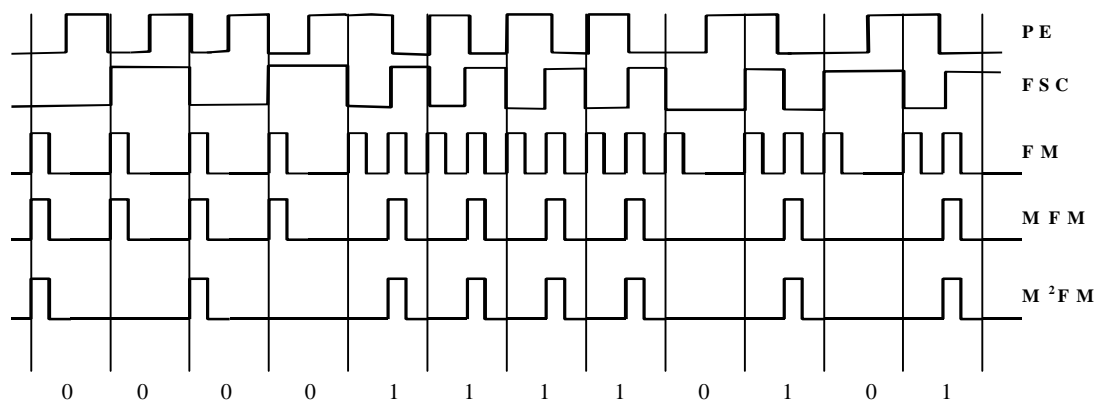
Codificación auto reloj.- Algunos métodos que contienen en la misma línea de datos información adicional para determinar la velocidad del reloj a costa de disminuir la cantidad de información útil a la mitad que los métodos no-auto reloj. En la siguiente figura se muestran los más utilizados, como son:

PE.- Codificación de fase.

FSC.- Codificación por cambio de frecuencia

FM.- Modulación de frecuencia.

MFM y M²FM.- Modulación de frecuencia modificadas.



Los métodos autoreloj son muy útiles cuando la velocidad de transmisión no es constante, por ejemplo, cuando los datos han sido grabados en un medio magnético giratorio, por ejemplo discos, cintas magnéticas, etc.

Sincronización de carácter.- Algunos sistemas utilizan líneas adicionales que envían impulsos para indicar el inicio de un bloque de caracteres. Otros sistemas que no requieren líneas adicionales a la línea de datos son:

Método Asíncrono.- Cada carácter va señalizado mediante dos bits: **un bit de inicio y un bit de paro**, estos dos bits permiten al receptor reconocer el inicio y el final de cada carácter. La especificación RS404 de EIA (Electronic Industries Association) define las características del método asíncrono para transmisión en serie de acuerdo a las siguientes reglas:

- 1) Cuando no se envían datos la línea debe mantenerse en estado 1.
- 2) Cuando se va a mandar un carácter se envía primero un bit de inicio de valor 0.
- 3) A continuación se envían todos los bits del carácter a transmitir al ritmo marcado por el reloj de transmisión.
- 4) Después del último bit del carácter enviado se envía un bit de paro de valor 1.

Método Síncrono.- Cada mensaje o bloque de transmisión va precedido de unos caracteres de sincronismo. Así, cuando el receptor identifica una configuración de bits

igual a la de los caracteres de sincronismo da por detectado el inicio y el tamaño de los datos.

Observación: Para el usuario de un microcontrolador que posee una USART o sistema similar la manera detallada como el sistema logra establecer la comunicación resulta transparente a él, ya que sólo tiene que configurar el protocolo del transmisor y del receptor para que estos logren la comunicación adecuada, es decir, el usuario usualmente sólo debe configurar:

tipo de comunicación (síncrona o asíncrona)
 velocidad de transmisión en Baudios (bits por segundo)
 longitud de los datos
 bits de inicio y de paro, bits de paridad, etc.

3.2.3.- La USART del PIC16F877

La USART del PIC puede ser configurada para operar en tres modos:

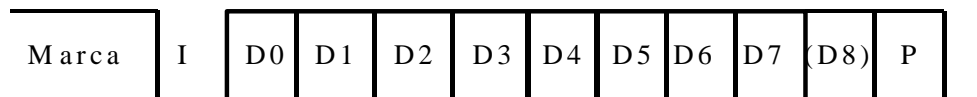
Modo Asíncrono (full duplex (transmisión y recepción simultáneas)),
 Modo Síncrono – Maestro (half duplex)
 Modo Síncrono – Esclavo (half duplex)

En estas notas sólo se cubre el modo asíncrono.

Modo Asíncrono.

En este modo la USART usa un formato estándar **NRZ** asíncrono, el cual para la sincronización usa: 1 bit de inicio (**I**), 8 o 9 bits de datos y 1 bit de paro (**P**). Mientras no se están transmitiendo datos la USART envía continuamente un bit de marca. El modo asíncrono se selecciona limpiando el bit SYNC del registro **TXSTA (98H)**. El modo asíncrono es deshabilitado durante el modo SLEEP.

Cada dato es transmitido y recibido comenzando por el LSB. El hardware no maneja bit de Paridad, pero el noveno bit puede ser usado para este fin y manejado por software.



El módulo Asíncrono de la USART consta de 3 módulos fundamentales:

El circuito de muestreo
 El generador de frecuencia de transmisión (Baud Rate)
 El transmisor asíncrono
 El receptor asíncrono.

El circuito de muestreo.- El dato en la patita de recepción (RC7/RX/DT) es muestreado tres veces para poder decidir mediante un circuito de mayoría, si se trata de un nivel alto o un nivel bajo.

3.2.3.1.- El Generador de Baud Rate (BRG)

Este generador sirve tanto para el modo síncrono como el asíncrono y consiste de un contador/divisor de frecuencia de 8 bits controlado por el registro **SPBRG (99H)**. De tal manera que la frecuencia de transmisión se calcula de acuerdo a la siguiente tabla:

SYNC	BRGH=0 (baja velocidad)	BRGH=1 (Alta velocidad)
0 (modo asíncrono)	Baud rate=Fosc/(64(X+1))	Baud rate=Fosc/(16(X+1))
1 (modo síncrono)	Baud rate=Fosc/(4(X+1))	-

En esta tabla X=valor de 8 bits en el registro del divisor, SPBRG. El bit BRGH corresponde a TXSTA<2>.

TABLE 10-3: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 0)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

TABLE 10-4: BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000	-	0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

Debido a que el divisor es de 8 bits, no se puede tener cualquier velocidad de transmisión deseada, ya que X se deberá redondear al entero más cercano. En las dos tablas anteriores se muestran algunos valores baud estándares, el divisor necesario (X=SPBRG) bajo diferentes frecuencias Fosc y el error producido en porcentaje

3.2.3.2.- El transmisor asíncrono

En la siguiente figura se muestra el diagrama de bloques del transmisor de la USART. El corazón de este módulo es el registro de corrimiento (transmit shift register, TSR). La única manera de acceder al registro TSR es a través del registro **TXREG (19H)**.

Para transmitir un dato, el programa deberá ponerlo primero en el registro TXREG. En cuanto el TSR termina de enviar el dato que tenía (en cuanto transmite el bit de paro) lee el dato contenido en TXREG (si hay alguno) esto ocurre en un ciclo T_{CY} . En cuanto el dato de TXREG es transferido al TSR el TXREG queda vacío esta condición es indicada mediante el bit bandera TXIF (que es el bit 4 del registro **PIR1 (0Ch)**), el cual se pone en alto. Este bit NO puede ser limpiado por software, sólo dura un instante en bajo cuando se escribe un nuevo dato a TXREG. Si se escribe un dato seguido de otro (back to back) a TXREG el primero se transfiere inmediatamente a TSR y el otro tiene que esperar hasta que el TSR termine de enviar el bit de Stop del primero. Durante esta espera TXIF permanece en bajo.

Existe otro bit, llamado TRMT (TXSTA<1>), el cual muestra el estado del TSR. TRMT se pone en alto cuando TSR está vacío, y en bajo cuando TSR está transmitiendo un dato. Mientras que TXIF puede generar una interrupción TRMT no lo puede hacer, TRMT está pensado para ser consultado por "poleo" (sin usar interrupciones).

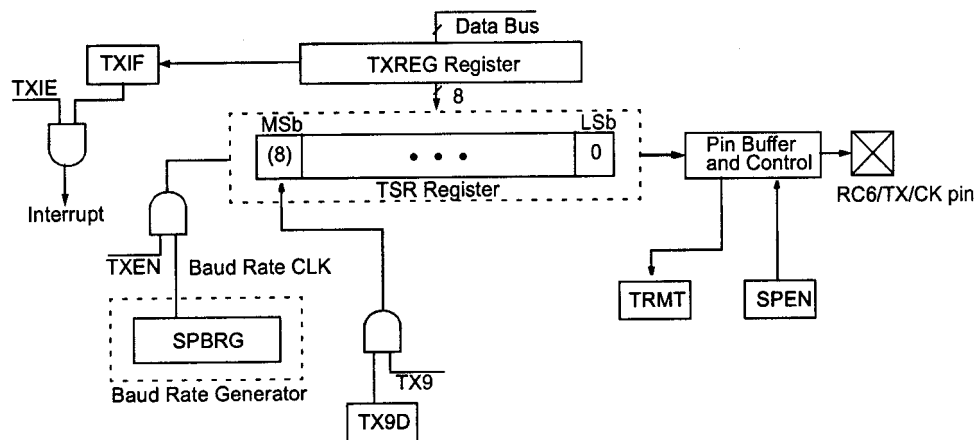


Diagrama de bloques del transmisor de la USART

Para habilitar el módulo de transmisión es necesario poner en alto el bit TXEN (TXSTA<5>), mientras no se habilite el módulo, la patita de transmisión (RC6/TX/CK) se mantiene en alta impedancia. Si TXEN es deshabilitada a la mitad de una transmisión, ésta será abortada y el transmisor será reseteado.

Si se está usando un noveno bit TX9 (TXSTA<6>), éste deberá ser escrito antes de escribir los 8 bits restantes a TXREG, ya que en cuanto se escribe un dato a este registro inmediatamente es transferido a TSR (si éste está vacío).

De acuerdo a lo anterior, la **inicialización del módulo de transmisión** consiste en los siguientes pasos:

1. Inicializar baud rate escribiendo al registro SPBRG el divisor adecuado y opcionalmente al bit BRGH.
2. Habilitar comunicación asíncrona limpiando el bit SYNC y poniendo el bit SPEN.
3. Si se van a usar interrupciones, poner el bit TXIE (PIE<4>).
4. Poner el bit TX9 si se desea transmitir datos de 9 bits.
5. Habilitar transmisión poniendo el bit TXEN, lo cual pondrá el bit TXIF.

6. Colocar el noveno bit del dato en TX9D si se están usando datos de 9 bits.
7. Cargar el dato al registro TXREG (inicia la transmisión).

Ejemplo 10.- Transmisión asíncrona.

El siguiente programa envía de manera asíncrona través de la USART una cadena de caracteres. Esta cadena puede ser recibida mediante el puerto serie RS232 de una PC usando un software de comunicación tal como la hipertextual de windows y un **cable de comunicación serie uno a uno** (es decir, un cable sin intercambio interno de líneas).

```

;* Este programa envía repetidamente una cadena de caracteres a través
;* del puerto serie asíncrono USART, La cadena utiliza como terminador
;* un carácter "$". Se supone un oscilador a cristal Fosc=14.7456 Mhz
;*****
    Include "p16f877.inc"
apun EQU 0x20
dato EQU 0x21
org 0x0000
trans BSF STATUS,RP0      ;banco 1
      BCF TXSTA,BRGH      ;pone bit BRGH=0 (velocidad baja)
      MOVLW 0x17          ;valor para 9600 Bauds (Fosc=14.7456 Mhz)
      MOVWF SPBRG         ;configura 9600 Bauds
      BCF TXSTA,SYNC      ;limpia bit SYNC (modo asíncrono)
      BSF TXSTA,TXEN      ;pone bit TXEN=1 (habilita transmisión)
      BCF STATUS,RP0      ;regresa al banco 0
      BSF RCSTA,SPEN      ;pone bit SPEN=1 (habilita puerto serie)
rep   CLRf apun           ;inicializa apuntador
cic2  CALL letrado        ;obtiene el siguiente carácter apuntado
      MOVWF dato          ;lo guarda en dato
      SUBLW "$"           ;Compara con el signo "$"
      BTFSC STATUS,Z      ;
      GOTO rep            ;si es, reinicia
      CALL envia          ;si no es "$" envía el dato
      INCF apun,1         ;apunta al siguiente carácter
      GOTO cic2           ;repite
;*****
;Subrutina para enviar un dato por el puerto serie
;*****
envia BSF STATUS,RP0      ;banco 1
esp   BTFSS TXSTA,TRMT    ;checa si el buffer de transmisión
      GOTO esp            ;si está ocupado espera
      BCF STATUS,RP0      ;regresa al banco 0
      MOVF dato,W         ;rescata dato a enviar
      MOVWF TXREG         ;lo envía
      RETURN
letrado:
      MOVF apun,W ;carga apuntador en W
      ADDWF PCL,1 ;Salta W instrucciones adelante
      DT "HOLA MUNDO 14.756 Mhz",0x0D,0x0A,"$"
      end

```

3.2.3.3.- El receptor asíncrono

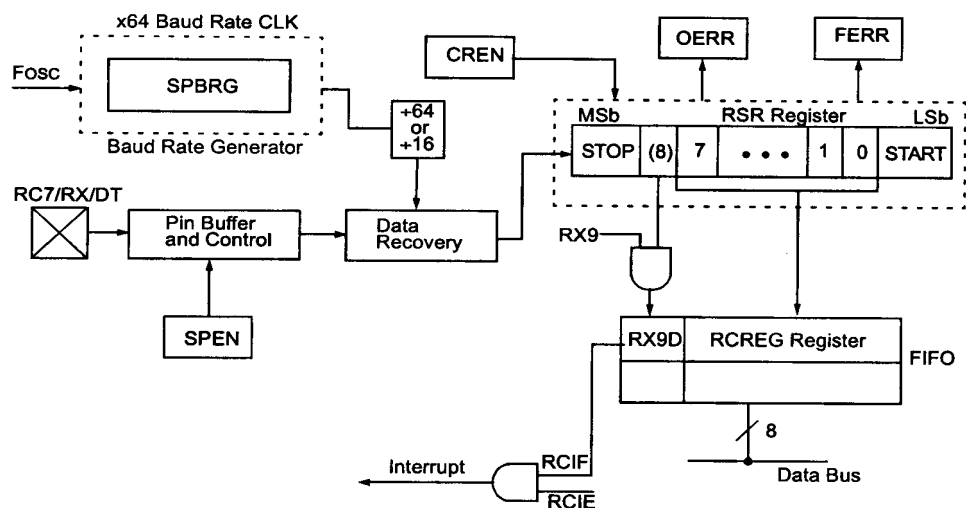
El módulo de recepción es similar al de transmisión, en la siguiente figura se muestran los bloques que lo constituyen. Una vez que se ha seleccionado el modo asíncrono, la recepción se habilita poniendo en alto el bit **CREN** (RCSTA<4>)

El dato es recibido mediante la línea RC7/RX/DT, la cual maneja un registro de corrimiento de alta velocidad (16 veces el Baud rate).

El corazón del receptor es el registro de corrimiento RSR. Este registro no es accesable por software, pero, cuando el dato recibido se ha completado (se ha recibido el bit de Stop) el dato de RSR es transferido automáticamente al registro **RCREG (1Ah)** si éste está vacío y al mismo tiempo es puesto en alto la bandera de recepción RCIF (PIR1<5>). La única manera de limpiar la bandera RCIF es leyendo los datos del registro RCREG. El registro RCREG puede contener hasta dos datos, ya que es un buffer doble que funciona como un cola de dos posiciones.

Si las dos posiciones del registro RCREG están llenas (no han sido leídas) y se detecta el bit de Stop de un tercer dato de recepción, lo cual ocasiona una transferencia automática del dato recibido a RCREG, esto destruirá el primer dato recibido y activará el indicador de sobreescritura OERR (RCSTA<1>). Para evitar esto, se deberán leer los dos datos en RSREG haciendo dos lecturas consecutivas.

La única manera de limpiar el bit OERR una vez que ha sido activado es reseteando el módulo de recepción (limpiando CREN y volviéndolo a poner), si no se limpia OERR se bloquea la transferencia de datos de RSR a RCREG y no puede haber más recepción de datos.



Si se detecta un bit nivel bajo en la posición del bit de stop se pone el indicador de error de encuadre (frame error) FERR RCSTA<2>. Tanto este indicador como el noveno bit RX9D de los datos están en una cola de dos posiciones al igual que los datos recibidos, de manera que al leer RCREG se actualizan FERR y RX9D con nuevos valores, por lo cual éstos bits deberán ser leídos antes de leer RCREG para no perder su información.

De acuerdo a lo anterior, la **inicialización del módulo de recepción** es como sigue:

1. Inicializar el baud rate escribiendo al registro SPBRG el divisor adecuado y opcionalmente al bit BRGH .
2. Habilitar el puerto serie asíncrono limpiando el bit SYNC y poniendo el bit SPEN.
3. Si se van a usar interrupciones, poner el bit RCIE (PIE<5>).

4. Si se desea recepción de datos de 9 bits se deberá poner el bit RX9 (RCSTA<0>).
5. Habilitar la recepción poniendo el bit CREN (RCSTA<4>)
6. El bit RCIF se pondrá cuando la recepción de un dato se complete y se generará una interrupción si RCIE está puesto.
7. Leer el registro RCSTA para obtener el noveno bit (si se están recibiendo datos de 9 bits) o para determinar si ha ocurrido un error de recepción.
8. Leer los 8 bits del dato recibido leyendo el registro RCREG.
9. Si ocurrió algún error este se limpia al limpiar el bit CREN, el cual deberá volver a ponerse si se desea continuar la recepción.

Para ilustrar este procedimiento se presenta a continuación un ejemplo de transmisión de datos en modo asíncrono a través de la USART.

Ejemplo 11.- Recepción asíncrona.

El siguiente programa recibe datos de manera asíncrona a través de la USART. Los datos recibidos son interpretados por el programa como cadenas de caracteres con un carácter de terminación retorno de carro <CR> (elegido arbitrariamente) cuyo código ASCII es un 0Dh.

Los datos pueden ser enviados mediante el puerto serie RS232 de una PC usando un software de comunicación tal como la hiperterminal de windows y un cable de comunicación serie uno a uno.

El programa recibe la cadena de caracteres y la compara con la palabra clave "enciende" (también elegida arbitrariamente), de manera que solamente cuando la cadena recibida coincide con la palabra clave encenderá un Led conectado a la línea RC0 del puerto C. De lo contrario, (cuando reciba cualquier otra cadena) apagará el LED.

- **Observación:** En realidad, por el diseño del programa, cuando se reciba alguna subcadena inicial de la palabra clave "enciende" (incluyendo la subcadena vacía) el Led no cambia de estado.

```

;* Este programa recibe datos a través del puerto serie asíncrono USART
;* La cadena de caracteres recibidos deberá terminar con un carácter <CR>
;* Si la cadena recibida es "enciende" se encenderá un led conectado a RC0
;* si no, se apagará. Se supone un oscilador Fosc=14.7456 Mhz
;*****
    Include "p16f877.inc"
apun  EQU 0x20
dato  EQU 0x21
      org 0x0000
trans BSF STATUS,RP0      ;banco 1
      BCF TRISC,0         ;pone RC0 como salida
      BCF TXSTA,BRGH      ;pone bit BRGH=0 (velocidad baja)
      MOVLW 0x17          ;valor para configurar 9600 Bauds
      MOVWF SPBRG         ;configura 9600 Bauds
      BCF TXSTA,SYNC      ;limpia bit SYNC (modo asíncrono)
      BCF STATUS,RP0      ;regresa al banco 0
      BSF RCSTA,SPEN      ;pone bit SPEN=1 (habilita puerto serie)
      BSF RCSTA,CREN      ;Habilita recepción
ciclo CLRF apun           ;inicializa apuntador

```

```

sig    CALL recibe          ;recibe un caracter del puerto serie
        MOVLW 0x0D          ;carga caracter de fin de cadena
        SUBWF dato,W        ;compara
        BTFSC STATUS,Z      ;ya es fin de cadena?
        GOTO longi          ;si es, checa longitud de cadena recibida
        CALL letrero        ;si no es, obtiene un caracter a comparar
        SUBWF dato,W        ;son iguales?
        BTFSS STATUS,Z      ;
        GOTO noes           ;si no son iguales sale del ciclo
        INCF apun,1         ;si son iguales incrementa apuntador
        GOTO sig            ;repite para el siguiente caracter
longi   MOVLW 0x08          ;carga longitud de la palabra clave
        SUBWF apun,W        ;compara con número de caracteres iguales
        BTFSC STATUS,Z      ;
        BSF PORTC,0         ;si coincide Enciende Led
        GOTO ciclo         ;si no reinicia ciclo
noes    CALL recibe          ;recibe un caracter del puerto serie
        MOVLW 0x0D          ;carga caracter de fin de cadena
        SUBWF dato,W        ;compara
        BTFSC STATUS,Z      ;ya es fin de cadena?
        GOTO apaga         ;si es, apaga led
        GOTO noes          ;si no es, repite
apaga   BCF PORTC,0         ;apaga el Led
        GOTO ciclo         ;regresa a esperar nueva cadena
;*****
;subrutina de recepción de un dato del puerto serie
;*****
recibe  BTFSS PIR1,RCIF      ;checa el buffer de recepción
        GOTO recibe        ;si no hay dato listo espera
        MOVF RCREG,W        ;si hay dato, lo lee
        MOVWF dato         ;lo almacena en dato
        RETURN
letrero:
        MOVF apun,W        ;carga apuntador en W
        ADDWF PCL,1        ;Salta W instrucciones adelante
        DT "enciende"
        end

```

Ejemplo 12.- Transmisión / Recepción Simultánea.

El siguiente programa ilustra la capacidad full duplex que posee la USART del PIC que le permite realizar simultáneamente la transmisión y recepción de datos.

La tarea que realiza el programa es muy simple, solamente hace el eco del carácter recibido, es decir, conforme recibe un carácter del puerto serie, lo regresa sin cambio por el mismo puerto. El proceso se detiene cuando el carácter recibido es un <Esc> o código ASCII 1Bh (elegido arbitrariamente)

```

;* Este programa recibe un carácter por el puerto serie asíncrono USART
;* y lo regresa tal cual por el mismo puerto, hasta recibir un <esc>
;* Se supone un oscilador a cristal Fosc=14.7456 Mhz
;*****
        Include "p16f877.inc"
dato    EQU 0x20
        org 0x0000
trans   BSF STATUS,RP0      ;banco 1
        BCF TXSTA,BRGH      ;pone bit BRGH=0 (velocidad baja)
        MOVLW 0x17          ;valor para 9600 Bauds (Fosc=14.7456 Mhz)
        MOVWF SPBRG         ;configura 9600 Bauds
        BCF TXSTA,SYNC      ;limpia bit SYNC (modo asíncrono)
        BSF TXSTA,TXEN      ;pone bit TXEN=1 (habilita transmisión)

```

```

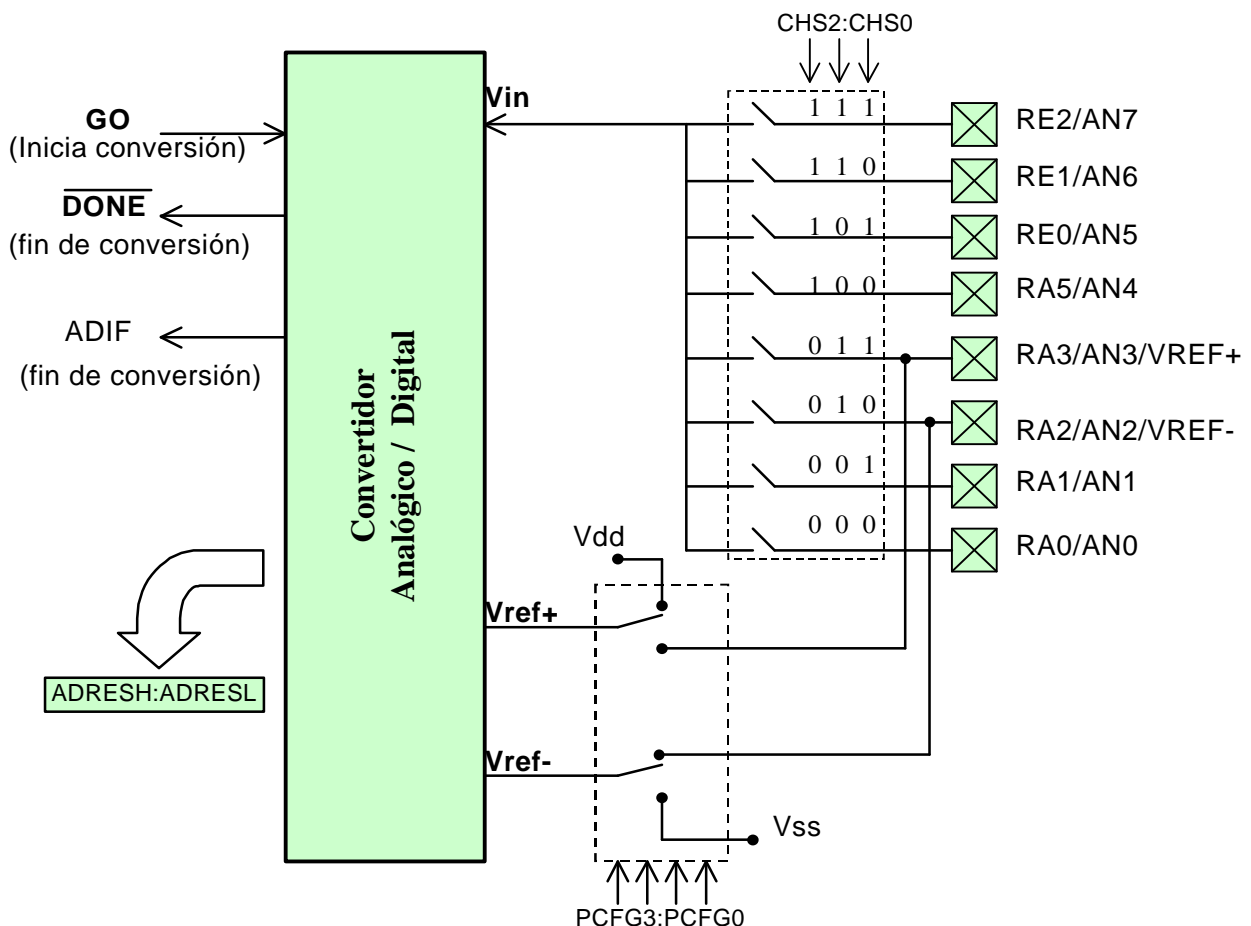
        BCF STATUS,RP0      ;regresa al banco 0
        BSF RCSTA,SPEN      ;pone bit SPEN=1 (habilita puerto serie)
        BSF RCSTA,CREN      ;Habilita recepción
rep     CALL recibe         ;recibe dato
        MOVLW 0x1B          ;carga código ASCII de <escape>
        SUBWF dato,W        ;es igual?
        BTFSC STATUS,Z      ;
        GOTO fin            ;si es igual termina
        CALL envia          ;si n, retransmite el dato
        GOTO rep            ;repite
fin     GOTO fin            ;ciclo infinito
;*****
;Subrutina para enviar un dato por el puerto serie
;*****
envia   BSF STATUS,RP0      ;banco 1
esp     BTFSS TXSTA,TRMT    ;checa si el buffer de transmisión
        GOTO esp            ;si está ocupado espera
        BCF STATUS,RP0      ;regresa al banco 0
        MOVF dato,W         ;rescata dato a enviar
        MOVWF TXREG         ;lo envía
        RETURN
;*****
;subrutina de recepción de un dato del puerto serie
;*****
recibe  BTFSS PIR1,RCIF     ;checa el buffer de recepción
        GOTO recibe         ;si no hay dato listo espera
        MOVF RCREG,W        ;si hay dato, lo lee
        MOVWF dato          ;lo almacena en dato
        RETURN
end

```

3.3.- El Convertidor Analógico / Digital

3.3.1.- Descripción General del Módulo Conversión Analógico Digital (ADC).

Los PIC16F87X poseen un módulo ADC interno que les permite manejar 5 entradas analógicas para los dispositivos de 28 pines y 8 para los otros dispositivos. En la siguiente figura se muestra un diagrama de bloques del módulo ADC.



El multiplexor.- El ADC es un convertidor de aproximaciones sucesivas de 10 bits, el cual puede realizar la conversión de una de las 8 entradas (o canales) analógicas AN0,...,AN7 multiplexadas por la lógica interna que utiliza como líneas de selección del canal los bits CHS2:CHS0, en donde se coloca el número en binario del canal a convertir.

Voltajes de Referencia.- Todo convertidor ADC requiere voltajes de referencia que determinan el valor de mínima escala (V_{REF-}) y el de plena escala (V_{REF+}), de manera que la conversión de un valor de voltaje analógico V_{in} en el rango de V_{REF-} a V_{REF+} producirá un valor equivalente binario D en el rango de 0 a 2^n , Donde n es la resolución del convertidor ($n = 10$).

Como la relación entre escalas es lineal, una regla de tres nos da la relación entre el voltaje analógico de entrada (V_{in}) y el valor digital (D) obtenido por el ADC

$$\frac{D}{2^n - 1} = \frac{V_{in} - V_{REF-}}{V_{REF+} - V_{REF-}}$$

Con la elección más común: $V_{REF+} = V_{DD} = 5v$, $V_{REF-} = V_{SS} = 0v$, y como $n=10$, obtenemos:

$$D = \frac{1023}{5} V_{in} = 204.6 V_{in}$$

De donde se ve que cuando V_{in} varía en todo su rango, desde 0 hasta 5v, el valor obtenido D varía también en todo su rango, de 0 a 1023.

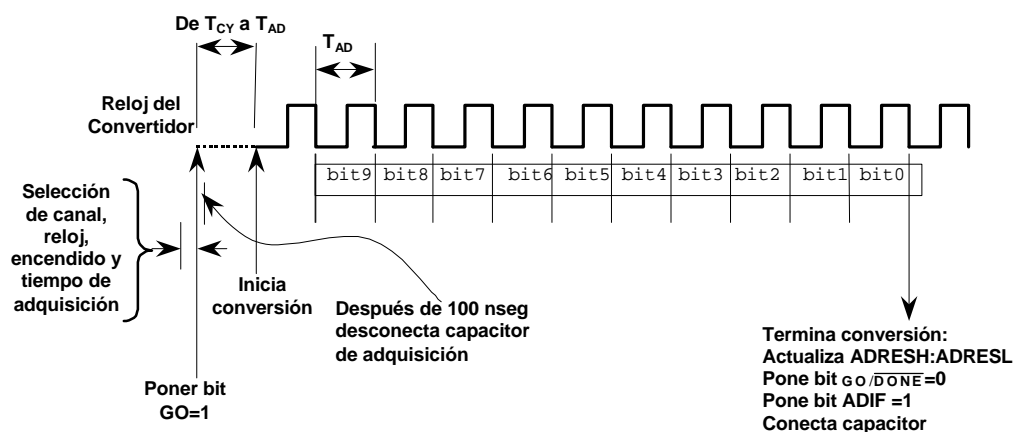
Si a la inversa, obtenemos un valor D y deseamos saber que voltaje representa, basta con despejar:

$$V_{in} = \left(\frac{5}{1023}\right)D = (0.004887585533)D$$

- **Observación:** Como puede verse, la conversión del dato D al voltaje correspondiente requiere una multiplicación por un número fraccionario, para lo cual el PIC no posee instrucciones, si deseamos realizar esta multiplicación en el PIC debemos hacer un programa que multiplique números de punto fijo o de punto flotante.

3.3.2.- El proceso de Conversión Analógico/Digital.

En el siguiente diagrama de tiempo se muestran los eventos que tienen lugar durante el proceso de una conversión analógico / digital.



De acuerdo a la figura, para echar a andar el convertidor se deberán seguir los siguientes pasos:

- 1) Configurar el módulo A/D:

- a. Configurar los pines analógicos y los Voltajes de referencia V_{REF-} y V_{REF+} , mediante el registro **ADCON1 (9Fh)** (y los correspondientes bits TRIS como entradas)
- b. Seleccionar el canal de entrada a convertir mediante los bits CHS2:CHS0 del registro **ADCON0 (1Fh)**
- c. Seleccionar el reloj de conversión mediante los bits ADCS1:ADCS2 (ADCON0<7:6>)
- d. Energizar el convertidor mediante el bit ADON (ADCON0<0>)
- 2) Configurar interrupciones para el convertidor A/D (si se desea), para ello: limpiar ADIF y poner ADIE, PEIE y GIE.
- 3) Esperar mientras transcurre el **tiempo de adquisición** (unos 20 μ seg).
- 4) Iniciar la conversión poniendo el bit **GO/DONE** (ADCON0<2>).
- 5) Esperar a que termine la conversión:
 - a. **Por "poleo" (Polling):** Consultando continuamente el bit **GO/DONE** (el cual es limpiado por el convertidor cuando la conversión está completa).
 - b. **Por interrupciones:** Cuando la conversión termina, la bandera ADIF se activa y esto genera una solicitud de interrupción, la cual deberá ser atendida por una rutina de atención a la interrupción diseñada para ello.
- 6) Leer el dato convertido D de los registros (ADRESH:ADRESL)
- 7) Para la siguiente conversión, esperar al menos $2T_{AD}$ (Donde T_{AD} es el tiempo de conversión por bit).

3.3.3.- Los Registros de Control

A continuación se presenta un resumen de los registros relacionados con la operación del convertidor:

	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
1Fh	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	-	ADON
Bit	7	6	5	4	3	2	1	0

Registro ADCON0 (1Fh)

bits 7-6 **ADCS1:ADCS0.-** Selección de reloj de acuerdo a la siguiente tabla:

ADCS1	ADCS0	Frec. seleccionada
0	0	$F_{OSC}/2$
0	1	$F_{OSC}/8$
1	0	$F_{OSC}/32$
1	1	F_{RC} (oscilador RC interno)*

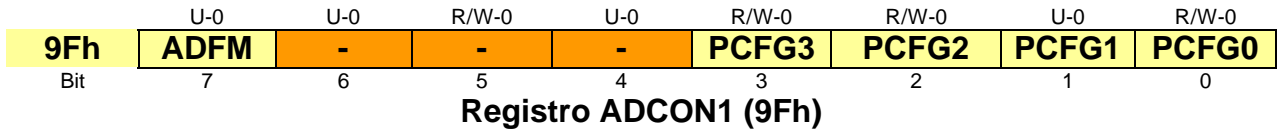
* El oscilador RC interno del convertidor tiene un T_{AD} típico de 4 μ seg, sin embargo, puede variar entre 2 y 6 μ seg.. Este reloj es recomendable para operación en modo SLEEP, ya que este modo desconecta la frecuencia del reloj externo.

* **Precaución:** El convertidor A/D **no** trabajará correctamente con un T_{AD} menor que $T_{AD(minimo)} = 1.6\mu$ seg. El usuario deberá cuidar la elección del reloj adecuado para no violar esta limitante.

bits 5-3 **CHS2:CHS0.-** Selección de canal analógico a convertir. Se selecciona uno de los ocho canales AN0,...,AN7 colocando en estos tres bits el número

binario correspondiente al canal deseado. (Los canales analógicos a usar deberán tener sus bits TRIS correspondientes seleccionados como entradas).

- bit 2 **GO/DONE**.- Bit de inicio y fin de conversión.- Con el convertidor encendido, poniendo este bit en 1 se inicia la conversión del canal seleccionado. Este bit permanece en 1 durante la conversión y es limpiado automáticamente por el convertidor al terminar la conversión.
- bit 0 **ADON**.-Encendido del convertidor. Al poner este bit en 1 el convertidor se enciende y al ponerlo en 0 se apaga y no consume corriente.



- bit 7 **ADFM**.- Selección de formato del resultado. Al ponerlo en 1 se selecciona resultado de 10 bits justificado a la derecha. Y con un 0 se selecciona justificación a la izquierda. En la siguiente sección se explica con mayor detalle.
- Bits 3-0 **PCFG3:PCFG0**.- Bits de configuración de las entradas del convertidor. Configuran las patitas de entrada del convertidor de acuerdo a la siguiente tabla, (en donde A = Entrada Analógica D = Entrada /Salida digital)

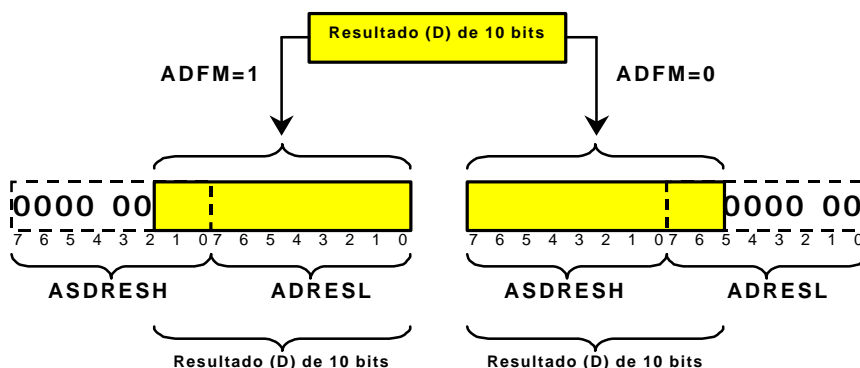
PCFG3: PCFG0	AN7 ⁽¹⁾ /RE2	AN6 ⁽¹⁾ /RE1	AN5 ⁽¹⁾ /RE0	AN4/ RA5	AN3/ RA3	AN2/ RA2	AN1/ RA1	AN0/ RA0	V _{REF+}	V _{REF-}	Can ⁽²⁾ /Refs
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	V _{REF+}	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	V _{REF+}	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	V _{REF+}	D	A	A	VDD	VSS	2/1
011x	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	V _{REF+}	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2	4/2
1100	D	D	D	A	V _{REF+}	V _{REF-}	A	A	RA3	RA2	3/2
1101	D	D	D	D	V _{REF+}	V _{REF-}	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	V _{REF+}	V _{REF-}	D	A	RA3	RA2	1/2

*Notas:(1) Estos tres canales no existen en los PIC16F873 / 76

(2) Esta columna indica el número de canales analógicos de entrada disponibles y el número de canales analógicos usados como entradas de voltaje de referencia.

3.3.4.- Los Registros de Resultados.

El par de registros **ADRESH:ADRESL (1Eh:9Eh)** son cargados con el dato (D) resultante de una conversión analógico / digital al terminar ésta. Cada uno de estos registros es de 8 bits, por lo tanto, juntos pueden guardar hasta 16 bits. Sin embargo, como el resultado D es de 10 bits, el módulo de conversión A/D permite justificarlo (alinearlo) en la parte izquierda o derecha de los 16 bits disponibles, para elegir alguna de las dos opciones se usa el bit ADFM ($ADCON1<7>$) como se muestra en la figura siguiente



Características Eléctricas del convertidor.- Las siguientes son algunas de las especificaciones más importantes, y son válidas para los PIC16F87X-04, PIC16F87X-10, PIC16F87X-20, PIC16LF87X-04:

Característica	mínimo	típico	máximo
$V_{REF+}-V_{REF-}$	2v	-	$V_{DD}+0.3v$
V_{REF+}	$V_{DD}-2.5v$	-	$V_{DD}+0.3v$
V_{REF-}	$V_{SS}-0.3v$	-	$V_{REF+}-2v$
Voltaje analógico V_{AIN}	$V_{SS}-0.3v$	-	$V_{REF+}+0.3v$
Impedancia de la fuente de señal externa Z_{AIN}	-	-	10 K Ω
Corriente promedio consumida por el convertidor I_{AD}	Estándar	220 μA	-
	Extendido	90 μA	-

Ejemplo 13.- Adquisición de una señal analógica por “poleo”.

El siguiente programa realiza la conversión repetitiva de una señal analógica conectada al canal AN0. El dato obtenido en cada conversión es convertido a 4 códigos ASCII de los respectivos 4 dígitos hexadecimales equivalentes para poder desplegarlos en la pantalla de una PC que los recibirá a través de su puerto serie RS232.

```
;*****
;* Este programa realiza la conversión de una señal analógica conectada *
;* al canal AN0 y envía a través del puerto serie el resultado de la    *
;* conversión en forma de 4 dígitos hexadecimales.                      *
;*****
;
    Include "p16f877.inc"
cont EQU 0x20
msnib EQU 0x22
lsnib EQU 0x23
org 0x0000
inic CALL initrans      ;inicializa el puerto serie para transmisión
```

```

        BSF STATUS,RP0      ;Banco 1
        CLRF ADCON1         ;configura 8 canales analógicos, VREF+=VDD y VREF-=VSS
        BSF ADCON1,ADFM     ;Elije resultado con justificación a la derecha
        BSF TRISA,0         ;configura como entrada el canal digital RA0
        BCF STATUS,RP0      ;Banco 0
        MOVLW 0x01          ;Selecciona el canal AN0, reloj de conversión Fosc/2
        MOVWF ADCON0        ;y enciende el convertidor
ciclo  CALL pausa           ;espera 30 µseg a que pase el tiempo de adquisición
        BSF ADCON0,GO       ;inicia conversión
espera BTFSC ADCON0,DONE
        GOTO espera         ;Espera a que termine la conversión
        MOVF ADRESH,W       ;Carga en W el Byte alto del resultado
        CALL Envbyte        ;envía el byte por el puerto serie
        BSF STATUS,RP0      ;banco 1
        MOVF ADRESL,W       ;Carga en W el Byte bajo del resultado
        CALL Envbyte        ;envía el byte por el puerto serie
        MOVLW 0x0D          ;carga código de retorno de línea <CR>
        CALL envia          ;lo envía
        MOVLW 0x0A          ;carga código de avance de línea <LF>
        CALL envia          ;lo envía
        GOTO ciclo          ;repite

;*****
; Subrutina que envía el byte en W por el puerto serie, separado
; en los códigos ASCII de sus dos nibbles hexadecimales
;*****
Envbyte:
        MOVWF msnib         ;pone byte en msnib
        MOVWF lsnib         ;y una copia en lsnib
        SWAPF msnib,1       ;intercambia nibbles en lsnib
        MOVLW 0x0F          ;máscara para limpiar el nibble alto
        ANDWF msnib,1       ;limpia parte alta de msnib
        ANDWF lsnib,1       ;limpia parte alta de lsnib
        MOVF msnib,W        ;carga msnib en W
        CALL asc            ;obtiene código ASCII equivalente
        CALL envia          ;lo envía por el puerto serie
        MOVF lsnib,W        ;carga lsnib en W
        CALL asc            ;obtiene código ASCII equivalente
        CALL envia          ;lo envía por el puerto serie
        RETURN
asc     ADDWF PCL,1          ;Calcula el código a retornar
        ;Saltando W instrucciones adelante
        DT "0123456789ABCDEF"

;*****
; Subrutina de pausa de aprox. 30 µseg (con Fosc=14.7456 MHZ)
;*****
pausa  MOVLW 0x23           ;Carga dato para 30 µseg.
        MOVWF cont          ;inicializa contador con el dato
rep    DECFSZ cont,1        ;Decrementa contador y escapa si cero
        GOTO rep            ;si no es cero, repite
esc    RETURN              ;regresa de esta subrutina

;*****
;Subrutina para inicializar el puerto serie USART como transmisor
;a 9600 Bauds, considerando un cristal de reloj de 14.7456 MHZ
;*****
initrans:
        BCF STATUS,RP1
        BSF STATUS,RP0      ;banco 1
        BCF TXSTA,BRGH      ;pone bit BRGH=0 (velocidad baja)
        MOVLW 0x17          ;valor para 9600 Bauds (Fosc=14.7456 Mhz)
        MOVWF SPBRG         ;configura 9600 Bauds
        BCF TXSTA,SYNC      ;limpia bit SYNC (modo asíncrono)
        BSF TXSTA,TXEN      ;pone bit TXEN=1 (habilita transmisión)

```

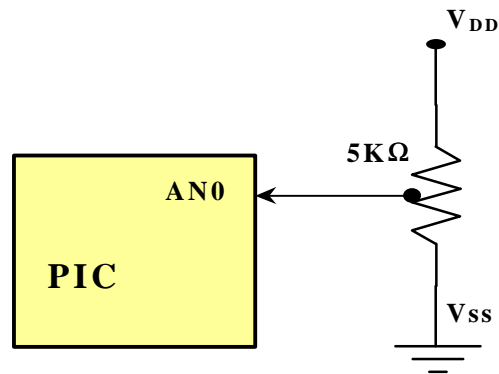
```

    BCF STATUS,RP0      ;regresa al banco 0
    BSF RCSTA,SPEN      ;pone bit SPEN=1 (habilita puerto serie)
    RETURN

;*****
;Subrutina para enviar el byte guardado en W por el puerto serie
;*****
envia BSF STATUS,RP0      ;banco 1
esp   BTFSS TXSTA,TRMT    ;checa si el buffer de transmisión
      GOTO esp            ;si está ocupado espera
      BCF STATUS,RP0      ;regresa al banco 0
      MOVWF TXREG         ;envía dato guardado en W
      RETURN
end

```

- **Observación.** La señal conectada a la línea AN0 deberá estar en el rango de V_{SS} a V_{DD} , para fines de prueba puede ser usado un potenciómetro (de 1 a 10 K Ω) como se muestra en la figura siguiente



3.4.- El Módulo Temporizador

3.4.1.- Descripción General del Módulo Temporizador (Timer)

Los PIC 16F87X poseen un módulo para el manejo preciso y eficiente de operaciones que involucran tiempo o conteo. Este módulo consta de:

- Tres contadores/temporizadores denominados TMR0, TMR1 y TMR2
- Dos módulos CCP (Captura, Comparación y PWM (Modulación de ancho de pulso) denominados CCP1 y CCP2

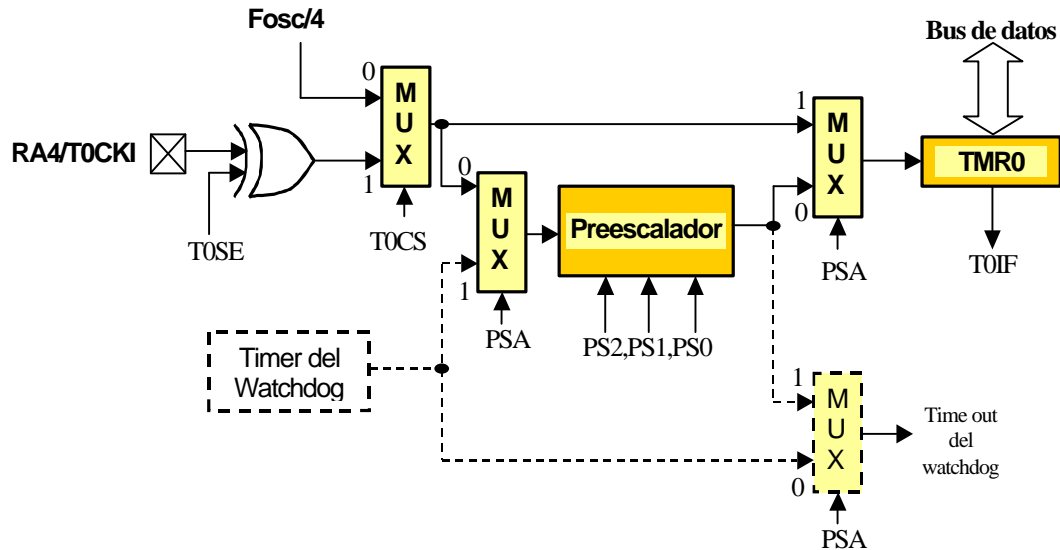
En la siguiente tabla se resumen las principales características de los módulos mencionados:

Módulo	Características
TMR0	<ul style="list-style-type: none"> ✓ TMR0 es un Contador/Temporizador de 8 bits ✓ Leíble y escribible ✓ Reloj interno o externo ✓ Selección de flanco activo en el reloj externo ✓ Preescalador de 8 bits programable ✓ Solicitud de interrupción opcional en el desbordamiento (de FFh a 00h)
TMR1	<ul style="list-style-type: none"> ✓ TMR1 es un Contador/Temporizador de 16 bits ✓ Leíble y escribible ✓ Reloj interno o externo ✓ Solicitud de interrupción opcional en el desbordamiento (de FFFFh a 0000h) ✓ Reinicialización opcional desde los módulos CCP
TMR2	<ul style="list-style-type: none"> ✓ TMR2 es un Contador/Temporizador de 8 bits ✓ Dispone de un registro de periodo de 8 bits (PR2) ✓ Leíble y escribible ✓ Preescalador programable ✓ Postescalador programable ✓ Solicitud de interrupción opcional al coincidir TMR2 y PR2 ✓ Posibilidad de generar impulsos al módulo SSP (puerto serie síncrono)
CCP1 y CCP2	<ul style="list-style-type: none"> ✓ Modo de captura ✓ Modo de comparación ✓ Modo PWM (modulación de ancho de pulso)

3.4.2.- El Módulo del Timer 0.

El Timer 0 es un contador / temporizador de 8 bits. El registro principal de este módulo es **TMR0 (01h, 101h)**. Este registro se incrementa continuamente a una frecuencia seleccionable manejada por un preescalador y el reloj interno $F_{osc}/4$ (**modo temporizador**) o bien, por un preescalador y una señal externa (**modo contador**).

En la siguiente figura se muestra un diagrama de bloques de este módulo, en donde se indican los bits que afectan su operación y la manera en que lo hacen.



3.4.2.1.-El modo Temporizador

En el modo temporizador la señal de reloj que controla el incremento del registro TMR0 es la frecuencia $F_{cy} = F_{osc}/4$, la cual puede ser dividida opcionalmente por el preescalador si así se desea. Como se puede ver en la figura anterior, este modo es seleccionado al limpiar el bit **T0CS (OPTION_REG<5>)**. En este modo, el contenido del registro TMR0 se incrementará a la frecuencia F_{cy} dividida de acuerdo al preescalador, sin embargo, si se realiza una escritura al registro TMR0, su incremento es inhibido por los siguientes dos ciclos de instrucción (T_{cy}).

3.4.2.2.- El modo Contador

En el modo contador, la señal que controla los incrementos del registro TMR0 es una señal externa que proviene de la patita **T0CKI**. En la figura anterior se puede ver que este modo se selecciona poniendo el bit **T0CS** en alto. Se puede seleccionar la transición que provoca los incrementos mediante el bit "Timer0 Source Edge Select" **T0SE (OPTION_REG<4>)**, limpiando este bit se selecciona la transición de subida, mientras que al ponerlo en alto se selecciona la de bajada.

Observación: En este modo, la señal conectada a T0CKI es muestreada durante los ciclos Q2 y Q4 del reloj interno, por ello es necesario que permanezca en alto al menos por 2 T_{osc} más un pequeño retardo de 20nseg y lo mismo en bajo. (es decir, señales demasiado estrechas (rápidas) no podrán ser detectadas).

3.4.2.3.- La Bandera T0IF

El registro TMR0 se incrementa continuamente en cualquiera de sus dos modos, desde 00h hasta FFh y en la siguiente cuenta se reinicia en 00h y así sucesivamente.

Al momento del reinicio se activa la bandera **T0IF (INTCON<2>)** poniéndose en 1. Esta activación puede usarse de dos maneras:

para solicitar una interrupción
para ser consultada por poleo

En ambos casos debe tenerse en cuenta que para poder detectar una activación (un 1) en esta bandera, previamente habrá que limpiarla por software. Esto debe realizarse en la inicialización del Timer y después de que un reciclo la ha activado. Lo último puede hacerse en la rutina de atención a la interrupción, o bien, en la rutina que la consulta por poleo (según sea el caso).

3.4.2.4.- El preescalador

El preescalador es un divisor de frecuencia de módulo seleccionable. Como se puede ver en la figura anterior, el preescalador está compartido entre el timer0 y el módulo watchdog, sin embargo sólo puede conectarse a uno de los dos y esto se establece mediante el bit **PSA (OPTION_REG<3>)**, así, con este bit en alto el preescalador es asignado al reloj del watchdog, mientras que con un nivel bajo en PSA el preescalador dividirá la frecuencia que maneja al timer 0.

La selección del módulo (valor de división de frecuencia) del preescalador se puede realizar mediante los bits **PS2,PS1,PS0 (OPTION_REG<2:0>)** de acuerdo a la siguiente tabla

PS2 PS1 PS0	Divisor (timer 0)	Divisor (Watchdog)
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

Observación: Cuando el preescalador está asignado al timer 0, cualquier escritura al registro TMR0 (cualquier BCF, BSF, MOVWF, CLRF, etc) limpiará el preescalador. En forma similar, cuando está asignado al watchdog, una instrucción CLRWDAT limpiará no solo el timer del watchdog, sino también el preescalador.

A manera de resumen se presenta a continuación una descripción de los bits del registro OPTION_REG que tienen relación con el timer 0:

	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
81h,181h	RBPUR	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit:	7	6	5	4	3	2	1	0

Registro OPTION_REG (81h, 181h)

bits 5 **T0CS.**- Bit de selección de la fuente de reloj para incrementar TMR0. Un 1 en este bit selecciona como reloj la patita T0CKI (modo contador),

mientras que un 0 selecciona el reloj del ciclo de instrucción interno (CLKout) (modo temporizador).

bit 4 **T0SE.**- Bit de selección de transición activa del reloj en modo contador. Un 1 en este bit selecciona el incremento de TMR0 en la transición de alto a bajo de T0CKI, mientras que un 0 selecciona la la transición de bajo a alto.

bit 3 **PSA.**- Bit de asignación del preescalador. Un 1 en este bit asigna el preescalador al watchdog y un 0 lo asigna al Timer0.

bits 2:0 **PS2:PS0.**- Bits de selección del valor del preescaler (ver tabla anterior).

Ejemplo 14.- Manejo del Timer 0 como contador.

El siguiente programa realiza el conteo de el número de veces que produce una transición de bajo a alto en la patita T0CKI. El valor del contador se incrementará una vez por cada dos transiciones y será enviado a través del puerto serie para ser desplegado en la pantalla de una PC, usando las subrutinas mostradas en el ejemplo 13.

```
;*****
;* Este programa realiza el conteo de una señal conectada a la patita T0CKI *
;* el valor del contador lo envía a través del puerto serie para su des-   *
;* pliegue en una PC                                                         *
;*****
    Include "p16f877.inc"
    org 0x0000
inic  CALL initrans                ;inicializa el puerto serie para transmisión
      BCF STATUS,RP0              ;Banco 0
      CLRF TMR0                   ;inicializa la cuenta de TMR0
      BSF STATUS,RP0              ;Banco 1
      MOVLW 0xE0                  ;dato de configuración para el timer0
      MOVWF OPTION_REG            ;configura modo contador, transición positiva,
                                  ;preescalador 1/2 asignado a timer0
      BCF STATUS,RP0              ;Banco 0
ciclo MOVF TMR0,W                  ;lee cuenta actual
      CALL Envbyte                ;envía el valor por el puerto serie
      MOVLW 0x0D                  ;carga carácter <CR>
      CALL envia                  ;lo envía
      MOVLW 0x0A                  ;carga carácter <LF>
      CALL envia                  ;lo envía
      GOTO ciclo                  ;repite
;*****
; Subrutina que envía el byte en W por el puerto serie, separado
; en los códigos ASCII de sus dos nibbles hexadecimales
;*****
msnib EQU 0x22
lsnib EQU 0x23
Envbyte:
      MOVWF msnib                ;pone byte en msnib
      MOVWF lsnib                ;y una copia en lsnib
      SWAPF msnib,1              ;intercambia nibbles en lsnib
      MOVLW 0x0F                  ;máscara para limpiar el nibble alto
      ANDWF msnib,1              ;limpia parte alta de msnib
      ANDWF lsnib,1              ;limpia parte alta de lsnib
      MOVF msnib,W                ;carga msnib en W
      CALL asc                    ;obtiene código ASCII equivalente
```

```

CALL envia          ;lo envía por el puerto serie
MOVF lsnib,W        ;carga lsnib en W
CALL asc            ;obtiene código ASCII equivalente
CALL envia          ;lo envía por el puerto serie
RETURN
asc  ADDWF PCL,1      ;Calcula el código a retornar
                        ;Saltando W instrucciones adelante

DT "0123456789ABCDEF"
;*****
;Subrutina para inicializar el puerto serie USART como transmisor
;a 9600 Bauds, considerando un cristal de reloj de 14.7456 MHZ
;*****
initrans:
    BCF STATUS,RP1
    BSF STATUS,RP0    ;banco 1
    BCF TXSTA,BRGH    ;pone bit BRGH=0 (velocidad baja)
    MOVLW 0x17        ;valor para 9600 Bauds (Fosc=14.7456 Mhz)
    MOVWF SPBRG       ;configura 9600 Bauds
    BCF TXSTA,SYNC    ;limpia bit SYNC (modo asíncrono)
    BSF TXSTA,TXEN    ;pone bit TXEN=1 (habilita transmisión)
    BCF STATUS,RP0    ;regresa al banco 0
    BSF RCSTA,SPEN    ;pone bit SPEN=1 (habilita puerto serie)
    RETURN
;*****
;Subrutina para enviar el byte guardado en W por el puerto serie
;*****
envia BSF STATUS,RP0    ;banco 1
esp  BTFSS TXSTA,TRMT   ;checa si el buffer de transmisión
    GOTO esp           ;si está ocupado espera
    BCF STATUS,RP0    ;regresa al banco 0
    MOVWF TXREG       ;envía dato guardado en W
    RETURN
end

```

Observación. Mediante el programa anterior podemos realizar el conteo de pares de “rebotes” provocados por un botón pulsador, basta con conectar la salida de dicho botón a la patita T0CKI. Al hacer lo anterior se verá que por cada pulsación del botón se incrementa la cuenta no de uno en uno, sino en un valor mayor por el efecto de los rebotes. Esto también quiere decir, que si no deseamos que el contador se vea afectado por el rebote de la señal a contar, se deberá incluir un limpiador de rebotes por hardware, externo al PIC.

Ejemplo 15.- Manejo del Timer 0 como temporizador.

El siguiente programa utiliza el timer 0 para realizar una pausa de máxima duración, la cual se intercala en el encendido / apagado de un LED conectado a la patita RC0, es decir, el LED parpadeará a la frecuencia F que se puede calcular como sigue:

$$F = 1/(T_H + T_L)$$

En donde T_H es el tiempo de encendido y T_L es el tiempo de apagado del LED. Como en el ejemplo son iguales, usaremos sólo $T = T_H = T_L$, por lo tanto

$$F = 1/(2T)$$

Para calcular T con una frecuencia de reloj Fosc dada y un valor del preescalador 1/M, para un ciclo de N incrementos del timer 0 tendremos que la duración (Tciclo) del ciclo será

$$T = T_{\text{ciclo}} = N \cdot M \cdot (4/F_{\text{osc}})$$

Así, para una duración máxima M = 256, N = 256 tendremos:

$$T_{\text{MAX}} = 262144/F_{\text{osc}}$$

Para un reloj de 14.7456 Mhz tendremos

$$T_{\text{MAX}} = 17.777... \text{ mseg}$$

Y la frecuencia de parpadeo del LED será F = 28.125 Hertz.

```
;*****
;* Este programa hace parpadear un LED conectado a la patita RC0 *
;* Usa el timer 0 para generar una pausa de 17.777777 mseg de    *
;* duración (supone un cristal de 14.7456 Mhz). La frecuencia de  *
;* parpadeo del LED es de 28.125 Hertz aprox.                    *
;*****
    Include "p16f877.inc"
    org 0x0000
inic  BSF STATUS,RP0      ;Banco1
      BCF TRISC,0         ;patita RC0 como salida
      BCF STATUS,RP0      ;Banco 0
rep   BSF PORTC,0         ;enciende LED
      CALL pausa          ;pausa de 17.77 mseg
      BCF PORTC,0         ;apaga LED
      CALL pausa          ;pausa de 17.77 mseg
      GOTO rep

;* Subrutina de pausa de 17.77 mseg
;*****
N1    EQU 0x00
pausa MOVLW N1            ;número de incrementos del timer
      MOVWF TMR0          ;inicializa la cuenta de TMR0
      BCF INTCON,T0IF      ;limpia bandera de sobreflujo
      BSF STATUS,RP0      ;Banco 1
      MOVLW 0xC7          ;dato de configuración para el timer0
      MOVWF OPTION_REG    ;modo temporizador, preescalador 1/256 asignado a timer0
      BCF STATUS,RP0      ;Banco 0
ciclo BTFSS INTCON,T0IF    ;checa bandera de sobreflujo (cuenta=256)
      GOTO ciclo          ;si no se ha activado, espera
      BCF INTCON,T0IF      ;si ya se activó, la desactiva
      RETURN              ;retorna
end
```

Observación: La rutina de pausa se puede modificar para una duración de N ciclos en general, simplemente definiendo como N1 lo que le falta a N para ser 256, es decir:

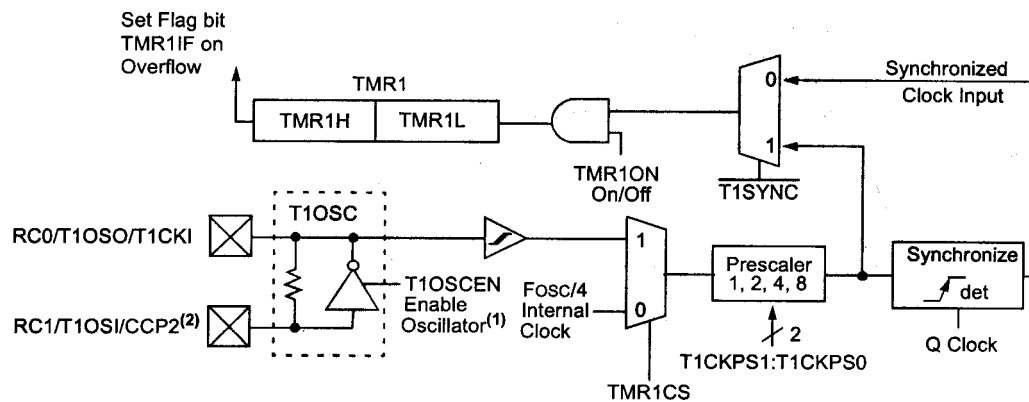
$$N1 = 256 - N$$

3.4.3.- El Módulo del Timer 1.

El Timer 1 a diferencia del Timer 0 es un contador / temporizador de 16 bits. El conteo es realizado por dos registros de 8 bits: (**TMR1H (0Fh)** y **TMR1L (0Eh)**), estos dos registros son tanto leíbles como escribibles. Al par de registros TMR1H:TMR1L los denominaremos por comodidad como si fueran un solo registro de 16 bits (**TMR1**).

Así, el registro TMR1 se incrementa de 0000h a FFFFh y en la siguiente cuenta se reinicia en 0000h y así sucesivamente, al reciclarse se activa (en alto) la bandera **TMR1IF (PIR1<0>)**, la cual puede ser utilizada para generar una interrupción, o bien, para ser consultada por poleo, teniendo las mismas precauciones que ya se explicaron antes para la bandera T0IF.

En la siguiente figura se muestra un diagrama de bloques de este módulo, en donde se indican los bits que afectan su operación y la manera en que lo hacen.



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

3.4.3.1.- Modo temporizador

En este modo el Timer 0 se incrementa (si no se considera preescalador) en cada ciclo de instrucción (a la frecuencia $F_{osc}/4$). Este modo se selecciona limpiando el bit **TMR1CS (T1CON<1>)**.

El preescalador que se puede intercalar entre el reloj $F_{osc}/4$ y el registro TMR1 puede tener sólo uno de 4 valores: 1/1, 1/2, 1/4 y 1/8.

3.4.3.2.- Modo contador

El Timer 1 también puede operar como contador, en este último caso, la entrada a contar se toma de la patita externa RC0/T1OSO/T1CKI. En estos apuntes no se describe este modo.

3.4.3.3.- Otras características

El Timer 1 también posee un bit para habilitación / deshabilitación, este es el bit **TMR1ON (T1CON<0>)** y habilita en alto.

Además, el Timer 1 posee una entrada interna de RESET, el cual puede ser activado por uno cualquiera de los módulos CCP que se describirán más adelante.

A continuación se describe el principal registro relacionado con el Timer 1 y todos sus bits, excepto los que tienen que ver con el modo contador:

	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
10h	-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1 SYNC	TMR1CS	TMR1ON
Bit:	7	6	5	4	3	2	1	0

Registro T1CON (10h)

bits 5:4 **T1CKPS1:T1CKPS0.**- Bits de selección del valor del divisor de frecuencia del preescalador:

1 1 = divisor 1/8
 1 0 = divisor 1/4
 0 1 = divisor 1/2
 0 0 = divisor 1/1

Observación: La cuenta interna del preescalador es limpiada cuando se hace una escritura a cualquiera de los registros TMR1H o TMR1L

bit 1 **TMR1CS.**- Bit de selección de la fuente de reloj
 1 = Modo contador (fuente de reloj: patita RC0/T1OSO/T1CKI)
 0 = Modo Temporizador (fuente de reloj Fosc/4)

bit 0 **TMR1ON.**- Bit de habilitación / deshabilitación del Timer 1:
 1 = habilita Timer 1
 0 = Deshabilita Timer 1

Ejemplo 16.- Manejo del Timer 1 como temporizador.

A continuación se describe un programa similar al ejemplo 16, en el cual se utiliza el Timer 1 para realizar una pausa de máxima duración, la cual se intercala en el encendido / apagado de un LED conectado a la patita RC0, es decir, el LED parpadeará a la frecuencia F que se puede calcular como sigue:

$$F = 1/(T_H + T_L)$$

En donde T_H es el tiempo de encendido y T_L es el tiempo de apagado del LED. Como en el ejemplo son iguales, usaremos sólo $T = T_H = T_L$, por lo tanto

$$F = 1/(2T)$$

Para calcular T con una frecuencia de reloj Fosc dada y un valor del preescalador 1/M, para un ciclo de N incrementos del registro TMR1 tendremos que la duración (Tciclo) del ciclo será

$$T = T_{\text{ciclo}} = N \cdot M \cdot (4/F_{\text{osc}})$$

Así, para una duración máxima M = 8, N = 65536 tendremos:

$$T_{\text{MAX}} = 2,097,152/F_{\text{osc}}$$

Para un reloj de 14.7456 Mhz tendremos

$$T_{\text{MAX}} = 142.222... \text{ mseg}$$

Y por lo tanto la frecuencia de parpadeo del LED será F = 3.515625 Hertz.

```
;*****
;* Este programa hace parpadear un LED conectado a la patita RC0 *
;* Usa el timer 1 para generar una pausa de 142.222.. mseg de      *
;* duración (supone un cristal de 14.7456 Mhz). La frecuencia de   *
;* parpadeo del LED es de 3.515625 Hertz aprox.                  *
;*****
    Include "p16f877.inc"
    org 0x0000
inic  BSF STATUS,RP0      ;Banco1
      BCF TRISC,0         ;patita RC0 como salida
      BCF STATUS,RP0      ;Banco 0
rep   BSF PORTC,0         ;enciende LED
      CALL pausa          ;pausa de 71.11 mseg
      BCF PORTC,0         ;apaga LED
      CALL pausa          ;pausa de 71.11 mseg
      GOTO rep

;* Subrutina de pausa de 71.111 mseg
;*****
N1    EQU 0x00
N0    EQU 0x00
pausa MOVLW N1             ;número de incrementos del timer msb
      MOVWF TMR1H          ;inicializa la cuenta de TMR1
      MOVLW N0             ;número de incrementos del timer lsb
      MOVWF TMR1L          ;inicializa la cuenta de TMR1
      BCF PIR1,TMR1IF      ;limpia bandera de sobreflujo
      MOVLW 0x31           ;dato de configuración para el timer1
      MOVWF T1CON          ;modo temporizador, preescalador 1/8, habilita timer 1
ciclo BTFSS PIR1,TMR1IF    ;checa bandera de sobreflujo (cuenta=65536)
      GOTO ciclo           ;si no se ha activado, espera
      BCF PIR1,TMR1IF      ;si ya se activó, la desactiva
      RETURN              ;retorna
end
```

Observación 1: El programa es idéntico al ejemplo 15, lo único que cambia es la subrutina de pausa, la cual ahora se ha realizado con el timer 1 y no con el Timer 0

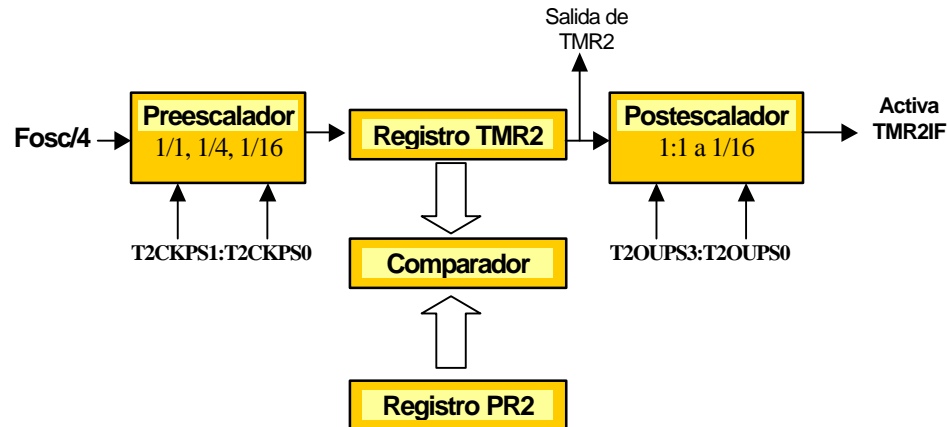
Observación 2: La rutina de pausa se puede adaptar para una duración de N ciclos en general, simplemente definiendo como N1:N0 lo que le falta a N para ser 65536, es decir:

$$N1:N0 = 65536 - N$$

3.4.4.- El Módulo del Timer 2.

El Timer es un temporizador (sin opción de trabajar como contador) de 8 bits. Su registro principal denominado **TMR2 (11h)** es un registro de 8 bits que se incrementa continuamente a la frecuencia seleccionada de $F_{osc}/4$ dividida por un preescalador.

En la siguiente figura se muestra un diagrama de bloques del módulo del Timer2.



3.4.4.1.- El preescalador

La frecuencia que incrementa al registro TMR2 puede ser dividida por un preescalador por un factor de 1/1, 1/4 o 1/16, seleccionable por los bits **T2CKPS1:T2CKPS0 (T2CON<1:0>)**

3.4.4.2.- El Registro de comparación o de Periodo

En operación, el contenido del registro TMR2 se compara continuamente con un registro de periodo denominado **PR2 (92h)** cuyo valor podemos establecer por software. Cada vez que la cuenta de TMR2 es igual a PR2, se reinicia el conteo en TMR2 desde cero, y además se genera una señal de salida, la cual es tratada por un postescalador, para poder generar una señal **TMR2IF (PIR1<1>)** que puede ser usada para solicitar una interrupción, o para ser leída por poleo.

3.4.4.3.- El Postescalador

El postescalador divide la frecuencia con que ocurre una activación de la bandera TMR2IF, es decir, si el valor del postescalador es 1/1, esta bandera se activará cada vez que TMR2 se reinicie, en cambio, si es 1/16 (por ejemplo), TMR2IF se activará cada 16 reinicios de TMR2. En forma similar a los otros dos Timers, esta bandera debe ser limpiada previamente, si se quiere detectar su activación, esto puede ser hecho en la rutina de atención a la interrupción, o bien en la rutina que la detecta por poleo.

El valor de división del postescalador puede establecerse por software mediante los bits **T2OUPS3:T2OUPS0 (T2CON<6:3>)**.

A continuación se describe el principal registro relacionado con el Timer 2 y todos sus bits.

	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
12h	-	T2OUPS3	T2OUPS2	T2OUPS1	T2OUPS0	TMR2ON	T2CKPS1	T2CKPS0
Bit:	7	6	5	4	3	2	1	0

Registro T2CON (12h)

bits 6:3 **T2OUPS3:T2OUPS0.**- Bits de selección del valor del divisor de frecuencia del postescalador, de acuerdo a la siguiente tabla:

0 0 0 0 = divisor 1/1
 0 0 0 1 = divisor 1/2
 0 0 1 0 = divisor 1/3
 ...
 1 1 1 1 = divisor 1/116

Observación: La cuenta interna del postescalador y el preescalador es limpiada cuando ocurre cualquiera de los siguientes eventos: Una escritura a alguno de los registros TMR2 o T2CON o bien, un Reset del sistema de cualquier tipo (POR, MCLR, WDT, o BOR).

bit 2 **TMR2ON.**- Bit de encendido del Timer 2
 1 = Enciende (energiza) el Timer 2
 0 = Apaga (desconecta) el Timer 2

bits 1:0 **T2CKPS1:T2CKPS0.**- Bits de configuración del valor del preescalador de acuerdo a la siguiente tabla:
 0 0 = divisor 1/1
 0 1 = divisor 1/4
 1 x = divisor 1/16

Ejemplo 17.- Manejo del Timer 2 como temporizador.

¿Cuál es la máxima duración de una pausa realizada mediante el Timer 2, usando el mismo esquema de los ejemplos 15 y 16 de dejar pasar el tiempo transcurrido en una sola activación de TMR2IF?.

Solución.

Sea T la duración de la pausa, con una frecuencia de reloj F_{osc} dada, un valor del preescalador $1/M$, y un valor del postescalador $1/P$. Para un ciclo de N incrementos del registro TMR2, es decir, para un valor de N del registro de periodo PR2, tendremos que la duración de la pausa dada por

$$T = N * M * P * (4/F_{osc})$$

Así, para una duración máxima $P = M = 16$, $N = 256$ tendremos:

$$T_{MAX} = 262144/F_{osc}$$

Para un reloj de 14.7456 Mhz tendremos

$$T_{MAX} = 17.777... \text{ mseg}$$

3.5.- Los Módulos de CCP (Captura / Comparación / PWM)

El PIC16F87X posee dos módulos CCP, denominados CCP1 y CCP2. Ambos módulos son prácticamente idénticos con la excepción de la operación del disparo de evento especial. Cada uno de estos dos módulos poseen un registro de 16 bits, el cual puede operar como:

- Registro de captura de 16 bits
- Registro de comparación de 16 bits
- Registro de Ciclo de Trabajo del módulo PWM.

Cada modo de operación requiere como recurso uno de los timers del PIC. En la siguiente tabla se muestran los timers usados por cada modo:

Modo de operación del CCP	Recurso utilizado
Captura	Timer 1
Comparación	Timer 1
PWM	Timer 2

A continuación se da un breve resumen de los registros relacionados con cada módulo:

El Módulo CCP1.

El registro principal de este módulo (**CCPR1**) se compone de dos registros de 8 bits, denominados **CCPR1H (16h)** (parte más significativa) y **CCPR1L (15h)** (parte menos significativa). La operación del módulo se controla mediante el registro **CCP1CON** y el disparo de evento especial, el cual es generado al alcanzarse la igualdad en un registro de comparación reseteará el Timer 1.

El Módulo CCP2.

El registro principal de este módulo (**CCPR2**) se compone de dos registros de 8 bits, denominados **CCPR2H** (parte más significativa) y **CCPR2L** (parte menos significativa). La operación del módulo se controla mediante el registro **CCP2CON** y el disparo de evento especial, el cual es generado al alcanzarse la igualdad en un registro de comparación reseteará el Timer 1 e iniciará una conversión analógico/digital (si el módulo convertidor A/D está habilitado).

Selección del modo de operación

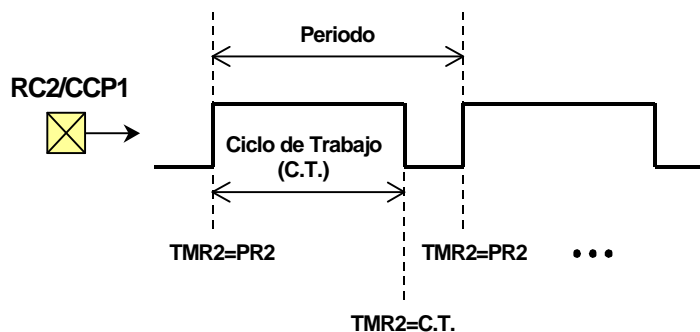
La selección del modo en que trabajara el módulo CCPx se realiza mediante los cuatro bits menos significativos del registro CCPxCON, es decir, mediante los bits CCPxM3:CCPxM0 (CCP<CON<3:0>) de acuerdo a la siguiente tabla

CCPxM3:CCPxM0	Modo seleccionado
0 0 0 0	Captura/Comparación/PWM deshabilitados
0 1 0 0	Captura cada transición de bajada
0 1 0 1	Captura cada transición de subida
0 1 1 0	Captura cada cuarta transición de subida
0 1 1 1	Captura cada 16 transiciones de subida
1 0 0 0	Comparación, pone salida cada coincidencia
1 0 0 1	Comparación, limpia salida cada coincidencia
1 0 1 0	Comparación, genera interrupción cada coincidencia (salida inalterada)
1 0 1 1	Comparación, dispara evento espacial (CCP1 resetea TMR1; CCP2 resetea TMR1 y arranca una conversión A/D).
1 1 x x	Modo PWM

A continuación se describe a detalle cada uno de los modos de operación, comenzando con el modo PWM. La descripción se realiza sólo para el módulo CCP1, ya que es prácticamente igual al CCP2.

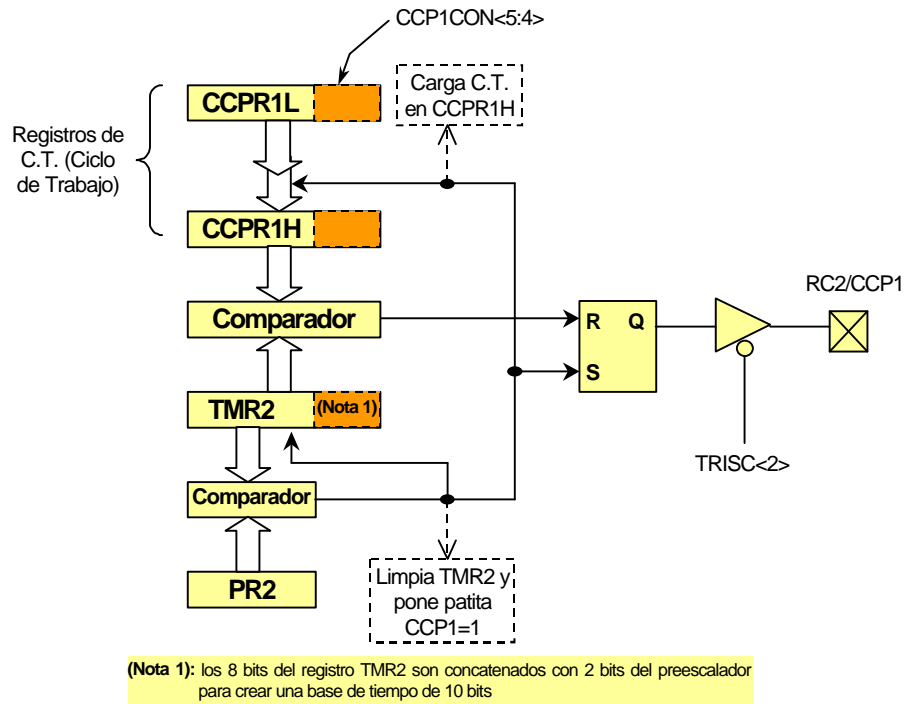
3.5.1.- Modo PWM (Modulación de Ancho de Pulso).

En este modo se puede producir una salida de frecuencia fija seleccionable modulada en ancho de pulso (o ciclo de trabajo) con una resolución de 10 bits, a través de la patita RC2/CCP1, como se muestra en la figura siguiente



Debido a que la patita CCP1 está multiplexada con RC2, este bit del puerto C deberá ser configurado como salida (TRISC<2>=0) para poder usar la salida CCP1.

En la siguiente figura se muestra un diagrama de bloques simplificado que resume la operación básica del PWM.



3.5.1.1.- Control del Periodo del PWM

Para especificar el periodo del PWM se usa el registro PR2, de manera que el valor del periodo será:

$$\text{Periodo}_{\text{PWM}} = (\text{PR2} + 1) * 4 * T_{\text{OSC}} * M$$

Donde 1/M es el valor del preescalador del Timer 2.

Cuando el valor en TMR2 alcanza el valor PR2 los siguientes tres eventos ocurren en el siguiente ciclo (Ver figura anterior):

- El registro TMR2 es limpiado
- La patita CCP1 es puesta en alto (Excepto si el ciclo de Trabajo del PWM vale cero).
- El Ciclo de Trabajo es cargado de **CCPR1L (15h)** a **CCPR1H (16h)**.

De esta manera, de acuerdo a la figura anterior, el siguiente valor de comparación para TMR2 en el comparador de 10 bits es el Ciclo de Trabajo, el cual al alcanzarse limpiará la patita CCP1.

3.5.1.2.- Control del Ciclo de Trabajo del PWM

El ciclo de Trabajo se especifica escribiendo un valor de 10 bits al registro CCPR1L (los 8 bits más significativos (msb)) y los dos bits menos significativos (lsb) a **CCP1CON<5:4>** este valor de 10 bits lo representaremos como

CT=CCPR1L:CCP1CON<5:4>. El valor de tiempo que dura el ciclo de trabajo para un valor del preescalador de 1/M, se calcula de acuerdo a la siguiente ecuación:

$$T_{PWM} = CT * T_{OSC} * M$$

Como se puede ver en la figura anterior, el valor que determina la duración de C.T. del PWM no es el cargado en **CT** (CCPR1L), sino en CCPR1H, el cual sólo se actualiza copiando el valor de **CT** en el momento en que TMR2 alcanza el valor de PR2 (es decir, cada vez que se completa un periodo). Por ello, aunque CCPR1L puede ser escrito en cualquier momento, el Ciclo de Trabajo solo se actualiza hasta que termina el periodo que está en transcurso.

No hay otra manera de escribir al registro CCPR1H, ya que este es un registro de sólo lectura.

Cuando el valor de TMR2 (concatenado con dos bits internos) alcanza el valor de CCPR1H (concatenado con dos bits internos también) la patita CCP1 es limpiada (ver figura anterior).

Como puede verse, el número de divisiones que se pueden tener en un Ciclo de Trabajo será 2^r , donde r es el número de bits usados, por lo tanto su duración máxima será este número de divisiones multiplicada por la duración del ciclo más pequeño del sistema T_{OSC} . Por lo tanto:

$$T_{PWM} = (2^r) * T_{OSC} * M$$

Sin embargo, dependiendo del valor de Ciclo de trabajo máximo (T_{PWM}) deseado, no será posible realizar las 2^r divisiones y por lo tanto no se podrán usar los r bits de resolución. O al revés, si se elige una resolución deseada r no será posible tener cualquier Ciclo de Trabajo máximo (T_{PWM}) Deseado.

⇒ Además, si se elige T_{PWM} mayor que el periodo del PWM (Periodo_{PWM}) la patita CCP1 no será limpiada (y por lo tanto no funcionará el PWM).

Ejemplo: Así, por ejemplo, suponiendo un cristal de 20 Mhz, si deseamos usar la máxima resolución $r = 10$ bits, el ciclo de trabajo máximo posible será (para $M=1$):

$$T_{PWM} = 1024 * 0.05 \times 10^{-6} = 0.0128 \text{ mseg}$$

O bien, la "Frecuencia del PWM" definida como $F_{PWM} = 1 / \text{Periodo}_{PWM}$, tendrá un valor de:

$$F_{PWM} = 1 / 0.0128 \times 10^{-3} = 19.53125 \text{ Khz}$$

O de lo contrario la patita CCP1 no podrá ser limpiada. Para tener este valor de F_{PWM} se requiere un valor en PR2 que como ya se dijo, está dado por

$$\text{Periodo}_{PWM} = (PR2+1) * 4 * T_{OSC} * M$$

Despejando:

$$PR2 = (\text{Periodo}_{\text{PWM}} / (4 * T_{\text{OSC}} * M)) - 1,$$

Sustituyendo, $PR2 = 255 = \text{FFh}$.

En la siguiente tabla se resumen diversas elecciones de resolución r y la correspondiente frecuencia F_{PWM} máxima, así como el valor de PR2 con el que se logra (para un frecuencia del cristal de 20 Mhz).

F_{PWM} máxima	1.22 KHz	4.88 KHz	19.53 KHz	78.12 KHz	156.3 KHz	208.3 KHz
Preescalador	16	4	1	1	1	1
PR2	FFh	FFh	FFh	3Fh	1Fh	17h
Resolución máxima	10	10	10	8	7	5.5

3.5.1.3.- Secuencia de configuración del PWM

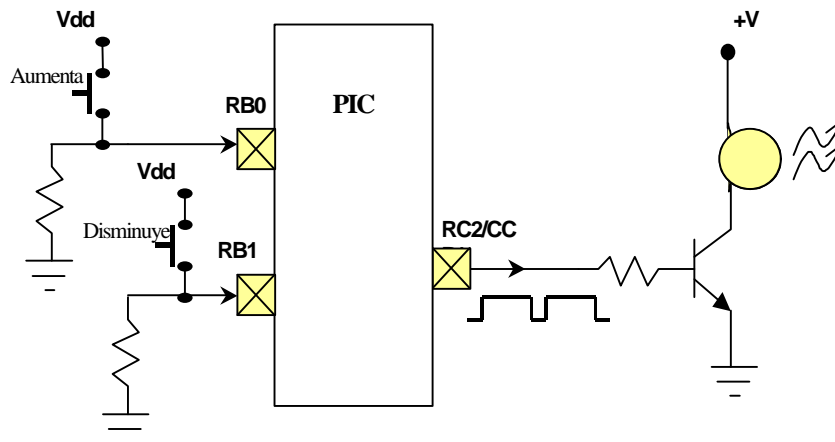
A continuación se resumen los pasos para realizar la configuración inicial del PWM:

1. Establecer el periodo del PWM escribiendo al registro PR2.
2. Establecer el Ciclo de Trabajo del PWM escribiendo al registro CCPR1L y a los bits CCP1CON<5:4>.
3. Configurar como salida la patita CCP1, limpiando el bit TRISC<2>.
4. Configurar el preescalador del Timer 2 y habilitar el Timer 2, escribiendo al registro T2CON.
5. Configurar el módulo CCP1 para operación PWM. Poniendo en alto los bits CCP1CON <2:3>.

A continuación se ilustra este proceso de configuración en el siguiente ejemplo.

Ejemplo 18. Control de iluminación en lazo abierto.

En el siguiente ejemplo se ilustra el uso de la salida PWM para controlar el nivel de iluminación producido por una lámpara de acuerdo a la siguiente figura



Elección de la frecuencia de operación del PWM:

Como la carga que se desea controlar es de tipo resistivo, no tendrá limitaciones respecto a frecuencias altas de operación, salvo las limitaciones de baja frecuencia que le impone la respuesta térmica para que el parpadeo no sea visible, es decir, se puede usar desde una frecuencia lenta (del orden de unos 20 hertz) hasta frecuencias tan altas como el PWM pueda soportar (del orden de los kilohertz).

Para usar los 10 bits de resolución, repetimos el cálculo del ejemplo anterior, pero ahora supondremos una $F_{OSC} = 14.7456 \text{ Mhz}$, es decir, $T_{OSC} = 0.06781684 \text{ } \mu\text{seg}$, entonces, el ciclo de trabajo máximo posible será (para $M=1$):

$$T_{PWM} = 1024 * 0.06781684 \times 10^{-6} = 0.069444 \text{ mseg}$$

O bien, la "Frecuencia del PWM" definida como $F_{PWM} = 1 / \text{Periodo}_{PWM}$, tendrá un valor de:

$$F_{PWM} = 14.4 \text{ Khz}$$

Y para lograr esto, se necesita:

$$PR2 = (\text{Periodo}_{PWM} / (4 * T_{OSC} * M)) - 1,$$

Sustituyendo, $PR2=255 = FFh$.

```
;*****
;* Este programa controla el ciclo de trabajo de la salida PWM *
;* (patita CCP1) con la cual controlará el nivel de iluminación *
;* promedio producido por una lámpara controlada con esta señal *
;* Se usa un botón conectado a RB0 para incrementar el nivel y *
;* otro conectado a RB1 para disminuirlo. *
;* Se supone un cristal de 14.7456 Mhz *
;*****
    Include "p16f877.inc"
cont EQU 0x20
cont1 EQU 0x21
CTH EQU 0x22
CTL EQU 0x23
    org 0x0000
inic BSF STATUS,RP0      ;Banco1
    BSF TRISB,0          ;Configura RB0 como entrada
    BSF TRISB,1          ;Configura RB1 como entrada
    MOVLW 0xFF           ;carga periodo
    MOVWF PR2            ;lo establece para el PWM
    BCF TRISC,2          ;patita RC2/CCP1 como salida
    BCF STATUS,RP0      ;Banco 0
    CLRF CCP1L           ;inicializa Ciclo de trabajo en cero
    BCF CCP1CON,CCP1X
    BCF CCP1CON,CCP1Y
    MOVLW 0x04           ;configura Timer 2
    MOVWF T2CON          ;preescalador 1/1, habilita Timer 2
    BSF CCP1CON,CCP1M3   ;Configura el modulo CCP1 para operación PWM
    BSF CCP1CON,CCP1M2
;en este punto la salida PWM tiene un Ciclo de trabajo CT=0
    CLRF CTL            ;inicializa CT de 10 bits en cero
    CLRF CTH
```

```

Esp0  BTFSC PORTB,0      ;Checa Botón RB0
      CALL incre         ;si está presionado incrementa CT
Esp1  BTFSC PORTB,1      ;si no se ha presionado Checa botón RB1
      CALL decre         ;si está presionado Decrementa CT
      MOVF CTL,W         ;si no se ha presionado obtiene copia de CT parte baja
      MOVWF CCP1L        ;actualiza parte baja del CT real
;**** a continuación actualiza la parte alta del CT real
      MOVLW 0xCF         ;máscara
      ANDWF CCP1CON,1     ;limpia los dos msbits del CT real
      MOVLW 0x03         ;máscara
      ANDWF CTH,1        ;limpia los 6 bits altos en CTH
      SWAPF CTH,W        ;copia los 2 bits bajos de CTH en el nibble alto de W
      IORWF CCP1CON,1     ;pone bits que deben ser 1 en los dos msb del CT real
;**** con esto queda actualizada la parte alta del CT real
      CALL pau           ;pausa para moderar la velocidad de incremento/decremento
      GOTO Esp0          ;repite
;*****
incre  INCF CTL,1         ;incrementa parte baja de la copia de CT
      BTFSS STATUS,Z     ;checa si se recicló a cero
      RETURN            ;si no, retorna
      INCF CTH,1         ;si se hizo cero incrementa parte alta de CT
      RETURN
decre  DECF CTL,1         ;decrementa parte baja de la copia de CT
      COMF CTL,W         ;copia negado de CTL a W (para ver si CTL=0xFF)
      BTFSS STATUS,Z     ;checa si W es cero
      RETURN            ;si no, retorna
      DECF CTH,1        ;si sí, Decrementa parte alta de CT
      RETURN
;** pausa de 50 miliseg. aproximadamente
pau    CLRF cont1
      CLRF cont
p1     DECFSZ cont,1
      GOTO p1
      DECFSZ cont1,1
      GOTO p1
      RETURN
end

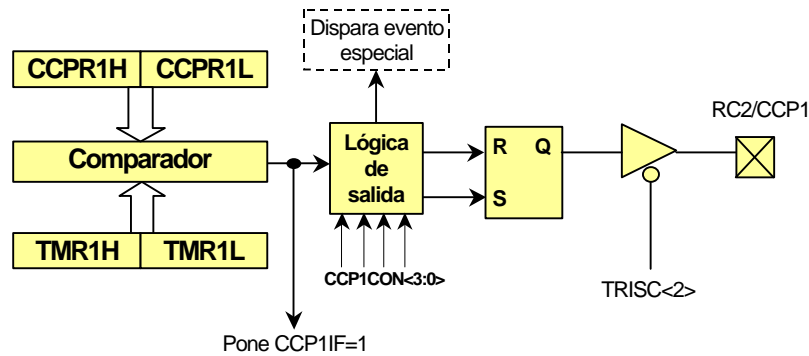
```

3.5.2.- El Modo Comparador

En el modo de comparación el registro de 16 bits **CCPR1** (CCPR1H:CCPR1L) se compara constantemente con el valor del registro de 16 bits **TMR1**. De manera que cuando sus valores coinciden además de activarse la bandera para solicitar interrupción CCP1IF (PIR1<2>), puede ocurrir en la patita RC2/CCP1 (previa configuración) alguna de las siguientes acciones:

- RC2/CCP1 Se pone en alto
- RC2/CCP1 Se pone en Bajo
- RC2/CCP1 no cambia

La acción que ocurra en esta patita se configura mediante los bits de control **CCP1M3:CCP1M0 (CCP1CON<3:0>)**. En la figura siguiente se muestra un diagrama de bloques en donde se ilustra la manera en que trabaja el módulo CCP en modo comparador



Configuración del modo de comparación

A continuación se hace un resumen de algunas consideraciones que se deberán hacer para configurar adecuadamente el modo de comparación:

- la patita RC2/CCP1 deberá configurarse como salida limpiando el bit TRISC<2>.
- Al limpiar el registro CCP1CON el latch de salida de la patita RC2/CCP1 se fuerza a su valor "default" de cero.
- El Timer 1 debe estar corriendo en modo temporizador (o en modo contador sincronizado)
- Si se está manejando por poleo la bandera de solicitud de interrupción, se deberá limpiar por software antes de un posible evento que la active, de lo contrario no se notará la activación.
- El manejo de evento especial no se describe en estos apuntes.

Ejemplo 19.- Generador de Frecuencia Variable.

En este programa se hace uso del modo de comparación para realizar la conmutación de una señal cada vez que transcurre un tiempo, el cual se ajusta al oprimir un botón de incremento o uno de decremento. El hardware utilizado es similar al del ejemplo anterior, sólo que la salida en lugar de manejar una lámpara, se puede simplemente monitorear mediante un zumbador piezoeléctrico, o visualizarla en un osciloscopio.

```
;*****
;* Este programa genera a través de la patita RC0, una señal *
;* oscilatoria. Se usa un botón conectado a RB0 para incremen-*
;* tar la frecuencia y otro conectado a RB1 para disminuirla. *
;* Se supone un cristal de 14.7456 Mhz *
;*****
Include "p16f877.inc"
org 0x0000
inic BSF STATUS,RP0      ;Banco1
      BSF TRISB,0        ;Configura RB0 como entrada
      BSF TRISB,1        ;Configura RB1 como entrada
      BCF TRISC,2        ;patita RC2/CCP1 como salida
      BCF STATUS,RP0     ;Banco 0
      MOVLW 0x01
      MOVWF T1CON        ;Configura Timer1 modo temporizador, preesc 1/1
      CLRF TMR1H         ;Inicializa en cero el timer 1
      CLRF TMR1L
      CLRF CCPR1H        ;inicializa periodo de comparación
```

```

        CLRF CCPR1L           ;al mínimo (cero)
        CLRF CCP1CON          ;limpia latch de CCP1
        BSF CCP1CON,CCP1M3    ;Habilita modulo CCP1 para modo de comparación
        BCF CCP1CON,CCP1M0    ;Poner salida al coincidir
        BCF PIR1,CCP1IF       ;limpia bandera de interrupcion.
checa   BTFSS PIR1,CCP1IF     ;checa bandera
        GOTO checa            ;si no se ha activado espera
        BCF PIR1,CCP1IF       ;si ya se activó, la limpia
        MOVLW 0x01            ;máscara
        XORWF CCP1CON,1       ;conmuta la acción al coincidir próxima comparación.
        CLRF TMR1L            ;limpia la cuenta del timer 1
        CLRF TMR1H
        BTFSC PORTB,0         ;Checa Botón RB0
        CALL decre             ;si está presionado decrementa periodo
        BTFSC PORTB,1         ;si no se ha presionado Checa botón RB1
        CALL incre             ;si está presionado incrementa periodo
        GOTO checa            ;repite
;*****
incre   INCF CCPR1L,1          ;incrementa parte baja del periodo
        BTFSS STATUS,Z        ;checa si se recicló a cero
        RETURN                ;si no, retorna
        INCF CCPR1H,1          ;si llegó a cero incrementa parte alta del periodo
        RETURN
decre   DECF CCPR1L,1          ;Decrementa parte baja del periodo
        COMF CCPR1L,W          ;copia negado de CCPR1L a W (para ver si es=0xFF)
        BTFSS STATUS,Z        ;checa si W es cero
        RETURN                ;si no, retorna
        DECF CCPR1H,1          ;si sí, Decrementa parte alta del periodo
        RETURN
end

```

3.5.3.- El Modo de Captura

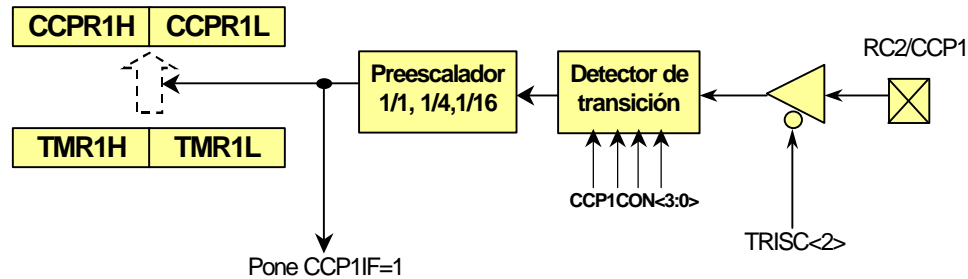
En el modo de captura el registro **CCPR1** (CCPR1H:CCPR1L) captura el valor de 16 bits registro **TMR1** cuando ocurre un evento en la patita RC2/CCP1. El evento en cuestión puede especificarse previamente como alguno de los siguientes:

- Cada transición de bajada
- Cada transición de subida
- Cada cuarta transición de subida
- Cada dieciseisava transición de subida

Además de que el valor de TMR1 es capturado, la bandera de solicitud de interrupción CCP1IF es activada, la cual deberá ser limpiada por software para poder detectarla si se está consultando por poleo.

El tipo de acción que se desea detectar en esta patita se configura mediante los bits de control **CCP1M3:CCP1M0 (CCP1CON<3:0>)**. Si ocurre otro evento de captura antes de que haya sido leído el registro CCPR1, el valor capturado anterior se perderá, ya que con la nueva captura este registro es reescrito.

En la figura siguiente se muestra un diagrama de bloques en donde se ilustra la manera en que trabaja el módulo CCP en modo de captura



El preescalador del CCP

El valor del preescalador se configura mediante los bits CCP1M3:CCP1M0. Sin embargo, al realizar un cambio en la configuración del preescalador se puede generar una interrupción falsa, para evitar lo anterior se deberá apagar el módulo CCP (limpiando el registro CCP1CON) previamente al cambio de valor del preescalador.

Este preescalador es independiente al preescalador del Timer 1 (el cual puede usarse como ya se explicó con sus posibles divisores de 1/1, 1/2, 1/4, 18).

Configuración del modo de captura

A continuación se hace un resumen de algunas consideraciones que se deberán hacer para configurar adecuadamente el modo de captura:

- En el modo de captura la patita RC2/CCP1 deberá configurarse como entrada poniendo en alto el bit TRISC<2>.
- Si por alguna razón la patita RC2/CCP1 es configurada como salida, se deberá tener en cuenta que una escritura al puerto C puede causar una condición de captura.
- El Timer 1 debe estar corriendo en modo temporizador (o en modo contador sincronizado), de lo contrario el modo de captura puede no funcionar.
- Cuando se realiza un cambio de un modo de captura a otro modo de captura se puede generar una solicitud de interrupción falsa. Esto debe ser evitado limpiando la máscara de interrupción correspondiente (CCP1IE (PIE1<2>)) cuando se realice un cambio de estos para evitar una interrupción falsa.
- Si se está manejando por poleo la bandera de solicitud de interrupción, se deberá limpiar por software antes de un posible evento que la active, de lo contrario no se notará la activación.

Ejemplo 20.- Medición de Periodo

En este programa se hace uso del modo de captura para realizar la medición del periodo de una señal oscilatoria. Para ello se configura el evento de captura para que ocurra cada vez que la patita RC2/CCP1 detecte una subida en la señal oscilatoria de entrada. El valor capturado se envía por el puerto serie para su despliegue


```
;*****
;* Este programa mide el periodo de una señal oscilatoria en la *
;* patita RC2/CCP1. El valor de periodo capturado representa el *
;* número de ciclos Tcy por periodo y se envía continuamente por*
;* el puerto serie. Se supone un cristal de 14.7456 Mhz          *
;*****
    Include "p16f877.inc"
    org 0x0000
msnib EQU 0x20
lsnib EQU 0x21
Inic  CALL initrans      ;inicializa puerto serie como transmisor
      BSF STATUS,RP0     ;Banco1
      BSF TRISC,2        ;patita RC2/CCP1 como entrada
      BCF STATUS,RP0     ;Banco 0
      MOVLW 0x01
      MOVWF T1CON        ;Configura Timer1 modo temporizador, preesc 1/1
      CLRF TMR1H         ;Inicializa en cero el timer 1
      CLRF TMR1L         ;apaga el módulo CCP para inicializar
      CLRF CCP1CON       ;limpia latch de CCP1
      BSF CCP1CON,CCP1M2 ;Habilita modulo CCP1 para modo de captura
      BSF CCP1CON,CCP1M0 ;en transición de subida
      BCF PIR1,CCP1IF    ;limpia bandera de interrupcion.
checa BTFSS PIR1,CCP1IF  ;checa bandera
      GOTO checa         ;si no se ha activado espera
      BCF PIR1,CCP1IF    ;si ya se activó, la limpia
      CLRF TMR1L         ;limpia la cuenta del timer 1
      CLRF TMR1H
      MOVF CCPR1H,W      ;copia periodo capturado
      CALL Envbyte       ;y lo envía por el puerto serie
      MOVF CCPR1L,W
      CALL Envbyte
      MOVLW 0x0D         ;envía separador
      CALL envia
      MOVLW 0x0A
      CALL envia
      GOTO checa        ;repite
;*****
; Subrutina que envía el byte en W por el puerto serie, separado
; en los códigos ASCII de sus dos nibbles hexadecimales
;*****
Envbyte:
    MOVWF msnib          ;pone byte en msnib
    MOVWF lsnib          ;y una copia en lsnib
    SWAPF msnib,1        ;intercambia nibbles en lsnib
    MOVLW 0x0F           ;máscara para limpiar el nibble alto
    ANDWF msnib,1        ;limpia parte alta de msnib
    ANDWF lsnib,1        ;limpia parte alta de lsnib
    MOVF msnib,W         ;carga msnib en W
    CALL asc             ;obtiene código ASCII equivalente
    CALL envia           ;lo envía por el puerto serie
    MOVF lsnib,W         ;carga lsnib en W
    CALL asc             ;obtiene código ASCII equivalente
    CALL envia           ;lo envía por el puerto serie
    RETURN
asc    ADDWF PCL,1        ;Calcula el código a retornar
      ;Saltando W instrucciones adelante
      DT "0123456789ABCDEF"
;*****
;Subrutina para inicializar el puerto serie USART como transmisor
;a 9600 Bauds, considerando un cristal de reloj de 14.7456 MHZ
;*****
initrans:
    BCF STATUS,RP1
    BSF STATUS,RP0      ;banco 1
    BCF TXSTA,BRGH      ;pone bit BRGH=0 (velocidad baja)
```

```

        MOVLW 0x17          ;valor para 9600 Bauds (Fosc=14.7456 Mhz)
        MOVWF SPBRG         ;configura 9600 Bauds
        BCF TXSTA,SYNC      ;limpia bit SYNC (modo asíncrono)
        BSF TXSTA,TXEN      ;pone bit TXEN=1 (habilita transmisión)
        BCF STATUS,RP0      ;regresa al banco 0
        BSF RCSTA,SPEN      ;pone bit SPEN=1 (habilita puerto serie)
        RETURN
;*****
;Subrutina para enviar el byte guardado en W por el puerto serie
;*****
envia   BSF STATUS,RP0      ;banco 1
esp     BTFSS TXSTA,TRMT    ;checa si el buffer de transmisión
        GOTO esp           ;si está ocupado espera
        BCF STATUS,RP0      ;regresa al banco 0
        MOVWF TXREG         ;envía dato guardado en W
        RETURN
end

```

En el ejemplo anterior, el valor del dato (N) de 16 bits desplegado se puede convertir a segundos (T) de acuerdo a la relación

$$T = 4 \cdot N / F_{osc} = (2.712673611 \cdot 10^{-7}) N \text{ seg.}$$

O bien, como frecuencia: $F = 1/T = 3,686,400 / N$ Hertz.

⇒ **Observación.** El programa debería leer sin problemas periodos entre $T_{\text{máx}} = 17.777$ mseg ($F_{\text{min}} = 56.25$ hertz) y un $T_{\text{min}} = 0.271 \mu\text{seg}$ ($F_{\text{máx}} = 3.6864$ Mhz), sin embargo debido al retardo de la rutina de transmisión del dato, (en la realidad el programa no puede detectar ninguna transición de subida durante la transmisión del dato) el programa sólo puede procesar correctamente la transición hasta una frecuencia $F_{\text{máx}} = 160\text{Hz}$ ($T_{\text{min}} = 6.25$ mseg)

3.5.- Manejo de Interrupciones.

Se le llama interrupción a un salto especial a una subrutina que no está contemplado en un punto específico del programa principal, sino que puede ocurrir en cualquier punto de éste y **no es provocado por una instrucción en el programa, sino por un evento** interno o externo al sistema del microcontrolador.

Los dispositivos que manejan eventos capaces de provocar una solicitud de interrupción se denominan fuentes de interrupción. La familia del PIC16F87x cuenta con hasta 14 fuentes de interrupción.

Cada fuente de interrupción posee dos bits asociados a ella:

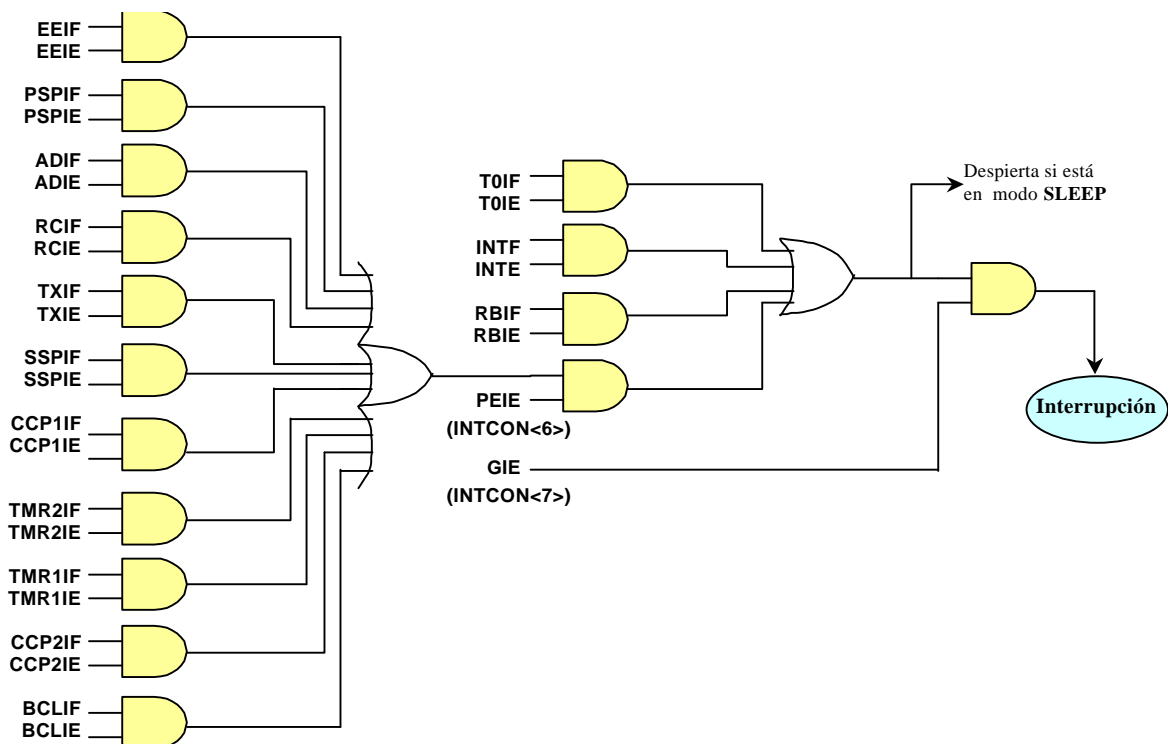
- Una Bandera (terminada en F) de Interrupción, la cual es activada (en alto) por el evento para solicitar una interrupción.
- Una Máscara (terminada en E) Local de Interrupción, la cual si está desactivada (en bajo) bloqueará la solicitud de interrupción correspondiente, pero si está activada (en alto) permitirá la solicitud de Interrupción.

- Además existe una máscara de interrupción global **GIE (INTCON<7>)**, la cual bloqueará todas las solicitudes de interrupción si está desactivada (**GIE=0**).
- Algunas fuentes de interrupción también poseen una segunda máscara de interrupción global denominada **PEIE (INTCON<6>)**. De hecho, actúa sobre todas las fuentes de interrupción, excepto las interrupciones debidas a la patita INT, el sobreflujo del Timer 0 y las interrupciones del puerto B (INTF, T0IF y RBIF).

De acuerdo a lo anterior, la única manera en que una solicitud de interrupción provoca en efecto una interrupción en el programa es cuando:

- La máscara global está activada (**GIE=1**).
- (En su caso) la máscara global de periféricos está activada (**PEIE=1**)
- La máscara local está activada
- Ocurre un evento que activa la bandera correspondiente.

La lógica de activación de máscaras y banderas descrita arriba puede entenderse en términos de el diagrama lógico mostrado en la siguiente figura. En este diagrama se muestran las 14 fuentes de interrupción del PIC16F87x y se usan los nombres específicos de cada fuente de interrupción para sus respectivas banderas y máscaras de interrupción.



Proceso de reconocimiento de una interrupción

Cuando se cumplen las siguientes tres condiciones simultáneamente:

- El bit GIE está activado (en alto)
- Se produce un evento que solicita interrupción (se activa alguna de las banderas de interrupción)
- Está activada la máscara correspondiente a la bandera activada.

Entonces la CPU es interrumpida inmediatamente y ejecuta lo siguiente:

- Termina la ejecución de la instrucción actual.
- Desactiva el bit GIE (GIE=0) para bloquear cualquier otra solicitud de interrupción.
- La dirección de programa de la siguiente instrucción a ejecutar es guardada en el stack.
- Ejecuta un salto a la localidad de programa **0004h** denominada **vector de interrupción**, en donde el usuario deberá haber colocado el inicio de la **rutina de atención a la interrupción**.
- Ejecuta la rutina de atención a la interrupción escrita por el usuario, en la cual éste podrá constatar la fuente de interrupción consultando por poleo las banderas de interrupción.
- La rutina de atención a la interrupción deberá limpiar los bits de la bandera que solicitó la interrupción antes de rehabilitar interrupciones, para evitar interrupciones recursivas.
- La rutina de atención deberá terminar con una instrucción **RETFIE**, la cual activa nuevamente el bit GIE (GIE=1) y lee el stack para continuar la ejecución del programa que fue interrumpido en la siguiente instrucción.

Salvando el contexto durante una interrupción.

Dado que el programa principal no puede prever en que punto será interrumpido, la rutina de atención a la interrupción debe ser tal que no altere ninguno de los registros que requiere paso a paso el programa principal para su operación normal. Estos registros son los que van guardando el contexto del programa (tal como en qué banco está, el resultado de la última operación, etc.) y son:

Registro STATUS

Registro W

Registro PCLATH (si se están usando las páginas 1, 2 o 3).

El siguiente es un ejemplo del código que deberá incluirse al inicio y al final de la rutina de atención a la interrupción para salvar y recuperar el contexto:

```
* Salva información de contexto previo a la rutina de atención a la interrupción
MOVWF W_Temp      ;Salva el registro W en un registro temporal
SWAPF STATUS,W    ;Copia STATUS en W (usa SWAP para no alterarlo al copiarlo)
CLRF STATUS       ;Banco cero, sin importar banco actual
MOVWF STATUS_temp ;Salva STATUS en STATUS_temp (Banco 0)
; MOVF PCLATH,W    ;sólo se requiere si se están usando las páginas 1,2,y/o 3
; MOVWF PCLATH_temp ;salva PCLATH
; CLRF PCLATH      ;página 0 sin importar página actual
...
;aquí se escribe el código de la rutina de atención a la interrupción
...
```

```
;* A continuación restablece la información de contexto que salvó al inicio
;   MOVF PCLATH_Temp,W;rescata PCLATH
;   MOVWF PCLATH      ;si se usan las paginas 1,2 y/o 3
;   SWAPF STATUS_temp,W;rescata el STATUS original
;   MOVWF STATUS      ;restablece banco original
;   SWAPF W_temp,F     ;rescata el W original
;   SWAPF W_temp,W     ;sin alterar el STATUS ya rescatado.
;   RETFIE
```

Ejemplo 21.- Medidor de periodo mejorado Por Interrupciones.

Como ya se dijo para el ejemplo 20, éste tiene limitantes muy grandes especialmente en el límite superior de frecuencia que puede procesar correctamente, debido a que durante el tiempo empleado en la transmisión del dato no puede detectar ninguna transición de la señal de entrada. Esto puede corregirse si la detección de la transición se realiza mediante interrupciones. La modificación se muestra en el siguiente listado.

```
;*****
;* Este programa mide el periodo de una señal oscilatoria en la *
;* patita RC2/CCP1. Por INTERRUPCIONES generadas por la captura *
;* de cada transición de 0 a 1 en dicha patita.                  *
;* El valor de periodo capturado representa el número de ciclos *
;* Tcy por periodo. Dicho valor se envía continuamente por      *
;* el puerto serie. Se supone un cristal de 14.7456 Mhz          *
;*****
;   Include "p16f877.inc"
msnib      EQU 0x20
lsnib      EQU 0x21
STATUS_temp EQU 0x70
W_temp     EQU 0x71

;   org 0x0000      ;inicia con un reset
;   GOTO inic
;   org 0x0004      ;vector de interrupción
;   GOTO interr      ;salta a la rutina de atención a la interrupción
inic CALL initrans   ;inicializa puerto serie como transmisor
;   BSF STATUS,RP0   ;Banco1
;   BSF TRISC,2      ;patita RC2/CCP1 como entrada
;   BCF STATUS,RP0   ;Banco 0
;   MOVLW 0x01
;   MOVWF T1CON      ;Configura Timer1 modo temporizador, preesc 1/1
;   CLRF TMR1H       ;Inicializa en cero el timer 1
;   CLRF TMR1L       ;apaga el módulo CCP para inicializar
;   CLRF CCP1CON      ;limpia latch de CCP1
;   BSF CCP1CON,CCP1M2;Habilita modulo CCP1 para modo de captura
;   BSF CCP1CON,CCP1M0;en transición de subida
;   BCF PIR1,CCP1IF   ;limpia bandera de interrupcion.
;   BSF STATUS,RP0    ;banco 1
;   BSF PIE1,CCP1IE   ;Habilita interrupciones del CCP1
;   BCF STATUS,RP0    ;banco 0
;   BSF INTCON,PEIE   ;habilita interrupciones de periféricos
;   BSF INTCON,GIE    ;Habilita interrupciones globales
;** Programa principal:
;** envía continuamente dato de captura al puerto serie
main MOVF CCPR1H,W    ;copia periodo capturado
;   CALL Envbyte      ;y lo envía por el puerto serie
;   MOVF CCPR1L,W
;   CALL Envbyte
;   MOVLW 0x0D        ;envía separador
;   CALL envia
;   MOVLW 0x0A
;   CALL envia
;   GOTO main         ;repite
;***** rutina de atención a la interrupción
```

```

interr
    MOVWF W_temp      ;salva contexto
    SWAPF STATUS,W
    CLRF STATUS
    MOVWF STATUS_temp
    BTFSS PIR1,CCP1IF ;checa bandera de captura de evento
    GOTO ret          ;si no es bandera de captura retorna
    BCF PIR1,CCP1IF   ;si es bandera de captura, la limpia
    CLRF TMR1L        ;limpia la cuenta del timer 1
    CLRF TMR1H
ret
    SWAPF STATUS_temp,W;restablece contexto
    MOVWF STATUS
    SWAPF W_temp,F
    SWAPF W_temp,W
    RETFIE
;*****
; Subrutina que envía el byte en W por el puerto serie, separado
; en los códigos ASCII de sus dos nibbles hexadecimales
;*****
Envbyte:
    MOVWF msnib      ;pone byte en msnib
    MOVWF lsnib      ;y una copia en lsnib
    SWAPF msnib,1    ;intercambia nibbles en lsnib
    MOVLW 0x0F       ;máscara para limpiar el nibble alto
    ANDWF msnib,1    ;limpia parte alta de msnib
    ANDWF lsnib,1    ;limpia parte alta de lsnib
    MOVF msnib,W      ;carga msnib en W
    CALL asc         ;obtiene código ASCII equivalente
    CALL envia       ;lo envía por el puerto serie
    MOVF lsnib,W      ;carga lsnib en W
    CALL asc         ;obtiene código ASCII equivalente
    CALL envia       ;lo envía por el puerto serie
    RETURN
asc
    ADDWF PCL,1      ;Calcula el código a retornar
                    ;Saltando W instrucciones adelante
    DT "0123456789ABCDEF"
;*****
;Subrutina para inicializar el puerto serie USART como transmisor
;a 9600 Bauds, considerando un cristal de reloj de 14.7456 MHZ
;*****
initrans:
    BCF STATUS,RP1
    BSF STATUS,RP0   ;banco 1
    BCF TXSTA,BRGH   ;pone bit BRGH=0 (velocidad baja)
    MOVLW 0x17       ;valor para 9600 Bauds (Fosc=14.7456 Mhz)
    MOVWF SPBRG      ;configura 9600 Bauds
    BCF TXSTA,SYNC   ;limpia bit SYNC (modo asíncrono)
    BSF TXSTA,TXEN    ;pone bit TXEN=1 (habilita transmisión)
    BCF STATUS,RP0   ;regresa al banco 0
    BSF RCSTA,SPEN    ;pone bit SPEN=1 (habilita puerto serie)
    RETURN
;*****
;Subrutina para enviar el byte guardado en W por el puerto serie
;*****
envia
    BSF STATUS,RP0   ;banco 1
esp
    BTFSS TXSTA,TRMT ;checa si el buffer de transmisión
    GOTO esp         ;si está ocupado espera
    BCF STATUS,RP0   ;regresa al banco 0
    MOVWF TXREG      ;envía dato guardado en W
    RETURN
end

```