

Curso avanzado de Microcontroladores PIC

Capítulo 1: El PIC16F84

Capítulo 2: Proyectos con el PIC16F84

Capítulo 3: El PIC16C71

Capítulo 4: Proyectos con el PIC16C71

Capítulo 5: El PIC12C50x

Apéndice A: Instrucciones de los PIC 16F84 y 12C71



El PIC16F84

- ***Pines y funciones***
- ***Arquitectura***
- ***Características especiales***
- ***El PIC16C84***
- ***Compatibilidad con otras familias***

El PIC16F84 es un microcontrolador con memoria de programa tipo *FLASH*, lo que representa gran facilidad en el desarrollo de prototipos y en su aprendizaje ya que no se requiere borrarlo con luz ultravioleta como las versiones EPROM sino, permite reprogramarlo nuevamente sin ser borrado con anterioridad. Por esta razón, lo usaremos en la mayoría de aplicaciones que se desarrollan a lo largo del curso.

Pines y funciones

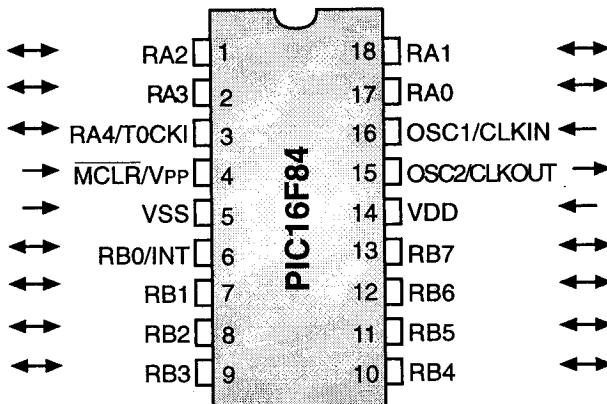


Figura 1.1. Diagrama de pines del PIC16F84

El PIC16F84 es un microcontrolador de *Microchip Technology* fabricado en tecnología CMOS, su consumo de potencia es muy bajo y además es completamente estático, esto quiere decir que el reloj puede detenerse y los datos de la memoria no se pierden.

El encapsulado más común para el microcontrolador es el DIP (*Dual In-line Pin*) de 18 pines, propio para usarlo en experimentación. La referencia completa es 16F84-04/P, para el dispositivo que utiliza reloj de 4 MHz. Sin embargo, hay otros tipos de encapsulado que se pueden utilizar según el diseño y la aplicación que se quiere realizar. Por ejemplo, el encapsulado tipo *surface mount* (montaje superficial) tiene un reducido tamaño y bajo costo, que lo hace propio para producciones en serie o para utilizarlo en lugares de espacio muy reducido, la figura 1.2 muestra los tipos de empaque que puede tener el integrado.

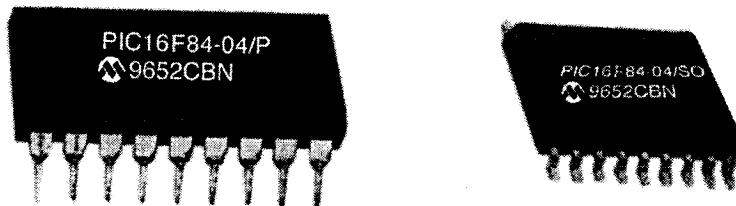


Figura 1.2. Tipos de encapsulado

Puertos del microcontrolador

Los puertos son el puente entre el microcontrolador y el mundo exterior. Son líneas digitales que trabajan entre cero y cinco voltios y se pueden configurar como entradas o como salidas.

El PIC16F84 tiene dos puertos. El puerto A con 5 líneas y el puerto B con 8 líneas, figura 1.3. Cada pin se puede configurar como entrada o como salida independiente programando un par de registros diseñados para tal fin. En ese registro un "0" configura el pin del puerto correspondiente como salida y un "1" lo configura como entrada.

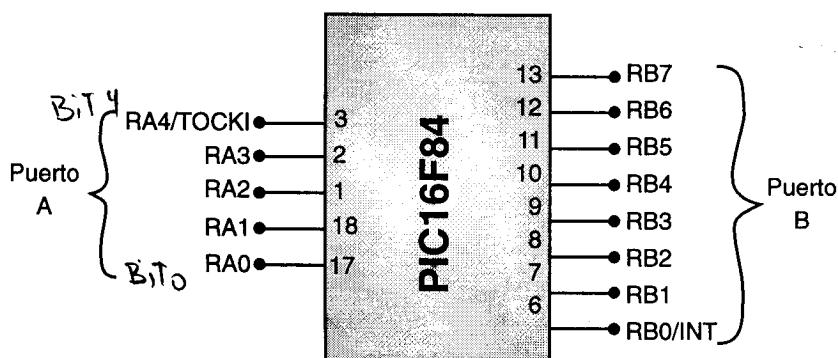


Figura 1.3. Puertos del PIC16F84

El puerto B tiene internamente unas resistencias de *pull-up* conectadas a sus pines (sirven para fijar el pin a un nivel de cinco voltios), su uso puede ser habilitado o deshabilitado bajo control del programa. Todas las resistencias de pull-up se conectan o se desconectan a la vez, usando el bit llamado RBPU que se encuentra en el registro (posición de memoria RAM) llamado OPTION. La resistencia de pull-up es desconectada automáticamente en un pin si este se programa como salida. El pin RB0/INT se puede configurar por software para que funcione como interrupción externa, para configurarlo se utilizan unos bits de los registros INTCON y OPTION.

El pin RA4/TOCKI del puerto A puede ser configurado como un pin de entrada/salida o como entrada del temporizador/contador. Cuando este pin se programa como entrada digital, funciona como un disparador de Schmitt (*Schmitt trigger*), puede reconocer señales un poco distorsionadas y llevarlas a niveles lógicos (cero y cinco voltios). Cuando se usa como salida digital se comporta como colector abierto, por lo tanto, se debe poner una resistencia de *pull-up* (resistencia externa conectada a un nivel de cinco voltios). Como salida, la lógica es inversa: un "0" escrito al pin del puerto entrega en el pin un "1" lógico. Además, como salida no puede manejar cargas como fuente, sólo en el modo sumidero.

Como este dispositivo es de tecnología CMOS, todos los pines deben estar conectados a alguna parte, nunca dejarlos al aire porque se puede dañar el integrado. Los pines que no se estén usando se deben conectar a la fuente de alimentación de +5V, como se muestra en la figura 1.4.

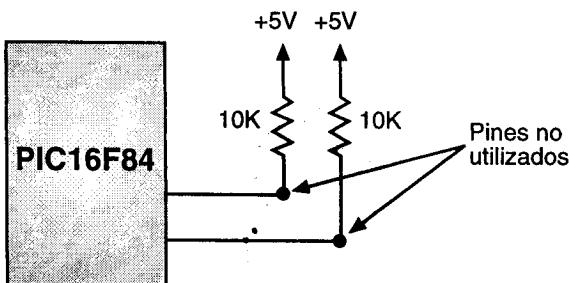


Figura 1.4. Los puertos no utilizados se deben conectar a la fuente

La máxima capacidad de corriente de cada uno de los puertos en modo sumidero (*sink*) es de 25 mA y en modo fuente (*source*) es de 20 mA, figura 1.5. La máxima capacidad de corriente total de los puertos es:

	PUERTO A	PUERTO B
Modo sumidero	80 mA	150 mA
Modo fuente	50 mA	100 mA

El consumo de corriente del microcontrolador para su funcionamiento depende del voltaje de operación, la frecuencia y de las cargas que tengan sus pines. Para un reloj de 4 MHz el consumo es de aproximadamente 2 mA; aunque este se puede reducir a 40 microamperios cuando se está en el modo *sleep* (en este modo el micro se detiene y disminuye el consumo de potencia). Se sale de ese estado cuando se produce alguna condición especial que veremos más adelante.

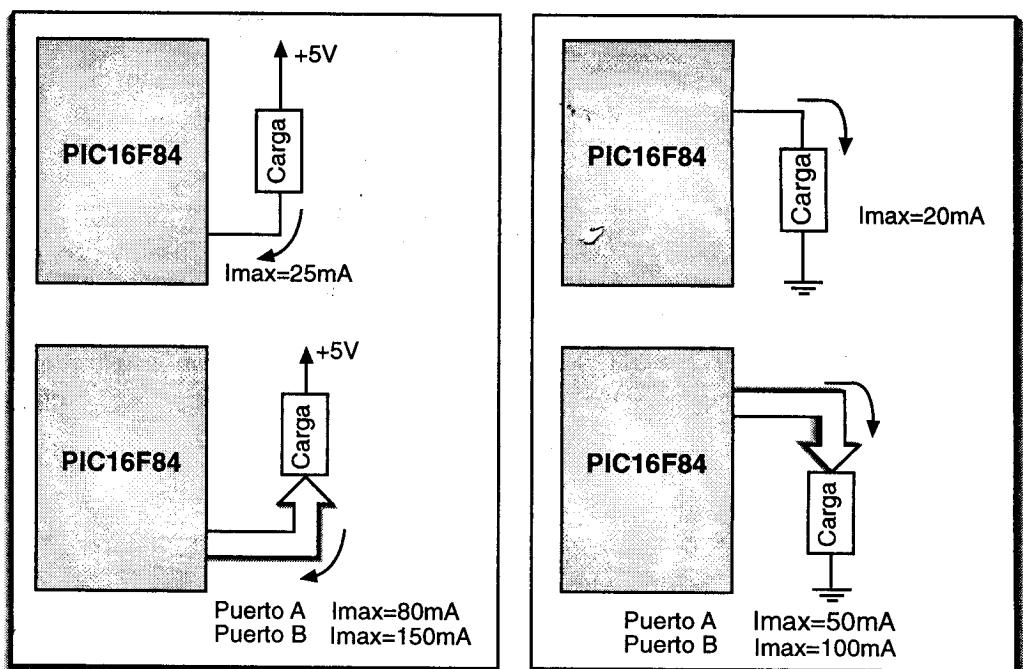


Figura 1.5. Capacidad de corriente del PIC16F84

El oscilador externo

Todo microcontrolador requiere un circuito externo que le indique la velocidad a la que debe trabajar. Este circuito, que se conoce como oscilador o reloj, es muy simple pero de vital importancia para el buen funcionamiento del sistema. El PIC16F84 puede utilizar cuatro tipos de oscilador diferentes. Estos tipos son:

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

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

El tipo de oscilador que se sugiere para las prácticas es el cristal de 4 MHz, porque garantiza mayor precisión y un buen arranque del microcontrolador. Internamente esta frecuencia es dividida por cuatro, lo que hace que la frecuencia efectiva de trabajo sea de 1 MHz, por lo que cada instrucción se ejecuta en un microsegundo. El cristal debe ir acompañado de dos condensadores y se conecta como se muestra en la figura 1.6.

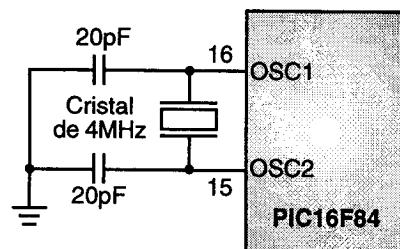


Figura 1.6. Conexión de un oscilador a cristal

Dependiendo de la aplicación, se pueden utilizar cristales de otras frecuencias; por ejemplo se usa el cristal de 3.579545 MHz porque es muy económico, el de 32.768 KHz cuando se necesita crear bases de tiempo de un segundo muy precisas. El límite de velocidad en estos microcontroladores es de 10 MHz.

Si no se requiere mucha precisión en el oscilador y se quiere economizar dinero, se puede utilizar una resistencia y un condensador, como se muestra en la figura 1.7.

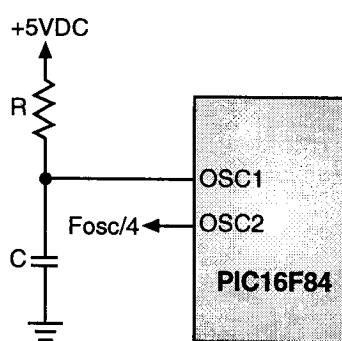


Figura 1.7. Conexión de un oscilador RC

Reset

En los microcontroladores se requiere un pin de *reset* para reiniciar el funcionamiento del sistema cuando sea necesario, ya sea por una falla que se presente o porque así se halla diseñado el sistema. El pin de reset en los PIC es llamado MCLR (*master clear*). El PIC16F84 admite diferentes tipos de *reset*:

- Al encendido (*Power On Reset*)
- Pulso en el pin MCLR durante operación normal

- Pulso en el pin **MCLR** durante el modo de bajo consumo (modo *sleep*)
- El rebase del conteo del circuito de vigilancia (*watchdog*) durante operación normal
- El rebase del conteo del circuito de vigilancia (*watchdog*) durante el modo de bajo consumo (*sleep*)

El *reset* al encendido se consigue gracias a dos temporizadores. El primero de ellos es el OST (*Oscillator Start-Up Timer*: Temporizador de encendido del oscilador), orientado a mantener el microcontrolador en *reset* hasta que el oscilador del cristal es estable. El segundo es el PWRT (*Power-Up Timer* : Temporizador de encendido), que provee un retardo fijo de 72 ms (nominal) en el encendido únicamente, diseñado para mantener el dispositivo en *reset* mientras la fuente se estabiliza. Para utilizar estos temporizadores, sólo basta con conectar el pin MCLR a la fuente de alimentación, evitándose utilizar las tradicionales redes RC externas en el pin de *reset*.

El *reset* por MCLR se consigue llevando momentáneamente este pin a un estado lógico bajo, mientras que el watchdog WDT produce el *reset* cuando su temporizador rebasa la cuenta, o sea que pasa de OFFh a 00h. Cuando se quiere tener control sobre el *reset* del sistema se puede conectar un botón como se muestra en la figura 1.8.

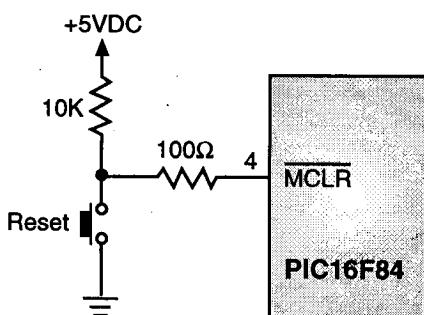


Figura 1.8. Conexión del botón de reset

Arquitectura

Este término se refiere a los bloques funcionales internos que conforman el microcontrolador y la forma en que están conectados, por ejemplo la memoria FLASH (de programa), la memoria RAM (de datos), los puertos, la lógica de control que permite que todo el conjunto funcione, etc.

La figura 1.9 muestra la arquitectura general del PIC16F84, en ella se pueden apreciar los diferentes bloques que lo componen y la forma en que se conectan. Se muestra la conexión de los puertos, las memorias de datos y de programa, los bloques especiales como el *watchdog*, los temporizadores de arranque, el oscilador, etc.

Todos los elementos se conectan entre sí por medio de buses. Un bus es un conjunto de líneas que transportan información entre dos o más módulos. Vale la pena destacar que el PIC16F84 tiene un bloque especial de memoria de datos de 64 bytes del tipo EEPROM, además de los dos bloques de memoria principales que son el de programa y el de datos o registros.

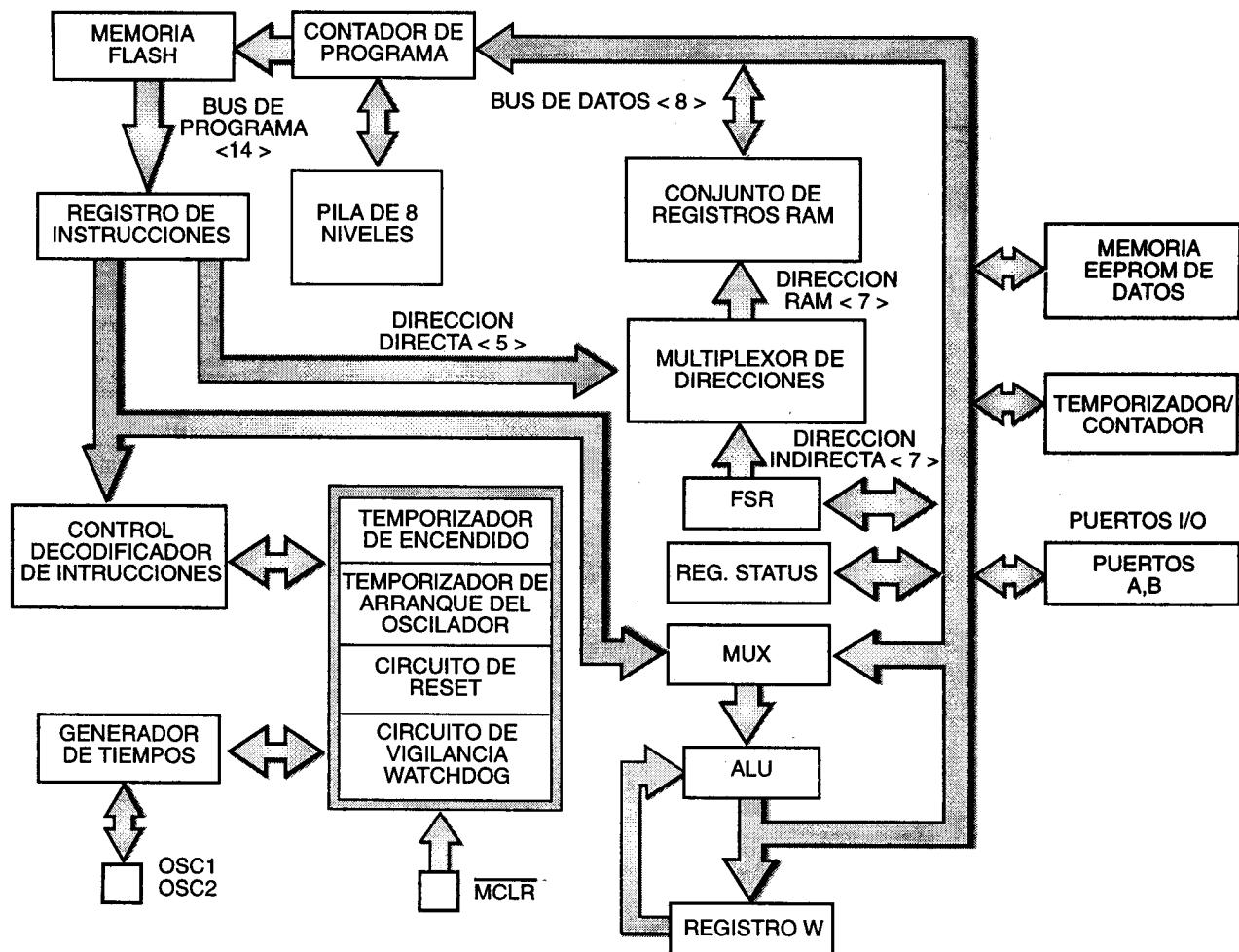


Figura 1.9. Arquitectura interna del PIC16F84

El PIC16F84 se basa en la arquitectura *Harvard*, en la cual el programa y los datos se pueden trabajar desde memorias separadas, lo que posibilita que las instrucciones y los datos posean longitudes diferentes. Esta misma estructura es la que permite la superposición de los ciclos de búsqueda y ejecución de las instrucciones, lo cual se ve reflejado en una mayor velocidad del microcontrolador.

Memoria de programa

Es una memoria de 1 Kbyte de longitud con palabras de 14 bits. Como es del tipo *FLASH* se puede programar y borrar eléctricamente, lo que facilita el desarrollo de los programas y la experimentación. En ella se graba, o almacena, el programa o códigos que el microcontrolador debe ejecutar. Dado que el PIC16F84 tiene un contador de programa de 13 bits, tiene una capacidad de direccionamiento de 8K x 14, pero solamente tiene implementado el primer 1K x 14 (0000h hasta 03FFh). Si se direccionan posiciones de memoria superiores a 3FFh se causará un solapamiento con el espacio del primer 1K. En la figura 1.10 se muestra el mapa de la memoria de programa.

Vector de reset. Cuando ocurre un reset al microcontrolador, el contador de programa se pone en ceros (000H). Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la iniciación del mismo.

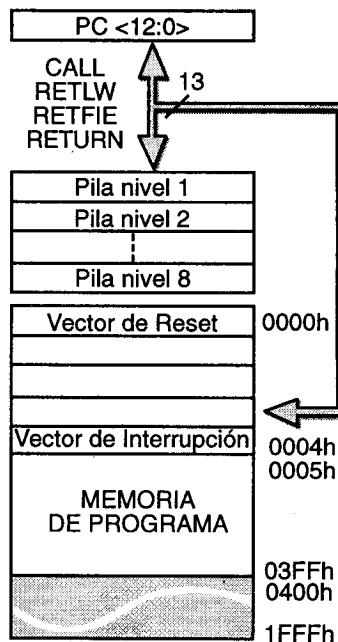


Figura 1.10. Mapa de la memoria de programa

Vector de interrupción. Cuando el microcontrolador recibe una señal de interrupción, el contador de programa apunta a la dirección 04H de la memoria de programa, por eso, allí se debe escribir toda la programación necesaria para atender dicha interrupción.

Registros (Memoria RAM)

El PIC16F84 puede direccionar 128 posiciones de memoria RAM, pero solo tiene implementados físicamente los primeros 80 (0-4F en hexadecimal). De estos los primeros 12 son registros que cumplen un propósito especial en el control del microcontrolador y los 68 siguientes son registros de uso general que se pueden usar para guardar los datos temporales de la tarea que se está ejecutando, figura 1.11.

Los registros están organizados como dos arreglos (páginas) de 128 posiciones de 8 bits cada una (128 x 8); todas las posiciones se pueden accesar directa o indirectamente (esta última a través del registro selector FSR). Para seleccionar que página de registros se trabaja en un momento determinado se utiliza el bit RP0 del registro STATUS. A continuación haremos una descripción de los registros:

00h o INDO: Registro para direccionamiento indirecto de datos. Este no es un registro disponible físicamente; utiliza el contenido del FSR y el bit RP0 del registro STATUS para seleccionar indirectamente la memoria de datos o RAM del usuario; la instrucción determinará que se debe realizar con el registro señalado.

01h o TMR0. Temporizador/contador de 8 bits. Este se puede incrementar con una señal externa aplicada al pin RA4/TOCKI o de acuerdo a una señal interna proveniente del reloj de instrucciones del microcontrolador. La tasa de incremento del registro se puede determinar por medio de un preescalador, localizado en el registro OPTION. Como una mejora con respecto a sus antecesores, se le ha agregado la generación de interrupción cuando se rebasa la cuenta (el paso de OFFh a 00h).

	*Direc. Indirecto	*Direc. Indirecto	
00h			80h
01h	TMRO	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h			87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 Registros de propósito general	Mapeado en página 0	8Ch
4Fh			CFh
S0h			D0h
7Fh			FFh

Página 0 Página 1

* No es un registro físico
■ Posiciones no implementadas

Figura 1.11. Registros del PIC16F84

02h o PCL: Contador de programa. Se utiliza para direccionar las palabras de 14 bits del programa del usuario que se encuentra almacenado en la memoria ROM; este contador de programas es de 13 bits de ancho, figura 1.12. Sobre el byte bajo, se puede escribir o leer directamente, mientras que sobre el byte alto, no. El byte alto se maneja mediante el registro PCLATH (0Ah). A diferencia de los PIC de primera generación, el 16F84 ante una condición de *reset* inicia el contador de programa con todos sus bits en “cero”. Durante la ejecución normal del programa, y dado que todas las instrucciones ocupan sólo una posición de memoria, el contador se incrementa en uno con cada instrucción, a menos que se trate de alguna instrucción de salto.

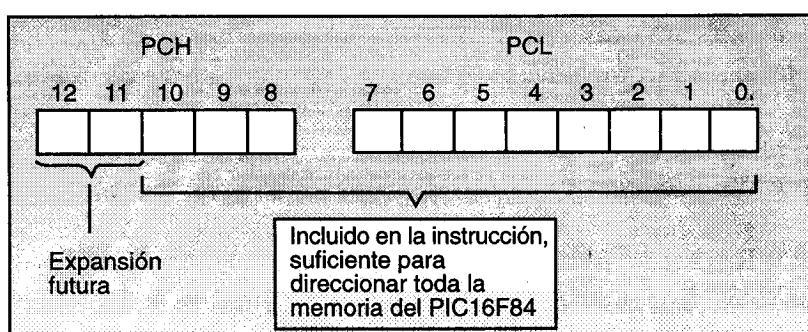


Figura 1.12. Contador de programa (13 bits)

En una instrucción CALL o GOTO, los bits PC<10:0> se cargan desde el código de operación de la instrucción, mientras que los bits PC<11:12> lo hacen desde el PCLATH<4:3>. Como solamente el primer 1K de memoria está implementado, el código de operación de la instrucción puede contener la dirección destino, eso quiere decir que se pueden hacer saltos y llamados a subrutinas sin necesidad de tener en cuenta la paginación de memoria de programa.

En otras instrucciones donde PCL es el destino, PC<12:8> se carga directamente desde el PCLATH<4:0>, por ejemplo en el caso de la instrucción ADDWF. Esto se debe tener en cuenta cuando se desea hacer lectura de tablas usando el comando: ADDWF PC,1 , en este caso se debe tener en cuenta que la tabla debe estar comprendida dentro de un solo bloque de 256 bytes (0-255, 256-511, etc.).

03h o STATUS: Registro de estados. Contiene el estado aritmético de la ALU, la causa del *reset* y los *bits* de preselección de página para la memoria de datos. La figura 1.13 muestra los *bits* correspondientes a este registro. Los *bits* 5 y 6 (RP0 y RP1) son los *bits* de selección de página para el direccionamiento directo de la memoria de datos; solamente RP0 se usa en los PIC16F84. RP1 se puede utilizar como un *bit* de propósito general de lectura/escritura. Los *bits* TO y PD no se pueden modificar por un proceso de escritura; ellos muestran la condición por la cual se ocasionó el último *reset*.

Registro: STATUS							
IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7							bit 0
Dirección: 03h							
condición de reset: 000??XXX							
IRP: Selector de página para direccionamiento indirecto. Este <i>bit</i> no se utiliza efectivamente en el PIC16F84, por lo que se puede utilizar como un <i>bit</i> de propósito general. RP1,0: Selectores de página para direccionamiento directo. Solamente RP0 se utiliza en el PIC16F84. RP1 se puede utilizar como un <i>bit</i> de propósito general. TO: Time Out o Bit de finalización del temporizador. Se coloca en 0 cuando el circuito de vigilancia Watchdog finaliza la temporización. PD: Power Down o Bit de bajo consumo. Se coloca en 0 por la instrucción SLEEP. Z: Zero o Bit de cero. Se coloca en 1 cuando el resultado de una operación lógica o aritmética es cero. DC: Digit Carry o Bit de acarreo de dígito. En operaciones aritméticas se activa cuando hay acarreo entre el <i>bit</i> 3 y el 4. C: Carry o Bit de acarreo. En instrucciones aritméticas se activa cuando se presenta acarreo desde el <i>bit</i> más significativo del resultado.							

Figura 1.13. Registro de estados

04h o FSR: Registro selector de registros. En asocio con el registro IND0, se utiliza para seleccionar indirectamente los otros registros disponibles. Mientras que los antecesores del PIC16F84 sólo poseían 5 *bits* activos, en este microcontrolador se poseen los 8 *bits*. Si en el programa no se utilizan llamadas indirectas, este registro se puede utilizar como un registro de propósito general.

Para entender mejor el funcionamiento de este registro veamos un programa simple que borra el contenido de la memoria RAM, empleando direccionamiento indirecto.

	MOVLW 20	;inicializa el puntero en la memoria RAM
	MOVWF FSR	;que se va a borrar
NEXT	CLRF INDO	;borra el registro indexado (es decir el que está siendo direccionado por el FSR)
	INCF FSR,R	;incrementa el puntero
	BTFSS FSR,5	;pregunta si ya acabó el banco de memoria
	GOTO NEXT	;sigue borrando los registros que faltan

continúa

05h o PORTA: Puerto de Entrada/Salida de 5 bits. Este puerto, al igual que todos sus similares en los PIC, puede leerse o escribirse como si se tratara de un registro cualquiera. El registro que controla el sentido (entrada o salida) de los pines de este puerto está localizado en la página 1, en la posición 85h y se llama TRISA.

06h o PORTB: Puerto de entrada/salida de 8 bits. Al igual que en todos los PIC, este puede leerse o escribirse como si se tratara de un registro cualquiera; algunos de sus pines tienen funciones alternas en la generación de interrupciones. El registro de control para la configuración de la función de sus pines se localiza en la página 1, en la dirección 86h y se llama TRISB.

08h o EEDATA: Registro de datos de la EEPROM. Este registro contiene el dato que se va a escribir en la memoria EEPROM de datos o el que se leyó de ésta.

09h o EEADR: Registro de dirección de la EEPROM. Aquí se mantiene la dirección de la EEPROM de datos que se va a trabajar, bien sea para una operación de lectura o para una de escritura.

0Ah o PCLATH: Registro para la parte alta de la dirección. Este contiene la parte alta del contador de programa y no se puede acceder directamente.

0Bh o INTCON: Registro para el control de interrupciones. Es el encargado del manejo de las interrupciones y contiene los *bits* que se muestran en la figura 1.14.

81h u OPTION: Registro de configuración múltiple. Posee varios *bits* para configurar el preescalador, la interrupción externa, el timer y las características del puerto B. Los *bits* que contiene y las funciones que realiza este registro se muestran en la figura 1.15. El preescalador es compartido entre el MTRO y el WDT; su asignación es mutuamente excluyente ya que solamente puede uno de ellos ser preescalado a la vez.

85h o TRISA: Registro de configuración del puerto A. Como ya se mencionó, es el registro de control para el puerto A. Un “cero” en el *bit* correspondiente al pin lo configura como salida, mientras que un “uno” lo hace como entrada.

86h o TRISB: Registro de configuración del puerto B. Orientado hacia el control del puerto B. Son válidas las mismas consideraciones del registro anterior.

Registro: INTCON							
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit 7				bit 0			
Dirección: 0Bh				condición de reset: 0000000Xb			
<p>GIE: <i>Global Interrupt Enable</i> o Habilitador general de interrupciones. 0: deshabilita todas las interrupciones 1: habilita las interrupciones</p> <p>EEIE: <i>EEPROM Write Interrupt Enable</i> o Habilitación de interrupción por escritura de la EEPROM 0: la deshabilita 1: la habilita</p> <p>TOIE: <i>TMRO Interrupt Enable</i> o Habilitación de interrupción del temporizador TMRO. 0: la deshabilita 1: la habilita</p> <p>INTE: <i>INT Interrupt Enable</i> o Habilitación de la interrupción INT. 0: la deshabilita 1: la habilita</p> <p>RBIE: <i>RBIF Interrupt Enable</i> o Habilitación de la interrupción RBIF. 0: la deshabilita 1: la habilita</p> <p>TOIF: <i>TMRO Overflow Interrupt Flag</i> o Bandera de la interrupción por sobrejugo del TMRO. Se coloca en 1 cuando el TMRO pasa de 0FFh a 00h; ésta debe ser puesta a 0 por programa.</p> <p>INTF: <i>INT Interrupt Flag</i> o Bandera de interrupción INT. Se coloca en 1 cuando la interrupción INT (RB<0>) ocurre; ésta debe ser puesta a 0 por programa.</p> <p>RBIF: <i>RB Port Change Interrupt Flag</i> o Bandera de interrupción por cambio en el puerto B. Se coloca en 1 cuando una de las entradas RB<7:4> cambia; ésta debe ser puesta a 0 por programa.</p>							

Figura 1.14. Registro INTCON

88h o EECON1: Registro para el control de la memoria EEPROM de datos. Este es el registro de control de la memoria de datos y sólo destina cinco *bits* para ello, los más bajos; los tres *bits* superiores permanecen sin implementar. En la figura 1.16 se muestran las funciones de estos *bits*.

89h o EECON2: Registro auxiliar para control de la memoria EEPROM de datos. Registro que no está implementado físicamente en el microcontrolador, pero que es necesario en las operaciones de escritura en la EEPROM de datos; ante cualquier intento de lectura se obtendrán “ceros”.

0Ch a 4Fh: Registros de propósito general. Estas 68 posiciones están implementadas en la memoria RAM estática, la cual conforma el área de trabajo del usuario; a ellas también se accede cuando en la página 1 se direccionan las posiciones 8Ch a

Registro: OPTION																																		
RBPU	INTEDG	GRTS	RTE	PSA	PS2	PS1	PSO																											
bit 7				bit 0																														
Dirección: 81h																																		
condición de reset: 11111111b																																		
RBPU: PortB Pull-up Enable o Habilitación de pull-up del puerto B. 0: habilita las pull-ups internas 1: las deshabilita																																		
INTEDG: INT Interrupt Edge Select o Selector de flanco de la interrupción INT 0: flanco de bajada 1: flanco de subida																																		
RTS: TMR0 Signal Source o Fuente de señal de TMR0. 0: ciclo de instrucciones interno (Temporizador) 1: transición en el pin RA4/TOCK (Contador)																																		
RTE: TMR0 Signal Edge o Flanco de la señal TMR0 0: incremento en transición de bajo a alto 1: incremento en transición de alto a bajo																																		
PSA: Prescaler Assignment o Asignación del preescalador 0: TMR0 (Contador/Temporizador) 1: WDT (Circuito de vigilancia)																																		
PS2,1,0: Prescaler Value o Valores del preescalador.																																		
<table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Valor</th> <th>TMR0</th> <th>WDT</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>1:2</td> <td>1:1</td> </tr> <tr> <td>001</td> <td>1:4</td> <td>1:2</td> </tr> <tr> <td>010</td> <td>1:8</td> <td>1:4</td> </tr> <tr> <td>011</td> <td>1:16</td> <td>1:8</td> </tr> <tr> <td>100</td> <td>1:32</td> <td>1:16</td> </tr> <tr> <td>101</td> <td>1:64</td> <td>1:32</td> </tr> <tr> <td>110</td> <td>1:128</td> <td>1:64</td> </tr> <tr> <td>111</td> <td>1:256</td> <td>1:128</td> </tr> </tbody> </table>								Valor	TMR0	WDT	000	1:2	1:1	001	1:4	1:2	010	1:8	1:4	011	1:16	1:8	100	1:32	1:16	101	1:64	1:32	110	1:128	1:64	111	1:256	1:128
Valor	TMR0	WDT																																
000	1:2	1:1																																
001	1:4	1:2																																
010	1:8	1:4																																
011	1:16	1:8																																
100	1:32	1:16																																
101	1:64	1:32																																
110	1:128	1:64																																
111	1:256	1:128																																

Figura 1.15. Registro OPTION

CFh. Esto se ha diseñado así para evitar un excesivo cambio de páginas en el manejo de la RAM del usuario, agilizando los procesos que se estén llevando a cabo y descomplicando la labor del programador.

Registro de trabajo W. Este es el registro de trabajo principal, se comporta de manera similar al acumulador en los microprocesadores. Este registro participa en la mayoría de las instrucciones.

Pila (Stack)

Estos registros no forman parte de ningún banco de memoria y no permiten el acceso por parte del usuario. Se usan para guardar el valor del contador de programa cuando se hace un llamado a una subrutina o cuando se atiende una interrupción; luego, cuando el micro regresa a seguir ejecutando su tarea normal, el contador de programa recupera su valor leyéndolo nuevamente desde la pila. El PIC16F84 tiene una pila de 8 niveles, esto significa que se pueden anidar 8 llamados a subrutina sin tener problemas.

Registro: EECON1											
U	U	U	EEIF	WRERR	WREN	WR	RD				
bit 7				bit 0							
Dirección: 88h				condición de reset: 0000X000b							
U:	<i>Unimplemented</i> No implementadas. Estos bits se leen como ceros.										
EEIF:	<i>EEPROM Write Completion Interrupt Flag</i> o Bandera de finalización de la escritura. Se coloca en "1" cuando finaliza con éxito la escritura en la EEPROM de datos; se debe colocar en "0" por programa. El bit de habilitación correspondiente es el EEIE, localizado en el registro INTCON (0B<6>).										
WRERR:	<i>Write Error Flag</i> o Bandera de error de escritura. Se coloca en 1 cuando la operación de escritura termina prematuramente, debido a cualquier condición de <i>reset</i> .										
WREN:	<i>Write Enable</i> o habilitación de escritura. Si se coloca en "0" no permite las operaciones de escritura; en "1" las habilita.										
WR:	<i>Write Control</i> o Control de escritura. Al colocarse en "1" inicia un ciclo de escritura. Este bit sólo es puesto a "0" por <i>hardware</i> , una vez la escritura termina.										
RD:	<i>Read Control</i> o Control de lectura. Al colocarse en "1" se inicia una lectura de la EEPROM de datos, la cual toma un ciclo del reloj de instrucciones. Este bit sólo se limpia (se coloca en "0") por <i>hardware</i> , al finalizar la lectura de la posición de la EEPROM.										

Figura 1.16. Registro EECON1

Características especiales

Algunos elementos que forman parte de los PIC no se encuentran en microcontroladores de otros fabricantes, o simplemente representan alguna ventaja o facilidad a la hora de hacer un diseño. Veamos una breve descripción de las más significativas:

Circuito de vigilancia (*Watchdog Timer*)

Su función es restablecer el programa cuando éste se ha perdido por fallas en la programación o por alguna razón externa. Es muy útil cuando se trabaja en ambientes con mucha interferencia o ruido electromagnético. Está conformado por un oscilador RC que se encuentra dentro del microcontrolador.

Este oscilador corre de manera independiente al oscilador principal. Cuando se habilita su funcionamiento, dicho circuito hace que el microcontrolador sufra un reset cada determinado tiempo (que se puede programar entre 18 ms y 2 segundos). Este reset lo puede evitar el usuario mediante una instrucción especial del microcontrolador (**CLRWDT**: borrar el conteo del watchdog), la cual se debe ejecutar antes de que termine el período nominal de dicho temporizador. De esta manera, si el programa se

ha salido de su flujo normal, por algún ruido o interferencia externa, el sistema se reiniciará (cuando se acabe el tiempo programado y no se haya borrado el contador) y el programa puede restablecerse para continuar con su funcionamiento normal.

En las primeras prácticas no se utiliza el circuito de vigilancia para facilitar el trabajo; por eso, en el momento de programar el microcontrolador se debe seleccionar en los fusibles de configuración “watchdog timer OFF”. Más adelante veremos algunos ejemplos que ilustran su funcionamiento y la manera de utilizarlo.

Temporizador de encendido (*Power-up Timer*)

Este proporciona un reset al microcontrolador en el momento de conectar la fuente de alimentación, lo que garantiza un arranque correcto del sistema. En el momento de grabar el micro se debe habilitar el fusible de configuración “*Power-up Timer*”, para ello se debe seleccionar la opción “ON”. Su tiempo de retardo es de 72 milisegundos.

Modo de bajo consumo (*sleep*)

Esta característica permite que el microcontrolador entre en un estado pasivo donde consume muy poca potencia. Cuando se entra en este modo el oscilador principal se detiene, pero el temporizador del circuito de vigilancia (watchdog) se reinicia y empieza su conteo nuevamente. Se entra en ese estado por la ejecución de una instrucción especial (llamada **SLEEP**) y se sale de él por alguna de las siguientes causas: cuando el microcontrolador sufre un reset por un pulso en el pin MCLR, porque el watchdog hace que se reinicie el sistema o porque ocurre una interrupción al sistema.

Interrupciones

Este microcontrolador incluye el manejo de interrupciones, lo cual representa grandes ventajas. El PIC16F84 posee cuatro fuentes de interrupción a saber:

- Interrupción externa en el pin RB0/INT
- Finalización del temporizador/contador TMRO
- Finalización de escritura en la EEPROM de datos
- Cambio de nivel en los pines RB4 a RB7

El registro 0Bh o INTCON contiene las banderas de las interrupciones INT, cambio en el puerto B y finalización del conteo del TMRO, al igual que el control para habilitar o deshabilitar cada una de las fuentes de interrupción, incluida la de escritura en memoria EEPROM. Sólo la bandera de finalización de la escritura reside en el registro 88h (EECON1<4>).

Si el *bit* GIE (*Global Interrupt Enable*) se coloca en 0, deshabilita todas las interrupciones. Cuando una interrupción es atendida, el *bit* GIE se coloca en 0 automáticamente para evitar interferencias con otras interrupciones que se pudieran presentar, la dirección de retorno se coloca en la pila y el PC se carga con la dirección 04h. Una vez en la rutina de servicio, la fuente de la interrupción se puede determinar examinando las banderas de interrupción. La bandera respectiva se debe colocar, por *software*, en cero antes de regresar de la interrupción, para evitar que se vuelva a detectar nuevamente la misma interrupción.

La instrucción RETFIE permite al usuario retornar de la interrupción, a la vez que habilita de nuevo las interrupciones, al colocar el *bit* GIE en uno. Debe tenerse presente que solamente el contador de programa es puesto en la pila al atenderse la interrupción; por lo tanto, es conveniente que el programador tenga cuidado con el registro de estados y el de trabajo, ya que se pueden producir resultados inesperados si dentro de ella se modifican.

Interrupción externa. Actúa sobre el pin RB0/INT y se puede configurar para activarse con el flanco de subida o el de bajada, de acuerdo al *bit* INTEDG (OPTION<6>). Cuando se presenta un flanco válido en el pin INT, la bandera INTF (INTCON<1>) se coloca en uno. La interrupción se puede deshabilitar colocando el *bit* de control INTE (INTCON<4>) en cero. Cuando se atiende la interrupción, a través de la rutina de servicio, INTF se debe colocar en cero antes de regresar al programa principal. La interrupción puede reactivar al microcontrolador después de la instrucción SLEEP, si previamente el *bit* INTE fue habilitado.

Interrupción por finalización de la temporización. La superación del conteo máximo (0FFh) en el TMR0 colocará el *bit* TOIF en uno (INTCON<2>). El *bit* de control respectivo es TOIE (INTCON<5>).

Interrupción por cambio en el puerto RB. Un cambio en los pines del puerto B <7:4> colocará en uno el *bit* RBIF (INTCON<0>). El *bit* de control respectivo es RBIE (INTCON<3>).

Interrupción por finalización de la escritura. Cuando la escritura de un dato en la EEPROM finaliza, se coloca en 1 el *bit* EEIF (EECON1<4>). El *bit* de control respectivo es EEIE (INTCON<6>)

Memoria de datos EEPROM

El PIC16F84 tiene una memoria EEPROM de datos de 64 posiciones (0h a 3Fh), de 8 *bits* cada una. Este bloque de memoria no se encuentra mapeado en ningún banco, el acceso a esas posiciones se consigue a través de dos registros de la RAM:

- el registro EEADR (posición 09), que debe contener la dirección de la posición de la EEPROM a ser accesada
- el registro EEDATA (posición 08), que contiene el dato de 8 *bits* que se va a escribir o el que se obtuvo de la última lectura.

Adicionalmente, existen dos registros de control: el EECON1 (88h), que posee cinco *bits* que manejan las operaciones de lectura/escritura y el EECON2 (89h), que aunque no es un registro físico, es necesario para realizar las operaciones de escritura.

La lectura toma un ciclo del reloj de instrucciones, mientras que la escritura, por ser controlada por un temporizador incorporado, tiene un tiempo nominal de 10 milisegundos, este tiempo puede variar con la temperatura y el voltaje. Cuando se va a realizar una operación de escritura, automáticamente se hace primero la operación de borrado. El número típico de ciclos de borrado/escritura de la EEPROM de datos es de 1.000.000.

Fusibles de configuración

El PIC16F84 posee cinco fusibles, cada uno de los cuales es un *bit*. Estos fusibles se pueden programar para seleccionar varias configuraciones del dispositivo: tipo de oscilador, protección de código, habilitación del circuito de vigilancia y el temporizador al encendido. Los *bits* se localizan en la posición de memoria 2007h, posición a la cual el usuario sólo tiene acceso durante la programación del microcontrolador. Cuando se programa la protección de código, el contenido de cada posición de la memoria no se puede leer completamente, de tal manera que el código del programa no se puede reconstruir. Adicionalmente, todas las posiciones de memoria del programa se protegen contra la reprogramación.

Una vez protegido el código, el fusible de protección sólo puede ser borrado (puesto a 1) si se borra toda la memoria del programa y la de datos.

Las pull-ups internas

Cada uno de los pines del puerto B tiene un débil elemento *pull-up* interno (250 µA típico); este elemento es automáticamente desconectado cuando el pin se configura como salida. Adicionalmente, el *bit* RBPU (OPTION<7>) controla todos estos elementos, los cuales están deshabilitados ante una condición de *reset*. Estos elementos *pull-up* son especialmente útiles cuando el microcontrolador va a colocarse en el modo de bajo consumo, ya que ayudan a no tener las entradas flotantes, significando una reducción en el consumo de corriente.

El conjunto de instrucciones

Estas se clasifican en orientadas a registros, orientadas al *bit* y operaciones literales y de control. Cada instrucción es una palabra de 14 *bits*, dividida en un código de operación (el cual especifica la orden a ejecutar) y uno o más operandos sobre los que se actúa. En el apéndice A se encuentra la lista completa de instrucciones, la cual incluye ejemplos y explicaciones. Como se puede observar allí, en total son 35, las cuales tardan un ciclo de máquina, a excepción de los saltos, que toman dos ciclos.

El PIC16C84

El PIC16C84 es un microcontrolador de la familia Microchip, totalmente compatible con el PIC16F84. Su principal característica es que posee memoria "EEPROM" en lugar de memoria *Flash*, pero su manejo es igual. Con respecto al PIC16F84, este microcontrolador presenta dos diferencias:

- La memoria de datos tiene menor tamaño, aquí se tienen 32 registros de propósito general (el mapa de memoria de datos llega hasta 2FH).
- En el momento de programar el microcontrolador, el fusible de selección del temporizador de arranque (*Power Up Timer*) trabaja de forma inversa, es decir, si en el PIC16F84 se selecciona la opción "Low" para activarlo, en el PIC16C84 se debe seleccionar "High".

Este microcontrolador ha sido reemplazado de forma gradual por el PIC16F84, por lo tanto, los diseños que lo utilicen como elemento de control deben ser actualizados. Aunque, como se ve, es un proceso casi transparente.

Compatibilidad con otras familias

Quienes están familiarizados con los PIC16C5X encontrarán básicamente las siguientes modificaciones:

- La longitud de las instrucciones se incrementó a 14 *bits*.
- Se puede omitir la paginación de la memoria de programa.
- La paginación de la memoria de datos se ha redefinido ligeramente, de tal manera que se elimina la necesidad de los *bits* PA2, PA1 y PA0 en el registro de estados.
- Se cuenta con cuatro nuevas instrucciones: RETURN, RETFIE, ADDLW y SUBLW.
- La manera en que se configuraban los puertos (con la instrucción TRIS) y se asignaba el preescalador (con la instrucción OPTION) ha sido modificada, de tal forma que los registros OPTION y TRIS ahora son direccionables; aún así, se conservan dichas instrucciones para mantener la compatibilidad con los PIC16C5X.
- Se agregaron capacidades de interrupción. El vector de interrupción es 0004h.
- El tamaño de la pila se incrementó a 8 niveles.
- El vector de *reset* se modificó a 0000h.
- Reactivación después de la instrucción SLEEP a través de interrupciones.
- El pin de entrada RTCC es ahora un pin del puerto A, llamado RA4/TOCK1
- El ancho del registro FSR se aumentó a 8 *bits*.
- La posición 07 no está implementada.
- Programación serial del microcontrolador.



Proyectos con el PIC16F84

- **Conexión de LED y dipswitch**
- **Manejo de un display de siete segmentos**
- **Multiplexaje de teclados y displays**
- **Conexión de memorias seriales al PIC**
- **Manejo de un módulo LCD**
- **Comunicación serial RS-232**
- **Características especiales de los PIC**
 - Interrupciones
 - Watchdog timer
 - EEPROM de datos del PIC16F84
- **Control de un motor paso a paso**

Proyecto N° 1: Conexión de LED y dipswitch

Como un ejercicio práctico que nos introduzca de manera rápida y sencilla en el manejo de los microcontroladores PIC, vamos a realizar un montaje simple, el cual consiste en conectar cuatro interruptores (dipswitch) como entradas del microcontrolador y cuatro LED como salidas. El programa que se escriba se debe encargar de verificar el estado de los dipswitch y de acuerdo a este, encender los LED. Este ejemplo aunque es muy simple, pero es fundamental para ejercitarse el manejo de los puertos. La figura 2.1 muestra el diagrama esquemático del circuito.

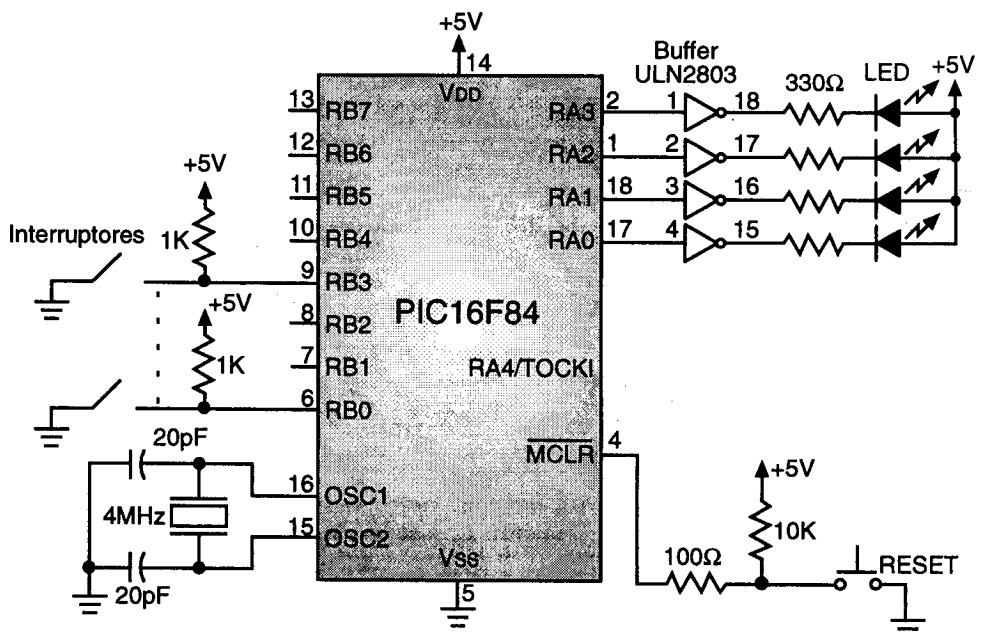


Figura 2.1. Conexión de los LED y dipswitch.

Debe notarse que los interruptores tienen resistencias conectadas a la fuente de alimentación, estas sirven para fijar un nivel alto cuando el dipswitch no está haciendo contacto. En este caso, cuando no se encuentra cerrado ningún interruptor el microcontrolador lee “unos” y cuando alguno se encuentre cerrado se leerá un “cero”. Por otra parte, para encender los LED se utiliza un circuito integrado ULN2803, el cual tiene un conjunto de transistores que invierten el pulso y amplifican la corriente. Por lo tanto, el pulso para encender un LED debe ser positivo.

Dado lo anterior, cuando se lee el estado de los dipswitch se debe invertir el valor leído, para asegurarse que el interruptor que esté cerrado se convierta en una señal positiva para encender el LED correspondiente. En la figura 2.2 se muestra el diagrama de flujo correspondiente al ejercicio y en la figura 2.3 el programa respectivo.

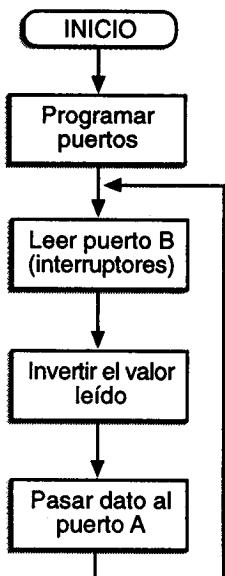


Figura 2.2. Diagrama de flujo para la conexión de los LED y dipswitch

```

;Este programa lee el estado de 4 interruptores y de acuerdo a ello enciende o
;no 4 LED
;En caso de que un número se escriba D'15': significa número decimal
;En caso de que el número se escriba B'00010101': significa número binario
;En caso de que un número se escriba 1SH: significa número hexadecimal
;Si no se especifica nada, se supone numeración hexadecimal
;definición de registros
pc      equ     02h
status  equ     03h
ptoa    equ     05h      ;el puerto A está en la dirección 05 de la RAM
ptob    equ     06h      ;el puerto B está en la dirección 06 de la RAM
trisa   equ     85h      ;registro de configuración del puerto A
trisb   equ     86h      ;registro de configuración del puerto B
w       equ     00h      ;indica que el resultado se guarda en W

reset   org     0         ;el vector de reset es la dirección 00
        goto  inicio    ;se salta al inicio del programa

        org     5         ;el programa empieza en la dirección de memoria 5

inicio   bsf     status,5 ;se ubica en el segundo banco de RAM
        movlw   0f0fh
        movwf   w
        trisa
        movlw   0ffh
        movwf   w
        trisb
        bcf    status,5 ;se ubica en el primer banco de memoria RAM

ciclo    movf   ptob,w   ;el valor de puerto B lo pasa al registro W
        xorlw  0ffh
        ;con una operación xor se invierte el valor
        ;del dato leído del puerto B
        movwf   ptoa
        goto  ciclo
        ;pasa el valor de W al puerto A

end

=====
; Fusibles de programación
; Osc XT
; Watchdog OFF
; Code protect OFF
; Power-Up-Timer ON
; Micro. PIC16F84
=====
```

Figura 2.3. Programa de la conexión de LED y dipswitch

Proyecto N° 2: Manejo de un display de siete segmentos

Los displays de siete segmentos son un elemento muy útil en el diseño de aparatos electrónicos, por ejemplo, cuando se requiere visualizar el dato proveniente de un conteo, de una temporización, el estado de una máquina, etc. El ejercicio que vamos a realizar consiste en hacer un contador decimal (de 0 a 9), el cual lleva el conteo del número de veces que se oprime una tecla (pulsador). Para manejar el display utilizaremos un decodificador 9368, que es compatible con el tradicional 7448, pero decodifica de binario a hexadecimal, es decir que puede mostrar los caracteres de A hasta F. En el ejercicio el microcontrolador debe encargarse de verificar cuando el conteo llega a 9 para empezar nuevamente en 0.

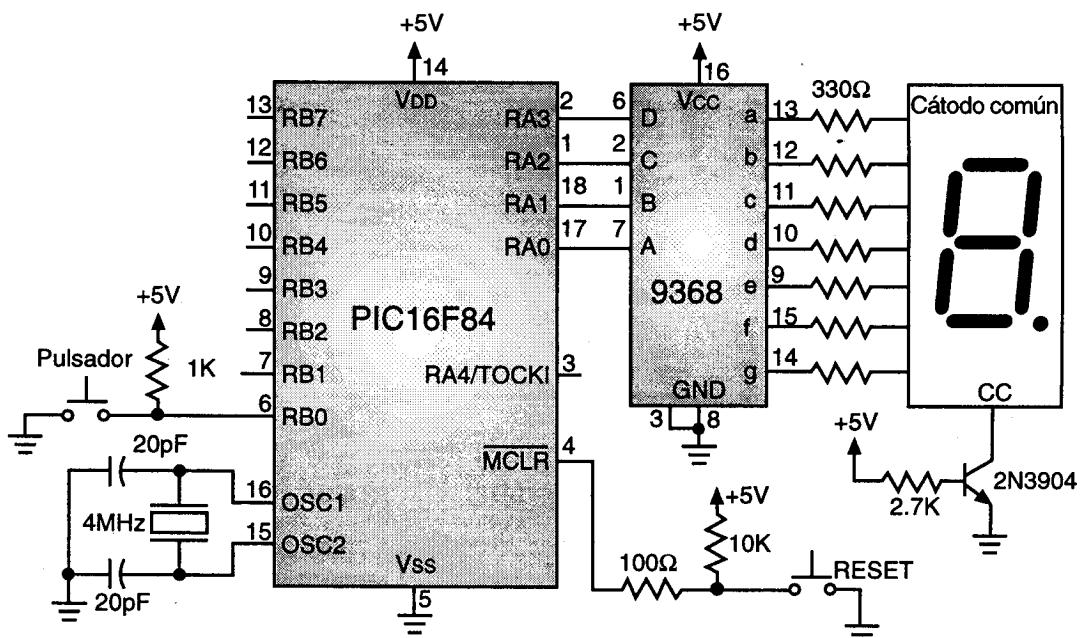


Figura 2.4. Manejo de un display de siete segmentos

El display utilizado es de cátodo común, para aumentar su visibilidad se conecta un transistor NPN que le entrega una buena corriente. En la figura 2.4 se muestra el diagrama correspondiente, en la figura 2.5 el diagrama de flujo y en la figura 2.6 el programa que realiza el control de las funciones.

Nota: Si se usa el decodificador 7448 en lugar del 9368, el pin 3 se debe dejar al aire

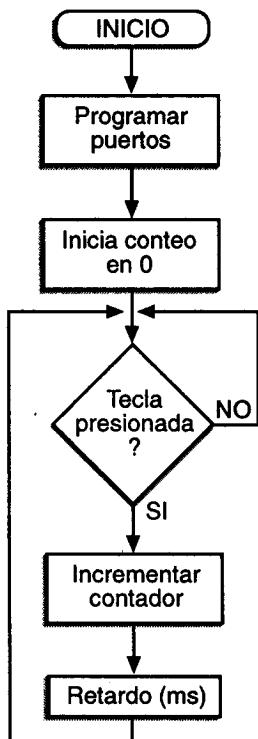


Figura 2.5. Diagrama de flujo del contador decimal

```

;Este programa hace un contador decimal en
;un display de 7 segmentos.
;En caso de que un número se escriba D'15': significa número decimal
;En caso de que el número se escriba B'00010101': significa número binario
;En caso de que un número se escriba 15H: significa número hexadecimal
;Si no se especifica nada, se supone numeración hexadecimal
;definición de registros
status equ 03h ;registro de estados
ptoA equ 05h ;el puerto A está en la dirección 05 de la RAM
ptoB equ 06h ;el puerto B está en la dirección 06 de la RAM
conta equ 0ch ;lleva el conteo de pulsaciones
loops equ 0dh ;utilizado en retardos (milisegundos)
loops2 equ 0eh ;utilizado en retardos
trisa equ 85h ;registro de configuración del puerto A
trisb equ 86h ;registro de configuración del puerto B
z equ 02h ;bandera de cero del registro de estados
reset org 0 ;el vector de reset es la dirección 00
goto inicio ;se salta al inicio del programa

retardo org 5 ;el programa empieza en la dirección de memoria 5
            ;subrutina de retardo de 100 milisegundos
            ;el registro loops contiene el número
            ;de milisegundos del retardo
            ;de milisegundos del retardo
top2      movlw D'100'
            movwf loops
            movlw D'110'
            movwf loops2
            decfsz loops2 ;pregunta si terminó 1 ms
            goto top
            decfsz loops ;pregunta si termina el retardo
            goto top2
            retlw 0

```

```

inicio    bsf      status,5 ;se ubica en el segundo banco de RAM
          movlw    0f0h   ;se carga el registro W con 0f
          movwf    trisa   ;se programa el puerto A como salidas
          movlw    0ffh   ;se carga el registro W con ff
          movwf    trisb   ;se programa el puerto B como entradas
          bcf      status,5 ;se ubica en el primer banco de memoria RAM
          clrf    conta   ;inicia contador en cero
          movf    conta,w ;el valor del contador pasa al registro W
          movwf    ptoa    ;pasa el valor de W al puerto A (display)
          call    retardo ;retardo esperando que suelten la tecla

ciclo     btfsc   ptob,0 ;pregunta si el pulsador está oprimido
          goto    pulsa  ;si no lo está continúa revisándolo
          call    retardo ;si está oprimido retarda 100 milisegundos
          btfsc   ptob,0 ;para comprobar
          goto    pulsa  ;si no lo está vuelve a revisar
          incf    conta   ;si lo confirma incrementa el contador
          movf    conta,w ;carga el registro W con el valor del conteo
          xorlw   0ah    ;hace operación xor para ver si es igual a 0ah
          btfsc   status,z ;prueba si el contador llegó a 0ah (diez)
          goto    inicio  ;si es igual el contador se pone en ceros
          goto    ciclo   ;si no llegó a diez incrementa normalmente
          end     ;y actualiza el display

;=====
;----- Fusibles de programación -----
;----- Osc XT
;----- Watchdog OFF
;----- Code protect OFF
;----- Power-Up-Timer ON
;----- Micro. PIC16F84
;-----

```

Figura 2.6. Programa del contador decimal

En la figura 2.8 se muestra el diagrama de flujo para la lectura de un teclado de esta naturaleza; observe como el proceso se queda enclavado mientras no detecta tecla presionada. En la figura 2.9 se muestra el programa respectivo, el cual asigna un valor numérico, comprendido entre 0 y 7, a la tecla presionada. El valor obtenido se lleva a un display de siete segmentos para comprobar que el programa funciona correctamente. El lector podrá determinar que pasa cuando dos o más teclas se presionan "simultáneamente", la prioridad que existe entre ellas y como puede modificarse ésta.

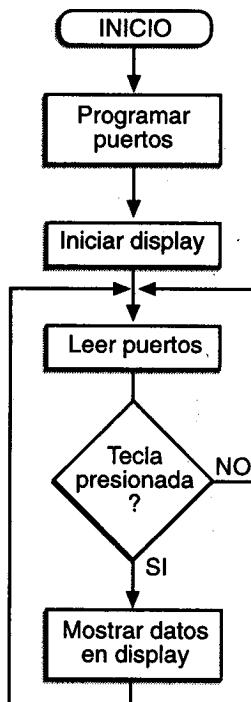


Figura 2.8. Diagrama de flujo para la lectura de un teclado sencillo

```

;Este programa lee un teclado sencillo compuesto por 8 interruptores y maneja
;un display de 7 segmentos.
;En caso de que un número se escriba D'15': significa número decimal
;En caso de que el número se escriba B'00010101': significa número binario
;En caso de que un número se escriba 15H: significa número hexadecimal
;Si no se especifica nada, se supone numeración hexadecimal
;definición de registros
status equ 03h ;registro de estados
ptoA equ 05h ;el puerto A está en la dirección 05 de la RAM
ptoB equ 06h ;el puerto B está en la dirección 06 de la RAM
conta equ 0ch ;contador de rotaciones para identificar la tecla
loops equ 0dh ;utilizado en retardos (milisegundos)
loops2 equ 0eh ;utilizado en retardos
rota equ 0fh ;registro que se rota para encontrar la tecla
trisa equ 85h ;registro de configuración del puerto A
trisb equ 86h ;registro de configuración del puerto B
z equ 02h ;bandera de cero del registro de estados
c equ 00h ;banderaq de carry del registro de estados
w equ 00h - ;indica que el resultado se guarda en W

reset org 0 ;el vector de reset es la dirección 00
goto inicio ;se salta al inicio del programa
org 5 ;el programa empieza en la dirección de memoria 5

retardo ;subrutina de retardo de 100 milisegundos
        movlw D'100' ;el registro loops contiene el número
  
```

```

movwf    loops    ;de milisegundos del retardo
top2     movlw    D'110'   ;
        movwf    loops2   ;
top      nop
        nop
        nop
        nop
        nop
        nop
        A decfsz  loops2   ;pregunta si terminó 1 ms
        goto    top
        decfsz  loops    ;pregunta si termina el retardo
        goto    top2
        retlw    0

inicio   bsf      status,5 ;se ubica en el segundo banco de RAM
        movlw    0f0h    ;se carga el registro W con 0f
        movwf    trisa   ;se programan los pines del puerto A como salidas
        movlw    0ffh    ;se carga el registro W con ff
        movwf    trisb   ;se programan los pines del puerto B como entradas
        bcf      status,5 ;se ubica en el primer banco de memoria RAM

ciclo    movlw    0ffh    ;si no hay tecla oprimida se muestra una F
        movwf    conta   ;en el display
        movf    conta,w  ;el valor del contador pasa al registro W
        ptoa
        call    retardo  ;pasa el valor de W al puerto A (display)
                    ;retardo

leer     movf    ptob,w  ;lee el puerto de los interruptores
        xorlw    0ffh    ;invierte el dato leído
        btfsc    status,z ;pregunta si el resultado de la inversión dió cero
        goto    inicio   ;si no hay tecla oprimida borra display y
                    ;vuelve a leer

sigue   movwf    rota
        clrf
        rrf
        rrr
        btfsc    status,c ;lleva valor de tecla oprimida a registro rota
        goto    salir
        incf
        conta
        goto    sigue
        ;se inicializa el contador de rotaciones
        ;se rota el dato para buscar en que posición
        ;se encuentra el interruptor activado
        ;pregunta si el carry es 1 luego de la rotación
        ;si es uno esa es la tecla oprimida y va a indicar
        ;en el display cual es su valor
        ;si el carry estaba en 0 luego de
        ;rotar el registro
        ;se vuelve a rotar y se vuelve a probar

salir   goto    ciclo
        end
        ;el valor de la tecla queda en el registro conta
        ;y se pasa a W para mostrarlo en el display

;=====;
; Fusibles de programación
; Osc XT
; Watchdog OFF
; Code protect OFF
; Power-Up-Timer ON
; Micro. PIC16F84
;=====;

```

Figura 2.9. Programa para la lectura de un teclado sencillo

Cuando el número de líneas I/O es suficiente, la configuración anterior simplifica el programa y podemos quedar satisfechos; pero cuando las líneas de entrada/salida empiezan a escasear, debemos pensar en rediseñar este teclado, optimizándolo. La figura 2.10 muestra una alternativa para este teclado, observe que para las mismas 8 teclas se tienen sólo 6 líneas de entrada/salida (nos hemos ahorrado 2 líneas, las cuales pueden ser aprovechadas para otros propósitos igualmente importantes); en la figura

2.11 se muestra un teclado de 16 elementos, precisando sólo 8 líneas de entrada/salida (necesitaríamos 8 líneas más si hubiésemos seguido el principio de diseño inicial). Estos dos últimos teclados tienen algo en común, están organizados matricialmente y para manejarlos se requiere el multiplexaje.

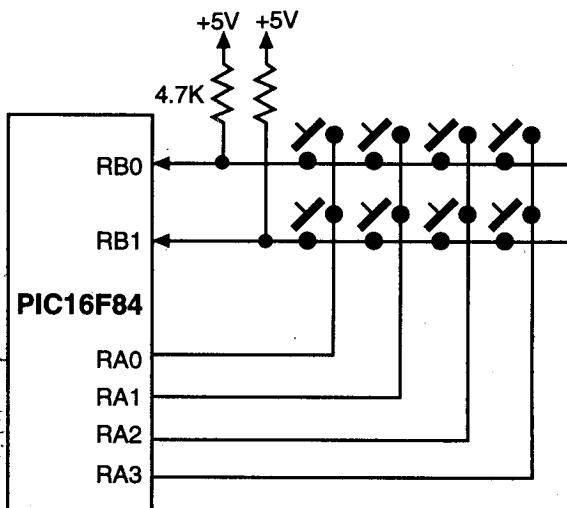


Figura 2.10. Teclado matricial de 2 filas x 4 columnas

Consideremos la figura 2.11, que muestra el teclado matricial de 4 filas por 4 columnas. En este caso, las líneas del microcontrolador correspondientes a las filas se han configurado como salidas y las correspondientes a las columnas como entradas.

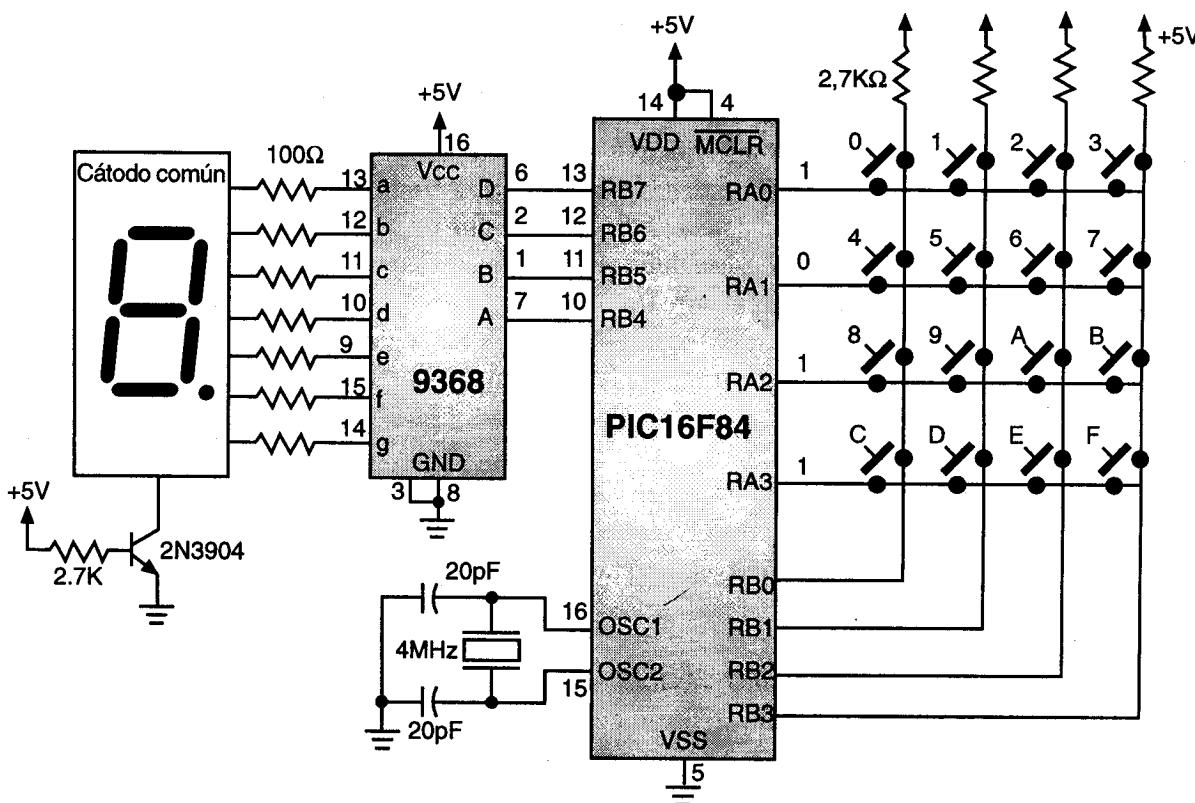


Figura 2.11. Teclado matricial de 4 filas x 4 columnas

Como se puede observar, normalmente, las líneas de entrada permanecen en un nivel lógico alto, gracias a los elementos *pull-up* (resistencias de 2.7K). La clave para manejar este tipo de teclados consiste en enviar por las líneas de salida sólo un cero por vez; por ejemplo si enviamos un cero por la línea RA1, cuando oprimimos una tecla de la segunda fila (el 4, 5, 6 ó 7), un nivel lógico bajo se reflejará en el pin correspondiente de las líneas de entrada (RB0, RB1, RB2 o RB3 respectivamente); así, si se encuentra un nivel lógico bajo en la línea RB3 podemos concluir que la tecla presionada fue el dígito 7.

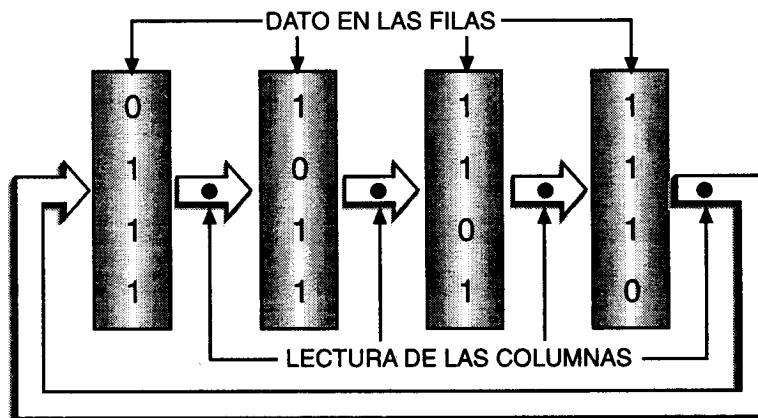


Figura 2.12. Secuencia para la lectura de un teclado matricial

Si queremos explorar todo este teclado, bastará con rotar el cero circularmente, de tal manera que solamente un cero se encuentre en las filas del teclado, cuando se realizan las lecturas de las líneas de entrada (las columnas) como se muestra en la figura 2.12. Cuando el cero llegue a la fila más significativa del teclado, debe reingresar en la próxima ocasión por la menos significativa, reiniciando la exploración del teclado. Un diagrama de flujo para este proceso, en donde el microcontrolador queda enclavado leyendo el teclado hasta que se detecta la presión de uno de sus elementos, se muestra en la figura 2.13; en la figura 2.14 se tiene el programa respectivo. Como resultado del programa, un valor comprendido entre 0 y 15 queda almacenado en un registro, dicho valor se muestra en un display de 7 segmentos que se ha conectado a los pines RB4 a RB7, para comprobar el funcionamiento del sistema.

El proceso se realiza a una gran velocidad, por lo que se tiene la sensación que todo el teclado se está sensando permanentemente. En el programa realizado, por ejemplo, la exploración total del teclado tarda menos de 60 μ s, si consideramos que el oscilador es de 4 MHz.

Otro aspecto que no se puede olvidar, son los rebotes causados por la pulsación de una tecla. Cuando una tecla se oprime, sus contactos actúan como resortes, y la unión eléctrica no es estable; se generan una serie de uniones y desuniones mecánicas durante un intervalo significativo de tiempo. Estos rebotes pueden dar lugar a que, en una aplicación real, el programa los interprete como si se hubieran generado muchas pulsaciones, si es que no se toman los correctivos del caso. Para ello existen

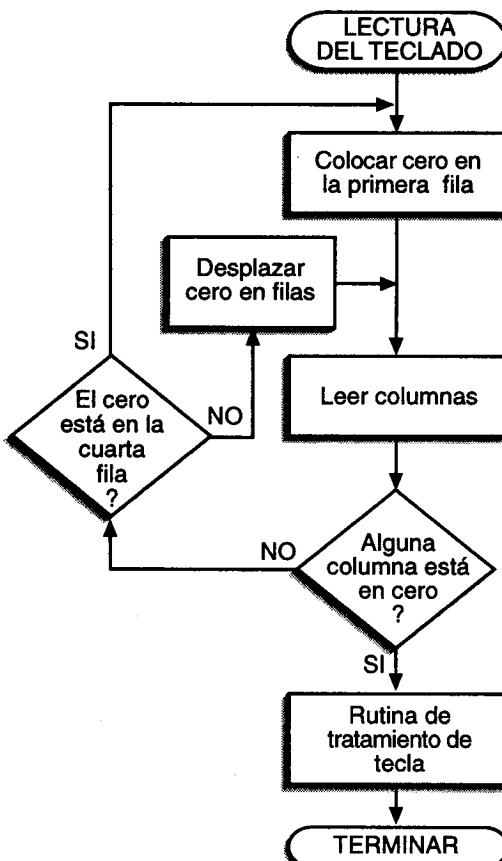


Figura 2.13. Diagrama de flujo para la lectura de un teclado matricial

soluciones de *hardware* y *software*, consideramos más interesantes las segundas ya que simplifican el diseño. Allí, la solución más obvia es que después de la detección de la tecla pulsada se genere un retardo en la lectura del teclado, de tal manera que se ignoren los contactos subsiguientes debidos a los rebotes.

Experimentalmente, se encuentra que un retardo aceptable tiene un valor comprendido entre los 100 a 125 ms; tiempos más pequeños pueden todavía interpretar los rebotes y tiempos más largos pueden tornar demasiado lento un teclado. En ocasiones, conviene también pensar en el tipo de usuarios de un sistema, ya que hay algunos de ellos que tienen la tendencia a mantener oprimida una tecla un tiempo más largo del común de las personas, lo que mal controlado en el programa puede dar lugar a un funcionamiento incorrecto del sistema.

```

;Este programa lee un teclado matricial de 4x4 y muestra la tecla
;oprimida en el display de 7 segmentos.
;definición de registros
pc      equ     02h      ;contador de programa
status  equ     03h      ;registro de estados
ptoap   equ     05h      ;el puerto A está en la dirección 05 de la RAM
ptob    equ     06h      ;el puerto B está en la dirección 06 de la RAM
tecla   equ     0ch      ;contienen el valor de la tecla oprimida
loops   equ     0dh      ;utilizado en retardos (milisegundos)
loops2  equ     0eh      ;utilizado en retardos
rota    equ     0fh      ;registro que rota para enviar unos a las filas
filas   equ     10h      ;contiene el número de la fila a probar
    
```

trisa	equ	85h	;registro de configuración del puerto A
trisb	equ	86h	;registro de configuración del puerto B
z	equ	02h	;bandera de cero del registro de estados
c	equ	00h	;bandera de carry del registro de estados
w	equ	00h	;indica que el resultado se guarda en W
reset	org	0	;el vector de reset es la dirección 00
	goto	inicio	;se salta al inicio del programa
	org	5	;el programa empieza en la dirección 5
retardo	;subrutina de retardo de 100 milisegundos		
	movlw	D'100'	;el registro loops contiene el número
	movwf	loops	;de milisegundos del retardo
top2	movlw	D'110'	;
	movwf	loops2	;
top	nop		
	decfsz	loops2	;pregunta si terminó 1 ms
	goto	top	
	decfsz	loops	;pregunta si termina el retardo
	goto	top2	
	retlw	0	
tabla	addwf	pc	;sumar W al PC
	nop		
	retlw	0	;primera columna
	retlw	1	;segunda columna
	nop		
	retlw	2	;tercera columna
	nop		
	nop		
	retlw	3	;cuarta columna
inicio	bsf	status,5	;se ubica en el segundo banco de RAM
	movlw	0f0h	;se carga el registro W con 0f0h
	movwf	trisa	;se programa el puerto A como salidas
	movlw	0fh	;se carga el registro W con 0fh
	movwf	trisb	;se programa el puerto B como entradas y salidas
	bcf	status,5	;se ubica en el primer banco de memoria RAM
	movlw	00h	;para empezar se muestra un 0 en el display
ciclo	movwf	tecla	;
	swapf	tecla,w	;intercambia 4 bits altos y bajos y quedan en W
	movwf	ptob	;pasa el valor de W al puerto B (display)
	call	retardo	;retardo
escan	clrfl	filas	
	movlw	b'1110'	;se prepara para enviar ceros a las filas
	movwf	rota	
probar	movf	rota,w	;envia el dato a las filas
	movwf	ptoal	
leer	nop		
	movf	ptob,w	;tiempo para estabilidad de las líneas
	andlw	0fh	;leer las columnas conectadas al puerto B
	xorlw	0fh	;elimina la parte alta del byte leído
	btfs	status,z	;invierte el dato para ver si hay algún cero
	goto	salir	;pregunta si el resultado es cero (alguna tecla)
	btfs	rota,3	;si hay tecla, mostrar en display
	goto	escan	;consulta si ya van 4 rotaciones
	bsf	status,c	;si terminó, vuelve a empezar el escan de teclado
	rlf	rota	;coloca bit de carry en 1
	movlw	4	;para rotar el 0 que va a ir hacia las filas
	addir	filas,1	;carga W con 4 para sumarlo al valor de filas
	goto	probar	;va a hacer la próxima prueba con el 0 rotado
	call	tabla	;para obtener valor de la columna
salir	addir	filas,w	;sumar columna y filas para obtener el dato real

```

        movwf    tecla      ;muestra el dato en display
        goto     ciclo
        end

;-----[Fusibles de programación]-----
; Osc          XT
; Watchdog     OFF
; Code protect OFF
; Power-Up-Timer ON
; Micro.       PIC16F84
;
```

Figura 2.14. Programa para la lectura de un teclado matricial

Multiplexaje de displays de siete segmentos

Consideremos la estructura de la figura 2.15. Allí se tienen cuatro displays de siete segmentos, con punto decimal; un puerto de 4 bits (o la mitad de uno de 8) controla cuatro transistores NPN, que finalmente alimentan los cátodos de cada uno de los displays, en donde el bit menos significativo controla el display de menor peso. Un puerto de 8 bits maneja cada uno de los segmentos de los displays, en donde el bit menos significativo del puerto controla el segmento **a**, el siguiente el **b**, y así sucesivamente, hasta llegar al más significativo, que controla el punto decimal del display. Las resistencias tienen como objeto limitar la corriente que fluye a través de los segmentos y que ingresa a los pines del microcontrolador.

Por la configuración, es fácil deducir que cuando se tiene un nivel lógico bajo en la base del transistor éste se comporta como un interruptor abierto y no se presenta corriente entre emisor y colector; se necesita tener un uno lógico en

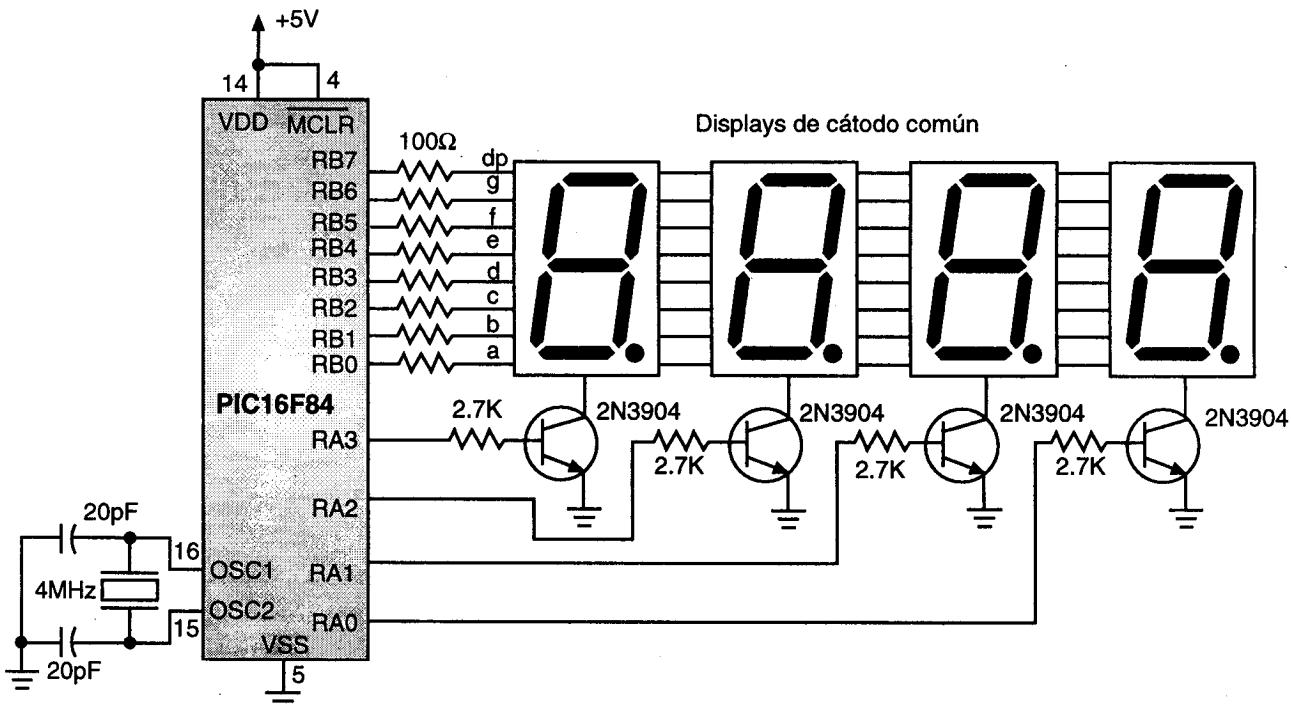


Figura 2.15. Configuración para manejo de displays de 7 segmentos

las bases de éstos para que el transistor se comporte como un interruptor cerrado y se presente dicha corriente. Pero aún cuando los transistores se comporten como interruptores cerrados, no se encenderá ningún display si la salida del puerto que maneja los segmentos tiene niveles lógicos bajos (ceros); para encenderlo, es necesario colocar un nivel lógico alto en el pin de salida correspondiente a cada segmento.

Así por ejemplo, si queremos mostrar un tres en el segundo display menos significativo debemos establecer básicamente los siguientes pasos:

- Colocar el número binario 0010 en el puerto que controla los transistores
- Enviar el número binario 01001111 por el puerto que controla los segmentos

Un buen observador encontrará que aunque los anteriores pasos consiguen el objetivo propuesto (mostrar en la pantalla el —3-, en donde la línea quiere decir espacios en blanco), este proceso puede conllevar efectos indeseados. Aquí, por ejemplo, si el puerto que controla los segmentos tenía un valor diferente del binario 00000000 o de 01001111, entre la ejecución de los dos pasos dados se mostrará en el display un número o un símbolo que es diferente del valor deseado. Por ello, es conveniente apagar momentáneamente los segmentos, de tal manera que nos permita seleccionar adecuadamente el display en cuestión y posterior a esto, enviar el dato correcto a los segmentos; por lo tanto, un paso 0 que se debe agregar a este proceso es colocar el número binario 00000000 en el puerto que controla los segmentos.

Otra opción que permite obtener el mismo resultado, con un proceso diferente, sería la siguiente:

0. Colocar el número binario 0000 en el puerto que controla los transistores
1. Enviar el número binario 01001111 por el puerto que controla los segmentos
2. Colocar el número binario 0010 en el puerto que controla los transistores

Ambos procesos conllevan al mismo objetivo propuesto, eliminando la posibilidad de los efectos indeseados.

Ahora, si pretendemos visualizar no uno sino los cuatro displays de siete segmentos, es necesario empezar a controlar los transistores secuencialmente a la vez que se envía por el puerto que controla los segmentos los datos correspondientes al display en cuestión, realizando este proceso a una velocidad tal que de nuevo parezca que el proceso se está realizando simultáneamente sobre todos los displays. El tiempo en que necesitamos sostener el dato en cada display puede variar significativamente, dependiendo fundamentalmente del valor de las resistencias limitadoras, del número de dígitos que se tengan por mostrar y de las características propias del display; experimentalmente se encuentra que mostrar cada dígito durante 3 milisegundos, cuando se tienen resistencias limitadoras de 100 ohm, proporcionan un brillo aceptable de un display “estándar” y una buena visualización a una distancia prudente.

El diagrama de flujo de la figura 2.16 muestra el proceso necesario para mostrar cuatro dígitos en un display y la figura 2.17 muestra el respectivo programa, el cual acude a tablas para consultar los segmentos que se deben encender en cada caso. La utilización de estas tablas permiten que se pueda alambrar de manera diferente las salidas del microcontrolador y las entradas de los segmentos del display; por ejemplo, se puede reorganizar la configuración de pines de tal manera que se simplifique el diseño del circuito impreso, bastando con reorganizar los valores de la tabla. La utilización de las tablas también nos permiten la generación de caracteres y signos especiales, ya que podemos controlar el encendido de todos y cada uno de los siete segmentos y el punto decimal.

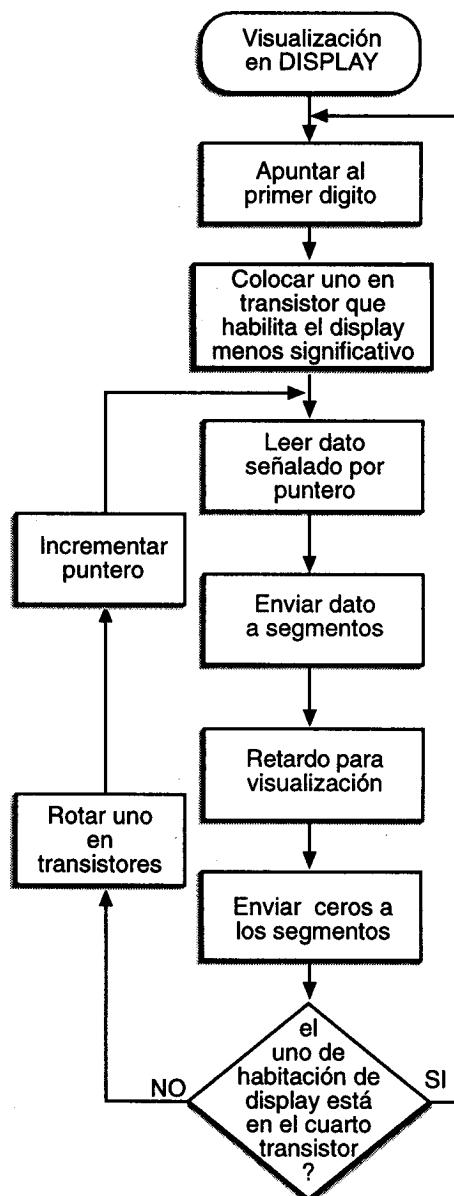


Figura 2.16. Diagrama de flujo para manejo de 4 displays sin decodificador

```

;Este programa maneja 4 displays de 7 segmentos
;en forma multiplexada.
;definición de registros
indo    equ    00h      ;registro de direccionamiento indirecto
pc      equ    02h      ;contador de programa
status  equ    03h      ;registro de estados
fsr     equ    04h      ;registro selector
ptoa   equ    05h      ;el puerto A está en la dirección 05 de la RAM
ptob   equ    06h      ;el puerto B está en la dirección 06 de la RAM
loops  equ    0dh      ;utilizado en retardos (milisegundos)
loops2 equ    0eh      ;utilizado en retardos
rota   equ    0fh      ;rota el uno para habilitar displays
dig1   equ    10h      ;primer dígito a mostrar
dig2   equ    11h      ;segundo dígito a mostrar
dig3   equ    12h      ;tercer dígito a mostrar
dig4   equ    13h      ;cuarto dígito a mostrar
trisa  equ    85h      ;registro de configuración del puerto A
trisb  equ    86h      ;registro de configuración del puerto B
z      equ    02h      ;bandera de cero del registro de estados
c      equ    00h      ;bandera de carry del registro de estados
w      equ    00h      ;indica que el resultado se guarda en W

reset  org    0        ;el vector de reset es la dirección 00
      goto  inicio   ;se salta al inicio del programa

      org    5        ;el programa empieza en la dirección de memoria 5

retardo ;subrutina de retardo de 3 milisegundos
movlw  03h      ;el registro loops contiene el número
movwf  loops    ;de milisegundos del retardo
top2   movlw  D'110' ;
      movwf  loops2   ;
top    nop
      nop
      nop
      nop
      nop
      nop
decfsz loops2   ;pregunta si termino 1 ms
goto   top
decfsz loops    ;pregunta si termina el retardo
goto   top2
retlw  0

tabla ;contiene los valores para encender segmentos en display de
;código común (cuando no se utiliza decodificador 9368)
addwf pc       ;sumar W al PC
;segmento gfedcba ;orden de los bits
retlw b'00111111' ;0
retlw b'00000110' ;1
retlw b'01011011' ;2
retlw b'01001111' ;3
retlw b'01100110' ;4
retlw b'01101101' ;5
retlw b'01111101' ;6
retlw b'00000111' ;7
retlw b'01111111' ;8
retlw b'01101111' ;9

inicio bsf    status,5 ;se ubica en el segundo banco de RAM
      movlw  00h      ;se carga el registro W con 0f0h
      movwf  trisa    ;se programan los pines del puerto A como salidas
      movlw  00h      ;se carga el registro W con 00h
      movwf  trisb    ;se programa el puerto B como entradas y salidas
      bcf    status,5 ;se ubica en el primer banco de memoria RAM

      movlw  01      ;datos que se muestran en los displays
      movwf  dig1
      movlw  02

```

```

        movwf    dig2
        movlw    03
        movwf    dig3
        movlw    04
        movwf    dig4

        movlw    00      ;envía ceros a los transistores para apagarlos
        movwf    ptoa
        empe    movlw    08h    ;iniciar un 1 en el registro de rotación
        movwf    rota
        disp    movlw    dig1    ;con el registro selector (fsr) se apunta
        movwf    fsr     ;al primer dato que se va a mostrar
        movlw    00h    ;colocar en cero el dato del display
        movwf    ptob
        movf     rota,w   ;para apagarlos
        movwf    ptoa
        movf     indo,w   ;lee el dato del registro apuntado por el fsr
;call    tabla    ;se utilizaría si no hubiera decodificador 9368
        movwf    ptob
        call     retardo  ;retardo de 3 milisegundos para visualización
        btfsc   rota,0    ;pregunta si terminaron las 4 rotaciones
        goto    empe    ;si ya rotaron todos, vuelve a empezar
        bcf    status,c   ;pone el carry en 0 para que no afecte rotaciones
        rrf     rota
        incf   fsr     ;rota el 1 habilitador de displays
        goto    disp    ;apunta al próximo dígito a mostrar

        end

=====
;          Fusibles de programación
;          Osc           XT
;          Watchdog      OFF
;          Code protect   OFF
;          Power-Up-Timer ON
;          Micro.        PIC16F84
=====

```

Figura 2.17. Programa para manejo de 4 displays sin decodificador

Una variante para este sistema consistiría en utilizar un decodificador 9368 para manejar los displays de 7 segmentos, en este caso no se requiere la lectura de tablas y el dato correspondiente se puede llevar al puerto directamente. Debe recordarse que con este sistema se pueden mostrar números hexadecimales (entre 0 y F). El diagrama de conexión se muestra en la figura 2.18. El programa es el mismo de la figura 2.17, sólo se debe omitir la subrutina llamada TABLA y el sitio donde se hace el llamado (call TABLA), esto debido a que el dato se puede mostrar directamente.

Manejo simultáneo de teclados y displays

Una estructura más completa es la que se muestra en la figura 2.19. Observe como se utilizan doce líneas para implementar un teclado y la visualización en siete segmentos. Aquí, se está utilizando el decodificador 7447; de esta manera se ahorran al menos cuatro líneas (sólo tres si utilizamos una adicional para manejar el punto decimal); un puerto de 8 bits se comporta totalmente como salida, controlando simultáneamente los datos que se mostrarán por los displays, los que saldrán por las filas y las bases de los transistores, mientras que un puerto de cuatro bits se comporta como entrada, para leer las columnas del teclado. En este caso se utilizan displays de ánodo común y transistores PNP, por lo tanto, el cero que se rota para leer el teclado matricial sirve también para encender los displays.

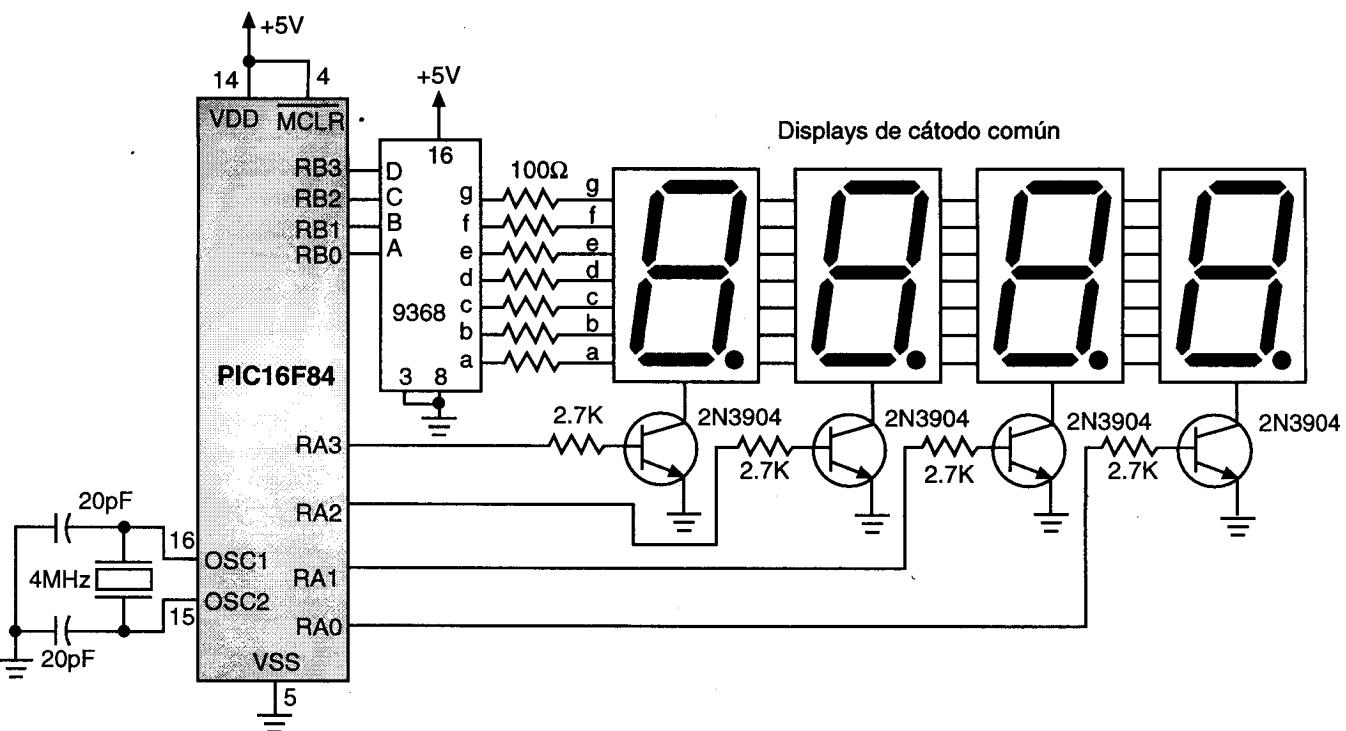


Figura 2.18. Conexión para el manejo de 4 displays con decodificador

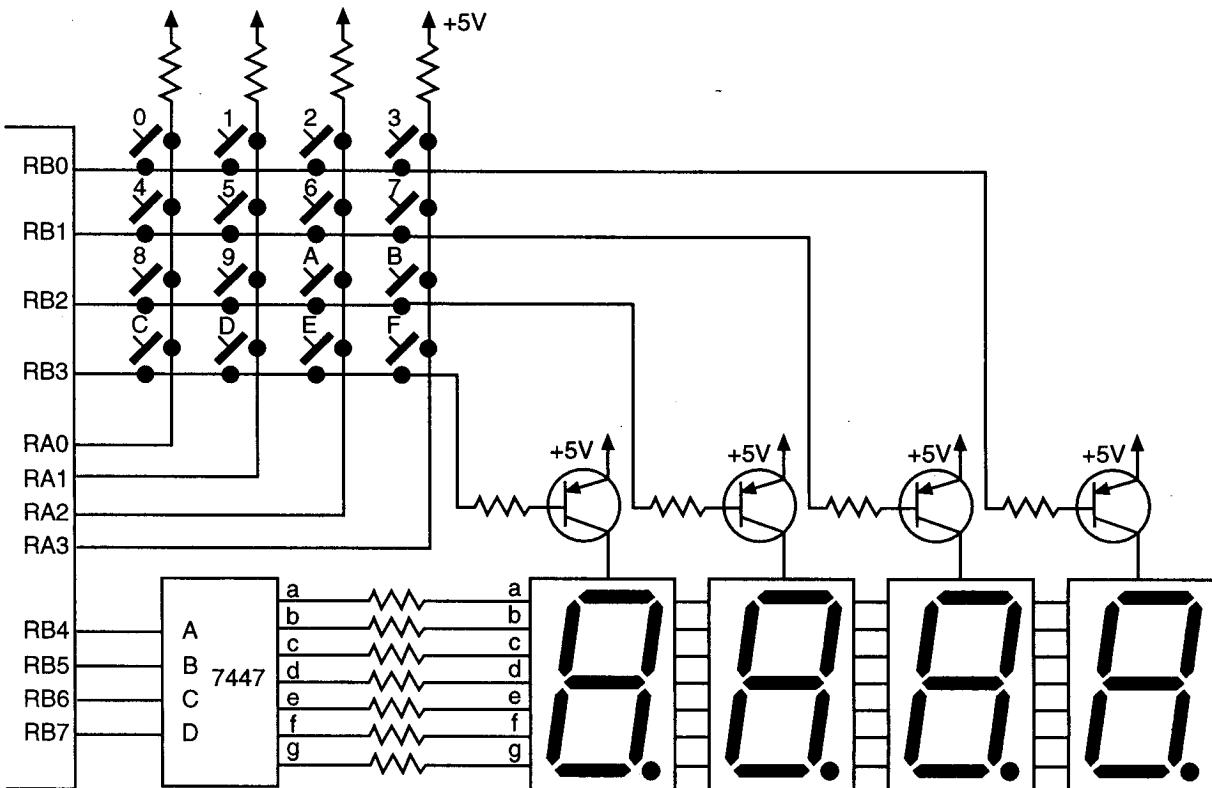


Figura 2.19. Multiplexaje de teclado y display al mismo tiempo

Aquí se resumen los anteriores ejemplos para crear uno sólo. El disparo del transistor es simultáneo con la salida del dato para activar cada segmento dentro del display, así que no es necesario preocuparse con apagar previamente los LED, antes de rotar el cero en las bases de los transistores.

Aquí dejamos al experimentador el desarrollo de los programas necesarios para leer el teclado y visualizar de manera “simultánea” la información en el display; puede tomar los programas anteriores como base. Con la práctica encontrará que puede mejorar estas rutinas y configuraciones, logrando minimizar el diseño y los costos, y maximizar el rendimiento.

Proyecto N° 4: Conexión de memorias seriales al PIC

Las técnicas para almacenar información en medios electrónicos se perfeccionan más cada día. A diario vemos ejemplos de su utilización en nuestros hogares y oficinas, por ejemplo, en receptores de televisión, reproductores de *compact disc*, sistemas de control remoto, impresoras, fotocopiadoras, teléfonos celulares, etc. Una de estas tecnologías corresponde a las llamadas memorias EEPROM seriales, las cuales tienen grandes ventajas si se comparan con otras posibilidades. Entre sus principales características se cuentan:

- Se pueden conectar fácilmente con microprocesadores o microcontroladores, inclusive algunos de ellos tienen pines dedicados para esta labor.
- Transferencia de datos de manera serial, lo que permite ahorrar pines del micro para dedicarlos a otras funciones.
- Ocupan la décima parte del espacio de las memorias que trabajan en paralelo, esto permite ahorrar dinero debido al menor tamaño del circuito impreso.
- El consumo de corriente es mucho menor que en las memorias que trabajan en paralelo, esto las hace ideales para sistemas portátiles que funcionan con baterías.

El objetivo de esta práctica es mostrar los aspectos más importantes de su tecnología y enseñar conceptos básicos para su utilización en circuitos reales, se basa en las memorias que tienen comunicación a 2 hilos empleando la interface I²C, cuyas referencias más conocidas son 24LC01/02/04/16. La velocidad de transferencia de información para estos dispositivos es de 100 ó 400 kHz (aunque el límite lo impone el protocolo I²C más no la tecnología del dispositivo). Como característica importante de este elemento se tiene la inmunidad al ruido, dado que este integrado tiene filtros en los pines de comunicación.

Memorias 24XX

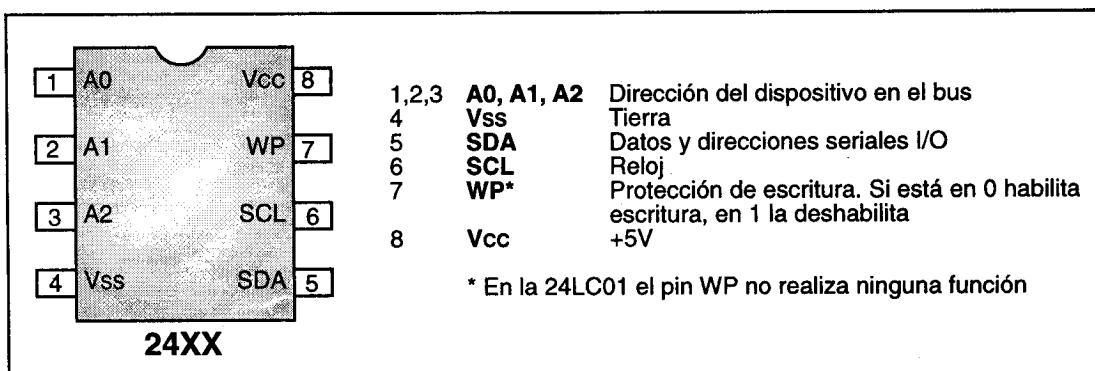


Figura 2.20. Configuración de pines de la memoria 24LCXX

Estas memorias utilizan el bus de 2 hilos para comunicarse con otros dispositivos. Dado que cumplen con el protocolo I²C, tiene un pin llamado SCL que recibe los pulsos generados por el dispositivo maestro (o sea el microcontrolador) y otro llamado SDA que maneja el flujo de datos de forma bidireccional (entrada/salida). En la figura 2.20 se muestra el diagrama de pines correspondiente a estas memorias.

Este dispositivo no requiere de un pin habilitador o *chip select*, ya que en este esquema la transferencia de información solo se puede iniciar cuando el bus esté libre. En este caso, como cada dispositivo tiene su dirección determinada mediante los pines A0, A1 y A2; solamente responderá la memoria cuya dirección coincida con la dirección que va encabezando la trama de información. En la figura 2.21 se muestra la capacidad de almacenamiento de estos dispositivos y las posibilidades de direccionamiento que tienen.

Referencia	Capacidad en K bits	Bloques internos	A0	A1	A2	Dispositivos en el bus
24LC01B, 24C01	1	1	1 6 0	1 6 0	1 6 0	8
24LC02B, 24C02	2	1	1 6 0	1 6 0	1 6 0	8
24LC04B, 24C04	4	2	X	1 6 0	1 6 0	4
24LC08B	8	4	X	X	1 6 0	2
24LC16B	16	8	X	X	X	1

Figura 2.21. Capacidad de memoria y direccionamiento de las memorias 24LCXX

Transferencia de la información. Cuando el microcontrolador desea establecer comunicación con la memoria, debe enviarle una serie de bits que llevan la siguiente información:

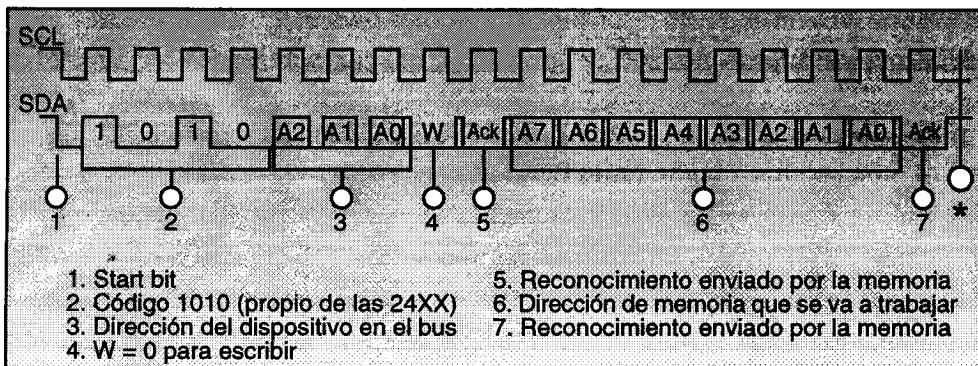
1. Se envía el bit de arranque o *start bit*
2. El código 1010 (propio de estas memorias)
3. La dirección del dispositivo (A2, A1, A0)
4. Un bit que indica que se desea escribir ('0') en la memoria

Luego de esto, la memoria debe enviar un reconocimiento para informarle al microcontrolador que recibió la información. Dicho asentimiento, llamado *ACK*, consiste en poner el bus en un nivel bajo (lo hace la memoria). Después el microcontrolador debe enviar los bits que corresponden a la posición de memoria que se quiere leer o escribir; nuevamente la memoria envía un reconocimiento. El paso siguiente depende de la operación que se vaya a ejecutar.

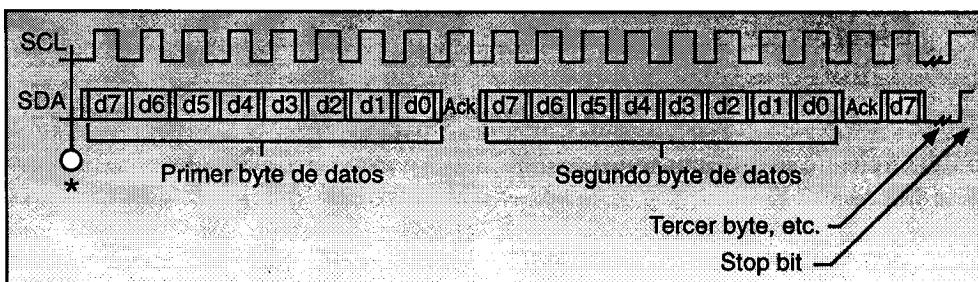
Si se trata de un proceso de escritura, el microcontrolador solo debe enviar el dato a ser almacenado y esperar el asentimiento por parte de la memoria para confirmar que llegó correctamente. Si se trata de una lectura, nuevamente se debe repetir los primeros cuatro pasos, solo que en lugar de un "0" que indica escritura, se debe enviar un "1" que indica lectura. Después se espera el asentimiento y acto seguido se puede leer el byte con el dato que estaba en la posición de memoria que se indicó anteriormente. Cuando se termina la operación, el microcontrolador debe enviar una señal de parada o *stop bit*. En la figura 2.22 se muestra el diagrama de tiempos correspondiente a todo el proceso descrito anteriormente.

Ejemplo de aplicación

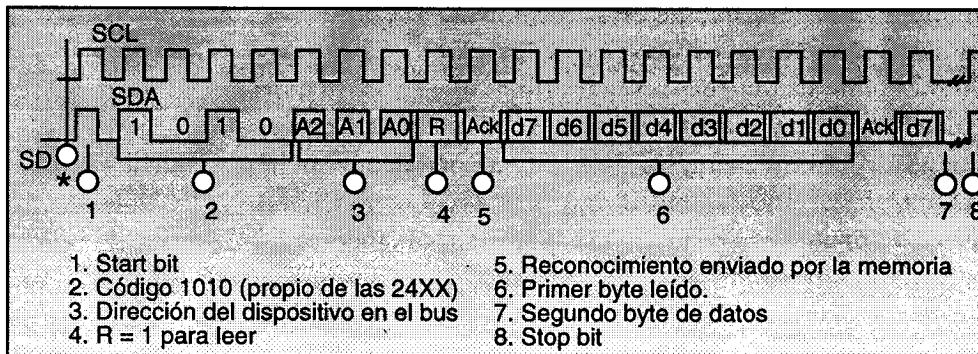
El ejercicio consiste en hacer un contador de 0 a 9 con un interruptor pulsador y un display de siete segmentos, similar al ejercicio de la figura 2.4. La diferencia radica en que el número que se muestra en el display se va a almacenar simultáneamente en una memoria 24LC01 (LC quiere decir que puede trabajar desde 2 voltios). Se va a utilizar un microcontrolador PIC16F84 (aunque se puede utilizar un 16C61 o 16C71).



A. Forma de direccionar la memoria 24XX



B. Escritura de byte



C. Lectura de byte

Figura 2.22. Diagrama de tiempos para la lectura y la escritura en una memoria 24LCXX

En la figura 2.23 se muestra el diagrama esquemático del circuito. En este caso los pines de dirección de la memoria se conectaron a tierra, al igual que el pin WP. La resistencia de 4.7 kohm conectada al pin SDA es necesaria dado que dicho pin tiene salida de colector abierto (*open collector*). El display se conecta al puerto A y el pulsador al pin RB0.

El programa que se escribe en el microcontrolador se muestra en la figura 2.24, su función principal es llevar el control del conteo decimal y almacenar en la memoria el mismo dato que se envía al display.

En el programa, la subrutina *WAIT* produce un retardo en milisegundos, la cantidad de milisegundos deseada debe escribirse en el registro *loops* antes de hacer el llamado correspondiente. Se utiliza principalmente para hacer un retardo de 10 mi-

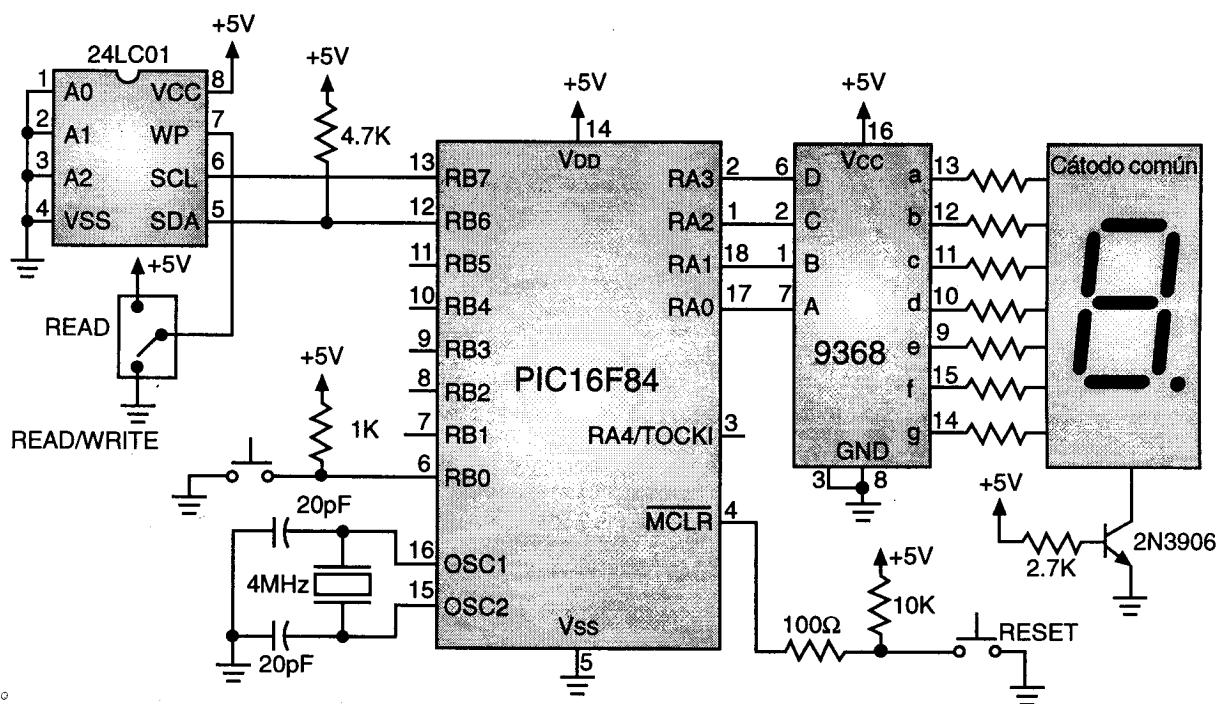


Figura 2.23. Diagrama esquemático del contador con PIC y memoria 24LC01

```
;Este programa realiza un contador decimal con un pulsador y un display de 7
;segmentos, el valor del conteo se guarda en la memoria serial 24LC01
;definición de bits
status equ 3h ;registro de estados
ptoaequ 5h ;
ptobequ 6h ;
addr equ 0dh ;posición de memoria que se lee o escribe
dataoequ 0eh ;registro para escribir datos en la memoria
slave equ 0fh ;dirección del dispositivo en el bus I2C (1010xxx0)
txbuf equ 10h ;buffer de transmisión
count equ 11h ;contador de bits
eeprom equ 12h ;buffer de bits
rxbuf equ 13h ;buffer de recepción
loops equ 15h ;se utilizan en retardos
loops2 equ 16h ;
di equ 7 ;bit de entrada desde eeprom
do equ 6 ;bit de salida para eeprom
sdata equ 6 ;línea de datos seriales (pin RB6)
sclk equ 7 ;reloj serial (pin RB7)
conta equ 17h ;lleva el conteo de pulsaciones
conta2 equ 18h ;
trisa equ 85h ;registro de configuración del puerto A
trisb equ 86h ;registro de configuración del puerto B
z equ 02h ;bandera de cero del registro de estados
w equ 00h ;indica que el resultado se guarda en W
c equ 00h ;bandera de carry

        org 00h ;Vector de reset
        goto INICIO
        org 03h

WAIT
top2    movlw .110 ;subrutina de retardo en milisegundos
        movwf loops2 ;el numero de milisegundos llega
top     nop ;cargado en el registro loops
        nop
```

```

nop
nop
nop
nop
decfsz loops2 ;pregunta si termino 1 ms
goto top
decfsz loops ;pregunta si termina el retardo
goto top2
retlw 0

retardo movlw .100 ;retardo de 100 milisegundos
movwf loops
call WAIT
retlw 0

BSTART ;Esta rutina genera el start bit para la comunicacion serial
movlw b'00111111'
tris ptob ;programar datos y reloj como salidas
bcf ptob,sclk ;linea de reloj en nivel bajo
bsf ptob,sdata ;se asegura linea de datos en alto
nop
bsf ptob,sclk ;linea de reloj en alto
nop
nop ;ajuste de tiempo
nop
nop
nop
nop
bcf ptob,sdata ;se baja la linea de datos
nop ;mientras el reloj està alto
nop
nop ;ajuste de tiempo
nop
nop
bcf ptob,sclk ;se baja la linea de reloj
nop ;para terminar el pulso
nop
retlw 0

BSTOP ;Esta rutina genera el stop bit para la comunicaciòn serial
movlw b'00111111'
tris ptob ;programa reloj y datos como salidas
bcf ptob,sdata ;asegura linea de datos en bajo
nop
nop
nop
bsf ptob,sclk ;linea de reloj en nivel alto
nop
nop
nop
bsf ptob,sdata ;la linea de datos pasa a nivel alto
nop ;mientras el reloj està alto
nop
bcf ptob,sclk ;la linea de reloj baja nuevamente
nop ;para completar el pulso
nop
nop
retlw 0

BITOUT ;Esta rutina toma el bit que se debe transmitir y lo saca al puerto
movlw b'00111111' ;ademas genera el pulso de reloj
tris ptob ;programa reloj y datos como salidas
bsf ptob,sdata ;asume que el bit es alto
btfs s eeprom,do ;pregunta estado del bit a transmitir
bcf ptob,sdata ;si el bit es bajo pone la salida en bajo
clkout
nop
nop
bsf ptob,sclk ;sube el nivel de la linea de reloj
nop
nop ;para formar el pulso

```

```

nop
nop
bcf    ptob,sclk ;termina pulso de reloj
retlw  0

BITIN   ;Esta rutina lee un bit de la memoria y lo pone en un registro
bsf    eeprom,di ;asume que el bit es de nivel alto
movlw b'01111111' ;programa pin de datos como entrada
tris   ptob
bsf    ptob,sclk ;sube la linea del reloj
nop
nop
nop
nop
nop
nop
nop
nop
btfss  ptob,sdata ;
bcf    eeprom,di ;si es bajo lo pone en ese nivel
bcf    ptob,sclk ;si es alto lo deja como se asumió antes
retlw  0

TX      ;Esta rutina se encarga de transmitir un byte hacia la memoria
movlw .8
movwf  count      ;el número de bits es 8
TXLP    bcf    eeprom,do ;asume que el bit a enviar es bajo
          btfsc txbuf,7 ;consulta el estado real del bit
          bsf   eeprom,do ;si era alto lo deja con dicho nivel
          call  BITOUT   ;saca el bit por el puerto
          rlf   txbuf,1 ;rota el byte que se está transmitiendo
          decfsz count   ;pregunta si ya pasaron los 8 bits
          goto  TXLP    ;si no ha terminado sigue transmitiendo
          call  BITIN    ;espera el reconocimiento enviado por la
          retlw 0         ;memoria (ACK)

RX      ;Esta rutina recibe un byte y lo entrega en el registro rdbuf
clrf   rdbuf       ;borra el buffer de entrada
movlw .8
movwf  count      ;indica que recibe 8 bits
RXLP    bcf    status,0 ;Borra el carry
          rlf   rdbuf, F ;rota a la izquierda
          call  BITIN    ;lee un bit
          btfsc eeprom,di
          bsf   rdbuf,0 ;si es necesario pone el bit en uno
          decfsz count   ;pregunta si completo 8 bits
          goto  RXLP    ;sino, recibe otro bit
          bsf   eeprom,do ;envia el ACK de asentimiento
          call  BITOUT   ;para terminar
          retlw 0

LEER    ;Esta rutina recibe la dirección que se
;desea LEER y devuelve el dato que tiene grabado
call  BSTART   ;genera el start bit
nop
nop
bcf   slave,0  ;selecciona la memoria
movf  slave,w  ;y selecciona modo de escritura
movwf txbuf
call  TX        ;envía esos datos a la memoria
movf  addr,w
movwf txbuf
call  TX        ;envía la posición de memoria a ser leída
nop
nop
call  BSTART   ;ahora se selecciona nuevamente la memoria
nop
nop
call  BSTART   ;y se le indica modo de lectura
nop
nop
bsf   slave,0  ;genera start bit
;indica que se va a LEER

```

	movf slave,w	;selecciona el dispositivo
	movwf txbuf	;
	call TX	;envia esa información a la memoria
	nop	
	call RX	;la memoria entrega el byte de esa dirección
	bsf eeprom,do	;envia el ACK de reconocimiento
	call BITOUT	
	call BSTOP	;se envia el stop bit para finalizar comunicación
	retlw 0	
ESCRIB	;Esta rutina escribe un dato en la posición ;de memoria que se le indique en el registro addr	
	call BSTART	;genera el start bit
	nop	;
	nop	;
	bcf slave,0	;selecciona la memoria
	movf slave,w	;y selecciona modo de escritura
	movwf txbuf	;
	call TX	;envia esos datos a la memoria
	movf addr,w	;
	movwf txbuf	;envía la posición de memoria a ser grabada
	call TX	;ahora se selecciona nuevamente la memoria
	nop	;y se le indica modo de lectura
	nop	
	movf datao,w	;toma el dato que va a ser grabado
	movwf txbuf	;y lo envía
	call TX	
	call BSTOP	
	movlw .10	;retardo de 10 ms al escribir
	movwf loops	;cada dato
	call WAIT	
	retlw 0	
INICIO	bsf status,5	;se ubica en el segundo banco de RAM
	movlw 0f0h	;se carga el registro W con 0f
	movwf trisa	;se programa el puerto A como salidas
	movlw 07fh	;se carga el registro W con 00
	movwf trisb	;se programa el puerto B como entradas
	bcf status,5	;se ubica en el primer banco de memoria RAM
	movlw b'10100000'	;La dirección A0, A1, y A2 de la memoria
	movwf slave	;en el bus I2C es 000
	clrf addr	;cuando se enciende el sistema se verifica que
	call LEER	;el dato guardado en memoria esté entre 0 y 9
	movlw 0ah	;la prueba se hace porque la primera vez que
	subwf rxbuf,w	;se encienda el sistema se puede tener un
	btfss status,c	número fuera del rango
	goto ciclo	;para las ocasiones posteriores no importa
ini2	clrf conta	;inicia contador en cero
	clrf datao	
ciclo	call ESCRIB	;inicia dato de memoria en 0
	call LEER	;LEER memoria, devuelve dato en W
	movf rxbuf,w	;pasa el valor de W al puerto A (display)
	movwf conta	
	movwf ptoa	
pulsa	call retardo	;retardo esperando que suelten la tecla
	btfsc ptob,0	;pregunta si el pulsador está oprimido
	goto pulsa	;si no lo está continúa revisándolo
	call retardo	;si está oprimido retarda 100 milisegundos
	btfsc ptob,0	;para comprobar
	goto pulsa	;si no lo está vuelve a revisar
	incf conta	;si lo confirma incrementa el contador
	movf conta,w	;carga el registro W con el valor del conteo
	movwf datao	;el dato del conteo lo guarda en memoria
	call ESCRIB	;para recuperarlo en caso de un apagón
	movf conta,w	
	xorlw 0ah	;hace operación xor para ver si es igual a 0ah
	btfss status,z	;prueba si el contador llegó a 0ah (diez)
	goto ciclo	;si no es igual se incrementa normalmente
	goto ini2	;

```

        end           ;
;-----;
;      Fusibles de programación
;      Osc          XT
;      Watchdog    OFF
;      Code protect OFF
;      Power-Up-Timer ON
;      Micro.       PIC16F84
;-----;

```

Figura 2.24. Programa del contador con PIC y memoria 24LC01

liseundos luego de escribir un dato en la memoria. Se debe tener en cuenta que los retardos están calculados para un oscilador de 4 MHz en el microcontrolador.

La subrutina *BSTART* genera el bit de inicio de la comunicación, con la temporización y estado de los pines adecuados. Por su parte, la subrutina *BSTOP* hace lo mismo con el bit de parada o de fin de la comunicación. La subrutina *BITOUT* toma el bit de dato que se debe transmitir y lo envia hacia la memoria, se encarga de programar el pin del microcontrolador como salida y de generar el pulso de reloj necesario para la sincronización. La rutina *BITIN* hace su parte cuando se está leyendo un bit enviado por la memoria, genera el pulso de reloj y pasa el bit leído al registro o buffer de entrada. Las rutinas *TX* y *RX* se encargan de transmitir y recibir un byte completo de datos, cada una hace 8 llamados seguidos a las rutinas *BITOUT* y *BITIN* respectivamente.

La rutina *LEER* recibe en el registro *addr* la posición de memoria que se debe leer y genera todas las señales necesarias (incluyendo el start y el stop bit) para obtener el dato que en ella se encuentra grabado, al final devuelve el dato que recibió de la memoria en el registro *rxbuf*. La rutina *ESCRIB* toma el dato contenido en el registro *datao* y lo escribe en la posición de la memoria que está direccinada en el registro *addr*.

Cada vez que se enciende el sistema el microcontrolador lee el dato que se encuentra en la primera posición de memoria y lo pasa al display. Cuando el pulsador sea oprimido se debe incrementar dicho dato y se actualiza el display al tiempo que se vuelve a almacenar ese número en la memoria. Un caso especial ocurre cuando se enciende el sistema por primera vez, como el dato que se encuentra grabado en la memoria es desconocido y podría ser superior a 9, este se debe probar y si se encuentra que es mayor, se borra y se empieza el conteo en 0.

Las rutinas que permiten leer y escribir en la memoria 24LC01se pueden utilizar como parte de cualquier programa sin que se tengan contratiempos, sólo se debe tener en cuenta que las temporizaciones están calculadas para un oscilador de 4 MHz. Con las rutinas *LEER* y *ESCRIB* se tiene una velocidad de transferencia de información de aproximadamente 60 kHz.

Una prueba que es muy interesante consiste en cambiar de posición el interruptor que selecciona la protección de escritura, cuando está en la posición READ/WRITE se puede incrementar el contador normalmente, cuando se encuentra en la posición READ el contador no se incrementa debido a que la memoria está protegida contra escritura.

Proyecto N° 5: Manejo de un módulo LCD

Cuando se trabaja en diseño de circuitos electrónicos es frecuente encontrarse con la necesidad de visualizar un mensaje, que tiene que ver con el estado de la máquina a controlar, con instrucciones para el operario, o si es un instrumento de medida, mostrar el valor registrado. En la mayoría de los casos, recurrimos a los displays de siete segmentos, pero estos además de no mostrar caracteres alfanuméricos ni ASCII, tienen un elevado consumo de corriente y son un poco dispuestos de manejar, cuando se requiere hacer multiplexaje.

Los módulos de cristal líquido o LCD, solucionan estos inconvenientes y presentan algunas ventajas, como un menor consumo de corriente, no hay que preocuparse por hacer multiplexaje, no hay que hacer tablas especiales con los caracteres que se desea mostrar, se pueden conectar fácilmente con microprocesadores o microcontroladores y además, los proyectos adquieren una óptima presentación y funcionalidad. En principio, vamos a conocer las características más importantes de los módulos, luego se muestra la forma de conectarlos con el microcontrolador y se hacen programas simples para escribir mensajes en la pantalla.

Módulos de cristal líquido o LCD

Antes de mostrar la forma de conectar estos módulos con el microcontrolador, haremos un pequeño recuento de las principales características que ellos tienen, las cuales nos servirán para entender mejor los programas y los diagramas que se muestran más adelante:

- Los módulos LCD se encuentran en diferentes presentaciones, por ejemplo (2 líneas por 16 caracteres), 2x20, 4x20, 4x40, etc. La forma de utilizarlos y sus interfaces son similares, por eso, los conceptos vistos aquí se pueden emplear en cualquiera de ellos. En nuestro caso, trabajaremos con un display de 2x16, ya que es de bajo costo, se consigue fácilmente en el comercio y tiene un tamaño suficiente para la mayoría de las aplicaciones.
- La figura 2.25 muestra dos tipos de configuración de pines que se encuentran comúnmente, aunque cambian su ubicación, estos conservan las mismas funciones. Algunos módulos LCD tienen luz posterior o “backlight”, para mejorar su visualiza-

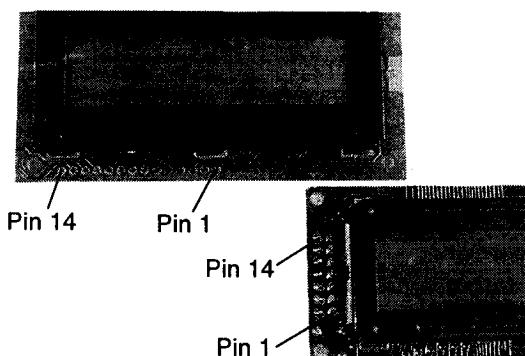


Figura 2.25. Configuración de pines de los módulos LCD

ción, ésta se maneja a través de dos pines que normalmente se conectan a +5V y a tierra. Para evitar que se presenten altas temperaturas, debido a la luz posterior, estos pines se deben manejar de manera pulsante (encendiendo y apagando), con una frecuencia de aproximadamente 60 Hz. Otra opción mucho más sencilla es utilizar una resistencia de 10 ohmios (a 1/2W) para alimentar el positivo del *backlight*.

- Los pines de conexión de estos módulos incluyen un bus de datos de 8 bits, un pin de habilitación (E), un pin de selección, que indica que el dato es una instrucción o un carácter del mensaje (RS) y un pin que indica si se va a escribir o leer en el módulo LCD (R/W). La figura 2.26 describe la función de cada uno de ellos.

Terminal	Símbolo	Nombre y Función
1	Vss	Tierra, 0V
2	Vdd	Alimentación +5V
3	Vo	Ajuste de Voltaje de contraste
4	\bar{RS}	Selección Dato/Control
5	R/W	Lectura/escritura en LCD
6	E	Habilitación
7	D0	D0 Bit menos significativo
8	D1	D1
9	D2	D2
10	D3	D3
11	D4	D4
12	D5	D5
13	D6	D6
14	D7	D7 Bit más significativo

Figura 2.26. Función de los pines del módulo LCD

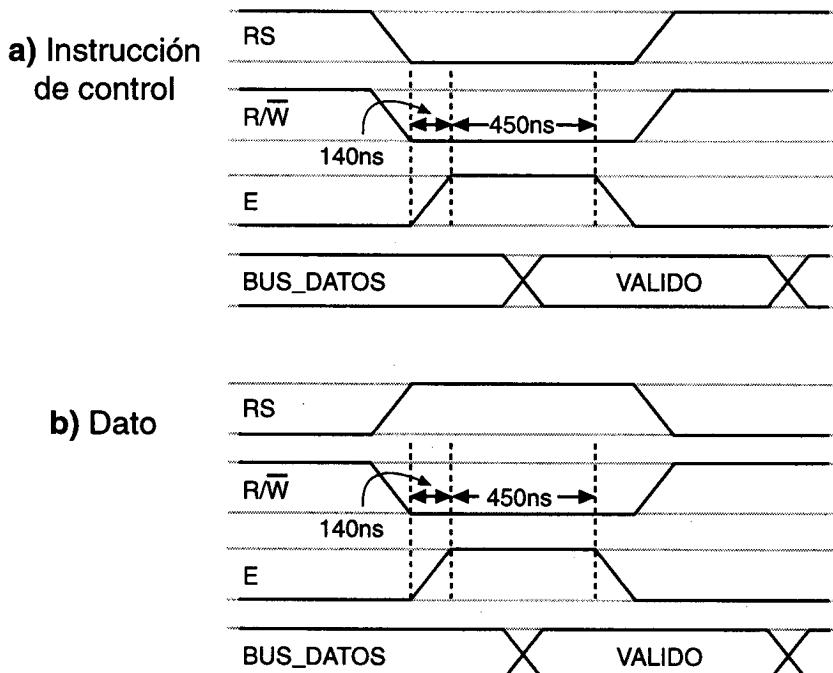


Figura 2.27. Diagrama de tiempo del módulo LCD

- Según la operación que se desee realizar sobre el módulo de cristal líquido, los pines de control E, RS y R/W deben tener un estado determinado. Además, debe tener en el bus de datos un código que indique un carácter para mostrar en la pantalla o una instrucción de control. En la figura 2.27 se muestra el diagrama de tiempos que se debe cumplir para manejar el módulo.
- El módulo LCD responde a un conjunto especial de instrucciones, estas deben ser enviadas por el microcontrolador o sistema de control al display, según la operación que se requiera. Estas instrucciones se emplean en los ejemplos que realizaremos más adelante, en ellos se explica la forma de utilizarlas. En la figura 2.28 se muestran las instrucciones del módulo.
- La interface entre el microcontrolador y el display de cristal líquido se puede hacer con el bus de datos trabajando a 4 u 8 bits. Las señales de control trabajan de la misma forma en cualquiera de los dos casos, la diferencia se establece en el momento de iniciar el sistema, ya que existe una instrucción que permite establecer dicha configuración. Estas conexiones se explican más adelante en forma detallada.

Control y dato	Señal de control	DATO/DIRECCION								DESCRIPCION	
		RS	RW	D7	D6	D5	D4	D3	D2	D1	
Borrar pantalla		0	0	0	0	0	0	0	0	0	Limpia todo display y retorna el cursor a la posición de inicio (dirección 0)
Cursor a casa		0	0	0	0	0	0	0	0	1	Retorna el cursor a la posición de inicio (dirección 0). También retorna el display, desplazándolo a la posición original. Los contenidos de la RAM DD permanecen sin cambios.
Seleccionar modo		0	0	0	0	0	0	0	1	I/D	Configura la dirección de movimiento del cursor y si se desplaza o no el display. Esta operación es realizada durante operaciones de lectura y escritura.
Encender/apagar pantalla		0	0	0	0	0	0	1	D	C	Configura el estado ON/OFF de todo el display (D), el cursor (C) y el parpadeo del carácter en la posición del cursor.
Desplazar Cursor/Pantalla		0	0	0	0	0	1	S/C	RL	*	Mueve el cursor y desplaza el display sin cambiar los contenidos de la RAM DD.
Activar función		0	0	0	0	1	DL	N	F	*	Configura el tamaño de la interfaz (DL), el número de líneas del display (N) y la fuente del carácter (F). N=0, 1 linea; N=1, 2 líneas
CG RAM		0	0	0	1	Dirección generador de RAM					Ajusta la dirección del generador de caracteres. El dato CG RAM es enviado y recibido después de este ajuste.
DD RAM		0	0	1	Direcciones de datos RAM						Ajusta la dirección de la RAM DD. La dirección es enviada y recibida después de este ajuste.
Bandera de ocupado		0	0	BF	AC						Leeura de la bandera Busy Flag, indicando que operaciones internas son realizadas y lectura de los contenidos del contador de direcciones.
Escritura CG RAM/DD RAM	1	0	Escritura de Dato								Escribe datos en la RAM DD o en la RAM CG.
LECTURA CG RAM/DD RAM	1	1	Lectura de Dato								Lectura de datos desde la RAM DD o la RAM CG

Significado de las abreviaturas

I/D	= 1 Incrementa = 0 Decrementa
S	= 1 Desplaza el mensaje en la pantalla = 0 Mensaje fijo en la pantalla
D	= 1 Encendér (activar) la pantalla = 0 Apagar la pantalla (desactivar)
C	= 1 Activar cursor = 0 Desactivar cursor
B	= 1 Parpadea carácter señalado por el cursor = 0 No parpadea el carácter
S/C	= 1 Desplaza pantalla = 0 Mueve cursor
RL	= 1 Desplazamiento a la derecha = 0 Desplazamiento a izquierda
DL	= 1 Datos de ocho bits = 0 Datos de cuatro bits
BF	= 1 Durante operación interna del módulo = 0 Finalizada la operación interna

Figura 2.28. Conjunto de instrucciones de los módulos LCD

- Los caracteres que se envían al display se almacenan en la memoria RAM del módulo. Existen posiciones de memoria RAM, cuyos datos son visibles en la pantalla y otras que no son visibles, estas últimas se pueden utilizar para guardar caracteres que luego se desplazan hacia la parte visible. En la figura 2.29 se muestran las direcciones de memoria visibles y no visibles, que conforman las dos líneas de caracteres del módulo.

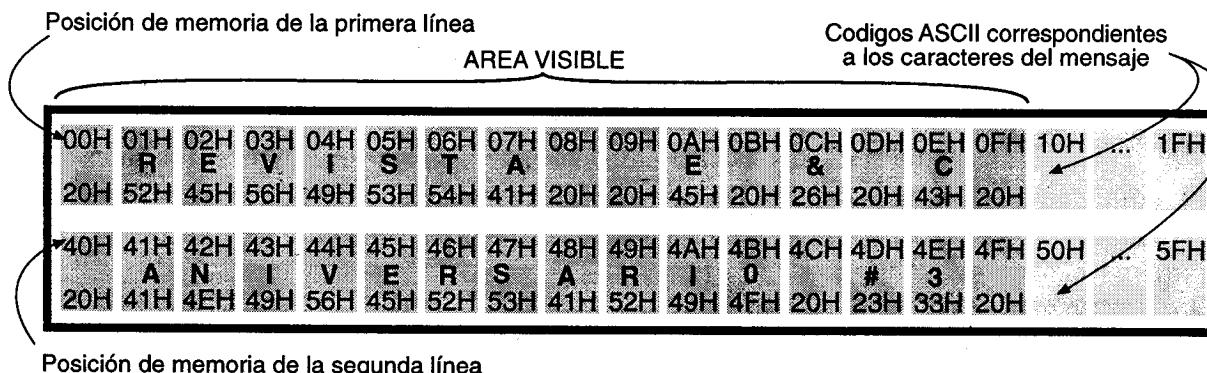


Figura 2.29. Mapa de memoria del módulo LCD

- Es importante anotar que sólo se pueden mostrar caracteres ASCII de 7 bits, por lo tanto algunos caracteres especiales no se pueden ver (se debe tener a la mano una tabla de los caracteres ASCII para conocer los datos que son prohibidos). Por otra parte, se tiene la opción de crear caracteres especiales (creados por el programador), y almacenarlos en la memoria RAM que posee el módulo.

Interface con microcontrolador a 8 bits

El proyecto consiste en conectar el módulo de cristal líquido a un microcontrolador PIC16F84, utilizando el bus de datos a 8 bits. En este caso se emplea el PIC16F84, aunque se puede implementar con otros microcontroladores que sean compatibles. En la figura 2.30 se muestra el diagrama de conexiones para este caso.

Para estos ejercicios en particular, sólo nos interesa escribir datos en la pantalla (no hacer lectura); por lo tanto el pin de selección de lectura/escritura (R/W) en el display, se conecta a tierra. El puerto B del microcontrolador se utiliza como bus de datos, y el puerto A se encarga de generar las señales de control.

En el oscilador del PIC16F84 se emplea un cristal de 4 MHz, por lo tanto tenemos ciclos de instrucción de un microsegundo. Para el módulo LCD, se emplea un potencímetro de 5Kohm, conectado entre +5V y tierra, para controlar el contraste de la pantalla.

En la figura 2.31 se muestra el listado del programa. Para este caso, el ejemplo consiste en hacer circular un mensaje en la línea superior de la pantalla. La explicación de los pasos contenidos en él es la siguiente:

- Se programan los puertos según las conexiones que se tienen en el circuito.
- Se debe inicializar el módulo LCD. El primer dato que se envía (30H) le dice al módulo que la comunicación es a 8 bits y que se empleará solo una línea de carac-

teres. El dato se puede deducir con la lista de las instrucciones que se muestra en la figura 3. La rutina llamada CONTROL, se encarga de generar las señales y los tiempos necesarios para que exista una correcta comunicación.

3. El segundo dato (07H), le dice al módulo que el mensaje se va a desplazar en la pantalla.
4. El tercer dato (0CH), hace que se encienda el display.
5. El siguiente paso es entrar en un ciclo que hace una lectura de la tabla donde se encuentra el mensaje y lo lleva a la memoria del módulo LCD. Cuando se termina de enviar todos los caracteres, se inicia el ciclo nuevamente.

Las rutinas CONTROL y DATO emplean las mismas instrucciones. La única diferencia es que cada una le da el nivel lógico adecuado al pin RS, que indica si el dato enviado es un carácter del mensaje (un dato) o una instrucción de control.

En el modo que desplaza el mensaje en la pantalla se tiene un tiempo de espera un poco largo antes de que aparezcan los caracteres en la pantalla, esto se debe a que primero se llena o se carga la memoria de datos de la parte no visible.

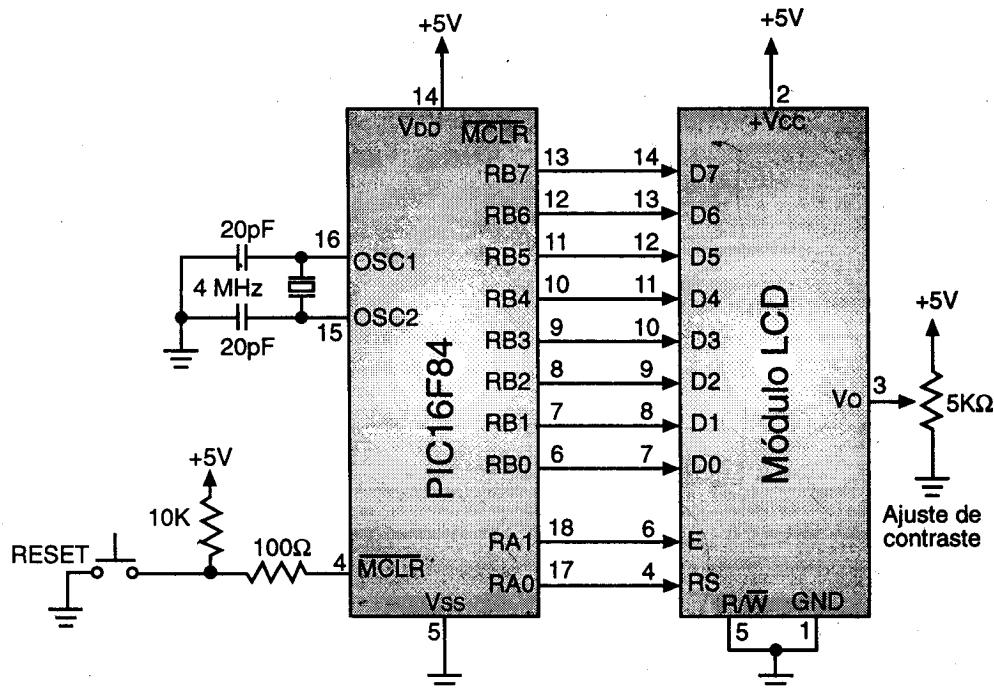


Figura 2.30. Diagrama esquemático de la conexión a 8 bits entre el microcontrolador y el módulo LCD

```
;Este programa hace que un mensaje circule en la pantalla
;de un modulo lcd      ; p=16f84, osc= xt, wdt = off
indf    equ    0h      ;para direccionamiento indirecto
tmro   equ    1h      ;contador de tiempo real
pc     equ    2h      ;contador de programa
status equ    3h      ;registro de estados y bits de control
fsr    equ    4h      ;selección de bancos de memoria y registros
ptoia  equ    5h      ;puertos
ptob   equ    6h
r0c   equ    0ch      ;
r0d   equ    0dh      ;
```

```

r13    equ    13h   ;
z      equ    2h    ;bandera de cero
c      equ    0h    ;bandera de carry
w      equ    0h    ;para almacenar en w
r      equ    1h    ;para almacenar en el mismo registro
e      equ    1h
rs     equ    0h

        org    00    ;vector de reset
        goto  inicio
        org    05h

retardo movlw 0ffh
decre   movwf  r13
        decfsz r13,r
        goto   decre
        retlw  0

control bcf    ptoa,rs      ;esta rutina genera las señales de control
dato    goto   dato2
dato2   bsf    ptoa,rs      ;y entrega el dato correspondiente al módulo
        bsf    ptoa,e       ;utiliza interface a 8 bits
        movwf  ptob
        call   retardo
        bcf    ptoa,e
        call   retardo
        retlw  0

tabla2 addwf pc,r          ;mensaje a ser rotado
retlw "C"
retlw "U"
retlw "R"
retlw "S"
retlw "O"
retlw " "
retlw "D"
retlw "E"
retlw " "
retlw "M"
retlw "I"
retlw "C"
retlw "R"
retlw "O"
retlw "C"
retlw "O"
retlw "N"
retlw "T"
retlw "R"
retlw "O"
retlw "I"
retlw "A"
retlw "D"
retlw "O"
retlw "R"
retlw "E"
retlw "S"
retlw " "
retlw "P"
retlw "I"
retlw "C"
retlw " "
retlw "C"
retlw "E"
retlw "K"
retlw "I"
retlw "T"
retlw " "
retlw " "
retlw " "

```

Nota: Las comillas que posee cada letra le indican al ensamblador que el dato requerido es el valor ASCII del carácter

Nota: Las comillas que posee cada letra le indican al ensamblador que el dato requerido es el valor ASCII del carácter.

```

retlw  " "
retlw  " "
retlw  0

inicio  movlw  0fch      ;programación de puertos
        tris   ptoa       ;según el circuito
        movlw  00h
        tris   ptob
begin   movlw  30h      ;
        call   control    ;inicia display a 8 bits y 1 línea
        movlw  07h      ;selecciona el modo de desplazamiento
        call   control    ;activa display
        movlw  0ch
        call   control
muestra movlw  0      ;inicia el envío de caracteres
        movwf  r0c
        call   tabla2    ;al módulo
        call   dato       ;hace barrido de la tabla
        movlw  09fh
        movwf  r0d
        call   control    ;retardo entre caracteres
retal1  call   retardo
        call   retardo
        decfsz r0d,r
        goto  retal1
        incf  r0c,r
        movlw  28h
        xorwf r0c,w
        btfss status,z
        goto  ciclo
        goto  muestra
end

```

Figura 2.31. Programa para la conexión a 8 bits

Interface con microcontrolador a 4 bits

La conexión entre el PIC y el módulo LCD se hará empleando un bus de datos de 4 bits, en este caso, se utilizarán los 4 pines de mayor peso del puerto B (RB4-RB7) del microcontrolador. Las señales de control (RS y E), se generarán con los dos pines de menor peso del puerto B (RB0 y RB1). Con esto, se libera todo el puerto A y los pines RB2 y RB3 del microcontrolador, los cuales se podrían emplear en otras funciones, figura 2.32.

El software que se implementará en el microcontrolador, se encarga de mostrar un mensaje en el display. Dicho mensaje ocupa las dos líneas de la pantalla y permanece fijo, a diferencia del ejemplo anterior, en el cual el mensaje se desplazaba.

En la figura 2.33 se muestra el listado del programa que realiza la tarea descrita anteriormente, este sufre unas modificaciones respecto al ejemplo anterior. Los pasos más importantes son:

1. Se programan los puertos según el circuito.
2. Se debe inicializar el módulo LCD. El primer dato que se envía (02H) le dice al módulo que la comunicación se va a realizar a 4 bits.
3. El segundo dato (28H) ratifica que la comunicación es a 4 bits y que se emplearán las dos líneas de caracteres del display. El dato se puede deducir con la lista de las instrucciones que se muestra en la figura 3. La rutina llamada CONTROL, se

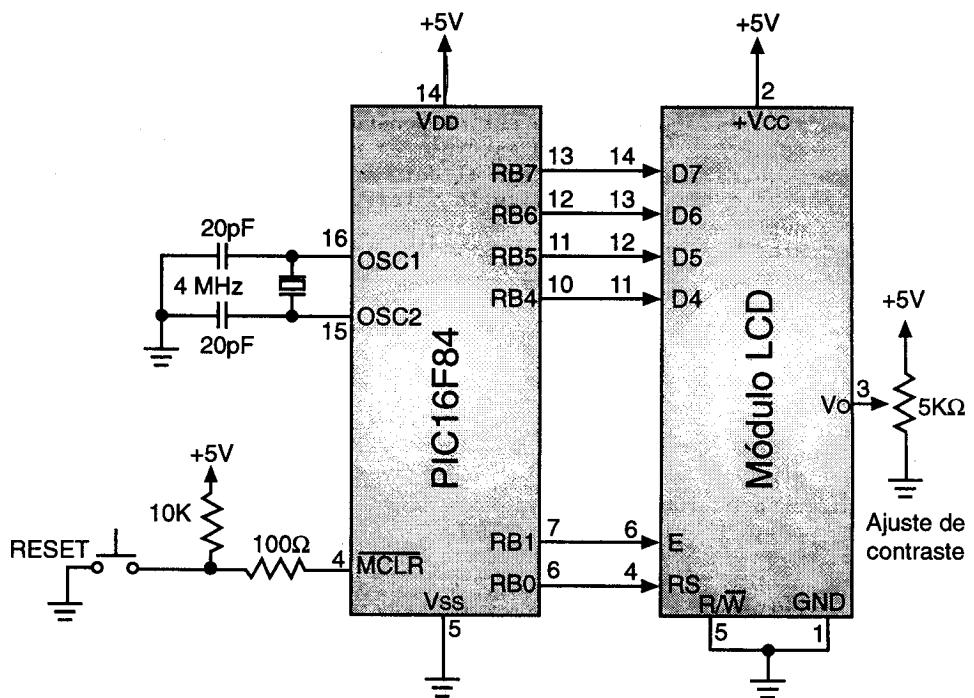


Figura 2.32. Diagrama esquemático de la conexión a 4 bits entre el microcontrolador y el módulo LCD

```

;este programa hace que un mensaje se repita indefinidamente
;en un modulo lcd de 2 lineas con 16 caracteres
indf    equ     0h      ;para direccionamiento indirecto
tmro   equ     1h      ;contador de tiempo real
pc     equ     2h      ;contador de programa
status equ     3h      ;registro de estados y bits de control
fsr    equ     4h      ;selecccion de bancos de memoria y registros
ptoa   equ     5h      ;puertos
ptob   equ     6h
r0c    equ     0ch     ;
r0d    equ     0dh     ;
r0e    equ     0eh     ;
r13    equ     13h     ;
z      equ     2h      ;bandera de cero
c      equ     0h      ;bandera de carry
w      equ     0h      ;para almacenar en w
r      equ     1h      ;para almacenar en el mismo registro
e      equ     1h      ;
rs    equ     0h      ;

        org    00          ;vector de reset
        goto inicio       ;va a iniciar programa principal
        org    05h

retardo movlw 0ffh
        movwf r13
decre  decfsz r13,r
        goto decre
        retlw 0

limpia clrf r0c
limpi  movlw "
        call dato
        incf r0c,r
        movlw 50h
        xorwf r0c,w

```

Capítulo 2. Proyectos con el PIC16F84

```
btfss status,z
goto limpi
retlw 0

control bcf ptob,rs      ;esta rutina genera las señales de control
goto dato2                      ;para escribir en el modulo lcd y
dato bsf ptob,rs                  ;entrega el dato a ser mostrado en la pantalla
dato2 bsf ptob,e                ;utiliza la interface a 4 bits
    movwf r0e
    movlw 0fh
    andwf ptob,r
    movf r0e,w
    andlw 0f0h
    iorwf ptob,r
    call retardo
    bcf ptob,e
    call retardo
    bsf ptob,e
    movlw 0fh
    andwf ptob,r
    swapf r0e,w
    andlw 0f0h
    iorwf ptob,r
    call retardo
    bcf ptob,e
    call retardo
    retlw 0

tabla addwf pc,r      ;mensaje que se muestra
    retlw " "
    retlw " "
    retlw "R"
    retlw "E"
    retlw "V"
    retlw "I"
    retlw "S"
    retlw "T"
    retlw "A"
    retlw " "
    retlw "E"
    retlw "&"
    retlw "C"
    retlw " "
    retlw " "
    retlw " "
    retlw "C"
    retlw "E"
    retlw "K"
    retlw "I"
    retlw "T"
    retlw " "
    retlw "_"
    retlw " "
    retlw "P"
    retlw "E"
    retlw "R"
    retlw "E"
    retlw "I"
    retlw "R"
    retlw "A"
    retlw " "
    retlw 0

    ;mensaje de la segunda linea

inicio movlw 0ffh      ;programación de puertos
    tris ptoa                      ;segun el circuito
    movlw 0ch
    tris ptob
begin   movlw 02h      ;inicia display a 4 bits
```

```

call    control
movlw  28h      ;display a 4 bits y 2 lineas
call    control
movlw  0ch      ;activa display
call    control
movlw  06h      ;hace que el mensaje permanezca fijo
call    control

blank   call    limpia
muestra clrf  r0c
ciclo   movf   r0c,w
          call    tabla
          call    dato
          movlw  0ffh
          movwf  r0d
retal   call    retardo
          decfsz r0d,r
          goto   retal
          incf   r0c,r
          movlw  11h
          subwf  r0c,w
          btfss  status,c
          goto   ciclo
          movlw  11h
          xorwf  r0c,w
          btfss  status,z
          goto   linea2
linea2 movlw  0c0h
          call    control
line2   movlw  21h
          xorwf  r0c,w
          btfss  status,z
mensaje goto   ciclo
          movlw  080h
          call    control
          goto   blank
          end

;***** pic16F84 *****
;***** wdt = off *****
;***** osc = xt *****
;***** cp = off *****

```

Figura 2.33. Programa para la conexión a 4 bits

encarga de generar las señales y los tiempos necesarios para que exista una correcta comunicación. Debe notarse que esta rutina ha cambiado respecto al primer ejemplo.

4. El tercer dato (0CH), hace que se encienda el display.
5. El cuarto dato (06H), le indica al módulo LCD que el mensaje debe permanecer fijo en la pantalla. Recuerde que en el ejemplo anterior, para desplazarlo se usó el dato 07H.
6. El siguiente paso, es entrar en un ciclo que hace un borrado de la pantalla (con la subrutina BLANK) y luego una lectura de la tabla donde se encuentra el mensaje, para llevarlo a la memoria del módulo LCD. Esta lectura incluye una verificación del momento en que se llena la primera línea de caracteres, con el fin de ubicar el puntero de la RAM del módulo LCD, en la segunda línea de caracteres. Para esto se envía el comando C0H al display (este dato se puede deducir observando la lista de instrucciones de la figura 3).

Cuando se está escribiendo en la segunda línea de caracteres, se verifica en que momento se llenó el área visible del display (se compara el puntero de la tabla con 21H), en ese momento el programa vuelve a iniciar el ciclo de borrar el display y escribir el mensaje en la pantalla.

Las rutinas CONTROL y DATO han cambiado de manera sustancial, como en este caso la interface es a cuatro bits, se debe enviar primero el nibble alto (4 bits de mayor peso) del dato y luego el nibble bajo (4 bits de menor peso). En el momento de enviar cada uno de los datos de 4 bits, las señales de control deben comportarse de la misma forma como si fuera un dato completo. Por eso en la rutina se ve que la señal RS conserva el nivel lógico adecuado y la señal E genera los dos pulsos que se requieren (el primero para el nibble alto y el segundo para el bajo).

Los módulos LCD tienen una instrucción especial para borrar la pantalla (comando 01H), pero en las pruebas que se realizaron en diferentes tipos de módulo se encontró que algunos no la aceptaban y otros requerían retardos diferentes. Por lo tanto, se optó por hacer el borrado con una subrutina llamada LIMPIA, la cual se encarga de mandar el dato correspondiente a un espacio en blanco (20H) a cada posición de memoria del módulo.

Proyecto N° 6: Comunicación serial RS-232

Interface serial RS-232

El puerto serial de las computadoras, conocido también como puerto RS-232, es muy útil ya que permite la comunicación no sólo con otras computadoras, sino también con otros dispositivos tales como el mouse, impresoras y por supuesto, microcontroladores.

Existen dos formas de intercambiar información binaria: la paralela y la serial. La comunicación paralela transmite todos los bits de un dato de manera simultánea y tiene la ventaja que la transferencia es rápida, pero la desventaja de necesitar una gran cantidad de hilos o líneas, situación que encarece los costos y se agrava cuando las distancias que separan los equipos entre los cuales se hace el intercambio es muy grande, debido a las capacitancias entre los conductores, la cual limita el correcto intercambio de datos a unos pocos metros.

La comunicación serial por su parte, transmite un bit a la vez, por lo cual es mucho más lenta, pero posee la ventaja de necesitar un menor número de líneas para la transferencia de la información y las distancias a las cuales se puede realizar el intercambio es mayor; a esto se suma que mediante dispositivos como los modem, la comunicación se pueda extender prácticamente a cualquier lugar del planeta.

Existen dos formas de comunicación serial: la sincrónica y la asincrónica. En la comunicación sincrónica, además de una línea sobre la que transfieren los datos, se necesita otra que contenga pulsos de reloj que indiquen cuando un dato es válido; la duración del bit está determinada por la duración del pulso de sincronismo. En la comunicación asincrónica, los pulsos de reloj no son necesarios y se acude a otros mecanismos para realizar la lectura/escritura de los datos; la duración de cada bit está determinada por la velocidad con la cual se realiza la transferencia de datos. En esta práctica sólo trataremos la comunicación asincrónica o asíncrona.

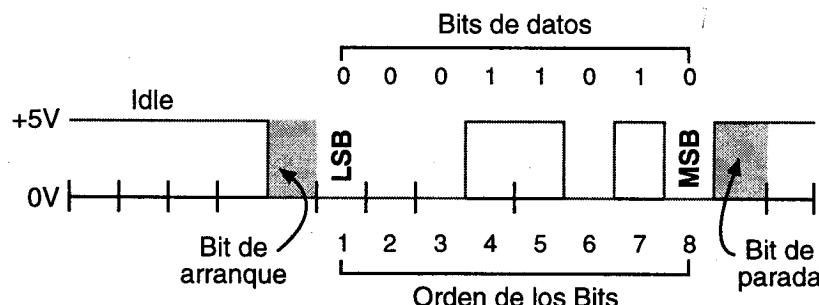


Figura 2.34. Estructura de un carácter que se transmite serialmente

La figura 2.34 muestra la estructura de un carácter que se transmite de forma asíncrona. Normalmente, cuando no se realiza ninguna transferencia de datos, la línea del transmisor es pasiva (*idle*) y permanece en un estado alto. Para empezar a transmitir datos, el transmisor coloca esta línea en bajo durante el tiempo de un bit, lo cual se conoce como bit de arranque (*start bit*) y a continuación, empieza a transmitir con el mismo intervalo de tiempo los bits correspondientes al dato (que pueden ser 7 u 8 bits), empezando por el menos significativo (*LSB*), y terminando con el

más significativo (*MSB*). Al finalizar se agrega el bit de paridad (*Parity*), si es que está activada esta opción, y los bits de parada (*Stop*) que pueden ser 1 ó 2, en los cuales la línea regresa a un estado alto. Al concluir esta operación el transmisor estará preparado para transmitir el siguiente dato.

El receptor no está sincronizado con el transmisor y desconoce cuando va a recibir datos. La transición de alto a bajo de la línea del transmisor activa al receptor y éste genera un conteo de tiempo de tal manera que realiza una lectura de la línea medio bit después del evento; si la lectura realizada es un estado alto, asume que la transición ocurrida fue ocasionada por ruido en la línea; si por el contrario, la lectura es un estado bajo, considera como válida la transición y empieza a realizar lecturas secuenciales a intervalos de un bit hasta conformar el dato transmitido. El receptor puede tomar el bit de paridad para determinar la existencia o no de errores y realizar las acciones correspondientes, al igual que los bits de parada para situaciones similares. Lógicamente, tanto el transmisor como el receptor deberán tener los mismos parámetros de velocidad, paridad, número de bits del dato transmitido y de bits de parada.

Dentro de los microcontroladores hay algunos que poseen funciones y registros especiales para las comunicaciones seriales, tales como la familia PIC16C63 o PIC16C73 de Microchip, los cuales se encargan de manejar todos los aspectos relacionados con las comunicaciones asíncronas, si previamente se han definido todos sus parámetros. Aún si el microcontrolador o microprocesador no posee la opción de las comunicaciones seriales, esta se puede implementar siempre y cuando se tenga presente la duración de cada uno de los bits en la línea. El elemento clave es detectar el bit de arranque, bien sea a través de interrupciones, o bien a través de la lectura frecuente de la línea que contiene los datos. En ambos casos, lo recomendable es que después de detectado el bit de arranque, la lectura de los bits restantes se realice en la mitad del bit, con un error permitido en cada uno de ellos del 3% del tiempo (aunque se podría extender hasta el 4%), sin que se presenten errores de lectura.

En los circuitos digitales, cuyas distancias son relativamente cortas, se pueden manejar transmisiones en niveles lógicos TTL (0 - 5V), pero cuando las distancias aumentan, estas señales tienden a degradarse debido al efecto capacitivo de los conductores y su resistencia eléctrica. El efecto se incrementa a medida que se incrementa la velocidad de la transmisión. Todo esto origina que los datos recibidos no sean iguales a los transmitidos, lo que no se puede permitir en una transferencia de datos. Una de las soluciones más inmediatas en este tipo de situaciones es aumentar los márgenes de voltaje con que se transmiten los datos, de tal manera que las perturbaciones causadas se puedan minimizar e incluso ignorar.

Ante la gran variedad de equipos, sistemas y protocolos que existen surgió la necesidad de un acuerdo que permitiera que los equipos de varios fabricantes pudieran comunicarse entre sí. A principios de los años sesenta se desarrollaron varias normas que pretendían hacer compatibles los equipos, pero en 1962 se publicó la que se convirtió en la más popular: la norma RS-232. Esta norma define la interface mecánica, las características, los pines, las señales y los protocolos que debía cumplir la comunicación serial. La norma ha sufrido algunas revisiones, como la RS-232C en 1969 y la EIA/TIA-232E en 1991.

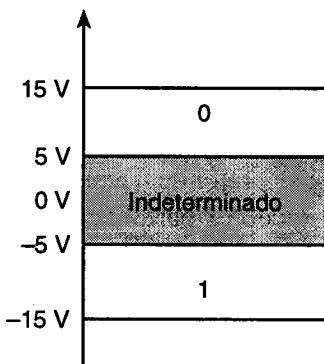


Figura 2.35. Niveles de voltaje RS-232

De todas maneras, todas las normas RS-232 cumplen básicamente con los mismos niveles de voltaje, como se puede observar en la figura 2.35:

- Un uno lógico es un voltaje comprendido entre -5V y -15V en el transmisor y entre -3V y -25V en el receptor,
- Un cero lógico es un voltaje comprendido entre 5V y 15V en el transmisor y entre 3V y 25V en el receptor.

Por lo tanto, deben existir dispositivos que permitan convertir niveles TTL a niveles RS-232 y viceversa. Los primeros dispositivos utilizados fueron los *drivers* MC1488 y los *receivers* MC1489 de Motorola, de los que se desarrollaron versiones mejoradas como los SN75188, SN75189 de *Texas Instruments* y algunos similares de otros fabricantes. Todos los dispositivos nombrados anteriormente necesitan tres voltajes diferentes para su operación cuando el equipo actúa como transmisor y receptor, lo cual no representa ningún problema en computadores tipo PC, ya que se disponen de estos voltajes en la fuente. Pero cuando se trata de sistemas de microcontroladores, en las cuales el espacio es importante y no se puede disponer de voltajes diferentes a 5 voltios, estos circuitos integrados no se pueden utilizar. Para esto se han desarrollado alternativas muy útiles, como el integrado MAX232, que describiremos más adelante.

Se debe tener presente que la norma RS-232 fue desarrollada hace más de 30 años, época en la cual los requerimientos y las capacidades de los equipos eran diferentes. En la actualidad esta norma es un poco limitada, tanto para la distancia a la cual se puede transmitir, como para la velocidad y número de transmisores y receptores que pueden estar simultáneamente conectados. Existen otras normas para la comunicación serial, en la cual se incrementa el número de transmisores o receptores, la velocidad de transmisión, la distancia, etc. Pero a pesar de esto, los principios rectores siguen siendo los mismos de la comunicación asincrónica y de la interface RS-232.

Aspectos prácticos de una comunicación serial

El envío de niveles lógicos (bits) a través de cables o líneas de transmisión necesita la conversión a voltajes apropiados. En un circuito lógico o con microprocesador se trabaja con niveles de voltaje inferiores a 0.8 para representar el valor

lógico 0 y voltajes mayores a 2.0 para representar el valor lógico 1. Por lo general, cuando se trabaja con familias TTL y CMOS se asume que un "0" es igual a cero voltios y un "1" a +5 V.

Cuando la comunicación que se pretende hacer es muy corta, se pueden conectar directamente el transmisor y el receptor para hacer la transferencia de bits usando los mismos niveles lógicos tradicionales de 0 y 5 V. Pero cuando la distancia es mayor a los dos metros, la información digital se afecta notablemente por acción de la atenuación en el cable, el ancho de banda del mismo y la velocidad con que se transmita. La interface RS-232C es una de las diferentes soluciones que hay para esta situación. Básicamente consiste en cambiar los niveles lógicos de la salida o envío de 0 y 5V a dos niveles de voltaje de magnitud mayor: uno positivo (+V) para representar el cero lógico y uno negativo (-V) para representar el uno. En el equipo receptor de la información se realiza el proceso contrario, los niveles positivos y negativos que lleguen se convierten a los niveles lógicos tradicionales de 0 y 5V, figura 2.36. Los niveles de voltaje son simétricos con respecto a tierra y son al menos de +3V para el "0" binario y -3V para el "1". En la figura 2.37 se muestra un ejemplo de la transmisión de un carácter sobre una línea RS-232, incluyendo sus respectivos niveles de voltaje.

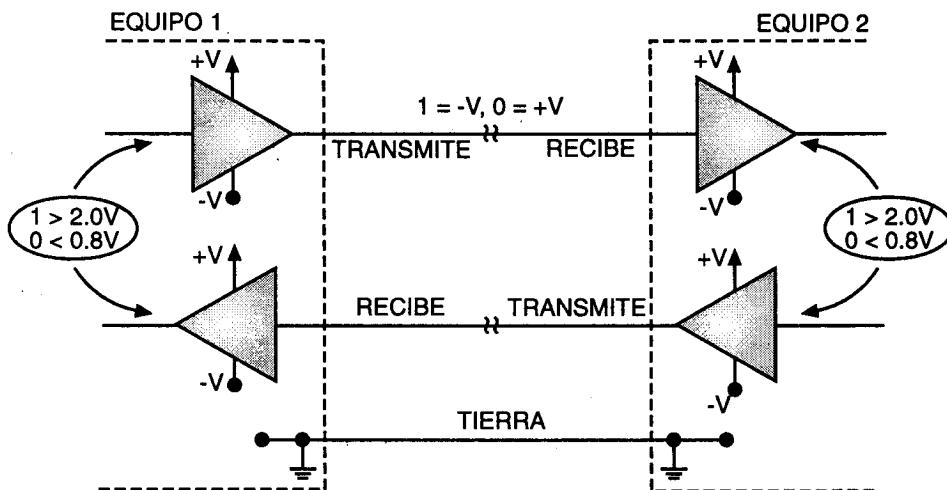


Figura 2.36. Representación de la interface RS-232

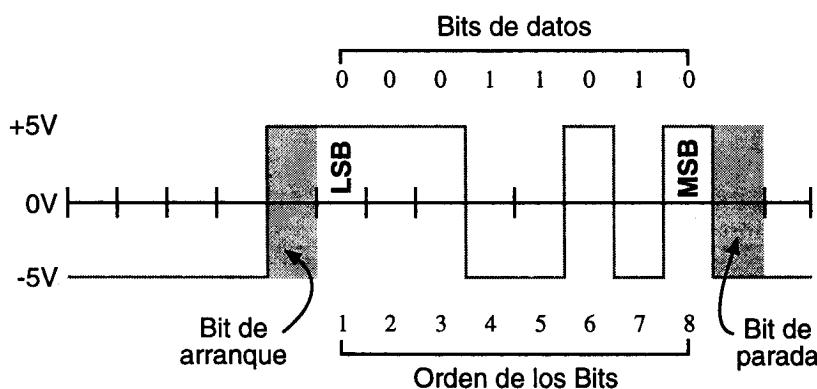
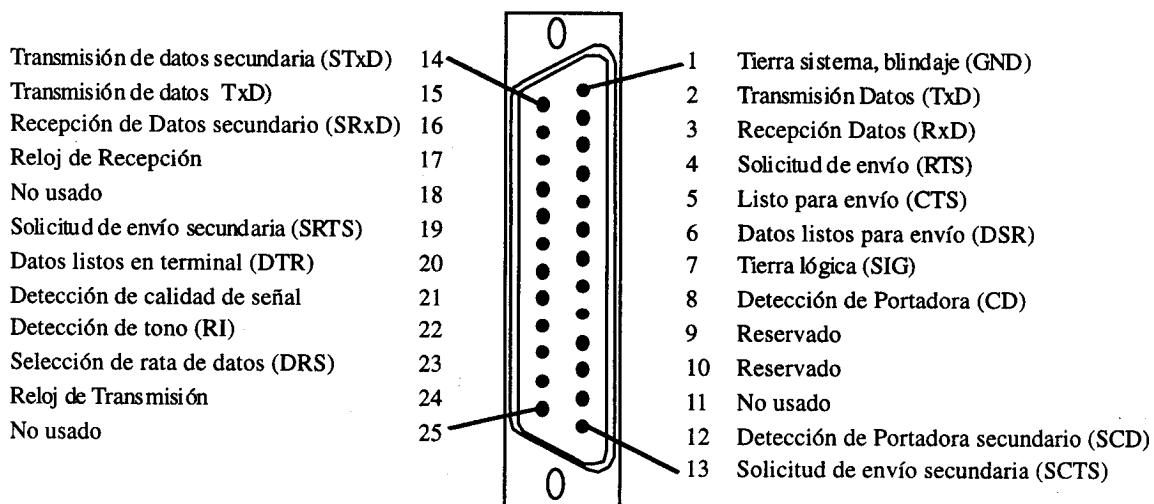


Figura 2.37. Señal presente sobre una línea RS-232

En la práctica, los niveles de voltaje los determinan las fuentes de alimentación que se apliquen a los circuitos de la interface; los niveles más comunes son desde $\pm 12V$ hasta $\pm 15V$. Una interface RS-232 está compuesta por el circuito **transmisor** que convierte la señal de bajo voltaje del equipo lógico a los niveles de voltaje alto que se necesitan en la línea de transmisión y un **receptor** que realiza la función inversa. En los manuales de circuitos integrados se llama *line drivers* y *line receivers*, respectivamente, a los circuitos que ejecutan esta conversión de niveles de voltaje.

Por lo general, se utiliza con las interfaces RS-232 cable multipar o cable *ribbon* con un solo conductor como referencia de tierra. El ruido que se capta a través de la línea aún puede originar problemas. Para reducir el efecto se suele conectar un condensador en paralelo con la salida del circuito transmisor. Según las reglamentación, los estándares de la interface RS-232 permiten una separación máxima de 15 metros a una velocidad de transmisión no mayor a 9.6 kbps (kilo bits por segundo). Sin embargo, se realizan conexiones a distancias mayores sin problema alguno. En la figura 2.38 se muestran los conectores de la interface RS-232.



Nº Pin	Nombre de la señal
1	Detector de portadora (CD)
2	Recepción de Datos (RxD)
3	Transmisión de Datos (TxD)
4	Datos listos en terminal (DTR)
5	Tierra (GND)
6	Datos listos para enviar (DSR)
7	Solicitud de envío (RTS)
8	Listo para envío (CTS)
9	Detector de tono (RI)

Figura 2.38. Conectores RS-232 con sus respectivos pines

El MAX232

Este circuito integrado soluciona los problemas de niveles de voltaje cuando se requiere enviar señales digitales sobre una línea RS-232. El MAX232 se usa en aquellas aplicaciones donde no se dispone de fuentes dobles de ± 12 voltios; por

ejemplo, en aplicaciones alimentadas con baterías de una sola polaridad. El MAX232 necesita solamente una fuente de +5V para su operación; un elevador de voltaje interno convierte el voltaje de +5V al de doble polaridad de $\pm 12V$.

Como la mayoría de las aplicaciones de RS-232 necesitan de un receptor y un emisor, el MAX232 incluye en un solo empaque 2 parejas completas de *driver* y *receiver*, como lo ilustra la estructura interna del integrado que se muestra en la figura 2.39. El MAX232 tiene un doblador de voltaje de +5V a +10 voltios y un inversor de voltaje para obtener la polaridad de -10V. El primer convertidor utiliza el condensador C1 para doblar los +5V de entrada a +10V sobre el condensador C3 en la salida positiva V+. El segundo convertidor usa el condensador C2 para invertir +10V a -10V en el condensador C4 de la salida V-. El valor mínimo de estos condensadores los sugiere el fabricante en el recuadro de la misma figura, aunque en la práctica casi siempre se utilizan condensadores de Tantalio de 10 μF . En la tabla de la figura 2.40 se presentan algunas características de funcionamiento de este circuito integrado.

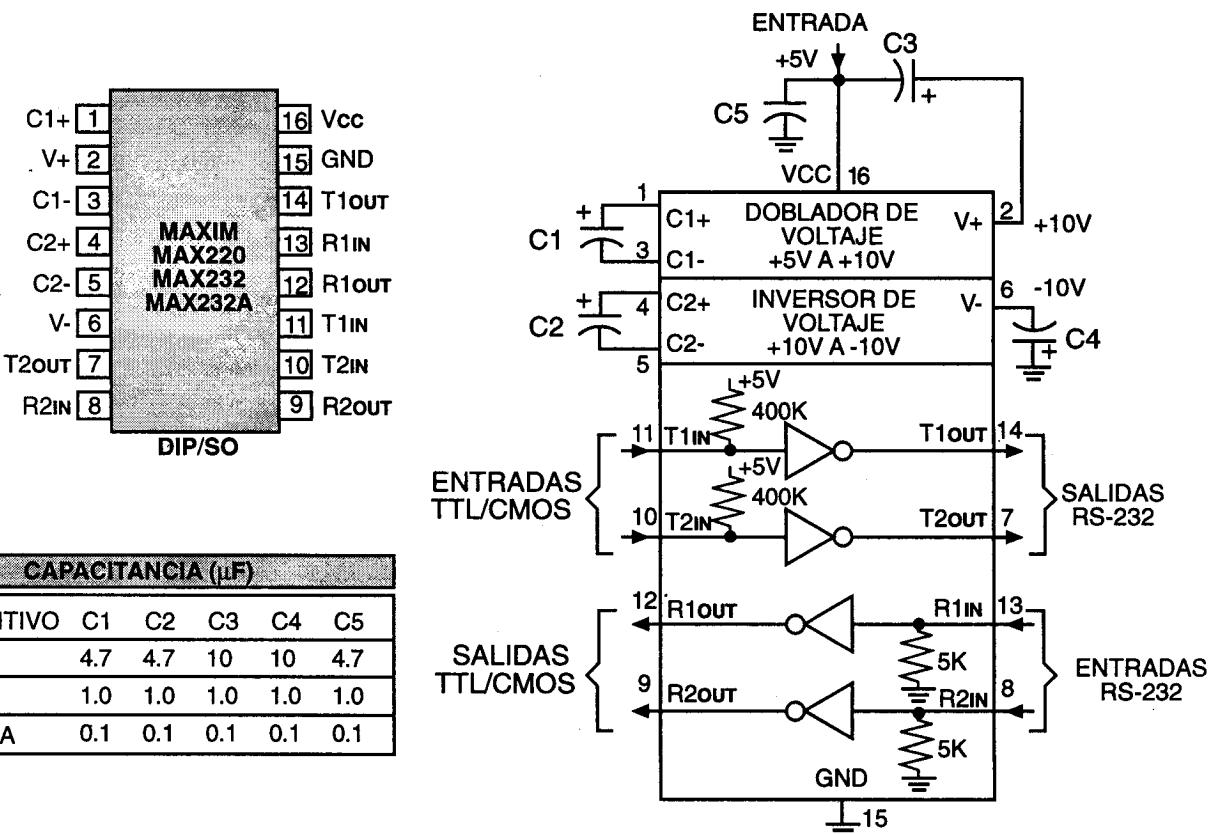


Figura 2.39. Diagrama de pines y estructura interna del MAX232

Una aplicación clásica consiste en conectar las salidas para trasmisión serial TX y RX de un microcontrolador a una interface RS-232 con el fin de intercambiar información con una computadora. La mayoría de los sistemas concentradores de datos están compuestos por sensores conectados a microcontroladores que, a su vez, vía RS-232 le comunican los datos recolectados a un computador central. El MAX232 implementa la interface con la misma fuente de alimentación de +5 voltios. En la figura 2.41 se ilustra la conexión serial de un microcontrolador a través del MAX232.

LIMITES:		
Fuente de alimentación	-0.3 a +6V	
Voltajes de entrada:		
Tin	-0.3V a (Vcc - 0.3V)	
Rin	$\pm 30V$	
Voltajes de salida:		
Tout	$\pm 15V$	
Rout	-0.3V a (Vcc + 0.3V)	
Protección Corto	Continua	
Dissipación de Potencia	842 mW	
CARACTERÍSTICAS a Vcc = +5V, C1-C4 = 0.1 µF		
	Min.	Tip.
TRANSMISOR		
Voltaje de salida (carga 3KΩ)	$\pm 5V$	$\pm 3V$
Entrada BAJA		1.4V
Entrada ALTA	2V	1.4V
Velocidad		200 Kb/seg.
RECEPTOR		
Rango de entrada		$\pm 30V$
Entrada BAJA	0.8V	1.3V
Entrada ALTA		1.8V
Resistencia de Entrada	3KΩ	5KΩ
		7KΩ

Figura 2.40. Características del MAX232

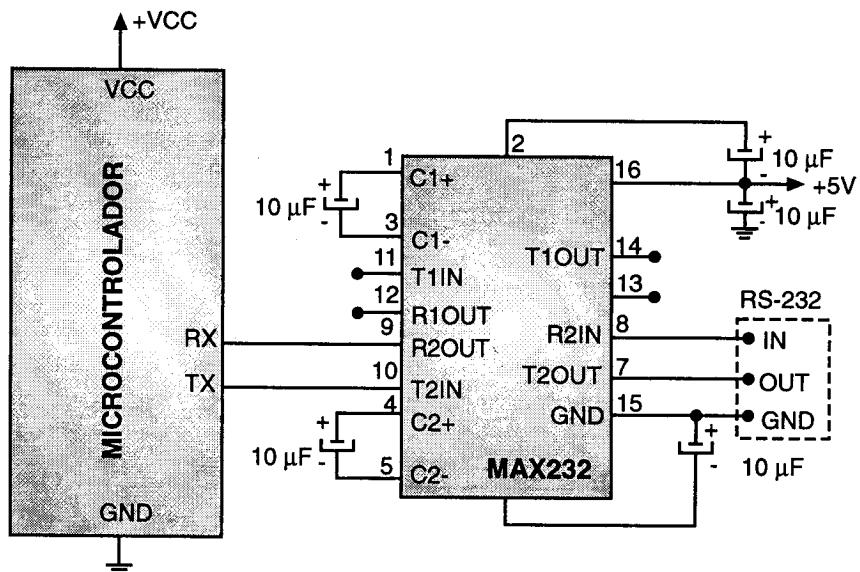


Figura 2.41. Aplicación típica del MAX232

Envío de datos seriales desde el microcontrolador hacia la computadora

El ejercicio que vamos a realizar tiene por objeto practicar la comunicación serial y entender los principios básicos que la rigen. Consiste en hacer un contador decimal (0 a 9), el cual se incrementa cada vez que se oprime un pulsador y muestra el dato del conteo en un display de 7 segmentos, a la vez que lo envía hacia la computadora para que sea mostrado en la pantalla. La comunicación entre el microcontrolador y la computadora se da en un solo sentido (del primero hacia el segundo), por lo tanto se utiliza sólo una línea de datos y el cable de tierra. En la figura 2.42 se muestra el diagrama esquemático del circuito.

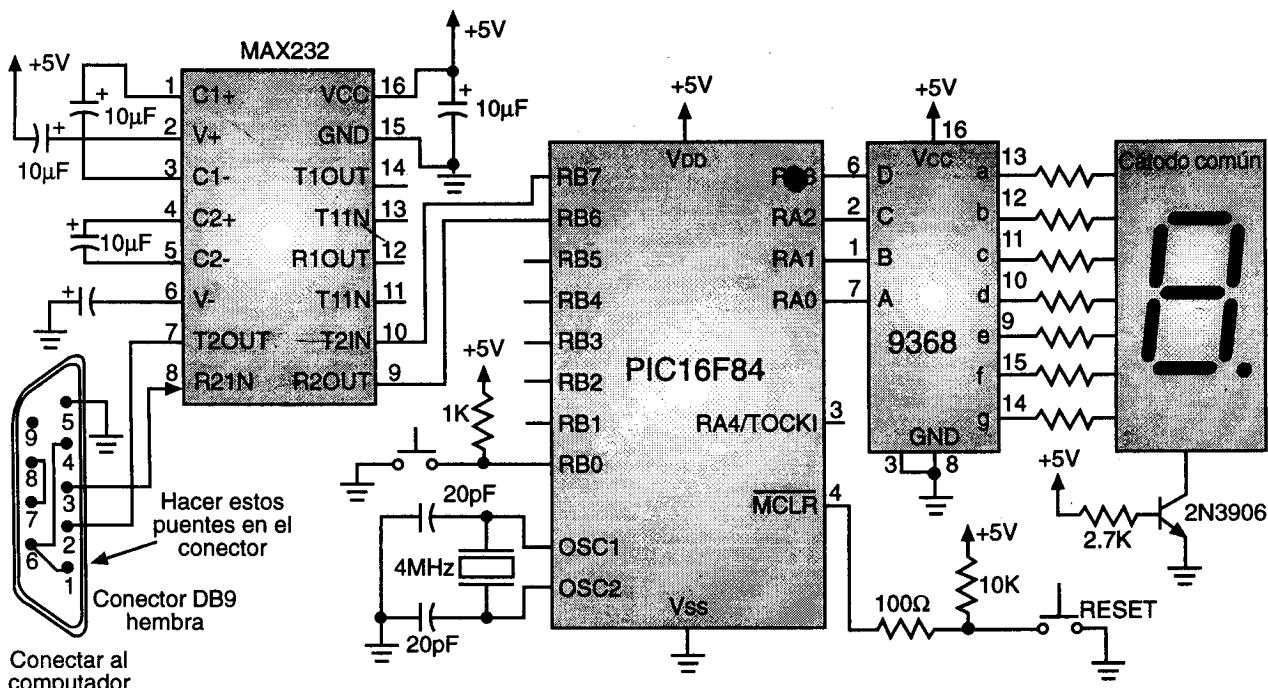


Figura 2.42. Diagrama del contador decimal que envía los datos serialmente hacia la computadora

El microcontrolador se encarga de enviar los datos serialmente con una velocidad de 1200 bps (bits por segundo), datos 8 bits, sin paridad y con un *stop bit*, esta configuración se representa como “1200, 8, N, 1”. El integrado MAX232 se encarga de convertir los datos a niveles de voltaje adecuados para la línea RS-232. Debe notarse que el pin de salida del MAX232 llamado T2OUT va a conectarse al pin de recepción del puerto serial de la computadora (comparar con la figura 2.38).

Dado que la conexión hacia la computadora se realiza con un conector de 9 pines, esta se puede hacer directamente al puerto COM1 (de 9 pines), donde normalmente se conecta el *mouse*. Si se desea que este permanezca en su sitio, se requiere un adaptador RS-232 de 9 a 25 pines para que se pueda hacer la conexión al COM2 (de 25 pines).

Programa del microcontrolador. En la figura 2.43 se muestra el listado completo del programa, es muy similar al del segundo ejercicio (contador decimal) que se encuentra en la figura 2.6, la diferencia radica en que se han agregado dos rutinas.

La primera de ellas, llamada ENVIAR, se encarga de tomar el dato del registro W y transmitirlo serialmente por el pin RB7 del microcontrolador. La rutina llamada DELAY1 se encarga de hacer el retardo de tiempo necesario para sostener cada bit transmitido en la línea; ese retardo está calculado para un oscilador de 4 MHz. El cálculo es muy sencillo: si se transmiten 1200 bits en un segundo, el tiempo de cada bit es de $833\mu\text{s}$ ($1/1200=833\mu\text{s}$); como la rutina de retardo tiene un ciclo que toma cinco períodos del reloj (5 µs), se divide 833/5 y se obtiene la constante de retardo 166. Un caso especial es la rutina que transmite un bit y medio para leer el primer bit, incluyendo el retardo del *start bit*, en este caso la constante es 1.5 veces la de un bit, es decir 249. Las otras partes del programa se encargan de llevar el conteo del número de veces que se oprime el pulsador y de actualizar el display.

```

; este programa envia datos al computador via rs-232
; velocidad = 1200 , datos de 8 bits , sin paridad , un stop bit
indf    equ     0h      ;para direccionamiento indirecto
tmro    equ     1h      ;contador de tiempo real
pc      equ     2h      ;contador de programa
status  equ     3h      ;registro de estados y bits de control
fsr     equ     4h      ;selección de bancos de memoria y registros
ptoa   equ     5h      ;puertos
ptob   equ     6h
trisa  equ     85h     ;programación de los registros
trisb  equ     86h
trans  equ     0ch
r0d    equ     0dh
r0e    equ     0eh
unidad equ     10h
decena equ     11h
centena equ     12h
r14    equ     14h
r1b    equ     1bh
loops  equ     13h
loops2 equ     14h
conta  equ     15h
z      equ     2
rp0    equ     5h      ;selección de página
z      equ     2h      ;bandera de cero
c      equ     0h      ;bandera de carry
w      equ     0h      ;para almacenar en w
r      equ     1h      ;para almacenar en el mismo registro
tx     equ     7h

        org     00      ;vector de reset
        goto   inicio  ;va a iniciar programa principal
        org     05h
delay1 movlw .166    ;carga para 833 µs aproximadamente
startup movwf r0e    ;llevar valor de carga al retardo
redo   nop      ;limpiar circuito de vigilancia
          nop
          decfsz r0e    ;decrementar retardo, saltar si cero
          goto   redo    ;repetir hasta terminar
          retlw  0       ;retornar
retardo ;subrutina de retardo de 100 milisegundos
        movlw  d'100'  ;el registro loops contiene el número
        movwf  loops   ;de milisegundos del retardo
top2   movlw  d'110'  ;
        movwf  loops2  ;
top    nop
      nop
      nop
      nop
      nop
      nop
      decfsz loops2  ;pregunta si terminó 1 ms
      goto   top
      decfsz loops   ;pregunta si termina el retardo
      goto   top2
      retlw  0
enviar ;rutina para enviar dato
        xmrt   movwf  trans   ;lleva el contenido de w a transmisión
        xmrt   movlw  8       ;cargar con número de bits
        xmrt   movwf  r0d    ;el contador
        xmrt   bcf    ptob,tx  ;colocar línea de transmisión en bajo
        xmrt   call   delay1  ;para generar bit de arranque
        xmrt   bcf    ptob,tx  ;colocar línea de transmisión en bajo
        xmrt   bcf    status,c ;limpiar carry
        xmrt   rrf    trans   ;rotar registro de transmisión
        xmrt   btfsc status,c ;preguntar por el carry
        xmrt   bsf    ptob,tx  ;si es uno, colocar línea en alto
        xmrt   call   delay1  ;llamar retardo de 1 bit

```

```

decfsz r0d      ;decrementar contador, saltar si cero
goto  xnext    ;repetir hasta transmitir todo el dato
bsf   ptob,tx  ;colocar línea de transmisión en alto
call  delay1   ;llamar retardo 1 bit -bit de parada-
retlw 0        ;retornar
inicio bsf status,rp0
            movlw 00h
            movwf trisa
            movlw 07fh
            movwf trisb
            bcf  status,rp0
            bsf  ptob,tx
            clrf conta ;inicia contador en cero
            movf conta,w ;el valor del contador pasa al registro w
            movwf ptoa
            addlw 30h
            call enviar
            call retardo ;retardo esperando que suelten la tecla
            btfsc ptob,0 ;pregunta si el pulsador est oprimido
            goto pulsa ;si no lo est continua revisndolo
            call retardo ;si est oprimido retarda 100 milisegundos
            btfsc ptob,0 ;para comprobar
            goto pulsa ;si no lo est vuelve a revisar
            incf conta ;si lo confirma incrementa el contador
            movf conta,w ;carga el registro w con el valor del conteo
            xorlw 0ah ;hace operación xor para ver si es igual a 0ah
            btfsc status,z ;prueba si el contador llegó a 0ah (diez)
            goto inicio ;si es igual el contador se pone en ceros
            goto ciclo ;si no llegó a diez incrementa normalmente
            end       ;y actualiza el display
; ***** pic16f84 *****
; ***** wdt = off *****
; ***** osc = xt *****
; ***** cp = on *****

```

Figura 2.43. Programa que contiene la transmisión serial hacia la computadora

Una característica bien importante del programa es que al dato que se envía hacia la computadora se le ha sumado la constante 30h (48 en decimal), esto se hace para convertir el dato decimal en su carácter ASCII equivalente. Se hace la conversión porque en transmisiones seriales es más común trabajar con caracteres de este tipo, además cuando la computadora lo reciba, se puede pasar directamente a la pantalla.

Programa de la computadora. En la computadora se requiere un programa que se encargue de configurar el puerto con los valores adecuados (1200, 8, N, 1) y de recibir el dato para pasarlo a la pantalla. En este caso utilizamos un programa en lenguaje C, debido a que es el más utilizado en aplicaciones electrónicas y permite configurar fácilmente los puertos.

En lenguaje C, existe una instrucción especial para manejar las comunicaciones seriales. Esta instrucción posee la siguiente sintaxis:

bioscom(cmd, char abyte, int port);

En realidad, esta instrucción acude a la interrupción 14H para permitir la comunicación serial sobre un puerto. Para este caso, cada uno de los parámetros tiene el

siguiente significado:

cmd	Especifica la operación a realizar
abyte	Es un carácter que se envía o recibe por el puerto serial
port	Es la identificación del puerto serial (desde 0 para COM1 hasta 3 para COM4)

El parámetro cmd puede tener los siguientes valores y significados:

0	Inicializa el puerto port con los valores dados por abyte
1	Envía el carácter abyte por el puerto port
2	Lee el carácter recibido por el puerto port
3	Retorna el estado del puerto port

Para la inicialización del puerto, el carácter abyte toma los siguientes valores y los suma para obtener el dato correspondiente:

0x02	7 bits de datos
0x03	8 bits de datos

0x00	1 bit de parada
0x04	2 bits de parada

0x00	Sin paridad
0x08	Paridad impar
0x18	Paridad par

0x00	110 bps
0x20	150 bps
0x40	300 bpss
0x60	600 bps
0x80	1200 bps
0xA0	2400 bps
0xC0	4800 bps
0xE0	9600 baudios

(0x es la notación en lenguaje C para los números hexadecimales)

Para configurar el puerto con algunos parámetros, bastará con realizar una operación OR con los deseados, por ejemplo, para el mismo ejemplo anterior, bastará con seleccionar la palabra dada por:

$$\text{abyte} = 0x80 \mid 0x00 \mid 0x00 \mid 0x03$$

o lo que es equivalente,

$$\text{abyte} = 0x83$$

Para configurar el puerto COM1 con los parámetros del ejemplo dado anteriormente, bastará con la instrucción:

```
bioscom(0,0x83,0); /* (inicializar, parámetros, COM1) */
```

Los programas en lenguaje C tienen su programa fuente con extensión .C, en este caso el programa que recibe los datos del microcontrolador por el puerto COM1 se llama RECIBE1.C. El programa que puede ejecutar el usuario se llama RECIBE1.EXE y se puede correr desde el sistema operativo DOS. El listado completo se muestra en la figura 2.44. Si se desea recibir los datos por el COM2 se debe usar la versión del programa llamada RECIBE2.C y RECIBE2.EXE que también van en el disquete que acompaña el curso. En la figura 2.45 se muestra un pantallazo del programa.

```
/* LA COMPUTADORA RECIBE LOS DATOS SERIALES ENVIADOS POR EL PIC */
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <math.h>
#include <bios.h>
int puerto,COM1,COM2;
int k,j,dato;           /*definición de variables*/
int config;
int COM1,COM2;
char lectura[1];
char dato1[2];
char leer()
{
    do{
        dato=bioscom(2,0x83,puerto);           /*leer dato recibido*/
        } while (((dato<31)|(dato>127))&(!kbhit()));
    return(dato);
}
void main(void)
{
    COM1=0;
    COM2=1;
    puerto=COM1;           /* definir cual puerto se utiliza */
    clrscr();              /*limpiar pantalla*/
    config=0x83;            /*configurar puerto: 1200 baudios,dato de 8 bits,
                           no paridad, 1 bit de parada*/
    bioscom(0,config,puerto);      /*configuracion de los puertos*/
    gotoxy(14,4);
    printf("Curso de Microcontroladores PIC - CEKIT");
    gotoxy(8,6);
    printf("La computadora recibe los datos enviados por el micro - COM1");
    gotoxy(29,8);
    printf("Escape = Salir");
    gotoxy(23,10);
    printf("El dato del contador es:");
do{
    if(!kbhit()) dato1[0]=leer();
    if(!kbhit())
    {
        gotoxy(40,12);
        printf("%1s",dato1);
    }
    }while(!kbhit());
    clrscr();
    printf("Elaborado por: Edison Duque");
}
```

Figura 2.44. Programa en lenguaje C que recibe los datos enviados por el PIC (recibe 1.C)

Curso de microcontroladores PIC - CEKIT

La computadora recibe los datos enviados por el micro - COM1

Escape = Salir

El dato del contador es:

2

Figura 2.45. Pantallazo del programa RECIBE1

Envío de datos seriales desde la computadora hacia el microcontrolador

Este ejercicio es similar al anterior, solo que en esta ocasión el conteo decimal se lleva a cabo en la computadora (se incrementa el contador cada vez que se oprime una tecla), mostrando en la pantalla el valor del conteo. A su vez, dicho dato es enviado serialmente hacia el microcontrolador PIC, el cual ~~les recibe luego que el~~ integrado MAX232 adecua los niveles de voltaje. El número también es mostrado en un display de siete segmentos. El circuito es el mismo de la figura 2.42.

Programa del microcontrolador. En este caso el pin RB6 del microcontrolador se debe programar como entrada para leer los datos seriales. El principal cambio respecto al ejercicio anterior consiste en que la rutina ENVIAR se ha cambiado por la rutina RECIBIR, la cual tiene las temporizaciones y el orden exacto para recibir el dato serial. Nótese que los datos salen del computador por el pin de transmisión y se reciben en el pin R2IN del MAX232, el cual a su vez los entrega al PIC.

Aquí son válidas todas las consideraciones que se hicieron en el ejemplo anterior, por lo tanto dejamos al lector la tarea de estudiar el programa. En la figura 2.46 se muestra el listado completo con sus respectivos comentarios. Se debe notar que al dato recibido se le resta el valor 30h (48 decimal) porque se supone que la computadora envía es el carácter ASCII del número.

```
;este programa recibe datos enviados por la computadora via rs-232
;velocidad = 1200 , datos de 8 bits , sin paridad , un stop bit
indf equ 0h ;para direccionamiento indirecto
tmro equ 1h ;contador de tiempo real
pc equ 2h ;contador de programa
status equ 3h ;registro de estados y bits de control
fsr equ 4h ;seleccion de bancos de memoria y registros
ptoal equ 5h ;puertos
ptob equ 6h
trisa equ 85h ;programación de los registros
trisb equ 86h
r0d equ 0dh ;
r0e equ 0eh ;
conta equ 10h
recep equ 11h
z equ 2
c equ 0
rp0 equ 5h ;selección de página
z equ 2h ;bandera de cero
c equ 0h ;bandera de carry
```

```

w      equ    0h      ;para almacenar en w
r      equ    1h      ;para almacenar en el mismo registro
rx     equ    6h
org    00      ;vector de reset
goto   inicio  ;va a iniciar programa principal
org    05h

unoymedio ;rutina para retardar bit y medio con un cristal de 4.00 mhz.
        movlw .249      ;carga para 1250  $\mu$ seg. aproximadamente
        goto  startup    ;ir a ejecutar el tiempo
delay1 movlw .166      ;carga para 833  $\mu$ s aproximadamente
startup movwf r0e      ;llevar valor de carga al retardo
redo   nop       ;limpiar circuito de vigilancia
        nop
        decfsz r0e      ;decrementar retardo, saltar si cero
        goto  redo       ;repetir hasta terminar
        retiw  0          ;retornar

recibir nop
        clrf   recep    ;limpiar registro de recepción
        btfsc  ptob,rx  ;línea de recepción est en bajo?
        goto   recibir  ;si no lo est, volver a leer
        call   unoymedio ;llamar rutina uno y medio bits
rcvr   movlw 8         ;cargar contador con
        movwf conta    ;el número de bits
rnnext bcf    status,c ;limpiar carry 0
        btfsc  ptob,rx  ;preguntar por el estado de la linea
        bsf   status,c  ;activar carry si está en alto
        rrf   recep    ;rotar registro de recepción
        call   delay1   ;llamar rutina de un bit
        decfsz conta    ;decrementar contador, saltar si cero
        goto   rnnext   ;repetir hasta completar dato
        retlw  0          ;retornar

inicio bsf    status,rp0
        movlw 00h
        trisa
        movlw 0ffh
        movwf trisb
        bcf    status,rp0
        clrf   recep
        clrf   ptoa
ciclo  call   recibir
        movlw 30h
        subwf recep,w
        movwf ptoa
        goto   ciclo
        end

```

Figura 2.46. Programa que recibe los datos seriales enviados por la computadora

Programa de la computadora. En este caso el programa se encarga de realizar el conteo del número de veces que se pulsa una tecla y de mostrar ese número en la pantalla, a la vez que lo envía por el puerto serial hacia el microcontrolador. En la figura 2.47 se muestra el listado completo del programa. En este caso también se tienen dos versiones, una llamada ENVIA1 para trabajar por el COM1 y otra llamada ENVIA2 para trabajar con el COM2. En la figura 2.48 se muestra un pantallazo del mismo.

En la instrucción que envía el dato del conteo se suma el valor 30h (48 decimal), para convertir el número del contador en su equivalente ASCII.

```

/* LA COMPUTADORA ENVIA DATOS SERIALES AL PIC */

#include <conio.h>
#include <stdio.h>
#include <bios.h>

int COM1,COM2,Puerto; /*definición de variables*/
int j,envio,configuracion;
int contador;
char tecla;

void main(void)
{
    clrscr();      /*limpiar pantalla*/
    COM1=0;         /*constantes de los puertos del PC*/
    COM2=1;
    Puerto=COM1;           /*Indicar si es COM1 o COM2*/

    configuracion=0x83;          /*conf.puerto: 1200,8,N,1*/
    bioscom(0,configuracion,Puerto); /*inicializa el COM del PC*/

    gotoxy(20,2);
    printf(" Curso de Microcontroladores PIC ");

    gotoxy(8,5);
    printf("Envio de datos seriales hacia el microcontrolador - COM1");

    gotoxy(10,7);
    printf("Enter = Incremento del contador      Escape = Salir");

    gotoxy(24,10);
    printf("El dato del contador es:");
    contador=0;

    do{      /*ciclo de lectura de medida*/
        tecla=getch();
        contador++;
        if(contador==10) contador=0;
        gotoxy(34,12);
        printf("%d",contador); /*Obtiene tecla oprimida*/
        envio=bioscom(1,contador+0x30,Puerto); /*envía caracter al micro*/
    }while(tecla!=27); /*Hasta que se oprime ESC/*/

    while(!kbhit());
    clrscr();
    printf("Elaborado por: Edison Duque");
}

```

Figura 2.47. Programa en lenguaje C para enviar datos seriales al PIC (envia1.C)

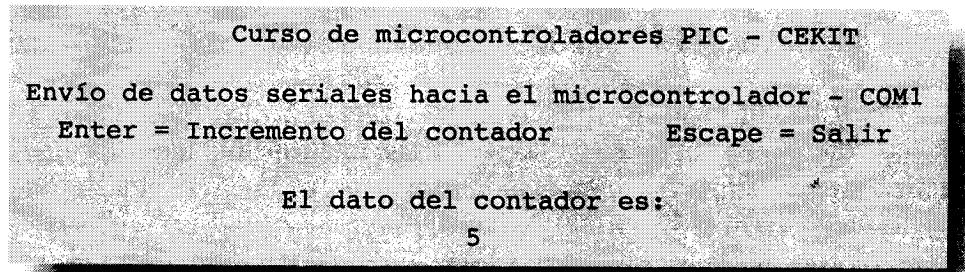


Figura 2.48. Pantallazo del programa ENVIA

Proyecto N° 7: Características especiales de los PIC

Los microcontroladores poseen muchas herramientas especiales, tales como temporizadores, convertidores A/D, interrupciones, comunicaciones seriales, etc. Pero en la mayoría de las ocasiones no se utilizan porque el diseñador desconoce la forma de emplearlos o porque el tiempo de desarrollo es corto y no hay tiempo de estudiarlas o practicar con ellas. En estos casos recurrimos a soluciones de *software* y *hardware* que nos sean familiares y de fácil uso para resolver el problema, aunque esto implique sacrificar el carácter óptimo que deben llevar los diseños.

La familia de microcontroladores PIC no es la excepción, ellos poseen muchas cualidades que la mayoría de las personas no utilizan. En esta práctica vamos a explicar algunas de ellas, de una manera clara y simple, para que el lector las pueda utilizar en sus propios diseños. Para esto vamos a tomar como base el circuito simple que se muestra en la figura 2.49, el cual consta de un microcontrolador PIC16F84 (se puede utilizar un 16C61 o un 16C71), un decodificador y un display, con ellos se pretende construir un contador de década (de 0 a 9) que nos permitirá estudiar todos los casos que veremos.

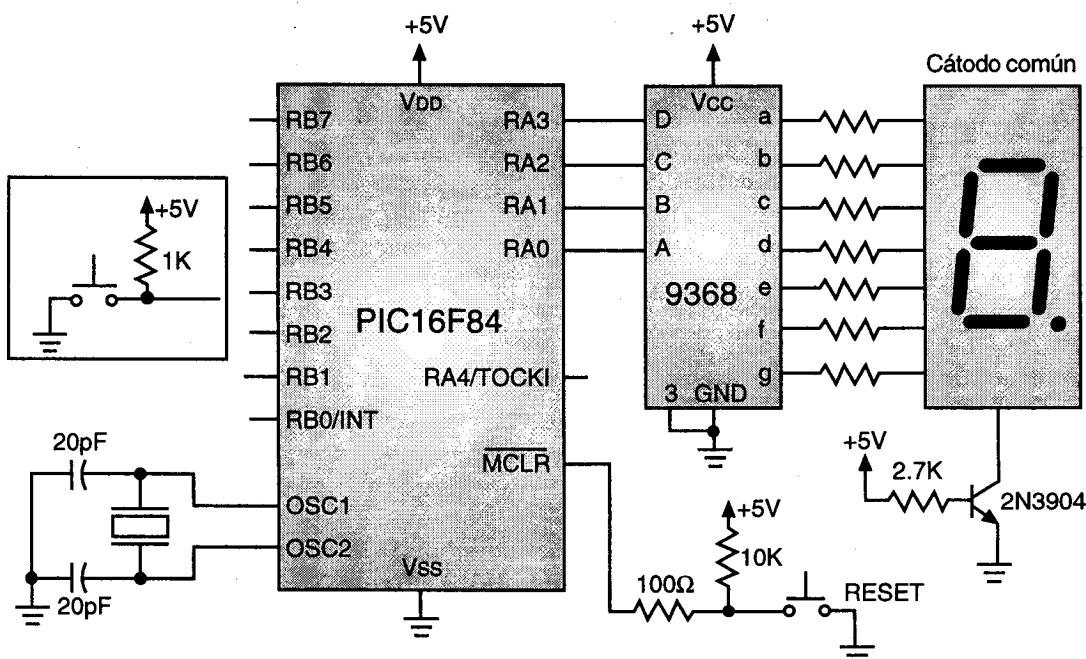


Figura 2.49. Circuito del contador decimal que se utiliza en los ejercicios

Uso de las interrupciones

Estos microcontroladores poseen varias fuentes de interrupción: interrupción externa, finalización del temporizador/contador, cambio en las líneas RB4 a RB7 y finalización de escritura en la EEPROM de datos. El registro 0Bh o INTCON contiene los bits que corresponden a las banderas de estado de las interrupciones y los bits de habilitación para cada una de ellas, figura 1.14. Sólo la bandera de finalización de la escritura reside en el registro 88h o EECON1.

Si el bit GIE (*Global Interrupt Enable*) se coloca en 0 deshabilita todas las interrupciones. Cuando una interrupción es atendida el bit GIE se coloca en 0 automáticamente para deshabilitar otras interrupciones que se puedan presentar, la dirección actual del PC se carga en la pila o stack y el PC se carga con el valor 04h, por lo tanto, la parte del programa que atiende las interrupciones se debe escribir a partir de dicha dirección. Una vez en la rutina de servicio, la fuente de interrupción se puede determinar examinando las banderas. Como la bandera que indica la causa se encuentra en "1", el programa debe encargarse de ponerla en "0" nuevamente antes de salir de la rutina que atiende las interrupciones, esto es para evitar que se vuelva a atender la misma interrupción en repetidas veces.

La instrucción RETFIE permite al usuario retornar de la interrupción, a la vez que habilita de nuevo las interrupciones, ya que pone el bit GIE en "1" automáticamente. A continuación veremos ejemplos de como trabajan algunas de las interrupciones:

Interrupción externa. Actúa en el pin RB0/INT y puede ser configurada para activarse con el flanco de subida o el de bajada, de acuerdo al bit INTEDG del registro OPTION, figura 1.15. Cuando se presenta un flanco válido en el pin INT, la bandera INTF (INTCON,1) se coloca en "1" y el contador de programa o PC salta a la dirección 04h. En ese momento se puede hacer la verificación de la causa de la interrupción consultando el estado de la bandera, si es del caso también se debe probar el estado de alguno de los pines del microcontrolador para confirmar. La interrupción se puede habilitar o deshabilitar utilizando el bit de control INTE (INTCON,4) en "0". Cuando se atiende la interrupción en la rutina de servicio, el programa debe poner la bandera INTF en "0", antes de regresar al programa principal.

El ejercicio que se ha tomado como ejemplo consiste en un contador decimal (0 a 9), el cual se incrementa cada vez que se presenta una interrupción a través del pin INT, provocada por un pulsador externo conectado a dicho pin y que tiene una resistencia de 1 kohm conectada a la fuente de alimentación para fijar un nivel lógico alto en estado de reposo, en la figura 2.49 se tiene un recuadro con el pulsador que se ha conectado al pin RB0/INT.

El programa que se graba en el microcontrolador se muestra en la figura 2.50 y tiene las siguientes características:

- Una subrutina de retardo de 100 milisegundos que sirve para comprobar que el pulsador si fue oprimido y descartar los pulsos de rebote.
- Al inicio se programan los puertos en los registros TRISA y TRISB, al igual que se habilita la interrupción externa con los bits respectivos en registro INTCON y OPTION.
- El ciclo en que se queda enclavado el programa principal no hace nada, solamente esperar a que se presente la interrupción para atenderla.
- La rutina que atiende la interrupción comprueba que el pulsador esté oprimido, además de probar que se haya activado la bandera correspondiente a la interrupción externa. Si las condiciones son favorables, se incrementa el contador y por lo tanto el número que se muestra en el display.

```

;Este programa hace un contador decimal en
;un display de 7 segmentos, se incrementa cada vez que el
;microcontrolador tienen una interrupción por el pin RB0/INT
status equ 03h ;registro de estados
ptoaa equ 05h ;el puerto A está en la dirección 05 de la RAM
ptob equ 06h ;el puerto B está en la dirección 06 de la RAM
intcon equ 0bh ;registro de control y banderas de interrupción
conta equ 0ch ;lleva el conteo de pulsaciones
loops equ 0dh ;utilizado en retardos (milisegundos)
loops2 equ 0eh ;utilizado en retardos
opcion equ 81h ;configuración del flanco de interrupción
trisa equ 85h ;registro de configuración del puerto A
trisb equ 86h ;registro de configuración del puerto B
z equ 02h ;bandera de cero del registro de estados
w equ 00h ;indica que el resultado se guarda en W

reset org 0 ;el vector de reset es la dirección 00
goto inicio ;se salta al inicio del programa
org 4 ;aquí se atiende la interrupción

        call retardo ;retardo para confirmar pulsador
        btfsc ptob,0 ;pregunta por el pin RB0
        goto sale ;si no está oprimido regresa
        btfss intcon,1 ;confirma si la interrupción fue por el pin INT
        goto sale ;si no lo es sale
        incf conta ;incrementa el contador
        movf conta,w ;carga el registro W con el valor del conteo
        xorlw 0ah ;hace operación xor para ver si es igual a 0ah
        btfsc status,z ;prueba si el contador llegó a 0ah (diez)
        clrf conta ;si llegó a diez pasa el conteo a 0
        movf conta,w ;pasa el dato al display
        movwf ptoaa ;
        call retardo ;retardo de 100 milisegundos
        bcf intcon,1 ;la bandera de la interrupción se debe
                      ;poner en 0 antes de regresar (INTF)
        retfie ;regresa al programa principal y habilita otra vez
              ;la interrupción al poner el bit GIE en 1

retardo ;subrutina de retardo de 100 milisegundos
        movlw D'100' ;el registro loops contiene el número
        movwf loops ;de milisegundos del retardo
top2    movlw D'110' ;
        movwf loops2 ;
top      nop
        nop
        nop
        nop
        nop
        nop
        decfsz loops2 ;pregunta si terminó 1 ms
        goto top
        decfsz loops ;pregunta si termina el retardo
        goto top2
        retlw 0

inicio  bsf status,5 ;se ubica en el segundo banco de RAM
        movlw 0f0h ;se carga el registro W con 0f0
        movwf trisa ;se programan los pines del puerto A como salidas
        movlw 0ffh ;se carga el registro W con ff
        movwf trisb ;se programan los pines del puerto B como entradas
        movlw 80h ;en el registro OPTION sólo se programa
        movwf opcion ;el flanco de bajada para el pin INT
        bcf status,5 ;se ubica en el primer banco de memoria RAM
        movlw 90h ;en el registro INTCON se habilita la
                  ;interrupción por el pin INT

        clrf conta ;inicia contador en cero
        movf conta,w ;el valor del contador pasa al registro W
        movwf ptoaa ;pasa el valor de W al puerto A (display)

```

```

ciclo    nop          ;espera que se presente una interrupción
        nop
        goto    ciclo
        end

;=====
;      Fusibles de programación
;      Osc       XT
;      Watchdog   OFF
;      Code protect OFF
;      Power-Up-Timer ON
;      Micro.     PIC16F84
;=====

```

Figura 2.50. Programa para probar la interrupción externa

Interrupción por cambio en el puerto B. El funcionamiento de esta interrupción es muy particular, se presenta cuando ocurre un cambio de nivel en alguno de los pines RB4 a RB7 del microcontrolador. En este caso la bandera RBIF (INTCON,0) se colocará en "1". El bit de control respectivo es RBIE (INTCON,3). El funcionamiento es similar al que se explicó para la interrupción externa, solamente que se deben cambiar los bits del registro INTCON que se utilizan.

El ejemplo que se utiliza emplea nuevamente el contador decimal, la diferencia es que en lugar del pulsador conectado al pin RB0/INT se emplean cuatro interruptores (dipswitch) conectados a los pines RB4 a RB7, como los que se muestran en el recuadro de la figura 2.49. Como se puede ver en estado normal los pines del microcontrolador tienen un estado lógico alto determinado por las resistencias de 1 kohm conectadas a los dipswitch, si el sistema se enciende en estas condiciones no ocurre nada especial, pero si luego de encendido se cambia la posición de alguno de los interruptores, el microcontrolador sufrirá una interrupción y al atenderla hará que el contador decimal se incremente como en el ejercicio anterior. Si el interruptor se deja en ese estado, el microcontrolador tendrá repetidas interrupciones, hasta el momento en que se regrese a su estado inicial.

El programa que se escribe en el microcontrolador se muestra en la figura 2.51, es muy similar al del primer ejercicio. La diferencia está en los bits que habilitan las interrupciones y que se graban en el registro INTCON.

Interrupción por finalización de la temporización. El temporizador/contador es una de las herramientas más valiosas que se tienen en el microcontrolador, se encuentra en la posición de memoria 01h. Su conteo se puede incrementar con una señal externa aplicada al pin RA4/TOCKI o con la señal del reloj interno del microcontrolador (en este caso cada microsegundo). Su tasa de incremento se puede afectar (prolongar) mediante una preescala o divisor de frecuencia que se programa en el registro OPTION. Allí también se selecciona si el incremento es con flanco de subida o de bajada y si la fuente de pulsos es externa (pin RA4/TOCKI) o interna (oscilador).

Cuando el conteo del temporizador llega a $\overline{0FFh}$ y pasa a 00h se genera una interrupción (siempre y cuando esté habilitada), el bit TOIF (INTCON,2) se pondrá en "1". El bit de control respectivo es TOIE (INTCON,5).

```

;Este programa hace un contador decimal en
;un display de 7 segmentos, se incrementa cada vez que el
;microcontrolador tiene una interrupción por cambio en los pines RB4 a RB7
status    equ     03h      ;registro de estados
ptoaa    equ     05h      ;el puerto A está en la dirección 05 de la RAM
ptob     equ     06h      ;el puerto B está en la dirección 06 de la RAM
intcon   equ     0bh      ;registro de control y banderas de interrupción
conta    equ     0ch      ;lleva el conteo de pulsaciones
loops    equ     0dh      ;utilizado en retardos (milisegundos)
loops2   equ     0eh      ;utilizado en retardos
trisa    equ     85h      ;registro de configuración del puerto A
trisb    equ     86h      ;registro de configuración del puerto B
z        equ     02h      ;bandera de cero del registro de estados
w        equ     00h      ;indica que el resultado se guarda en W

reset    org     0          ;el vector de reset es la dirección 00
        goto  inicio      ;se salta al inicio del programa
        org     4          ;aquí se atiende la interrupción

        call   retardo     ;retardo
        btfss intcon,0      ;confirma interrupción por cambio en RB<7:4>
        goto  sale         ;si no lo es sale
        incf  conta        ;incrementa el contador
        movf  conta,w      ;carga el registro W con el valor del conteo
        xorlw 0ah          ;hace operación xor para ver si es igual a 0ah
        btfsc status,z     ;prueba si el contador llegó a 0ah (diez)
        clrf  conta        ;si llegó a diez pasa el conteo a 0
        movf  conta,w      ;pasa el dato al display
        movwf ptoaa         ;
        call   retardo     ;retardo de 100 milisegundos
        bcf   intcon,0      ;la bandera de la interrupción se debe
        ;poner en 0 antes de regresar (RBIF)
        retfie             ;regresa al programa principal y habilita otra
        ;vez la interrupción al poner el bit GIE en 1

retardo  ;subrutina de retardo de 100 milisegundos
        movlw  D'100'       ;el registro loops contiene el número
        movwf  loops         ;de milisegundos del retardo
top2     movlw  D'110'       ;
        movwf  loops2        ;
top      nop
        nop
        nop
        nop
        nop
        nop
        decfsz loops2       ;pregunta si termino 1 ms
        goto  top           ;
        decfsz loops         ;pregunta si termina el retardo
        goto  top2          ;
        retlw  0              ;

inicio   bsf    status,5      ;se ubica en el segundo banco de RAM
        movlw  0f0h          ;se carga el registro W con 0f0
        movwf  trisa         ;se programa el puerto A como salidas
        movlw  0ffh          ;se carga el registro W con ff
        movwf  trisb         ;se programa el puerto B como entradas
        bcf   status,5      ;se ubica en el primer banco de memoria RAM
        movlw  88h           ;en el registro INTCON se habilita la
        movwf  intcon         ;interrupción cambio en los pines RB4 a RB7
        clrf  conta         ;inicia contador en cero
        movf  conta,w      ;el valor del contador pasa al registro W
        movwf  ptoaa         ;pasa el valor de W al puerto A (display)
ciclo    nop
        nop
        goto  ciclo          ;espera que se presente una interrupción
        end

```

Figura 2.51. Prueba de la interrupción por cambio en puerto B

El ejercicio que realizamos para esta prueba consiste en el incremento del temporizador por medio del reloj interno del microcontrolador, dicho conteo está afectado por la máxima preescala que se puede seleccionar (división por 256). Como el oscilador es un cristal de 4 MHz se tiene un ciclo de instrucción de 1 microsegundo, por lo tanto, dado que el registro TMR0 es de 8 bits y que la preescala es de 256 , el tiempo que se tarda en contar hasta 0FFh es de 256×256 microsegundos, es decir 65.536 microsegundos.

El tiempo logrado es muy pequeño para ser percibido , por lo que recurrimos a establecer un contador dentro de la rutina que atiende las interrupciones, este se incrementa cada vez que el programa pase por ese sitio, es decir cada 65,5 milisegundos. Cuando llegue a 10 hará que se incremente el contador decimal que se ha venido tratando en los otros ejercicios. De esta manera el experimentador verá que el display se incrementa cada 0.65 segundos aproximadamente.

En el programa que se muestra en la figura 2.52 se explica cada uno de los pasos que se llevan a cabo. Los valores con que al inicio se programan el registro INTCON y el registro OPTION, se pueden comparar con las tablas que se muestran en las figuras 1.14 y 1.15 para entender mejor la función de cada uno de ellos.

Consideraciones especiales. Se debe tener en cuenta que el registro de trabajo W y el registro de estados no se guardan cuando se atiende una interrupción, por lo tanto se recomienda almacenar su contenido en algún registro temporal ya que estos pueden sufrir cambios en la rutina que atiende dichas interrupciones, lo que puede causar errores cuando se regresa al programa principal.

Uso del circuito de vigilancia o watchdog timer

El *watchdog* es una herramienta que se utiliza para restablecer el programa del microcontrolador cuando este se ha salido de su flujo normal o se ha perdido por un ruido o interferencia electromagnética. Su funcionamiento se habilita por medio de un fusible de configuración que se selecciona (ON/OFF) cada vez que se graba el microcontrolador.

Funciona como un oscilador interno, independiente del oscilador principal del micro, con un período nominal de 18 milisegundos. Cada vez que se cumple el tiempo del *watchdog* el microcontrolador sufre un reset, por lo tanto, se debe usar una instrucción especial que reinicie dicho conteo antes de que se termine. Esta instrucción es CLRWD. El período de 18 milisegundos se puede ampliar hasta casi 2.3 segundos utilizando la preescala del registro OPTION, en el cual existe también un bit que permite seleccionar si la preescala se asigna al *watchdog* o al temporizador/contador. El uso de la preescala se permite sólo para uno de los dos a la vez.

En el programa que se muestra en la figura 2.53 se tiene un ejemplo muy sencillo que ilustra el funcionamiento del *watchdog*. Consiste en hacer que el microcontrolador sufra reset continuamente, esto se logra habilitando el circuito de vigilancia en el momento de grabar el microcontrolador (seleccionando *watchdog timer ON*). Cada vez que el programa se reinicia hace que se convierte el dato que se muestra en

```

;Este programa hace un contador decimal en
;un display de 7 segmentos, se incrementa cada vez que el
;microcontrolador tiene una interrupción por el tmer TMR0
status equ 03h ;registro de estados
ptoaa equ 05h ;el puerto A está en la dirección 05 de la RAM
ptob equ 06h ;el puerto B está en la dirección 06 de la RAM
intcon equ 0bh ;registro de control y banderas de interrupción
conta equ 0ch ;lleva el conteo de pulsaciones
conta2 equ 0fh ;cuenta 10 interrupciones de TMR0
opcion equ 81h ;configuración del temporizador TMR0
trisa equ 85h ;registro de configuración del puerto A
trisb equ 86h ;registro de configuración del puerto B
z equ 02h ;bandera de cero del registro de estados
w equ 00h ;indica que el resultado se guarda en W

reset org 0 ;el vector de reset es la dirección 00
goto inicio ;se salta al inicio del programa

org 4 ;aquí se atiende la interrupción

btfs goto intcon,2 ;confirma si la interrupción fue por el TMR0
goto sale ;si no lo es sale
incf conta2 ;incrementa el contador de diez interrupciones
movf conta2,w ;carga el registro W con el valor del conteo
xorlw 0ah ;hace operación xor para ver si es igual a 0ah
btfs status,z ;prueba si el contador llegó a 0ah (diez)
goto sale
clrf conta2 ;si llegó a diez pasa el conteo a 0
incf conta ;incrementa el contador
movf conta,w ;carga el registro W con el valor del conteo
xorlw 0ah ;hace operación xor para ver si es igual a 0ah
btfs status,z ;prueba si el contador llegó a 0ah (diez)
clrf conta ;si llegó a diez pasa el conteo a 0
movf conta,w ;pasa el dato al display
movwf ptoaa ;
bcf intcon,2 ;la bandera de la interrupción se debe
             ;poner en 0 antes de regresar (TOIF)
             ;regresa al programa principal
             ;la interrupción al poner el bit GIE en 1

inicio bsf status,5 ;se ubica en el segundo banco de RAM
        movlw 0f0h ;se carga el registro W con 0f0
        movwf trisa ;se programan los pines del puerto A como salidas
        movlw 0ffh ;se carga el registro W con ff
        movwf trisb ;se programan los pines del puerto B como entradas
        movlw b'11000111' ;en el registro OPTION sólo se programa
        movwf opcion ;el flanco de bajada para el pin INT
        bcf status,5 ;se ubica en el primer banco de memoria RAM
        movlw b'10100000' ;en el registro INTCON se habilita la
        intcon ;interrupción por el TMR0
        clrf conta2 ;
        clrf conta ;inicia contador en cero
        movf conta,w ;el valor del contador pasa al registro W
        movwf ptoaa ;pasa el valor de W al puerto A (display)
nop ;espera que se presente una interrupción
ciclo goto ciclo
end

=====
; Fusibles de programación
; Osc XT
; Watchdog OFF
; Code protect OFF
; Power-Up-Timer ON
; Micro. PIC16F84
=====
```

Figura 2.52. Programa para comprobar la interrupción por fin de la temporización

```

;Este programa hace que el microcontrolador sufra un reset
;cada 2 segundos aproximadamente, cada vez que se resetea cambia
;el dato del display, el reset lo causa el watchdog timer
status equ 03h ;registro de estados
pta equ 05h ;el puerto A está en la dirección 05 de la RAM
ptob equ 06h ;el puerto B está en la dirección 06 de la RAM
cambia equ 0fh ;registro con el bit que hace cambiar el display
opcion equ 81h ;configuración del flanco de interrupción
trisa equ 85h ;registro de configuración del puerto A
trisb equ 86h ;registro de configuración del puerto B

reset org 0 ;el vector de reset es la dirección 00
goto inicio ;se salta al inicio del programa

inicio bsf status,5 ;se ubica en el segundo banco de RAM
        movlw 0f0h ;se carga el registro W con 0f
        movwf trisa ;se programa el puerto A como salidas
        movlw 0ffh ;se carga el registro W con 00
        movwf trisb ;se programa el puerto B como entradas
        movlw b'11001111' ;en el registro OPTION se programa
        movwf opcion ;la preescala del watchdog (en este caso 128)
        bcf status,5 ;se ubica en el primer banco de memoria RAM
        btfss cambia,0 ;pregunta el estado de la bandera de prueba
        goto uno

cero clrf pta ;pone en 0 el dato del display
        bcf cambia,0 ;commuta la bandera de prueba
;clrwdt ;borrar el conteo del watchdog timer
        goto cero ;se queda en este ciclo esperando el reset

uno movlw 01 ;dato para el display
        movwf pta
        bsf cambia,0 ;commuta la bandera de prueba
;clrwdt ;borrar el conteo del watchdog timer
        goto uno ;se queda en este ciclo esperando el reset
;
===== Fusibles de programación =====
; Osc XT
; Watchdog ON
; Code protect OFF
; Power-Up-Timer ON
; Micro. PIC16F84
;

```

Figura 2.53. Programa del ejercicio con el *watchdog timer*

el display de siete segmentos (1 ó 0), dado que se ha programado el registro OPTION para que se asigne la máxima preescala al temporizador del *watchdog*, el display cambia cada 2.3 segundos aproximadamente.

De esta manera comprobamos que el micro se está reseteando cada vez que termina el conteo del *watchdog* y no se utiliza la instrucción CLRWDT para reiniciarlo. Luego de esta prueba quitamos el punto y coma (;) que precede a la instrucción CLRWDT con el objeto de habilitarla. En este caso el display no cambia de estado, lo que nos demuestra que el microcontrolador no está sufriendo reset como en el caso anterior.

Cuando se va a utilizar el *watchdog* en un programa complejo debemos asegurarnos que el flujo del programa permita ubicar una o varias instrucciones CLRWDT en sitios específicos, por los cuales se pase antes de que se cumpla el período del temporizador, con el objeto de hacer que el programa se reinicie en caso de que el

programa se quede enclavado en un ciclo en el que no debía quedarse. Se debe tener claridad en que por tratarse de un oscilador del tipo RC, su período de tiempo puede variar un poco con las condiciones de voltaje y temperatura, por lo que se recomienda tener unos buenos niveles de tolerancia en los tiempos que se manejan.

Uso de la memoria de datos EEPROM en el PIC16F84

La memoria de datos EEPROM es una de las herramientas más valiosas que tiene el PIC16F84, se puede emplear para almacenar los datos de calibración en un instrumento, un código o clave de seguridad en una alarma, la hora en un sistema de tiempo real, etc. Este microcontrolador posee 64 bytes de memoria EEPROM, el acceso a estas posiciones se consigue a través de dos registros de memoria RAM:

- El registro EEADR (posición 09h), que debe contener la dirección o posición de la memoria EEPROM de datos que se va a leer o escribir.
- El registro EEDATA (posición 08h), que contiene el dato de 8 bits que se va a escribir o el que se leyó.

Adicionalmente existen dos registros de control, el EECON1 (posición 88h) que posee cinco bits que controlan las operaciones de lectura y escritura y el EECON2 (posición 89h), que aunque no está implementado físicamente, es necesario para las operaciones de escritura. En la figura 1.16 se muestra el registro EECON1, debe ponerse especial atención a la función de sus bits para comparalos con los valores que toman en el programa que se utiliza como ejemplo.

La lectura toma un ciclo del reloj de instrucciones (en este caso un microsegundo), mientras que la escritura, por ser controlada por un temporizador incorporado y requerir una operación previa de borrado de la posición de interés, tiene un tiempo nominal de 10 milisegundos; este tiempo puede variar con la temperatura y el voltaje. Según el fabricante, el número típico de ciclos de borrado/escritura de la EEPROM de datos es de 1'000.000 (un millón).

El programa que se muestra en la figura 2.54 es un simple ejemplo de un contador decimal que se incrementa con un pulsador conectado al pin RB0 y muestra el dato en un display de siete segmentos, lo interesante es que el número del conteo se guarda en la posición 00 de la memoria EEPROM de datos, por lo tanto, si se retira la alimentación del sistema, esta puede recuperar el valor que tenía una vez que se restablezca el voltaje. Las rutinas LEER y ESCRIB se encargan de recuperar el dato guardado en la memoria y de almacenarlo nuevamente cuando se ha incrementado, respectivamente.

Como función especial se ha dispuesto al inicio del programa una verificación que permite establecer si el número que se lee de la memoria de datos está entre 0 y 9, esto se hace con el fin de garantizar que la primera vez que se conecte la alimentación al sistema, el dato que se muestre en el display sea menor o igual a 9. Se supone que para las ocasiones siguientes en que se conecte la fuente, el dato ya estará entre los valores adecuados, por lo tanto en esas ocasiones esa verificación será transparente.

```

;Este programa hace un contador decimal en
;un display de 7 segmentos. el número que lleva el conteo
;se guarda en la EEPROM de datos del micro
status equ 03h ;registro de estados
ptoap equ 05h ;el puerto A está en la dirección 05 de la RAM
ptobp equ 06h ;el puerto B está en la dirección 06 de la RAM
eedata equ 08h ;registro de datos de la memoria EEPROM
eeadrp equ 09h ;registro de direcciones de la memoria EEPROM
conta equ 0ch ;lleva el conteo de pulsaciones
loops equ 0dh ;utilizado en retardos (milisegundos)
loops2 equ 0eh ;utilizado en retardos
conta2 equ 0fh ;utilizado en retardos
trisa equ 85h ;registro de configuración del puerto A
trisbp equ 86h ;registro de configuración del puerto B
eecon1 equ 88h ;registro de control de la memoria EEPROM
eecon2 equ 89h ;registro de control de la memoria EEPROM
z equ 02h ;bandera de cero del registro de estados
w equ 00h ;indica que el resultado se guarda en W
c equ 00h ;bandera de carry
;bits especiales del registro eecon1
eeif equ 04h
wrerr equ 03h
wren equ 02h
wr equ 01h
rd equ 00h

reset org 0 inicio ;el vector de reset es la dirección 00
                ;se salta al inicio del programa
                org 5 ;el programa empieza en la dirección 5

retardo ;subrutina de retardo de 100 milisegundos
        movlw D'100' ;el registro loops contiene el número
        movwf loops ;de milisegundos del retardo
top2    movlw D'110' ;
        movwf loops2 ;
top     nop
        nop
        nop
        nop
        nop
        nop
        decfsz loops2 ;pregunta si terminó 1 ms
        goto top
        decfsz loops ;pregunta si termina el retardo
        goto top2
        retlw 0

leer    bsf status,5 ;se ubica en segundo banco de RAM
        bsf eecon1,rd ;pone el bit que inicia la lectura
        bcf status,5 ;vuelve al primer banco de memoria
        movf eedata,w ;el dato leído se pasa al registro W
        movwf conta2 ;se guarda el dato en conta2
        movwf conta
        return

escrib   bsf status,5 ;se ubica en el segundo banco de RAM
        bsf eecon1,wren ;habilita escritura en memoria EEPROM
        bcf eecon1,eeif ;se asegura que la bandera esté en cero
        movlw 055h ;esta secuencia es obligatoria
        movwf eecon2 ;para escribir en la memoria de datos EEPROM
        movlw 0aaah
        movwf eecon2

        bsf eecon1,wr ;orden de escribir el dato que se cargo
                        ;previamente en el registro eedata en la
                        ;posición de memoria direccionada por eeadr

```

Capítulo 2. Proyectos con el PIC16F84

```

espera    btfss    eecon1,eeif ;pregunta si terminó la escritura
          goto     espera   ;si no, espera a que termine
          bcf      eecon1,eeif ;borra la bandera de fin de escritura
          bcf      eecon1,wren ;deshabilita la escritura en memoria EEPROM
          bcf      status,5 ;se ubica en el primer banco de RAM
          retlw    0

inicio     bsf      status,5 ;se ubica en el segundo banco de RAM
          movlw    0f0h   ;se carga el registro W con 0f0
          movwf    trisa   ;se programan los pines del puerto A como salidas
          movlw    0ffh   ;se carga el registro W con ff
          movwf    trisb   ;se programan los pines del puerto B como entradas
          bcf      status,5 ;se ubica en el primer banco de memoria RAM
          clrf    eeadr   ;cuando se enciende el sistema se verifica
          call    leer    ;que el dato guardado en memoria esté entre 0 y 9
          movlw    0ah    ;la prueba se hace porque la primera vez que
          subwf    conta2,w ;se encienda el sistema se puede tener un número
          btfss    status,c ;fuera del rango, para las ocasiones
          goto    ciclo   ;posteriores el proceso es invisible

ini2       clrf    conta   ;inicia contador en cero
          clrf    eedata  ;inicia dato de memoria en 0

ciclo      call    leer    ;leer memoria, devuelve dato en W
          movwf    ptoa   ;pasa el valor de W al puerto A (display)
          call    retardo ;retardo esperando que suelten la tecla

pulsa      btfsc    ptob,0 ;pregunta si el pulsador está oprimido
          goto     pulsa  ;si no lo está continúa revisándolo
          call    retardo ;si está oprimido retarda 100 milisegundos
          btfsc    ptob,0 ;para comprobar
          goto     pulsa  ;si no lo está vuelve a revisar
          incf    conta   ;si lo confirma incrementa el contador
          movf    conta,w ;carga el registro W con el valor del conteo
          movwf    eedata  ;el dato del conteo lo guarda en memoria
          call    escrib  ;para recuperarlo en caso de un apagón
          movf    conta,w
          xorlw   0ah    ;hace operación xor para ver si es igual a 0ah
          btfss    status,z ;prueba si el contador llegó a 0ah (diez)
          goto     ciclo   ;si no es igual se incrementa normalmente

          goto    ini2    ;
          end     ;;

=====

; Fusibles de programación
; Osc XT
; Watchdog OFF
; Code protect OFF
; Power-Up-Timer ON
; Micro. PIC16F84
=====
```

Figura 2.54. Programa que utiliza la memoria EEPROM de datos

Proyecto N°8: Control de un motor paso a paso

Por sus características y precisión de movimientos, los motores de paso se constituyen en un elemento muy valioso a la hora de diseñar un sistema de control o una máquina especializada. Además, dado que las señales necesarias para controlar esta clase de motores son de naturaleza digital, éstos se pueden conectar fácilmente a sistemas de ese tipo.

En este proyecto emplearemos un microcontrolador PIC como elemento principal para guiar la posición de un motor de esta clase. El proyecto se ha dividido en varias partes. Primero, se hace una breve descripción del tipo de motor que se utilizará, luego se muestra la forma de conectarlo al microcontrolador para conseguir movimientos de paso completo y por último se hace un ejercicio con movimientos de medio paso.

Motores paso a paso

Para el ejercicio se utiliza un motor de 4 bobinas. Este tipo de motores se identifica porque tienen 5, 6 u 8 cables. En el primer caso existe un cable que es común a todos los demás, para el de 6 cables se tiene un común para cada pareja de bobinas y en el de 8 cables cada bobina es independiente. Para el proyecto empleamos uno de 8 cables. En la figura 2.55 se muestra su estructura, incluyendo los colores de los cables (que son estándares).

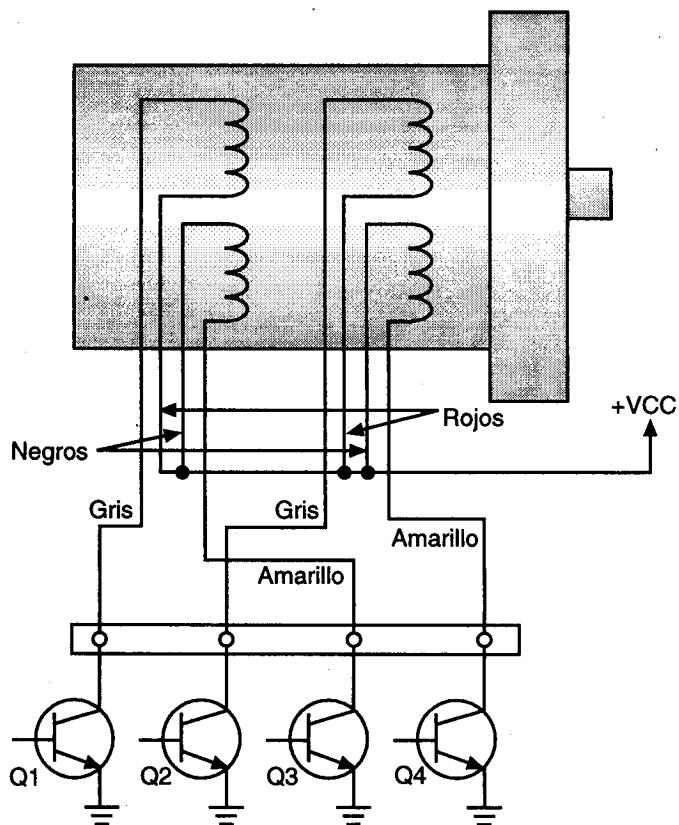


Figura 2.55. Conexión de los transistores que manejan el motor

Este tipo de motores tiene la ventaja de operar con una sola fuente, mientras que los motores de dos bobinas requieren polaridad positiva y negativa, haciéndose necesario utilizar circuitos en puente. En la figura 2.55 también se muestra la forma de conectar el motor de 8 cables. Debe notarse que los cables rojos y negros se unen para conectarlos a la fuente positiva. De aquí se puede deducir como sería la conexión para 6 y 5 cables. El transistor que se utiliza para activar las bobinas es el típico TIP122, que es un Darlington con protección para el manejo de cargas induc- tivas (diodo interno). Nótese que se ha invertido el orden de las bobinas de los transistores Q2 y Q3.

El movimiento del motor se consigue energizando las bobinas en un orden determinado. Esto se puede hacer de dos formas: La primera consiste en activar una sola bobina a la vez (manejo por ola), la segunda es el manejo de dos fases o bobinas al mismo tiempo. Con este último método se puede conseguir mayor torque, por eso es el método que empleamos en estos ejemplos. La figura 2.56 muestra los diagramas de tiempo de los dos métodos anteriores. De ahora en adelante se supondrá que el motor se ha conectado de la forma que se muestra en la figura 2.55; esto se hace con el fin de no repetir la misma gráfica en cada ejercicio.

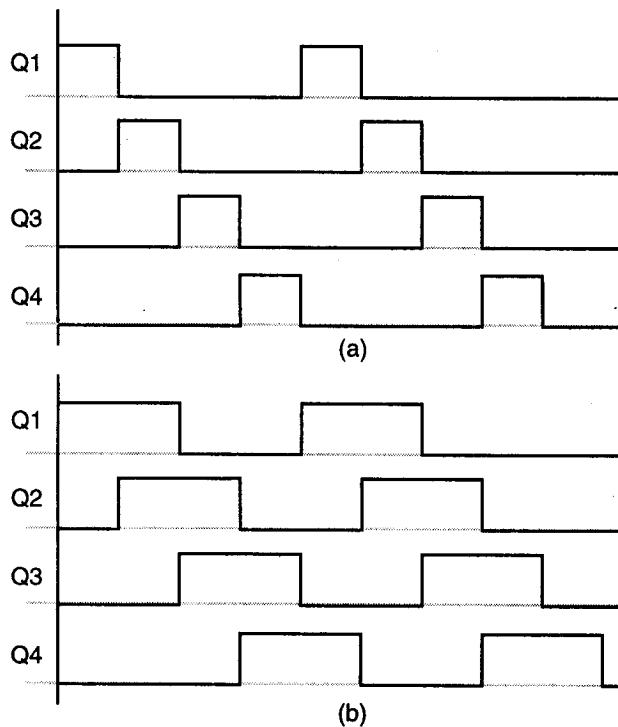


Figura 2.56. Secuencias para manejar las bobinas del motor de paso.
a) una fase, b) dos fases

Control del motor a pasos completos

El motor que se escogió para el ejercicio tiene un paso de 7.5° , es decir que en un círculo completo (360°) tiene 48 posiciones de reposo. Cada uno de los pasos se logra cuando se encienden las bobinas correspondientes según la secuencia. También se debe aclarar que el punto común de las bobinas del motor se debe conectar a $+5V$, ya que este es el dato suministrado por el fabricante en la placa de datos del mismo.

Para controlar el motor se utiliza el microcontrolador PIC16F84, el cual está empleando un cristal de 4 MHz y utiliza los cuatro pines bajos del puerto A (RA0-RA3) como salidas, para activar o desactivar los transistores de potencia TIP122 que manejan cada una de las bobinas. En la figura 2.57 se muestra el diagrama esquemático correspondiente, el cual incluye una tabla con la secuencia que se debe seguir para conseguir que el motor gire.

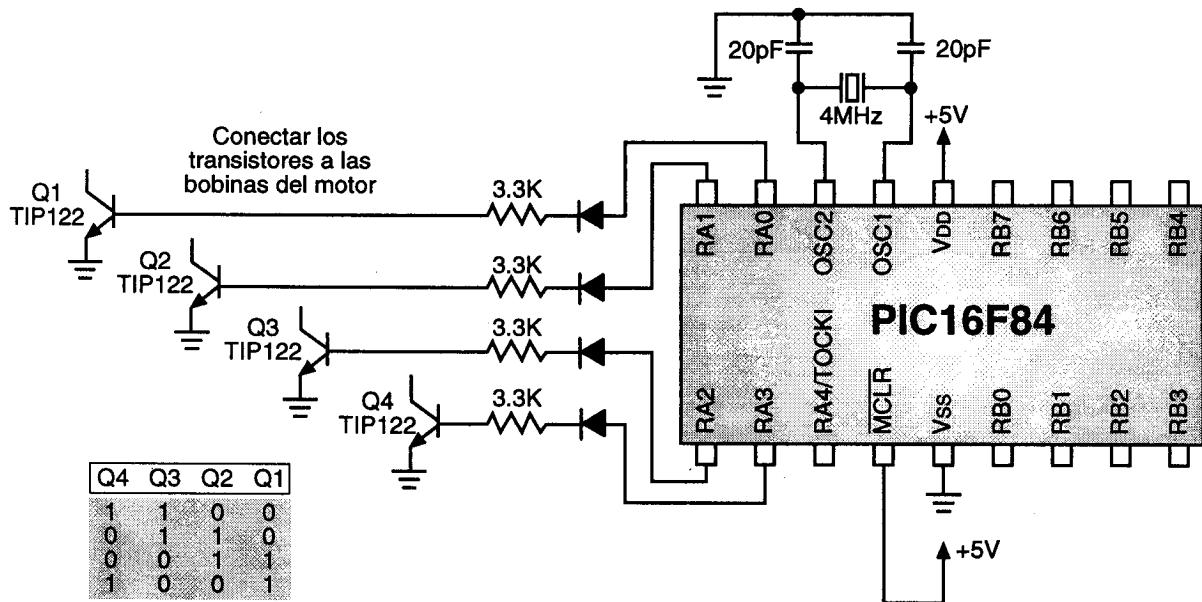


Figura 2.57. Diagrama del circuito para mover el motor a paso completo

Como se había mencionado anteriormente, el control del motor se hará con dos bobinas energizadas al mismo tiempo. Viendo la tabla de la secuencia se puede deducir que con simples rotaciones de un registro se puede conseguir el objetivo (nótese que la tabla tiene cuatro valores diferentes). Cada vez que se encienden las dos bobinas correspondientes a un valor de la tabla, este se mueve un paso. Por lo tanto, dado que el motor es de 48 pasos por vuelta, se debe repetir los valores de la tabla 12 veces para conseguir que el eje del motor de un giro completo.

Si se desea que el motor gire en sentido contrario al que se logró con la secuencia anterior, sólo se debe invertir la secuencia, es decir, leer la tabla en sentido inverso (de abajo hacia arriba). Cuando el motor no está en movimiento se debe garantizar que todas las bobinas estén desenergizadas. Esto se hace con el fin de evitar posibles daños del motor.

El primer ejercicio que vamos a implementar consiste en hacer que el motor de un giro completo en un sentido, permanezca quieto un momento y luego gire en sentido contrario, para repetir el mismo ciclo. Dado que la secuencia es simple, el programa del microcontrolador también lo es. La figura 2.58 muestra el listado completo.

```

;ESTE PROGRAMA PERMITE CONTROLAR EL GIRO DE UN MOTOR DE
;PASO, A PASOS COMPLETOS
; ***** PIC16F84 *****
; ***** WDT = OFF, OSC = XT *****
TMRO EQU 1H ;CONTADOR DE TIEMPO REAL
PC EQU 2H ;CONTADOR DE PROGRAMA
STATUS EQU 3H ;REGISTRO DE ESTADOS
PTOA EQU 5H ;PUERTOS
PTOB EQU 6H
R0D EQU 0DH
R10 EQU 10H
R11 EQU 11H
Z EQU 2H ;BANDERA DE CERO
C EQU 0H ;BANDERA DE CARRY
W EQU 0H ;PARA ALMACENAR EN W
R EQU 1H ;PARA ALMACENAR EN EL REGISTRO

ORG 00 ;Vector de reset
GOTO INICIO ;Inicia programa principal
ORG 05H
RETARDO CLRF TMRO ;Retardo de 8.192 ms
NOP RETARDO ;empleando el TMRO
RETAR1 BTFSS TMRO,7
GOTO RETAR1
RETLW 0

RETARD2 MOVLW 38H ;Retardo auxiliar
MOVWF R0D ;del giro del motor
DECR2 CALL RETARDO
CALL RETARDO
DECFSZ R0D,R
GOTO DECR2
RETLW 0

PASOAD BCF STATUS,C ;Esta rutina hace mover el
RRF R10,R ;motor solo un paso en un
BTFSC STATUS,C ;sentido, lo que hace es
BSF R10,3 ;rotar los dos unos lógicos que
MOVF R10,W ;encienden dos transistores
MOVWF PTOA ;al tiempo
CALL RETARDO
CLRF PTOA
RETLW 0

PASOAT BCF STATUS,C ;Esta rutina hace mover el
RLF R10,R ;motor un paso, en sentido
BTFSC R10,4 ;contrario a la anterior
BSF R10,0
MOVF R10,W
MOVWF PTOA
CALL RETARDO
CLRF PTOA
RETLW 0

; ***** PROGRAMA PRINCIPAL *****
INICIO MOVLW 00H ;Los puertos se programan de
TRIS PTOA ;acuerdo al circuito
MOVLW 0FFH
TRIS PT0B
MOVLW 0C5H ;se programa el TMRO con
OPTION ;preescala de 64
CLRF PTOA
CLRF R10
BSF R10,2 ;Estos son los unos que rotan
BSF R10,1 ;en el registro R10

BEGIN MOVLW D'48' ;En este ciclo el motor hace un
MOVWF R11 ;giro completo

```

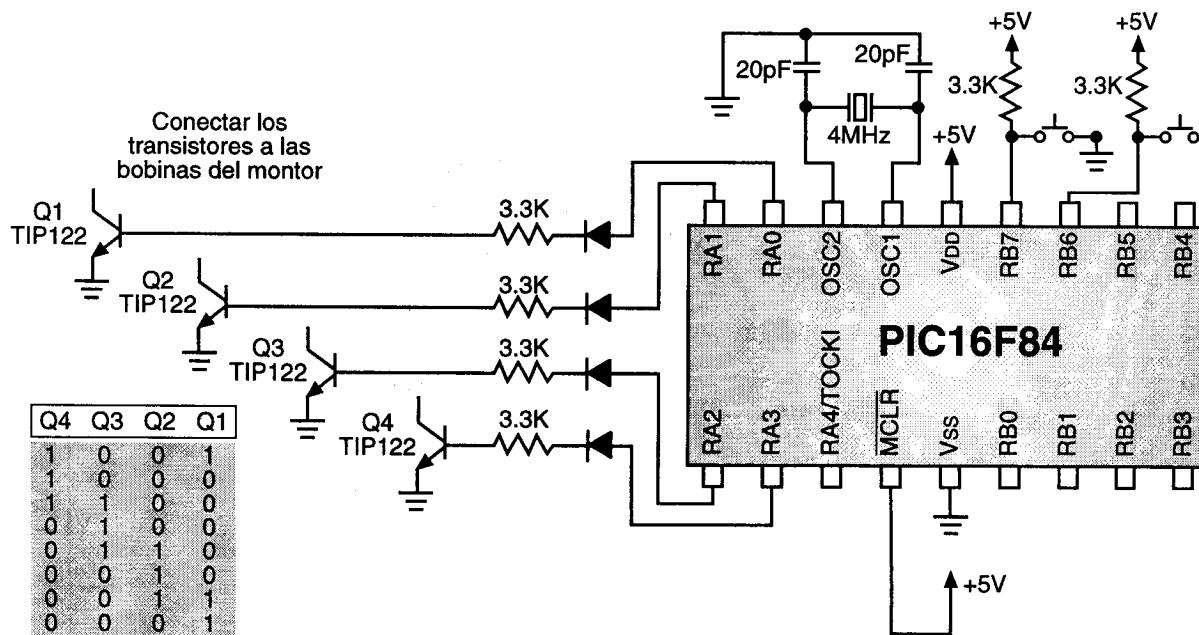
VUEL	CALL PASOAD DECFSZ R11,R GOTO VUEL CALL RETARD2 CALL RETARD2 MOVLW D'48' MOVWF R11	;se llama 48 veces la rutina ;que hace mover el motor 1 paso
VUEL2	CALL PASOAT DECFSZ R11,R GOTO VUEL2 CALL RETARD2 CALL RETARD2 GOTO BEGIN END	;se llama 48 veces la rutina ;que lo mueve un paso en ;sentido contrario

Figura 2.58. Programa para controlar el motor a pasos completos

Las rutinas llamadas PASOAD y PASOAT se encargan de hacer la rotación de las señales que controlan la activación y desactivación de las bobinas del motor. El estado de las rotaciones se conserva en el registro de memoria RAM R10 y se pasa al puerto A cada vez que se desea mover el motor un paso. En el bloque principal del programa solamente se requiere cargar un contador con el número de pasos deseado (en este caso 48) para llamar la rutina correspondiente igual número de veces. La rutina de retardo está basada en el temporizador interno del microcontrolador para garantizar su precisión.

Control del motor a 1/2 paso

En el primer ejercicio se hizo girar el motor avanzando en pasos completos. Ahora la idea es hacerlo girar avanzando medio paso cada vez que se deseé. Con esto podremos conseguir que en una vuelta completa del eje del motor, se tengan 96 intervalos diferentes, o sea que el eje se desplaza 3.75° en cada paso. Esta clase de movimientos de precisión hacen del motor de paso un elemento muy útil.

**Figura 2.59.** Circuito para controlar el motor a 1/2 paso

Para este ejercicio la idea es hacer que el motor gire en el sentido que se le indique oprimiendo uno de los dos pulsadores dispuestos para tal fin. El motor debe moverse tanto tiempo como esté oprimido el interruptor correspondiente. Para este ejercicio empleamos el mismo circuito del ejemplo anterior, pero con unas pequeñas variaciones. En la figura 2.59 se muestra el diagrama esquemático del circuito de control.

En los pines RB7 y RB6 del microcontrolador se han dispuesto los interruptores pulsadores que se encargan de controlar el sentido y la duración del movimiento. Además, en la gráfica se incluye la tabla con la secuencia que se requiere para que el motor pueda moverse a 1/2 paso. Como se ve, ahora la tabla tiene ocho valores diferentes. El microcontrolador se debe encargar de hacer la lectura de dichos valores en orden ascendente o descendente para obtener el valor que debe entregar por el puerto A para manejar las bobinas del motor.

El software que se graba en el microcontrolador aparece en la figura 2.60. Para este ejemplo se han modificado las rutinas debido a que la secuencia no es una simple rotación de las bobinas. Los valores que debe entregar el microcontrolador se han escrito en una tabla, para que la estructura del programa sea la misma en el momento que se desee trabajar a 1/4 de paso e inclusive a 1/8 de paso. En estos dos últimos casos sólo se debe cambiar los valores de la tabla y el límite máximo del contador que se encarga de leerla. Como ejercicio el lector puede deducir la secuencia que se emplearía en movimientos a 1/4 y 1/8 de paso.

```
;ESTE PROGRAMA PERMITE CONTROLAR EL GIRO DE UN MOTOR DE PASO, A 1/2 PASO
;      ***** PIC16F84 *****
;      ***** WDT = OFF, OSC = XT, CP = OFF *****

TMRO EQU 1H          ;CONTADOR DE TIEMPO REAL
PC   EQU 2H          ;CONTADOR DE PROGRAMA
STATUS EQU 3H         ;REGISTRO DE ESTADOS
PTOA  EQU 5H          ;PUERTOS
PTOB  EQU 6H
R0D   EQU 0DH
R10   EQU 10H
Z    EQU 2H          ;BANDERA DE CERO
C    EQU 0H          ;BANDERA DE CARRY
W    EQU 0H          ;PARA ALMACENAR EN W
R    EQU 1H          ;PARA ALMACENAR EN EL REGISTRO
ADE  EQU 7            ;BITS DEL PTOB
ATRA EQU 6

;
***** PROGRAMA PRINCIPAL *****
;

ORG  00              ;Vector de reset
GOTO INICIO          ;Va a iniciar programa principal
ORG  05H

RETARDO CLRF TMRO   ;Retardo de 2.048 ms
NOP
RETAR1 BTFSS TMRO,5  ;usando el TMRO
GOTO RETAR1
RETLW 0

PASOAD INCF R10,R    ;Esta rutina mueve el motor
MOVF R10,W
CALL SECUEN          ;medio paso en un sentido
MOVWF PTOA           ;está basada en una tabla
                      ;que contiene el estado de
```

	CALL CLRF MOVLW XORWF BTFS GOTO MOVLW MOVWF SALIR	RETARDO PTOA 07H R10,W STATUS,Z SALIR 0FFH R10 RETLW	;activación de los transistores ;cuando se termine la ultima ;posición de la tabla se ;empieza nuevamente
PASOAT	DECFS MOVF CALL MOVWF CALL CLRF MOVLW ANDWF BTFS GOTO MOVLW MOVWF	R10,R R10,W SECUEN PTOA RETARDO PTOA 0FFH R10,W STATUS,Z SALE 08H R10 RETLW	;esta rutina mueve el motor en ;sentido contrario al anterior ;está basada en la misma tabla ;que controla las salidas
SALE			;cuando se termina la tabla ;vuelve a empezar nuevamente
SECUEN	ADDWF RETLW RETLW RETLW RETLW RETLW RETLW RETLW RETLW RETLW RETLW	PC,R B'00000100' B'00000110' B'00000010' B'00000011' B'00000001' B'00001001' B'00001000' B'00001100' 0	;Esta tabla contiene el ;estado de las salidas ;Si se desea trabajar ;a 1/4 de paso sólo se debe ;cambiar la tabla
INICIO	MOVLW TRIS MOVLW TRIS MOVLW OPTION CLRF MOVLW MOVWF	00H PTOA 0FFH PTOB 0C5H PTOA 03H R10	;programación de puertos ;según el circuito ;se programa preescala en 64 ;para el TMRO ;se inicializa la secuencia ;en algún estado
PRUE1	BTFS GOTO CALL CLRF GOTO	PTOB,ADE PRUE2 PASOAD PTOA PRUE1	;si se oprime uno de los ;pulsadores se llama la rutina ;que mueve el motor medio paso
PRUE2	BTFS GOTO CALL CLRF GOTO	PTOB,ATRA PRUE1 PASOAT PTOA PRUE2	;si se oprime el otro pulsador ;el motor gira en sentido ;opuesto
	END		

Figura 2.60. Programa de control del motor a 1/2 paso

Las rutinas PASOAD y PASOAT se encargan de incrementar o decrementar el contador que sirve como puntero de la tabla. Además, según el sentido en que esté girando el motor, las rutinas hacen la verificación correspondiente cuando llega al final de la tabla, para volver a iniciar la lectura de la misma. La rutina SECUEN

Capítulo 2. Proyectos con el PIC16F84

contiene los valores que se deben entregar por el puerto del microcontrolador para energizar las bobinas correspondientes. Esta tabla es la que se debe modificar para conseguir movimiento a 1/4 o 1/8 de paso.

Como parte del ejercicio, el estudiante podría insertar un retardo entre cada movimiento (llamando una subrutina con un retardo largo), para ver en detalle como se mueve el motor y comprobar que en un círculo completo se tienen 96 escalas o posiciones de reposo.



Capítulo 3

El PIC16C71

- **Arquitectura**
- **Características especiales**
- **El PIC16C710/711/715**

Las personas que se mueven entre los mundos análogo y digital encontrarán en este microcontrolador la solución a muchos de sus problemas de diseño. Como se recordará, la más sencilla de las lecturas de cantidades análogas a través de sistemas digitales se convertía en toda una pesadilla: la necesidad de fuentes duales en los convertidores (anteriormente se trabajaba hasta con tres voltajes de alimentación diferentes), el alambrado y conexión de buses de datos, direcciones y control, entre otras razones, aumentaban las dificultades. Este microcontrolador promete resolver con ventaja estas situaciones, ya que posee un convertidor análogo a digital interno bastante interesante:

- Cuatro canales de entrada
- Tiempo de conversión mínimo de $20\ \mu s$
- Voltaje de referencia interno o externo
- Resolución de 8 bits con precisión de ± 1 LSB
- Rango de entrada análoga desde V_{ss} hasta V_{ref}

Pines y funciones

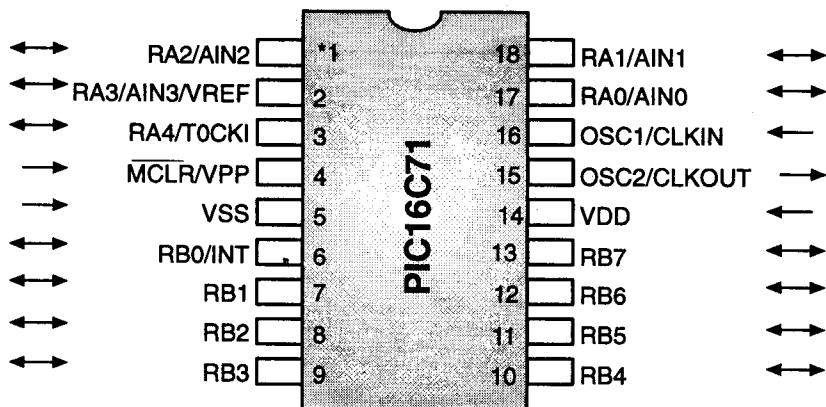


Figura 3.1. Diagrama de pines del PIC16C71

El PIC16C71 es un microcontrolador de *Microchip Technology* fabricado en tecnología CMOS, su consumo de potencia es muy bajo y además es completamente estático (esto quiere decir que el reloj puede detenerse y los datos de la memoria no se pierden).

El encapsulado más común para el microcontrolador es el DIP (*Dual In-line Pin*) de 18 pines, propio para usarlo en experimentación. La referencia completa es 16C71-04/P, para el dispositivo que utiliza reloj de 4 MHz. Sin embargo, hay otros tipos de encapsulado que se pueden utilizar según el diseño y la aplicación que se quiere realizar. Por ejemplo, el encapsulado tipo *surface mount* (montaje superficial) tiene un reducido tamaño y bajo costo, que lo hace propio para producciones en serie o para utilizarlo en lugares de espacio muy reducido, la figura 3.2 muestra los tipos de empaque que puede tener el integrado.

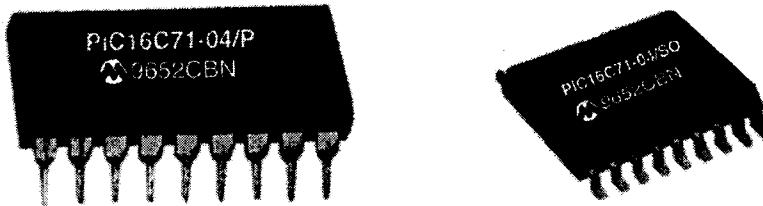


Figura 3.2. Tipos de encapsulado

Puertos del microcontrolador

Los puertos son el puente entre el microcontrolador y el mundo exterior. Son líneas que se pueden configurar como entradas o salidas digitales y algunas de ellas se pueden configurar como entradas analógicas, las cuales aceptan voltajes continuos que pueden variar entre cero y cinco voltios. Estas últimas son las que toma el convertidor analógico a digital para realizar su trabajo.

El PIC16C71 tiene dos puertos. El puerto A con 5 líneas y el puerto B con 8 líneas, figura 3.3. Cada pin se puede configurar como entrada o como salida independiente, programando un par de registros diseñados para tal fin. En ese registro un "0" configura el pin del puerto correspondiente como salida y un "1" lo configura como entrada.

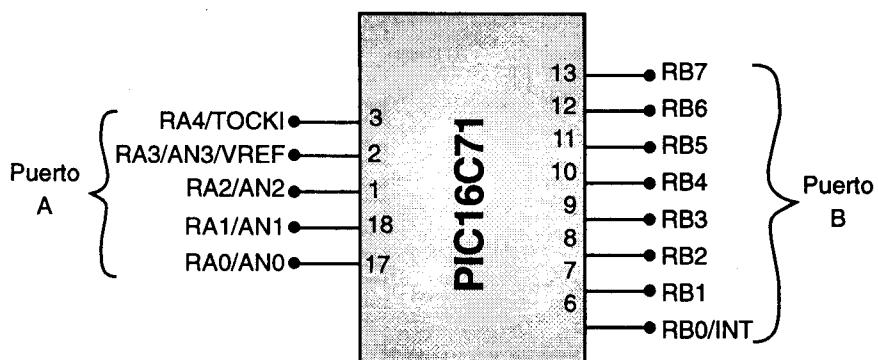


Figura 3.3. Puertos del PIC16C71

El puerto B tiene internamente unas resistencias de *pull-up* conectadas a sus pines (sirven para fijar el pin a un nivel de cinco voltios), su uso puede ser habilitado o deshabilitado bajo control del programa. Todas las resistencias de *pull-up* se conectan o se desconectan a la vez, usando el bit llamado RBPU que se encuentra en el registro (posición de memoria RAM) llamado OPTION. La resistencia de *pull-up* es desconectada automáticamente en un pin si este se programa como salida. Cuando ocurre un *reset* se deshabilitan todas las resistencias de *pull-up*, por lo tanto, si se están usando deben ser habilitadas nuevamente por control del programa. El pin RB0/INT puede ser configurado por software para que funcione como interrupción externa, para ello se utilizan algunos bits de los registros INTCON y OPTION.

El pin RA4/TOCKI del puerto A puede ser configurado como un pin de entrada/salida o como entrada del temporizador/contador. Cuando este pin se programa como entrada digital, funciona como un disparador de *Schmitt* (*Schmitt trigger*), puede reconocer señales un poco distorsionadas y llevarlas a niveles lógicos (cero y cinco

voltios). Cuando se usa como salida digital se comporta como colector abierto (*open collector*); por lo tanto se debe poner una resistencia de *pull-up* (resistencia externa conectada a un nivel de cinco voltios). Como salida, la lógica es inversa: un "0" escrito al pin del puerto entrega a la salida un "1" lógico. Este pin como salida no puede manejar cargas en modo fuente, sólo en el modo sumidero.

Como este dispositivo es de tecnología CMOS, todos los pines deben estar conectados a alguna parte, nunca dejarlos al aire porque se puede dañar el integrado. Los pines que no se estén usando se deben conectar a la fuente de alimentación de +5V, como se muestra en la figura 3.4.

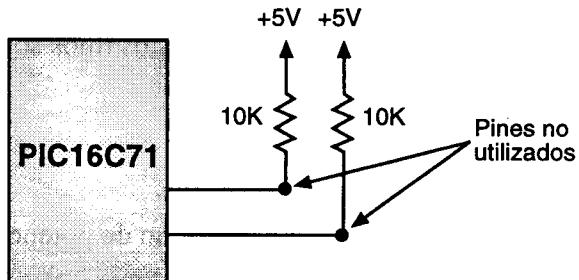


Figura 3.4. Los puertos no utilizados se deben conectar a la fuente

La máxima capacidad de corriente de cada uno de los pines de los puertos en modo sumidero (*sink*) es de 25 mA y en modo fuente (*source*) es de 20 mA, figura 3.5. La máxima capacidad de corriente total de los puertos es:

	PUERTO A	PUERTO B
Modo sumidero	80 mA	150 mA
Modo fuente	50 mA	100 mA

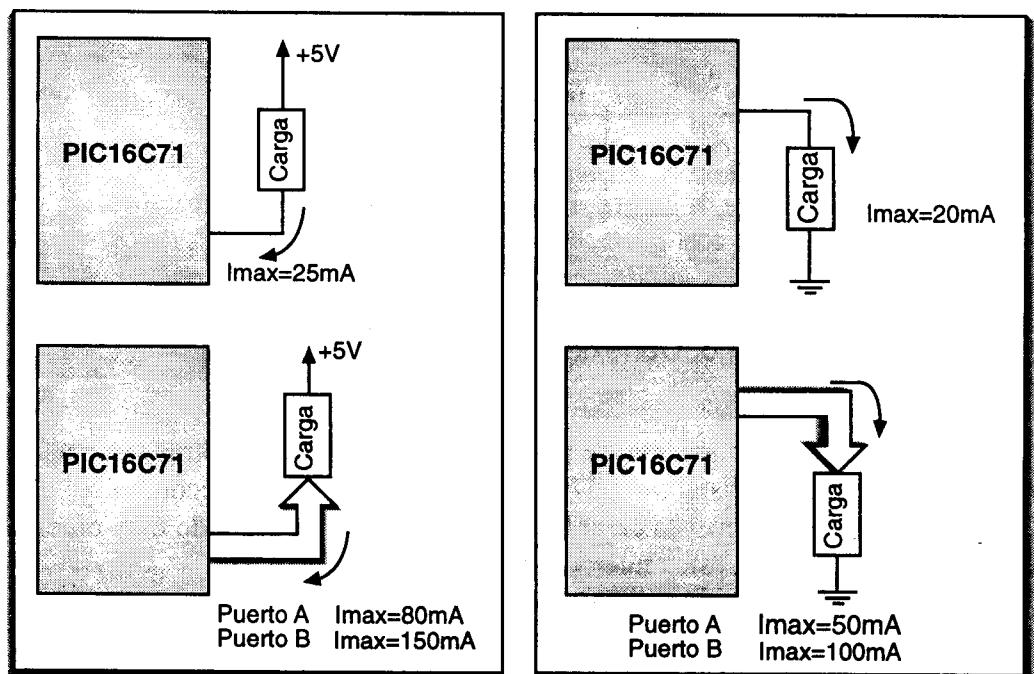


Figura 3.5. Capacidad de corriente del PIC16C71

El consumo de corriente del microcontrolador para su funcionamiento depende del voltaje de operación, la frecuencia y las cargas que tengan sus pines. Para un reloj de 4 MHz el consumo es de aproximadamente 2 mA; aunque este se puede reducir a 40 microamperios cuando se entra en el modo *sleep* (en este modo el micro se detiene y disminuye el consumo de potencia), se sale de ese estado cuando se produce alguna condición especial que veremos más adelante.

El oscilador externo

Todo microcontrolador requiere un circuito externo que le indique la velocidad a la que debe trabajar. Este circuito, que se conoce como oscilador o reloj, es muy simple pero de vital importancia para el buen funcionamiento del sistema. El PIC16C71 puede utilizar cuatro tipos de reloj diferentes. Estos tipos son:

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

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

El tipo de oscilador que se sugiere para las prácticas es el cristal de 4 MHz, porque garantiza mayor precisión y un buen arranque del microcontrolador. Internamente esta frecuencia es dividida por cuatro, lo que hace que la frecuencia efectiva de trabajo sea de 1 MHz, por lo que cada instrucción se ejecuta en un microsegundo. El cristal debe ir acompañado de dos condensadores y se conecta como se muestra en la figura 3.6.

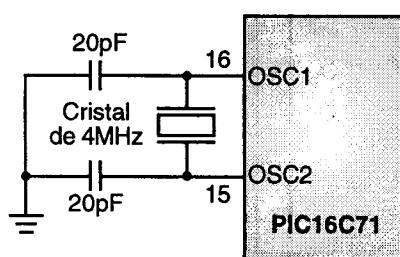


Figura 3.6. Conexión de un oscilador a cristal

Dependiendo de la aplicación, se pueden utilizar cristales de otras frecuencias; por ejemplo, se usa el cristal de 3.579545 MHz porque es muy económico, el de 32.768 KHz cuando se necesita crear bases de tiempo de un segundo muy precisas. El límite de velocidad en estos microcontroladores es de 20 MHz.

Si no se requiere mucha precisión en el oscilador y se quiere economizar dinero, se puede utilizar una resistencia y un condensador, como se muestra en la figura 3.7.

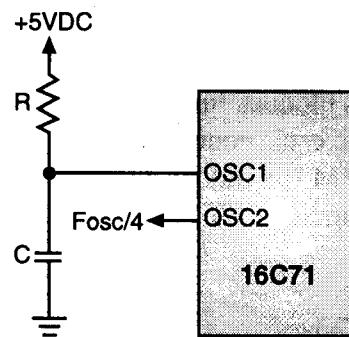


Figura 3.7. Conexión de un oscilador RC

Reset

En los microcontroladores se requiere un pin de *reset* para reiniciar el funcionamiento del sistema cuando sea necesario, ya sea por una falla que se presente o porque así se halla diseñado el sistema. El pin de *reset* en los PIC es llamado MCLR (*master clear*).

El PIC16C71 admite diferentes tipos de *reset*:

- Al encendido (*Power On Reset*)
- Pulso en el pin MCLR durante la operación normal
- Pulso en el pin MCLR durante el modo de bajo consumo
- El rebase del conteo del circuito de vigilancia (*watchdog*) durante la operación normal
- El rebase del conteo del circuito de vigilancia (*watchdog*) durante el modo de bajo consumo (*sleep*)

El *reset* al encendido se consigue gracias a dos temporizadores. El primero de ellos es el OST (*Oscillator Start-Up Timer*: Temporizador de encendido del oscilador), orientado a mantener el microcontrolador en *reset* hasta que el oscilador del cristal es estable. El segundo es el PWRT (*Power-Up Timer* : Temporizador al encendido), que provee un retardo fijo de 72 ms en el encendido únicamente, diseñado para mantener el dispositivo en *reset* mientras la fuente se estabiliza. Para utilizar estos temporizadores sólo se requiere conectar el pin MCLR a la fuente de alimentación, evitándose utilizar las tradicionales redes RC externas en el pin de *reset*.

El *reset* por MCLR se consigue llevando momentáneamente este pin a un estado lógico bajo, mientras que el WDT produce el *reset* cuando el temporizador rebasa su cuenta, o sea que pasa de OFFh a 00h. Cuando se quiere tener control sobre el *reset* del sistema se puede conectar un botón como se muestra en la figura 3.8.

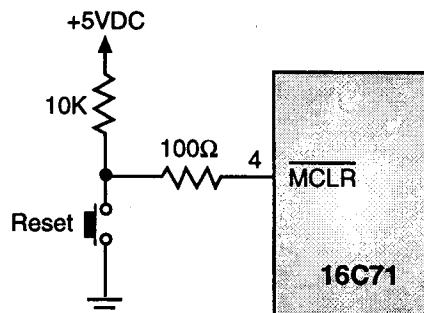


Figura 3.8. Conexión del botón de *reset*

Arquitectura

Este término se refiere a los bloques funcionales internos que conforman el microcontrolador y la forma en que están conectados, por ejemplo la memoria FLASH (de programa), la memoria RAM (de datos), los puertos, la lógica de control que permite que todo el conjunto funcione, etc.

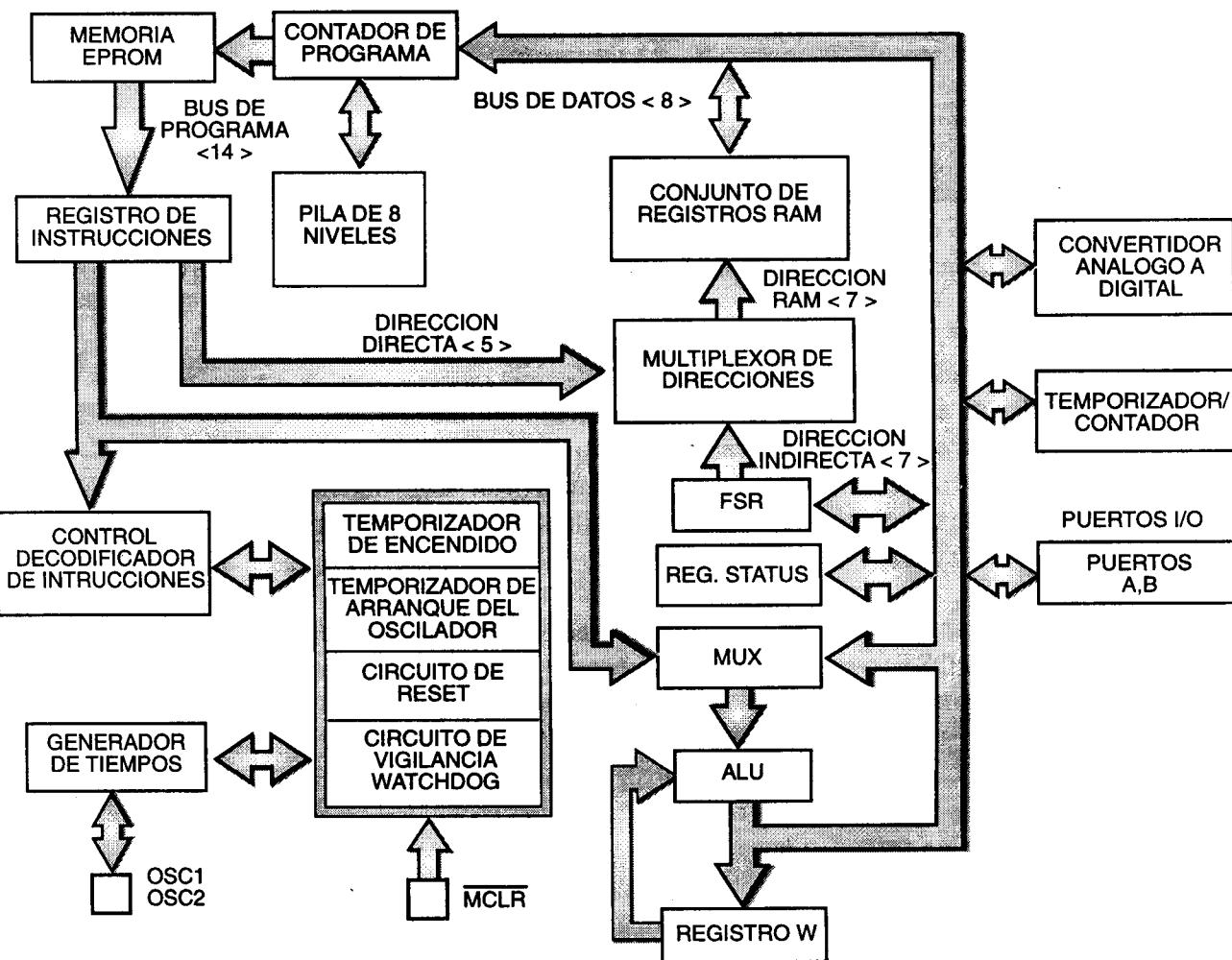


Figura 3.9. Arquitectura interna del PIC16C71

La figura 3.9 muestra la arquitectura general del PIC16C71, en ella se pueden apreciar los diferentes bloques que lo componen y la forma en que se conectan. Se muestra la conexión de los puertos, las memorias de datos y de programa, los bloques especiales como el watchdog, los temporizadores de arranque, el oscilador, etc.

Todos los elementos se conectan entre sí por medio de buses. Un bus es un conjunto de líneas que transportan información entre dos o más módulos. Vale la pena destacar que el PIC16C71 tiene un bloque especial conformado por el convertidor analógico a digital.

El PIC16C71 se basa en la arquitectura *Harvard*, en la cual el programa y los datos se pueden trabajar desde memorias separadas, lo que posibilita que las instrucciones y los datos posean longitudes diferentes. Esta misma estructura es la que permite la superposición de los ciclos de búsqueda y ejecución de las instrucciones, lo cual se ve reflejado en una mayor velocidad del microcontrolador.

Memoria de programa

Es una memoria de 1 Kbyte de longitud con palabras de 14 bits. En ella se graba, o almacena, el programa o códigos que el microcontrolador debe ejecutar. Dado que el PIC16C71 tiene un contador de programa de 13 bits, tiene una capacidad de direccionamiento de 8K x 14, pero solamente tiene implementado el primer 1K x 14 (0000h hasta 03FFh). Si se direccionan posiciones de memoria superiores a 3FFh se causará un solapamiento con el espacio del primer 1K. En la figura 3.10 se muestra el mapa de la memoria de programa.

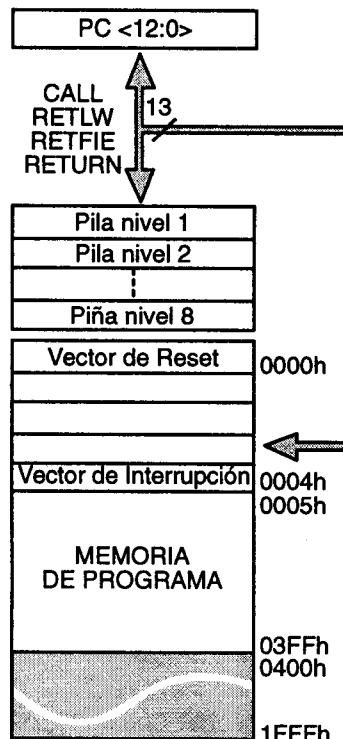


Figura 3.10. Mapa de la memoria de programa

Este microcontrolador se consigue en dos presentaciones, el de memoria tipo EPROM (posee ventana para borrado) que se utiliza para experimentar y el llamado OTP (One Time Programmable: Programable una sola vez) que se emplea en producción, cuando se está seguro que el programa funciona correctamente.

Vector de reset. Cuando ocurre un *reset* al microcontrolador, el contador de programa se pone en ceros (000H). Por esta razón, en la primera dirección del programa se debe escribir todo lo relacionado con la iniciación del mismo.

Vector de interrupción. Cuando el microcontrolador recibe una señal de interrupción, el contador de programa apunta a la dirección 04H de la memoria, por eso, allí se deben escribir todas las instrucciones necesarias para atender dicha interrupción.

Registros (Memoria RAM)

El PIC16C71 puede direccionar 128 posiciones de memoria RAM, pero solo tiene implementados físicamente los primeros 48 (0-2F en hexadecimal). De estos, los primeros 12 son registros que cumplen un propósito especial en el control del microcontrolador y los 36 siguientes son registros de uso general que se pueden usar para guardar los datos temporales de la tarea que se está ejecutando, figura 3.11.

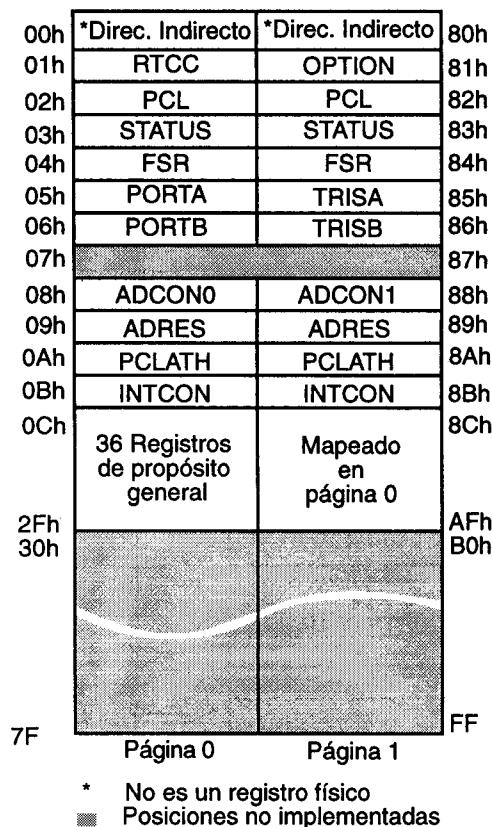


Figura 3.11. Registros del PIC16C71

Los registros están organizados como dos arreglos (páginas) de 128 posiciones de 8 bits cada una (128 x 8); todas las posiciones se pueden acceder directa o indirectamente (a través del registro selector FSR). Para seleccionar que página de memoria RAM se trabaja en un momento determinado, se utiliza el bit RP0 del registro STATUS. A continuación se hace una descripción de cada uno de los registros:

00h o INDO: Registro para direccionamiento indirecto de datos. Este no es un registro disponible físicamente; utiliza el contenido del FSR y el bit RP0 del registro STATUS para seleccionar indirectamente la memoria de datos o RAM del usuario; la instrucción determinará que se debe realizar con el registro señalado.

01h o TMR0: Temporizador/contador de 8 bits. Este se puede incrementar con una señal externa aplicada al pin RA4/TOCKI o de acuerdo a una señal interna proveniente del reloj de instrucciones del microcontrolador. La tasa de incremento del registro se puede determinar por medio de un preescalador, loca-

lizado en el registro OPTION. Como una mejora con respecto a sus antecesores, se le ha agregado la generación de interrupción cuando se rebasa la cuenta (el paso de OFFh a 00h).

02h o PCL: Contador de programa. Se utiliza para direccionar las palabras de 14 bits del programa del usuario que se encuentra almacenado en la memoria ROM; este contador de programas es de 13 bits de ancho, figura 1.12. Sobre el byte bajo, PCL se puede escribir o leer directamente, mientras que sobre el byte alto, no. El byte alto se maneja a través del registro PCLATH (0Ah). A diferencia de los PIC de primera generación, el 16C71 ante una condición de *reset* inicia el contador de programa con todos sus bits en "cero". Durante la ejecución normal del programa, y dado que todas las instrucciones ocupan sólo una posición de memoria, el contador se incrementa en uno con cada instrucción, a menos que se trate de alguna instrucción especial.

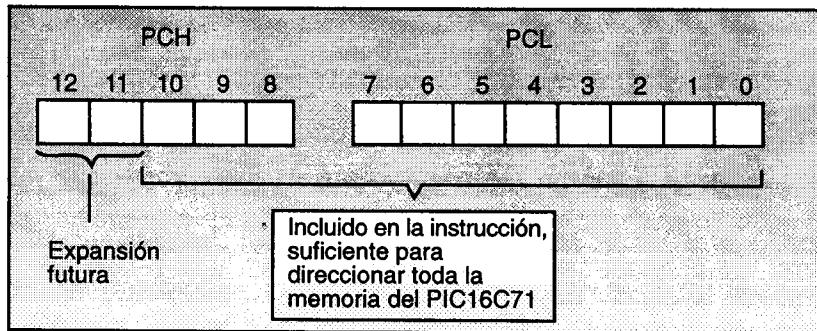


Figura 3.12. Contador de programa (13 bits)

En una instrucción CALL o GOTO, los bits PC<10:0> se cargan desde el código de operación de la instrucción, mientras que los bits PC<11:12> lo hacen desde el PCLATH<4:3>. Como solamente el primer 1K de memoria está implementado, el código de operación de la instrucción puede contener la dirección destino, eso quiere decir que se pueden hacer saltos y llamados a subrutinas sin necesidad de tener en cuenta la paginación de memoria de programa.

En otras instrucciones donde PCL es el destino, PC<12:8> se carga directamente desde el PCLATH<4:0>, por ejemplo en el caso de la instrucción ADDWF. Esto se debe tener en cuenta cuando se desea hacer lectura de tablas usando el comando: ADDWF PC,1 , en este caso se debe tener en cuenta que la tabla debe estar comprendida dentro de un solo bloque de 256 bytes (0-255, 256-511, etc.).

03h o STATUS: Registro de estados. Contiene el estado aritmético de la ALU, la causa del *reset* y los bits de preselección de página para la memoria de datos. La figura 3.13 muestra los bits correspondientes a este registro. Los bits 5 y 6 (RP0 y RP1) son los bits de selección de página para el direccionamiento directo de la memoria de datos; solamente RP0 se usa en los PIC16C71. RP1 se puede utilizar como un bit de propósito general de lectura/escritura. Los bits TO y PD no se pueden modificar por un proceso de escritura; ellos muestran la condición por la cual se occasionó el último *reset*.

Registro: STATUS											
	IRP	RP1	RP0	T0	PD	Z	DC	C			
	bit 7					bit 0					
Dirección:	03h										
condición de reset:	000??XXXb										
IRP:	Selector de página para direccionamiento indirecto. Este <i>bit</i> no se utiliza efectivamente en el PIC16C71, por lo que se puede utilizar como un <i>bit</i> de propósito general.										
RP1,0:	Selectores de página para direccionamiento directo. Solamente RP0 se utiliza en el PIC 16C71. RP1 se puede utilizar como un <i>bit</i> de propósito general.										
T0:	<i>Time Out o Bit</i> de finalización del temporizador. Se coloca en 0 cuando el circuito de vigilancia <i>Watchdog</i> finaliza la temporización.										
PD:	<i>Power Down o Bit</i> de bajo consumo. Se coloca en 0 por la instrucción <i>SLEEP</i> .										
Z:	<i>Zero o Bit</i> de cero. Se coloca en 1 cuando el resultado de una operación lógica o aritmética es cero.										
DC:	<i>Digit Carry o Bit</i> de acarreo de dígito. En operaciones aritméticas se activa cuando hay acarreo entre el <i>bit</i> 3 y el 4.										
C:	<i>Carry o Bit</i> de acarreo. En instrucciones aritméticas se activa cuando se presenta acarreo desde el <i>bit</i> más significativo del resultado.										

Figura 3.13. Registro de estados

04h o FSR: Registro selector de registros. En asocio con el registro IND0, se utiliza para seleccionar indirectamente los otros registros disponibles. Mientras que los antecesores poseían sólo 5 *bits* activos, en este microcontrolador se poseen los 8 *bits*. Si en el programa no se utilizan llamadas indirectas, este registro se puede utilizar como un registro de propósito general.

Para entender mejor el funcionamiento de este registro veamos un programa simple que borra el contenido de la memoria RAM, empleando direccionamiento indirecto.

	MOVLW	20	; inicializa el puntero en la memoria RAM
	MOVWF	FSR	; que se va a borrar
NEXT	CLRF	IND0	; borra el registro indexado (es decir el que está siendo direccionado por el FSR)
	INCF	FSR,R	; incrementa el puntero
	BTFSS	FSR,4	; pregunta si ya acabó el banco de mem.
	GOTO	NEXT	; sigue borrando los registros que faltan

continua

5h o PORTA: Puerto de Entrada/Salida de 5 bits. Este puerto, al igual que todos sus similares en los PIC, puede leerse o escribirse como si se tratara de un registro cualquiera. Los pines de este puerto, además de comportarse como entradas y salidas digitales, tienen funciones alternas, las cuales se muestran en la figura 3.14.

Pin	Función alterna
RA0 / AIN0	Entrada Análoga Canal 0
RA1 / AIN1	Entrada Análoga Canal 1
RA2 / AIN2	Entrada Análoga Canal 2
RA3 / AIN3 / Vref	Entrada Análoga Canal 3 o Voltaje de referencia (Vref)
RA4 / TOCK1	Entrada Externa de reloj para contador TMR0

Figura 3.14. Funciones alternas de los pines del puerto A

Dos *bits* en el registro ADCON1 (registro 88h) configuran los cuatro pines menos significativos como E/S digitales o entradas análogas. Bajo la condición de *reset*, estos pines se configuran como entradas análogas. El registro de control de este puerto (TRISA) está localizado en la página 1, en la posición 85h.

06h o PORTB: Puerto de Entrada/Salida de 8 bits. Al igual que en todos los PIC, este puede leerse o escribirse como si se tratara de un registro cualquiera; algunos de sus pines tienen funciones alternas en la generación de interrupciones. El registro de control para la configuración de la función de sus pines se localiza en la página 1, en la dirección 86h y se llama TRISB.

08h o ADCON0: Registro de control del convertidor. Este registro, junto con ADCON1 se dedica al control del convertidor análogo a digital. Los bits que éste contiene y la función que cumplen se muestran en la figura 3.15.

09h o ADRES: Registro para el resultado de la conversión. En esta posición se almacena el resultado digital de la conversión del valor análogo.

0Ah o PCLATH: Registro para la parte alta de la dirección. Este contiene la parte alta del contador de programa y no se puede acceder directamente.

0Bh o INTCON: Registro para el control de interrupciones. Es el encargado del manejo de las interrupciones y contiene los *bits* que se muestran en la figura 3.16.

81h u OPTION: Registro de configuración múltiple. Posee varios *bits* para configurar el preescalador, la interrupción externa, el timer y las características del puerto B. Los *bits* que contiene y sus funciones se muestran en la figura 3.17. El preescalador es compartido entre el RTCC y el WDT, su asignación es mutuamente excluyente ya que solamente puede uno de ellos ser preescalado a la vez.

Registro: ADCONO															
ADSC1	ADSC0	Res.	CHS1	CHS0	Go/Done	ADIF	ADON								
bit 7				bit 0											
Dirección:				08h											
condición de reset: 00000000b															
ADSC1:0: A/D Conversion Clock Select o Selector del reloj del convertidor.															
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>00</td><td>fosc/2</td></tr> <tr> <td>01</td><td>fosc/8</td></tr> <tr> <td>10</td><td>fosc/32</td></tr> <tr> <td>11</td><td>frc</td></tr> </table>		00	fosc/2					01	fosc/8	10	fosc/32	11	frc		
00	fosc/2														
01	fosc/8														
10	fosc/32														
11	frc														
Res: Reservado Este bit no se utiliza															
CHS1:0: Analog Channel Select o Selector del canal analógico.															
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>00</td><td>canal 0 (AIN0)</td></tr> <tr> <td>01</td><td>canal 1 (AIN1)</td></tr> <tr> <td>10</td><td>canal 2 (AIN2)</td></tr> <tr> <td>11</td><td>canal 3 (AIN3)</td></tr> </table>		00	canal 0 (AIN0)	01	canal 1 (AIN1)	10	canal 2 (AIN2)	11	canal 3 (AIN3)						
00	canal 0 (AIN0)														
01	canal 1 (AIN1)														
10	canal 2 (AIN2)														
11	canal 3 (AIN3)														
GO/DONE: Comienza la conversión cuando se coloca en 1. Al finalizar la conversión es puesto a 0 por hardware.															
ADIF: A/D Conversion Complete Interrupt Flag o Bandera de interrupción por finalización de la conversión. Se coloca en 1 cuando termina la conversión; debe ser puesta en 0 por programa.															
ADON: Bit para controlar la activación del convertidor. 0: módulo del convertidor desconectado y por lo tanto no exige corriente. 1: módulo del convertidor A/D en operación.															

Figura 3.15. Registro ADCONO

85h o TRISA: Registro de configuración del puerto A. Como ya se había mencionado, es el registro de control para el puerto A, cuando éste se trabaja digitalmente. Un «cero» en el bit correspondiente al pin lo configura como salida, mientras que un «uno» lo hace como entrada.

86h o TRISB: Registro de configuración del puerto B. Orientado hacia el control del puerto B. Son válidas las mismas consideraciones del registro anterior.

88h o ADCON1: Registro auxiliar para control del convertidor. Se dedica a la configuración de las entradas analógicas y solo destina dos bits para ello; el resto de bits permanece sin implementar. En la figura 3.18 se muestran las funciones de estos bits.

Registro: INTCON											
GIE	ADIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF				
bit 7				bit 0							
Dirección: 0Bh				condición de reset: 0000000Xb							
GIE:	Global Interrupt Enable o Habilitador general de interrupciones. 0: deshabilita todas las interrupciones 1: habilita las interrupciones										
ADIE:	A/D Conversion Interrupt Enable o Habilitación de interrupción del convertidor A/D 0: la deshabilita 1: la habilita										
TOIE:	TMRO Interrupt Enable o Habilitación de interrupción del temporizador TMRO. 0: la deshabilita 1: la habilita										
INTE:	INT Interrupt Enable o Habilitación de la interrupción INT. 0: la deshabilita 1: la habilita										
RBIE:	RBIF Interrupt Enable o Habilitación interrupción RBIF. 0: la deshabilita 1: la habilita										
TOIF:	TMRO Overflow Interrupt Flag o Bandera de la interrupción por sobrepasamiento del TMRO. Se coloca en 1 cuando el TMRO pasa de 0FFh a 00h; ésta debe ser puesta a 0 por programa.										
INTF:	INT Interrupt Flag o Bandera de interrupción por INT. Se coloca en 1 cuando la interrupción INT ocurre; ésta debe ser puesta a 0 por programa.										
RBIF:	RB Port Change Interrupt Flag o Bandera de interrupción por cambio en el puerto RB. Se coloca en 1 cuando una de las entradas RB<7:4> cambia; ésta debe ser puesta a 0 por programa.										

Figura 3.16. Registro INTCON

0Ch a 2Fh: Registros de propósito general. Estas 36 posiciones están implementadas en la memoria RAM estática, la cual conforma el área de trabajo del usuario; a ellas también se accede cuando en la página 1 se direccionan las posiciones 8Ch a AFh. Esto se ha diseñado así para evitar un excesivo cambio de páginas en el manejo de la RAM del usuario, agilizando los procesos que se estén llevando a cabo y descomplicando la labor del programador.

Registro de trabajo W. Este es el registro de trabajo principal, se comporta de manera similar al acumulador en los microprocesadores. Este registro participa en la mayoría de las instrucciones.

Registro: OPTION																																		
RBPU	INTED	GRTS	RTE	PSA	PS2	PS1	PSO																											
bit 7				bit 0																														
Dirección: 81h																																		
condición de reset: 1111111b																																		
RBPU: <i>PortB Pull-up Enable o Habilitación de pull-up del puerto B.</i>																																		
0: habilita las <i>pull-ups</i> internas 1: las deshabilita																																		
INTEDG: <i>INT Interrupt Edge Selector o Selector de flanco de la interrupción INT</i>																																		
0: flanco de bajada 1: flanco de subida																																		
RTS: <i>TMR0 Signal Source o Fuente de la señal de TMR0.</i>																																		
0: ciclo de instrucciones interno (Temporizador) 1: transición en el pin RA4/TOCK1 (Contador)																																		
RTE: <i>TMR0 Signal Edge o Flanco de la señal TMR0</i>																																		
0: incremento en transición de bajo a alto 1: incremento en transición de alto a bajo																																		
PSA: <i>Prescaler Assignment o Asignación del preescalador</i>																																		
0: TRM0 (Contador/Temporizador) 1: WDT (Circuito de vigilancia)																																		
PS2,1,0: <i>Prescaler Value o Valores del preescalador.</i>																																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>Valor</th> <th>TMR0</th> <th>WDT</th> </tr> </thead> <tbody> <tr><td>000</td><td>1:2</td><td>1:1</td></tr> <tr><td>001</td><td>1:4</td><td>1:2</td></tr> <tr><td>010</td><td>1:8</td><td>1:4</td></tr> <tr><td>011</td><td>1:16</td><td>1:8</td></tr> <tr><td>100</td><td>1:32</td><td>1:16</td></tr> <tr><td>101</td><td>1:64</td><td>1:32</td></tr> <tr><td>110</td><td>1:128</td><td>1:64</td></tr> <tr><td>111</td><td>1:256</td><td>1:128</td></tr> </tbody> </table>								Valor	TMR0	WDT	000	1:2	1:1	001	1:4	1:2	010	1:8	1:4	011	1:16	1:8	100	1:32	1:16	101	1:64	1:32	110	1:128	1:64	111	1:256	1:128
Valor	TMR0	WDT																																
000	1:2	1:1																																
001	1:4	1:2																																
010	1:8	1:4																																
011	1:16	1:8																																
100	1:32	1:16																																
101	1:64	1:32																																
110	1:128	1:64																																
111	1:256	1:128																																

Figura 3.17. Registro OPTION

Pila (Stack)

Estos registros no forman parte de ningún banco de memoria y no permiten el acceso por parte del usuario. Se usan para guardar el valor del contador de programa cuando se hace un llamado a una subrutina o cuando se atiende una interrupción; luego, cuando el micro regresa a seguir ejecutando su tarea normal, el contador de programa recupera su valor leyéndolo nuevamente desde la pila. El PIC16C71 tiene una pila de 8 niveles, esto significa que se pueden anidar 8 llamados a subrutina sin tener problemas.

Registro: ADCON1											
U	U	U	U	U	U	PCFG1	PCFG0				
bit 7				bit 0							
Dirección: 88h				condición de reset: 00000000b							
U: Unimplemented No implementadas. Estos bits se leen como ceros.											
PCFG1,0: Configuración de los pines RA0-RA3											
PCFG1,0	RA0,RA1	RA2	RA3	Vref							
00	E. análogas	E. análoga	E. análoga	Vdd							
01	E. análogas	E. análoga	Referencia	RA3							
10	E. análogas	E/S digital	E/S digital	Vdd							
11	E/S digitales	E/S digital	E/S digital	Vdd							

Figura 3.18. Registro ADCON1

Características especiales

Algunos elementos que forman parte de los PIC no se encuentran en microcontroladores de otros fabricantes, o simplemente representan alguna ventaja o facilidad a la hora de hacer un diseño. Veamos una breve descripción de las más significativas:

Convertidor Análogo a Digital

El módulo del convertidor posee cuatro entradas análogas multiplexadas a un solo circuito de muestreo y sostenimiento y a un convertidor. El voltaje de referencia del convertidor se puede configurar por software para que sea el voltaje de alimentación del microcontrolador (VDD) o un voltaje externo aplicado al pin RA3, en este caso, el valor mínimo que puede tomar el voltaje de referencia es de 3 voltios. Este convertidor es del tipo de aproximaciones sucesivas y el tiempo de conversión está en función del ciclo del oscilador, considerándose un tiempo mínimo de 20 μ s.

La conversión se inicia colocando el bit de control GO/DONE (ADCON0<2>) en «uno», previa selección del canal deseado y de configurar el registro ADCON1, el cual define los modos en los cuales trabaja el convertidor. Al final de la conversión, el bit GO/DONE se coloca en "cero" y la bandera de interrupción ADIF del convertidor (ADCON0<1>) en "uno", almacenándose el resultado de la conversión en el registro ADRES. Estas características permiten que el resultado de la conversión se pueda leer por medio de interrupciones o sin ellas.

Esquema del reloj del convertidor. El convertidor funciona con su propio reloj, derivado del reloj de entrada OSC1 o de su propio oscilador RC incorporado. El tiempo de conversión para cada bit es *tad* y el tiempo de conversión total es $10 \times tad$. La selección del reloj, a través de los bits ADC1,0 del registro ADCON0, se debe hacer de tal manera que *tad* sea al menos de 2 μ s. Al seleccionar el tipo de oscilador debe tenerse presente que la frecuencia varía con el voltaje, la temperatura y otros parámetros propios del proceso de fabricación del microcontrolador.

Círculo de vigilancia (*Watchdog Timer*)

Su función es restablecer el programa cuando este se ha perdido por fallas en la programación o por alguna razón externa. Es muy útil cuando se trabaja en ambientes con mucha interferencia o ruido electromagnético. Está conformado por un oscilador RC que se encuentra dentro del microcontrolador.

Este oscilador corre de manera independiente al oscilador principal. Cuando se habilita su funcionamiento, dicho circuito hace que el microcontrolador sufra un *reset* cada determinado tiempo (que se puede programar entre 18 ms y 2 segundos). Este *reset* lo puede evitar el usuario mediante una instrucción especial del microcontrolador (**CLRWDT**: borrar el conteo del watchdog), la cual se debe ejecutar antes de que termine el período nominal de dicho temporizador. De esta manera, si el programa se ha salido de su flujo normal, por algún ruido o interferencia externa, el sistema se reiniciará (cuando se acabe el tiempo programado y no se haya borrado el contador) y el programa puede restablecerse para continuar con su funcionamiento normal.

En las primeras prácticas no se utiliza el circuito de vigilancia para facilitar el trabajo; por eso, en el momento de programar el microcontrolador se debe seleccionar en los fusibles de configuración «*watchdog timer OFF*». Más adelante veremos algunos ejemplos que ilustran su funcionamiento y la manera de utilizarlo.

Temporizador de encendido (*Power-up Timer*)

Este proporciona un *reset* al microcontrolador en el momento de conectar la fuente de alimentación, lo que garantiza un arranque correcto del sistema. En el momento de grabar el micro se debe habilitar el fusible de configuración "*Power-up Timer*", para ello se debe seleccionar la opción "*ON*". Su tiempo de retardo es de 72 milisegundos.

Modo de bajo consumo (*sleep*)

Esta característica permite que el microcontrolador entre en un estado pasivo donde consume muy poca potencia. Cuando se entra en este modo el oscilador principal se detiene, pero el temporizador del circuito de vigilancia (watchdog) se reinicia y empieza su conteo nuevamente. Se entra en ese estado por la ejecución de una instrucción especial (llamada **SLEEP**) y se sale de él por alguna de las siguientes causas: cuando el microcontrolador sufre un *reset* por un pulso en el pin MCLR, porque el *watchdog* hace que se reinicie el sistema o porque ocurre una interrupción al sistema.

Interrupciones

Este microcontrolador incluye el manejo de interrupciones, lo cual representa grandes ventajas. El PIC16C71 posee cuatro fuentes de interrupción a saber:

- Interrupción externa en el pin RB0/INT
- Finalización del temporizador/contador
- Finalización de la conversión A/D
- Cambio de nivel en los pines RB4 a RB7

El registro 0Bh o INTCON contiene las banderas de las interrupciones INT, cambio en el puerto B y finalización del conteo del TMRO, al igual que el control para habilitar o deshabilitar cada una de las fuentes de interrupción, incluida la de fin de conversión análogo a digital. Sólo la bandera de fin de conversión (bit ADIF) reside en el registro 08h o ADCON0.

Si el *bit* GIE (*Global Interrupt Enable*) se coloca en 0, deshabilita todas las interrupciones. Cuando una interrupción es atendida, el bit GIE se coloca en 0 automáticamente para evitar interferencias con otras interrupciones que se pudieran presentar, la dirección de retorno se coloca en la pila y el PC se carga con la dirección 04h. Una vez en la rutina de servicio, la fuente de la interrupción se puede determinar examinando las banderas de interrupción. La bandera respectiva se debe colocar, por software, en cero antes de regresar de la interrupción, para evitar que se vuelva a detectar nuevamente la misma interrupción.

La instrucción RETFIE permite al usuario retornar de la interrupción, a la vez que habilita de nuevo las interrupciones, al colocar el *bit* GIE en uno. Debe tenerse presente que solamente el contador de programa es puesto en la pila al atenderse la interrupción; por lo tanto, es conveniente que el programador tenga cuidado con el registro de estados y el de trabajo, ya que se pueden producir resultados inesperados si dentro de ella se modifican.

Interrupción externa. Actúa sobre el pin RB0/INT y se puede configurar para activarse con el flanco de subida o el de bajada, de acuerdo al *bit* INTEDG (OPTION<6>). Cuando se presenta un flanco válido en el pin INT, la bandera INTF (INTCON<1>) se coloca en uno. La interrupción se puede deshabilitar colocando el *bit* de control INTE (INTCON<4>) en cero; cuando se atiende la interrupción, a través de la rutina de servicio, INTF se debe colocar en cero por *software*, antes de rehabilitar la interrupción. La interrupción puede reactivar al microcontrolador después de la instrucción SLEEP, si previamente el *bit* INTE fue habilitado. El estado del *bit* GIE decide si el procesador salta o no al vector de interrupción después de haberse reactivado.

Interrupción por finalización de la temporización. La superación del conteo máximo (0FFh) en el TMR0 colocará el *bit* TOIF en uno (INTCON<2>). El *bit* de control respectivo es TOIE (INTCON<5>).

Interrupción por cambio en el puerto RB. Un cambio en los pines del puerto B <7:4> colocará en uno el *bit* RBIF (INTCON<0>). El *bit* de control respectivo es RBIE (INTCON<3>).

Interrupción por conversión A/D. El convertidor A/D coloca en uno el bit ADIF (INTCON<1>) al finalizar una conversión. El bit de control respectivo es ADIE (INTCON<6>).

Fusibles de configuración

El PIC16C71 posee cinco fusibles, cada uno de los cuales es un *bit*. Estos fusibles se pueden programar para seleccionar varias configuraciones del dispositivo: tipo de

oscilador, protección de código, habilitación del circuito de vigilancia y el temporizador al encendido. Los *bits* se localizan en la posición de memoria 2007h, posición a la cual el usuario sólo tiene acceso durante la programación del microcontrolador. Cuando se programa la protección de código, el contenido de cada posición de la memoria no se puede leer completamente, de tal manera que el código del programa no se puede reconstruir. Adicionalmente, todas las posiciones de memoria del programa se protegen contra la reprogramación.

Una vez protegido el código, el fusible de protección sólo puede ser borrado (puesto a 1) si se borra toda la memoria del programa y la de datos.

Las pull-ups internas

Cada uno de los pines del puerto B tiene un débil elemento *pull-up* interno ($250\ \mu A$ típico); este elemento es automáticamente desconectado cuando el pin se configura como salida. Adicionalmente, el *bit* RBPU (OPTION<7>) controla todos estos elementos, los cuales están deshabilitados ante una condición de *reset*. Estos elementos *pull-up* son especialmente útiles cuando el microcontrolador va a colocarse en el modo de bajo consumo, ya que ayudan a no tener las entradas flotantes, significando una reducción en el consumo de corriente.

El conjunto de instrucciones

Estas se clasifican en orientadas a registros, orientadas al *bit* y operaciones literales y de control. Cada instrucción es una palabra de 14 *bits*, dividida en un código de operación, el cual especifica la orden a ejecutar y uno o más operandos sobre los que se actúa. En el apéndice A se encuentra la lista completa de instrucciones, la cual incluye ejemplos y explicaciones. Como se puede observar, en total son 35, las cuales tardan un ciclo de máquina a excepción de los saltos, que toman dos ciclos.

El PIC16C710/711/715

Los PIC16C710/711/715 son totalmente compatibles con el PIC16C71, por ser de reciente introducción al mercado no son tan populares. La diferencia radica en lo siguiente:

PIC16C710. Posee circuito de Brown-Out, esto quiere decir que el pin de *reset* (MCLR) se puede conectar directamente a la fuente de alimentación, ya que el sistema interno detecta cuando el voltaje está por debajo de 4 voltios y hace que el microcontrolador se reinicie. La memoria de programa de este microcontrolador sólo tiene 512 bytes (0,5 Kbytes).

PIC16C711. Posee además del Brown-Out, 32 posiciones de memoria RAM adicionales. Por lo tanto, el mapa de memoria llega hasta la dirección 4Fh. La memoria de programa es de 1kbyte (igual al PIC16C71).

PIC16C715. Posee circuito de Brown-Out, 84 posiciones adicionales de memoria RAM y memoria de programa de 2 kbytes.



Proyectos con el PIC16C71

- *Termómetro digital con módulo LCD*
- *Adquisición de datos vía RS-232*

Proyecto N° 1: Termómetro digital con módulo LCD

Como ejemplo de la utilización del convertidor A/D del PIC16C71, vamos a implementar un termómetro que muestra la temperatura medida, en grados centígrados, en la pantalla de un display o módulo LCD.

Para medir la temperatura emplearemos un LM35. Este sensor tiene tres pines: Alimentación, tierra y la salida analógica. Este dispositivo presenta en su salida una variación de 10 mV/°C, por lo tanto, el valor de la temperatura se puede obtener directamente, sin necesidad de hacer modificaciones al dato obtenido.

El LM35 puede trabajar en un rango de temperatura entre -55 y 150°C, la fuente de alimentación positiva puede estar entre 4 y 30 Voltios. Además, su precisión es de 0.5°C. Este elemento viene en un encapsulado plástico TO-92 y tiene la misma apariencia de un transistor.

El módulo de cristal líquido se conecta al microcontrolador utilizando interface de 4 bits. En este ejercicio, conectamos la entrada analógica al pin RA2/AN2 del microcontrolador y el voltaje de referencia del convertidor analógico a digital, se configura para que sea el voltaje de alimentación del PIC (VDD). En la figura 4.1 se muestra el diagrama esquemático del termómetro digital.

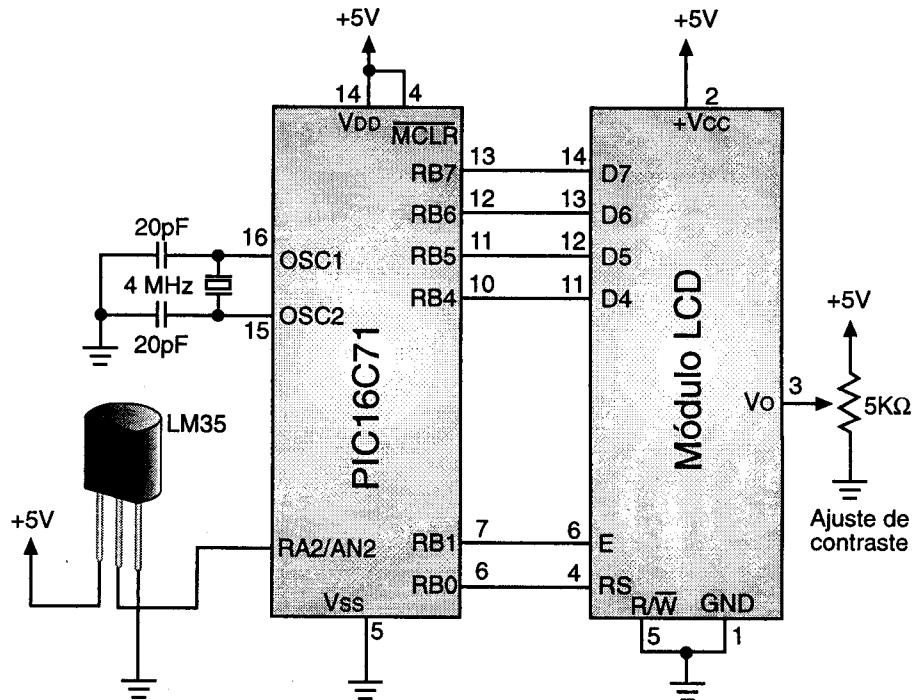


Figura 4.1. Diagrama esquemático del termómetro digital con PIC

La máxima lectura del convertidor analógico a digital (0FFH) se obtiene cuando la entrada analógica sea superior a 5 voltios. De tal manera, que si hacemos la división de 5 Voltios entre 255 (FFH), obtenemos una relación de aproximadamente 20 mV por unidad. Por ejemplo, si la entrada analógica está en 160 mV (es decir 16°C), el valor

digital que entrega el convertidor es 8 (160/20). Dado lo anterior, el valor que entrega el convertidor se debe duplicar para compensar la diferencia (variación de 10mV/°C y resolución de lectura de 20mV). Aunque esto hace que la lectura siempre sea un número par, se puede aceptar en vista de que se trata de ejercicios netamente didácticos.

Para solucionar el problema de tener que duplicar el valor leído y garantizar que el resultado de la conversión corresponda a la temperatura real, se puede utilizar un amplificador operacional para amplificar la señal entregada por el sensor LM35, dicho operacional se debe configurar como amplificador no inversor de ganancia 2. De esta forma, con el valor análogo adecuado en la entrada del convertidor y con la fuente de alimentación como voltaje de referencia, se puede obtener una lectura precisa y directa. En la figura 4.2 se muestra el diagrama del circuito que utiliza el amplificador operacional, si se utiliza este montaje se puede obviar la parte del programa que duplica el valor leído (la parte sombreada en el listado).

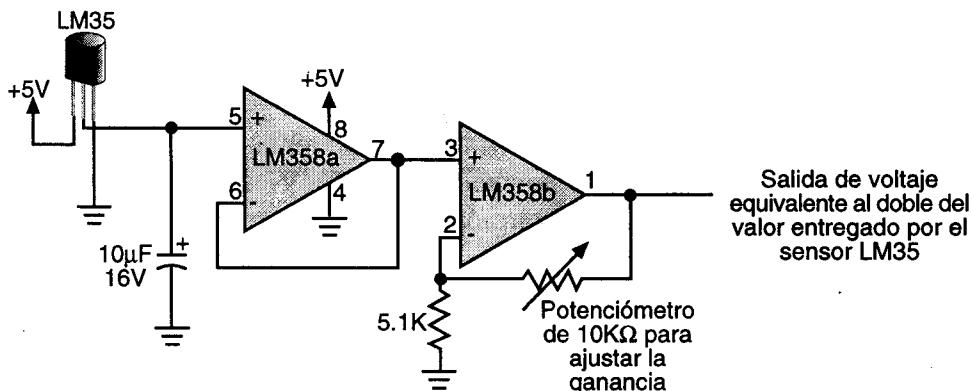


Figura 4.2. Circuito para acople del sensor LM35

En la figura 4.3 se muestra el listado del programa, este incluye todo el programa de la interfaz a 4 bits con el display de cristal líquido, una sección que se encarga de leer el convertidor análogo a digital, y una sección que convierte el dato hexadecimal obtenido luego de la conversión, a su equivalente en decimal, para ser mostrado en la pantalla.

Pasos a seguir para hacer una conversión A/D

1. Configure los pines del puerto A, los que vayan a trabajar como entradas análogas deben ser configurados como entradas en el registro TRISA.
2. Configure las entradas análogas y el voltaje de referencia (interno o externo) en el registro ADCON1.
3. Seleccione el canal de entrada en el registro ADCON0.
4. Seleccione el reloj de la conversión en ADCON0.
5. Encienda el convertidor (bit ADON del registro ADCON0) para que se tome la muestra de la señal análoga.
6. Si se va a utilizar la interrupción se debe borrar el bit ADIF del registro ADCON0 y poner en "1" el bit ADIE y el bit GIE del registro INTCON.
7. Luego de iniciar el convertidor se hace un pequeño retardo (para el muestreo).
8. Se debe iniciar la conversión mediante el bit GO/DONE del registro ADCON0.

9. Esperar a que se termine la conversión probando el estado del bit GO/DONE o esperar el cambio en la bandera de interrupción, es decir probando el bit ADIF del registro ADCON0.
10. El resultado se debe leer en el registro ADRES.
11. Borrar el bit ADIF.

Para el caso del termómetro digital los pasos seguidos fueron los siguientes:

- En el primer bloque del programa principal (INICIO), además de configurar los puertos, se programa el pin RA2/AN2 como una entrada analógica. El bloque que inicia con la etiqueta BEGIN se encarga de ubicar en la pantalla el mensaje del medidor de temperatura, utilizando la interface a 4 bits como se vió en el capítulo 2.
- La parte del programa que inicia con la etiqueta MIDE se encarga de configurar la velocidad del convertidor analógico a digital, de poner el bit ADIF en "0" y de poner el bit ADON del registro ADCON0 en "1" antes de empezar la conversión, luego se hace un retardo. El trabajo del convertidor se inicia cuando el bit ADCON0.GO se pone en «1», nuevamente se hace un pequeño retardo. Para saber si el convertidor ha terminado se prueba el estado del bit llamado ADCON0.ADIF.
- Una vez que se está seguro de tener el dato digital que resulta de la conversión, este se puede leer del registro ADRES y se debe duplicar antes de enviarlo a la pantalla. Como este número se encuentra en hexadecimal, se debe utilizar una rutina que se encargue de convertirlo a su equivalente en decimal. Esto se hace con la rutina llamada DECIMAL, la cual devuelve el valor obtenido en los registros CENTENA, DECENA y UNIDAD. A estos datos se les suma el valor 30h para convertirlos en su equivalente ASCII antes de enviarlos a la pantalla.

Esta práctica se puede realizar con cualquier tipo de sensor cuya salida esté entre 0 y 5 voltios, de lo contrario se deberá utilizar algún medio para acondicionar su señal a los rangos permitidos por el microcontrolador, por ejemplo un amplificador operacional. En el momento de grabar el microcontrolador se debe tener cuidado de seleccionar el tipo correcto, ya que se ha venido trabajando con el 16F84 y esto nos puede hacer equivocar.

```

; este programa toma el valor entregado por el sensor de
; temperatura y lo muestra en la pantalla del modulo lcd
; ***** pic16c71 *****
; ***** wdt = off *****
; ***** osc = xt *****
; ***** definiciones *****
;
indf    equ      0h          ;para direccionamiento indirecto
tmro    equ      1h          ;contador de tiempo real
pc      equ      2h          ;contador de programa
status  equ      3h          ;registro de estados y bits de control
fsr     equ      4h          ;selección de bancos de memoria
ptoa    equ      5h          ;puertos
ptob    equ      6h
adcon0  equ      8h          ;registros del convertidor a/d
adcon1  equ      88h
adres   equ      9h
trisa   equ      85h          ;programación de los registros

```

```

trisb    equ     86h
r0c      equ     0ch
r0d      equ     0dh
r0e      equ     0eh
unidad   equ     10h
decena   equ     11h
centena  equ     12h
r14      equ     14h
r1b      equ     1bh
rp0      equ     5h      ;seleccion de pagina
z        equ     2h      ;bandera de cero
c        equ     0h      ;bandera de carry
```       equ     0h      ;para almacenar en w
r equ 1h ;para almacenar en el mismo registro
pcfg0 equ 0
pcfg1 equ 1
adif equ 1h ;bits del registro adcon1
go equ 2h
e equ 1h
rs equ 0h

;
***** programa principal *****
;

org 00 ;vector de reset
goto inicio ;va a iniciar programa principal
org 05h
retardo movlw 0ffh
decre decfsz r1b,r
goto decre
retlw 0
retar2 movlw 0ffh
decr2 movwf r14
call retardo
call retardo
decfsz r14,r
goto decre2
retlw 0

decimal clrf decena
clrf centena
movlw d'100'
otra subwf unidad,r
btfs status,c
goto sum
incf centena
goto otra
sum addwf unidad
movlw d'10'
repite subwf unidad
btfs status,c
goto sum1
incf decena
goto repite
sum1 addwf unidad
retlw 0

;rutina que convierte binario en bcd
;borrar registros de trabajo

sum addwf unidad
movlw d'10'
repite subwf unidad
btfs status,c
goto sum1
incf decena
goto repite
sum1 addwf unidad
retlw 0

;restar 100 al valor inicial
;verifica el carry
;si es cero deja de restar 100
;si es 1 incrementa centena
;volver a restar
;sumarle 100

;restar 10 al valor
;verifica el carry
;si es 0 deja de restar
;si es 1 incrementa decena

;sumarle 10 al valor
;el valor de la conversion binario a
;decimal
;se devuelve en los registros centena,
;decena y unidad

control bcf ptob,rs
dato goto dato2
dato bsf ptob,rs

dato2 bsf ptob,e
movwf r0e
movlw 0fh
andwf ptob,r
movf r0e,w
andlw 0f0h
iorwf ptob,r

```

	call	retardo	
	bcf	ptob,e	
	call	retardo	
	bsf	ptob,e	
	movlw	0fh	
	andwf	ptob,r	
	swapf	r0e,w	
	andlw	0f0h	
	iorwf	ptob,r	
	call	retardo	
	bcf	ptob,e	
	call	retardo	
	retlw	0	
tabla	addwf	pc,r	;mensaje que se muestra
	retlw	" "	
	retlw	"L"	
	retlw	"A"	
	retlw	" "	
	retlw	"T"	
	retlw	"E"	
	retlw	"M"	
	retlw	"P"	
	retlw	"E"	
	retlw	"R"	
	retlw	"A"	
	retlw	"T"	
	retlw	"U"	
	retlw	"R"	
	retlw	"A"	
	retlw	" "	
	retlw	" "	
	retlw	"A"	
	retlw	"C"	
	retlw	"T"	
	retlw	"U"	
	retlw	"A"	
	retlw	"L"	
	retlw	" "	
	retlw	"E"	
	retlw	"S"	
	retlw	" "	
	retlw	"C"	
	retlw	" "	
	retlw	0	
inicio	bsf	status, rp0	;seleccionar pagina 1
	movlw	0ffh	;configura ptoa como entradas
	movwf	trisa	;
	bcf	adcon1,pcfg1	;configura ra3 como voltaje de referencia
	bcf	adcon1,pcfg0	;y ra0-ra2 como entradas analogas
	movlw	0ch	;configura ptob
	movwf	trisb	;
	bcf	status, rp0	;vuelve a pagina 0
begin	movlw	02h	;inicia display a 4 bits
	call	control	
	movlw	28h	
	call	control	
	movlw	0ch	
	call	control	
	movlw	06h	
	call	control	
blank	movlw	01h	
	call	control	
muestra	movlw	0	
	movwf	r0c	
ciclo	movf	r0c,w	
			;inicia contador de caracteres
			;hace barrido de la tabla1

*Capítulo 4. Proyectos con el PIC16C71*

```

call tabla
call dato
movlw 05fh ;retardo entre caracteres
retal r0d
call retardo
decfsz r0d,r
goto reta1
incf r0c,r ;sigue con la tabla
movlw 10h
subwf r0c,w ;pregunta si esta mostrando el mensaje dela
btffss status,c ;segunda linea
goto ciclo
movlw 10h ;pregunta si es la primera vez que entra
xorwf r0c,w ;a la segunda linea para ir a iniciar
btffss status,z ;el puntero de la ram del modulo lcd
goto line2
movlw 0c0h ;ubica puntero de la ram del modulo lcd
call control
line2 20h ;en la segunda linea
movlw r0c,w ;pregunta si termino la segunda linea
xorwf status,z ;para ir a iniciar de nuevo el mensaje o
btffss ciclo ;para continuar en la segunda linea
line2
mide
movlw b'01010000' ;se hace la conversionanálogo a digital
movwf adcon0 ;configura el conver., selecciona canal,
 ;velocidad de conver. y lo activa
 ;adif=0 antes de empezar la conversion
 ;adcon0,0
bsf adcon0,0
movlw 0fh
movwf r1b
decfsz r1b,r
goto dec
bsf adcon0,go ;empezar la conversion
movlw 0fh
movwf r1b
decfsz r1b,r
goto de2
de2
consu
nop
btffss adcon0,adif ;pregunta fin de conversion
goto consu ;si no ha terminado sigue esperando
bcf adcon0,0
bcf adcon0,adif ;apaga el convertidor
movf adres,w ;si termino de convertir debe pasar
movwf unidad
addwf unidad,r ;duplicar el valor leido
call decimal
movlw 0cbh ;convertir el dato binario a decimal
call control
movf centena,w ;ubicopunterode ram donde va el numero
addlw 30h
call dato
movlw 0cch ;ubicopunterode ram donde va el numero
call control
movf decena,w ;segundo digito = decenas
addlw 30h
call dato
movlw 0cdh ;ubicopunterode ram donde va el numero
call control
movf unidad,w ;segundo digito = unidades
addlw 30h
call dato
call retar2
clrff adres
goto mide ;termina
end

```

**Figura 4.3.** Programa completo del termómetro digital

## Proyecto N° 2: Adquisición de datos vía RS-232

Esta práctica consiste en leer una señal analógica con un microcontrolador PIC16C71 y enviar a una computadora el dato digital equivalente; esta última se encargará de mostrar en la pantalla (de forma gráfica) la manera en que la señal varía con el tiempo.

Como fuente de señal se puede usar un potenciómetro de 5 kohm, el cual se conecta directamente a un pin del microcontrolador que se ha configurado como entrada analógica. La forma de enviar los datos serialmente se vió en el capítulo 2, utilizando como interface un integrado MAX232. También se muestra el dato leído en una pantalla de cristal líquido, con el fin de comprobar que todo funciona correctamente, es decir que se debe mostrar el mismo dato que en la computadora. Por su parte, la computadora tiene un programa en lenguaje C que se encarga de leer el dato serial y de mostrarlo en la pantalla. En la figura 4.4 se muestra el diagrama del circuito.

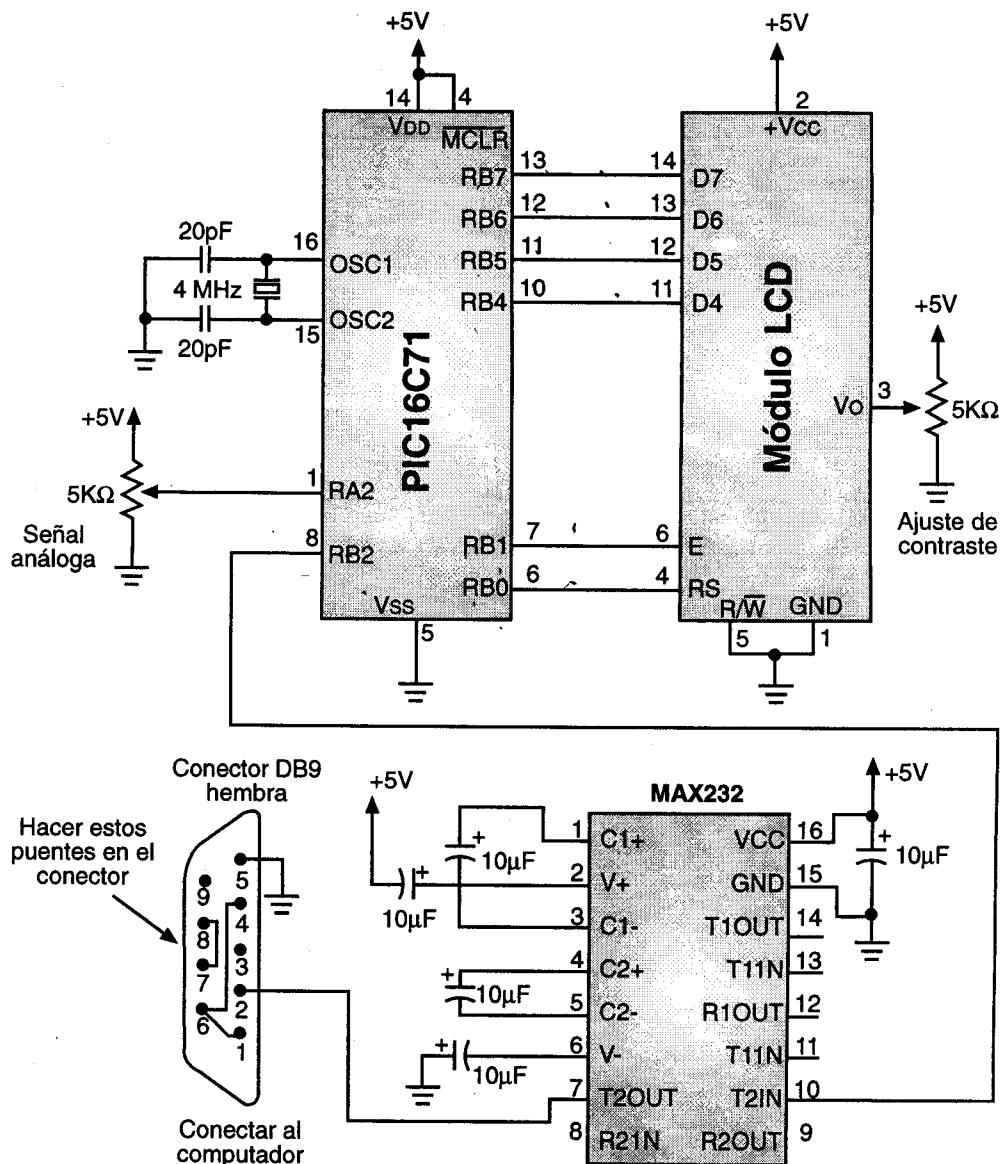


Figura 4.4 Diagrama esquemático del circuito de adquisición de datos

La comunicación se hará con las mismas especificaciones del ejercicio anterior, es decir a 1200 bps, 8 bits, sin paridad y con un *stop bit* (1200, 8, N, 1). En la figura 4.5 se muestra el programa que se graba en el microcontrolador, los bloques más importantes son: el que hace la conversión análogo a digital, que inicia con la etiqueta MIDE y el que hace la transmisión serial que lo compone la subrutina llamada ENVIAR. Estas rutinas se han explicado en ejercicios anteriores.

La computadora se debe encargar de leer el dato serial y de mostrarlo en la pantalla, para ello se debe utilizar el llamado *modo gráfico*. En este modo se traza la ventana donde aparece la curva obtenida, al igual que se van ubicando punto a punto todos los datos que entrega el microcontrolador. En la figura 4.6 se muestra el programa en lenguaje C que realiza la tarea descrita, llamado ADQUISI1.C. Nuevamente se tienen dos versiones, una para utilizar el COM1 llamada ADQUISI1 y otra para usar el COM2 llamado ADQUISI2. El programa ejecutable que puede correr el usuario desde el DOS tiene los mismos nombres pero con extensión .EXE. En la figura 4.7 se tiene un pantallazo del programa ejecutable, mostrando una gráfica obtenida.

```
; este programa toma el valor análogo leido y lo envía
; a la computadora de forma serial
; ***** pic16c84 *****
; ***** wdt = off *****
; ***** osc = xt *****
; ***** cp = on *****

; ***** definiciones *****
indf equ 0h ;para direccionamiento indirecto
tmro equ 1h ;contador de tiempo real
pc equ 2h ;contador de programa
status equ 3h ;registro de estados y bits de control
fsr equ 4h ;selección de bancos de memoria
ptoap equ 5h ;puertos
ptob equ 6h
adcon0 equ 8h ;registros del convertidor a/d
adcon1 equ 88h
adres equ 9h
trisa equ 85h ;programación de los registros
trisb equ 86h
r0c equ 0ch
r0d equ 0dh
r0e equ 0eh
unidad equ 10h
decena equ 11h
centena equ 12h
r14 equ 14h
r1b equ 1bh
r1c equ 1ch
r1d equ 1dh
trans equ 1eh ;bits especiales
 ;bits del registro status
 ;seleccion de pagina
 ;bandera de cero
 ;bandera de carry

rp0 equ 5h
z equ 2h
c equ 0h

w equ 0h ;para almacenar en w
r equ 1h ;para almacenar en el mismo registro
pcf0 equ 0 ;bits del registro adcon1
pcf1 equ 1
adif equ 1h ;bits del registro adcon0
go equ 2h
tx equ 2 ; pines del puerto a
```



control	bcf	ptob,rs	;esta rutina genera las señales de control
dato	goto	dato2	;para escribir en el modulo lcd y
pantalla	bsf	ptob,rs	;entrega el dato a ser mostrado en la
dato2	bsf	ptob,e	;utiliza la interface a 4 bits
	movwf	r0e	
	movlw	0fh	
	andwf	ptob,r	
	movf	r0e,w	
	andlw	0f0h	
	iorwf	ptob,r	
	call	retardo	
	bcf	ptob,e	
	call	retardo	
	bsf	ptob,e	
	movlw	0fh	
	andwf	ptob,r	
	swapf	r0e,w	
	andlw	0f0h	
	iorwf	ptob,r	
	call	retardo	
	bcf	ptob,e	
	call	retardo	
	retlw	0	
tabla	addwf	pc,r	;mensaje que se muestra
	retlw	"E"	
	retlw	"L"	
	retlw	" "	
	retlw	"D"	
	retlw	"A"	
	retlw	"T"	
	retlw	"O"	
	retlw	" "	
	retlw	"A"	
	retlw	"N"	
	retlw	"A"	
	retlw	"L"	
	retlw	"O"	
	retlw	"G"	
	retlw	"O"	
	retlw	" "	;mensaje de la segunda linea
	retlw	" "	
	retlw	"A"	
	retlw	"C"	
	retlw	"T"	
	retlw	"U"	
	retlw	"A"	
	retlw	"L"	
	retlw	" "	
	retlw	"E"	
	retlw	"S"	
	retlw	" "	
	retlw	0	
inicio	bsf	status,rp0	;seleccionar pagina 1
	movlw	0ffh	;configura pto4 como entradas
	movwf	trisa	;
	bcf	adcon1,pcf1	;configura ra3 como voltaje de referencia
	bcf	adcon1,pcf0	;y ra0-ra2 como entradas analogas

	movlw	08h	;configura ptob
	movwf		;
	bcf	status, rp0	;vuelve a pagina 0
begin	movlw	02h	;inicia display a 4 bits
	call	control	
	movlw	28h	;display a 4 bits y 2 lineas
	call	control	
	movlw	0ch	;activa display y desactiva cursor
	call	control	
	movlw	06h	;selecciona el modo de desplazamiento
	call	control	
blank	movlw	01h	;borra display y cursor a casa
muestra	movlw	0	;inicia contador de caracteres
ciclo	movwf	r0c	
	movf	r0c,w	;hace barrido de la tabla1
	call	tabla	
	call	dato	
	movlw	05fh	
	movwf	r0d	
reta1	call	retardo	
	decfsz	r0d,r	
	goto	reta1	
	incf	r0c,r	
	movlw	10h	
	subwf	r0c,w	
	btfss	status,c	
	goto	ciclo	
	movlw	10h	
	xorwf	r0c,w	
	btfss	status,z	
	goto	line2	
linea2	movlw	0c0h	
	call	control	
line2	movlw	20h	
	xorwf	r0c,w	
	btfss	status,z	
	goto	ciclo	
mide	movlw	b'01010000'	
	movwf	adcon0	
	bsf	adcon0,0	
	movlw	0fh	
	movwf	r1b	
dec	decfsz	r1b,r	
	goto	dec	
	bsf	adcon0,go	
	movlw	0fh	
	movwf	r1b	
de2	decfsz	r1b,r	
	goto	de2	
consu	nop		
	btfss	adcon0,adif	
	goto	consu	
	bcf	adcon0,0	
	bcf	adcon0,adif	
	movf	adres,w	
	movwf	unidad	
	call	enviar	
	call	decimal	
	movlw	0cbh	
	call	control	

```

movf centena,w ;primer digito = centenas
addlw 30h
call dato
movlw 0cch ;ubico puntero de ram donde va el numero
 ;leido

call control
movf decena,w ;segundo digito = decenas
addlw 30h
call dato
movlw 0cdh ;ubico puntero de ram donde va el numero
 ;leido

call control
movf unidad,w ;segundo digito = unidades
addlw 30h
call dato
call retar2
clrf adres
goto mide ;termina
end

```

Figura 4.5. Programa que convierte el dato análogo a digital y lo envía a la computadora

```

#include <bios.h>
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

int puerto,COM1,COM2;
int k,j,dato;
int config;
int COM1,COM2;
int maxx,maxy;
float x;
char pattern[8] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

void leer (void);
void leer(void)
{
do{
dato=bioscom(2,0x83,puerto); /*leer dato recibido*/
} while (((dato<0)|(dato>255))&(!kbhit()));
}
void main(void)
{
/* autodetección */
int gdriver = DETECT, gmode, errorcode; /* inicializa modo gráfico */

initgraph(&gdriver, &gmode, "c:\\borlandc\\bgi\\");
errorcode = graphresult(); /* lee resultado de inicialización */
if (errorcode != grOk) /* un error ocurrió */
{
printf("Graphics error: %s\n", grapherrmsg(errorcode));
printf("Press any key to halt:");
getch();
exit(1);
}
maxx=getmaxx();
maxy=getmaxy();

COM1=0;
COM2=1;
puerto=COM1
/* definir cual puerto se utiliza */

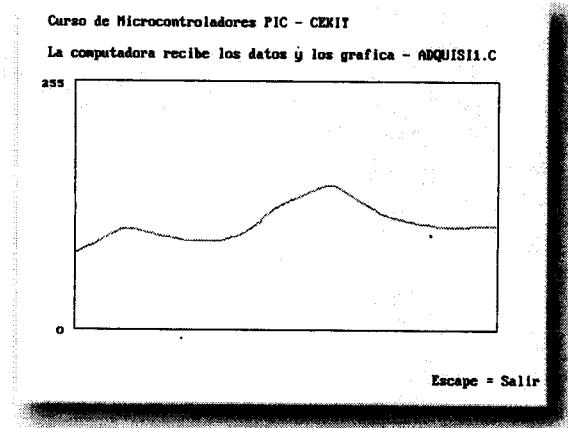
```

```

clrscr(); /*limpiar pantalla*/
config=0x83; /*configurar puerto: 1200 bps,8 bits, sin paridad,1 stop bit*/
bioscom(0,config,puerto); /*configuración de los puertos*/
cleardevice();
gotoxy(10,2);
printf("Curso de Microcontroladores PIC - CEKIT");
gotoxy(10,4);
printf("La computadora recibe los datos y los grafica - ADQUISI1.C");
gotoxy(60,25);
printf("Escape = Salir");
x=0;
moveto(66,maxy/2-155);
outtext("255");
moveto(82,maxy/2+100);
outtext("0");
setfillpattern(pattern,0);
bar(100,maxy/2-155,maxx-100,maxy/2+100);
rectangle(100,maxy/2-156,maxx-100,maxy/2+101);
moveto(100,maxy/2+100);
do{
leer();
if(!kbhit())
{
lineto(x+100,maxy/2+100-dato);
gotoxy(10,24);
printf(" ");
gotoxy(10,24);
printf("El dato leido es: %d",dato);
moveto(x+100,maxy/2+100-dato);
x+=1;
if (x>maxx-200)
{
setfillpattern(pattern,0);
bar(100,maxy/2-155,maxx-100,maxy/2+100);
rectangle(100,maxy/2-155,maxx-100,maxy/2+100);
x=0;
moveto(100,maxy/2+100);
}
}
}while(!kbhit());
closegraph();
clrscr();
}

```

**Figura 4.6.** Programa en lenguaje C que lee el dato serial y lo grafica en la pantalla (adquisi1.C)



**Figura 4.7.** Pantallazo del programa ADQUISI1



## *Capítulo 5*

# **El PIC12C50X**

- **Arquitectura**
- **Características especiales**
- **Conjunto de instrucciones**

La vida moderna exige que los objetos de uso cotidiano sean de tamaño reducido y de mayor funcionalidad; en especial, lo que a sistemas electrónicos se refiere. Esta familia de microcontroladores es ideal para uso en sitios de espacio reducido, inclusive en aplicaciones donde nunca antes se habían considerado. El PIC12C508 y el PIC12C509 son microcontroladores de 8 bits basados en la familia PIC16C5X; son ideales para utilizarlos como codificadores/decodificadores en sistemas de control remoto, sistemas de seguridad, receptor de frecuencia, timer, funciones de coprocesador, etc. Su precio y su rendimiento lo ubican como uno de los microcontroladores con mejor relación costo/beneficio.

Una de las mayores ventajas de esta familia es la eliminación de componentes externos ya que posee internamente su circuito de Reset al encender el sistema (Power-On-Reset), circuito de vigilancia (Watchdog Timer) incorporado, cuatro posibles configuraciones de circuito de reloj, incluyendo un modo de oscilador interno (INTRC).

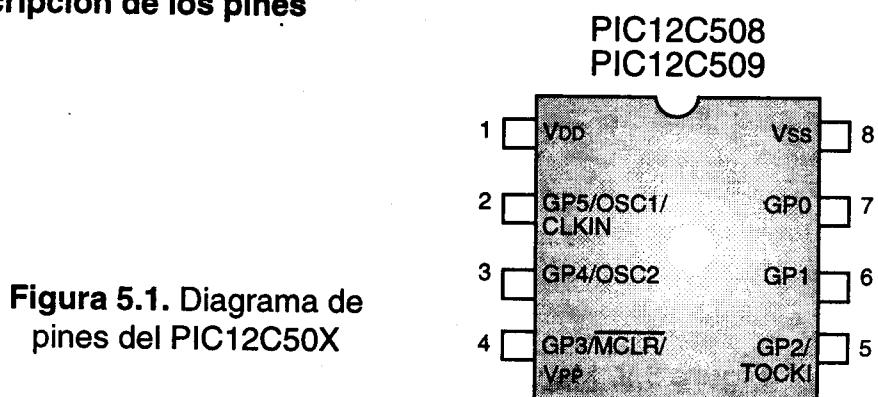
#### Principales características del PIC12C50X:

- 8 pines en total, de los cuales 5 se pueden usar como entrada/salida (I/O)
- Únicamente 33 instrucciones.
- Velocidad de operación de 4 MHz.
- El PIC12C508 tiene una memoria de programa de 512 posiciones. El PIC12C509 tiene memoria de programa de 1024 posiciones.
- El PIC12C508 tiene una memoria RAM de 25 posiciones. El PIC12C509 tiene memoria RAM de 41 posiciones.
- Pila de 2 niveles.
- Oscilador interno de 4 MHz con calibración programable.
- Temporizador/contador de 8 bits.
- Pull-up interno en los pines de entrada/salida y en el pin de reset.
- Bajo consumo de potencia.

### Arquitectura

Los PIC12C50X están basados en la arquitectura Harvard, la cual se basa en tener buses independientes para memoria de programa y para memoria de datos, lo que permite ejecutar una instrucción al mismo tiempo que se prepara la siguiente. La memoria de datos tiene una palabra de 8 bits. El bus y la memoria de programa tienen un ancho de 12 bits, lo que hace posible tener instrucciones poderosas que se ejecutan en un solo ciclo.

### Descripción de los pines



**Figura 5.1.** Diagrama de pines del PIC12C50X

**GP0 - GP1:** Pin bidireccional de entrada/salida, puede ser programado por software para conectar el pull-up interno.

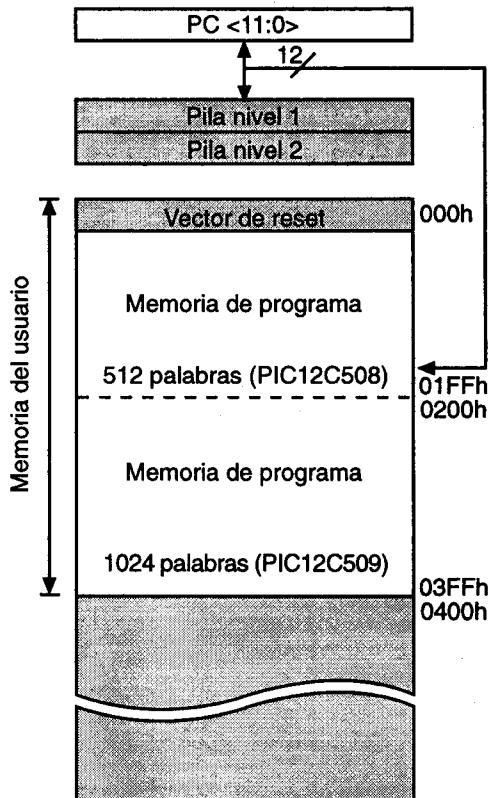
**GP2/TOCKI:** Pin bidireccional de entrada/salida; puede ser programado por software para utilizarlo como TOCKI en el conteo de eventos externos.

**GP3/MCLR /Vpp :** Pin de entrada solamente. Puede ser configurado por software como MCLR y es activado por un nivel bajo. Cuando se configura en este modo, siempre se activa su pull-up interno. En el momento de programar el micro, este pin se conecta al voltaje de programación (Vpp).

**GP4/OSC2:** Pin bidireccional de entrada/salida. En caso de tener oscilador con cristal o resonador solo se puede utilizar para tal fin. Si se utiliza el oscilador interno, se puede usar como puerto (I/O).

**GP5/OSC1 /CLKIN:** Pin bidireccional de entrada/salida. Cuando se utiliza un oscilador a cristal o resonador solo se puede utilizar para tal fin. Si se utiliza el oscilador interno, se puede usar como puerto (I/O).

## Organización de la memoria de programa



Nota 1: El vector de reset está en la dirección 0000H. La posición 01FFh (PIC12C508) ó 03FFh (PIC12C509) contiene la instrucción MOVLW XX para la calibración del reloj interno.

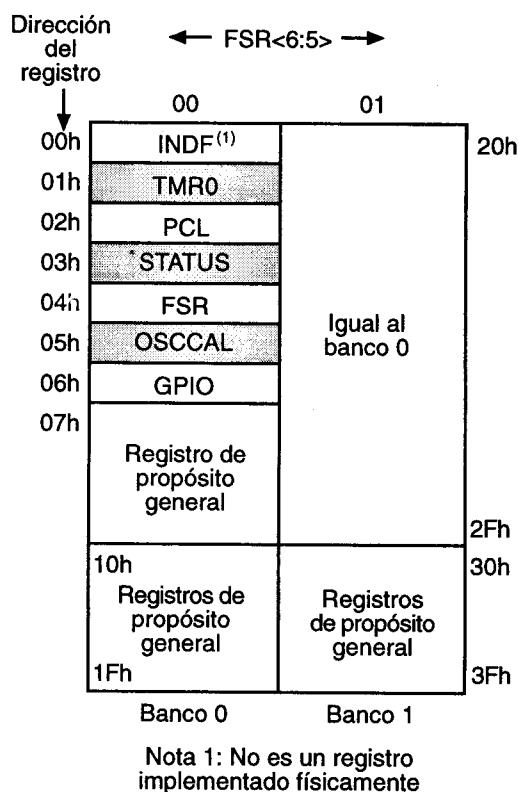
Figura 5.2. Mapa de la memoria de programa

El PIC12C50X tiene un contador de programa de 12 bits capaz de direccionar 4k de memoria. Unicamente los primeros 512 (0000h - 01ffh) para el PIC12C508 y 1k (0000h - 03ffh) para el PIC12C509 son implementados físicamente, figura 5.2. En el caso del PIC12C509 se tiene un esquema de paginación de memoria; para accesarlo se debe usar un bit del registro de estado.

El vector de reset está en la dirección 0000h. La posición 01ffh en el PIC12C508 y la 03ffh en el PIC12C509 contiene el valor de calibración para el oscilador interno; este valor nunca podrá ser sobreescrito.

### Organización de la memoria de datos

La memoria de datos está compuesta de registros o bytes de RAM. Estos se dividen en dos grupos: registros de función especial y registros de propósito general, figura 5.3.



**Figura 5.3. Mapa de la memoria de datos**

Los registros de función especial están compuestos por el registro contador y reloj de tiempo real TMRO, el contador de programa (PC), el registro de estado, el puerto GP y el selector de registros FSR. Los registros de propósito general se utilizan para datos e información de control.

En el PIC12C508 la memoria de datos está compuesta por 7 registros de función especial y 25 registros de propósito general. Para el PIC12C509 se tiene la misma configuración anterior más un grupo de 16 registros de propósito general que se encuentran en el segundo banco de memoria y se pueden direccionar usando el FSR.

## Registros de función especial

**INDF:** *Registro de direccionamiento indirecto.* Este no es un registro que esté implementado físicamente; utiliza el contenido del FSR para direccionar la memoria de datos. Lo que se haga con el dato del registro escogido depende de la instrucción que se utilice.

**TMRO:** *Contador y reloj de tiempo real de 8 bits.* Este registro se puede configurar por software para que trabaje como temporizador, cuya base de tiempo es el oscilador del micro. El contenido inicial del conteo se puede establecer ya que es un registro que se puede leer y escribir. También se puede configurar como un contador de eventos externos que ocurren en el pin TOCKI. El incremento del conteo se puede afectar con una preescala para lograr un período de tiempo más largo; dicha preescala se ajusta en el registro OPTION. Vale la pena aclarar que la preescala se puede utilizar tanto por el TMRO como por el Watchdog Timer, pero solo uno de los dos a la vez.

**PC:** *Contador de programa.* Es utilizado para señalar cual es la instrucción que se va a ejecutar. El PC se incrementa en un paso, a menos que la instrucción lo modifique. Cuando se ejecuta un llamado de subrutina (CALL), el bit 8 del PC se borra (puesto a cero), lo que hace que todas las subrutinas deban ser escritas en las primeras 256 posiciones de cada página. Cuando se ejecuta una instrucción de salto (GOTO), el bit 9 del PC se carga con el contenido del bit PA0 del registro de estado. Esto hace que si el destino está en otra página de memoria, se deba ajustar primero el bit PA0 del registro de estado con el valor correspondiente.

	GPWUF	—	PAO	TO	PD	Z	DC	C
bit 7:	<b>GPWUF:</b> Bit de reset 1 = Reset al despertar de SLEEP por cambio en un pin 0 = Despues de encender el sistema o por otro reset							
bit 6:	<b>No implementado</b>							
bit 5:	<b>PAO:</b> Bit de selección de página 1 = Página 1 (200h - 3FFh) - PIC12C509 0 = Página 0 (000h - 1FFh) - PIC12C508 Cada página tiene 512 bytes.							
bit 4:	<b>TO:</b> Bit de Time-out 1 = Luego de encender el sistema 0 = Ocurrió un Time - out por WDT							
bit 3:	<b>PD:</b> Power Down o Bit de bajo consumo. Se coloca en 0 por la instrucción SLEEP.							
bit 2:	<b>Z:</b> Cero bit 1 = El resultado de una operación aritmética es cero 0 = El resultado de una operación aritmética no es cero							
bit 1:	<b>DC:</b> Digit Carry o Bit de acarreo de dígito. En operaciones aritméticas se activa cuando hay acarreo entre el bit 3 y el 4.							
bit 0:	<b>C:</b> Carry o Bit de acarreo. En instrucciones aritméticas se activa cuando se presenta acarreo desde el bit más significativo del resultado.							

Figura 5.4. Registro de estados

**STATUS:** *Registro de estado.* Contiene el estado aritmético de la ALU, la causa del último Reset y el bit de selección de página de memoria, figura 5.4.

**FSR:** *Registro selector de registros.* Se utiliza con el registro INDF para apuntar a los otros registros disponibles en el modo de direccionamiento indirecto.

**OSCCAL:** *Registro de calibración del oscilador interno.* El PIC12C50X posee un oscilador RC interno de 4 MHz. Una instrucción de calibración de dicho oscilador se debe programar en el tope de la memoria. Este valor, OSCCAL, es programado con una instrucción "MOVLW XX", donde XX es el valor de la calibración, esta instrucción debe ser colocada en última disposición de memoria.

**GPIO:** *Puerto de entrada/salida.* Como cualquier otro registro, el GPIO puede ser leído y escrito bajo control del programa. Este es un registro de 8 bits, de los cuales solamente los 6 más bajos están implementados.

**TRIS:** *Registro de configuración del puerto.* Este registro es cargado con el contenido de el registro W al ejecutar la instrucción "TRIS f". Un "1" en un bit del registro TRIS configura el bit correspondiente de GPIO como entrada, un "0" lo configura como salida; a excepción del pin GP3 que solo se puede configurar como entrada. Este registro es solo de escritura y luego de una condición de Reset, se coloca todo en unos.

**OPTION:** *Registro de configuración múltiple.* Es un registro de 8 bits de sólo escritura, figura 5.5, el cual contiene varios bits de control para configurar el TMRO y la preescala del Timer/WDT. La ejecución de una instrucción "OPTION" mueve el contenido de W al registro OPTION. Cuando sucede un Reset todos los bits del registro son puestos a uno.

	GPWU	GPPU	TOCS	TOSE	PSA	PS2	PS1	PS0
	bit7	6	5	4	3	2	1	bit0
bit 7:	<b>GPWU:</b> Habilita despertar en un cambio de estado del pin (GP0, GP1, GP3) 1 = Deshabilitado 0 = Habilitado							
bit 6:	<b>GPPU:</b> Habilita pull-ups en los pines (GP0, GP1, GP3) 1 = Deshabilitado 0 = Habilitado							
bit 5:	<b>TOCS:</b> Fuente de la señal de TMRO 1 = Cambio en el pin TOCKI 0 = Transición en el reloj interno							
bit 4:	<b>TOSE:</b> Flanco de la señal de TMRO 1 = Flanco de bajada 0 = Flanco de subida							
bit 3:	<b>PSA:</b> Prescaler Assignment o Asignación del preescalador 0: TMRO (Contador/Temporizador) 1: WDT (Circuito de vigilancia)							
bit 2-0:	<b>PS2,1,0:</b> Prescaler Value o Valores del preescalador.							
	Valor	TMRO	WDT					
	000	1:2	1:1					
	001	1:4	1:2					
	010	1:8	1:4					
	011	1:16	1:8					
	100	1:32	1:16					
	101	1:64	1:32					
	110	1:128	1:64					
	111	1:256	1:128					

Figura 5.5. Registro OPTION

## Características especiales

### Palabra de configuración para el PIC12C50X.

Este registro no es direccionable durante la operación normal del microcontrolador, sólo se puede utilizar en el momento de programarlo, figura 5.6.

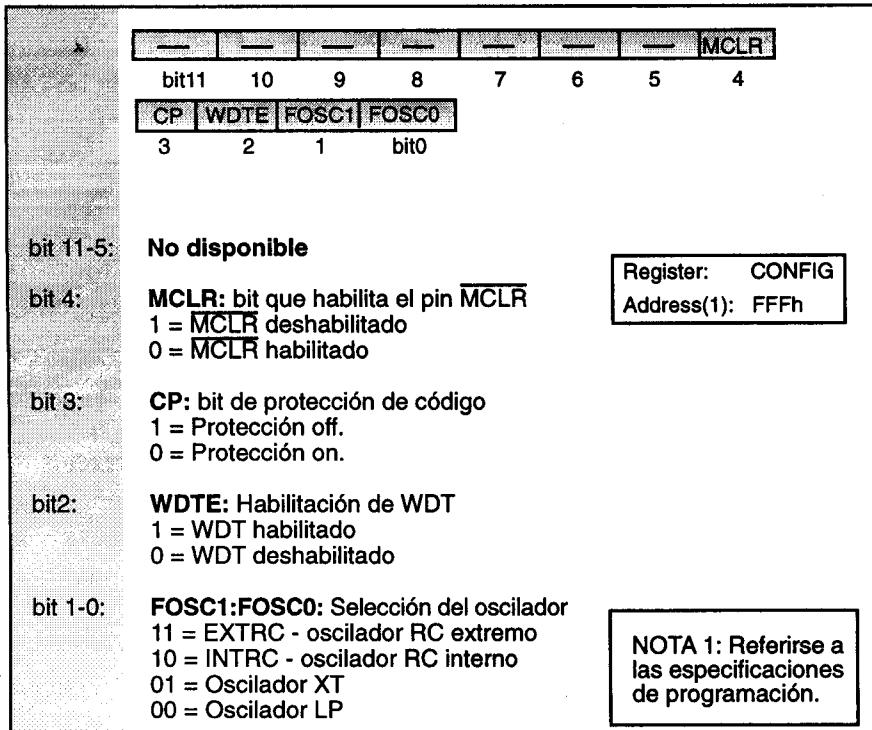


Figura 5.6. Palabra de configuración del PIC12C50X

### Stack o Pila

El PIC12C50X tiene una pila de dos niveles y 12 bits de ancho. Una instrucción "CALL" guarda el valor de stack 1 en el stack 2 y el valor actual del contador de programa incrementado en uno en el stack 1. Si se ejecutan más de dos "CALL", solamente los dos últimos son almacenados. Una instrucción "RETLW" retorna el valor de stack 1 al contador de programa y el valor de stack2 al stack 1. Una característica importante es que el registro W se carga con el literal especificado en la instrucción; esto es de gran utilidad cuando se trabaja con tablas usando la memoria de programa.

### Watchdog Timer

Este es un oscilador interno del micro que corre independientemente del reloj de 4 MHz. El WDT tiene un período nominal de 18 milisegundos, luego de los cuales el micro sufre un Reset. Para evitar esto, se debe utilizar la instrucción CLRWDAT antes de que se complete el tiempo de este oscilador interno.

El tiempo del WDT de 18 milisegundos se puede extender hasta 2 segundos haciendo uso de la preescala, como se vió cuando hablamos del registro OPTION. Esto es, el período del oscilador interno del WDT es dividido por el valor que se le dé a la preescala. Debe tenerse en cuenta que la preescala se puede utilizar con el WDT o con el temporizador TMRO, pero solo con uno a la vez.

El WDT se puede habilitar o deshabilitar en el momento de programar el microcontrolador utilizando el fusible correspondiente de la palabra de configuración.

## Reset

El PIC12C50X tiene varias fuentes de reset, vamos a describirlas en forma breve y clara:

- El Power-On-Reset (POR) es un dispositivo interno del micro que garantiza un tiempo de reset de 18 milisegundos en el momento de energizar el sistema. Esto es muy útil porque evita posibles conflictos que se podrían presentar antes de que se estabilicen las señales. Además, evita tener que utilizar componentes externos en el pin de reset.
- Un nivel bajo en el pin de Master Clear (MCLR) durante operación normal, este pin posee un pull-up interno que evita el uso de elementos externos.
- Un reset causado por el Watchdog Timer, el cual se presenta si no se ha borrado el contador interno antes de completar su período de tiempo.
- Cuando el micro está en modo de bajo consumo por la instrucción SLEEP, el cambio de nivel en un pin programado previamente hace que el micro "despierte" y sufra un reset.

## Configuraciones del oscilador

El PIC12C50X puede ser operado en cuatro modos diferentes de oscilador. El usuario puede escoger en el momento de programar el micro que tipo de oscilador va a utilizar. Esto se hace en la palabra de configuración utilizando los bits FOSC1 y FOSC0. Los modos de oscilador son :

**LP:** Cristal de baja potencia

**XT:** Cristal o resonador cerámico, máximo de 4 MHz.

**EXT RC:** Oscilador externo RC.

**INT RC:** Oscilador interno de 4 MHz. Este no requiere ningún componente externo y nos permite utilizar los dos pines que quedan disponibles como entrada/salida. Esta es una de las ventajas más importantes de esta familia.

Para utilizar el reloj interno se deben tener en cuenta varios aspectos:

1. El oscilador requiere un valor de calibración, dicho número se escribe en la última posición de la memoria de programa, en la forma de una instrucción: MOVLW XX, donde XX corresponde a dicho valor. Si se trata de un microcontrolador en versión OTP (*One Time Programmable*: programable una sola vez), dicha instrucción con el valor de calibración preestablecido en la fábrica ya está escrita, por lo tanto se puede obviar su escritura.

Si se trata de un microcontrolador en versión JW (ventana), dicho valor viene grabado de la misma forma en la última posición; por lo tanto el usuario debe leer el microcontrolador y tomar nota de dicho valor. Esto es necesario ya que en el momento de borrar el microcontrolador con luz ultravioleta para volver a cargar otro programa se borra también el valor de calibración. Por consiguiente, el usuario debe

incluir en su programa la instrucción que cargue el valor de calibración en la última posición de memoria.

Con un ejemplo se puede ilustrar mejor este caso: Al comprar un PIC12C508 de ventana, lo primero que se hace es leerlo para ver que dato tiene escrito en la posición de memoria 1FFh, el valor es por ejemplo 0C50. Este código equivale a la instrucción: MOVLW 50h. Por lo tanto el valor de calibración de ese microcontrolador en particular es 50h. Se debe aclarar que no todos los PIC12C508 tienen el mismo valor de calibración.

Después de ese ejercicio se debe tener en cuenta que todos los programas escritos para ese microcontrolador deben llevar las siguientes líneas:

ORG	1FFh	;escribe la siguiente instrucción en la dirección 1FFh
MOVLW	50h	;carga el valor de calibración en el registro W

2. Para cargar el valor de calibración en el registro OSCCAL se debe escribir la siguiente línea al comienzo de todos los programas que se escriban para el PIC12C508:

ORG	00h	;escribe la siguiente instrucción en la direcciónn 00h
MOVWF	OSCCAL	;carga el valor de W en el registro OSCCAL

Todos estos pasos se deben a la ruta que sigue el contador de programa o PC en el momento en que el microcontrolador sufre un reset, el cual, primero apunta a la última posición de memoria (en este caso 1FFh) y luego a la primera (00h).

## ***Conjunto de instrucciones***

El conjunto de instrucciones es el mismo que se tiene para la familia PIC16C5X, figura 5.7. Debe notarse que solamente se tienen 33 instrucciones, a diferencia del PIC16F84 y del PIC16C71 que tienen 35.

### **Conclusión.**

El PIC12C50X es una de las mejores opciones cuando se pretende diseñar un pequeño sistema microcontrolado ya que no requiere componentes externos y tiene un precio muy competitivo. Además, el software es compatible con las otras familias de microcontroladores de Microchip, lo que hace que el sistema se pueda extender sin problemas.

## Conjunto de instrucciones

$d = 0$  si el destino es W       $d = 1$  si el destino es el registro

Nombre	Nemónico	Operando	Operación	Estados afectados
<b>OPERACIONES ORIENTADAS A REGISTROS</b>				
Suma W y f	ADDWF	f, d	$W + f \rightarrow d$	C, DC, Z
AND entre W y f	ANDWF	f, d	$W \& f \rightarrow d$	Z
Limpia f	CLRF	f	$0 \rightarrow f$	Z
Limpia W	CLRW	-	$0 \rightarrow W$	Z
Complementa f	COMF	f, d	$f \rightarrow d$	Z
Decrementa f	DECFSZ	f, d	$f - 1 \rightarrow d$	Z
Dec f, salta si cero	DECFSZ	f, d	$f - 1 \rightarrow d$ , salta si cero	Ninguno
Incrementa f	INCF	f, d	$f + 1 \rightarrow d$	Z
Increm. f, salta si cero	INCFSZ	f, d	$f + 1 \rightarrow d$ , salta si cero	Ninguno
Or inclusiva entre W y f	IORWF	f, d	$W \vee f \rightarrow d$	Z
Mueve f	MOVF	f, d	$f \rightarrow d$	Z
Mueve W a f	MOVWF	f	$W \rightarrow f$	Ninguno
No operación	NOP	-	-	Ninguno
Rota a la izquierda f	RLF	f, d	$f(n) \rightarrow d(n+1), C \rightarrow d(0), f(7) \rightarrow C$	C
Rota a la derecha f	RRF	f, d	$f(n) \rightarrow d(n-1), C \rightarrow d(7), f(0) \rightarrow C$	C
Resta W de f	SUBWF	f, d	$f - W \rightarrow d$ [f+W+1→d]	C, DC, Z
Intercambia mitades de f	SWAPF	f, d	$f(0-3) \leftrightarrow f(4-7) \rightarrow d$	Ninguno
OR exclusiva entre W y f	XORWF	f, d	$W \oplus f \rightarrow d$	Z
<b>OPERACIONES ORIENTADAS A BITS</b>				
Limpia bit f	BCF	f, b	$0 \rightarrow f(b)$	Ninguno
Activa bit f	BSF	f, b	$1 \rightarrow f(b)$	Ninguno
Prueba bit f, salta si cero	BTFSZ	f, b	prueba bit(b) en registro(f): salta si cero	Ninguno
Prueba bit f, salta si uno	BTFSZ	f, b	prueba bit(b) en registro(f): salta si uno	Ninguno
<b>OPERACIONES LITERALES Y DE CONTROL</b>				
AND entre W y un literal	ANDLW	k	$k \& W \rightarrow W$	Z
Llama subrutina	CALL	k	$PC +1 \rightarrow Stack, k \rightarrow PC$	Ninguno
Limpia Watchdog	CLRWD	-	$0 \rightarrow WDT$ (y prescalador)	TO, PD
Salta a dirección	GOTO	k	$k \rightarrow PC$ (9 bits)	Ninguno
OR Incl. entre W y literal	IORLW	k	$k \vee W \rightarrow W$	Z
Mueve literal a W	MOVLW	k	$k \rightarrow W$	Ninguno
Carga registro OPTION	OPTION	-	$W \rightarrow Registro\ OPTION$	Ninguno
Retorna, carga literal en W	RETLW	k	$k \rightarrow W$ , pila $\rightarrow PC$	Ninguno
Va a modo de Espera	SLEEP	-	$0 \rightarrow WDT$ , detiene oscilador	TO, PD
Configura puerto	TRIS	f	$W \rightarrow control\ I/O\ del\ registro\ f$	Ninguno
OR excl. entre literal y W	XORLW	k	$k \oplus W \rightarrow W$	Z

**Figura 5.7. Instrucciones del PIC12C50X**

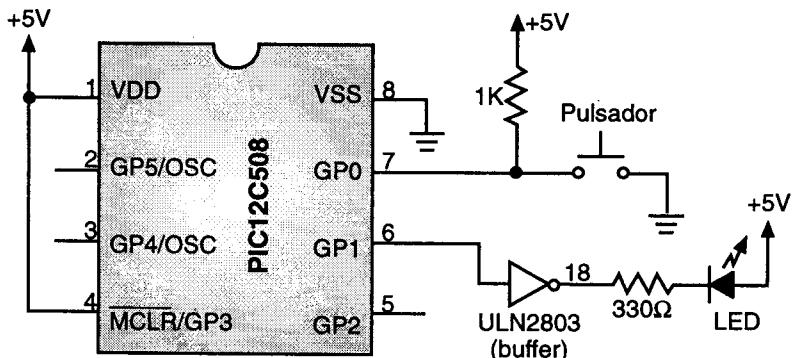
## *Capítulo 5*

# **El PIC12C50X**

- **Arquitectura**
- **Características especiales**
- **Conjunto de instrucciones**

## Proyecto N° 1: Cómo utilizar el PIC12C508

Para comprender mejor el uso de las características especiales de estos microcontroladores vamos a realizar un ejercicio simple que consiste en encender un LED cada vez que se oprime un pulsador. En este ejemplo se puede ver la forma de emplear el oscilador interno, de configurar los puertos y de configurar el pin de reset o MCLR (interno o externo). El circuito que se utiliza se muestra en la figura 6.1, consta de un pulsador, un LED y el microcontrolador. Cuando el pulsador está libre el micro lee un nivel lógico alto por el pin GP0 y cuando está oprimido un nivel lógico bajo; con este último estado el microcontrolador hace que se encienda el LED que se maneja a través del pin GP1.



**Figura 6.1.** Diagrama del circuito de la primera práctica con PIC12C508

El programa del microcontrolador debe tener en cuenta la programación del valor de calibración del reloj y el manejo de los puertos. Debe recordarse que el valor de calibración del reloj puede ser diferente para cada microcontrolador y que debe ser leído antes de grabarlo o borrarlo por primera vez. Para esta primera práctica vamos a utilizar el pin de reset o MCLR conectado a la fuente de alimentación externa, por lo tanto en el momento de grabar el microcontrolador se debe escoger la opción que permite habilitar dicho pin. Lo mismo se debe hacer cuando se selecciona el tipo de oscilador, en este caso debe escogerse la opción INTR (oscilador RC interno). En la figura 6.2 se tiene el listado del programa.

```
;Este programa enciende un LED cada vez que se oprime
;un pulsador
osccal equ 05h
gpio equ 06h

org 1ffh ;carga valor de calibración del
movlw 50h ;oscilador interno

org 00h ;pasa el valor de calibración
movwf osccal ;al registro osccal

inicio
 movlw 0fdh ;programa puertos
 tris gpio ;deja GP1 como salida para el LED
 bcf gpio,1 ;apaga el LED
 bcf gpio,1 ;
 btfsc gpio,0 ;pregunta estado del pulsador
 goto ciclo ;no está oprimido vuelve a preguntar
 bsf gpio,1 ;si está oprimido enciende el LED
 nop
 goto ciclo ;vuelve a probar estado del pulsador
end
```

**Figura 6.2.** Programa que lee el pulsador y enciende un LED con PIC12C508

Si se desea comprobar el funcionamiento del sistema de reset cuando este se configura para trabajar conectado internamente a la fuente de alimentación, se debe escoger la opción de deshabilitar el pin MCLR en el momento de grabar el microcontrolador. Se debe entonces desconectar dicho pin de la fuente de alimentación y verificar el funcionamiento del sistema.

Si se desea comprobar el funcionamiento del sistema de reset cuando este se configura para trabajar conectado internamente a la fuente de alimentación, se debe escoger la opción de deshabilitar el pin MCLR en el momento de grabar el microcontrolador. Se debe entonces desconectar dicho pin de la fuente de alimentación y verificar el funcionamiento del sistema.

## Proyecto N° 2: Manejo de un potenciómetro digital

Un potenciómetro digital tiene algunas ventajas sobre uno electromecánico como son: mayor precisión, mayor velocidad, menor tamaño, no existe desgaste mecánico, etc. Sus aplicaciones pueden ser: control de volumen digital, convertidor digital a análogo, control digital de una fuente variable y en general, cualquier aplicación que involucre la variación de una resistencia como medio para hacer control.

Esta práctica consiste en manejar un potenciómetro digital marca Xicor, utilizando como elemento principal un microcontrolador PIC12C508. Además, se tienen dos botones pulsadores para que el usuario indique si el valor de la resistencia debe aumentar o debe disminuir.

El circuito integrado del potenciómetro digital que se escogió es de marca Xicor. Su principal ventaja es poseer internamente una memoria EEPROM donde conserva el valor de resistencia que debe mantener entre sus terminales. Esto es de gran utilidad cuando se utiliza en sistemas que no poseen baterías de respaldo al momento de cortar la alimentación.

### Potenciómetro digital X9C503 de Xicor

Es un potenciómetro de estado sólido no volátil que contiene un arreglo de 99 resistencias y cuyas uniones son accesibles por el cursor o pin central del potenciómetro. La posición del cursor es controlada por los pines CS, INC y U/D; su posición entre las 99 resistencias puede ser almacenada en memoria EEPROM para no perder su valor cuando se retire la alimentación.

La alimentación del integrado (VCC) es de +5V y los extremos del potenciómetro (VH y VL) se pueden conectar entre  $\pm 5V$ . El hecho de tener 99 resistencias hace que el potenciómetro tenga 100 pasos, es decir que si se polariza entre 0 y +5V, se obtienen variaciones de 50mV en el cursor. Como se indica en la referencia del integrado (X9C104), su resistencia total es de 100kohm, también se consiguen en valores de 10kohm (X9C103) y 50kohm (X9C503). La figura 6.3 muestra el diagrama de pines del potenciómetro.

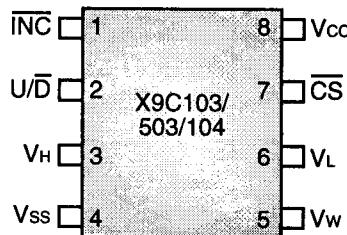


Figura 6.3. Diagrama de pines del potenciómetro digital

**Operación del potenciómetro.** Como se muestra en la figura 6.4, el X9C104 tiene 3 bloques funcionales: el primero lo componen el bloque de control, el contador y el decodificador; el segundo lo conforma la memoria no volátil y el tercero el arreglo de resistencias de salida.

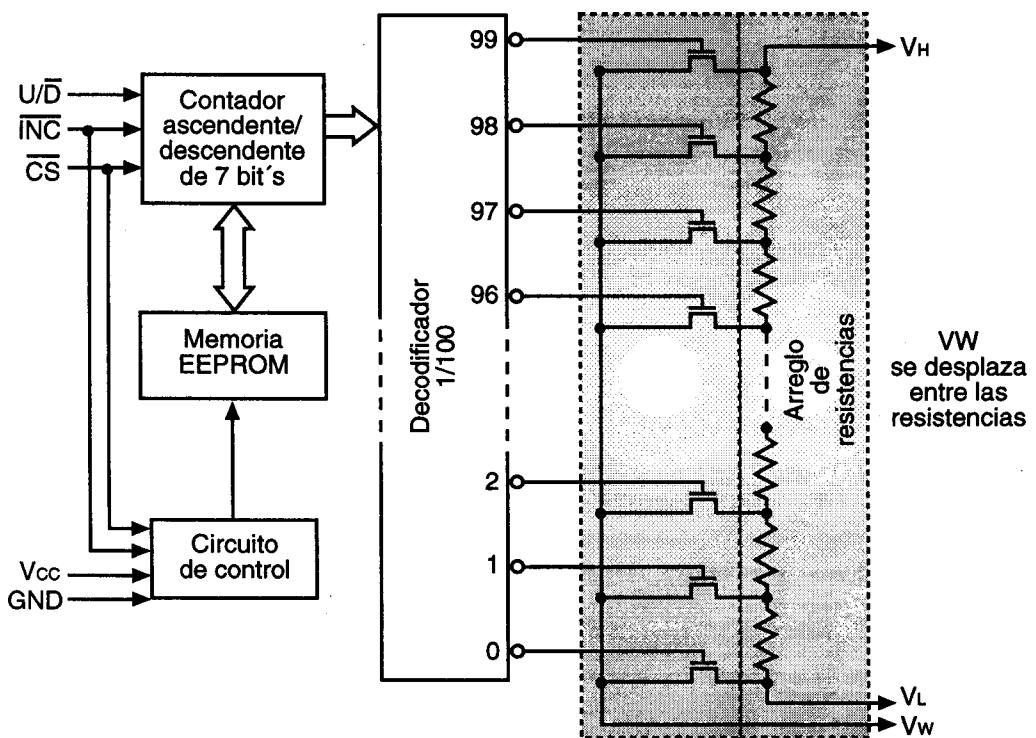


Figura 6.4. Diagrama funcional del potenciómetro digital

El bloque de control opera como un contador ascendente/descendente (Up/Down). La salida del contador es decodificada para seleccionar una sola salida que se conecte al cursor del potenciómetro. En condiciones normales, el contenido del contador puede ser almacenado en la memoria EEPROM.

Las entradas INC, U/D y CS controlan el movimiento del cursor a lo largo del arreglo de resistencias. El X9C104 es habilitado cuando el pin CS tiene un nivel bajo, un flanco de bajada en el pin INC hará que se incremente o decremente el contador de 7 bits (dependiendo de la polaridad del pin U/D). La salida de ese contador es decodificada para seleccionar 1 de 100 posiciones en el arreglo de resistencias.

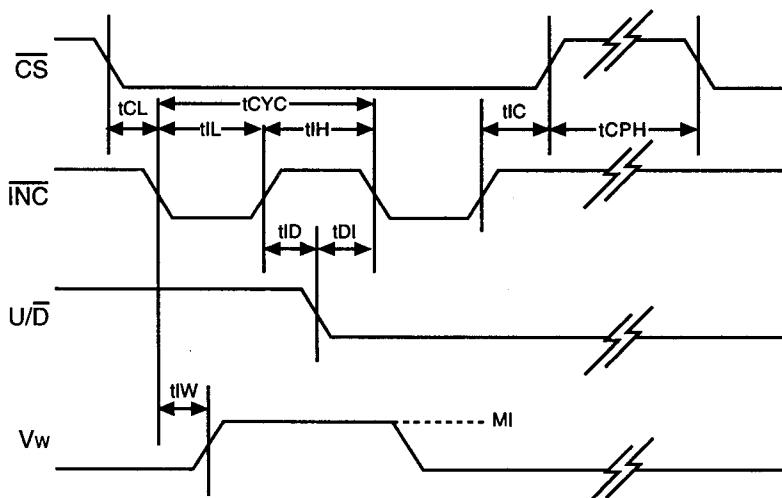


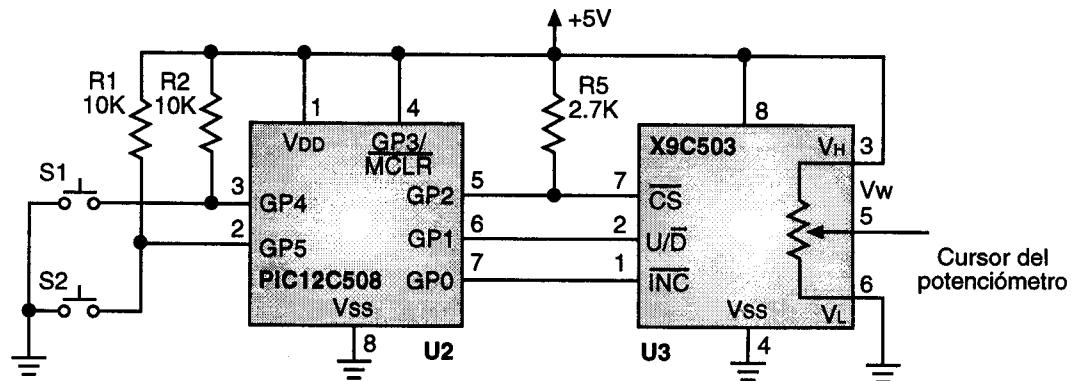
Figura 6.5. Diagrama de tiempo de las señales de control del potenciómetro

El valor del contador es almacenado en la memoria EEPROM cuando el pin CS retorna a un nivel alto, siempre y cuando el pin INC esté en ese nivel previamente. Cuando se retira la fuente de alimentación y se vuelve a conectar, el contador se carga con el valor almacenado en la memoria. El manejo de las señales INC, U/D y CS requiere una temporización adecuada. En la figura 6.5 se muestra el diagrama de tiempos y los valores mínimos para un correcto funcionamiento.

### Interface con el microcontrolador

Como se explicó anteriormente, el manejo adecuado de las señales del potenciómetro digital impone unas reglas específicas. Para obviar este problema, facilitar el manejo de las mismas y hacer más simple el manejo para el usuario, se utilizó un microcontrolador de 8 pines PIC12C508.

En la figura 6.6 se muestra la unión entre el PIC y el X9C104. Se puede ver que los pines de control del potenciómetro están unidos directamente al microcontrolador, en el pin de CS se ha conectado una resistencia de 2.7kohm a fuente. Esto para garantizar que la memoria EEPROM no altere su contenido con el ruido que se genera en el momento del encendido. Los pulsadores S1 (subir) y S2 (bajar) se conectan directamente a los pines del PIC, se utiliza una resistencia conectada a la fuente de alimentación para garantizar un nivel alto cuando no se oprimen las teclas. El pin de MCLR del microcontrolador se ha conectado a la fuente de alimentación para garantizar un reset adecuado.



**Figura 6.6. Diagrama esquemático del circuito completo**

En este circuito se utiliza la principal característica del PIC12C508, que consiste en eliminar la necesidad de componentes externos. Como se puede ver, no se ha conectado ningún tipo de oscilador ya que se utiliza la opción de oscilador interno de 4 MHz.

Los pines 3, 5 y 6 del X9C104 corresponden al potenciómetro digital propiamente dicho. Con este circuito se puede tener una resistencia variable idéntica a un potenciómetro electromecánico, la única restricción es que los extremos del mismo no se pueden conectar a una fuente de alimentación que sobrepase los  $\pm 5V$ .

**Software del microcontrolador.** El programa que se graba en el microcontrolador solamente debe encargarse de leer los pulsadores S1 y S2, de acuerdo a ellos debe generar las señales para que el potenciómetro aumente o disminuya su resistencia;

dichas señales corresponden a las mostradas en el diagrama de tiempo de la figura 3. Cuando se detecta que han oprimido uno de los botones, se debe colocar la señal del pin U/D en el estado lógico correspondiente (alto para aumentar resistencia y bajo para disminuirla). Luego se llama la rutina que genera el nivel lógico bajo en el pin de CS y que genera el flanco de bajada en el pin INC. Esta rutina es igual en ambos casos ya que la dirección en que se mueve el cursor del potenciómetro se ha definido previamente con el pin U/D.

En la figura 6.7 se muestra el programa correspondiente a la rutina de variación de resistencia. Cuando se está usando el reloj interno del microcontrolador (4 MHz), basta con escribir una instrucción NOP luego de asignar el estado lógico correspondiente al pin INC. Esto con el fin de cumplir los requisitos de tiempos exigidos por el potenciómetro digital (período de 4  $\mu$ s).

```
;Este programa controla un potenciómetro digital

osccal equ 5h ;puertos
gpio equ 6h
loops equ 0ch
loops2 equ 0dh
inc equ 0h ;salida para pulsos de variar resistencia
ud equ 1h ;salida escoge si aumenta o baja resistencia
cs equ 2h ;salida para seleccionar el integrado
mas equ 4h ;entrada botón de aumentar resistencia
menos equ 5h ;entrada botón de bajar resistencia

 org 1ffh
 movlw 50h

 org 000
 movwf osccal
 goto inicio ;va a iniciar programa principal

 org 005h

retardo ;subrutina de retardo de 200 milisegundos
 movlw d'200' ;el registro loops contiene el número
 movwf loops ;de milisegundos del retardo
top2 movlw d'110'
 movwf loops2
top nop
 nop
 nop
 nop
 nop
 nop
 decfsz loops2 ;pregunta si terminó 1 ms
 goto top
 decfsz loops ;pregunta si termina el retardo
 goto top2
 retlw 0

; ***** programa principal *****
inicio movlw 0f8h ;gpio se programa según el circuito
 tris gpio
 movlw 0c0h
 option
 btfss gpio,mas ;*** ciclo de lectura de teclas ***
 goto subir
 btfss gpio,menos
```

```

 movlw 0ffh
 movwf gpio
 goto inicio

subir bcf gpio,cs
 nop
 bsf gpio,ud
 nop
 nop
 nop
 nop
 bcf gpio,inc
 nop
 nop
 nop
 bcf gpio,inc
 nop
 nop
 bcf gpio,inc
 nop
 nop
 bcf gpio,cs
 nop
 call retardo
 goto inicio

bajar bcf gpio,cs
 nop
 bcf gpio,ud
 nop
 nop
 nop
 nop
 bcf gpio,inc
 nop
 nop
 nop
 bcf gpio,inc
 nop
 nop
 bcf gpio,inc
 nop
 nop
 bcf gpio,cs
 nop
 call retardo
 goto inicio

 end

```

**Figura 6.7.** Programa que controla la variación de la resistencia

Debe notarse que el pin de CS vuelve al estado lógico alto luego del pin INC. Esto garantiza que la posición del cursor del potenciómetro se almacene en la memoria EEPROM, para que pueda recuperarse luego de un corte de energía. Debe tenerse especial cuidado con los tiempos en la lectura de las teclas, por un lado para eliminar los problemas de ruido y por otro para generar los retardos adecuados luego de variar la resistencia. Esto se hace con el fin de permitir al usuario que avance de a un paso a la vez cuando lo desee. Si esto no se tiene en cuenta, bastará con un leve toque a las teclas para que el potenciómetro pase de la mínima a la máxima resistencia y viceversa.

Un detalle especial que se nota en este programa es que se configura el registro OPTION a pesar de que no se está utilizando ninguno de los elementos que allí se controlan (temporizador/contador o watchdog). Esto se debe a que el pin GP2 también se puede utilizar como entrada del temporizador/contador, por lo tanto si este se va a emplear como un pin de entrada/salida normal, en el registro OPTION se debe indicar que la fuente de pulsos para el temporizador/contador es el reloj interno y no el pin GP2/TOCKI.

El circuito que trabajamos en esta práctica se puede utilizar para realizar conversiones digitales a análogas, la única restricción es que se tienen sólo 100 valores diferentes, pero en muchos casos estos pueden ser suficientes.

## Proyecto N° 3: Temporizador con relevo

Esta práctica es muy simple pero de muchas aplicaciones, consiste en activar una carga de 110 VAC durante un tiempo determinado, la señal de activación la genera un interruptor. El tiempo de activación lo determina el programador en el momento de grabar el microcontrolador, para el ejemplo utilizamos un tiempo de 10 segundos. La carga que se conecta es un bombillo de 100 W, la cual se maneja mediante un relevo que es comandado desde un pin del micro. En la figura 6.8 se muestra el diagrama del circuito.

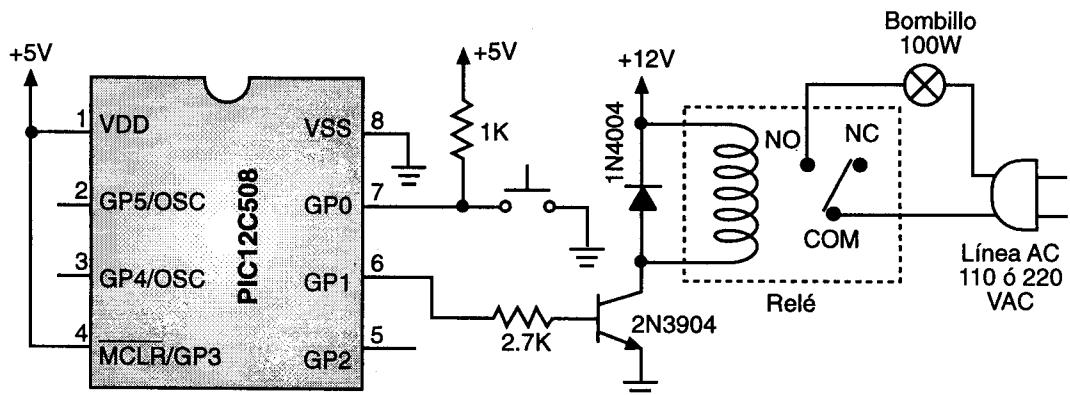


Figura 6.8. Diagrama del temporizador con relevo

El programa del microcontrolador se muestra en la figura 6-9, uno de los bloques más importantes lo forma el retardo programable que empieza con la etiqueta RETARDO. En este caso el registro W se carga con el número de milisegundos deseado y luego se hace el llamado a la subrutina. Para saber si se debe activar la carga el microcontrolador consulta el estado de pin donde se ha conectado el pulsador, cuando se detecta una señal válida se hace un retardo de 100 milisegundos para confirmar que no sea un pulso de ruido. Cuando se confirma que se debe activar la salida, se pone en «1» el pin que maneja el relevo.

```
;Este programa maneja un temporizador de x segundos,
;inicia su conteo cuando se presiona un pulsador
osccal equ 5h ;puertos
gpio equ 6h
loops equ 0ch
loops2 equ 0dh
veces equ 0eh
rele equ 1h
pulsa equ 0h

 org 1ffh ;calibraci&n del reloj
 movlw 50h
 org 000
 movwf osccal

 goto inicio ;va a iniciar programa principal
 org 005h

retardo ;subrutina de retardo de x milisegundos
 movwf loops ;# de milisegundos del retardo
```

```

top2 movlw d'110'
 movwf loops2
top nop
 nop
 nop
 nop
 nop
 nop
 decfsz loops2 ;pregunta si termino 1 ms
 goto top
 decfsz loops ;pregunta si termina el retardo
 goto top2
 retlw 0

; ***** programa principal *****

inicio movlw 0fdh ;gpio se programa segun el circuito
 tris gpio
 nop
 bcf gpio,rele ;apaga la salida del rele
 btfsc gpio,pulsa ;consulta estado del pulsador
 goto inicio ;si esta suelto vuelve a probar
 movlw d'100' ;si esta oprimido retarda 100 ms
 call retardo ;y vuelve a consultar
 btfsc gpio,pulsa ;si no confirma regresa a probar
 goto inicio ;si lo confirma va a encender la
 ;salida del relevo durante
 movlw d'40' ;veces * 1/4 de segundo = 10 segundos
 movwf veces

tempo bsf gpio,rele ;activa el rele
 movlw d'250'
 call retardo ;hace retardo de 0,25 s
 movlw 0fdh ;reprograma puertos
 tris gpio
 decfsz veces,1 ;prueba si terminaron 10 segundos
 goto tempo
 goto inicio ;si terminaro vuelve a empezar
 end

```

**Figura 6.9.** Programa del temporizador con relevo

Para hacer que la salida se encienda durante 10 segundos se realiza un ciclo donde se llama un retardo de 250 milisegundos durante el número de veces que sea necesario. Para el ejemplo se repite el ciclo 40 veces, es decir, si se llama 40 veces un retardo de 0.25 segundos se obtiene un tiempo total de 10 segundos. Si el usuario desea cambiar el tiempo de activación, sólo se debe cambiar el número que se carga en el registro llamado VECES. Luego de terminado ese período se regresa al ciclo donde se pregunta el estado del interruptor.

Este ejercicio puede aplicarse en diferentes formas, por ejemplo: Si se desea encender la luz de un pasillo cuando pasa alguna persona, solo se debe reemplazar el interruptor de activación por un sensor infrarrojo o por un sensor de presión en el piso.



## *Apéndice A*

# **Instrucciones de los PIC16F84 y 16C71**

Estos microcontroladores responden a una serie de instrucciones o códigos que se deben grabar en su memoria de programa, en total son 35. A continuación se encuentra una tabla con la lista completa y después una descripción de cada una de ellas con el fin de facilitar su aprendizaje.

Si d = 0 el resultado se almacena en W Si d = 1 el resultado se almacena en el registro				
Operaciones orientadas a registros				
Nemotécnico	Operación	Cód. de operación msb	lsb	Estados afectados
ADDWF f,d	Sumar W y f	00 0111	ffff	C,DC,Z
ANDWF f,d	AND entre W y f	00 0101	ffff	Z
CLRF f	Limpiar f	00 0001	ffff	Z
CLRW	Limpiar w	00 0001	0XXX XXXX	Z
COMF f,d	Complementar f	00 1001	ffff	Z
DECFSZ f,d	Decrementar f	00 0011	ffff	Z
DECFSZ f,d	Decrementar f, saltar si cero	00 1011	ffff	
INCF f,d	Incrementar f	00 1010	ffff	Z
INCFSZ f,d	Incrementar f, saltar si cero	00 1111	ffff	
IORWF f,d	OR entre W y f	00 0100	ffff	Z
MOVF f,d	Mover f	00 1000	ffff	Z
MOVWF f	Mover W a f	00 0000	1fff	
NOP	No operación	00 0000	0XX0	0000
RLF f,d	Rotar a la izquierda a través del carry	00 1101	ffff	C
RRF f,d	Rotar a la derecha a través del carry	00 1100	ffff	C
SUBWF f,d	Restar W de f	00 0010	ffff	C,DC,Z
SWAPF f,d	Intercambiar nibbles de f	00 1110	ffff	
XORWF f,d	OR exclusiva entre W y f	00 0110	ffff	Z
Operaciones orientadas a bits				
BCF f,b	Limpiar bit b de f	01 00bb	bfff	
BSF f,b	Activar bit b de f	01 01bb	bfff	
BTFSZ f,b	Probar bit b de f, saltar si es cero	01 10bb	bfff	
BTFSZ f,b	Probar bit b de f, saltar si es uno	01 11bb	bfff	
Operaciones literales y de control				
ADDLW k	Sumar literal k a W	11 111X	kkkk	C,DC,Z
ANDLW k	AND entre k y W	11 1001	kkkk	— Z —
CALL k	Llamar subrutina	10 0kkk	kkkk	T0,PD
CLRWD	Limpiar WDT	00 0000	0110	
GOTO k	Salta a dirección k	10 1kkk	kkkk	Z
IORLW k	OR entre k y W	11 1000	kkkk	
MOVLW k	Cargar a W con literal k	11 00XX	kkkk	
RETFIE	Retornar de interrupción	00 0000	0000	1001
RETLW k	Retornar y cargar a W con k	11 01XX	kkkk	
RETURN	Retornar de subrutina	00 0000	0000	1000
SLEEP	Ir al modo de bajo consumo	00 0000	0110	0011
SUBLW k	Restarle k a W	11 110X	kkkk	T0,PD
XORLW k	OR exclusiva entre k y W	11 1010	kkkk	C,DC,Z

<b>ADDLW</b>	Suma un valor literal al contenido del registro W.
--------------	----------------------------------------------------

Sintaxis: ADDLW k  
Operación: (W) + k k (W)  
Ciclos de instrucción: 1  
Bits del registro de estados que se afectan: C, DC, Z

Ejemplo : ADDLW 15  
Antes de la instrucción W = 10  
Después de la instrucción W = 25

<b>ADDWFF</b>	Suma el contenido de un registro al contenido del registro W.
---------------	---------------------------------------------------------------

Sintaxis: ADDWF f,d  
Operación: (W) + (f) k (destino)  
Ciclos de instrucción: 1  
Bits del registro de estados que se afectan: C, DC, Z

Ejemplo: ADDWF FSR,0  
Antes de la instrucción W = 17 FSR = C2  
Después de la instrucción W = D9 FSR = C2

<b>ANDLW</b>	Operación lógica AND entre un literal y el registro W.
--------------	--------------------------------------------------------

Sintaxis : ANDLW k  
Operación : (W) AND (k) k (W)  
Ciclos de instrucción : 1  
Bits del registro de estados que se afectan: Z

Ejemplo: ANDLW 5F  
Antes de la instrucción W = A3  
Después de la instrucción W = 03

<b>ANDWF</b>	Operación lógica AND entre un registro “f” y el registro W.
--------------	-------------------------------------------------------------

Sintaxis :	ANDWF f,d
Operación :	(W) AND (f) k (destino)
Ciclos de instrucción :	1
Bits del registro de estados que se afectan:	Z
Ejemplo :	ANDWF FSR,1 Antes de la instrucción                    W = 17      FSR = C2 Después de la instrucción                W = 17      FSR = 02

<b>BCF</b>	Pone en cero el bit “b” del registro “f”.
------------	-------------------------------------------

Sintaxis :	BCF            f,b
Operación :	0 k (f<b>)
Ciclos de instrucción :	1
Bits del registro de estados que se afectan:	Ninguno
Ejemplo :	BCF regis,7 Antes de la instrucción                regis = C7 Después de la instrucción            regis = 47

<b>BSF</b>	Pone en uno el bit “b” del registro “f”.
------------	------------------------------------------

Sintaxis :	BSF            f,b
Operación :	1k (f<b>)
Ciclos de instrucción :	1
Bits del registro de estados que se afectan:	Ninguno
Ejemplo :	BSF regis,7 Antes de la instrucción                regis = 0A Después de la instrucción            regis = 8A

**BTFSC** Prueba el bit “b” del registro “f” y salta una línea si está en cero.

Sintaxis : BTFSC f,b

Operación : salta si ( $f < b >$ ) = 0

Ciclos de instrucción : 1 (2)

Bits del registro de estados

que se afectan: Ninguno

Ejemplo : aquí BTFSC regis,0  
falso GOTO inicio  
verdad .....

Antes de la instrucción

contador de programa = aquí

Después de la instrucción

si el bit 0 del registro regis = 0

contador de programa = verdad

si el bit 0 del registro regis = 1

contador de programa = falso

**BTFSS** Prueba el bit “b” del registro “f” y salta una línea si está en uno.

Sintaxis : BTFSS f,b

Operación : salta si ( $f < b >$ ) = 1

Ciclos de instrucción : 1 (2)

Bits del registro de estados

que se afectan: Ninguno

Ejemplo : aquí BTFSS regis,0  
falso GOTO inicio  
verdad .....

Antes de la instrucción

contador de programa = aquí

Después de la instrucción

si el bit 0 del registro regis = 1

contador de programa = verdad

si el bit 0 del registro regis = 0

contador de programa = falso

<b>CALL</b>	Llama una subrutina que está ubicada en la posición de memoria o etiqueta "k".
-------------	--------------------------------------------------------------------------------

Sintaxis : CALL k  
 Operación : (PC) + 1 k pila, k k (PC)  
 Ciclos de instrucción : 2  
 Bits del registro de estados que se afectan: Ninguno  
 Ejemplo : aquí CALL rutina

Antes de la instrucción contador de programa = aquí  
 Después de la instrucción contador de programa = rutina  
 pila = dirección aquí

<b>CLRF</b>	Borra el contenido del registro "f", lo carga con 00.
-------------	-------------------------------------------------------

Sintaxis : CLRF f  
 Operación : 00 k (f)  
 Ciclos de instrucción : 1  
 Bits del registro de estados que se afectan: Z

Ejemplo : CLRF regis

Antes de la instrucción regis = 5A  
 Después de la instrucción regis = 00

<b>CLRW</b>	Borra el contenido del registro W, lo carga con 00.
-------------	-----------------------------------------------------

Sintaxis : CLRW  
 Operación : 00 k (W)  
 Ciclos de instrucción : 1  
 Bits del registro de estados que se afectan: Ninguno

Ejemplo : CLRW

Antes de la instrucción W = 5A  
 Después de la instrucción W = 00

**CLRWDT**

Borra el conteo del Watchdog Timer.

Sintaxis : CLRWDT

Operación : 00 k WDT

Ciclos de instrucción : 1

Bits del registro de estados

que se afectan: TO, PD

Ejemplo : CLRWDT

Antes de la instrucción

Después de la instrucción

Contador WDT = ?

Contador WDT = 00

TO = 1, PD = 1

**COMF**

Complementa el contenido del registro “f”.

Sintaxis : COMF f,d

Operación : (f) k (destino)

Ciclos de instrucción : 1

Bits del registro de estados

que se afectan: Z

Ejemplo : COMF regis,0

Antes de la instrucción

Después de la instrucción

regis = 13, W = ?

regis = 13, W = EC

**DECF**

Decrementa el contenido del registro “f”.

Sintaxis : DECF f,d

Operación : (f) - 1 k (destino)

Ciclos de instrucción : 1

Bits del registro de estados

que se afectan: Z

Ejemplo : DECF regis, 1

Antes de la instrucción

Después de la instrucción

regis = 13

regis = 12

<b>DECFSZ</b>	Decrementa el contenido del registro “f”; si el contenido queda en 00, salta una línea.
---------------	-----------------------------------------------------------------------------------------

Sintaxis : DECFSZ f,d  
 Operación : (f) - 1 k (destino), salta si el resultado es cero  
 Ciclos de instrucción : 1 (2)  
 Bits del registro de estados que se afectan: Ninguno

Ejemplo : aquí DECFSZ regis, 1  
                   GOTO ciclo  
                   continúa .....

Antes de la instrucción	contador de programa = aquí
Después de la instrucción	regis = regis - 1
	si regis = 0, entonces
	contador de programa = continua
	si regis ≠ 0, entonces
	contador de programa = aquí + 1

<b>GOTO</b>	El contador de programa salta a la dirección “k”.
-------------	---------------------------------------------------

Sintaxis : GOTO k  
 Operación : k k PC  
 Ciclos de instrucción : 2  
 Bits del registro de estados que se afectan: Ninguno

Ejemplo : GOTO ciclo

Antes de la instrucción	contador de programa = ?
Después de la instrucción	contador de programa = ciclo

<b>INCF</b>	Incrementa el contenido del registro “f”.
-------------	-------------------------------------------

Sintaxis : INCF f,d  
 Operación : (f) + 1 k (destino)  
 Ciclos de instrucción : 1  
 Bits del registro de estados que se afectan: Z

Ejemplo : INCF regis, 1

Antes de la instrucción	regis = 24
Después de la instrucción	regis = 25

<b>INCFSZ</b>	Incrementa el contenido del registro “f”; si el contenido de “f” queda en 00, salta una línea.
---------------	------------------------------------------------------------------------------------------------

Sintaxis :                   **INCFSZ f,d**  
Operación :                 (f) + 1 k (destino), salta si el resultado es cero  
Ciclos de instrucción :    1 (2)  
Bits del registro de estados  
que se afectan:           Ninguno

Ejemplo :                 aquí **DECFSZ regis, 1**  
                                **GOTO ciclo**  
                                continúa.....

Antes de la instrucción      contador de programa = aquí  
Después de la instrucción    regis = regis + 1  
                                  si regis = 0 , entonces  
                                  contador de programa = continua  
                                  si regis ≠ 0 , entonces  
                                  contador de programa = aquí + 1

<b>IORLW</b>	Operación lógica OR entre el registro W y el literal “k”.
--------------	-----------------------------------------------------------

Sintaxis :                   **IORLW k**  
Operación :                 (W) OR (k) k (W)  
Ciclos de instrucción :    1  
Bits del registro de estados  
que se afectan:           Z

Ejemplo :                 **IORLW 35**

Antes de la instrucción      W = 9A  
Después de la instrucción    W = BF

<b>IORWF</b>	Operación lógica OR entre el registro W y el registro “f”.
--------------	------------------------------------------------------------

Sintaxis :                   **IORWF f,d**  
Operación :                 (W) OR (f) k (destino)  
Ciclos de instrucción :    1  
Bits del registro de estados  
que se afectan:           Z

Ejemplo :                 **IORWF regis,0**

Antes de la instrucción      regis = 13 , W = 91  
Después de la instrucción    regis = 13 , W = 93

**MOVLW**

Carga el registro W con el literal "k".

Sintaxis : MOVLW k

Operación : k k (W)

Ciclos de instrucción : 1

Bits del registro de estados  
que se afectan: Ninguno

Ejemplo : MOVLW 5A

Antes de la instrucción	W = ?
Después de la instrucción	W = 5A

**MOVF**

Mueve el contenido del registro "f".

Sintaxis : MOVF f,d

Operación : (f) k (destino)

Ciclos de instrucción : 1

Bits del registro de estados  
que se afectan: Z

Ejemplo : MOVF regis,0

Antes de la instrucción	W = ?
Después de la instrucción	W = valor guardado en regis

**MOVWF**

Mueve el contenido del registro W al registro "f".

Sintaxis : MOVWF f

Operación : (W) k (f)

Ciclos de instrucción : 1

Bits del registro de estados  
que se afectan: Ninguno

Ejemplo : MOVWF OPTION

Antes de la instrucción	OPTION = ?, W = 48
Después de la instrucción	OPTION = 48, W = 48

<b>NOP</b>	No hace nada.
------------	---------------

Sintaxis : NOP  
Operación : k k (W)  
Ciclos de instrucción : 1  
Bits del registro de estados que se afectan: Ninguno

Ejemplo : NOP

<b>OPTION</b>	Carga el registro OPTION con el contenido del registro W .
---------------	------------------------------------------------------------

Sintaxis : OPTION  
Operación : W k OPTION (Programa algunas funciones especiales)  
Ciclos de instrucción : 1  
Bits del registro de estados que se afectan: Ninguno

Ejemplo : OPTION

Esta instrucción se considera obsoleta, pero por facilidad en el aprendizaje, se utiliza en el nivel básico.

<b>RETFIE</b>	Retorno del llamado a interrupción.
---------------	-------------------------------------

Sintaxis : RETFIE  
Operación : pila k contador de programa , 1 k intcon,gie  
Ciclos de instrucción : 2  
Bits del registro de estados que se afectan: Ninguno

Ejemplo : RETFIE

Antes de la instrucción	contador de programa = ?
Después de la instrucción	contador de programa = pila

<b>RETLW</b>	Retorno de interrupción y carga el registro W con el literal "k".
--------------	-------------------------------------------------------------------

Sintaxis : RETLW k  
 Operación : k k W, pila k contador de programa  
 Ciclos de instrucción : 2  
 Bits del registro de estados que se afectan: Ninguno

Ejemplo : CALL tabla

tabla ADDWF PC  
 RETLW k1  
 RETLW k2  
 .  
 .  
 RETLW kn

Antes de la instrucción W = 07  
 Despues de la instrucción W = k7

<b>RETURN</b>	Retorno desde subrutina.
---------------	--------------------------

Sintaxis : RETURN  
 Operación : pila k contador de programa  
 Ciclos de instrucción : 2  
 Bits del registro de estados que se afectan: Ninguno  
 Ejemplo : RETURN

Despues de la interrupción, contador de programa = pila

<b>RLF</b>	Rote el contenido del registro "f" a la izquierda, usando el carry
------------	--------------------------------------------------------------------

Sintaxis : RLF f,d  
 Operación : Rota el contenido del registro "f" a la izquierda a través del carry. Si d=0, el resultado se guarda en W, si d=1 en "f".  
 Ciclos de instrucción : 1  
 Bits del registro de estados que se afectan: C  
 Ejemplo : RLF regis,0

Antes de la instrucción C = 0, W=? , regis = 1110 0110  
 Despues de la instrucción C = 1, W = 1100 1100  
 regis = 1110 0110

**RRF**

Rote el contenido del registro “f” a la derecha, usando el carry

Sintaxis :

RRF f,d

Operación :

Rota el contenido del registro “f” a la derecha a través del carry. Si d=0, el resultado se guarda en W, si d=1 en “f”.

Ciclos de instrucción : 1

Bits del registro de estados que se afectan: C

Ejemplo : RRF regis,0

Antes de la instrucción C = 0, W=? , regis = 1110 0110  
Después de la instrucción C = 0, W = 0111 0011  
regis = 1110 0110

**SLEEP**

Entra en modo dormido (standby).

Sintaxis : SLEEP

Operación : 00 k WDT, 1 k TO, 0 k PD

Ciclos de instrucción : 1

Bits del registro de estados que se afectan: TO, PD

Ejemplo : SLEEP

**SUBLW**

Resta el contenido del registro W de el literal “k” (usando el método de complemento a dos).

Sintaxis : SUBLW k

Operación : k - (W) k (W)

Ciclos de instrucción : 1

Bits del registro de estados que se afectan: C, DC, Z

Ejemplo 1 : SUBLW 02

Antes de la instrucción W = 1, C = ?  
Después de la instrucción W = 1, C = 1 (positivo)

Ejemplo 2 : Antes de la instrucción W = 2, C = ?

Después de la instrucción W = 0, C = 1 (cero)

Ejemplo 3 : Antes de la instrucción W = 3, C = ?

Después de la instrucción W = FF , C = 0 (negativo)

<b>SUBWF</b>	Resta el contenido del registro W de el registro “f” (usando el método de complemento a dos).
--------------	--------------------------------------------------------------------------------------------------

Sintaxis : SUBWF f,d  
 Operación : (f) - (W) k (destino)  
 Ciclos de instrucción : 1  
 Bits del registro de estados que se afectan: C, DC, Z

Ejemplo 1 : SUBWF regis,1

Antes de la instrucción	regis = 3, W = 2, C = ?
Después de la instrucción	regis = 1, W = 2, C = 1 (positivo)

Ejemplo 2 :	Antes de la instrucción	regis = 2, W = 2, C = ?
	Después de la instrucción	regis = 0, W = 2, C = 1 (cero)

Ejemplo 3 :	Antes de la instrucción	regis = 1, W = 2, C = ?
	Después de la instrucción	regis = FF, W = 2, C = 0 (negativo)

<b>SWAPF</b>	Intercambia los cuatro bits altos y los cuatro bits bajos del registro “f”.
--------------	-----------------------------------------------------------------------------

Sintaxis : SWAPF f,d  
 Operación : f<3:0> k d<7:4>  
               f<7:4> k d<3:0>  
 Ciclos de instrucción : 1  
 Bits del registro de estados que se afectan: Ninguno

Ejemplo 1 : SWAPF regis,0

Antes de la instrucción	regis = A5, W = ?
Después de la instrucción	regis = A5, W = 5A

<b>TRIS</b>	Carga el registro TRIS/(programación de los puertos como entrada/salida) con el contenido del registro W.
-------------	-----------------------------------------------------------------------------------------------------------

Sintaxis : TRIS f  
Operación : W k (registro TRIS del puerto “A ó B”)  
Ciclos de instrucción : 1  
Bits del registro de estados que se afectan: Ninguno

Ejemplo 1 : TRIS puertoA

Esta instrucción se considera obsoleta, pero por facilidad en el aprendizaje, se utiliza en el nivel básico.

<b>XORLW</b>	Operación lógica XOR entre el registro W y el literal “k”.
--------------	------------------------------------------------------------

Sintaxis : XORLW k  
Operación : (W) XOR (k) k (W)  
Ciclos de instrucción : 1  
Bits del registro de estados que se afectan: Z

Ejemplo : XORLW AF

Antes de la instrucción W = B5  
Después de la instrucción W = 1A

<b>XORWF</b>	Operación lógica XOR entre el registro W y el registro “f”.
--------------	-------------------------------------------------------------

Sintaxis : XORWF f,d  
Operación : (W) XOR (f) k (destino)  
Ciclos de instrucción : 1  
Bits del registro de estados que se afectan: Z

Ejemplo : XORWF regis,1

Antes de la instrucción regis = AF, W = B5  
Después de la instrucción regis = 1A, W = B5