



UNIVERSIDAD
AUTÓNOMA SAN FRANCISCO
Resolución N° 196-2010-CONAFU

MICROCONTROLADORES PIC

**FUNDAMENTOS Y APLICACIONES
UN ENFOQUE DIDÁCTICO**



ING. DAVID APAZA CONDORI



UNIVERSIDAD
AUTÓNOMA SAN FRANCISCO
Resolución N° 196-2010-CONAFU

ESCUELA DE POSGRADO

MICROCONTROLADORES PIC

**FUNDAMENTOS Y APLICACIONES
UN ENFOQUE DIDÁCTICO**

MATERIAL EDUCATIVO

ING. DAVID APAZA CONDORI

DEDICATORIA

DEDICO ESTE LIBRO A MI FAMILIA
A MI QUERIDA ESPOSA CARMEN Y A MI AMA-
DO HIJO FERNANDO DAVID, QUIENES HAN
ESTADO A MI LADO TODO ESTE TIEMPO, LOS
QUIERO MUCHO, GRACIAS

RESUMEN

El microcontrolador es uno de los logros más sobresalientes del siglo XX. Hoy en día, existen casi 15,000 millones de PIC's de alguna clase en uso. Para la mitad del siglo próximo, es posible que el microcontrolador típico tenga mayor poder de cómputo que las supercomputadoras más veloces de hoy. Actualmente los podemos encontrar en cualquier sitio: microondas, frigoríficos, coches, aviones, mandos a distancia, radios, televisores, etc.

El objetivo principal de este libro consiste en que el lector comprenda el funcionamiento de los microcontroladores y sea capaz de emplearlos en situaciones que requieren una solución práctica. Dentro de este libro es posible encontrar información acerca del origen de esta línea de microcontroladores PIC, así como las diferentes familias que integran la amplia gama de dispositivos PIC Micro. Un aspecto fundamental es la explicación clara y explícita de la operación del microcontrolador PIC16F84A, dispositivo al cual está enfocado este libro y del cual se tratan temas relacionados con sus características, su arquitectura, la organización de memoria y los diversos periféricos que posee dicho PIC. En lo que se refiere a la programación de este dispositivo tan popular, se ha creado un software llamado MPLAB, el cual es el programa compilador y simulador más popular para este propósito.

De la misma forma, para efectos de facilitar la grabación de un programa en el PIC, existen grabadores de microcontrolador como el: NOPPP, PonyProg2000. El primero de ellos consiste en un programa muy elemental, mientras que el segundo es un software que soporta la programación de varios dispositivos.

Finalmente, se considera que la práctica es un factor fundamental para la implementación de sistemas mediante microcontroladores, es por esto que se ha incluido dentro de esta obra, una sección de proyectos que ayudarán al usuario a familiarizarse con el manejo del PIC16F84A.

ÍNDICE

DEDICATORIA.....	1
RESUMEN.....	3
ÍNDICE.....	5
CAPÍTULO 1	
1.1 Introducción.....	8
1.2 Los puertos del PIC.....	12
1.3 Los registros TRISA y TRISB.....	21
1.4 Periféricos.....	23
1.5 Set de instrucciones.....	27
CAPÍTULO 2	
1. Introducción.....	56
2. Funcionamiento del TMR0.....	58
2.1. Estructura general del TMR0.....	58
2.2. Entrada de reloj del módulo TMR0.....	59
2.3. El prescaler.....	60
2.4. El registro TMR0.....	62
2.5. Sincronización del TMR0.....	64
3. Cálculo de temporizaciones.....	64
CAPÍTULO 3	
1. Introducción.....	92
2. Las interrupciones.....	93
2.1. Registro de control INTCON.....	96
2.2. Interrupción externa INT.....	99
2.3. Interrupción del TMR0.....	100
2.4. Interrupción del PORTB.....	101
2.5. Interrupción de la EEPROM.....	102
CAPÍTULO 4	
1. Introducción.....	114
2. Tipos de memorias del LCD.....	117
2.1. DD RAM (Display Data RAM).....	117
2.2. CG RAM (carácter generator RAM).....	118
3. Periféricos del LCD.....	119
4. Inicialización del LCD.....	121
4.1. Secuencia de inicialización del módulo LCD.....	121
5. Comandos del LCD.....	124
5.1. Comandos de control.....	127
CAPÍTULO 5	
1. Introducción.....	158
2. Conversión analógica digital.....	159
3. ADC del PIC.....	161
3.1. Configuración de ADC.....	161
3.2. Operación del ADC.....	167



CAPÍTULO 1

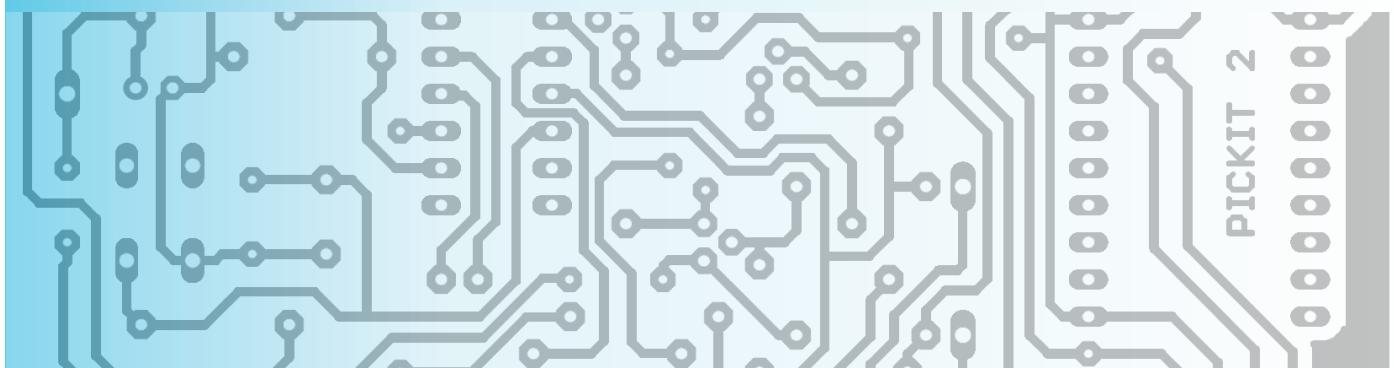
MANEJO DE PUERTOS

CONTENIDO DEL CAPÍTULO

- 1.1 Introducción
- 1.2 Los puertos del PIC
- 1.3 Los registros TRISA y TRISB
- 1.4 Periféricos
- 1.5 Set de instrucciones
- Proyecto 1: Lectura de puertos
- Proyecto 2: Compuerta lógica AND
- Proyecto 3: Decodificador de tres bits

APRENDEREMOS A:

- Identificar las diferentes partes que componen un PIC
- Identificar los puertos del PIC y sus características eléctricas
- Configurar correctamente los puertos del PIC ya sea como entrada o salida
- Utilizar correctamente las instrucciones binarias básicas del PIC





1.1 Introducción

Un microcontrolador es un computador completo de limitadas prestaciones, que está contenido en un chip de un circuito integrado y se destina a gobernar una sola tarea. Para que el microcontrolador realice las operaciones que deseamos es necesario grabar en su memoria de programa un conjunto de instrucciones que constituyen el programa de aplicación. En primer lugar debemos conocer perfectamente las especificaciones de la tarea que debe ejecutar. Hay que tener en cuenta que podemos llegar al mismo resultado con distintos programas, siendo óptimo aquél que esté mejor estructurado y no realice operaciones innecesarias, ahorrando de esta forma espacio en la memoria de programa y tiempo.

Podemos citar como ejemplos de programas de aplicación a: el programa que controla un semáforo, el programa que controla un brazo de robot, el programa que controla una alarma de casa, entre otras aplicaciones. Las distintas aplicaciones requieren de ingreso y salida de datos hacia y desde la parte interna del PIC, esta tan importante función la cumplen los puertos del PIC, que dependiendo de la gama pueden tener de 2 a 5 puertos, los cuales pueden ser configurados como entrada o salida, según los requerimientos de la aplicación.

Configurar correctamente los puertos del PIC es lo primero que un programador debe aprender, seguido del manejo del software de desarrollo MPLAB, y un buen simulador electrónico por ejemplo el PROTEUS que a criterio de muchos es el mejor. El la siguiente figura presentamos uno de los primeros microcontroladores en llegar a nuestro país, uno de mucha aceptación y aplicación, ideal para los principiantes.

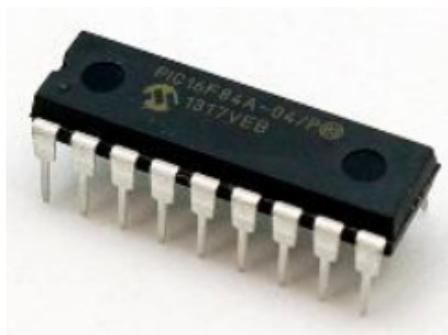


Figura 1.1. El microcontrolador 16F84 de Microchip, un PIC de mucha aplicación.



Para escribir el programa fuente de la tarea o aplicación, será necesario un editor de texto que trabaje con caracteres ASCII. El lenguaje de programación más utilizado es el lenguaje ensamblador, aunque también se usa el lenguaje Basic o el C. El lenguaje ensamblador es el lenguaje natural de los PIC, y está más cerca al lenguaje máquina, como todo lenguaje presenta ventajas y desventajas, entre otras ventajas podemos citar las siguientes:

1. Permite utilizar menos memoria de programa.
2. Permite optimizar el tiempo de ejecución.
3. Permite trabajar directamente con los registros.
4. Facilita la paginación de la memoria de programa.
5. Permite el uso directo del temporizador y contador.

Recomendamos utilizar el lenguaje asembler, todas las aplicaciones desarrolladas en este texto están en dicho lenguaje, solo es una recomendación, se puede trabajar con el lenguaje C o Basic, esto depende del programador.



¿ Sabías que . . . ?

Algunos puertos del PIC están multiplexados, es decir que cumplen doble o triple función, normalmente las funciones van separadas por un carácter que es el /.

Para desarrollar aplicaciones, proyectos entre otros, se requieren de herramientas tanto lógicas como físicas, es decir software y hardware respectivamente, respecto al software la herramienta principal es el conocido y famoso MPLAB IDE, que lo proporciona y facilita el mismo fabricante de los PIC, es decir MICROCHIP, es necesario indicar que existen diferentes marcas de PIC, podemos citar algunas como por ejemplo: ATMEL, MOTOROLA, INTEL y MICROCHIP, entonces porque elegimos la marca MICROCHIP, es simple, estos PIC son más comerciales, son económicos y hay mucha información de los mismos en INTERNET, como proyectos, manuales, tutoriales, videos, software libre, simuladores, sin desmerecer a las otras marcas.



1 MANEJO DE PUERTOS

Otra herramienta lógica es el simulador, que es de mucha utilidad cuando se requiere analizar un prototipo inicial, analizar los registros internos, analizar la secuencia de ejecución del programa, analizar las subrutinas e interrupciones, como vemos esta herramienta es de mucha utilidad, permite detectar y corregir errores en el programa, antes de implementar el proyecto. En la siguiente figura se ve el entorno principal del MPLAB, que como todo software tiene diferentes versiones, hace muchos años atrás inicio con la versión 5, ahora esta en la 8.

The screenshot shows the MPLAB IDE v8.14 interface. The window title is "MPLAB IDE v8.14 - [D:\Documentos\oscilador.asm]". The menu bar includes File, Edit, View, Project, Debugger, Programmer, Tools, Configure, Window, and Help. The toolbar contains icons for file operations like Open, Save, and Print, along with other development tools. A status bar at the bottom shows "Checksum: 0x3bff", "PIC16F84A", "W:0", "z dc c", and "bank 0". The main area displays assembly code:

```
_config H'3FF1'
#include<"p16f84.inc">
dato1 EQU 0x0C
ORG 0x00
    bsf STATUS,5
    movlw 0x00
    movwf TRISA
    clrf TRISB
    movlw b'00000111'
    movwf OPTION_REG
    bcf STATUS,5
    clrf PORTB
;=====
lab_01 baf PORTB,0
    call retardo
    bcf PORTB,0
    call retardo
    goto lab_01
;=====
retardo movlw d'3'
    movwf dato1
label_2 clrf TMRO
    bcf INTCON,2
label_1 btfss INTCON,2
    goto label_1
    decfsz dato1,1
    goto label_2
    return
END
```

Figura 1.2. El MPLAB versión 8.14, software que permite el desarrollo de aplicaciones, facilita un editor de texto, ensambla y compila los códigos.

Otra herramienta lógica clave es el grabador de PIC, quien se encarga de enviar el programa hacia la memoria interna del PIC, es decir realiza la transferencia del programa de la PC hacia el PIC, esta herramienta tiene su complemento que es la parte física, es decir un circuito físico sin el cual no sirve de nada. El grabador es una tarjeta electrónica, que posee entre otros elementos un zócalo para insertar en PIC, un puerto de comunicaciones, que puede ser serial, USB, en algunos casos paralelo.



Existen diferentes modelos de grabadores, desde el punto de vista del puerto pueden ser serial o paralelo, desde el punto de vista de la alimentación eléctrica, pueden ser con alimentación o sin alimentación, pueden ser universales o no universales, los grabadores actuales son universales, usan el puerto USB y no requieren alimentación externa adicional, es decir toman energía del mismo puerto USB de la PC, en la siguiente figura se observa un grabador de PIC universal, que utiliza como parte de su circuito a otro PIC que normalmente es el 18F2550.

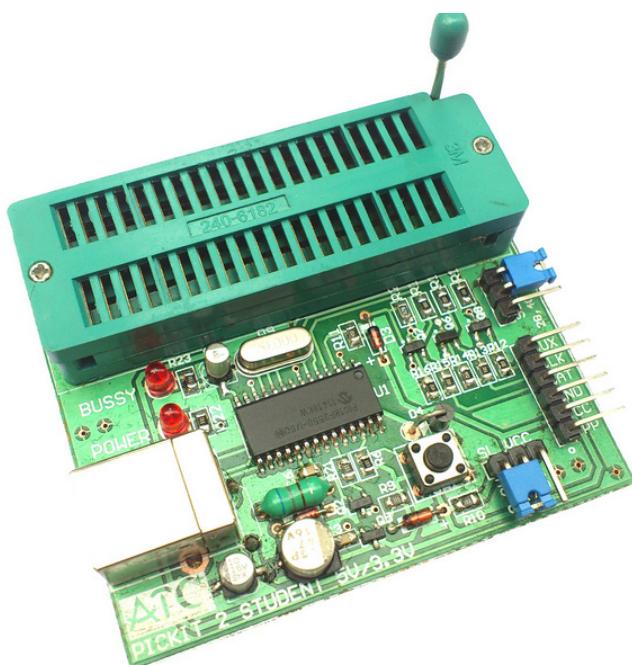


Figura 1.3. El grabador de PIC universal, no requiere alimentación externa, hace uso de otro microcontrolador para lograr la comunicación serial.

Como ya hemos visto existen herramientas lógicas y físicas, que permiten desarrollar al 100% los proyectos basados en microcontrolador, otra herramienta física (hardware) importante es el entrenador de PIC, que es una tarjeta electrónica básicamente de desarrollo, que puede tener diferentes elementos desde los más básicos hasta los más complejos, normalmente vienen con un PIC, uno o dos display, pulsadores digitales, conectores, teclado matricial, pantalla LCD entre otros, esta tarjeta permite ver en tiempo real el funcionamiento del programa, es decir si funciona o no el programa, es de mucha utilidad cuando se desarrollan aplicaciones con PIC, en la siguiente figura se muestra un entrenador básico pero de mucha utilidad.



1 MANEJO DE PUERTOS



Figura 1.4. El entrenador básico para el microcontrolador 16F84 de MICROCHIP.

De no contar con esta herramienta física, se puede optar por usar un protoboard, que es una base fija con diferentes conexiones internas, estas conexiones son uniformes y siguen un patrón único, el único inconveniente es el ruido electromagnético, ruido que se puede filtrar por los cables o terminales de los diferentes componentes, por lo que se recomienda reducir al máximo el tamaño de los terminales de los diferentes dispositivos y mantener un orden en el cableado. El ruido electromagnético, puede alterar el normal funcionamiento del PIC.

1.2 Los puertos del PIC

Los puertos del PIC son un conjunto de pines, líneas, que forman en su conjunto el puerto en si, no se debe confundir un pin con un puerto, el pin es una parte del puerto, entonces un conjunto de pines forman un puerto, por ejemplo el puerto B posee un total de 8 pines, que pueden ser configurados como entrada o salida, los mismos que cumplen una función elemental, que es permitir el ingreso o salida de datos, desde y hacia la parte interna del PIC, que no es otra cosa que comunicar la parte interna del PIC con el mundo externo, entonces un factor importante para el desarrollo de proyectos basados en microcontroladores es la interfaz de entrada y salida. A través de los pines del PIC asociados a los puertos de entrada y salida, el PIC puede interactuar con otros circuitos externos, enviando señales digitales en forma de unos y ceros, señales de control o comando, recibiendo señales digitales en forma de unos y ceros, que no son mas que estímulos correspondientes a variables externas.



¿ Sabías que . . . ?

Los pines de un puerto pueden configurarse como entrada o salida según los requerimientos, podemos configurar todos los pines como entrada o todos los pines como salida, unos pines como entrada y los restantes como salida, también se pueden configurar alternadamente, todo depende de la aplicación.

Los pines configurados como entrada pueden adquirir datos interpretando el valor de voltaje como un valor lógico 0 o 1, si el voltaje a la entrada es de 0 voltios el PIC lo interpreta como un 0 lógico, si el voltaje a la entrada es de 5 voltios el PIC lo interpreta como un 1 lógico, debemos indicar que el voltaje a la entrada no tiene que ser exacto, existe un rango de valores que son aceptados por el PIC, este rango de valores lo determina el fabricante en su data sheet, para el caso de las salidas, es lo mismo, si un puerto esta configurado como salida, sus pines se energizan con 5 voltios, lo que corresponde a un 1 lógico como salida, esta señal puede activar cualquier otro sistema, activar un motor, o simplemente prender un led y poner sus pines a 0 voltios cuando la salida es un 0 lógico, esto apaga cualquier sistema, motor o led.

Por lo general los puertos de datos son bidireccionales, es decir pueden configurarse como entrada o salida, según los requerimientos de la aplicación, para configurar un puerto como entrada o salida se utilizan los registros TRISA y TRISB, siendo esta la única función de dichos registros, el procedimiento para configurar los puertos lo veremos en los siguientes puntos.



¿ Sabías que . . . ?

Los puertos del PIC pueden configurarse como entrada o salida, pero nunca como entrada y salida al mismo tiempo, un pin nunca puede funcionar como entrada y salida al mismo tiempo.



1 MANEJO DE PUERTOS

Gracias a sus puertos el microcontrolador puede responder a eventos externos y realizar una cierta acción, como variar las señales de salida de acuerdo al valor en las entradas. Para responder a eventos externos, el microcontrolador cuenta con un recurso conocido como interrupciones. Las interrupciones son eventos aleatorios, que se generan dentro o fuera del microcontrolador, detienen la ejecución del programa principal, para atender un programa secundario o subprograma, el uso y aplicación de las interrupciones se explicará en los siguientes capítulos.

Una aplicación real es por ejemplo, un botón pulsador conectado a un pin de entrada. Una vez pulsado, se genera una señal de interrupción, que iniciará la ejecución de la subrutina de interrupción, que puede ser un temporizador, que al finalizar el tiempo programado active una señal de salida, esta señal podría apagar un horno por ejemplo o simplemente encender un led.

Para determinar las necesidades de entradas y salidas del sistema es conveniente dibujar un diagrama de bloques del mismo, de tal forma que sea sencillo identificar la cantidad y tipo de señales a controlar. Una vez realizado este análisis puede ser necesario añadir periféricos, hardware externo o cambiar a otro microcontrolador más adecuado a ese sistema.

1.2.1 Los puertos A y B

Los puertos del microcontrolador 16F84 de Microchip, son un conjunto de pines o líneas que están agrupados, cada grupo de pines o líneas conforma un puerto completo, el 16F84 tiene implementado dos puertos, el puerto A y el puerto B, ambos puertos son bidireccionales y pueden ser configurados como entrada o salida, de esta forma los puertos permiten el ingreso o salida de datos hacia o desde la parte interna del microcontrolador, el puerto A está conformado por cinco pines o líneas, que van desde el RA0 hasta el RA4, como se observa en la figura 1.5, el pin 3 está multiplexado, es decir cumple doble función, que corresponde al RA4/T0CKI, éste es el único pin del puerto A que está multiplexado, y puede trabajar como puerto RA4 o como entrada del contador de eventos externos T0CKI, para este último el pin 3 debe ser configurado como entrada.

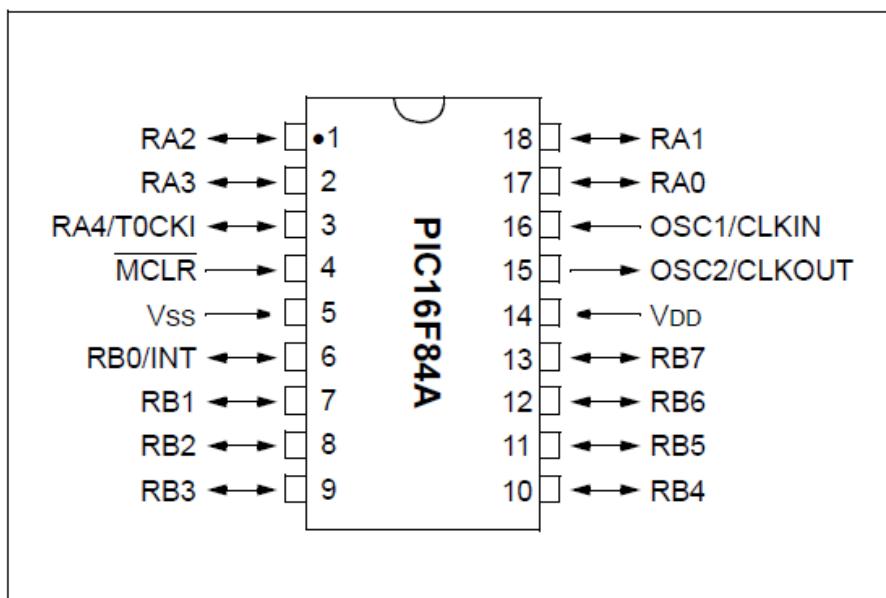


Figura 1.5. El diagrama de periféricos, muestra con mucho detalle la función que cumple cada pin.

El pin 3 cumple doble función, y no se puede utilizar ambas funciones al mismo tiempo, a través de la configuración se elige que función cumplirá dicho pin, y si será de entrada o salida, el otro puerto con más pines es el puerto B, y está conformado por ocho pines o líneas, que van desde el RB0 hasta el RB7, como se observa en la figura 1.5, el pin 6 está multiplexado, es decir cumple doble función, que corresponde al RB0/INT, éste es el único pin o línea que está multiplexado, y puede trabajar como puerto RB0 o como entrada de interrupción externa INT, pero nunca como puerto y entrada de interrupción externa al mismo tiempo, a través de la configuración se elige la función que cumplirá el pin o línea y si será de entrada o salida.

Los puertos A y B del microcontrolador pueden ser configurados como entrada o salida, según los requerimientos de la aplicación, si el puerto está configurado como salida, los pines del puerto trabajan en modo fuente, suministran corriente, y la máxima corriente que puede suministrar una línea o pin es de 20mA, ya sea del puerto A o del puerto B, y la máxima corriente que puede suministrar todo un puerto es de 50mA, para el puerto A, y de 100mA, para el puerto B, si el puerto está configurado como entrada, los pines trabajan en modo sumidero, reciben corriente, y la máxima corriente que puede ingresar por una línea o pin es de



1 MANEJO DE PUERTOS

25mA, la corriente máxima de entrada para todo el puerto es de 80mA, para el puerto A y de 150mA para el puerto B. Si en la aplicación existen cargas que consumen corrientes mayores, como por ejemplo: motores paso a paso, relés, varios leds y otros periféricos que necesiten más corriente de la indicada por el fabricante, habrá que aplicar un circuito acoplador como por ejemplo: los buffers, transistores de potencia, que se encarguen de controlar la corriente.

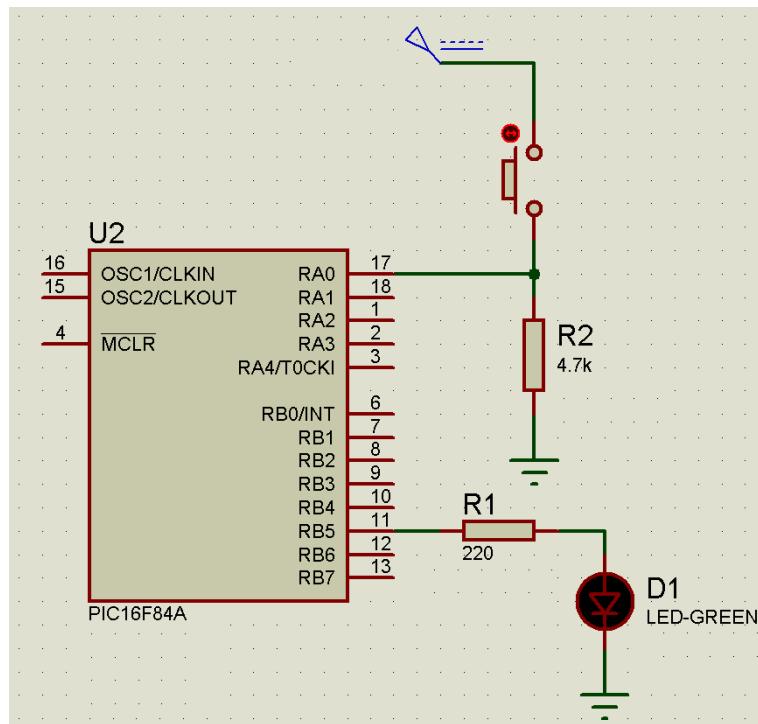


Figura 1.6. El pin RA0 configurado como entrada y trabajando en modo sumidero, el pin RB5 configurado como salida y trabajando en modo fuente.

Se debe activar las resistencias de pull-up cada vez que sea necesario, y no dejar pines o líneas de entrada en estado flotante, en la electrónica digital se manejan dos estados lógicos, que son el 1 y el 0, por ejemplo: un sistema basado en microcontrolador con entradas y salidas, si una de las entradas se conecta a la referencia negativa a través de un interruptor, observando la conexión, veremos que el cero ya está definido por el interruptor cuando éste está cerrado, pero si el interruptor no se activa, es decir está abierto, no hay un estado definido porque la entrada no tiene contacto con la referencia negativa, sería un pin sin conexión, esto se llama estado flotante y puede causar el mal funcionamiento del sistema.



Ese problema se soluciona activando una resistencia interna entre el pin de entrada y Vdd, llamada resistencia de pull-up, su objetivo es asegurar un 1 lógico cuando el interruptor esté abierto. Si lo que se desea es asegurar un 0 lógico, la resistencia se conecta a Vss y recibe el nombre de resistencia de pull-down. Se aconseja no dejar un pin sin conexión o en estado flotante, si un pin o línea está configurado como entrada, y está en estado flotante, por ese pin ingresan unos y ceros aleatoriamente, eso altera el normal funcionamiento del microcontrolador.

¿ Sabías que . . . ?

Existen microcontroladores con más de dos puertos, por ejemplo el 16F877 posee 5 puertos y muchos de ellos están multiplexados. Por ejemplo el puerto A puede trabajar como analógico o digital según configuración.

1.2.2 Los Registros PORTA y PORTB

Es necesario indicar que existen dos registros internos asociados a los puertos, no se debe confundir registro con puerto, así como existe un puerto A con 5 pines, existe un registro interno llamado PORTA, el puerto A con el registro PORTA están íntimamente relacionados, ya explicamos que un puerto es un conjunto de pines a través del cual ingresan o salen datos, pero esos datos cuando ingresan al PIC lo hacen directamente a un registro interno, o cuando salen lo hacen desde un registro interno, esa es la relación íntima a la que hacemos referencia, un registro es un conjunto de bits, para el PIC 16F84 todos sus registros internos son de 8 bits, como se observa en la figura 1.7.

STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7						bit 0	

Figura 1.7. El registro STATUS o de estado, está conformado por ocho bits y cada bit cumple una función importante.



1 MANEJO DE PUERTOS

El puerto B tiene ocho pines, y su registro correspondiente está formado por ocho bits, cada pin se corresponde con un bit, por ejemplo: el pin RB0 se corresponde con el bit 0 del registro PORTB, el pin RB1 se corresponde con el bit 1 del registro PORTB, y eso ocurre con todos los pines, como se muestra en la figura 1.8, para el puerto A es lo mismo, pero existe una diferencia, el puerto A tiene solamente 5 pines y su registro correspondiente PORTA tiene 8 bits, la relación es la siguiente, el bit 0 del registro PORTA se conecta con el pin RA0 del puerto A, el bit 1 se conecta con el RA1, el bit 2 se conecta con el RA2, como no existen los pines RA5, RA6 y RA7, los bits 5, 6 y 7 del PORTA no se utilizan, pero si existen internamente y su valor por defecto es cero.

Estos registros en algunos casos son llamados palabras o bytes, que significa lo mismo, en los registros se puede realizar las operaciones de lectura o escritura de datos, aplicando las instrucciones de movimiento de datos MOV y sus variantes, deseo precisar que hay algunos registros cuyos bits son de solo lectura, por ejemplo: el bit Z del registro STATUS, el bit RBIF del registro INTCON, entre otros bits. Esos bits son controlados por el sistema y no por el usuario.

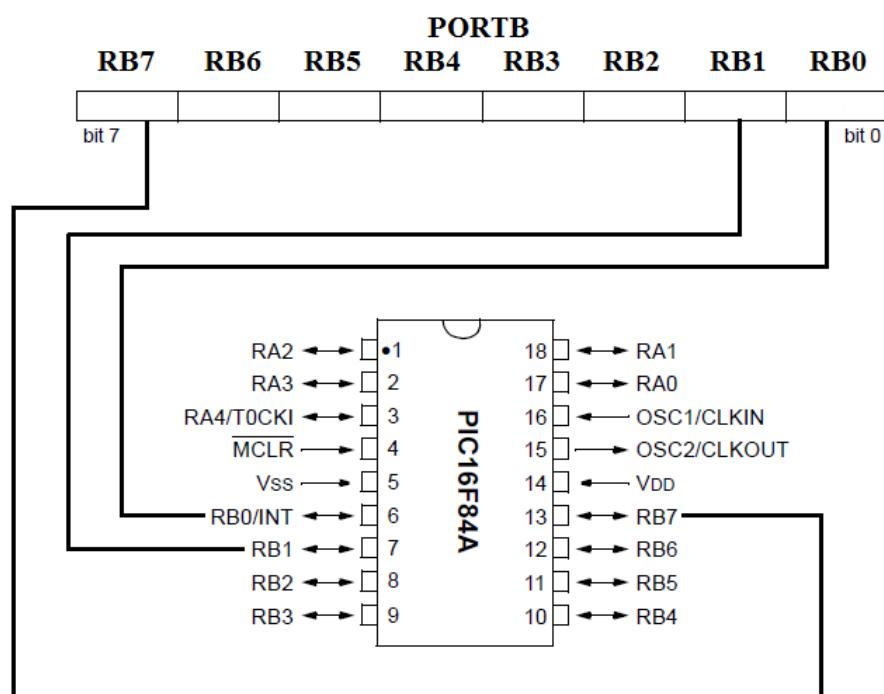


Figura 1.8. La relación que existe entre el puerto B y el registro PORTB, la relación es de pin a bit.



El desconocimiento de los atributos de los bits, lectura o escritura, los bancos de memoria y sus registros, el direccionamiento de los bancos de memoria, pueden generar errores en el programa y en consecuencia el mal funcionamiento de la aplicación. En el PIC existen muchos registros internos, estos registros están contenidos en bancos de memoria, un banco de memoria no es más que un conjunto de registros ordenados secuencialmente, cada registro tiene un nombre técnico y único, que lo diferencia de los otros registros, además ocupa una posición en el banco de memoria, esa posición es la dirección de memoria y se representa por un número en formato hexadecimal, hay microcontroladores que tienen dos hasta cuatro bancos de memoria, el 16F84 tiene dos bancos de memoria, a cada banco se le asigna un nombre, banco 0 y banco 1, se debe tener muy en cuenta que registros están en el banco 0 y que registros están en el banco 1, hay registros que únicamente se encuentran en el banco 0, por ejemplo: PORTA y PORTB, y hay registros que únicamente se encuentran en el banco 1, por ejemplo: TRISA y TRISB, y hay registros que se encuentran en ambos bancos, por ejemplo: STATUS y PLC, antes de utilizar uno de estos registros se debe direccionar o seleccionar correctamente el banco de memoria.



¿ Sabías que . . . ?

Los bits 5 y 6 del registro STATUS permiten seleccionar los bancos de memoria, para el 16F84 solo se usa el bit 5, puesto que solo tiene dos bancos de memoria.

En los bancos de memoria existe un espacio de 68 bytes que el fabricante dejó libre, este espacio es conocido como la memoria RAM y es ocupado por las variables del programa, este espacio inicia en la dirección 0x0C y termina en la dirección 0x4F del banco 0, de acuerdo al espacio libre se pueden declarar hasta 68 variables, que es una cantidad suficiente para el desarrollo de aplicaciones, el procedimiento para direccionar los bancos de memoria se explicará en los siguientes puntos, en la figura 1.9 se muestra la representación gráfica de los dos bancos de memoria del PIC 16F84, el cual fue copiado del manual del fabricante.



1 MANEJO DE PUERTOS

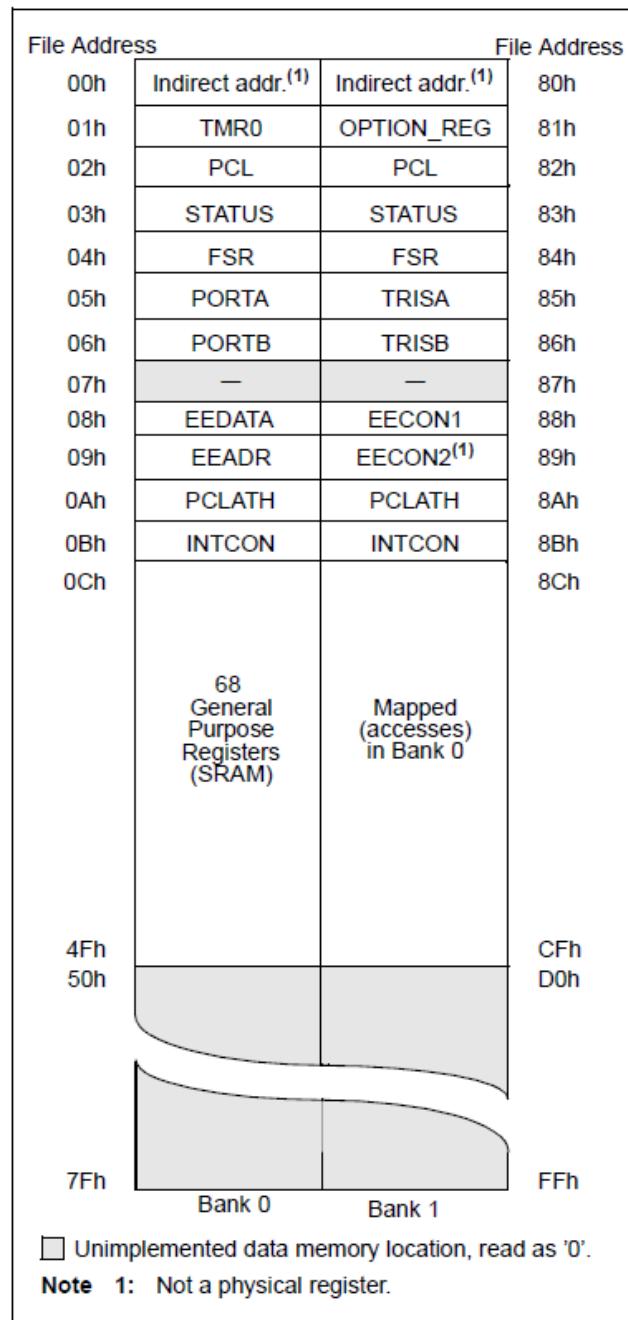


Figura 1.9. Los dos bancos de memoria del PIC 16F84, el bank 0 y el bank 1, los 68 bytes de memoria RAM para las variables.

Como se observa en la figura anterior, el PIC 16F84 tiene dos bancos de memoria, llamados bank 0 y bank 1, en el bank 0 se encuentra la memoria RAM, que inicia en la posición 0Ch y termina en la posición 4Fh, también se observa que de la posición 50h hasta la 7Fh no está implementado y se lee como cero.



1.3 Los registros TRISA y TRISB

Para el desarrollo de aplicaciones debemos configurar los puertos del microcontrolador como entrada o salida, según sea el caso, si le asignamos un cero 0 a un pin éste será salida y si le asignamos un uno 1 éste será entrada, ésta asignación de pines se hace programando los registros TRISA y TRISB, el TRISA es un registro donde se almacenan los bits que definen un pin como entrada o salida del puerto A. Recordemos que el puerto A sólo tiene 5 pines, por lo tanto los tres últimos bits no se utilizan.

Los registros TRISA y TRISB sirven para configurar los puertos A y B del microcontrolador, configuran cada pin de los puertos, definen si el pin será de entrada o salida, existe una correspondencia entre el registro TRISA y el puerto A, el registro TRISA configura el puerto A, bit a bit, si el bit 0 del TRISA vale 0, entonces el pin RA0 del puerto A queda configurado como salida, si el bit 3 del TRISA vale 1, entonces el pin RA3 del puerto A queda configurado como entrada, es así para todos los bits y pines, para todos los puertos existentes, siempre debemos recordar que:

- Un 1 configura el pin como entrada
- Un 0 configura el pin como salida
- TRISA para los pines del PUERTO A (5 bits)
- TRISB para los pines del PUERTO B (8 bits)
- El bit 0 del TRISA configura al RA0
- El bit 1 del TRISA configura al RA1
- El bit 2 del TRISA configura al RA2
- El bit 3 del TRISA configura al RA3

Cuando el PIC arranca todos sus puertos están configurados como entrada, el banco seleccionado es el 0, todo esto es por defecto, es importante verificar en que banco se encuentran los registros a utilizar, los registros TRISA y TRISB se encuentran en el banco 1, antes de leer o escribir en estos registros, se debe acceder al banco 1, esto se logra a través del registro STATUS, escribiendo un 1 en el bit 5, utilizando las instrucciones correspondientes.



1 MANEJO DE PUERTOS

Por ejemplo:

Si TRISA es igual a 00011110, entonces todos los pines del puerto A serán entradas, salvo RA0 que está como salida. Tal como se observa en la tabla 1.1.

Si TRISB es igual a 00000001, entonces todos los pines del puerto B serán salidas, salvo RB0 que está como entrada. Tal como se observa en la tabla 1.2.

Tabla 1.1. Configuración del puerto A, relación entre el valor del registro TRISA y el estado del puerto A, es decir entrada o salida.

TRISA		PUERTO A	
BIT	VALOR	PIN	ESTADO
Bit 0	0	RA0	SALIDA
Bit 1	1	RA1	ENTRADA
Bit 2	1	RA2	ENTRADA
Bit 3	1	RA3	ENTRADA
Bit 4	1	RA4	ENTRADA
Bit 5	0	NO EXISTE	INDETERMINADO
Bit 6	0	NO EXISTE	INDETERMINADO
Bit 7	0	NO EXISTE	INDETERMINADO

Tabla 1.2. Configuración del puerto B, relación entre el valor del registro TRISB y el estado del puerto B, es decir entrada o salida.

TRISB		PUERTO B	
BIT	VALOR	PIN	ESTADO
Bit 0	1	RB0	ENTRADA
Bit 1	0	RB1	SALIDA
Bit 2	0	RB2	SALIDA
Bit 3	0	RB3	SALIDA
Bit 4	0	RB4	SALIDA
Bit 5	0	RB5	SALIDA
Bit 6	0	RB6	SALIDA
Bit 7	0	RB7	SALIDA



El código asembler para configurar el puerto A y el puerto B, del ejemplo anterior y de acuerdo a las tablas 1.1 y 1.2. Es el siguiente:

No se pretende explicar o detallar el código asembler, no corresponde a este punto, pero considero necesario mostrar un adelanto de la programación.

```
bsf      STATUS,5    ;Acceder al banco 1
movlw   b'00000001'
movwf   TRISB     ;TRISB igual a 00000001
movlw   b'00011110'
movwf   TRISA     ;TRISA igual a 00011110
bcf     STATUS,5    ;Acceder al banco 0
```

1.4 Periféricos

El microcontrolador 16F84 es un chip con diferentes periféricos, los puertos forman parte de estos periféricos, pero existen otros que son de mucha importancia ver la figura 1.10, como se observa en la figura, el microcontrolador tiene un total de 18 pines, cada pin cumple una función elemental, por ejemplo: el reset, osc, Vss, Vdd, entre otros. Veamos:

Los pines 5 y 14, son respectivamente las patitas de masa y alimentación Vss y Vdd, Vdd está comprendido entre 2V y 6V, valor típico 5V, existe un rango de tolerancia que lo indica el fabricante, no necesariamente es un valor exacto de 5V, se recomienda utilizar el regulador de voltaje LM7805 y verificar que el voltaje de rizado sea mínimo.

Los pines 15 y 16, OSC1/CLKIN y OSC2/CLKOUT, son los pines de entrada para la señal de reloj, un tren de pulsos a una determinada frecuencia, esta frecuencia normalmente está en el orden de los MHz, existen diferentes fuentes que generan esta señal de reloj, por ejemplo: una red RC, un cristal, un oscilador con el timer 555, un generador de señales, en fin, un factor elemental es la precisión de la señal, una base de tiempo o periodo del pulso con una precisión absoluta, de esto depende la velocidad de trabajo del PIC y los retardos programados.



1 MANEJO DE PUERTOS

El cristal es la fuente más utilizada debido a su gran precisión, se conecta directamente a estos dos pines en paralelo, cualquier otro tipo de fuente sea una red RC, oscilador externo, entre otros, se utiliza las opciones de CLKIN y CLKOUT, que son: la entrada para una señal de reloj externa y la salida para una señal de reloj respectivamente, por el pin CLKOUT sale una señal de reloj cuya frecuencia es la cuarta parte de la frecuencia inicial, esta señal de salida es opcional y se puede aplicar a otros sistemas digitales, los distintos elementos de la familia 16F8X, dependiendo de la nomenclatura que utilizan tienen distintas características de funcionamiento, que se relacionan directamente con la frecuencia máxima de funcionamiento, el tipo de oscilador utilizado, entre otros.

Los microcontroladores PIC, permiten cuatro tipos de osciladores externos para aplicarles la frecuencia de funcionamiento. Durante el proceso de grabación, antes de introducir el programa en memoria, debe indicarse el tipo de oscilador empleado en los bits FOSC1 y FOSC2 de la palabra de configuración. Los tipos de osciladores que puede utilizar el 16F84 son:

1. Oscilador de cristal o resonador de alta velocidad “HS” (High Speed Crystal/Resonator): Es un oscilador de alta frecuencia, el rango de la frecuencia está comprendida entre 4MHz y 20MHz.
2. Oscilador o resonador cerámico “XT” (Crystal/Resonator): Se trata de un oscilador estándar que permite una frecuencia de reloj comprendida entre 100KHz y 4MHz, es un oscilador de frecuencia media.
3. Oscilador de cristal de cuarzo o resonador cerámico de baja potencia “LP” (Low Power Crystal): Se trata de un oscilador de bajo consumo, con un cristal o resonador diseñado para trabajar con frecuencias comprendidas entre 32KHz y 200KHz, es un oscilador de baja frecuencia.
4. Oscilador tipo “RC”: Es un oscilador de bajo coste formado por una red RC. Se trata de un oscilador de baja precisión que depende de la estabilidad de la red RC, pero como contrapartida esta su bajo precio que lo hace interesante para muchas aplicaciones.



El circuito para cualquiera de las configuraciones anteriores se representa en la figura 1.10, el valor de los condensadores “22pF” depende de la frecuencia del cristal y lo indica el fabricante en su manual, no olvidar que la frecuencia máxima de trabajo del PIC es de 20MHz, considerar estos valores para el buen funcionamiento del mismo. Se recomienda ver la tabla de datos del fabricante.

El pin 4, MCLR/Vpp: es la entrada de Reset, se activa con un nivel bajo, es decir con un 0 lógico, su función es reiniciar el programa, colocar a 0x00 el registro PC, también es utilizada para cambiar el modo del PIC a modo programación, aplicando una tensión de programación, que es de 12V.

Los pines 17, 18, 1, 2 y 3, RA0-RA4/T0CKI: respectivamente, corresponden a cinco líneas bidireccionales de E/S del PORTA. Es capaz de entregar niveles TTL cuando la tensión de alimentación aplicada en Vdd es de 5V. El pin RA4, si se programa como salida es de colector abierto. Como entrada puede programarse en funcionamiento normal o como entrada del contador/temporizador TMR0, este pin está multiplexado y cumple doble función.

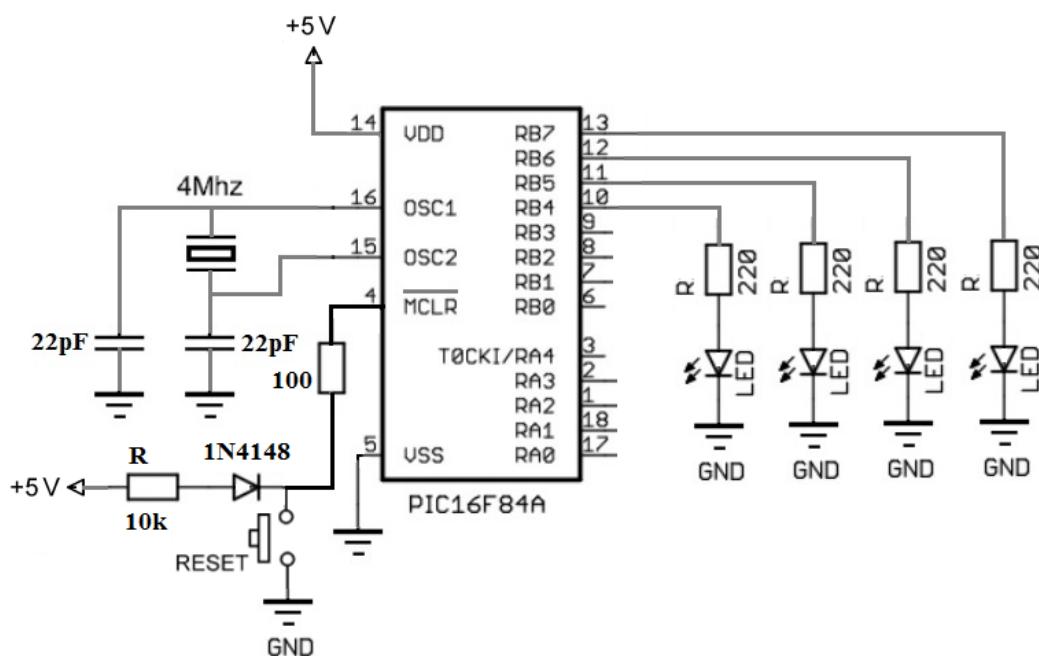


Figura 1.10. Circuito electrónico básico para el PIC 16F84, se observa los componentes elementales como son: el cristal con sus respectivos condensadores, el botón de reset, Vdd y Vss.



1 MANEJO DE PUERTOS



¿ Sabías que . . . ?

El PIC 16F84 tiene una palabra de configuración de 14 bits, los bits 0 y 1 de esta palabra permiten seleccionar el tipo de oscilador que utilizará el microcontrolador.

RB0-RB7: Pines 6, 7, 8, 9, 10, 11, 12 y 13 respectivamente, corresponden a ocho líneas bidireccionales de E/S del PORTB. Es capaz de entregar niveles TTL cuando la tensión de alimentación aplicada en Vdd es de 5V.

El pin RB0 puede programarse además como entrada de interrupción externa INT. Los pines RB4 a RB7 pueden programarse para responder a interrupciones por cambio de estado. Los pines RB6 y RB7 corresponden con las líneas de entrada de reloj y entrada de datos respectivamente, cuando está en modo programación.

En resumen los datos técnicos del 16F84 son los siguientes:

- Memoria de programa: 1Kx14 de tecnología Flash
- Memoria de datos RAM: 68 bytes
- Memoria de datos EEPROM: 64 bytes
- Pila (stack): De 8 niveles
- Interrupciones: 4 tipos de interrupción, internas y externas
- Encapsulado: Plástico DIP de 18 patitas
- Frecuencia de trabajo: 20MHz máxima
- Temporizadores: Sólo uno, el TMR0. También tiene “Perro guardián WDT”
- Líneas de E/S Digitales: 13 (5 Puerto A y 8 Puerto B)
- Corriente máxima absorbida: 80mA Puerto A y 150mA Puerto B
- Corriente máxima suministrada: 50mA Puerto A y 100mA Puerto B
- Corriente máxima absorbida por línea: 25mA
- Corriente máxima suministrada por línea: 20mA
- Voltaje de alimentación (Vdd): De 2V a 6V DC



1.5 Set de instrucciones

El PIC16F84 es un microcontrolador de gama media y es de tipo RISC, esto quiere decir que tiene un juego de instrucciones reducido, en total 35 instrucciones o nemáticos que son la base de funcionamiento del PIC. Al igual que los bits de los registros, sería complicado memorizar estas instrucciones, se recomienda utilizar este texto como guía, las instrucciones fundamentalmente se dividen en tres tipos. Esta división depende del tipo de datos con los que trabajan:

1. Instrucciones orientadas a los registros o bytes.
2. Instrucciones orientadas a los bits.
3. Operaciones con literales y de control.

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

f : Registro al que afecta la instrucción

w : Acumulador (Working register)

b : Número de bit (hay instrucciones que afectan a un solo bit)

k : Constante (un número)

d : Selección de destino del resultado de la instrucción, puede ser “0” o “1”, si es “0” el resultado se guarda en el acumulador (w) y si es “1” se guarda en el registro f al que afecta la instrucción.

Instrucciones orientadas a registros:

addwf f,d

Suma w y el registro f, el resultado lo guarda según d (si d = 0 se guarda en w y si d = 1 se guarda en f)

andwf f,d

Realiza la operación AND lógica entre w y f, el resultado lo guarda según d



1 MANEJO DE PUERTOS

clrf f

Borra el registro f (pone todos sus bits a cero)

clrw

Borra el acumulador w

comf f,d

Calcula el complemento del registro f, los bits que están a “0” los pone a “1” y viceversa. El resultado lo guarda según d

decf f,d

Decrementa f en uno (le resta uno). El resultado lo guarda según d

decfsz f,d

Decrementa f en uno y salta la siguiente instrucción si el resultado es cero. El resultado lo guarda según d

incf f,d

Incrementa f en uno (le suma uno). El resultado lo guarda según d

incfsz f,d

Incrementa f y se salta la siguiente instrucción si el resultado es 0 (cuando se desborda un registro vuelve al valor 00h). El resultado lo guarda según d

iorwf f,d

Realiza la operación lógica OR entre w y f. El resultado lo guarda según d

movf f,d

Mueve el contenido del registro f a w si d = 0 (si d = 1 lo vuelve a poner en el mismo registro f)

movwf f

Mueve el valor de w a f

**nop**

No hace nada, solo pierde el tiempo durante un ciclo

rlf f,d

Rota el registro f hacia la izquierda a través del bit CARRY (todos los bits se mueven un lugar hacia la izquierda, el bit 7 de f pasa al CARRY y el bit CARRY pasa al bit 0 de f). El resultado lo guarda según d

rrf f,d

Lo mismo que rlf pero hacia la derecha

subwf f,d

Resta f y w ($f - w$). El resultado lo guarda según d

swapf f,d

Intercambia los 4 primeros bit de f por los otros cuatro. El resultado lo guarda según d

xorwf f,d

Realiza la operación lógica XOR (OR exclusiva) entre w y f. El resultado lo guarda según d

Instrucciones orientadas a bits:**bcf f,b**

Pone a “0” el bit b del registro f

bsf f,b

Pone a “1” el bit b del registro f

btfsc f,b

Se salta la siguiente instrucción si el bit b del registro f es “0”



1 MANEJO DE PUERTOS

btfss f,b

Se salta la siguiente instrucción si el bit b del registro f es “1”

Instrucciones orientadas a constantes y de control:

addlw k

Le suma el valor k al acumulador w (resultado en w)

andlw k

Operación lógica AND entre w y el valor k (resultado en w)

call k

Llamada a subrutina cuyo inicio esta en la dirección k

clrwdt

Borra el registro Watchdog

goto k

Salta a la dirección k de programa

iorlw k

Operación lógica OR entre w y el valor k (resultado en w)

movlw k

Carga el acumulador con el valor k

retfie

Instrucción para retornar de la interrupción

retlw k

Carga el valor k en W y retorna de la subrutina

return

Retorna de una subrutina



sleep

El pic pasa a modo de Standby

sublw k

Resta a w el valor de k (w - k)

xorlw k

Realiza la operación lógica XOR (OR exclusiva) entre w y el valor de k. (El resultado en w)

Proyecto 1: Lectura de puertos

A) Planteamiento del programa:

El programa consiste en ingresar un dato por el puerto A y sacarlo por el puerto B, el dato es un número en formato binario de 4 bits, en formato decimal los números van desde el 0 hasta el 15, el dato ingresado por el puerto A sale casi de inmediato por el puerto B, el tiempo que demora en salir es de 2 microsegundos, este tiempo depende de la frecuencia del cristal, se ha determinado que los datos ingresan por el puerto A, éste debe ser configurado como entrada, los pines a utilizar son: RA0, RA1, RA2 y RA3, y los datos salen por el puerto B, éste debe ser configurado como salida, los pines a utilizar son: RB0, RB1, RB2 y RB3, el esquema electrónico se muestra en la figura 1.11.

Estimado lector, la importancia de este primer programa es ilustrar como se configuran los puertos a través del código asembler, es un programa básico de no más de 10 líneas, a medida que se avance en este texto los programas serán de mayor complejidad y de hasta 200 líneas, existen directivas, registros que aún no se han explicado, se explicarán durante el desarrollo de este proyecto, el código fue editado, ensamblado y compilado en el MPLAB, la simulación fue realizada en el PROTEUS, el código está comentado en las líneas más importantes, esto facilita la comprensión del mismo y permite realizar futuros cambios con éxito, se recomienda consultar con el datasheet del fabricante ante cualquier duda o consulta, cada aplicación será tratada con mucho detalle, pero no se puede abarcar todo sería algo ideal.



1 MANEJO DE PUERTOS

B) Esquema electrónico:

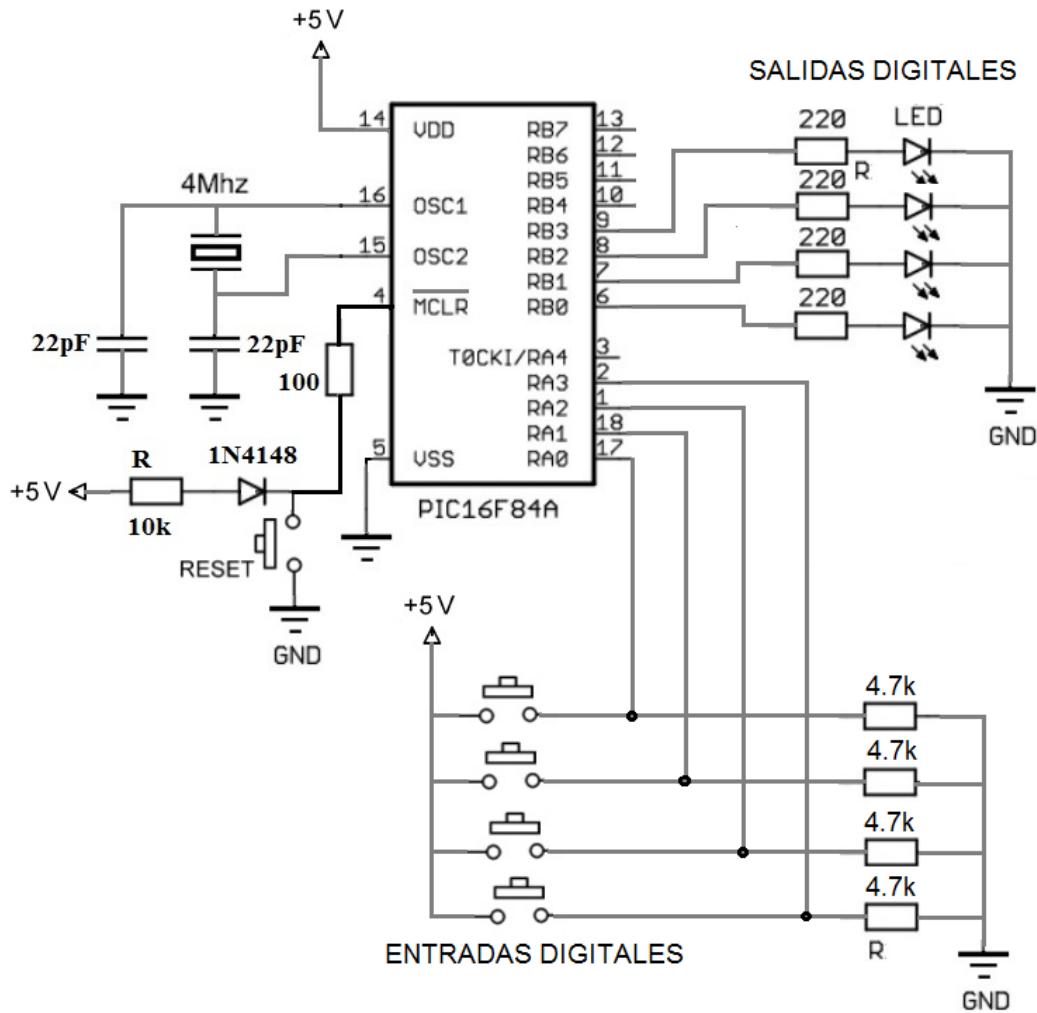


Figura 1.11. Circuito electrónico para el proyecto 1.

C) Instrucciones a utilizar:

bcf f,b

Bit Clear File

Esta instrucción coloca a cero un bit de un registro.

Ejemplo:

bcf ESTADO,3

Antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 11111111 b

Al ejecutarse la instrucción, el registro queda con el valor:

ESTADO = 11110111 b

**bsf f,b**

Bit Set File

Esta instrucción coloca a uno un bit de un registro.

Ejemplo:

bsf ESTADO,3

Antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 00000000 b

Al ejecutarse la instrucción, el registro queda con el valor:

ESTADO = 00001000 b

movlw k

Move l a w

Esta instrucción mueve un literal al registro w, se entiende por literal a un número en formato binario o hexadecimal, este número va desde 0 hasta un máximo de 255, puesto que el registro w tiene un ancho de 8 bits.

Ejemplo:

movlw d'10'

Antes de ejecutar la instrucción el registro w vale por ejemplo:

w = 0 en decimal

Al ejecutarse la instrucción, el registro w queda con el valor:

w = 10 en decimal

movwf f

Mover w a f

Esta instrucción mueve el valor de w al registro f.

Ejemplo:

movwf ESTADO

Antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 0 en decimal y w = 10 en decimal

Al ejecutarse la instrucción, el registro ESTADO queda con el valor:

ESTADO = 10 en decimal



1 MANEJO DE PUERTOS

movf f,d

Mover el contenido de f a f o de f a w.

Esta instrucción mueve el contenido del registro f a f, si el bit d vale 1, el bit d es el bit destino, indica a donde se mueve el contenido, si este bit vale 0, mueve el contenido de f a w.

Ejemplo:

movf ESTADO,0

Antes de ejecutar la instrucción los registros ESTADO y w valen:

ESTADO = 23 en decimal y w = 12 en decimal

Al ejecutarse la instrucción, los registros ESTADO y w quedan con el valor:

ESTADO = 23 en decimal y w = 23 en decimal

Como se observa el valor de w ha cambiado de 12 a 23, la instrucción ha copiado el valor de ESTADO osea 23 a w. El bit destino es igual a 0.

movf ESTADO,1

Antes de ejecutar la instrucción los registros ESTADO y w valen:

ESTADO = 23 en decimal y w = 12 en decimal

Al ejecutarse la instrucción, los registros ESTADO y w quedan con el valor:

ESTADO = 23 en decimal y w = 12 en decimal

Como se observa el valor de w no ha cambiado, la instrucción ha copiado el valor de ESTADO a ESTADO, no tiene sentido, pero es posible. El bit destino es igual a 1.

goto k

Saltar a la etiqueta k

Esta instrucción realiza un salto a una etiqueta determinada, también significa **ir a**, la etiqueta en realidad es una dirección en memoria del programa, el alcance máximo del salto es de 2k.

Ejemplo:

inicio nop

goto inicio

Salta a la etiqueta inicio



D) Código del programa:

```
;-----  
;Nombre del programa: lect.asm  
;Autor: Ing. David Apaza Condori  
;Ensamblador: MPLAB V8.14  
;  
;  
__config H'3FF1'           ;palabra de configuración  
#include "p16f84.inc"      ;llamar a las librerías  
  
ORG      0x00  
        bsf    STATUS,5      ;ir al banco 1  
        movlw  0x0F  
        movwf  TRISA         ;puerto A como entrada  
        movlw  0x00  
        movwf  TRISB         ;puerto B como salida  
        bcf    STATUS,5      ;ir al banco 0  
        clrf   PORTB  
  
inicio   movf   PORTA,0     ;leer el puerto A  
        movwf  PORTB         ;escribir en el puerto B  
        goto   inicio         ;ir a inicio  
  
END      ;fin de programa  
;
```



1 MANEJO DE PUERTOS

E) Simulación del programa:

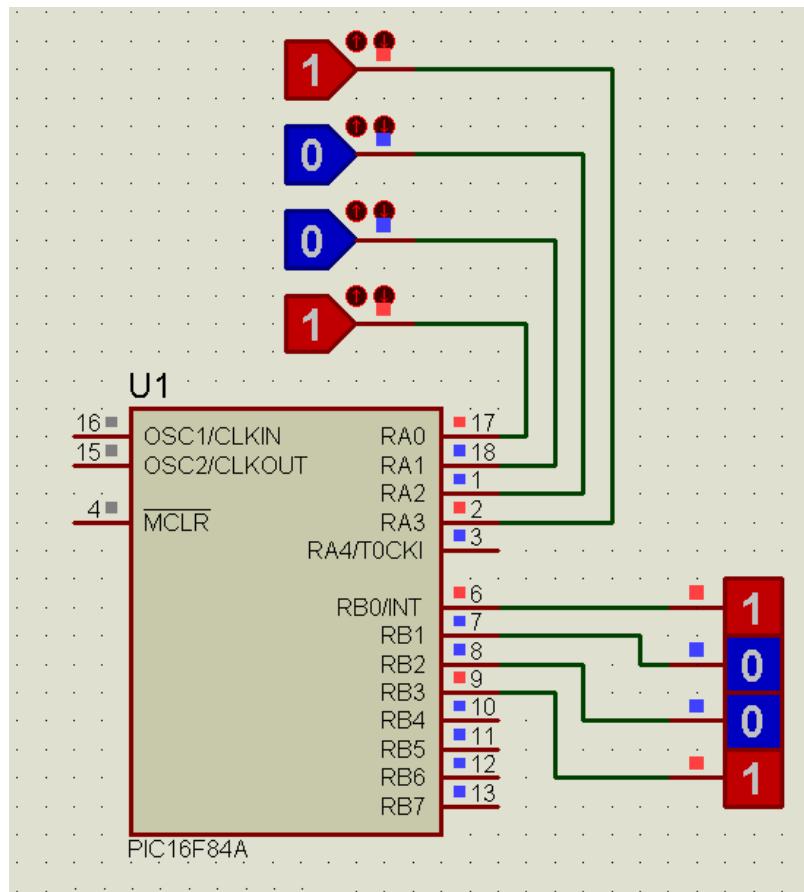


Figura 1.12. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.

F) Registros STATUS y CONFIGURATION WORD:

Hay recursos y opciones en la palabra de configuración que el programador debe activar o desactivar, de acuerdo a los requerimientos de la aplicación y solo cuando se programa el PIC, para cambiar el estado de estos recursos se tiene que reprogramar el PIC, por ejemplo: el perro guardián, el temporizador de encendido, se tiene en total 4 elementos programables que se encuentran en la llamada palabra de configuración.

La palabra de configuración se encuentra en la dirección de memoria de programa 2007 h, solamente se puede acceder a este espacio durante la programación del dispositivo.



CONFIGURATION WORD

R/P-u													
CP	PWRTE	WDTE	FOSC1	FOSC0									

bit13

bit0

CP, bit 4 a 13: Bits de protección de código

- 1: Protección de código deshabilitada
- 0: Memoria protegida por código

PWRTE, bit 3: Bit para habilitar el Power-up Timer o temporizador de encendido

- 1: Power-up timer deshabilitado
- 0: Power-up Timer habilitado

WDTE, bit 2: Bit para habilitar el Watchdog timer o temporizador perro guardián

- 1: WDT habilitado
- 0: WDT deshabilitado

FOSC1, FOSC0, bit 0 y 1: Selección del oscilador

- 00: Oscilador LP
- 01: Oscilador XT
- 10: Oscilador HS
- 11: Oscilador RC

Activando CP, Code Protection, tendremos la garantía de que el código escrito en el PIC no pueda ser leído por otra persona, para que no se copie, modifique, etc. Activando el bit PWRTE, conseguimos que se genere un retardo en la inicialización del microcontrolador. Esto se utiliza para que la tensión se estabilice, por lo que se recomienda su uso.

El “perro guardián” del PIC se configura aquí, con esto el PIC tiene la capacidad de reiniciarse internamente. Es útil ante problemas que impidan el funcionamiento del programa del PIC, como un bucle infinito, el WDT lo sacará de ese estado reiniciando el PIC.



1 MANEJO DE PUERTOS

STATUS REGISTER

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C

bit 7

bit 0

Bit 7 IRP: Selección de banco en direccionamiento indirecto

0 = Banco 0 - 1 (00H - FFH)

1 = Banco 2 - 3 (100H - 1FFH)

Bit 6-5 RP1-RP0: Selección de banco en direccionamiento directo

0 0 = Banco 0 (00H - 7FH)

0 1 = Banco 1 (80H - FFH)

1 0 = Banco 2 (100H - 17FH)

1 1 = Banco 3 (180H - 1FFH)

Bit 4 TO: Time Out

1 = Despues de la conexión de alimentación o al ejecutarse clrwdt
o sleep

0 = Cuando se produce desbordamiento del watchdog

Bit 3 PD: Power Down

1 = Luego de energizar al uC o al ejecutar la instrucción clrwdt
0 = Al ejecutar la instrucción sleep

Bit 2 Z: Indicador de cero

1 = El resultado de una operación lógica o aritmética es cero
0 = El resultado es diferente de cero

Bit 1 DC: Indicador de acarreo en el cuarto bit de menos peso

1 = Acarreo en la suma

0 = No ha ocurrido acarreo

Bit 0 C: Indicador de acarreo en el octavo bit

1 = Acarreo en la suma

0 = No existe acarreo



Proyecto 2: Compuerta lógica AND

A) Planteamiento del programa:

El programa consiste en implementar una compuerta lógica AND de dos entradas y una salida, aplicando lógica programada, las entradas y salida se pueden asignar a cualquier puerto y pin, vamos a utilizar el puerto A para las entradas y el puerto B para la salida, el tiempo de respuesta de esta compuerta está en el orden de los microsegundos, depende de la frecuencia del cristal.

La compuerta AND da como respuesta la multiplicación lógica de sus entradas, es decir que la salida es 1, si la entrada A y la entrada B están ambas en 1, de otra manera, la salida es 0, estas condiciones también son especificadas en la tabla de verdad correspondiente a la compuerta AND. La tabla y el símbolo para esta compuerta se muestra en la figura 1.13.

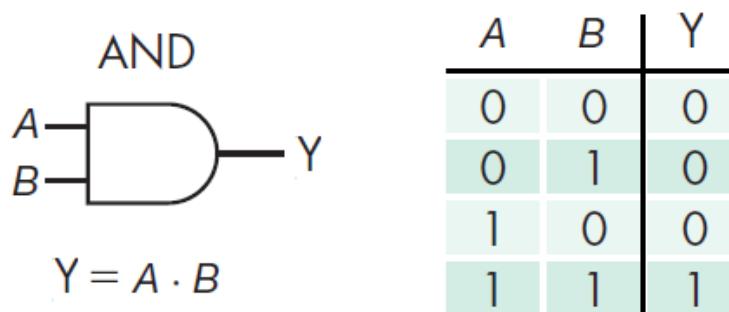


Figura 1.13. Compuerta lógica AND, símbolo y tabla de verdad.

Se utilizará el puerto A para las entradas, éste debe ser configurado como entrada y los pines asignados son: RA0, RA1, entrada 0 y entrada 1 respectivamente, se utilizará el puerto B para la salida, éste debe ser configurado como salida y el pin asignado es el RB0, el esquema electrónico se muestra en la figura 1.14.

Estimado lector, la importancia de este segundo programa es ilustrar como se implementa una compuerta lógica a través de un programa, no es óptimo hacer esto con un PIC, puesto que el PIC es un dispositivo de mayor capacidad, pero si es posible implementar una compuerta lógica, también es posible implementar funciones lógicas de mayor complejidad.



1 MANEJO DE PUERTOS

B) Esquema electrónico:

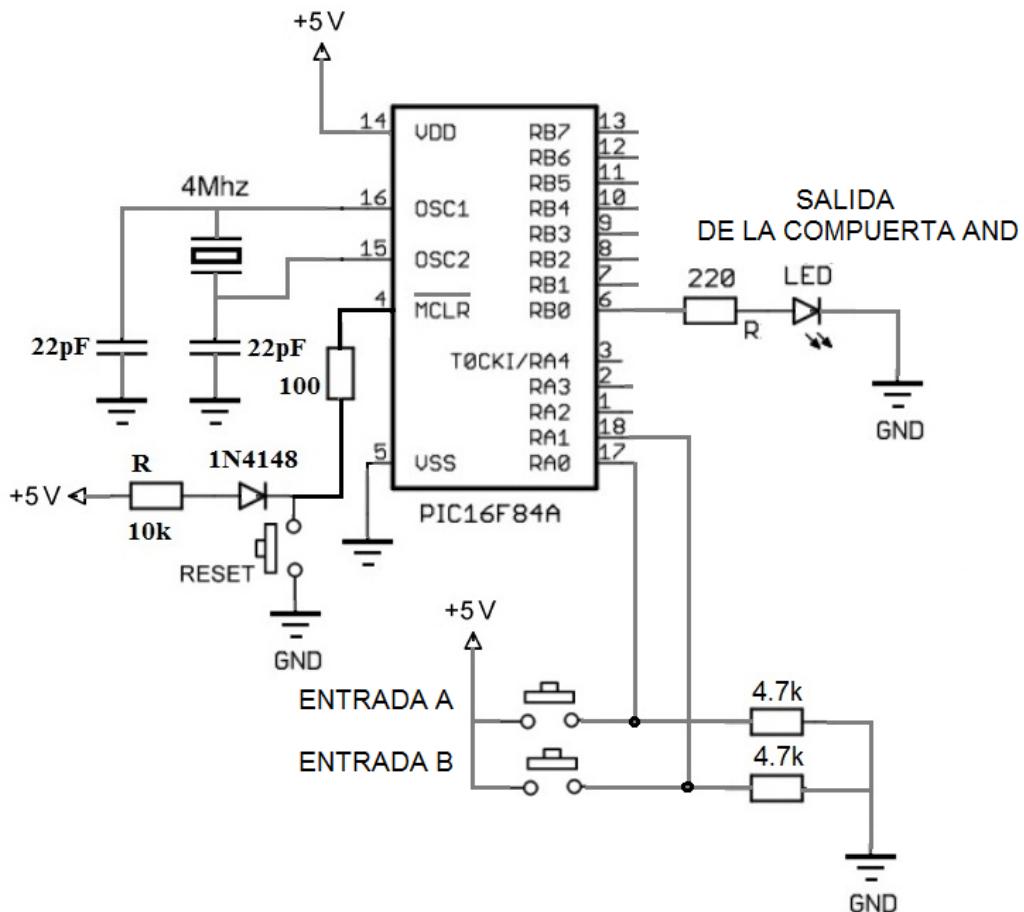


Figura 1.14. Circuito electrónico para el proyecto 2.

C) Instrucciones a utilizar:

btfsc f,b

bit test file skip clear

Si el bit número b del registro f está en 0, la instrucción que sigue a ésta se ignora y se trata como un NOP (skip). En este caso, y sólo en este caso, la instrucción precisa dos ciclos para ejecutarse.

Ejemplo:

```
btfsc ESTADO,3  
goto izquierda  
goto derecha
```



Si antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 11111111 b

Al ejecutarse la instrucción el programa no salta y pasa a ejecutar goto izquierda, esto debido a que no se cumplió la condición de: salta si el bit 3 del registro ESTADO es 0, ese bit no es 0 es 1. Veamos otra situación.

Si antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 00000000 b

Al ejecutarse la instrucción el programa si salta y pasa a ejecutar goto derecha, esto debido a que si se cumplió la condición de: salta si el bit 3 del registro ESTADO es 0 y efectivamente ese bit es 0.

btfs f,b

bit test file skip set

Si el bit número b del registro f está en 1, la instrucción que sigue a ésta se ignora y se trata como un NOP (skip). En este caso, y sólo en este caso, la instrucción precisa dos ciclos para ejecutarse.

Ejemplo:

```
btfs ESTADO,3
goto izquierda
goto derecha
```

Si antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 00000000 b

Al ejecutarse la instrucción el programa no salta y pasa a ejecutar goto izquierda, esto debido a que no se cumplió la condición de: salta si el bit 3 del registro ESTADO es 1, ese bit no es 1 es 0. Veamos otra situación.

Si antes de ejecutar la instrucción el registro ESTADO vale:

ESTADO = 11111111 b

Al ejecutarse la instrucción el programa si salta y pasa a ejecutar goto derecha, esto debido a que si se cumplió la condición de: salta si el bit 3 del registro ESTADO es 1 y efectivamente ese bit es 1.



1 MANEJO DE PUERTOS

D) Código del programa:

```
;  
;;Nombre del programa: and.asm  
;Autor: Ing. David Apaza Condori  
;Ensamblador: MPLAB V8.14  
;  
;  
_config H'3FF1'           ;palabra de configuración  
#include"p16F84.inc"      ;llamar a las librerias  
ORG    0x00  
        bsf      STATUS,5      ;ir al banco 1  
        movlw   b'00000011'  
        movwf   TRISA          ;RA0 y RA1 como entrada  
        movlw   b'00000000'  
        movwf   TRISB          ;RB0 como salida  
        bcf      STATUS,5      ;ir al banco 0  
        clrf    PORTB          ;limpiar el PORTB  
inicio  btfss  PORTA,0      ;verifica RA0 es 0 o 1  
        goto   lab_01  
        btfss  PORTA,1      ;verifica RA1 es 0 o 1  
        goto   lab_01  
        bsf    PORTB,0        ;activa la salida  
        goto   inicio  
lab_01  bcf    PORTB,0      ;apaga la salida  
        goto   inicio  
END  
;
```



El siguiente código de programa es para una compuerta or:

```
;-----  
;Nombre del programa: or.asm  
Autor: Ing. David Apaza Condori  
Ensamblador: MPLAB V8.14  
;  
_____  
__config H'3FF1'           ;palabra de configuración  
#include"p16F84.inc"       ;llamar a las librerias  
ORG      0x00  
        bsf    STATUS,5      ;ir al banco 1  
        movlw  b'00000011'  
        movwf  TRISA         ;RA0 y RA1 como entrada  
        movlw  b'00000000'  
        movwf  TRISB         ;RB0 como salida  
        bcf    STATUS,5      ;ir al banco 0  
        clrf   PORTB         ;limpiar el PORTB  
inicio   btfsc  PORTA,0    ;verifica RA0 es 0 o 1  
        goto   lab_01  
        btfsc  PORTA,1    ;verifica RA1 es 0 o 1  
        goto   lab_01  
        bcf    PORTB,0      ;apaga la salida  
        goto   inicio  
lab_01   bsf    PORTB,0    ;activa la salida  
        goto   inicio  
END  
-----;
```



E) Simulación del programa:

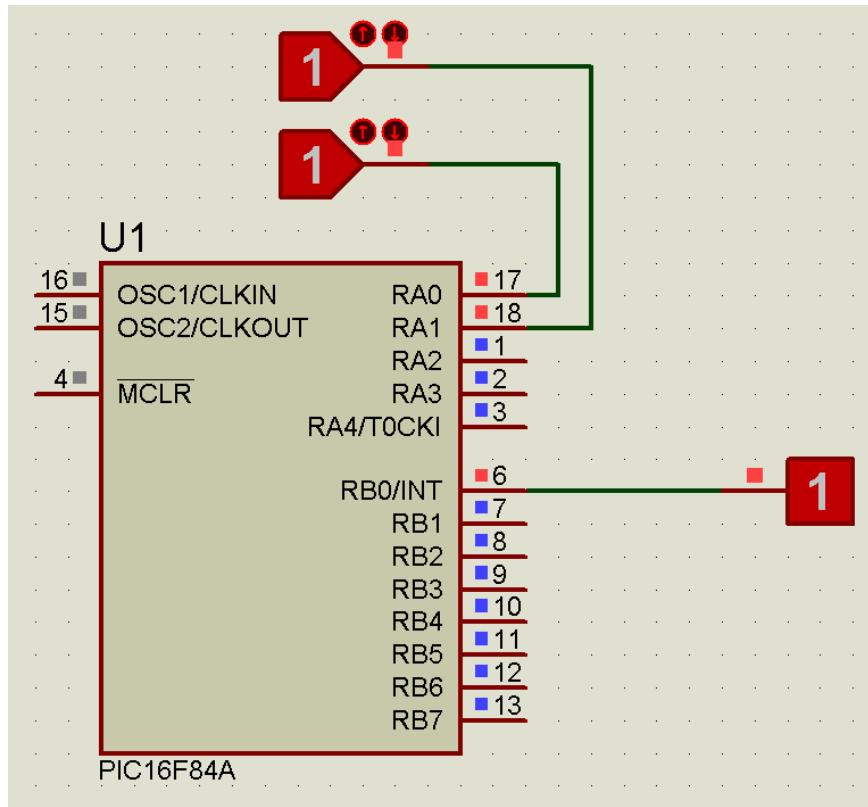


Figura 1.15. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.

Proyecto 3: Decodificador de tres bits

A) Planteamiento del programa:

El programa consiste en implementar un decodificador de tres bits, aplicando lógica programada, las entradas y salidas se pueden asignar a cualquier puerto y pin, vamos a utilizar el puerto A para las entradas y el puerto B para las salidas, el tiempo de respuesta del decodificador está en el orden de los microsegundos y depende de la frecuencia del cristal. El decodificador recibe un número en formato binario y da como respuesta el equivalente en formato decimal, normalmente se utiliza un display de 7 segmentos para visualizar la respuesta, éste puede ser de ánodo común o cátodo común, existen decodificadores en circuitos integrados como por ejemplo el 74LS47, tal como se muestra en la figura 1.16.

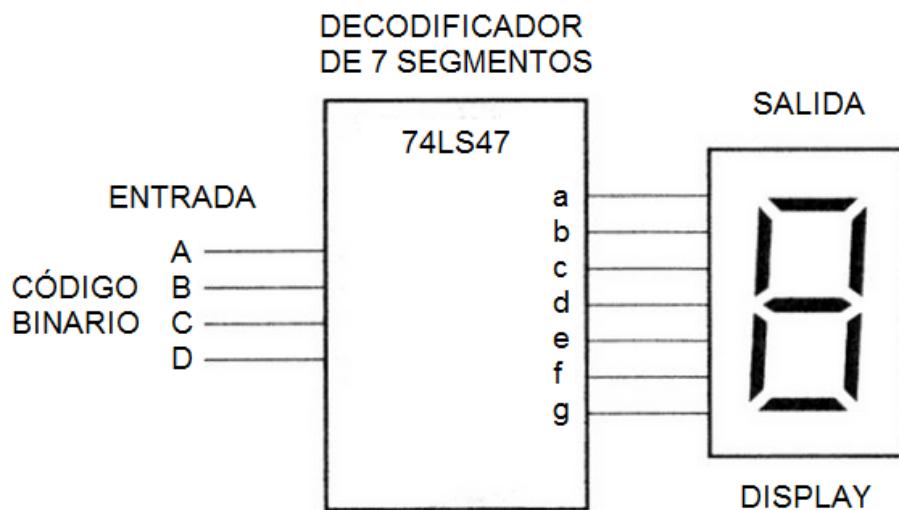


Figura 1.16. Representación del decodificador de 7 segmentos en circuito integrado 74LS47.

El circuito integrado 74LS47 recibe un código en formato binario y muestra en el display el número equivalente en formato decimal, por ejemplo: si la entrada es 000 en binario, el display muestra 0, si la entrada es 011 en binario, el display muestra 3, con 3 entradas el número máximo es 7, en la tabla 1.3 se muestra las equivalencias entre el formato binario y decimal.

Tabla 1.3. Equivalencias entre el formato binario y decimal.

ENTRADAS CÓDIGO BINARIO				SALIDAS DISPLAY							SALIDA EN DECIMAL
D	C	B	A	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	0	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9



1 MANEJO DE PUERTOS

Se utilizará el puerto A para las entradas, éste debe ser configurado como entrada y los pines asignados son: RA0, RA1 y RA2, entrada 0, entrada 1 y entrada 2 respectivamente, se utilizará el puerto B para las salidas, éste debe ser configurado como salida y los pines asignados son: RB1, RB2, RB3, RB4, RB5, RB6 y RB7, el esquema electrónico se muestra en la figura 1.17.

B) Esquema electrónico:

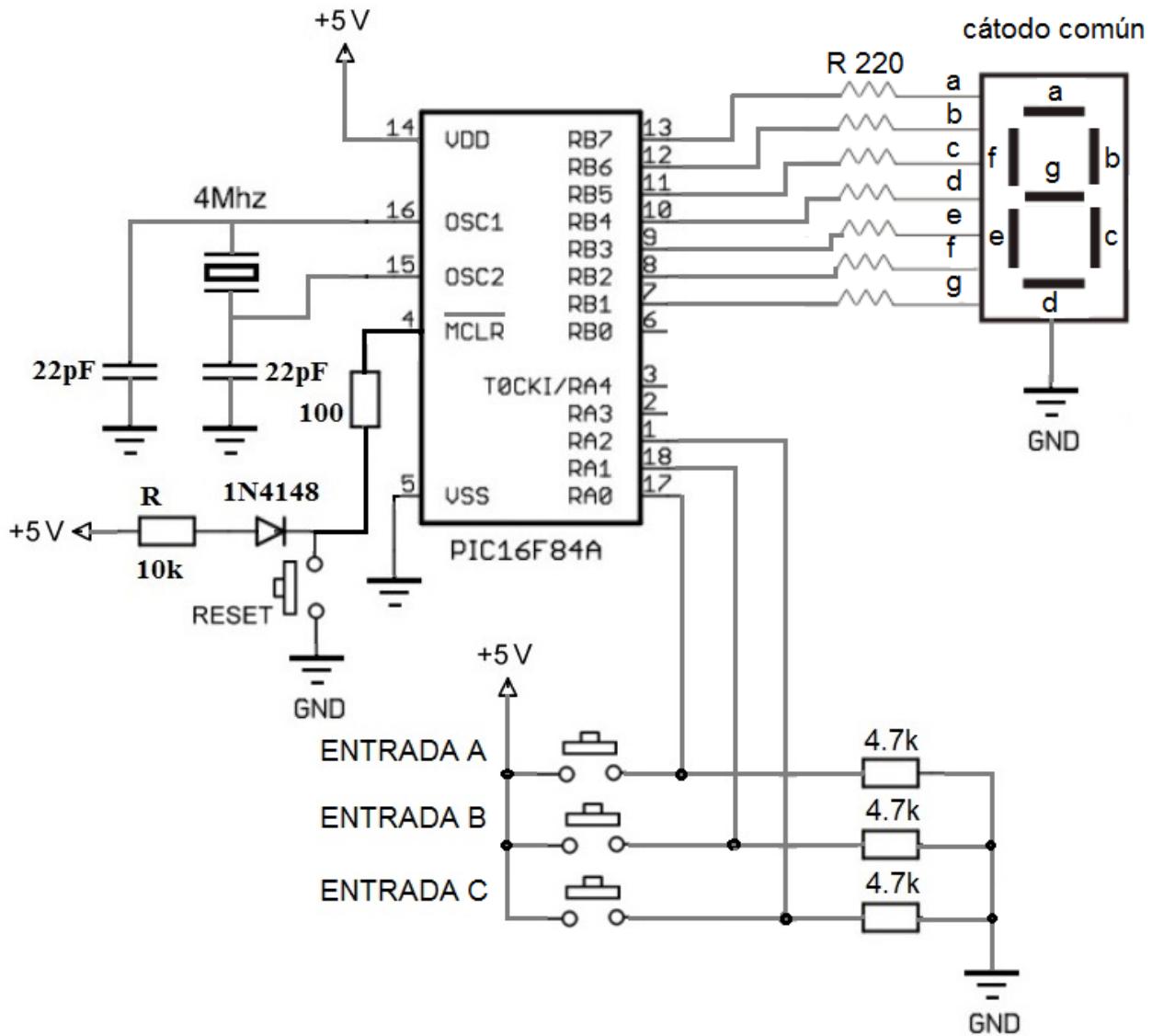


Figura 1.17. Circuito electrónico para el proyecto 3.



C) Instrucciones a utilizar:

La instrucción addwf: suma el valor de los registros w y f, por ejemplo:

addwf ESTADO,1

Si antes de ejecutar la instrucción $w = 12$ y $ESTADO = 24$, al ejecutarse la instrucción se suma el valor de w y ESTADO, el resultado es 36, este resultado se guarda en ESTADO, debido a que el bit destino es 1, finalmente $ESTADO = 36$.

La instrucción call: guarda la dirección de retorno en la pila y después llama a la subrutina situada en la dirección cargada en el PC, por ejemplo:

call	sumar
goto	inicio
sumar	addwf ESTADO,1
return	

La instrucción call llama a la subrutina sumar, salta a la etiqueta sumar y ejecuta todas las instrucciones de la subrutina hasta encontrar la instrucción return, que indica retorno de subrutina o fin de subrutina.

La instrucción retlw: cumple la misma función que return, con la diferencia que retlw retorna con un valor cargado en w, por ejemplo:

call	sumar
goto	inicio
sumar	addwf ESTADO,1
retlw	d'10'

Carga 10 en decimal a w y luego retorna de la subrutina.



1 MANEJO DE PUERTOS

D) Código del programa:

```
;-----  
;Nombre del programa: deco.asm  
;Autor: Ing. David Apaza Condori  
;Ensamblador: MPLAB V8.14  
;  
;  
_config H'3FF1'  
#include"p16F84.inc"  
  
ORG      0x00  
          bsf      STATUS, 5  
          movlw   0x07  
          movwf   TRISA  
          clrf    TRISB  
          bcf     STATUS, 5  
  
inicio   movf    PORTA, 0  
          andlw   0x07  
          call    tabla  
          movwf   PORTB  
          goto    inicio  
;  
;  
tabla    addwf  PCL, 1  
          retlw   b'01111111'  
          retlw   b'00001101'  
          retlw   b'10110111'  
          retlw   b'10011111'  
          retlw   b'11001101'  
          retlw   b'11011011'  
          retlw   b'11111011'  
          retlw   b'00001111'  
;  
;  
END  
;
```



E) Simulación del programa:

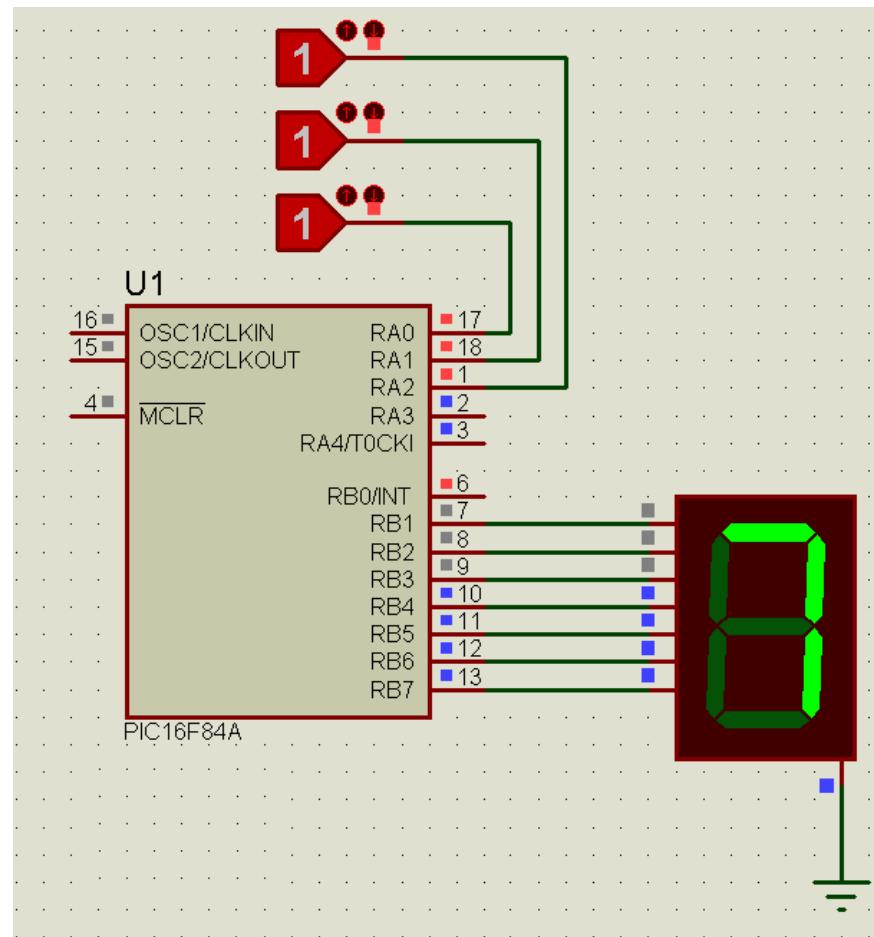


Figura 1.18. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



Actividades

A) Indicar en que formato está escrito cada número o si no pertenece a ningún formato:

0x15

- a) Hexadecimal b) Decimal c) Binario d) N.A.

b'10010111'

- a) Hexadecimal b) Decimal c) Binario d) N.A.

H'AB57'

- a) Hexadecimal b) Decimal c) Binario d) N.A.

d'123'

- a) Hexadecimal b) Decimal c) Binario d) N.A.

.37

- a) Hexadecimal b) Decimal c) Binario d) N.A.

37h

- a) Hexadecimal b) Decimal c) Binario d) N.A.

5

- a) Hexadecimal b) Decimal c) Binario d) N.A.

0xBGH

- a) Hexadecimal b) Decimal c) Binario d) N.A.

0h3A

- a) Hexadecimal b) Decimal c) Binario d) N.A.

0010

- a) Hexadecimal b) Decimal c) Binario d) N.A.

79d

- a) Hexadecimal b) Decimal c) Binario d) N.A.

b1001

- a) Hexadecimal b) Decimal c) Binario d) N.A.

0x00

- a) Hexadecimal b) Decimal c) Binario d) N.A.

101010b

- a) Hexadecimal b) Decimal c) Binario d) N.A.

**B) Responder las siguientes preguntas:**

1. ¿Qué entiende por TRISA?

2. Nombre 4 recursos especiales del PIC 16F84

3. ¿Cuáles son las instrucciones para mover datos?

4. ¿Cuál es la diferencia entre las instrucciones btfsc y btfss?

5. En el PIC 16F84 la memoria de datos (RAM) es mucho más grande que la memoria de programa verdadero o falso. ¿por qué?

6. ¿Qué registros se encuentran en el banco 1?

7. ¿Qué función cumple el registro de ESTADO?

8. ¿Para qué se utiliza la directiva EQU?

9. ¿Para qué se utiliza la directiva ORG?

10. ¿Para qué se utiliza la directiva INCLUDE?



1 MANEJO DE PUERTOS

C) Para el siguiente programa identificar los errores de sintaxis y corregir.

```
;  
-----  
_config H'3FF1'  
#include"p16F84.inc"  
ORG 0x00  
    bsf STATUS.5      ;-----  
    movlw b'00000011'  ;-----  
    movwf TRISA       ;-----  
    movwl b'00000000'  ;-----  
    movwf TRISB       ;-----  
    bzf STATUS,5      ;-----  
    clrf PORTB        ;-----  
inicio  btfs PORTA,0   ;-----  
        goto lab_01    ;-----  
        btfss PORTA,1  ;-----  
        goto lab_01    ;-----  
        bsf PORTB.0    ;-----  
        goto inicio     ;-----  
lab_01  bcf PORTB,0   ;-----  
        goto inicio     ;-----  
  
END  
;  
-----
```

D) Para las siguientes instrucciones indicar que hace cada instrucción.

goto sumar

```
-----
```

btfsc PORTB,1

```
-----
```

call parar

```
-----
```

bsf PORTA,0

```
-----
```

bcf dato01,2

```
-----
```



movlw b'00000011'

movwf TRISA

bsf STATUS,5

clrf PORTB

btfss PORTA,0

bcf PORTA,3





1 MANEJO DE PUERTOS



CAPÍTULO 2

TEMPORIZADORES

CONTENIDO DEL CAPÍTULO

- 1. Introducción
- 2. Funcionamiento del TMR0
 - 2.1. Estructura general del TMR0
 - 2.2. Entrada de reloj del módulo TMR0
 - 2.3. El prescaler
 - 2.4. El registro TMR0
 - 2.5. Sincronización del TMR0
- 3. Cálculo de temporizaciones
- Proyecto 4: El oscilador
- Proyecto 5: Semáforo electrónico
- Proyecto 6: Temporizador de 0 a 9
- Proyecto 7: Temporizador de 00 a 99

APRENDEREMOS A:

- Conocer el funcionamiento del módulo TMR0 y sus aplicaciones
- Identificar los registros y sus bits que permiten configurar el módulo TMR0
- Programar, configurar y aplicar el módulo TMR0
- Calcular retardos de tiempo en base al ciclo de instrucción
- Utilizar las instrucciones de llamada a subrutina, salto y comparación





2 TEMPORIZADORES

1. Introducción

El temporizador TMR0 tiene una amplia gama de aplicaciones en la práctica. Sólo unos pocos programas no lo utilizan de alguna forma. Es muy conveniente y fácil de utilizar en programas o subrutinas para generar pulsos de duración arbitraria, en medir tiempo o en contar los pulsos externos (eventos) casi sin limitaciones. El módulo del temporizador TMR0 es un temporizador / contador de 8 bits con las siguientes características:

1. Temporizador / contador de 8 bits
2. Pre-escalador de 8 bits (lo comparte con el temporizador perro guardián)
3. Fuente de reloj interna o externa programable
4. Generación de interrupción por desbordamiento
5. Selección del flanco de reloj externo programable

En la figura 2.1 se muestra el esquema del temporizador TMR0 con todos los bits que determinan su funcionamiento. Estos bits se almacenan en el registro OPTION_REG. A menudo al utilizar un microcontrolador nos encontramos con la necesidad de contar o generar eventos cada cierto tiempo. Para ayudar en este tipo de tareas, es habitual que los microcontroladores dispongan de circuitos internos para ello. Este circuito, es comúnmente denominado Timer / Counter (Temporizador / Contador) aunque también por motivos históricos es habitual encontrarlo con el nombre de RTCC (Real Time Clock Counter).

Nos vamos a centrar en el Timer / Counter de 8 bits habitual en los microcontroladores PIC 16F84, denominado en la hoja de especificaciones como módulo TMR0 (en otros modelos es posible encontrar adicionales módulos de 8 ó 16 bits cuyo funcionamiento básico es el mismo). Antes de explicar el funcionamiento y uso del TMR0, vamos de definir los siguientes conceptos para evitar confusiones:

Frecuencia de oscilación (Fosc): Frecuencia de trabajo externa del PIC (un cristal de cuarzo, un resonador, etc.).

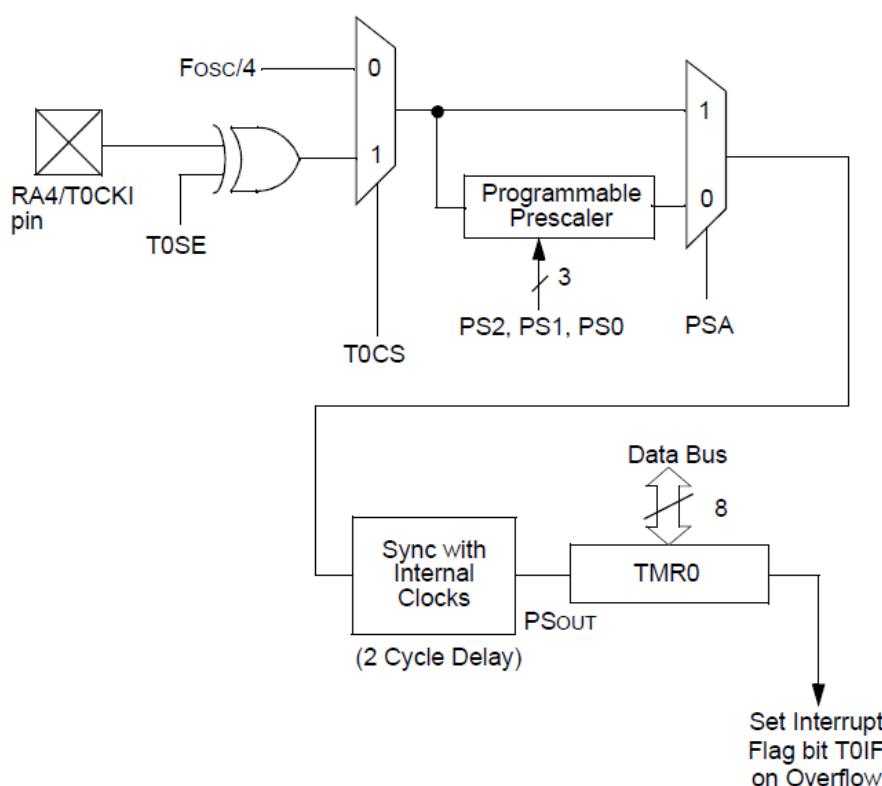


Frecuencia interna de instrucciones (Fins): Frecuencia del reloj interno de instrucciones generada a partir de la frecuencia de oscilación externa. Para los microcontroladores PIC esta frecuencia no coincide con Fosc, ya que para mantener la compatibilidad con los diseños originales es necesario dividirla por cuatro:

$$\text{Fins} = \text{Fosc} / 4$$

Periodo de instrucción (Tins): Periodo de la frecuencia interna de instrucciones (Fins), también se le conoce como ciclo de instrucción y se mide en segundos, su valor se obtiene de la siguiente forma:

$$\text{Tins} = 1 / \text{Fins}$$



- Note**
- 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
 - 2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

Figura 2.1. Representación del módulo TMR0 en bloques, se observa el prescaler, el registro TMR0, los bits de control y habilitación contenidos en el registro OPTION_REG.



2 TEMPORIZADORES

2. Funcionamiento del TMR0

El TMR0 es un dispositivo que puede funcionar conceptualmente de dos formas: como contador de pulsos externos o como temporizador para calcular intervalos de tiempo. En el caso que dicha señal provenga del reloj interno de instrucciones (Fins), el TMR0 se utiliza para generar interrupciones periódicas a través de una cuenta programada. Ya que conocemos la frecuencia de funcionamiento y en base a un valor cargado en el contador del TMR0 (lo veremos más adelante) podemos temporizar eventos.

En el caso que dicha señal sea de una fuente externa al microcontrolador (Fext), es especialmente útil para contar el número de pulsos que dicha señal genera en el tiempo ya que cada pulso de dicha señal incrementa el TMR0.

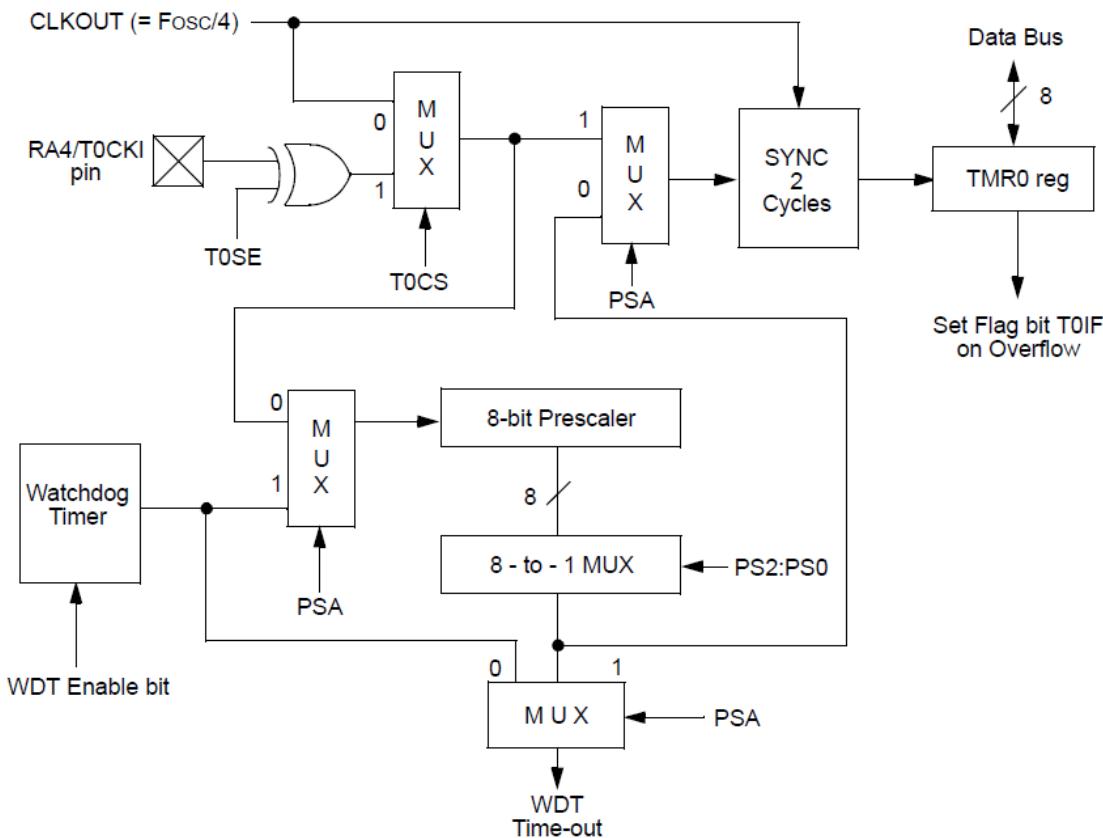
Ejemplo: Queremos medir el intervalo de tiempo entre los pulsos del cardiograma de un paciente.

Usando el TMR0 como temporizador, podemos contar por ejemplo: períodos de 1ms, acumulándolos en un registro auxiliar. Cada vez que ocurra un pulso del electrocardiograma, podemos consultar el registro auxiliar y ver cuantos incrementos han sucedido desde el último pulso y sabiendo que el temporizador cuenta períodos de 1ms, calcular el tiempo transcurrido, una vez conocido el tiempo, iniciamos la cuenta de los incrementos desde 0 para el siguiente pulso.

2.1. Estructura general del TMR0

El esquema del TMR0 corresponde a la figura 2.2, donde destacamos lo siguiente:

1. La entrada de reloj (patilla RA4/T0CKI o el reloj interno de instrucciones)
2. Un circuito divisor de frecuencias programable o prescaler
3. Un registro contador TMR0 de 8 bits
4. Varios registros de control del TMR0: OPTION_REG e INTCON



Note: T0CS, T0SE, PSA, PS2:PS0 are (OPTION_REG<5:0>).

Figura 2.2. Representación del módulo TMR0 en bloques, se observa el prescaler, el registro WDT, los respectivos multiplexores.

2.2. Entrada de reloj del módulo TMR0

Como anteriormente se indicó, la señal de reloj base de entrada al módulo TMR0 se puede obtener de dos maneras: con un oscilador externo a través de la patilla RA4/T0CKI Fext, o bien con el oscilador interno de instrucciones Fins (recordemos que es un cuarto de la frecuencia de oscilación del microcontrolador). También podemos indicar si el disparo se realizará por flanco ascendente o flanco descendente de la señal. La configuración la podemos realizar con los siguientes bits de control:

T0SC (Timer0 Clock Select) en el registro OPTION_REG bit 5: Indica el origen del reloj del contador: oscilador interno o señal externa.

T0SE (Timer0 Set Edge) en el registro OPTION_REG bit 4: Cuando se selecciona señal externa, indica el flanco activo que se usará.



2 TEMPORIZADORES

2.3. El prescaler

El prescaler es un circuito que permite modificar la frecuencia del reloj de entrada del TMR0, dividiendo esta y generando una nueva señal de menor frecuencia a la salida que será la señal de reloj de entrada al registro TMR0. El prescaler es una facilidad para cuando la señal de entrada es demasiado rápida para nuestros propósitos y necesitamos ralentizarla.

Nota: El prescaler también se puede utilizar con el registro WDT (Watch Dog Timer) del PIC, pero en este caso recibe el nombre de postscaler ya que se usa a la salida del WDT, no pudiendo estar asignado a la vez al TMR0 o al WDT.

Nota: El prescaler del registro TMR0 es configurado a través de 4 bits del registro OPTION_REG, y permite dividir la frecuencia de una señal por 1, 2, 4, 8, 16, 32, 64, 128 o 256. En caso de utilizar un divisor por 1, la señal de salida es la de entrada sin ningún cambio.



¿ Sabías que . . . ?

El prescaler del TMR0 indica de cuanto en cuanto se lleva el conteo, puede ser de 2 en 2, de 4 en 4, de 256 en 256, son en total 8 opciones y se configura en el registro OPTION_REG.

Por ejemplo, si usamos como oscilador externo del PIC un cristal de 4Mhz, entonces el reloj interno de instrucciones funciona a $F_{ins} = 4\text{Mhz} / 4 = 1\text{Mhz}$. Si el TMR0 usa la señal del reloj interno y la pasamos por el prescaler configurado para una división por 4, la señal a la salida del prescaler será $F_{presc} = 250\text{ KHz}$. La configuración se realiza con los siguientes bits de control:

PSA (Post Scaler Assignment) en el registro OPTION_REG bit 3: Indica si el postscaler es asignado al WDT (1) o al TMR0 (0).

PS2:0 en el registro OPTION_REG bits 0, 1 y 2: Indican el valor del divisor a utilizar en el postscaler (consultar datasheet para los valores).



No debemos confundir la frecuencia de trabajo base del TMR0, con la frecuencia de trabajo del registro TMR0: la primera es la frecuencia de entrada utilizada por el módulo, mientras que la segunda es dicha frecuencia dividida por el prescaler y que es el reloj del registro TMR0.

OPTION REGISTER (ADDRESS 81h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

RBPU – PORTB Pull-up enable bit (resistencia Pull Up del puerto PORTB)

0: Resistencias pull-up del puerto PORTB están deshabilitadas

1: Pines del puerto PORTB pueden estar conectados a las resistencias pull-up

INTEDG – Interrupt Edge Select bit (bit selector de flanco activo de la interrupción externa)

0: Interrupción por flanco ascendente en el pin INT (0-1)

1: Interrupción por flanco descendente en el pin INT (1-0)

T0CS – TMR0 Clock Select bit (bit selector de tipo de reloj para el Timer0)

0: Los pulsos se llevan a la entrada del temporizador/contador Timer0 por el pin RA4

1: El temporizador utiliza los pulsos de reloj internos (Fosc/4)

T0SE – TMR0 Source Edge Select bit (bit selector de tipo de flanco)

0: Incrementa en flanco descendente en el pin TMR0

1: Incrementa en flanco ascendente en el pin TMR0

PSA – Prescaler Assignment bit (bit de asignación del pre-escalador)

0: Pre-escalador se le asigna al WDT

1: Pre-escalador se le asigna al temporizador/contador Timer0

PS2, PS1, PS0 – Prescaler Rate Select bit (bit selector del valor del divisor de frecuencias)

El valor del divisor de frecuencias se ajusta al combinar estos bits. Como se muestra en la tabla 2.1, la misma combinación de bits proporciona los diferentes valores del divisor de frecuencias para el temporizador/contador y el temporizador perro guardián, respectivamente.



2 TEMPORIZADORES

Tabla 2.1. Diferentes valores del divisor de frecuencias para el temporizador / contador.

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

2.4. El registro TMR0

El registro TMR0 es el corazón del módulo TMR0. Es un registro contador de 8 bits (podemos contar hasta $2^8 = 256$ valores, entre 0 y 255) que incrementa automáticamente su contenido en cada oscilación de su señal de reloj Ftmr0. Cuando el valor del registro TMR0 pasa de 255 a 0, se produce una interrupción por desbordamiento u “overflow”. El PIC indica esta situación activando el flag de interrupción por desbordamiento T0IF (Timer0 Interrupt Flag) del registro INTCON. Para ejecutar la rutina de interrupción además deben estar habilitadas tanto las interrupciones globales como la específica para el TMR0:

1. Interrupciones globales activadas: bit GIE (Global Interrupt Enable) de INTCON bit 7 activo
2. Mascara de interrupción de TMR0 activada: bit T0IE (Timer0 Interrupt Enable) de INTCON bit 5 activo
3. Ocurrió un desbordamiento en el TMR0: bit T0IF (Timer0 Interupt Enable) de INTCON bit 2 activo

El proceso en general es muy similar al que se sigue con cualquier otra interrupción: si se dan las tres condiciones el PIC comienza a ejecutar la rutina de servicio de interrupción, deshabilitando estas (GIE = 0). La rutina de interrupción debería comprobar el flag T0IF para determinar el origen de la interrupción y, si ocurrió una interrupción por desbordamiento, borrar el flag T0IF (debemos



hacerlo nosotros pues no es automático), para evitar que el PIC vuelva a la rutina de interrupción cuando las interrupciones sean de nuevo habilitadas. A la salida de la rutina de interrupción con la instrucción “retfie” el PIC habilita estas de nuevo (GIE = 1).

El registro TMR0 continúa incrementándose siempre según dicta su reloj, por lo que si se ha producido un desbordamiento, el registro seguirá contando desde 0. Si interesa que la cuenta comience desde un valor predeterminado, nosotros debemos precargar el TMR0 con dicho valor una vez detectado el desbordamiento. Esto se realiza normalmente dentro de la rutina de interrupción.



¿ Sabías que . . . ?

Una interrupción es un evento externo aleatorio que detiene la ejecución del programa principal y da inicio a un subprograma que se encuentra en la dirección 0x04.

Observaciones:

Si el microcontrolador está dormido (mediante una instrucción SLEEP) y se utiliza como señal de reloj del TMR0 el reloj interno de instrucciones, el TMR0 estará desactivado y no se incrementará el contador TMR0 ya que el reloj interno se haya detenido. Por tanto jamás se producirá ninguna interrupción por desbordamiento que permita salir del estado SLEEP.

Si la fuente de la señal de reloj del módulo TMR0 es externa, si se puede producir interrupción por desbordamiento, ya que aunque el TMR0 este desactivado, el contador no depende activamente del reloj interno (parado) del microcontrolador sino que se incrementa en cada pulso de la señal externa. En este caso si se puede salir del estado SLEEP a través de la interrupción.



2 TEMPORIZADORES

2.5. Sincronización del TMR0

Esta modificación del TMR0 podría plantear algunos problemas si se modifica al mismo tiempo que se incrementa. El PIC resuelve esto internamente, mediante un mecanismo de sincronización de la señal de reloj del TMR0 cuando se escribe un dato en el TMR0, se usa un registro de desplazamiento de 2 bits a la salida del reloj del TMR0 de forma que produce un retraso de $2 * Fosc / 4$ en la señal que incrementa el contador (2us con un cristal de 4MHz; 1us con 8MHz).

Esto se traduce en que cuando se escribe un dato en el TMR0, se conecta al reloj interno introduciendo un retraso de 2 cuentas, en los cuales no ocurre nada en el TMR0 y ese tiempo es aprovechado para escribir el dato escrito. Estas dos cuentas de retraso nos permiten escribir y leer sin interferencias con la cuenta actual del contador. Una vez asignado el prescaler al TMR0, una instrucción que escriba el TMR0, provocará el borrado del prescaler y sincronizará el reloj con el TMR0.

3. Cálculo de temporizaciones

Supongamos que el módulo TMR0 se va a alimentar con una señal de frecuencia (puede ser de una fuente externa o el oscilador interno del microcontrolador), el registro interno TMR0 cuenta los pulsos de la señal de frecuencia interna o externa, para generar retardos normalmente se utiliza la señal de frecuencia interna, es decir $Fins$, que es la frecuencia de oscilación entre 4, para un cristal de 4MHz, la $Fosc = 4MHz$, la $Fins = 1MHz$.

El periodo de instrucción es $Tins = 1us$, la inversa de la frecuencia $Fins$, cada pulso representa 1us, cada instrucción se ejecuta en 1us excepto las instrucciones de salto que necesitan dos ciclos, al $Tins$ también se le conoce como ciclo de instrucción o periodo de instrucción, ahora bien tenemos una señal de frecuencia interna $Fins$ que es un tren de pulsos y cada pulso tiene un periodo o duración de 1us, este tiempo depende directamente de la $Fosc$, el registro TMR0 cuenta estos pulsos, lo hace internamente, el registro TMR0 es de 8 bits, su capacidad va



desde 0 hasta 255, puede contar hasta 256 pulsos, cuando el valor del TMR0 llega a 255 y se incrementa en 1 pasa a 0, si su valor pasa a 0 el TMR0 se ha desbordado o reiniciado, se activa la interrupción correspondiente.

Veamos, si el TMR0 cuenta los pulsos de Fins, si el TMR0 inicia en 0 y se ha desbordado por primera vez, el tiempo que ha pasado es de 256 pulsos, cada pulso es 1us, entonces el tiempo total ocurrido es de 256us, este tiempo ya es un retardo, si a este tiempo lo multiplicamos por 1000, el retardo es de 256 000us o 256ms, de esa forma se generan los retardos en los microcontroladores, pero y el prescaler, en el ejemplo anterior el TMR0 cuenta de 1 en 1 los pulsos, pero puede contar de 2 en 2, de 4 en 4, de 256 en 256, esto depende del prescaler, son en total 8 opciones a elegir.

Ejemplo:

Si Fosc = 4MHz

Fins = 1MHz

Tins = 1us

TMR0 = 0

Prescaler = 1:4

¿Cuál es el tiempo total transcurrido cuando el TMR0 se desborda por primera vez?

Cada pulso equivale a 1us, la cuenta va de 4 en 4, el TMR0 se incrementa en 1 cada 4 pulsos, el TMR0 inicia en 0, para que el TMR0 pase a 1 deben pasar 4 pulsos, para que el TMR0 pase a 2 deben pasar 4 pulsos más, en total ya 8 pulsos, el TMR0 llega a 255 y luego de 4 pulsos más se desborda y pasa a 0, en total la cantidad de pulsos es:

$$4 * 256 = 1024 \text{ pulsos}$$

Si cada pulso es 1us el tiempo total transcurrido es de 1024us o 1.024ms, en los siguientes ejemplos veremos los códigos de programa para un retardo de 1 segundo aproximado y exacto.



2 TEMPORIZADORES

Ejemplo 1:

Código de programa para un segundo exacto, programa creado por el software PICDel.

```
;-----  
; Code generated by PDEL ver 1.0  
; Description: Waits 1000000 cycles  
;  
PDelay    movlw      .14          ; 1 set number of repetitions (C)  
           movwf      PDel0        ; 1 |  
  
PLoop0    movlw      .72          ; 1 set number of repetitions (B)  
           movwf      PDel1        ; 1 |  
  
PLoop1    movlw      .247         ; 1 set number of repetitions (A)  
           movwf      PDel2        ; 1 |  
  
PLoop2    clrwdt     ; 1 clear watchdog  
           decfsz    PDel2,1      ; 1 + (1) is the time over? (A)  
           goto      PLoop2      ; 2 no, loop  
           decfsz    PDel1,1      ; 1 + (1) is the time over? (B)  
           goto      PLoop1      ; 2 no, loop  
           decfsz    PDel0,1      ; 1 + (1) is the time over? (C)  
           goto      PLoop0      ; 2 no, loop  
  
PDelL1    goto      PDelL2      ; 2 cycles delay  
PDelL2    clrwdt     ; 1 cycle delay  
           return     ; 2+2       Done  
;
```

Ejemplo 2:

Código de programa para generar un retardo de un segundo aproximado, este código es de creación propia, el registro dato1 debe ser declarado previamente y el prescaler en la máxima opción es decir 7.



```
-----
retardo    movlw      d'15'
            movwf      dato1

label_2    clrf       TMR0
            bcf        INTCON, 2

label_1    btfss      INTCON, 2
            goto       label_1
            decfsz   dato1, 1
            goto       label_2
            return

;-----
```

Para utilizar el TMR0 apropiadamente, es necesario:

Paso 1: Seleccionar el modo:

- El modo de temporizador se selecciona por el bit TOSC del registro OPTION_REG (TOSC: 0 = temporizador, 1 = contador)
- Cuando se asigna el pre-escalador al temporizador / contador se debe poner a cero el bit PSA del registro OPTION_REG. El valor del divisor de frecuencias se configura al utilizar los bits PS2 - PS0 del mismo registro
- Al utilizar una interrupción, los bits GIE y TMR0IE del registro INTCON deben estar a uno

Paso 2: Medir y contar

Para medir tiempo:

- Reiniciar el registro TMR0 o escribir un valor conocido en él
- El tiempo transcurrido (en microsegundos al utilizar el oscilador de 4MHz) se mide al leer el registro TMR0
- El bit de bandera TMR0IF del registro INTCON se pone a uno automáticamente siempre que ocurra el desbordamiento del registro TMR0. Si está habilitada, ocurre una interrupción



2 TEMPORIZADORES

Para contar pulsos:

La polaridad de pulsos a contar en el pin RA4 se selecciona por el bit TOSE del registro OPTION_REG (T0SE: 0 = pulsos positivos, 1 = pulsos negativos).

Varios pulsos se pueden leer del registro TMR0. El pre-escalador y la interrupción se utilizan de la misma forma que en el modo de temporizador.

A parte de lo dicho anteriormente, cabe destacar lo siguiente:

- Al asignarle el pre-escalador al temporizador/contador, el pre-escalador se pondrá a 0 con cualquier escritura en el registro TMR0
- Al asignar el pre-escalador al temporizador perro guardián, tanto el WDT como el preescalador se pondrán a 0 con la instrucción CLRWDT
- Al escribir en el registro TMR0, utilizado como un temporizador, no se inicia el conteo de los pulsos inmediatamente, sino con retraso de dos ciclos de instrucciones. Por consiguiente, es necesario ajustar el valor escrito en el registro TMR0
- Al poner el microcontrolador en el modo de reposo se apaga el oscilador de reloj. No puede ocurrir el desbordamiento ya que no hay pulsos a contar. Es la razón por la que la interrupción por el desbordamiento del TMR0 no puede “despertar” al procesador del modo de reposo
- Si se utiliza como un contador de reloj externo sin pre-escalador, la longitud de pulso mínima o tiempo muerto entre dos pulsos deberá ser $2T_{osc} + 20nS$ (T_{osc} es el período de señal de reloj del oscilador)
- Si se utiliza como un contador de reloj externo con pre-escalador, la longitud de pulso mínima o tiempo muerto entre dos pulsos es sólo $10nS$
- El registro del pre-escalador de 8 bits no está disponible al usuario, lo que significa que no es posible leerlo o escribir en él directamente
- Al cambiar de asignación del pre-escalador del Timer0 al temporizador perro guardián, es necesario ejecutar la siguiente secuencia de instrucciones escritas en ensamblador para impedir reiniciar el microcontrolador



Proyecto 4: El oscilador

A) Planteamiento del programa:

El programa consiste en implementar un oscilador, aplicando lógica programada, las entradas y salidas se pueden asignar a cualquier puerto y pin, vamos a utilizar el puerto A para las entradas y el puerto B para las salidas, el tiempo de oscilación es de 1 segundo para el encendido y un segundo para el apagado, los retardos dependen de la frecuencia del cristal. El oscilador prende y apaga un led cada segundo, se utilizará el puerto B para la salida, éste debe ser configurado como salida y el pin asignado es el RB0, el esquema electrónico se muestra en la figura 2.3.

B) Esquema electrónico:

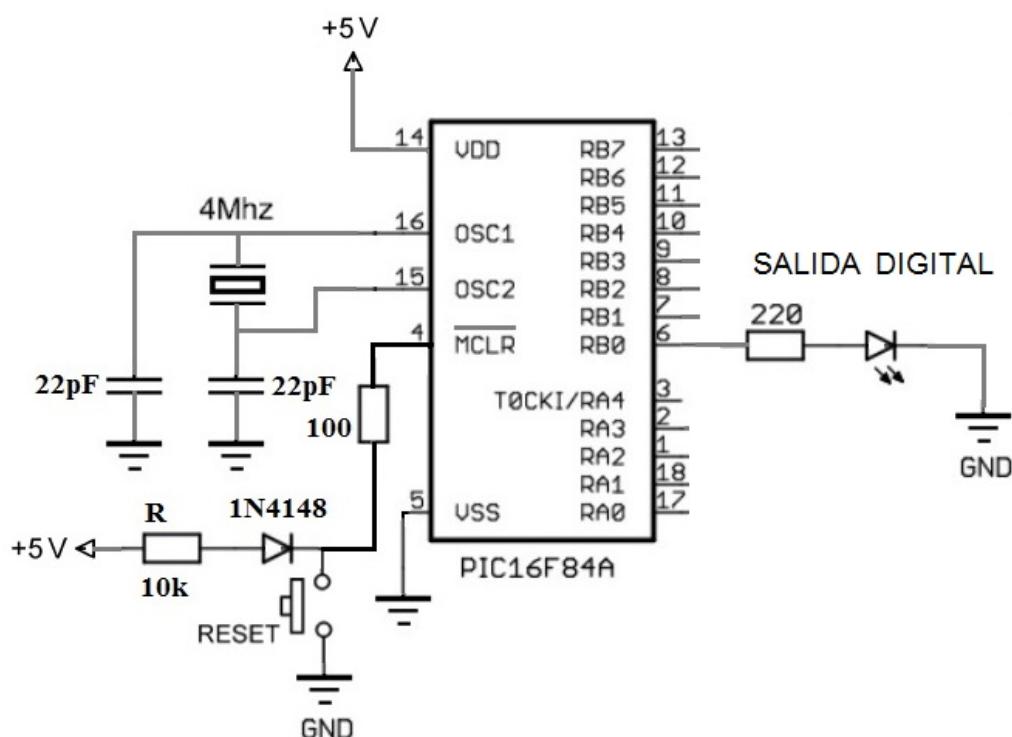


Figura 2.3. Circuito electrónico para el proyecto 4.



2 TEMPORIZADORES

C) Instrucciones a utilizar:

La instrucción decfsz: decrementa el valor de f en uno y salta una instrucción si el valor de f es 0:

decfsz ESTADO,1

Decrementa el valor del registro ESTADO en 1 y el resultado lo almacena en el mismo registro, si el valor del registro ESTADO resulta 0 salta a la subsiguiente instrucción, de lo contrario no salta y la secuencia de ejecución continua de manera normal.

La instrucción call: guarda la dirección de retorno en la pila y después llama a la subrutina situada en la dirección cargada en el PC, por ejemplo:

	call	sumar
	goto	inicio
sumar	addwf	ESTADO,1
	return	

La instrucción call llama a la subrutina sumar, salta a la etiqueta sumar y ejecuta todas las instrucciones de la subrutina hasta encontrar la instrucción return, que indica retorno de subrutina o fin de subrutina.

La instrucción return: cumple la misma función que retlw, con la diferencia que return retorna directamente, por ejemplo:

	call	sumar
	goto	inicio
sumar	addwf	ESTADO,1
	return	

Retorna a goto inicio.



D) Código del programa:

```
;-----  
;Nombre del programa: oscilador.asm  
Autor: Ing. David Apaza Condori  
Ensamblador: MPLAB V8.14  
;  
_____  
_config H'3FF1'  
#include"p16F84.inc"  
  
dato1 EQU 0x0c  
  
ORG 0x00  
  
        bsf STATUS,5  
        movlw 0x00  
        movwf TRISA  
        clrf TRISB  
        movlw b'00000111'  
        movwf OPTION_REG      ; configura el prescaler  
        bcf STATUS,5  
        clrf PORTB  
  
inicio    bsf PORTB,0          ; activa el led  
        call retardo          ; espera un segundo  
        bcf PORTB,0          ; apaga el led  
        call retardo  
        goto inicio  
  
retardo   movlw d'15'          ; retardo aproximado de 1s  
        movwf dato1  
  
label_2   clrf TMR0  
        bcf INTCON,2  
  
label_1   btfss INTCON,2  
        goto label_1  
        decfsz dato1,1  
        goto label_2  
        return  
  
END  
;
```



E) Simulación del programa:

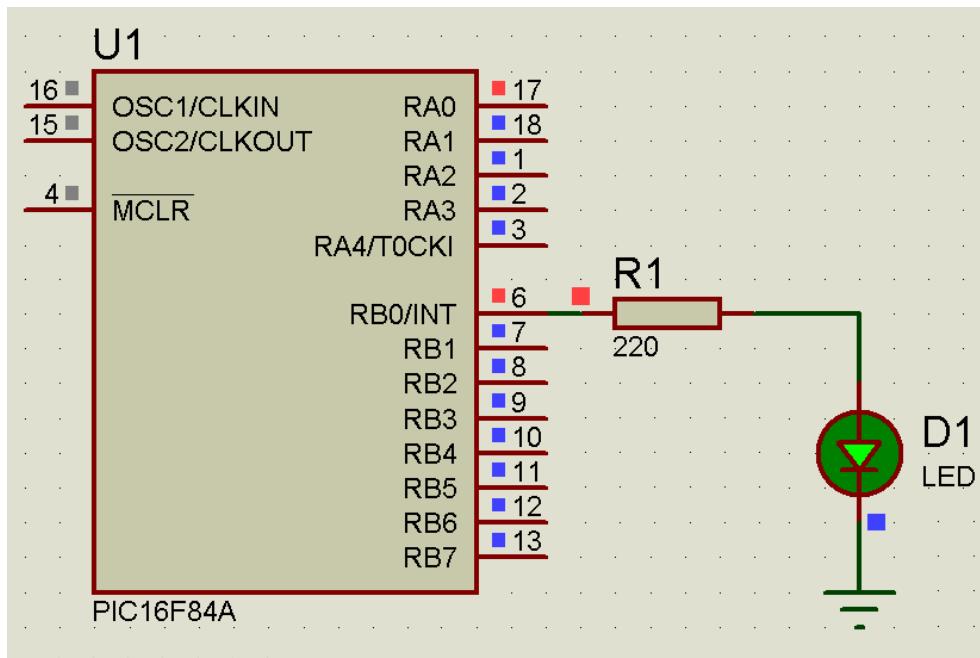


Figura 2.4. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.

Proyecto 5: Semáforo electrónico

A) Planteamiento del programa:

El programa consiste en implementar un semáforo electrónico, aplicando lógica programada, las entradas y salidas se pueden asignar a cualquier puerto y pin, vamos a utilizar el puerto A para las entradas y el puerto B para las salidas, los tiempos de encendido y apagado para las luces del semáforo se indican en las condiciones de funcionamiento, los retardos dependen de la frecuencia del cristal. El semáforo activa la luz roja de la calle A y la luz verde de la calle B, cumplido el tiempo apaga ambas luces y activa la luz ambar de ambas calles durante un segundo, luego apaga la luz ambar de ambas calles y activa la luz verde de la calle A y la luz roja de la calle B, cumplido el tiempo apaga ambas luces y activa nuevamente la luz ambar para ambas calles, finalmente reinicia el programa, se utilizará el puerto B para las salidas, éste debe ser configurado como salida y los pines asignados son: RB0, RB1, RB2, RB3, RB4, RB5, el esquema electrónico se muestra en la figura 2.5.



El diseño del sistema debe tener las siguientes condiciones:

Luz en Rojo: La luz roja debe estar encendida durante 5 segundos

Luz en Amarillo: La luz amarilla debe estar encendida durante 1 segundo

Luz en Verde: La luz verde debe estar encendida durante 7 segundos

La solución se centra en el programa de control que gobernara la secuencia de encendido de los indicadores LED en el semáforo y sus tiempos de espera entre cada cambio de luz, se puede trabajar un modelo previo que represente estas funciones antes de empezar a escribir código en ensamblador, la idea principal es tener un modelo que represente los objetivos y además tuviera soporte para futuras modificaciones lo que le agregaría mayor escalabilidad al programa.

B) Esquema electrónico:

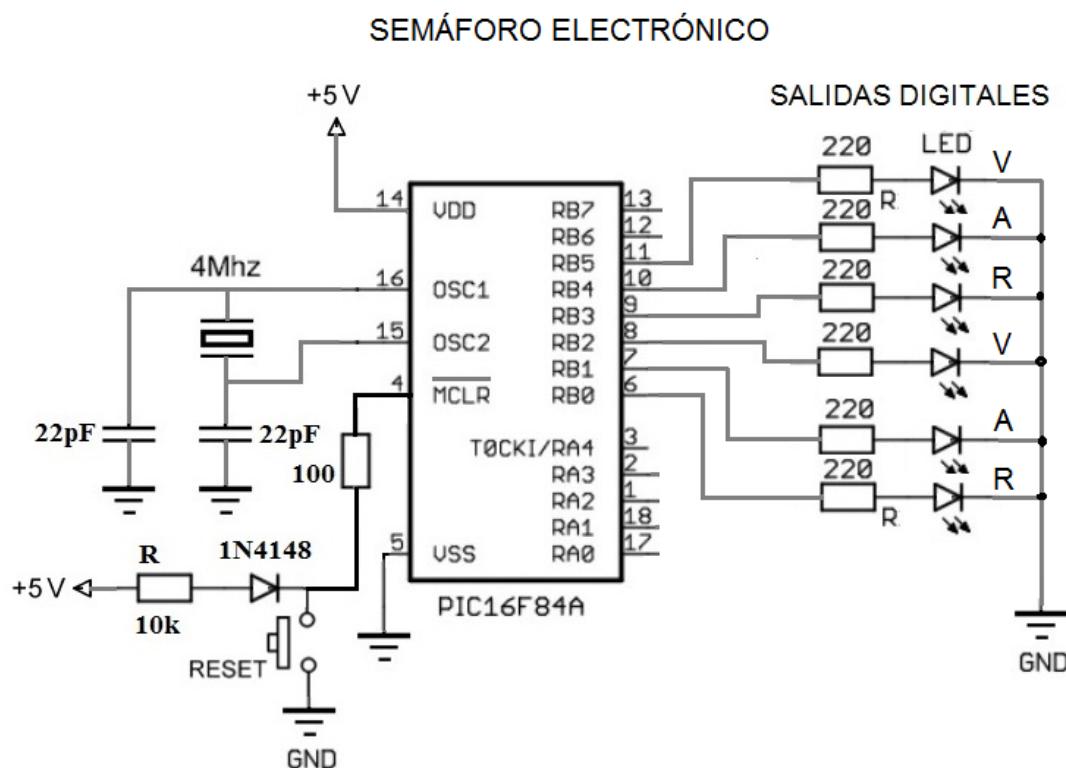


Figura 2.5. Circuito electrónico para el proyecto 5.



2 TEMPORIZADORES

C) Código del programa:

```
;-----  
;Nombre del programa: semáforo.asm  
Autor: Ing. David Apaza Condori  
Ensamblador: MPLAB V8.14  
;  
_config H'3FF1'  
#include "p16F84.inc"  
  
dato1 EQU 0x0C  
dato2 EQU 0x0D  
  
ORG 0x00  
  
        bsf STATUS, 5  
        movlw 0x00  
        movwf TRISA  
        clrf TRISB  
        movlw b'00000111'  
        movwf OPTION_REG  
        bcf STATUS, 5  
  
inicio    clrf PORTB           ; reinicia el semáforo  
        bsf PORTB, 5            ; activa el verde calle B  
        bsf PORTB, 0            ; activa el rojo calle A  
        call retar_5s          ; espera 5 segundos  
        bcf PORTB, 0  
        bcf PORTB, 5  
        bsf PORTB, 1            ; activa el ambar calle B  
        bsf PORTB, 4            ; activa el ambar calle A  
        call retar_1s          ; espera 1 segundo  
        bcf PORTB, 4            ; apaga el ambar calle B  
        bcf PORTB, 1            ; apaga el ambar calle A  
        bsf PORTB, 3            ; activa el rojo calle B  
        bsf PORTB, 2            ; activa el verde calle A  
        call retar_7s          ; espera 7 segundos  
        bcf PORTB, 3  
        bcf PORTB, 2
```



```
bsf      PORTB, 4
bsf      PORTB, 1
call    retar_1s
goto    inicio

;-----
retar_5s  movlw    d'75'      ;retardo de 5 segundos
          movwf    dato1
lab_1     clrf    TMRO
          bcf     INTCON, 2
lab_2     btfss   INTCON, 2
          goto    lab_2
          decfsz dato1,1
          goto    lab_1
          return
retar_1s  movlw    d'15'      ;retardo de 1 segundo
          movwf    dato1
lab_3     clrf    TMRO
          bcf     INTCON, 2
lab_4     btfss   INTCON, 2
          goto    lab_4
          decfsz dato1,1
          goto    lab_3
          return
retar_7s  movlw    d'105'     ;retardo de 7 segundos
          movwf    dato1
lab_5     clrf    TMRO
          bcf     INTCON, 2
lab_6     btfss   INTCON, 2
          goto    lab_6
          decfsz dato1,1
          goto    lab_5
          return
END           ;fin de programa
;-----
```



2 TEMPORIZADORES

D) Simulación del programa:

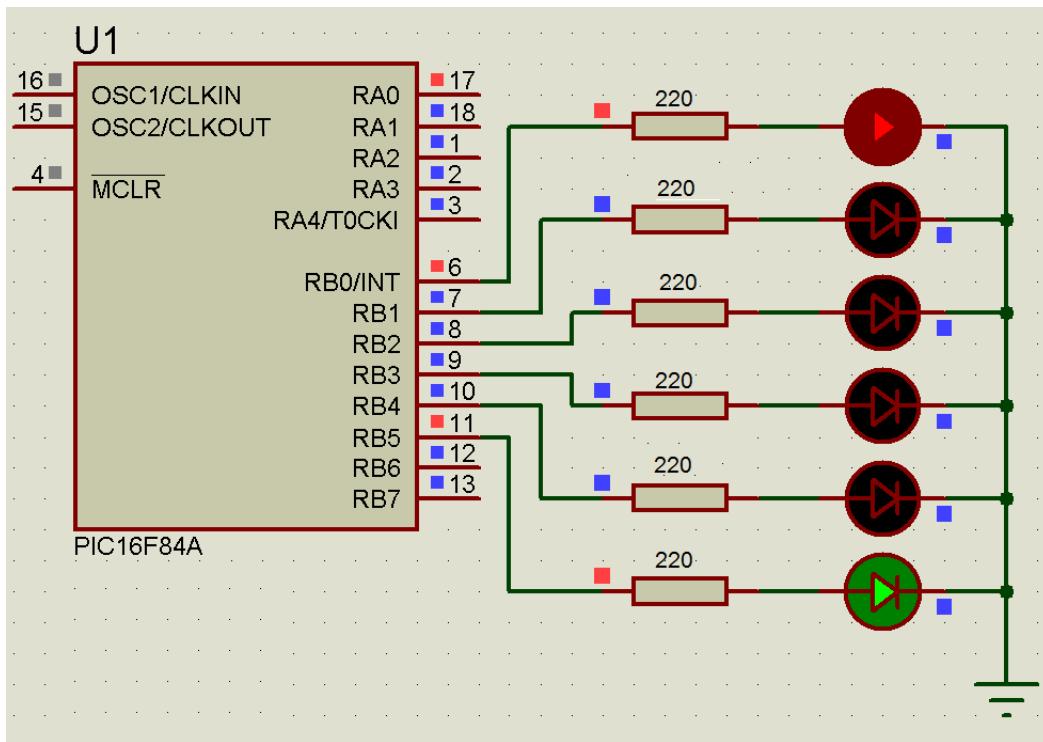


Figura 2.6. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.

Proyecto 6: Temporizador de 0 a 9

A) Planteamiento del programa:

El programa consiste en implementar un temporizador de 0 a 9 segundos, aplicando lógica programada y un display de siete segmentos cátodo común, el mismo que permite visualizar los números del 0 al 9, los códigos en formato binario para cada número serán almacenados en una tabla en la memoria del PIC, el funcionamiento de este programa debe ajustarse a las siguientes condiciones:

1. El valor inicial del temporizador es cero
2. El tiempo base es de un segundo
3. La cuenta se incrementa de uno en uno cada tiempo base
4. Si la cuenta está en 9 luego de un tiempo base pasa a 0



Se debe tener en cuenta que el código cambia dependiendo si usamos un display de ánodo común o de cátodo común, para este proyecto aplicaremos un display de cátodo común, lo que implica un 1 lógico para activar un segmento, y un 0 lógico para apagar un segmento.

Para visualizar los números del 0 al 9 en el display utilizaremos la misma tabla del proyecto 3, que contiene todos los códigos en formato binario y hexadecimal, estos códigos serán almacenados en una tabla en memoria y de esta tabla se sacarán los códigos para ser enviados al PORTB del PIC, todo esto nos será útil al momento de realizar la programación.

El display se conectará al PORTB, éste debe ser configurado como salida y los pines asignados son: RB1, RB2, RB3, RB4, RB5, RB6 y RB7, el RB1 se conecta al segmento a, el RB2 se conecta al segmento b, en ese orden hasta el RB7 que se conecta al segmento g. El esquema electrónico se muestra en la figura 2.7.

B) Esquema electrónico:

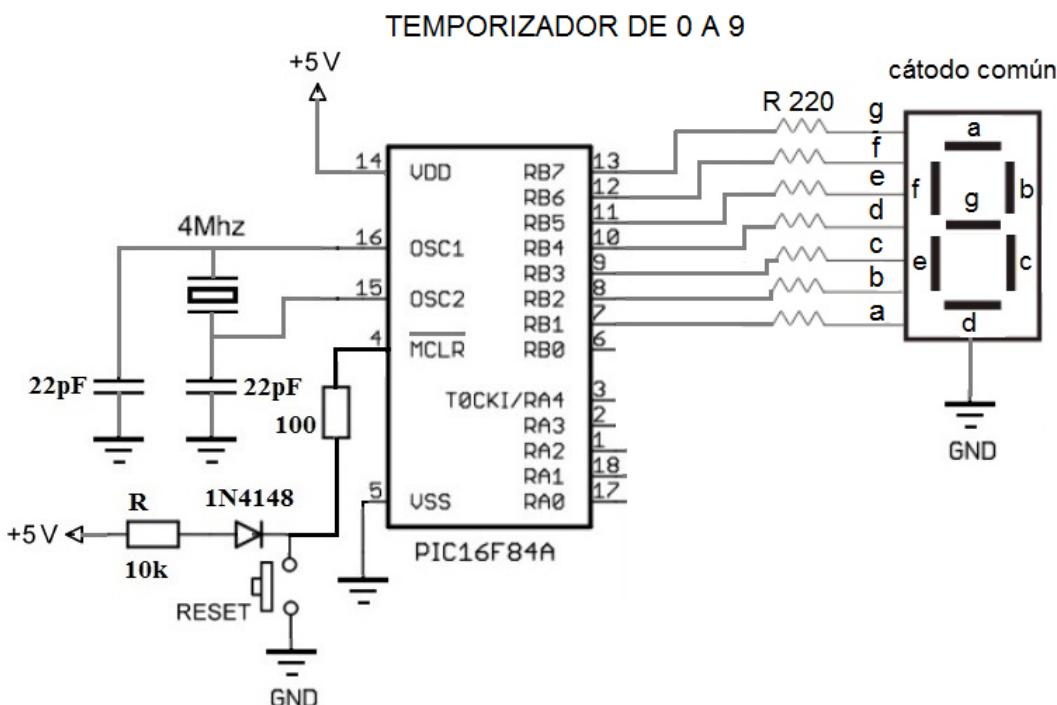


Figura 2.7. Circuito electrónico para el proyecto 6.



2 TEMPORIZADORES

C) Instrucciones a utilizar:

La instrucción decf: decrementa el valor de f en uno y el resultado lo guarda en f o w, dependiendo del destino. Ejemplo:

decf ESTADO,1

Decrementa el valor del registro ESTADO en 1 y el resultado lo almacena en el mismo registro.

La instrucción incf: incrementa el valor de f en uno y el resultado lo guarda en f o w, dependiendo del destino. Ejemplo:

incf ESTADO,1

Incrementa el valor del registro ESTADO en 1 y el resultado lo almacena en el mismo registro.

La instrucción call: guarda la dirección de retorno en la pila y después llama a la subrutina situada en la dirección cargada en el PC, por ejemplo:

	call	sumar
	goto	inicio
sumar	addwf	ESTADO,1
	return	

La instrucción call llama a la subrutina sumar, salta a la etiqueta sumar y ejecuta todas las instrucciones de la subrutina hasta encontrar la instrucción return, que indica retorno de subrutina o fin de subrutina.

La instrucción return: cumple la misma función que retlw, con la diferencia que return retorna directamente, sin cargar valor alguno a w.



D) Código del programa:

```
;-----  
;Nombre del programa: temporizador0a9.asm  
;Autor: Ing. David Apaza Condori  
;Ensamblador: MPLAB V8.14  
;  
;  
__config H'3FF1'  
  
#include "p16F84.inc"  
  
dato1 EQU 0x0C  
dato2 EQU 0x0D  
  
ORG 0x00  
  
        bsf STATUS, 5  
        movlw 0x00  
        movwf TRISA  
        clrf TRISB  
        movlw b'00000111'  
        movwf OPTION_REG  
        bcf STATUS, 5  
  
lab_2    clrf dato2      ;reinicia la cuenta a 0  
lab_1    movf dato2, 0  
        call tabla      ;saca el número de la tabla  
        movwf PORTB      ;envía el número al display  
        call retardo    ;espera 1 segundo  
        btfss dato2, 3    ;verifica si llegó a 9  
        goto lab_3  
        btfss dato2, 0    ;verifica si llegó a 9  
        goto lab_3  
        goto lab_2  
lab_3    incf dato2, 1    ;incrementa la cuenta en 1  
        goto lab_1  
;  
;
```



2 TEMPORIZADORES

```
;  
tabla    addwf    PCL,1  
         retlw    b'01111111' ;número 0 display  
         retlw    b'00001101' ;número 1 display  
         retlw    b'10110111' ;número 2 display  
         retlw    b'10011111' ;número 3 display  
         retlw    b'11001101' ;número 4 display  
         retlw    b'11011011' ;número 5 display  
         retlw    b'11111011' ;número 6 display  
         retlw    b'00001111' ;número 7 display  
         retlw    b'11111111' ;número 8 display  
         retlw    b'11011111' ;número 9 display  
  
retardo  movlw    d'15'      ;retardo de 1 segundo  
         movwf    datol  
label_2  clrf    TMR0  
         bcf     INTCON,2  
label_1  btfss   INTCON,2  
         goto    label_1  
         decfsz datol,1  
         goto    label_2  
         return  
END      ;fin de programa  
;
```



E) Simulación del programa:

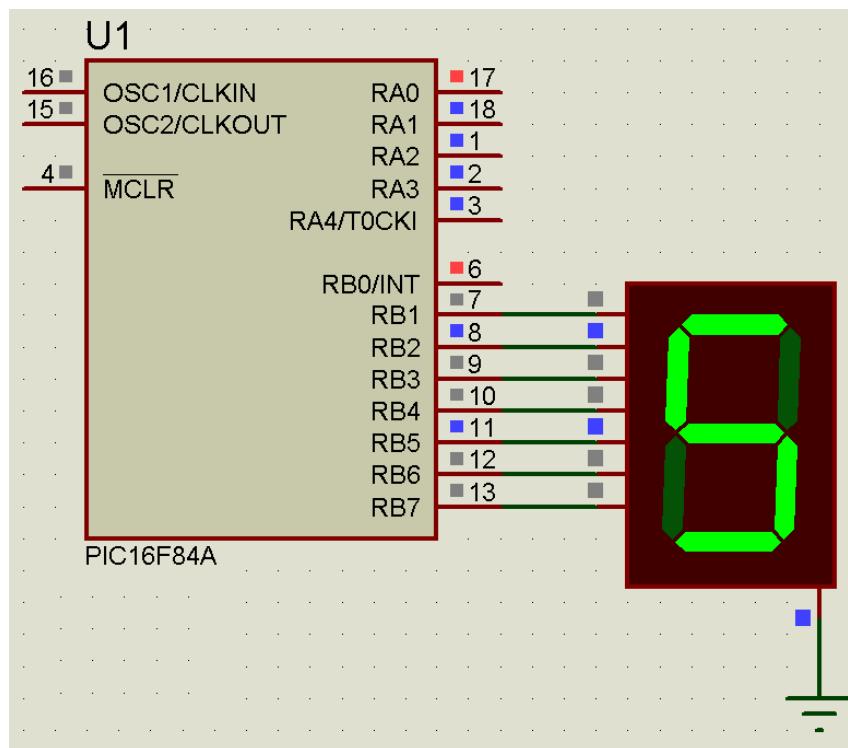


Figura 2.8. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.

Proyecto 7: Temporizador de 00 a 99

A) Planteamiento del programa:

El programa consiste en implementar un temporizador de 00 a 99 segundos, aplicando lógica programada y dos displays de siete segmentos cátodo común, los mismos que permiten visualizar los números del 00 al 99, los códigos en formato binario para cada número serán almacenados en una tabla en la memoria del PIC, el funcionamiento de este programa debe ajustarse a las siguientes condiciones:

1. El valor inicial del temporizador es cero
2. El tiempo base es de un segundo
3. La cuenta se incrementa de uno en uno cada tiempo base
4. Si la cuenta está en 99 luego de un tiempo base pasa a 00



2 TEMPORIZADORES

Al igual que en el proyecto anterior se debe tener en cuenta que el código cambia dependiendo si usamos un display de ánodo común o de cátodo común, para este proyecto aplicaremos dos displays de cátodo común, lo que implica un 1 lógico para activar un segmento, y un 0 lógico para apagar un segmento.

Para visualizar los números del 00 al 99 en los displays utilizaremos la misma tabla del proyecto 3, que contiene todos los códigos en formato binario y hexadecimales, estos códigos serán almacenados en una tabla en memoria y de esta tabla se sacarán los códigos para ser enviados al PORTB del PIC, todo esto nos será útil al momento de realizar la programación.

A diferencia del proyecto anterior ahora tenemos dos displays, éstos deben estar conectados en paralelo, el segmento a del display 1 con el segmento a del display 2, el segmento b del display 1 con el segmento b del display 2, así sucesivamente hasta el segmento g del display 1 con el segmento g del display 2. Ya conectados los dos displays en paralelo se procede a conectar en paralelo al PORTB del PIC de la misma forma como se hizo en el proyecto 6. El esquema electrónico se muestra en la figura 2.9.

B) Esquema electrónico:

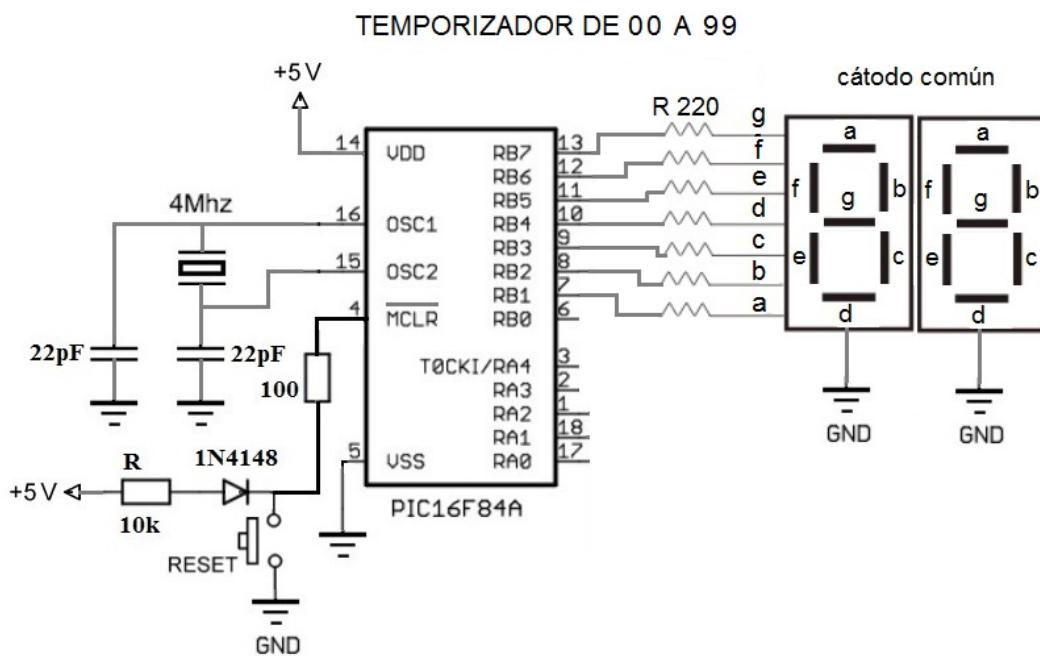


Figura 2.9. Circuito electrónico para el proyecto 7.



C) Código del programa:

```
;-----  
;Nombre del programa: temporizador00a99.asm  
Autor: Ing. David Apaza Condori  
Ensamblador: MPLAB V8.14  
;  
__config H'3FF1'  
#include "p16F84.inc"  
  
dato1 EQU 0x0C  
dato2 EQU 0x0D  
dato3 EQU 0x0E  
dato4 EQU 0x0F  
  
ORG 0x00  
    bsf STATUS, 5  
    movlw 0x00  
    movwf TRISA  
    clrf TRISB  
    movlw 0x03  
    movwf OPTION_REG  
    bcf STATUS, 5  
    clrf PORTA  
lab_06 clrf dato4  
lab_01 clrf dato3  
lab_03 movlw d'20'  
        movwf dato2  
inicio movf dato3, 0  
        call tabla  
        movwf PORTB  
        bsf PORTA, 0  
        call retardo  
        bcf PORTA, 0  
        movf dato4, 0  
;  
-----
```



2 TEMPORIZADORES

```
-----;  
call      tabla  
movwf    PORTB  
bsf       PORTA,1  
call      retardo  
bcf       PORTA,1  
decfsz   dato2,1  
goto     inicio  
btfs    dato3,3  
goto     lab_02  
btfs    dato3,0  
goto     lab_02  
goto     lab_04  
  
lab_02    incf    dato3,1  
goto     lab_03  
lab_04    btfs    dato4,3  
goto     lab_05  
btfs    dato4,0  
goto     lab_05  
goto     lab_06  
lab_05    incf    dato4,1  
goto     lab_01  
  
retardo  movlw   d'2'  
          movwf   dato1  
label_2  clrf    TMR0  
          bcf     INTCON,2  
label_1  btfs    INTCON,2  
          goto    label_1  
          decfsz dato1,1  
          goto    label_2  
          return  
-----;
```



```

;-----;
tabla    addwf    PCL,1
          retlw    b'01111111'
          retlw    b'00001101'
          retlw    b'10110111'
          retlw    b'10011111'
          retlw    b'11001101'
          retlw    b'11011011'
          retlw    b'11111011'
          retlw    b'00001111'
          retlw    b'11111111'
          retlw    b'11011111'

END
;-----;

```

D) Simulación del programa:

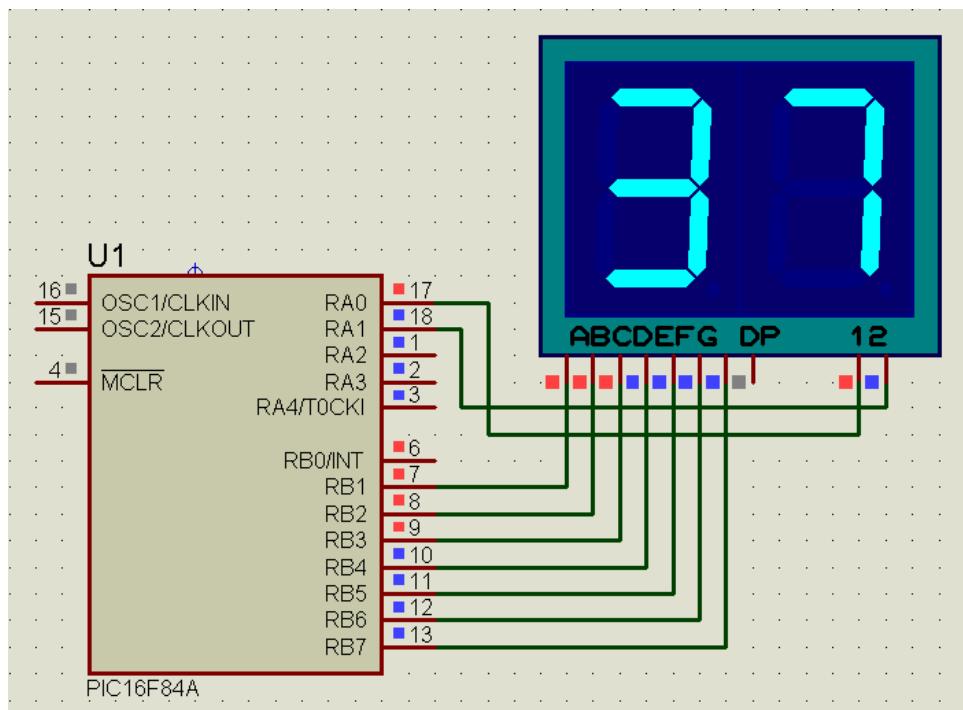


Figura 2.10. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



2 TEMPORIZADORES

Actividades

A) Responda los espacios en blanco:

1. Si ejecutamos la instrucción addwf PORTA,W cuando el W = 0xF8 y el PORTA = 0x88 el registro STATUS vale `xxxxx _ _ _ b`
2. Cuando usamos el modulo TMR0 solo podríamos contar desde..... hasta..... ciclos de instrucción.
3. Cuando colocamos los bits del preescaler a indicamos que se lleve acabo una división de frecuencia entre 64.
4. La interrupción RB0/INT es una interrupción externa

B) Responda “V” si es verdadero y “F” si es falso, a las siguientes afirmaciones:

1. La instrucción movwf TRISA, coloca todos los bits de TRISA a 1 si W = 0xFF. ()
2. La instrucción bsf STATUS,6 coloca el bit 6 de STATUS a 1.....()
3. La instrucción return se ejecuta en un solo ciclo de instrucción.....()
4. Una subrutina puede llamar a otra subrutina.....()
5. La instrucción clrf INTCON pone a cero todos los bits del registro INTCON...()

C) Indique que hace cada una de las siguientes instrucciones:

1. bcf STATUS,4

2. movlw d'259'

3. decfsz dato3

4. movf PORTA

5. movlw TRISB

6. call activar

7. retlw d'4'

**D) Resaltar las líneas de código con error y corregir a lado.**

```
;  
        bsf      STATUS, 9  
        movlw    0x0G  
        movwf    TRISA  
        clrf    TRIB  
        bcf      STATUS, 5  
        movlw    b00000001'  
        movwf    dato2  
lab_02    movf    dato2, 0  
          movf    PORTB  
          call    retardo  
          btfsc   dato2, 7  
          goto    lab_01  
lab_03    rlf     dato2, 1  
          goto    lab_02  
lab_04    movf    dato2, 0  
          movwf    PORTB  
          call    retardo  
          btfsc   dato2, 8  
          goto    lab_03  
lab_01    rrf     dato2, 1  
          gotu    lab_04  
retardo  movlw    d'7'  
          movwf    dato1  
label_2   clrf    TMR0  
          bcf     INTCON, 2  
label_1   btfsz   INTCON, 2  
          goto    label_1  
          decfsz  dato1, 3  
          goto    label_2  
          return  
;
```



2 TEMPORIZADORES

E) Marque la alternativa correcta:

1. Si un microcontrolador trabaja con un XTAL de 4MHz, el máximo tiempo en ejecutar la instrucción decfsz es de:
a) 0.5us b) 1us c) 2us d) 3us e) N.A.
2. Si cargamos el valor de 0xFF a W y luego le sumamos 1 a W, entonces el registro STATUS vale:
a) xxxx000 b) xxxx111 c) xxxx010 d) xxxx011 e) N.A.
3. Si cargamos el registro TMR0 con el valor 0x00, cuando TMR0 sea igual a 0x07 se puede afirmar que el tiempo transcurrido es: (Considere Fosc = 4MHz y preescalador 111)
a) 1536us b) 1992us c) 2048us d) 3044us e) N.A.

F) Responder:

1. ¿Qué tamaño puede tener la memoria de programa de un PIC de gama media?

2. ¿Qué tamaño tienen las localizaciones de memoria?

3. ¿Qué tamaño tienen las páginas de esta memoria?

4. ¿Qué registro del microcontrolador “apunta” a la memoria de programa de un PIC?

5. ¿Cómo opera este registro durante la ejecución de un programa?

6. ¿De qué formas se puede modificar el contenido del contador de programa?

7. ¿Se puede leer o escribir un dato en la memoria de programa de un PIC?



8. ¿Qué valor toma el contador de programa después de un reset?

9. ¿Cuál es el objetivo del bloque PWRT?

10. ¿Cuál es la diferencia entre WDT y TMR0?

11. ¿Cuál es la frecuencia de instrucción para un cristal de 20MHz?

12. ¿De qué forma se puede lograr un retardo de un segundo exacto?





2 TEMPORIZADORES



CAPÍTULO 3

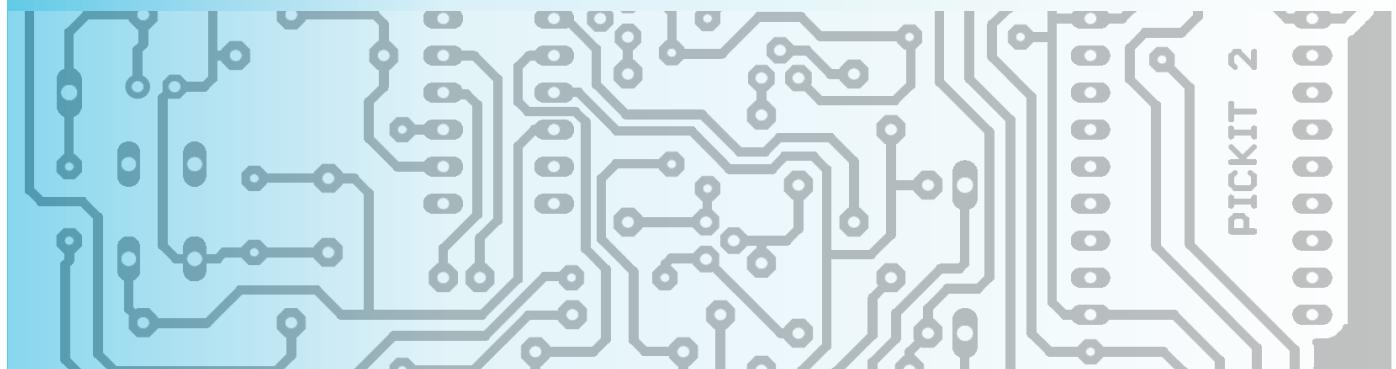
INTERRUPCIONES

CONTENIDO DEL CAPÍTULO

- 1. Introducción
- 2. Las interrupciones
 - 2.1. Registro de control INTCON
 - 2.2. Interrupción externa INT
 - 2.3. Interrupción del TMR0
 - 2.4. Interrupción del PORTB
 - 2.5. Interrupción de la EEPROM
- Proyecto 8: Teclado matricial 4x4

APRENDEREMOS A:

- Identificar las diferentes fuentes de interrupción de un PIC
- Identificar los puertos del PIC y su relación con las interrupciones
- Configurar correctamente las interrupciones del PIC
- Utilizar correctamente los registros internos y sus bits relacionados con las interrupciones





3 INTERRUPCIONES

1. Introducción

Una interrupción consiste en un mecanismo por el cual un evento interno o externo puede interrumpir la ejecución de un programa en cualquier momento. A partir de entonces se produce automáticamente un salto a una subrutina de atención a la interrupción, ésta atiende inmediatamente el evento y retoma luego la ejecución del programa exactamente donde estaba en el momento de ser interrumpido, continuando su tarea justo donde la dejó. La interrupción tiene la característica de la inmediatez, nace de la necesidad de ejecutar una subrutina en el instante preciso y, por tanto, se considera su intervención urgente.

Las interrupciones constituyen el mecanismo más importante para la conexión del microcontrolador con el exterior ya que sincroniza la ejecución de programas con los acontecimientos externos. Esto es muy útil, por ejemplo, para el manejo de dispositivos de entrada que requieren de una atención inmediata, tales como detección de pulsos externos, recepción de datos, activación de pulsadores, etc.

El funcionamiento de las interrupciones es similar al de las subrutinas, de las cuales se diferencian, principalmente, en los procedimientos que las ponen en marcha. Así como las subrutinas se ejecutan cada vez que en el programa aparece una instrucción CALL, las interrupciones se ponen en marcha al aparecer en cualquier instante un evento externo al programa, por lo tanto, una interrupción se activa por un mecanismo hardware.

El PIC16F84 dispone de 4 posibles fuentes de interrupción:

1. Interrupción INT por activación del pin RB0 / INT
2. Interrupción RBI por cambio de estado en una o varias de las 4 líneas de más peso del puerto B (RB7:RB4)
3. Interrupción T0I por desbordamiento del TMR0
4. Interrupción EEI por la finalización de la escritura en la EEPROM de datos

El PIC16F877A también dispone de estas fuentes de interrupción y su configuración y funcionamiento es igual a la del PIC16F84A. Cuando se produce cualquiera de los sucesos indicados anteriormente se origina una petición de interrupción que, si se acepta, origina el siguiente mecanismo hardware:



1. Salva el valor actual del PC guardando su contenido en la pila
2. El bit GIE del registro INTCON es puesto a cero, lo que prohíbe cualquier otra interrupción
3. El PC se carga con el valor 0004h, que es la posición de vector de interrupción
4. Comienza a ejecutarse el programa de atención a la interrupción que se encuentra a partir de la dirección 0004h

En el PIC 16F84, los bits de control localizados en el registro INTCON habilitan y configuran las interrupciones. Cada causa de interrupción actúa con dos flags (banderas): un flag indica si se ha producido o no la interrupción: T0IF, INTF, RBIF y EEIF, y el otro flag funciona como permiso o prohibición de la interrupción en sí: T0IE, INTE, RBIE, EEIE y GIE.

Hay un único vector de interrupción en la dirección 0004h. La instrucción RETFIE utilizada al final de la subrutina de interrupción es idéntica a un retorno de subrutina RETURN, pero además, coloca automáticamente a “1” el bit GIE, volviendo a habilitar las interrupciones.



¿ Sabías que . . . ?

Las interrupciones son eventos aleatorios y pueden generarse en la parte interna o externa del PIC, es importante el uso de la instrucción RETFIE para finalizar una interrupción.

2. Las interrupciones

Las interrupciones constituyen el procedimiento más rápido y eficaz del microcontrolador para el tratamiento de los acontecimientos del mundo exterior. Si algún parámetro o circunstancia tiene influencia en el desarrollo del programa, la interrupción es el medio por el cual el microcontrolador lo atiende inmediatamente. Al programa especial que trata dicho acontecimiento se le llama “rutina de interrupción”.



3 INTERRUPCIONES

Las interrupciones son desviaciones del flujo de control del programa originadas asincrónicamente por diversos sucesos que no se hallan bajo la supervisión de las instrucciones. Dichos sucesos pueden ser externos al sistema, como la generación de un flanco o nivel activo en una terminal del microcontrolador, o bien internos, como el desbordamiento de un contador. El comportamiento del microcontrolador ante la interrupción es similar al de la instrucción CALL de llamada a subrutina.

Cuando se utiliza la instrucción CALL, se detiene la ejecución del programa en curso, se guarda la dirección actual del PC en la pila y éste se carga con una nueva dirección que se especifica en dicha instrucción. En el caso de una interrupción, el proceso inicia de la misma forma; se detiene la ejecución del programa en curso, se guarda la dirección actual del PC en la pila y éste **se carga con una dirección «reservada» de la memoria de programa (0004h)**, llamada **Vector de Interrupción**.

En los PIC16X8X el Vector de interrupción se halla situado en la dirección 0004h, en donde comienza la Rutina de Servicio a la Interrupción (RSI). En general, en dicho Vector se suele colocar **una instrucción de salto incondicional (GOTO)**, que traslada el flujo de control a la zona de la memoria de código destinada a contener la rutina de atención a la interrupción.



¿ Sabías que . . . ?

El microcontrolador 16F84 tiene un solo vector de interrupción, y ocupa la posición 0x04h de la memoria de programa, otros sistemas tienen mas de un vector de interrupción.

En la figura 3.1. se muestra la dirección del vector de interrupción que está localizada en la posición 0004h de la memoria de programa del PIC16F84. La RSI suele comenzar guardando en la memoria de datos algunos registros específicos del procesador. Concretamente aquellos que la RSI va a emplear y va a alterar su contenido. Antes del retorno al programa principal se recuperan los vectores guardados y se restaura completamente el estado del procesador.

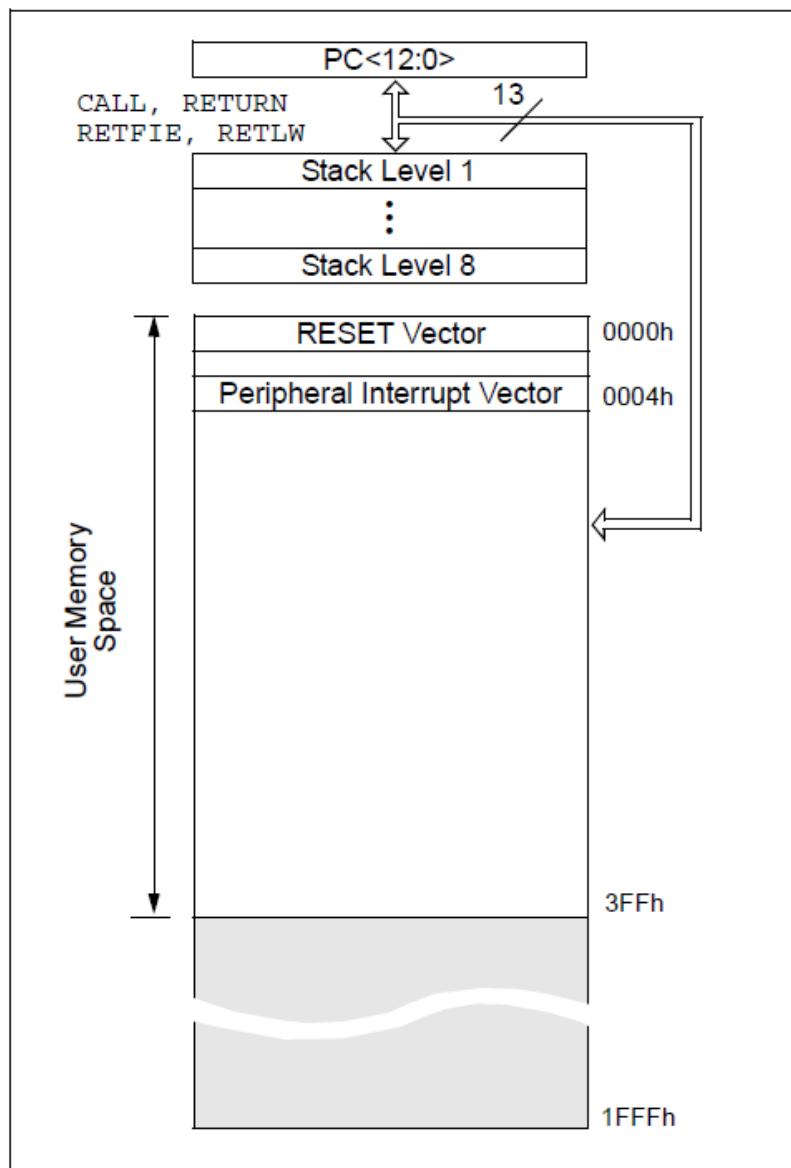


Figura 3.1. Mapeo de la memoria de programa de PIC 16F84.

Algunos procesadores salvan estos registros en la pila, pero los PIC no disponen de instrucciones para introducir (push) y extraer (pop) información de la pila, utilizando para este fin registros de propósito general de la memoria de datos. Los PIC 16X8X pueden ser interrumpidos por 4 causas diferentes, pero todas ellas desvían el flujo de control a la dirección 0004h, por lo que otra de las operaciones iniciales de la RSI es averiguar cuál de las posibles causas ha sido la responsable de la interrupción en curso. Para ello se exploran los señalizadores de las fuentes de interrupción. Otro detalle importante de la RSI de los PIC 16X8X es que estos microcontroladores poseen un bit GIE (Global Interrupt Enable) que cuando vale 0 inhabilita todas las interrupciones.



3 INTERRUPCIONES

Pues bien, al comenzar la RSI dicho bit GIE se pone automáticamente a 0, con objeto de no atender nuevas interrupciones hasta que se termine la que ha comenzado. En el retorno final de la interrupción, GIE pasa a valer automáticamente 1 para volver a tener en cuenta las interrupciones. Dicho retorno de interrupción se realiza mediante la instrucción RETFIE. Antes del retorno conviene borrar el señalizador de la causa de interrupciones que se ha atendido, porque si bien los señalizadores se ponen a 1 automáticamente en cuanto se produce la causa que indican, la puesta a 0 se hace por programa.

Una dificultad inicial que presentan estas interrupciones es que sea cual sea la que ha originado la interrupción, la rutina de atención siempre empieza en la dirección del vector de interrupción que es la 0004h. Otros procesadores disponen de un vector de interrupción para cada causa que la produce. Con el PIC16F84 no queda otro remedio que comenzar averiguando cuál ha sido la causa de la interrupción, probando los señalizadores correspondientes, y desviando hacia el procedimiento particular que a tiende a cada una.

Cuando ocurre cualquiera de los 4 sucesos indicados, se origina una petición de interrupción, que si se acepta y se atiende comienza depositando el valor del PC actual en la pila, poniendo el bit GIE = 0 y cargando en el PC el valor 0004h, que es el vector de interrupción donde se desvía el flujo de control. Cada fuente de interrupción dispone de un señalizador o flag, que es un bit que se pone automáticamente a 1 cuando se produce. Además cada fuente de interrupción tiene otro bit de permiso, que según su valor permite o prohíbe la realización de dicha interrupción.

2.1. Registro de control INTCON

A veces, muchos programas no precisan la ayuda de las interrupciones y no las usan. Por este motivo debe existir una llave general que permita o prohíba las interrupciones en su conjunto. En el PIC16F84 existe un registro de control en el área SFR de la RAM, llamado INTCON, que regula el funcionamiento de las interrupciones, la dirección que ocupa dicho registro es la 0Bh del banco 0 y la dirección 8Bh del banco 1, el registro INTCON y sus 8 bits de control se detallan a continuación.

El bit de más peso de INTCON (INTCON bit 7) se denomina GIE (Permiso Global de Interrupciones).

**INTCON REGISTER (ADDRESS 0Bh, 8Bh)**

R/W-0	R/W-x						
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit 7				bit 0			

Bit 7 GIE: Habilitación global de Interrupciones

1 = Habilita todas las interrupciones

0 = Deshabilita todas las interrupciones

Bit 6 EEIE: Habilitación de la interrupción de memoria EEPROM

1 = Interrupción habilitada

0 = Interrupción deshabilitada

Bit 5 TOIE: Habilitación de la interrupción del TMR0

1 = Habilita la interrupción de TMR0

0 = Deshabilita la interrupción de TMR0

Bit 4 INTE: Habilitación de la interrupción externa RB0/INT

1 = Habilita la interrupción externa RB0/INT

0 = Deshabilita la interrupción externa RB0/INT

Bit 3 RBIE: Habilitación de la interrupción del puerto B

1 = Habilita la interrupción por cambio en el puerto B

0 = Deshabilita la interrupción por cambio en el puerto B

Bit 2 TOIF: Indicador de que el TMR0 ha rebosado (overflow)

1 = El TMR0 ha rebosado se borra por software

0 = El TMR0 no ha rebosado

Bit 1 INTF: Indicador de estado de una interrupción externa RB0/INT

1 = Ocurrió una interrupción por RB0/INT, debe ser borrada por software

0 = No ha ocurrido interrupción externa

Bit 0 RBIF: Indicador de cambio de estado de bits 4-7 en puerto B

1 = Uno o más de los pines RB4-RB7 ha cambiado, debe borrarse por SW

0 = Ninguno de los pines RB4-RB7 ha cambiado de estado



3 INTERRUPCIONES

Si se escribe este bit y se pone GIE = 0, todas las interrupciones están prohibidas y el procesador no admite ninguna. En cambio, si escribimos GIE = 1 el procesador admite cualquier interrupción que se produzca y que el programador haya habilitado para que actúe. Es decir, cada interrupción dispone de un bit particular de permiso que también debe habilitarse si se quiere que cuando la causa de la interrupción se produzca el procesador lance las fases de su desarrollo. El registro INTCON es un registro de lectura / escritura que contiene los bits de habilitación de las fuentes de interrupción. Las banderas de interrupción se ponen en alto cuando una interrupción ocurre, sin tener en cuenta el estado de su correspondiente bit de habilitación o el permiso global de interrupciones, GIE (INTCON bit 7).

Obsérvese que en cuanto se admite una interrupción el procesador pone automáticamente GIE = 0, para evitar que cuando está atendiendo a una interrupción se produzcan otras nuevas. Al completarse la rutina de atención a la interrupción y ejecutarse la última instrucción RETFIE, que es la de retorno de la rutina de interrupción, el bit GIE vuelve a tomar el valor 1 de manera automática, para que al retomar el programa principal queden habilitadas las interrupciones y puedan atenderse. Cabe señalar que los señalizadores de interrupción deben ponerse a 0 por programa antes del retorno de la interrupción y son operativos aunque la interrupción esté prohibida con su bit de permiso correspondiente. En la figura 3.2. se ofrece el esquema de la lógica de control que origina la interrupción.

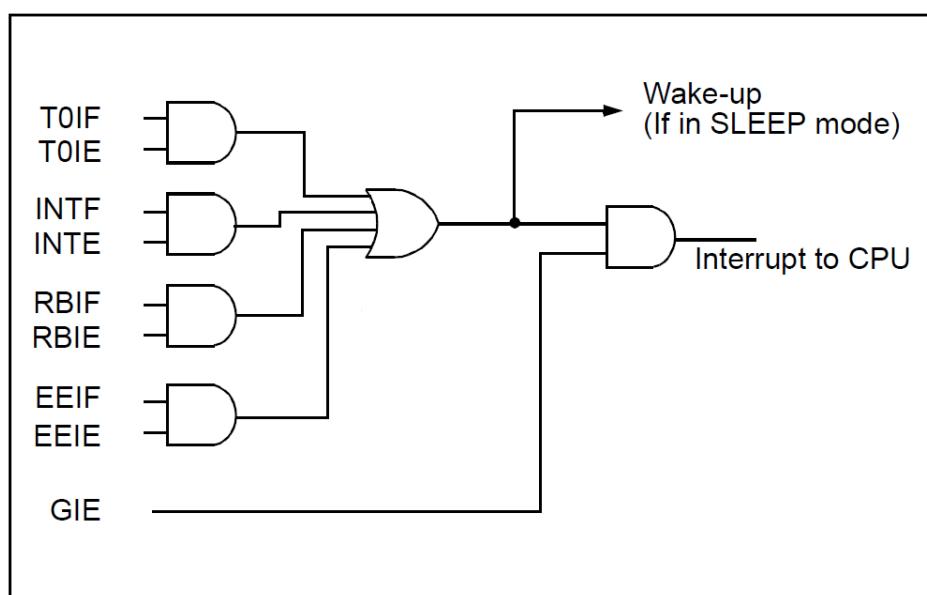


Figura 3.2. Lógica de control para la generación de una interrupción en los PIC16X8X.



2.2. Interrupción externa INT

La fuente de interrupciones externa INT es muy importante para atender eventos externos en tiempo real. Cuando la línea RB0 / INT se hace una petición de interrupción el bit INTF del registro INTCON se pone a “1” de forma automática y, si el bit GIE está a “1”, se pone en marcha el mecanismo ya comentado de la interrupción. Mediante el bit INTDEG del registro OPTION es seleccionado el flanco activo de RB0 / INT, ya que con éste puesto a “1” el flanco activo es el ascendente y cuando está a “0” el flanco activo es el descendente.

El programa de atención a la interrupción antes de regresar al programa principal debe borrar el flag INTF, puesto que en caso contrario al ejecutar la instrucción de retorno de interrupción RETFIE se volverá a desarrollar el mismo proceso de interrupción.



¿ Sabías que . . . ?

Los teclados matriciales aplican las interrupciones del PORTB, de esta forma cuando una tecla es activada la respuesta es en tiempo real y de inmediato.

De las cuatro posibles causas de interrupción que admite el PIC16F84, hay una que le permite estar en contacto permanente con los acontecimientos que ocurren en el mundo exterior. Esta interrupción está soportada por el pin 6 del microcontrolador, y se llama RB0/INT. Se trata de un pin multifunción que puede actuar de dos formas diferentes.

En primer lugar puede funcionar como línea de entrada / salida digital (RB0), también puede actuar como línea de petición de interrupción externa (INT). Para que este pin pueda recibir las peticiones externas de interrupción hay que habilitarla para tal trabajo, para lo cual hay que comenzar poniendo a 1 el bit de permiso global de interrupciones GIE. Además, también habrá que habilitar esta interrupción poniendo a 1 el bit INTE. Tanto GIE como INTE se ubican en el registro INTCON.



3 INTERRUPCIONES

Mediante el bit 6, llamado INTEG, del registro OPTION se puede seleccionar cuál será el flanco activo en RB0 / INT. Si se desea en flanco ascendente se escribe un 1 en dicho bit, y si se desea que sea el flanco descendente se escribe un 0.

Dentro de la rutina de interrupción y antes de retornar al programa principal, el programador debe encargarse de poner el señalizador INTF = 0, pues de lo contrario, al ejecutarse la instrucción de retorno RETFIE volveríamos a repetir el proceso de interrupción, ya que el flag INTF se encontraría a 1.

2.3. Interrupción del TMR0

El temporizador TMR0 es un contador ascendente de 8 bits, cuando TMR0 se desborda y pasa del valor FFh al 00h, el señalizador T0IF se pone automáticamente a 1, en el caso del TMR0 su valor se incrementa al ritmo de los impulsos de reloj que se aplican por el pin RA4 / T0CKI o desde el oscilador principal interno con la frecuencia Fosc / 4. Cuando el valor del TMR0 pasa de FFh a 00h el señalizador T0IF, que es el bit 2 de INTCON, se pone a 1 automáticamente.



¿ Sabías que . . . ?

El prescaler del TMR0 indica de cuanto en cuanto se lleva el conteo, puede ser de 2 en 2, de 4 en 4, de 256 en 256, son en total 8 opciones y se configura en el registro OPTION_REG.

Para que el procesador sepa cuando se desborda el TMR0 debe explorar cada poco tiempo el estado del señalizador T0IF. El chequeo de dicho bit supone una constante atención y pérdida de rendimiento. Para conocer inmediatamente cuándo se ha desbordado el TMR0 es mucho más eficaz generar una interrupción cuando ese hecho ocurra. Para ello es necesario habilitar la interrupción del temporizador, lo cual implica inicialmente poner el bit GIE = 1, que es el permiso global de interrupciones, y después activar el bit de permiso de la interrupción del TMR0, o sea, escribir un 1 en T0IE. En la figura 3.3. se muestra el diagrama a bloques de la interrupción por desbordamiento del temporizador TMR0.

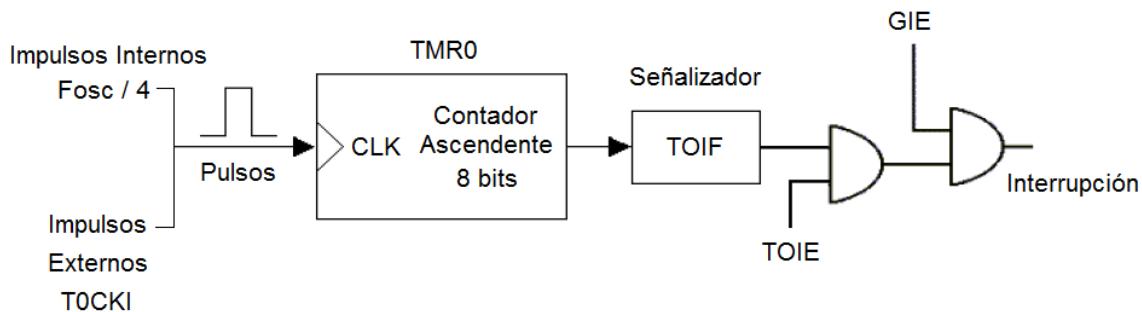


Figura 3.3. Diagrama de bloques de la Interrupción por desbordamiento del TMR0.

2.4. Interrupción del PORTB

Para activar la interrupción por cambio de nivel en los pines RB7:RB4 los bits RBIE y GIE del registro INTCON deben de estar a “1”, en estas condiciones cuando se produce un cambio de nivel en cualquiera de las líneas RB7 a RB4 se activa el flag RBIF del registro INTCON. A continuación se expone un programa ejemplo (teclado.asm) para la tarjeta del PIC, este programa comprueba el funcionamiento de la interrupción por cambio de estado de una línea de la parte alta del Puerto B, por ejemplo cada vez que presiona una tecla el programa automáticamente decodifica el valor de la tecla y lo muestra en un display de siete segmentos, este programa es parte del proyecto 8.

Una de las cuatro causas que producen la interrupción del PIC16F84 es el cambio del estado lógico en alguna de las cuatro líneas de más peso del puerto B (RB7:RB4). Si en un momento una de las mencionadas líneas, que actúan como entradas digitales, cambia el estado lógico que recibe del mundo exterior, el señalizador RBIF se pone a 1, y si el bit de permiso de esta interrupción también está activo (RBIE = 1) se genera una interrupción, para que esta ocurra es necesario que el bit de permiso global de interrupciones, GIE valga 1, y además, que el bit de permiso particular de la interrupción por cambio de estado en RB7:RB4 (RBIE) también esté a 1.

La razón por la que existe esta extraña causa de interrupción es la de atender a los teclados matriciales, tan usuales en las aplicaciones con microcontroladores. El teclado es un periférico de entrada de información fácil de manejar, de reducido tamaño y economía, tal como se muestra en la figura 3.4.



3 INTERRUPCIONES

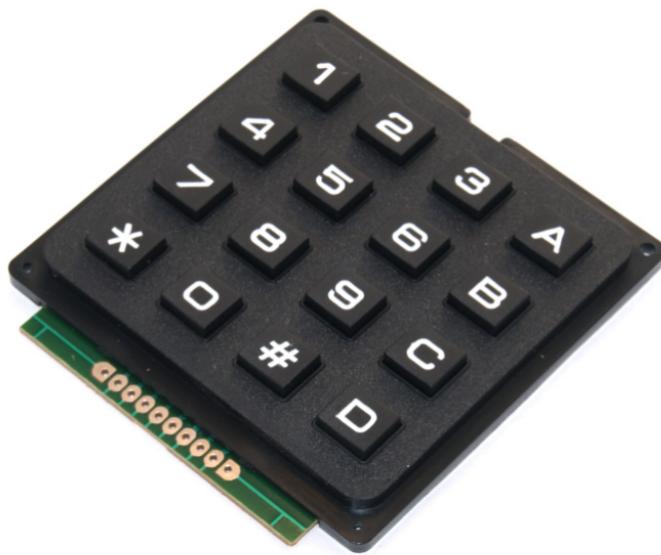


Figura 3.4. Teclado matricial de 4x4, 4 filas por 4 columnas, requiere de las interrupciones del PORTB para decodificar las teclas activas.

2.5. Interrupción de la EEPROM

El tiempo típico que tarda en desarrollarse una operación de escritura en la EEPROM de datos de los PIC16X8X es de 10 ms, que es considerable comparado con la velocidad a la que el procesador ejecuta instrucciones.

Para asegurarse que se ha completado la escritura y puede continuarse con el flujo de control del programa es aconsejable manejar la interrupción que se origina al finalizar la escritura, que pone automáticamente el señalizador EEIF a 1, y se autoriza siempre que los bits de permiso EEIE = 1 y GIE = 1.

Cuando se describió el proceso de escritura de la EEPROM de datos se indicó que se usaba un registro no real para asegurar la misma. Se trataba del EECON2, en el que se grababan dos valores, el 55h y el AAh. Durante la escritura de este registro debe prohibirse la aceptación de interrupciones para salvaguardar la operación de escritura, por eso en ese módulo se pone GIE = 0, tal como se indica en el siguiente segmento de programa orientado a escribir la memoria EEPROM. Se supone que la dirección a acceder ya se ha cargado en el registro EEADR y el dato a escribir en EEDATA.



¿ Sabías que . . . ?

La memoria EEPROM es un tipo de memoria ROM que puede ser programada, borrada y reprogramada eléctricamente, a diferencia de la EPROM que ha de borrarse mediante un aparato que emite rayos ultravioleta.

```

bsf      STATUS, rp0    ;Selecciona el banco 1
bcf      INTCON, gie   ;GIE = 0. Prohibe interrupciones
bsf      INTCON, eeie   ;Permite la interrupción de la EE
                  ;PROM
bsf      EECON1, wren  ;Permiso de escritura de la EE
                  ;PROM
movlw   0x55
movwf   EECON2        ;Se escribe en EECON2 el dato 55h
movlw   0xAA
movwf   EECON2        ;Se escribe AAh en EECON2
bsf      EECON1, wr    ;Comienza el proceso de escritura
bsf      INTCON, gie   ;Permiso de interrupciones

```

En los PIC16C84 y PIC16F8X se puede leer y escribir la EEPROM de datos aunque se haya protegido el código. En los PIC16CF8X, que disponen de memoria ROM para el código, existen dos bits para el código de protección: uno dedicado a la ROM de código y el otro a la EEPROM de datos.

Proyecto 8: Teclado matricial 4x4

A) Planteamiento del programa:

El programa consiste en decodificar las teclas del teclado matricial de 4x4, aplicando lógica programada y un display de siete segmentos ánodo común, los mismos que permiten visualizar las teclas del 0 al 9, los códigos en formato binario para cada número serán almacenados en una tabla en la memoria del PIC, el funcionamiento de este programa debe ajustarse a las siguientes condiciones:



3 INTERRUPCIONES

1. El valor inicial del display es cero
 2. El tiempo de barrido es de 4us
 3. Se aplicarán las interrupciones del PORTB
 4. Si se presionan dos teclas hace efecto la primera en ser activada

B) Esquema electrónico:

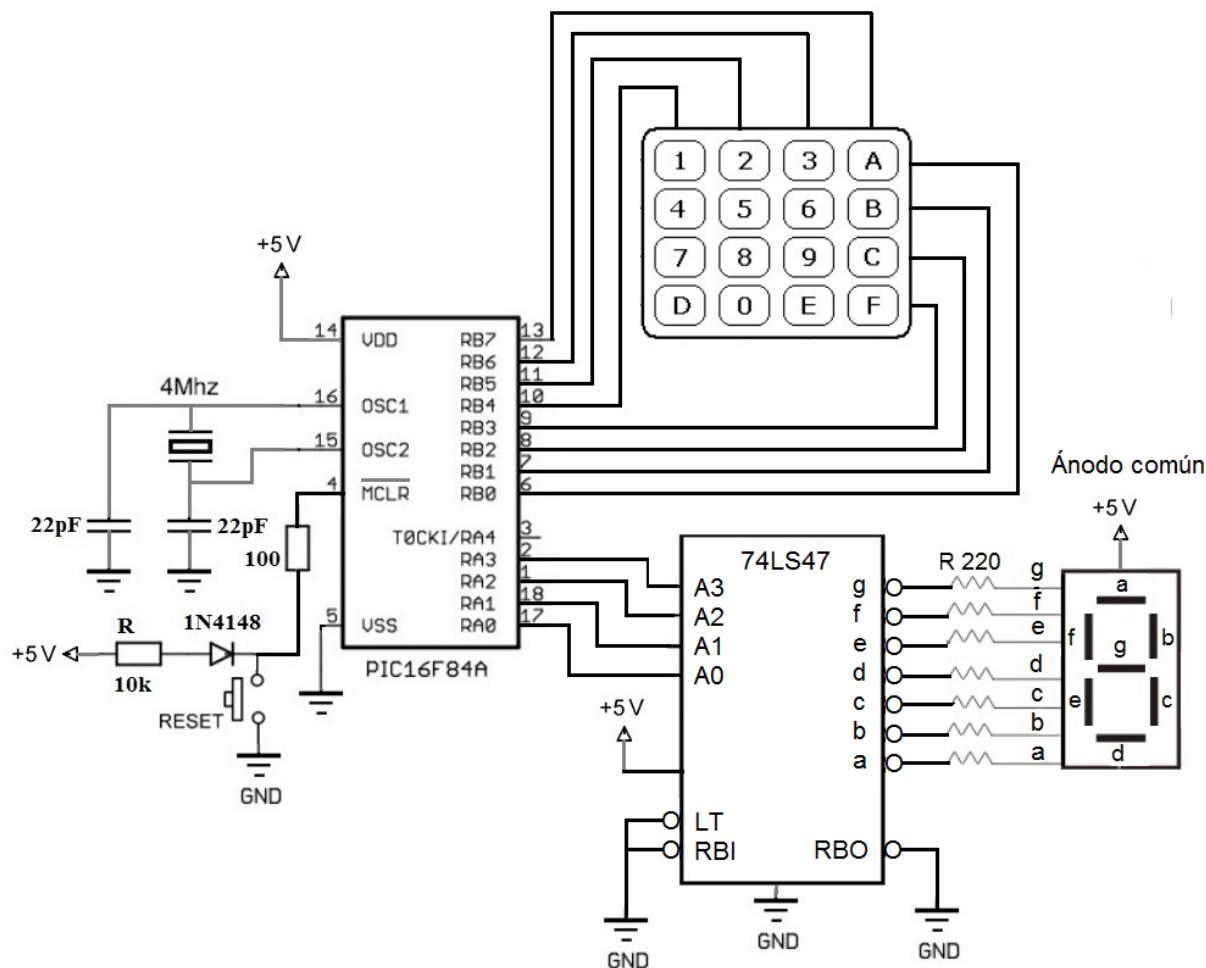


Figura 3.5. Circuito electrónico para el proyecto 8.



C) Código del programa:

```
;-----  
;Nombre del programa: teclado.asm  
;Autor: Ing. David Apaza Condori  
;Ensamblador: MPLAB V8.14  
;  
-----  
__config H'3FF1'  
#include "p16F84.inc"  
  
dato1 EQU 0x0C  
dato2 EQU 0x0D  
dato3 EQU 0x0E  
dato4 EQU 0x0F  
  
  
ORG 0x00  
    goto inicio  
ORG 0x04  
    goto inter  
  
  
inicio bsf STATUS, 5  
        movlw b'11110000'  
        movwf TRISB  
        movlw b'00000000'  
        movwf TRISA  
        movlw b'10000111'  
        movwf OPTION_REG  
        movlw b'10001000';Config. del INTCON  
        movwf INTCON  
        bcf STATUS, 5 ;ir al banco 0  
        movlw b'10001000';Config. del INTCON  
        movwf INTCON  
        clrf PORTB  
        clrf PORTA  
        clrf dato3  
;  
-----
```



3 INTERRUPCIONES

```
;<-----  
lab_00    movlw      b'00001111'  
           movwf      PORTB  
           goto      lab_00  
inter     movlw      b'00000001'  
           movwf      dato1  
loop      movf      dato1,0  
           movwf      PORTB  
           btfsc      PORTB,4  
           goto      col_01  
           btfsc      PORTB,5  
           goto      col_02  
           btfsc      PORTB,6  
           goto      col_03  
           btfsc      PORTB,7  
           goto      col_04  
           btfsc      dato1,3  
           goto      volver  
           rlf       dato1,1  
           goto      loop  
col_01    movlw      0x01  
           movwf      dato2  
           goto      dato  
col_02    movlw      0x02  
           movwf      dato2  
           goto      dato  
col_03    movlw      0x03  
           movwf      dato2  
           goto      dato  
col_04    movlw      0x00  
           movwf      dato2  
           goto      dato  
;<-----
```



```
-----;  
dato    btfsc    dato1,0  
        goto     fila1  
        btfsc    dato1,1  
        goto     fila2  
        btfsc    dato1,2  
        goto     fila3  
        btfsc    dato1,3  
        goto     fila4  
        goto     volver  
fila1   goto     volver  
fila2   movlw    d'3'  
        addwf    dato2,1  
        goto     volver  
fila3   movlw    0x06  
        addwf    dato2,1  
        goto     volver  
fila4   clrf     dato2  
        goto     volver  
volver  movf     dato2,0  
        movwf    PORTA  
        bcf      INTCON,1  
        bcf      INTCON,0  
        retfie  
  
END  
-----;
```



3 INTERRUPCIONES

D) Simulación del programa:

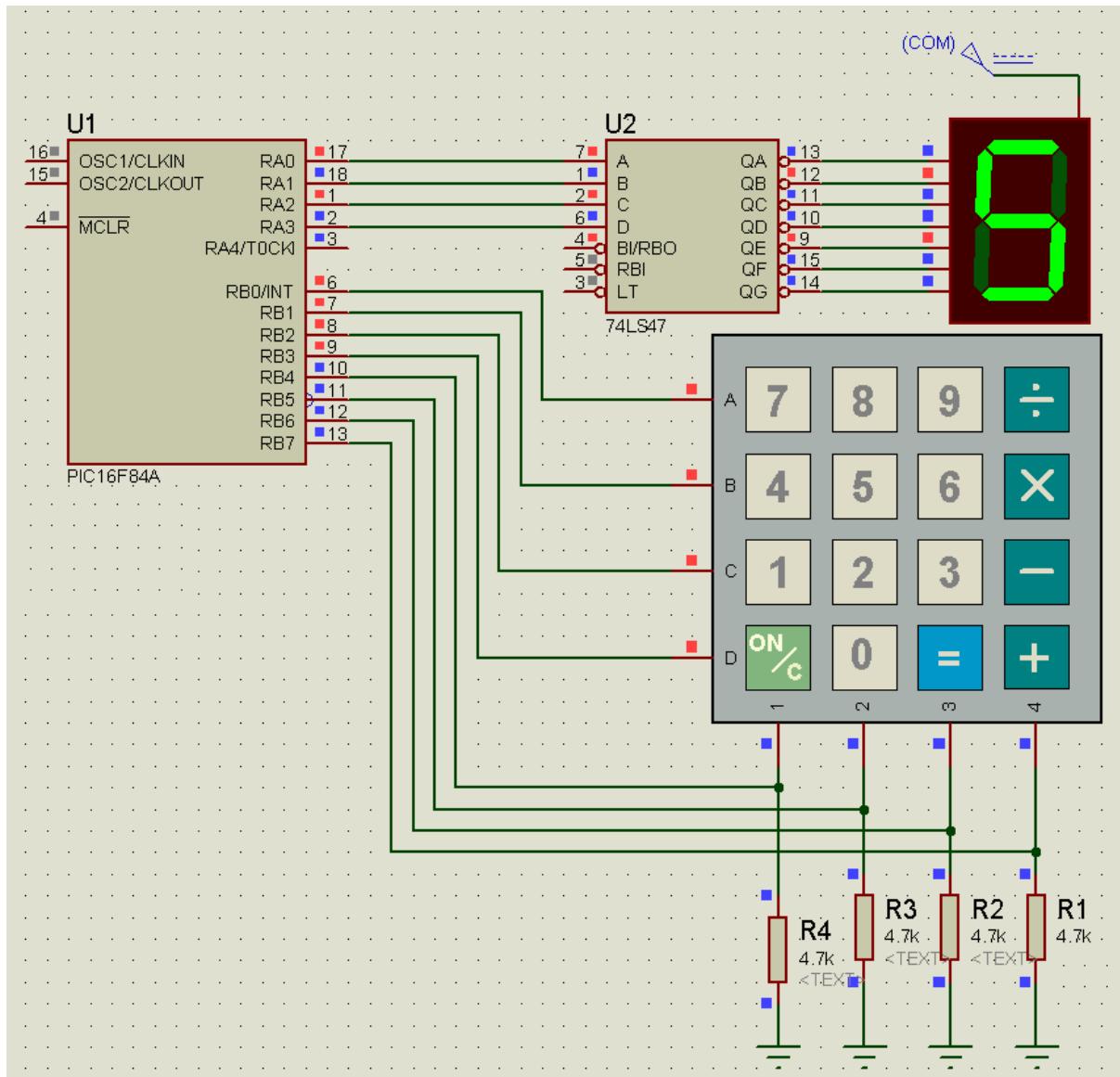


Figura 3.6. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



Actividades

A) Comentar el siguiente código de programa:

```
ORG      0x00
        goto    inicio

ORG      0x04
        goto    inter

inter   btfsc   INTCON,RBIF
        goto    Int_01
        retfie

Int_01  incf    PORTA
        bcf     INTCON,RBIF
        retfie

inicio  bsf     STATUS,RP0
        movlw   b'11110000'
        movwf   TRISA
        movlw   b'11111111'
        movwf   TRISB
        bcf     OPTION_REG,NOT_RBPU
        bcf     OPTION_REG,INTEDG
        bcf     STATUS,RP0
        movlw   b'00000000'
        movwf   PORTA
        bsf     INTCON,GIE
        bsf     INTCON,RBIE

fin     goto    fin

End
```



3 INTERRUPCIONES

B) Responder las siguientes preguntas o cuestiones:

1. Una forma de recordar la dirección de retorno al programa interrumpido podría ser guardar el contador de programa en una localización de la memoria de datos del microcontrolador como un dato más, sin embargo, se prefiere utilizar la pila para ello. ¿Por qué?

2. Comente la secuencia de etapas que transcurren al atender una solicitud de interrupción en un microcontrolador.

3. Comente la estructura general de la subrutina que atiende una interrupción, mostrada en el proyecto 8. ¿Qué la diferencia de una subrutina “convencional”?

4. ¿Qué alternativas hay para informar a la CPU de la dirección donde comienza la subrutina de atención a la interrupción?

5. ¿Qué estado toma (habilitado o inhabilitado) el sistema de interrupción de un PIC después de una acción de reset?

6. Atendiendo a que las interrupciones pueden ser enmascarables o no enmascarables, y fijas o vectorizadas, clasifique las interrupciones de los microcontroladores PIC de gama media.

7. ¿Qué registros de funciones especiales están relacionados con las interrupciones en un PIC de gama media?



8. ¿Cuántos bits están asociados a cada fuente de interrupción en un PIC y qué función tienen?
-
-

9. Supongamos que el sistema de interrupción del PIC está habilitado. Si se produce una solicitud de interrupción y ésta llega a la CPU (pues está habilitada), el sistema de interrupción se inhabilita. ¿Cómo se vuelve a habilitar el sistema de interrupción?
-
-

10. ¿Qué diferencias hay entre las instrucciones `return` y `retfie`?
-
-

11. ¿Qué condiciones deben existir en el sistema de interrupción del PIC para que una solicitud de interrupción cause efectivamente la interrupción del programa en curso?
-
-





3 INTERRUPCIONES



CAPÍTULO 4

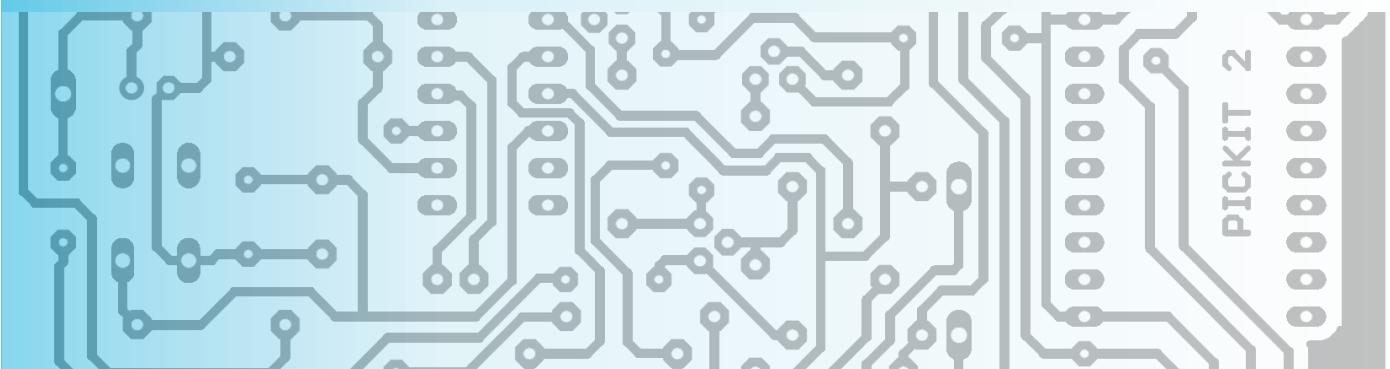
EL MÓDULO LCD

CONTENIDO DEL CAPÍTULO

- 1. Introducción
- 2. Tipos de memorias del LCD
 - 2.1. DD RAM (Display Data RAM)
 - 2.2. CG RAM (carácter generator RAM)
- 3. Periféricos del LCD
- 4. Inicialización del LCD
 - 4.1. Secuencia de inicialización del módulo LCD
- 5. Comandos del LCD
 - 5.1. Comandos de control
- Proyecto 9: Mensaje simple en el LCD
- Proyecto 10: Mensajes Alternados
- Proyecto 11: Mensaje en dos líneas
- Proyecto 12: Mensaje en movimiento

APRENDEREMOS A:

- Identificar el módulo LCD, sus periféricos, sus características eléctricas.
- Identificar los comandos aplicados a los módulos LCD, sus tiempos de respuesta.
- Configurar correctamente el módulo LCD.
- Utilizar correctamente los registros internos del módulo LCD y sus bits relacionados.





4 EL MÓDULO LCD

1. Introducción

Antes de aparecer los módulos LCD, se utilizaban los Displays de siete segmentos para poder mostrar la información. Tenían una gran limitación de poder mostrar los caracteres alfa numéricos y símbolos especiales, también consumían demasiada corriente y ocupaban demasiado espacio físico. Posteriormente aparecieron otros tipos de displays mas complejos que podían mostrar algunos caracteres y símbolos; pero tenían de igual manera mucho consumo de corriente y espacio físico. Finalmente aparecieron los módulos LCD o pantallas de cristal líquido la cual tiene la capacidad de mostrar cualquier carácter alfa numérico.

Estos dispositivos ya vienen con su pantalla y toda la lógica de control preprogramada en la fabrica y lo mejor de todo es que el consumo de corriente es mínimo y no hace falta realizar tablas especiales como se hacia anteriormente con los displays de siete segmentos. Las aplicaciones de los módulos LCD son infinitas ya que pueden ser aplicados en la informática, comunicaciones, telefonía, instrumentación, robótica, automóviles, equipos industriales, etc.

La definición mas clara de un LCD es: una pantalla de cristal liquido que visualiza unos ciertos caracteres. Para poder hacer funcionar un LCD, debe estar conectado a un circuito impreso en el que estén integrados los controladores del display y los pines para la conexión del display. Sobre el circuito impreso se encuentra el LCD en sí, rodeado por una estructura metálica que lo protege.

En total se pueden visualizar 2 líneas de 16 caracteres cada una, es decir, $2 \times 16 = 32$ caracteres. A pesar de que el display sólo puede visualizar 16 caracteres por línea, puede almacenar en total 40 por línea. Es el usuario el que especifica qué 16 caracteres son los que se van a visualizar. Tiene un consumo de energía de menos de 5mA y son ideales para dispositivos que requieran una visualización pequeña o media. El LCD dispone de una matriz de 5x8 puntos para representar cada carácter. En total se pueden representar 256 caracteres diferentes, 240 caracteres están grabados dentro del LCD y representan las letras mayúsculas, minúsculas, signos de puntuación, números, etc. Existen 8 caracteres que pueden ser definidos por el usuario.



Figura 4.1. El módulo LCD de 2x16, solo tiene 2 líneas para visualizar el mensaje, tiene en total 16 pines para conectarse a otros periféricos.

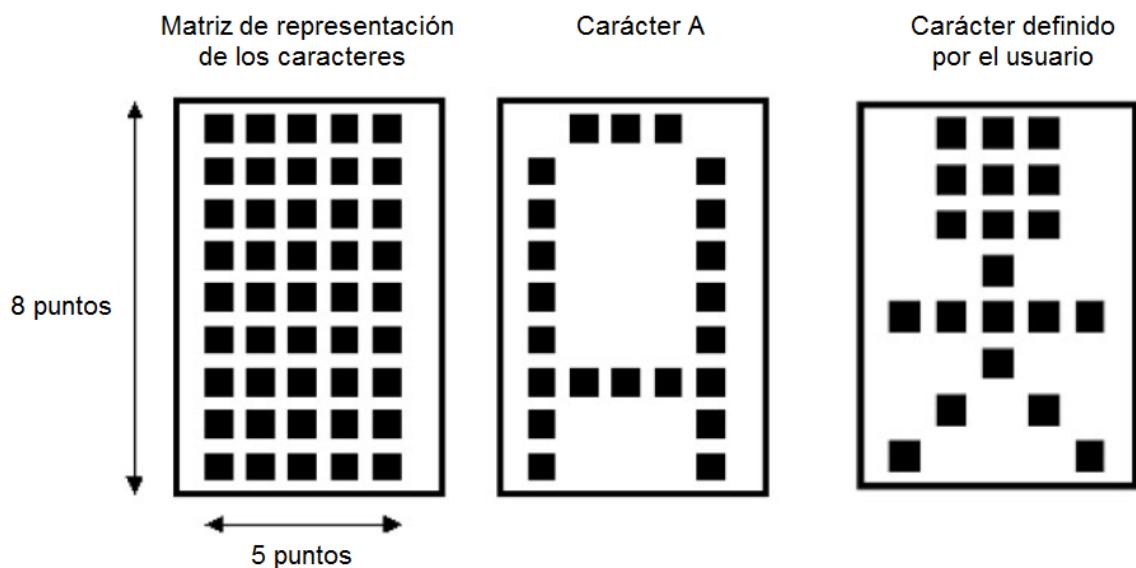


Figura 4.2. Matriz de representación de los caracteres en cada celda de la pantalla del módulo LCD.



4 EL MÓDULO LCD

En la siguiente tabla se muestran los caracteres más importantes que es capaz de representar la pantalla LCD. No están representados los caracteres correspondientes a los códigos desde el \$80 hasta el \$FF, que corresponden a símbolos extraños. Los códigos comprendidos entre el 0 y el 7 están reservados para que el usuario los defina.

Tabla 4.1. Caracteres que es capaz de representar en módulo LCD.

xxxx	Upper 4 Bits Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	CG RAM (1)			Ø	ø	P	^	P				-	‐	𩫂	𩫃	ø	p
0001	(2)			!	!	A	Q	a	q			▪	՞	՞	՞	՞	՞
0010	(3)			"	"	Z	B	R	b	r		՝	イ	ツ	メ	՞	՞
0011	(4)			#	#	C	S	C	s			՝	ウ	テ	モ	՞	՞
0100	(5)			\$	\$	4	D	T	d	t		、	エ	ト	タ	՞	՞
0101	(6)			%	%	5	E	U	e	u		・	オ	ナ	ユ	՞	՞
0110	(7)			&	&	6	F	V	f	v		ヲ	カ	ニ	ヨ	՞	՞
0111	(8)			'	'	7	G	W	g	w		ア	キ	ヌ	ラ	՞	՞
1000	(1)			((8	H	X	h	x		イ	カ	ネ	リ	՞	՞
1001	(2)))	9	I	V	i	y		タ	ル	ル	‐	՞	՞
1010	(3)			*	*	:	J	Z	j	z		エ	コ	ハ	レ	՞	՞
1011	(4)			+	+	;	K	C	k	{		オ	サ	ヒ	ロ	՞	՞
1100	(5)			,	,	<	L	¥	l	}		タ	シ	フ	ワ	՞	՞
1101	(6)			-	=	M	J	M)		ュ	ズ	ヘ	ン	է	՞	
1110	(7)			.	.	>	N	^	n	→		Յ	է	ホ	՞	՞	
1111	(8)			/	/	?	O	_	o	*		Ա	Ն	Ր	՞	՞	



En la actualidad existe una gran variedad de versiones, clasificadas en dos grupos. El primer grupo esta referido a los módulos LCD de caracteres (solamente se podrán presentar caracteres y símbolos especiales en las líneas predefinidas en el modulo LCD) y el segundo grupo esta referido a los módulos LCD matriciales (se podrán presentar caracteres, símbolos especiales y gráficos). Los módulos LCD varían su tamaño físico dependiendo de la marca.

2. Tipos de memorias del LCD

Un dispositivo LCD dispone de dos tipos de memorias ambas independientes. Estas memorias se denominan DD RAM y CG RAM.

2.1. DD RAM (Display Data RAM)

Es la memoria encargada de almacenar los caracteres de la pantalla que se esten visualizando en ese momento, o bien, que esten en una posición no visible. El display tiene una capacidad de 2 líneas horizontales por 40 caracteres cada una, de los cuales solo serán visibles 2 líneas de 16 caracteres cada una. La DDRAM tiene un tamaño de $2 \times 40 = 80$ bytes. Una vez conocida la disposición de almacenamiento del display, es fácil pensar en un display de 2 líneas de 40 caracteres sobre el que se desplaza una ventana de 2 líneas por 16 caracteres como se muestra en la figura 4.3.

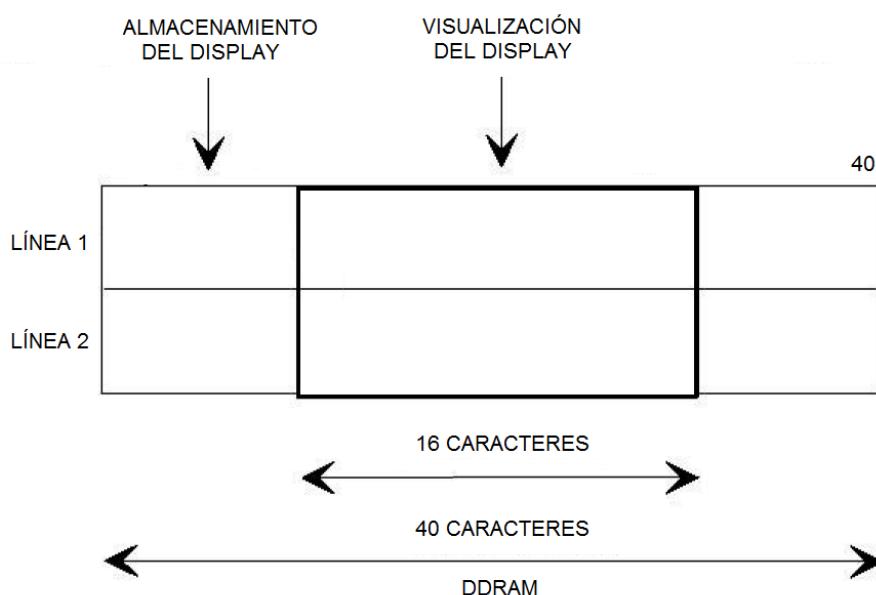


Figura 4.3. Representación de la memoria DD RAM del LCD.



4 EL MÓDULO LCD

El orden de los caracteres sería empezando de izquierda a derecha, de tal modo que el carácter 1 sería el primero de la izquierda y el 40 sería el situado más a la derecha. Para localizar los caracteres se utilizan dos coordenadas (X,Y) siendo Y el valor vertical comprendido entre 1 y 2 (valor de línea) y X el valor horizontal (de 1 hasta 40) que direccionará el carácter.

Si por ejemplo queremos visualizar ALUMNOS DE LA UNIVERSIDAD AUTÓNOMA, pero en el display solo se visualizaría ALUMNOS DE LA UN que correspondería a los 16 caracteres visibles. Para visualizar toda la información almacenada en el display, podremos tratar al recuadro de 2 líneas por 16 caracteres como si se tratara de una ventana móvil. Cuando inicializamos el LCD la pantalla tendría un aspecto de 2x16.

El mapa de memoria de la DD RAM esta constituido por dos bloques de 40 bytes cada uno. El primer bloque corresponde con los 40 caracteres de la línea 1 y el segundo bloque con los de la línea 2. Las direcciones en hexadecimal \$00-\$27 están asociadas con las posiciones de almacenamiento del display (1,1) a (40,1), y las direcciones \$40-\$67 con las de almacenamiento (1,2) a (40,2).



¿ Sabías que . . . ?

El microcontrolador 16F84 también puede trabajar con el módulo LCD, utilizando el PORTB para el bus de datos y el PORTA para el bus de control.

2.2. CG RAM (carácter generator RAM)

La CG RAM contiene los caracteres que pueden ser definidos por el usuario, es decir que pueden ser personalizados. Está formada por 64 posiciones, con direcciones \$00-\$3F. Cada posición es de 5 bits. La memoria esta dividida en 8 bloques que corresponden a los posibles caracteres creados por el usuario que van del 0 al 7, cada carácter esta constituido por una matriz de 5 columnas por 8 filas. De este modo un 1 indica un punto de la matriz encendido, y un 0 apagado como se ve en la siguiente figura.

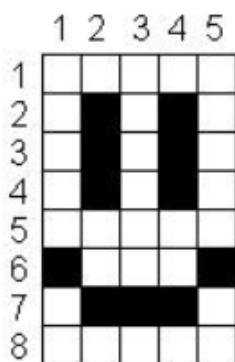


A continuación si quisiéramos almacenar este carácter en la posición 0 de la CG RAM, tendríamos que almacenar cada una de las líneas de 5 bits en las direcciones de la CG RAM que van de la \$00 hasta la \$07 como se ve en el mapa de memoria CG RAM.

MEMORIA CG RAM

A Custom 5x8 Pixel Character:

Image Coding:



Binary Coding:

```

0b000 00000
0b000 01010
0b000 01010
0b000 01010
0b000 00000
0b000 10001
0b000 01110
0b000 00000
  
```

1 = Black, 0 = White

Figura 4.4. Matriz de 5x8 que representa un carácter especial creado por el usuario.

3. Periféricos del LCD

Los pines 1 y 2, son los utilizados para la alimentación del módulo LCD. La tensión utilizada es de 5 voltios.

El pin 3 se utiliza para ajustar el contraste de la pantalla LCD. Por medio de un potenciómetro regularemos la intensidad de los caracteres, a mayor tensión mayor intensidad. Se suele utilizar un potenciómetro de unos 10 o 20 k, que regulará la misma tensión que se utiliza para la alimentación.

El pin 4 se utiliza para indicar al bus de datos si la información que le llega es una instrucción o por el contrario es un carácter. Si RS = 0 indicará que en el bus de datos hay presente una instrucción, y si RD = 1, indicará que tiene un carácter alfanumérico.



4 EL MÓDULO LCD

El pin 5 es de escritura o lectura. Si esta a 0 el módulo escribe en pantalla el dato que haya en el bus de datos, y si está a 1 leeremos lo que hay en el bus de datos.

El pin 6 es el indicado de hacer que el módulo LCD funcione, o por el contrario no acepte ordenes de funcionamiento. Cuando E = 0 no se podrá utilizar el display y cuando E = 1 se podran transferir datos y realizar las demás operaciones.

Los pines del 7 al 14 son los del bus de datos.



Figura 4.5. Periféricos del módulo LCD, cada pin cumple una función importante para el funcionamiento del LCD.

Las conexiones entre un microcontrolador y una pantalla LCD, serán la del bus de datos de 8 bits y las de los pines de control, es decir, E, RS y R/W. Debido a esto, el microcontrolador deberá utilizar 11 bits para controlar la pantalla. El bus de datos de un módulo LCD puede funcionar como un bus de 8 bits o como un bus multiplexado de 4 bits. Esto permitiría ahorrar 4 patillas al microcontrolador utilizado pero complicaría mucho el control del LCD ya que habría que multiplexar y demultiplexar los datos. Si se realizara la multiplexación, solo



utilizariamos los pines del D4 al D7, y primero se retransmitirían los bits más significativos y luego los menos significativos. El bus de control está formado por las señales RS, R/W y E.

Tabla 4.2. Diferentes pines que posee el LCD.

PIN	SÍMBOLO	Nombre y función
1	VSS	GND (Tierra 0V)
2	VDD	Alimentación +5V
3	VO	Ajuste del contraste
4	/RS	Selección DATO/CONTROL
5	/RW	Lectura o 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
15	LED+	Ánodo de LED backlight
16	LED-	Cátodo de LED backlight

4. Inicialización del LCD

4.1. Secuencia de inicialización del módulo LCD

La secuencia de inicialización para cualquier módulo LCD es imprescindible, y debe ser operada en la pantalla del módulo. La inicialización se basa en una serie de instrucciones introducidas por nosotros y posteriormente procesadas por el módulo LCD para su funcionamiento normal. Las instrucciones que están dentro de la inicialización solamente se ejecutan después que se enciende el módulo LCD y no podrán ser cambiadas posteriormente. Ejemplos de instrucciones que solo podrán ejecutarse cuando inicializamos el módulo LCD:

1. Selección de la longitud del bus de datos (4 Bits / 8 Bits).
2. Activar el número de líneas que se visualizaran en el módulo LCD.
3. Encender el módulo LCD.



4 EL MÓDULO LCD

Las siguientes instrucciones también podrán ser colocadas en la inicialización, con la diferencia que podrán ser cambiadas en cualquier parte del programa.

1. Mantener el mensaje fijo y desplazar el cursor.
2. Desplazar el mensaje y mantener el cursor fijo.
3. Hacer que el carácter señalado parpadee o no.

El módulo ejecuta automáticamente una secuencia de inicio interna en el instante de aplicarle la tensión, y hay unos requisitos de estabilidad. El tiempo mínimo que tarda en estabilizarse la tensión entre 0,2 y 4,5 voltios debe estar comprendido entre 0,1 y 10 milisegundos. Por otro lado el tiempo mínimo de desconexión debe ser de 0,1 milisegundos antes de volver a conectar.

La secuencia de inicio ejecutada es la siguiente:

1. Se ejecuta el comando CLEAR DISPLAY borrando la pantalla. El flag BUSY se mantiene a “1” (ocupado) durante 15ms hasta que finaliza la inicialización.
2. Se ejecuta el comando FUNCTION SET, que establece el interfaz con el Bus de datos. Se elige por defecto el tamaño del bus de datos a 8 bits (DL = 1) y el número de renglones del display en 1 (N = 0).
3. Se ejecuta el comando DISPLAY ON/OFF CONTROL, que hace que el display que en OFF (D = 0); también cursor en OFF (C = 0) y sin parpadeo del cursor en (B = 0).
4. Se ejecuta el comando ENTRY MODE SET, que establece la dirección de movimiento del cursor con autoincremento del cursor (I/D = 1) y modo normal, no desplazamiento, del display (S = 0).

Si la conexión de la alimentación no reúne las condiciones que exige el módulo LCD, habría que realizar la secuencia de inicialización por software. En cualquier caso, es importante enviar al LCD la primera instrucción de trabajo después de que hayan transcurrido 15ms, para completar dicha secuencia de inicialización.



Para el caso de 8 bits no hay ningún problema, sin embargo el caso de 4 bits es un poco más complejo. Después de encender el LCD aparecerá la linea superior un poco más oscura que la inferior. Esto quiere decir que el display no ha sido inicializado todavia. En el caso de 4 bits sólo se conectan los 4 bits mas significativos del LCD, dejando los otros 4 al aire. Al enviar el código 2 (Bits 0 0 1 0) el display se configura para trabajar a 4 bits. Se puede observar cómo la línea superior deja de estar más oscura que la inferior. A partir de este momento las transferencias hay que realizarlas en dos partes:

Primero se envían los 4 bits mas significativos y después los 4 bits menos significativos. Para confirmar que la transferencia es a 4 bits hay que enviar el código \$28; primero los bits 0 0 1 0 y después los bits 1 0 0 0. De aquí en adelante la inicialización es igual tanto para 8 bits como para 4, con la salvedad de que en el segundo caso hay que enviar los datos multiplexados.

Los Pines de control (E, RS y E/W) están relacionados ya que por medio de ellos podemos especificar la opción de ejecutar una instrucción o leer / escribir un dato en la pantalla o la memoria RAM; sin embargo existe una condición importante que deberá tomarse en cuenta referida directamente al tiempo necesario que se necesita para cambiar de un estado a otro en los pines de control (E, RS y R/W). En el caso de que este tiempo sea mas pequeño que el tiempo mínimo requerido, entonces el módulo LCD no tendrá el tiempo suficiente para responder a las instrucciones solicitadas por el usuario y por consecuencia se perderán los datos o instrucciones según sea el caso.

Es decir, no debemos obviar la velocidad propia del módulo LCD con la propia del microcontrolador que estemos utilizando, ya que si no se tiene en cuenta la velocidad del microprocesador y esta sobrepasa la del módulo LCD, éste último no tendrá capacidad suficiente como para ir procesando y ejecutando todo el flujo, y por lo tanto perderemos información. Para ello los programas o los circuitos electrónicos que manejan un módulo LCD deberán respetar los tiempos de respuesta.



4 EL MÓDULO LCD

Si se quiere enviar una instrucción al módulo LCD, colocamos el comando en el bus de datos (pines del 7 al 14). Posteriormente se ejecuta el diagrama de tiempo requerido para una instrucción en los pines de control. Este diagrama de tiempo es muy sencillo de entender, tan solo usted deberá colocar: el Pin RS = 0, el Pin R/W = 0 y el Pin E = 0. Luego, se cambia el estado del Pin E a 1, debiendo permanecer en dicho estado al menos durante 450ns. De no ser así el módulo LCD no podrá aceptar el comando.

En el caso de querer escribir un dato al módulo LCD, colocamos el dato en el bus (pines del 7 al 14) como antes. Y entonces ejecutamos el diagrama de tiempo requerido para escribir un dato en los pines de control. Ponemos el Pin RS = 1, el Pin R/W = 0 y el Pin E = 0. Cuando ya terminemos este proceso, se cambia el estado del Pin E = 1, debiendo permanecer en dicho estado al menos durante 450 ns. De no ser así el módulo LCD no podrá aceptar el dato.



¿ Sabías que . . . ?

Existen LCDs gráficos que permiten visualizar cualquier carácter, símbolo o figura y aplican colores en algunos casos, pero la programación es de mayor complejidad.

En el caso de querer leer un dato de la pantalla LCD o de la memoria LCD, los pines de control hay que configurarlos así; Pin RS = 1, Pin R/W = 1 y el Pin E = 0. Seguidamente cambiamos el Pin E a 1, debiendo permanecer en dicho estado al menos durante 450ns. De no ser así el módulo LCD no podrá aceptar la instrucción.

5. Comandos del LCD

La forma de controlar un LCD es a través de comandos que se envian al registro de control del LCD, seleccionado al poner la señal RS (Register Select, selecciona el registro interno que se va a leer o escribir) a nivel bajo(0).



Cuando lo que se quiere es imprimir caracteres en el display o enviar información a la CG RAM para definir caracteres se selecciona el registro de datos poniendo RS a nivel alto (1).

Existe un controlador de direcciones para la DD RAM y otro para la CG RAM, el cual contiene la dirección a la que se va a acceder. Modificando el control de direcciones es posible acceder a cualquier posición de la CG RAM como de la DD RAM.

Cada vez que se realiza una acceso a memoria, el contador de direcciones se incrementa o se decrementa automáticamente, según como se haya configurado el LCD. Al LCD le lleva cierto tiempo procesar cada comando enviado. Por ello, para que se ejecute el comando especificado es necesario asegurarse de que el comando anterior ha finalizado. Existen 2 estrategias para realizar esto:

La primera se basa en leer del display el bit ocupado. Si este bit se encuentra a 1 quiere decir que el LCD está ocupado procesando el comando anterior y por tanto no puede procesar nuevos comandos.

La segunda estrategia, menos elegante pero más comoda de implementar, consiste en realizar una pausa antes de volver a enviar el siguiente comando. Los tiempos máximos que tarda el display en procesar los comandos están especificados por el fabricante y tienen un valor típico de $40\mu s$. Si se realiza una pausa mayor o igual a esta se tiene la garantía de que el display ha terminado de ejecutar el comando.



¿ Sabías que . . . ?

El módulo LCD tiene internamente un microcontrolador especial, que se encarga de decodificar y ejecutar los comandos, de facilitar la comunicación con el PIC.



4 EL MÓDULO LCD

Tabla 4.3. Diferentes comandos para el módulo LCD.

CONTROL Y DATO	SEÑAL DE CONTROL		DATO / DIRECCIÓN																				
	RS	RW	D7	D6	D5	D4	D3	D2	D1	D0													
Borrar pantalla	0	0	0	0	0	0	0	0	0	1													
Cursor a casa	0	0	0	0	0	0	0	0	1	-													
Seleccionar modo	0	0	0	0	0	0	0	1	I/D	S													
Encender/ apagar pantalla	0	0	0	0	0	0	1	D	C	B													
Desplazar Cursor / Pantalla	0	0	0	0	0	1	S/C	R/L	-	-													
Activar función	0	0	0	0	1	D/L	N	F	-	-													
CG RAM	0	0	0	1	Dirección generador de RAM																		
DD RAM	0	0	1	Dirección de datos RAM																			
Bandera de ocupado	0	0	BF	AC																			
Escritura CG RAM/DD RAM	1	0	Escritura de dato																				
Lectura CGRAM/ DDRAM	1	1	Lectura de dato																				



5.1. Comandos de control

Los comandos se envían a través del bus de datos. Para que la pantalla los reconozca hay que poner las señales RS y R/W a nivel bajo.

1. CLEAR DISPLAY.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	0	0	0	0	0	0	0	1	1.64 ms

Borra la pantalla del módulo LCD y coloca al cursor en la primera posición que es la dirección 0. Por defecto, pone el bit I/D = 1 para auto incremento de la posición del cursor.

2. HOME.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	0	0	0	0	0	0	1	X	1.64 ms

Pone el cursor en la dirección 0. No varia el contenido de la memoria DDRAM que guarda los datos y que queda direccionada desde la posición 0.

X : Indeterminado.

3. ENTRY MODE SET.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	0	0	0	0	0	1	I/D	S	40 ms

Establece la dirección del movimiento del cursor.

S = 1, desplaza la visualización cada vez que se escribe un dato.

S = 0, funciona en modo normal.

I/D = 1, se incrementa la dirección del cursor.

I/D = 0, se decrementa la dirección del cursor.



4 EL MÓDULO LCD

4. DISPLAY ON/OFF.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	0	0	0	0	1	D	C	B	40 ms

Activa o desactiva al display y al cursor y determina si este parpadea o no.

B = 1, hay parpadeo del cursor (si esta activo).

B = 0, no hay parpadeo del cursor.

C = 1, el cursor esta activo.

C = 0, el cursor esta desactivo.

D = 1, la pantalla esta activada.

D = 0, la pantalla esta desactivada.

5. CURSOR DISPLAY SHIFT.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	0	0	0	1	S/C	R/L	X	X	40 ms

Mueve el cursor y desplaza la visualización sin cambiar el contenido de la memoria DDRAM.

R/L = 1, el desplazamiento es a la derecha.

R/L = 0, el desplazamiento es a la izquierda.

S/C = 1, desplaza la visualización.

S/C = 0, desplaza el cursor.

6. FUNCTION SET.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	0	0	1	DL	N	F	X	X	40 ms

Establece el numero de líneas, establece el ancho del bus de datos, siendo habitual que sea de 8 bits, además del formato del carácter.



F = 1, el carácter es de 5x10 pixels.

F = 0, el carácter es de 5x7 pixels.

N = 1, la presentación se hace en dos líneas.

N = 0, la presentación se hace en una línea.

DL = 1, se trabaja con bus de datos de 8 bits.

DL = 0, se trabaja con bus de datos de 4 bits.

7. SET DDRAM ADDRESS.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	1								40 ms

Establece la dirección de la DDRAM a partir de la cual todos los datos que se lean o escriban posteriormente lo harán desde esa posición, los 16 caracteres del primer renglón ocupan las direcciones 80h – 8Fh, y los del segundo renglón desde la C0h – CFh.

8. READ BUSY FLAG ADDRESS.

RS	R/W	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ej.
0	0	1	BF								40 ms

Permite leer el flag BUSY que indica si todavía se está ejecutando un comando previo en el modulo LCD, además proporciona la dirección de la CGRAM o DDRAM que se haya utilizado la ultima vez.

BF = 1, el modulo LCD esta ocupado.

BF = 0, el modulo LCD esta disponible.

Los controladores de Hitachi se inicializarán de manera automática si durante el encendido conseguimos que la tensión de alimentación suba desde 0.5V hasta 4.5V en un tiempo comprendido entre 0.1 y 10ms. Puesto que esto puede ser difícil de conseguir, podemos recurrir a la inicialización por software. Consiste en enviar una serie de comandos de los descritos anteriormente y esperar un determinado tiempo.



4 EL MÓDULO LCD

Nota: Cuando se va a cargar la dirección de la DDRAM donde se va a escribir el próximo carácter, nótese que el D7 siempre es 1. Por lo tanto cuando apunto a una dirección de memoria en el display hay que considerar este 1 adicional. Se verá con más detalle al describir el mapa de memoria del módulo LCD.

La interfase entre el microcontrolador y el LCD se puede hacer con el bus de datos del PIC 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 display, ya que existe una instrucción que permite establecer dicha configuración. O sea tenemos que avisarle al LCD que vamos a operar en 8 o a 4 bits.

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 a la zona visible.

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

Proyecto 9: Mensaje simple en el LCD

A) Planteamiento del programa:

El programa consiste en visualizar un mensaje simple en la pantalla del LCD, en la línea superior, aplicando lógica programada y un LCD de 2x16, el mensaje estará almacenado en una tabla en memoria, el programa configura el LCD, saca los datos de la tabla y los envía al LCD, los caracteres son enviados en secuencia pero debido a gran velocidad no es perceptible a la vista, el mensaje puede ser cambiado en la tabla y el número máximo de caracteres es de 16, el PIC a utilizar es el 16F877 de microchip, el puerto B en conexión directa con el bus de datos del LCD y el puerto E en conexión directa con el bus de control del LCD.

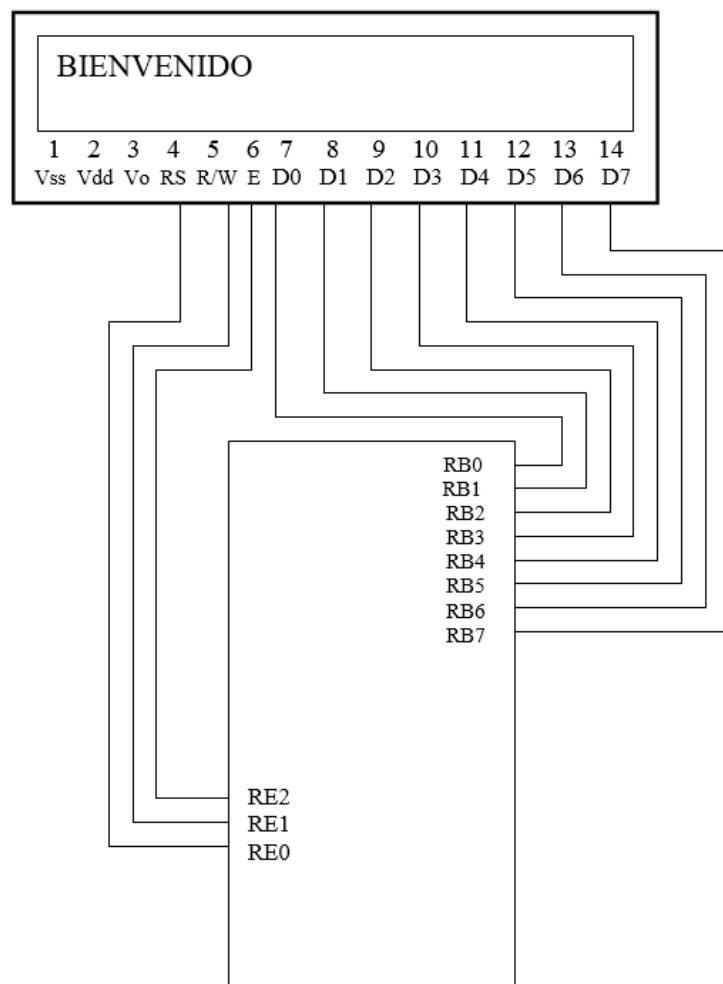
**B) Esquema electrónico:****PIC16F877**

Figura 4.6. Circuito electrónico para el proyecto 9.



4 EL MÓDULO LCD

C) Código del programa:

```
;-----  
#include "p16F877.inc"           ; incluir librerías del PIC 16F877  
#include "lcd_1.inc"             ; incluir macros para los bancos  
_config 3F31h                   ; palabra de configuración para el PIC 16F877  
  
dato1 EQU 0x20                 ; declarar la variable dato1  
dato2 EQU 0x21  
dato3 EQU 0x22  
dato4 EQU 0x23  
  
ORG 0x00                      ; inicio de programa en el vector reset  
banco1                         ; ir al banco1  
    clrf TRISB                  ; PORTB como salida  
    clrf TRISE                  ; PORTE como salida  
    movlw b'10000111'            ; puertos A y E como puertos digitales  
    movwf ADCON1  
    movlw b'00000111'            ; preescaler 1:256, para el retardo  
    movwf OPTION_REG  
banco0                         ; ir al banco0  
    clrf dato2                  ; para sacar datos de la tabla  
    movlw d'5'  
    movwf dato4                  ; para discernir comandos-datos en la tabla  
;  
inicio movf dato2,0              ; sacar comandos o datos de la tabla  
    call tabla                   ; según el valor de dato2  
    movwf dato1  
    xorlw 0x00                  ; para verificar si el ultimo dato de la  
    btfsc STATUS,2               ; tabla ha sido enviado al LCD  
    goto fin                     ; el programa ha terminado  
    call enviar                  ; envía el dato o comando al LCD  
    incf dato2,1                ; para sacar el siguiente dato de la tabla  
    goto inicio  
;  
-----
```



```

;-----

enviar bcf      PORTE,2      ;deshabilita el modulo LCD
      bsf      PORTE,1      ;LCD en modo lectura
      bcf      PORTE,0      ;para trabajar con comandos
      banco1
      movlw   0xFF          ;PORTB como entrada
      movwf   TRISB
      banco0
      bsf      PORTE,2      ;habilita el modulo LCD
espera btfsc   PORTB,7      ;espera a que el LCD este desocupado
      goto    espera
      bcf      PORTE,2
      banco1
      clrf    TRISB         ;PORTB como salida
      banco0
      movf    dato1,0        ;carga el PORTB con un comando o dato
      movwf   PORTB
      bcf      PORTE,1      ;LCD en modo escritura
      movf    dato2,0
      xorwf   dato4,0        ;verifica si el dato a enviar es un comando
      btfss   STATUS,2       ;o es un dato del mensaje
      goto    comando         ;si es un comando se va a comando
      bsf      PORTE,0      ;para trabajar con datos
      incf    dato4,1        ;actualiza dato4
      bsf      PORTE,2      ;habilita el modulo LCD
      bcf      PORTE,2      ;deshabilita el modulo LCD
      call    delay_1         ;retardo entre caracteres a ser escritos
      return           ;como parte del mensaje
comando bsf      PORTE,2
      bcf      PORTE,2
      call    delay_1         ;retardo entre caracteres a ser escritos
      return
fin     nop             ;fin de programa
      goto    fin

```



4 EL MÓDULO LCD

```
;-----  
delay_1    movlw      d'5'          ;retardo aproximado de 0.33 segundos  
            movwf      dato3  
  
label_2    clrf      TMR0  
            bcf       INTCON,2  
  
label_1    btfss     INTCON,2  
            goto     label_1  
            decfsz   dato3,1  
            goto     label_2  
            return  
  
tabla     addwf     PCL,1  
            retlw     b'00000001' ;comando CLEAR DISPLAY  
            retlw     b'00000110' ;comando ENTRY MODE SET  
            retlw     b'00001111' ;comando DISPLAY ON / OFF  
            retlw     b'00111111' ;comando FUNCTION SET  
            retlw     b'10000000' ;comando SET DDRAM ADDRESS  
            retlw     'I'        ;primer elemento del mensaje  
            retlw     'N'  
            retlw     'G'  
            retlw     'R'  
            retlw     'E'  
            retlw     'S'  
            retlw     'E'  
            retlw     ' '       '  
            retlw     'N'  
            retlw     'U'  
            retlw     'M'  
            retlw     'E'  
            retlw     'R'  
            retlw     'O'  
            retlw     '='  
            retlw     0x00      ;indica al programa fin del mensaje  
  
END  
;-----
```



D) Simulación del programa:

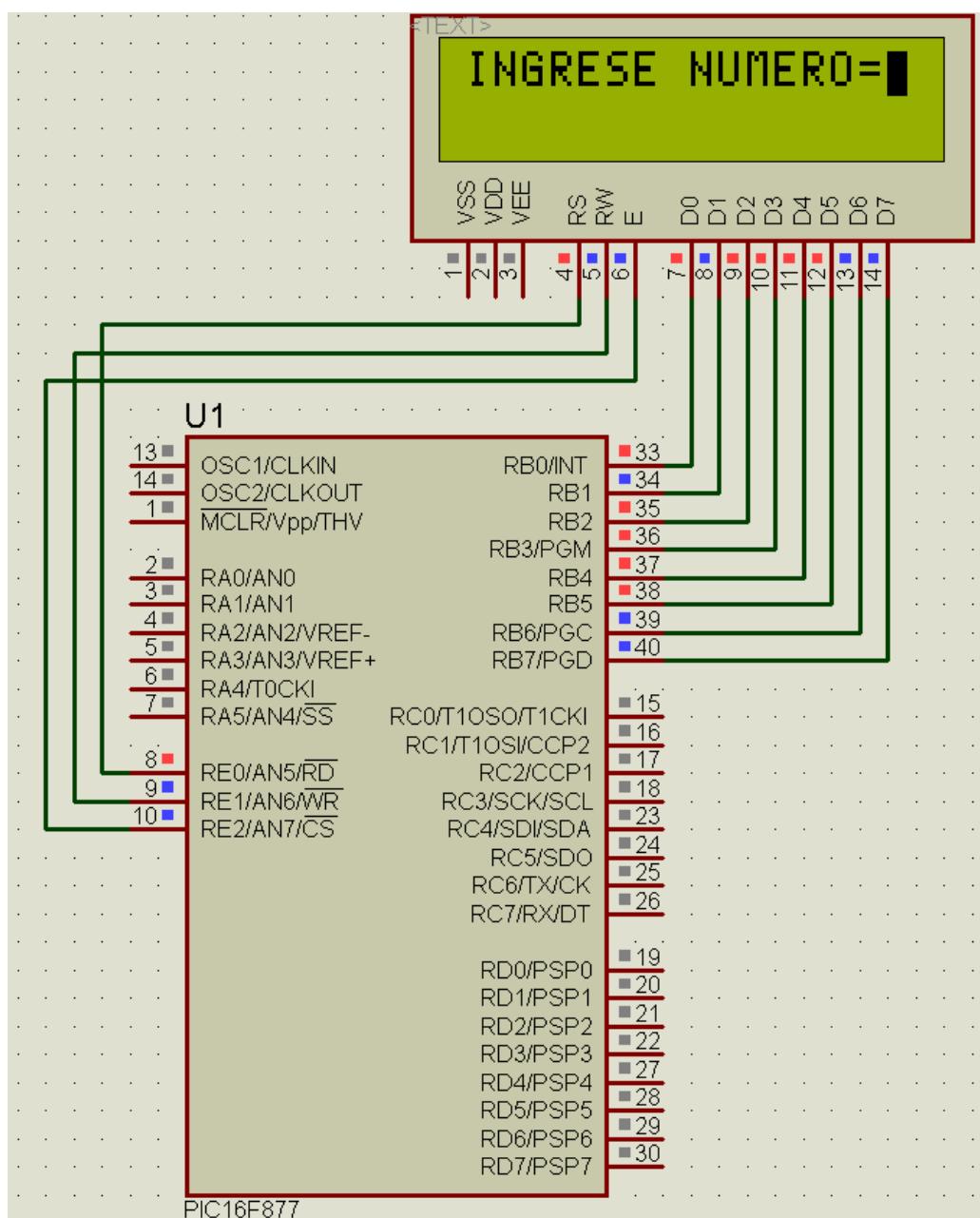


Figura 4.7. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



4 EL MÓDULO LCD

Proyecto 10: Mensajes Alternados

A) Planteamiento del programa:

El programa consiste en visualizar dos mensajes en la pantalla del LCD, alternadamente en la línea superior, aplicando lógica programada y un LCD de 2x16, los mensajes estarán almacenados en una tabla en memoria, el programa primero configura el LCD, luego extrae de la tabla el primer mensaje, lo envia a la pantalla del LCD, espera 2 segundos, luego de este tiempo borra la pantalla y extrae de la tabla el segundo mensaje, lo envia a la pantalla del LCD, espera 2 segundos y repite la secuencia, los caracteres son enviados en secuencia pero debido a gran velocidad no es perceptible a la vista, los mensajes pueden ser cambiados en la tabla y el número máximo de caracteres es de 16, el PIC a utilizar es el 16F877 de microchip, el puerto B en conexión directa con el bus de datos del LCD y el puerto E en conexión directa con el bus de control del LCD.

Primer mensaje: B I E N V E N I D O S A L A

Segundo mensaje: U N I V E R S I D A D U A S F

B) Esquema electrónico:

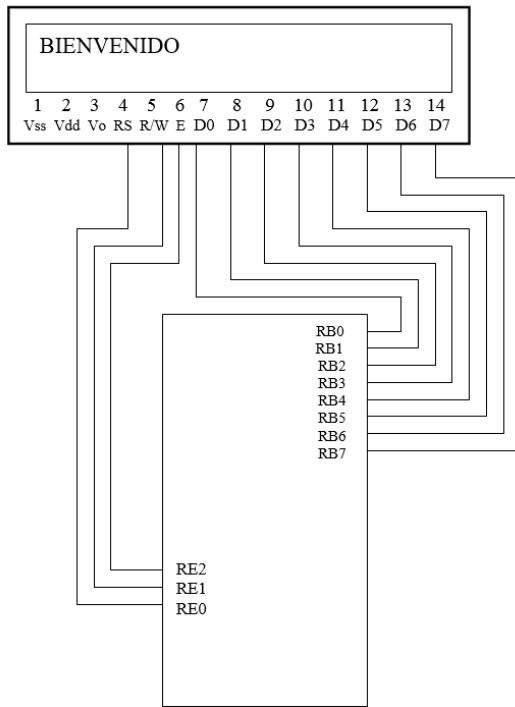


Figura 4.8. Circuito electrónico para el proyecto 10.



C) Código del programa:

```

;-----

#include"p16F877.inc"           ;incluir librerias del PIC 16F877
#include"lcd_2.inc"             ;incluir macros para los bancos
__config 3F31h                  ;palabra de configuración para el PIC 16F877

dato1    EQU    0x20            ;declarar la variable dato1
dato2    EQU    0x21
dato3    EQU    0x22
dato4    EQU    0x23
dato5    EQU    0x24
dato6    EQU    0x25

ORG      0x00                  ;inicio de programa en el vector reset

        banco1                ;ir al banco1
        clrf      TRISB       ;PORTB como salida
        clrf      TRISE       ;PORTE como salida
        movlw     b'10000111'  ;puertos A y E como puertos digitales
        movwf     ADCON1
        movlw     b'00000111'  ;preescaler 1:256, para el retardo
        movwf     OPTION_REG
        banco0                ;ir al banco0

mens_1   clrf     dato2        ;para sacar datos de la tabla
        clrf     dato5        ;para identificar el mensaje 1 o 2
        movlw     d'5'
        movwf     dato4        ;para discernir comandos-datos en la tabla
inicio   movf     dato2,0      ;sacar comandos o datos de la tabla segun el
        call     tabla         ;valor de dato2
        movwf     dato1
        xorlw     0x00          ;para verificar si el ultimo dato de la
        btfsc    STATUS,2      ;tabla ha sido enviado al LCD
        goto    fin            ;el programa ha terminado
        call     enviar         ;envia el dato o comando al LCD
        incf     dato2,1      ;para sacar el siguiente dato de la tabla
        goto    inicio
;
;
```



4 EL MÓDULO LCD

```
;-----  
enviar bcf      PORTE,2      ;deshabilita el modulo LCD  
      bsf      PORTE,1      ;LCD en modo lectura  
      bcf      PORTE,0      ;para trabajar con comandos  
      banco1  
      movlw    0xFF      ;PORTB como entrada  
      movwf    TRISB  
      banco0  
      bsf      PORTE,2      ;habilita el modulo LCD  
espera btfsc    PORTB,7      ;espera a que el LCD este desocupado  
      goto    espera  
      bcf      PORTE,2  
      banco1  
      clrf    TRISB      ;PORTB como salida  
      banco0  
      movf    dato1,0      ;carga el PORTB con un comando o dato  
      movwf    PORTB  
      bcf      PORTE,1      ;LCD en modo escritura  
      movf    dato2,0  
      xorwf    dato4,0      ;verifica si el dato a enviar es un comando  
      btfss    STATUS,2      ;o es un dato del mensaje  
      goto    comando      ;si es un comando se va a comando  
      bsf      PORTE,0      ;para trabajar con datos  
      incf    dato4,1      ;actualiza dato4  
      bsf      PORTE,2      ;habilita el modulo LCD  
      bcf      PORTE,2      ;deshabilita el modulo LCD  
      call    delay_1      ;retardo entre caracteres a ser escritos  
      return      ;como parte del mensaje  
comando bsf      PORTE,2  
      bcf      PORTE,2  
      call    delay_1      ;retardo entre caracteres a ser escritos  
      return  
;-----
```



```

;-----

fin      call      delay_2      ;retardo entre los mensajes
          btfss    dato5,0      ;verifica el mensaje actual
          goto     mens_2      ;ordena visualizar el mensaje 1
          goto     mens_1      ;ordena visualizar el mensaje 2
mens_2   incf     dato2,1
          movlw    d'27'
          movwf    dato4
          incf     dato5,1      ;actualiza dato5
          goto     inicio

;-----

delay_1  movlw    d'3'       ;retardo aproximado de 0.2 segundos
          movwf    dato3
label_2  clrf    TMR0
          bcf     INTCON,2
label_1  btfss    INTCON,2
          goto     label_1
          decfsz  dato3,1
          goto     label_2
          return

;-----

delay_2  movlw    d'30'      ;retardo aproximado de 2 segundos
          movwf    dato6
label_4  clrf    TMR0
          bcf     INTCON,2
label_3  btfss    INTCON,2
          goto     label_3
          decfsz  dato6,1
          goto     label_4
          return

;-----
```



4 EL MÓDULO LCD

```
;-----  
tabla addwf      PCL,1  
  
    retlw      b'00000001' ; comando CLEAR DISPLAY  
    retlw      b'00000110' ; comando ENTRY MODE SET  
    retlw      b'00001111' ; comando DISPLAY ON / OFF  
    retlw      b'00111111' ; comando FUNCTION SET  
    retlw      b'10000000' ; comando SET DDRAM ADDRESS  
    retlw      'B'          ;primer elemento del primer mensaje  
    retlw      'I'  
    retlw      'E'  
    retlw      'N'  
    retlw      'V'  
    retlw      'E'  
    retlw      'N'  
    retlw      'I'  
    retlw      'D'  
    retlw      'O'  
    retlw      'S'  
    retlw      ' '  
    retlw      'A'  
    retlw      ' '  
    retlw      'L'  
    retlw      'A'  
    retlw      0x00          ;indica al programa fin del mensaje  
    retlw      b'00000001' ; comando CLEAR DISPLAY  
    retlw      b'00000110' ; comando ENTRY MODE SET  
    retlw      b'00001111' ; comando DISPLAY ON / OFF  
    retlw      b'00111111' ; comando FUNCTION SET  
    retlw      b'10000000' ; comando SET DDRAM ADDRESS  
    retlw      'U'          ;primer elemento del segundo mensaje  
    retlw      'N'  
    retlw      'I'  
    retlw      'V'  
;-----
```



```
;-----  
        retlw 'E'  
        retlw 'R'  
        retlw 'S'  
        retlw 'I'  
        retlw 'D'  
        retlw 'A'  
        retlw 'D'  
        retlw ' '  
        retlw 'U'  
        retlw 'A'  
        retlw 'S'  
        retlw 'F'  
        retlw 0x00      ;indica al programa fin del mensaje  
END  
;-----
```



4 EL MÓDULO LCD

D) Simulación del programa:

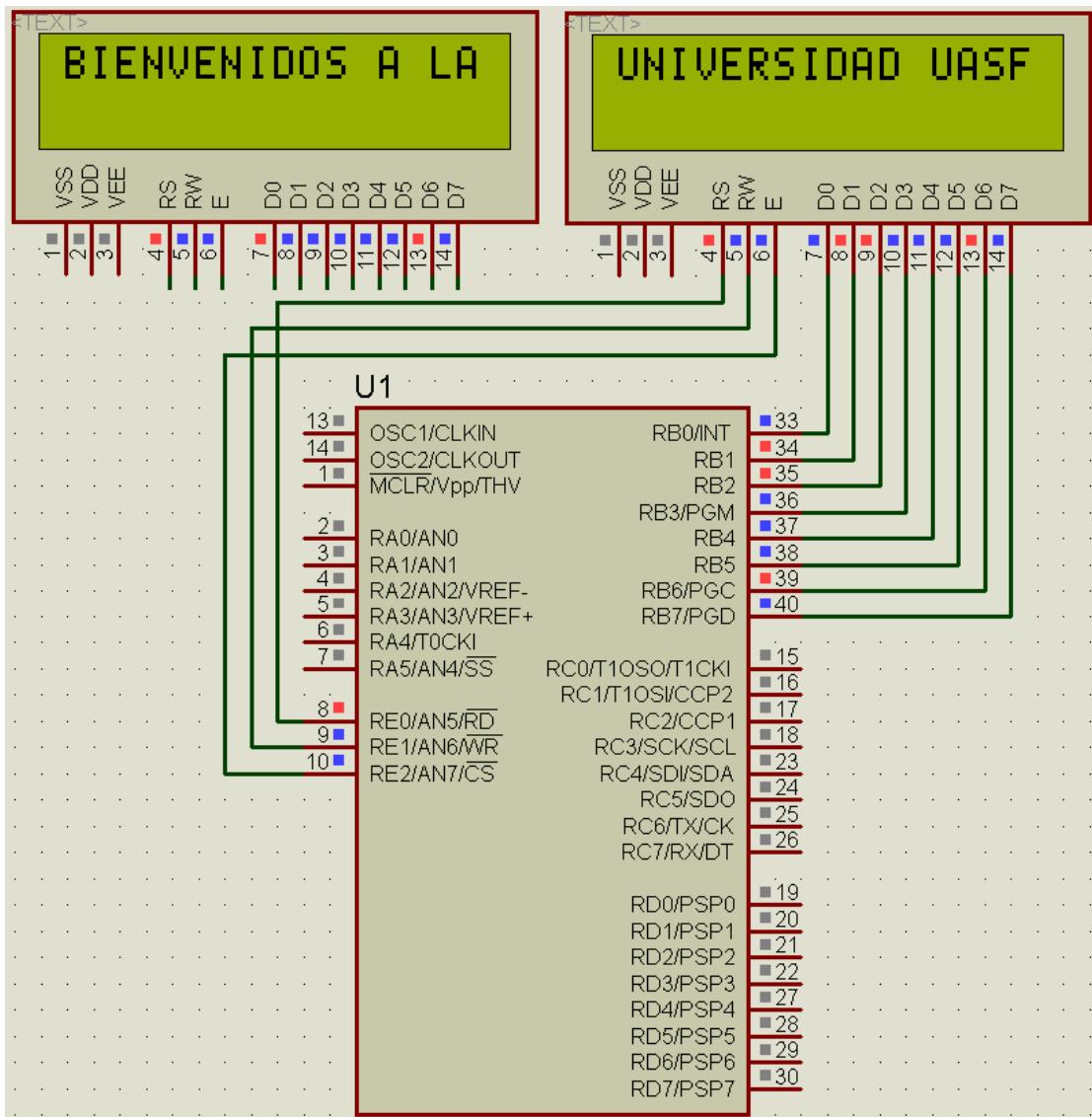


Figura 4.9. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



Proyecto 11: Mensaje en dos líneas

A) Planteamiento del programa:

El programa consiste en visualizar un mensaje de hasta 32 caracteres en la pantalla del LCD en ambas líneas, aplicando lógica programada y un LCD de 2x16, el mensaje estará almacenado en una tabla en memoria, el programa primero configura el LCD, luego extrae de la tabla el mensaje los primeros 16 caracteres, los envía a la pantalla del LCD, pasa a la segunda línea del LCD y envía los 16 últimos caracteres del mensaje, los caracteres son enviados en secuencia pero debido a gran velocidad no es perceptible a la vista, el mensaje puede ser cambiado en la tabla y el número máximo de caracteres es de 32, el PIC a utilizar es el 16F877 de microchip, el puerto B en conexión directa con el bus de datos del LCD y el puerto E en conexión directa con el bus de control del LCD.

El mensaje total es:

Primera línea: M I C R O C O N T R O L A D O R

Segunda línea: M I C R O C H I P 1 6 F 8 7 7

B) Esquema electrónico:

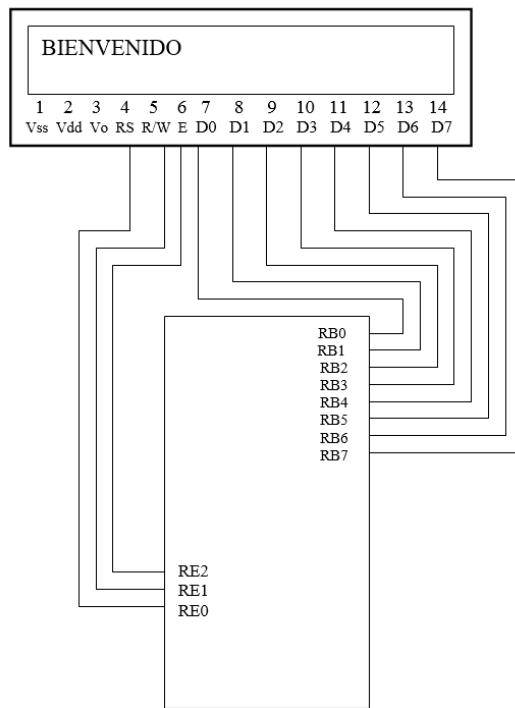


Figura 4.10. Circuito electrónico para el proyecto 11.



4 EL MÓDULO LCD

C) Código del programa:

```
;-----  
#include "p16F877.inc"           ; incluir librerias del PIC 16F877  
#include "lcd_3.inc"             ; incluir macros para los bancos  
_config 3F31h                   ; palabra de configuración para el PIC 16F877  
dato1 EQU 0x20                 ; declarar la variable dato1  
dato2 EQU 0x21  
dato3 EQU 0x22  
dato4 EQU 0x23  
dato5 EQU 0x24  
ORG 0x00                       ; inicio de programa en el vector reset  
        banco1                  ; ir al banco1  
        clrf        TRISB      ; PORTB como salida  
        clrf        TRISE      ; PORTE como salida  
        movlw      b'10000111' ; puertos A y E como puertos digitales  
        movwf      ADCON1  
        movlw      b'00000111' ; preescaler 1:256, para el retardo  
        movwf      OPTION_REG  
        banco0                  ; ir al banco0  
        clrf        dato2      ; para sacar datos de la tabla  
        clrf        dato5      ; mensaje inferior o superior  
        movlw      d'5'  
        movwf      dato4      ; para discernir comandos-datos en la tabla  
inicio  movf      dato2,0    ; sacar comandos o datos de la tabla segun el  
        call       tabla      ; valor de dato2  
        movwf      dato1  
        xorlw      0x00      ; para verificar si el ultimo dato de la  
        btfsc     STATUS,2    ; tabla ha sido enviado al LCD  
        goto      label_3    ; el programa ha terminado  
        call       enviar      ; envia el dato o comando al LCD  
        incf      dato2,1    ; para sacar el siguiente dato de la tabla  
        goto      inicio  
;-----
```



```

;-----

enviar bcf      PORTE,2      ;deshabilita el modulo LCD
        bsf      PORTE,1      ;LCD en modo lectura
        bcf      PORTE,0      ;para trabajar con comandos
        banco1
        movlw    0xFF          ;PORTB como entrada
        movwf    TRISB
        banco0
        bsf      PORTE,2      ;habilita el modulo LCD
espera  btfsc    PORTB,7      ;espera a que el LCD este desocupado
        goto    espera
        bcf      PORTE,2
        banco1
        clrf    TRISB          ;PORTB como salida
        banco0
        movf    dato1,0          ;carga el PORTB con un comando o dato
        movwf    PORTB
        bcf      PORTE,1      ;LCD en modo escritura
        movf    dato2,0
        xorwf    dato4,0          ;verifica si el dato a enviar es un comando
        btfss    STATUS,2      ;o es un dato del mensaje
        goto    comando
        bcf      PORTE,0      ;para trabajar con datos
        incf    dato4,1          ;actualiza dato4
        bsf      PORTE,2      ;habilita el modulo LCD
        bcf      PORTE,2      ;deshabilita el modulo LCD
        call    delay_1          ;retardo entre caracteres a ser escritos
        return
                           ;como parte del mensaje

comando bsf      PORTE,2
        bcf      PORTE,2
        call    delay_1          ;retardo entre caracteres a ser escritos
        return
;-----
```



4 EL MÓDULO LCD

```
;-----  
label_3  btfss    dato5,0      ;verifica msn superior o inferior  
        goto     msn_inf       ;ordena escribir el msn inferior  
        goto     fin  
msn_inf   movlw    d'23'  
        movwf    dato4  
        incf    dato2,1  
        incf    dato5,1  
        goto     inicio  
fin       nop          ;fin de programa  
        goto     fin  
delay_1   movlw    d'5'        ;retardo aproximado de 0.33 segundos  
        movwf    dato3  
label_2   clrf    TMR0  
        bcf     INTCON,2  
label_1   btfss    INTCON,2  
        goto     label_1  
        decfsz  dato3,1  
        goto     label_2  
        return  
tabla    addwf    PCL,1  
        retlw    b'00000001' ;comando CLEAR DISPLAY  
        retlw    b'00000110' ;comando ENTRY MODE SET  
        retlw    b'00001111' ;comando DISPLAY ON / OFF  
        retlw    b'00111111' ;comando FUNCTION SET  
        retlw    b'10000000' ;comando SET DDRAM ADDRESS  
        retlw    'M'         ;primer elemento del mensaje superior  
        retlw    'I'  
        retlw    'C'  
        retlw    'R'  
        retlw    'O'  
;-----
```



```
;-----  
        retlw    'C'  
        retlw    'O'  
        retlw    'N'  
        retlw    'T'  
        retlw    'R'  
        retlw    'O'  
        retlw    'L'  
        retlw    'A'  
        retlw    'D'  
        retlw    'O'  
        retlw    'R'  
        retlw    0x00      ;indica al programa fin del mensaje  
        retlw    b'11000000' ;comando SET DDRAM ADDRESS  
        retlw    'M'       ;primer elemento del mensaje inferior  
        retlw    'I'  
        retlw    'C'  
        retlw    'R'  
        retlw    'O'  
        retlw    'C'  
        retlw    'H'  
        retlw    'I'  
        retlw    'P'  
        retlw    ' '  
        retlw    '1'  
        retlw    '6'  
        retlw    'F'  
        retlw    '8'  
        retlw    '7'  
        retlw    '7'  
        retlw    0x00      ;indica al programa fin del mensaje  
  
END  
;-----
```



4 EL MÓDULO LCD

D) Simulación del programa:

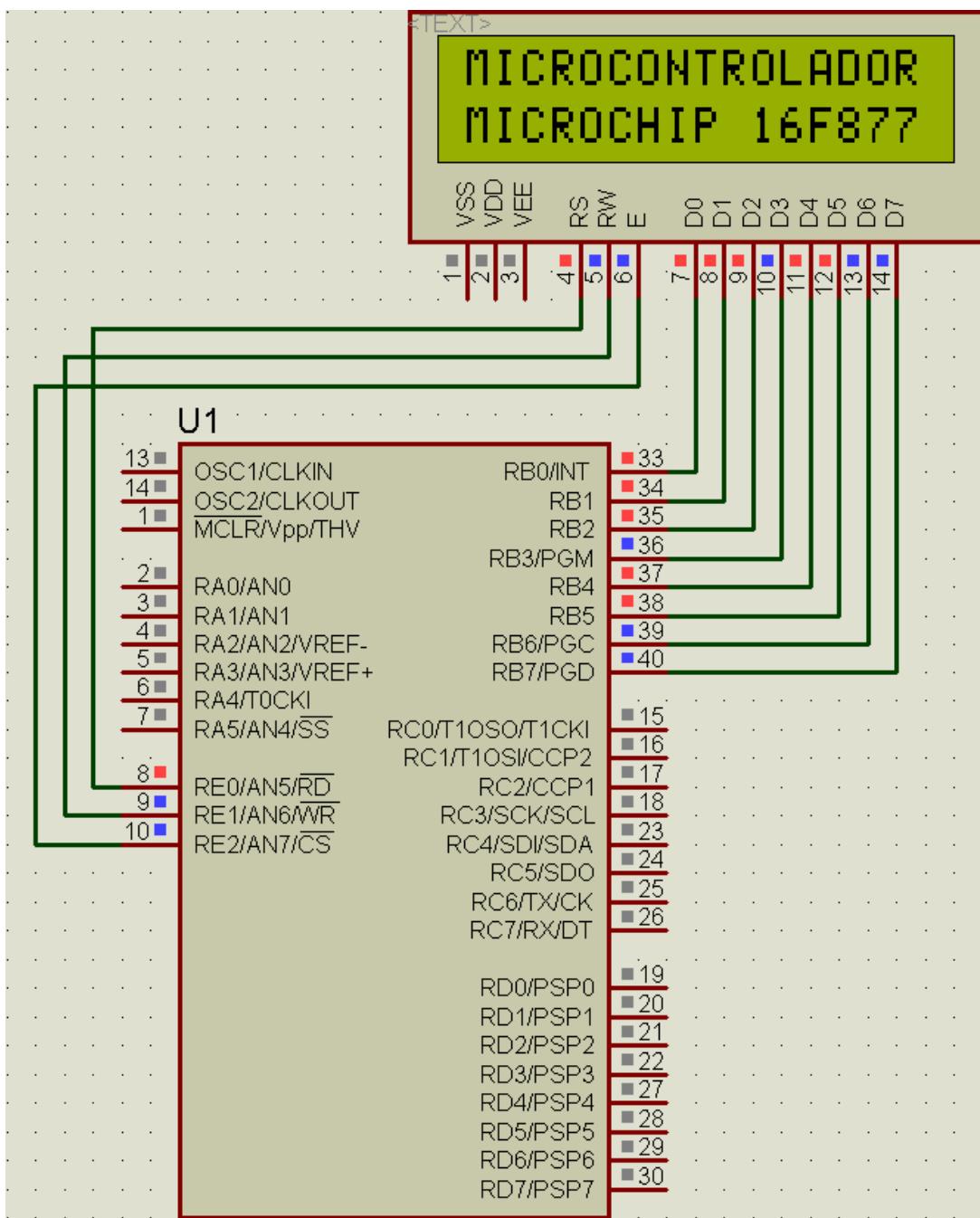


Figura 4.11. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.

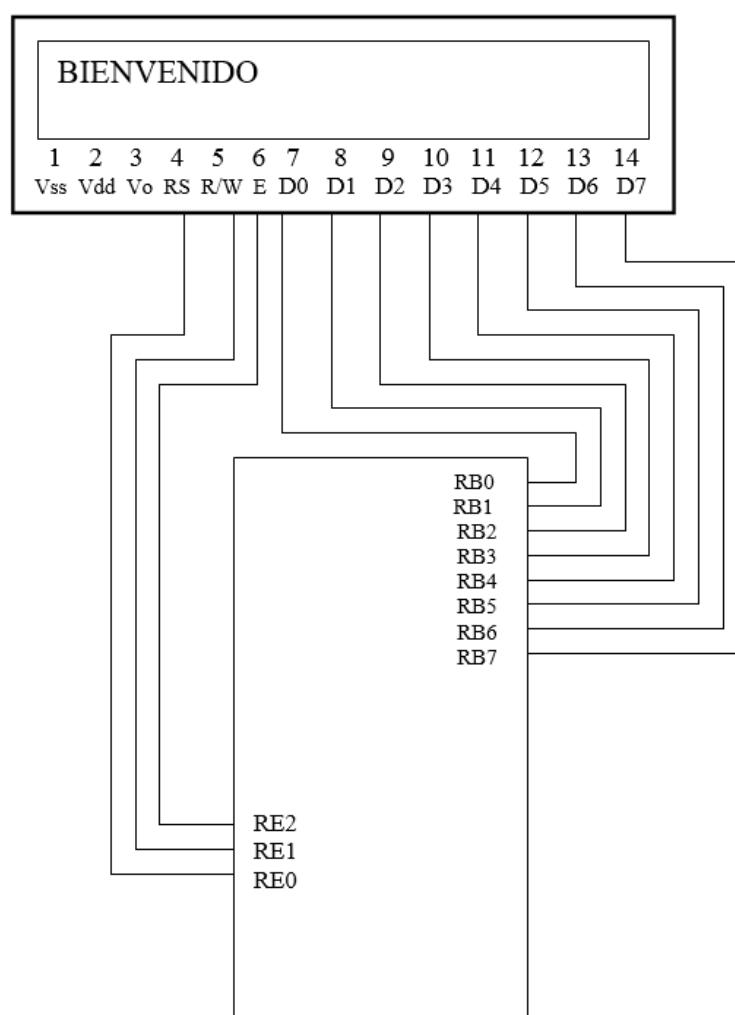


Proyecto 12: Mensaje en movimiento

A) Planteamiento del programa:

El programa es una variante de los anteriores, lo que hace es desplazar un mensaje de 42 caracteres de derecha a izquierda, el tiempo de desplazamiento es de un segundo, cada segundo los caracteres del mensaje se desplazan una posición a la izquierda, terminado de pasar todo el mensaje se reinicia la secuencia.

B) Esquema electrónico:



PIC16F877

Figura 4.12. Circuito electrónico para el proyecto 12.



4 EL MÓDULO LCD

C) Código del programa:

```
;-----  
#include "p16F877.inc"           ; incluir librerias del PIC 16F877  
#include "lcd_4.inc"             ; incluir macros para los bancos  
_config 3F31h                   ;palabra de configuración para el PIC 16F877  
dato1    EQU      0x20          ;declarar la variable dato1  
dato2    EQU      0x21  
dato3    EQU      0x22  
dato4    EQU      0x23  
ORG      0x00                  ;inicio de programa en el vector reset  
        banco1                ;ir al banco1  
        clrf      TRISB       ;PORTB como salida  
        clrf      TRISE       ;PORTE como salida  
        movlw     b'10000111'  ;puertos A y E como puertos digitales  
        movwf     ADCON1  
        movlw     b'00000111'  ;preescaler 1:256, para el retardo  
        movwf     OPTION_REG  
        banco0                ;ir al banco0  
label_3  clrf     dato2        ;para sacar datos de la tabla  
        movlw     d'5'  
        movwf     dato4        ;para discernir comandos-datos en la tabla  
inicio   movf     dato2,0      ;sacar comandos o datos de la tabla segun el  
        call      tabla         ;valor de dato2  
        movwf     dato1  
        xorlw     0x00          ;para verificar si el ultimo dato de la  
        btfsc    STATUS,2      ;tabla ha sido enviado al LCD  
        goto     fin            ;el programa ha terminado  
        call      enviar         ;envia el dato o comando al LCD  
        incf     dato2,1      ;para sacar el siguiente dato de la tabla  
        goto     inicio         ;-----
```



```

;-----

enviar    bcf      PORTE,2      ;deshabilita el modulo LCD
          bsf      PORTE,1      ;LCD en modo lectura
          bcf      PORTE,0      ;para trabajar con comandos
          banco1
          movlw    0xFF        ;PORTB como entrada
          movwf    TRISB
          banco0
          bsf      PORTE,2      ;habilita el modulo LCD
espera    btfsc   PORTB,7      ;espera a que el LCD este desocupado
          goto    espera
          bcf      PORTE,2
          banco1
          clrf    TRISB        ;PORTB como salida
          banco0
          movf    dato1,0      ;carga el PORTB con un comando o dato
          movwf   PORTB
          bcf      PORTE,1      ;LCD en modo escritura
          movf    dato2,0
          xorwf   dato4,0      ;verifica si el dato a enviar es un comando
          btfss   STATUS,2      ;o es un dato del mensaje
          goto    comando       ;si es un comando se va a comando
          bsf      PORTE,0      ;para trabajar con datos
          incf    dato4,1      ;actualiza dato4
          bsf      PORTE,2      ;habilita el modulo LCD
          bcf      PORTE,2      ;deshabilita el modulo LCD
          call    delay_1       ;retardo entre caracteres a ser escritos
          return           ;como parte del mensaje
comando   bsf      PORTE,2
          bcf      PORTE,2
          call    delay_1       ;retardo entre caracteres a ser escritos
          return
;-----
```



4 EL MÓDULO LCD

```
;-----  
fin      movlw    d'45'          ;retardo aproximado de 3 segundos  
         movwf    dato3  
         call     label_2  
         goto    label_3  
delay_1   movlw    d'10'          ;retardo aproximado de 0.66 segundos  
         movwf    dato3  
label_2   clrf    TMR0  
         bcf     INTCON,2  
label_1   btfss   INTCON,2  
         goto    label_1  
         decfsz  dato3,1  
         goto    label_2  
         return  
tabla    addwf   PCL,1  
         retlw   b'00000001' ;comando CLEAR DISPLAY  
         retlw   b'00000111' ;comando ENTRY MODE SET  
         retlw   b'00001100' ;comando DISPLAY ON / OFF  
         retlw   b'00110111' ;comando FUNCTION SET  
         retlw   b'10010000' ;comando SET DDRAM ADDRESS  
         retlw   'B'           ;primer elemento del mensaje  
         retlw   'I'  
         retlw   'E'  
         retlw   'N'  
         retlw   'V'  
         retlw   'E'  
         retlw   'N'  
         retlw   'I'  
         retlw   'D'  
         retlw   'O'  
         retlw   ' '  
         retlw   'A'  
;-----
```



```
;-----  
    retlw    'L'  
    retlw    ' '  
    retlw    'M'  
    retlw    'U'  
    retlw    'N'  
    retlw    'D'  
    retlw    'O'  
    retlw    ' '  
    retlw    'D'  
    retlw    'E'  
    retlw    ' '  
    retlw    'L'  
    retlw    'A'  
    retlw    ' '  
    retlw    'T'  
    retlw    'E'  
    retlw    'C'  
    retlw    'N'  
    retlw    'O'  
    retlw    'L'  
    retlw    'O'  
    retlw    'G'  
    retlw    'I'  
    retlw    'A'  
    retlw    ' '  
    retlw    'B'  
    retlw    'I'  
    retlw    'E'  
    retlw    'N'  
    retlw    'V'  
    retlw    'E'  
    retlw    'N'  
;-----
```



4 EL MÓDULO LCD

```
;-----  
retlw    'I'  
retlw    'D'  
retlw    'O'  
retlw    ' '  
retlw    'A'  
retlw    ' '  
retlw    'L'  
retlw    'A'  
retlw    ' '  
retlw    'U'  
retlw    'A'  
retlw    'S'  
retlw    'F'  
retlw    ' '  
retlw    0x00      ;indica al programa fin del mensaje  
END  
;-----
```



D) Simulación del programa:

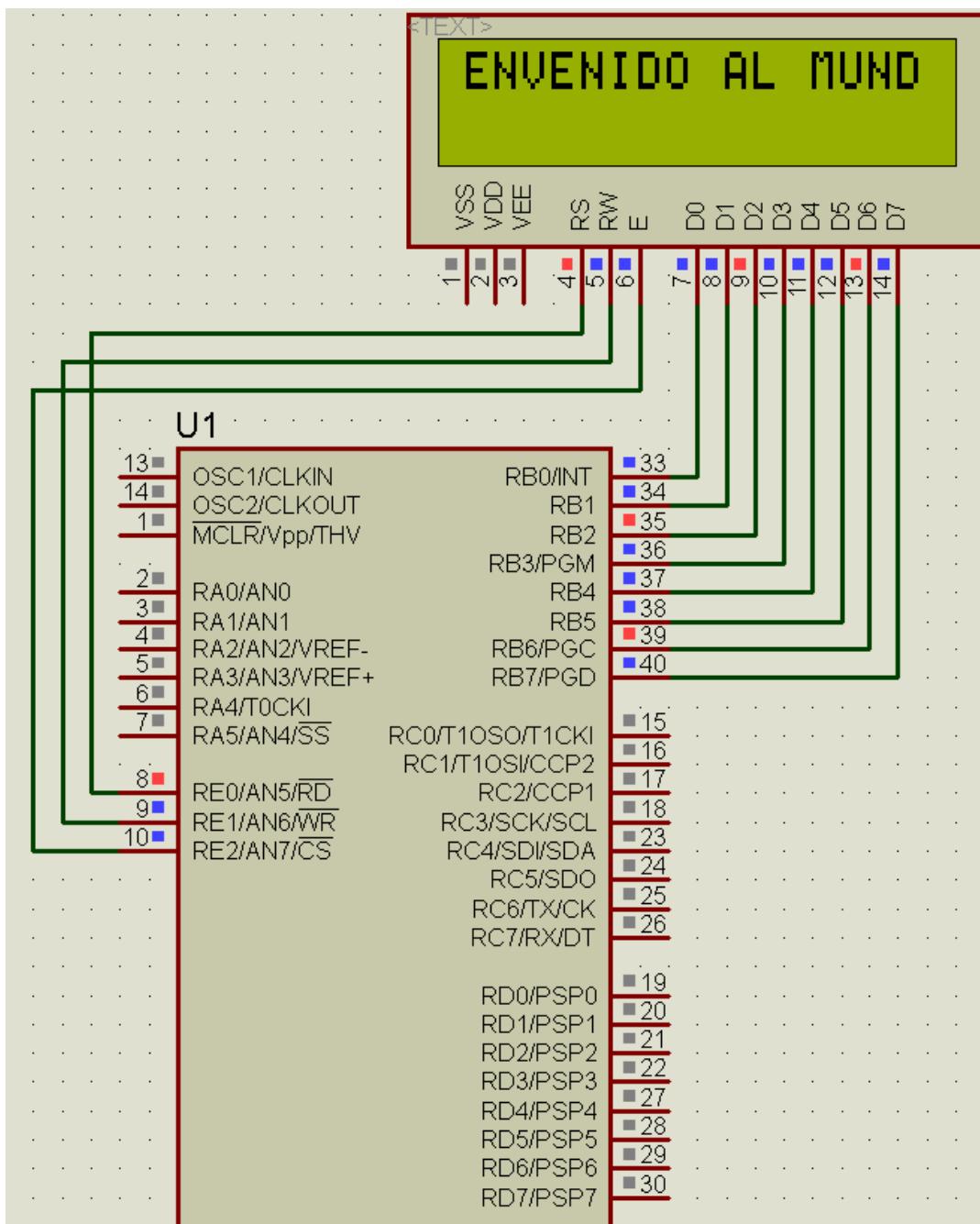


Figura 4.13. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



4 EL MÓDULO LCD

Actividades

A) Desarrollar las siguientes aplicaciones:

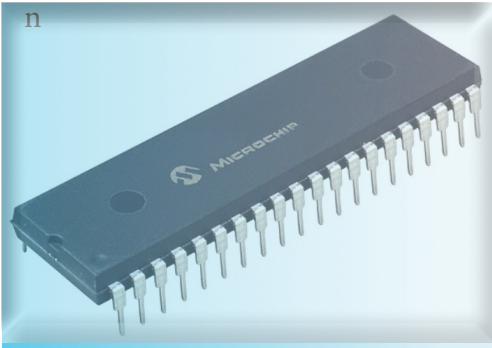
1. El proyecto 09 para el PIC16F877, produce que en la pantalla del módulo LCD se visualice un mensaje. Modificar el programa para que el mensaje se muestre an ambas líneas.

2. El proyecto 09 para el PIC16F877, produce que en la pantalla del módulo LCD se visualice un mensaje. Modificar el programa para que en la primera línea se escriba tu nombre y en la segunda línea tus apellidos.

3. Modificar el programa del proyecto 09 para que cada vez que se presiona el pulsador conectado al pin RA4 se incremente un contador que se visualiza en el centro de la primera línea de la pantalla.

4. Escribe un programa que realice la siguiente tarea: cada vez que se presione el pulsador conectado al pin RA4 se visualice el mensaje “HOLA” en el centro de la primera línea de la pantalla, y cuando se presione el pin RA5 se visualice el mensaje “ADIOS”, también en el centro de la primera línea de la pantalla.





CAPÍTULO 5

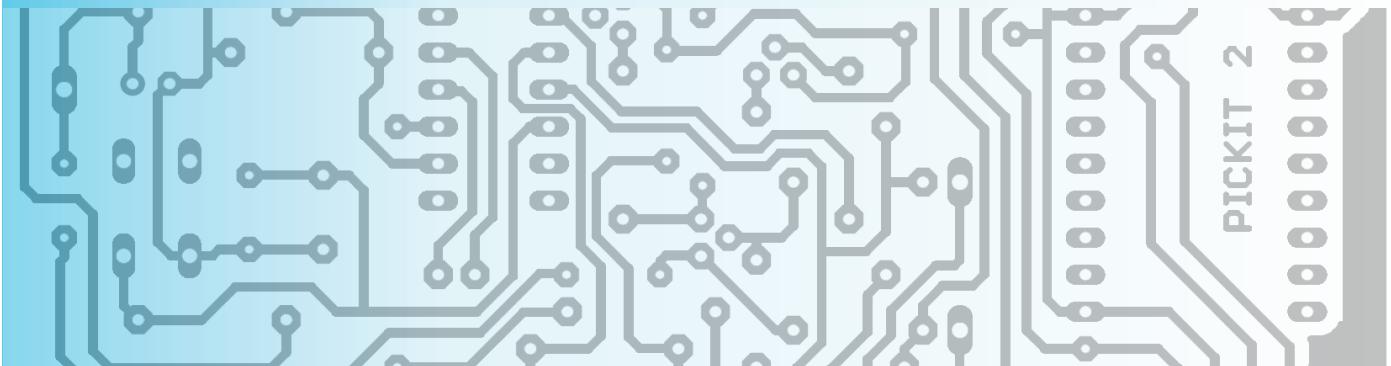
EL MÓDULO ADC

CONTENIDO DEL CAPÍTULO

- 1. Introducción
- 2. Conversión analógica digital
- 3. ADC del PIC
 - 3.1. Configuración de ADC
 - 3.2. Operación del ADC
- Proyecto 13: Termómetro digital

APRENDEREMOS A:

- Configurar el módulo ADC del PIC.
- Aplicar el módulo ADC en la digitalización de señales analógicas.
- Configurar los registros de configuración de este periférico del PIC.
- Utilizar y configurar las interrupciones asociadas a este periférico.





1. Introducción

Como podemos observar en nuestro día a día el mundo real todavía es y siempre será fundamentalmente un lugar analógico. Para poder aprovechar los beneficios del procesado de los PIC y llevar sus beneficios a aplicaciones para el mundo real, las señales analógicas deberán de ser traducidas a un formato que el PIC pueda entender. Esta es la función del módulo conversor analógico digital (ADC). Después de procesarlas por el PIC, el resultado será una ristra de bits de información que deberá de ser devuelta a su forma analógica gracias al módulo DAC.

Los ADC y DAC están en prácticamente todos los sistemas electrónicos, como reproductores de compact disk, mp3, móviles, módem, tarjetas de sonido, que pueden contener uno de ellos o ambos. Como la arquitectura de los ADCs es tan útil y es requerida por muchas pequeñas aplicaciones de microprocesadores, las arquitecturas de los microprocesadores a menudo incluyen un ADC como un periférico incorporado, y los diseñadores del PIC hicieron eso mismo.

Desde el punto de vista de los programadores los módulos ADCs y DACs pueden verse como cajas negras. El ADC acepta la entrada de una cantidad analógica, normalmente tensión, y proporciona a su salida un código digital de n-bits cada f_s segundos que representa esa entrada analógica. Al número f_s se le denomina frecuencia de muestreo (sampling frequency).

La caja negra del DAC acepta una palabra de un código digital de n-bits cada f_s segundos y genera una salida analógica equivalente, normalmente tensión. Esto puede ser una interpretación suficiente de los convertidores, pero el entender como se realiza la conversión de datos puede ayudarnos a entender las limitaciones de las operaciones de los módulos ADC y DAC, y puede ayudarnos en la selección de conversores de datos para la aplicación en la que estemos trabajando.



2. Conversión analógica digital

Los métodos a través de los cuales se genera un código digital dentro de un ADC son varios. Mientras que los ADCs pueden tener recibir casi cualquier medida analógica (corriente, carga, voltaje, temperatura, presión acústica, etc.) como entrada, lo más común es que el módulo ADC convierta un valor de tensión analógico en un número digital. Normalmente, los sistemas que pueden convertir una amplia variedad de valores primero convierten esas señales en tensiones, y después usando el ADC en modo-tensión convierten el valor en un número digital. El número digital que genera el ADC puede codificarse en cualquier sistema, pero lo más común es que se represente como un número binario con o sin signo.

Los ADCs y sus capacidades se pueden describir mediante un amplísimo número de parámetros. Pero con solo unos pocos parámetros básicos y bastante descriptivos del ADC se puede entender como seleccionar y hacer un buen uso de él. La velocidad de un ADC se medida a través del mínimo periodo de muestreo T_{min} ; que se refiere al mínimo tiempo necesario para convertir la entrada de una tensión a un número digital. El mínimo periodo de muestreo es equivalente a la máxima frecuencia de muestreo, el número máximo de muestras que el ADC puede convertir en un segundo. La máxima frecuencia de muestreo f_{max} se calcula como:

$$f_{max} = 1 / T_{min}$$

Antes de entrar en más detalles existen ciertos conceptos que tienen que quedar claros, algunos de ellos ya los he mencionado:

ADC (Analog - to - Digital Converter): convierte una señal analógica (tensión / corriente) en un valor digital.

DAC (Digital - to - Analog Converter): convierte un valor digital en un valor analógico (tensión / corriente).



5 EL MÓDULO ADC

Periodo de muestreo: para el ADC, es el tiempo entre conversiones, normalmente, los muestreos se realizan a una velocidad fija.

Vref (Voltaje de referencia): las señales analógicas varían entre 0 y Vref, o entre + / - Vref.

Resolución: es el número de bits usados en la conversión (8bits, 10 bits, 12 bits, 16 bits, etc.).

Tiempo de conversión: es el tiempo que se necesita para la conversión analógica-digital.

La resolución de un ADC es el mínimo cambio en la entrada analógica que se detecta en la salida, normalmente es un cambio de +1 en el número de la salida. En otras palabras, la resolución representa el cambio en la entrada analógica que corresponde al cambio de 1 Lsb en la salida o también para un ADC de N bits:

$$\text{Resolución} = 1 / 2^N(V_{REF+} - V_{REF-})$$

Donde VREF+ es el nivel de tensión de referencia positivo, y VREF- el nivel de tensión de referencia negativo. Aunque como normalmente vamos a usar VREF- = 0V:

$$\text{Resolución} = 1 / 2^N * V_{REF+}$$

La precisión ADC es el número de niveles que el ADC puede distinguir. Algunas veces, la precisión se define con el número de bits que se necesitan para codificar el número de niveles.

$$\text{Código salida} = V_{IN} / V_{REF} * 2^N$$



El rango del ADC es la diferencia entre el valor máximo y mínimo de entradas que puede aceptar para realizar la conversión. Normalmente el rango se encuentra entre VREF+ y VREF- en el caso de conversión de tensiones, y se proporcionan como entradas al ADC.

3. ADC del PIC

Los PIC 16f87X tienen un módulo ADC de 10-bits. Y es compatible con los ADC de 8 bits de los PIC16C7X. El módulo ADC como otros periféricos del PIC es controlado por un número de registros de configuración, habilitación y flags. Las conexiones externas de entrada al ADC están restringidas a unos pines específicos (el puerto A).

Tabla 5.1. Diferentes registros internos del módulo ADC.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on MCLR, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSP1F ⁽¹⁾	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSP1E ⁽¹⁾	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0	--0- 0000	--0- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h ⁽¹⁾	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h ⁽¹⁾	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used for A/D conversion.

Note 1: These registers/bits are not available on the 28-pin devices.

3.1. Configuración de ADC

El módulo ADC tiene diversos canales de entrada, como se puede ver en la figura 5.1. Por supuesto que no es necesario utilizar todos los canales, se puede seleccionar usar desde ninguno a todos. El número de canales del ADC que se usan se selecciona en el primer registro de control del ADC, el ADCON1. Otro registro importante es el ADCON0, ambos registros se muestran a continuación:

Nota: ambos registros se copiaron desde el datasheet del fabricante.



5 EL MÓDULO ADC

ADCON0 REGISTER (ADDRESS: 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON

bit 7

bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits

00 = Fosc/2

01 = Fosc/8

10 = Fosc/32

11 = FRC (clock derived from the internal A/D module RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = channel 0, (RA0/AN0)

001 = channel 1, (RA1/AN1)

010 = channel 2, (RA2/AN2)

011 = channel 3, (RA3/AN3)

100 = channel 4, (RA5/AN4)

101 = channel 5, (RE0/AN5)⁽¹⁾

110 = channel 6, (RE1/AN6)⁽¹⁾

111 = channel 7, (RE2/AN7)⁽¹⁾

bit 2 **GO/DONE:** A/D Conversion Status bit

If ADON = 1:

1 = A/D conversion in progress (setting this bit starts the A/D conversion)

0 = A/D conversion not in progress (this bit is automatically cleared by hardware when the A/D conversion is complete)

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is operating

0 = A/D converter module is shut-off and consumes no operating current

Note 1: These channels are not available on PIC16F873/876 devices.

Legend:

R = Readable bit

W = Writable bit

U = Unimplemented bit, read as '0'

- n = Value at POR

'1' = Bit is set

'0' = Bit is cleared

x = Bit is unknown



ADCON1 REGISTER (ADDRESS 9Fh)

U-0	U-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	—	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7						bit 0	

bit 7 **ADFM:** A/D Result Format Select bit

1 = Right justified. 6 Most Significant bits of ADRESH are read as '0'.

0 = Left justified. 6 Least Significant bits of ADRESL are read as '0'.

bit 6-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits:

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

A = Analog input D = Digital I/O

Note 1: These channels are not available on PIC16F873/876 devices.

2: This column indicates the number of analog channels available as A/D inputs and the number of analog channels used as voltage reference inputs.



5 EL MÓDULO ADC

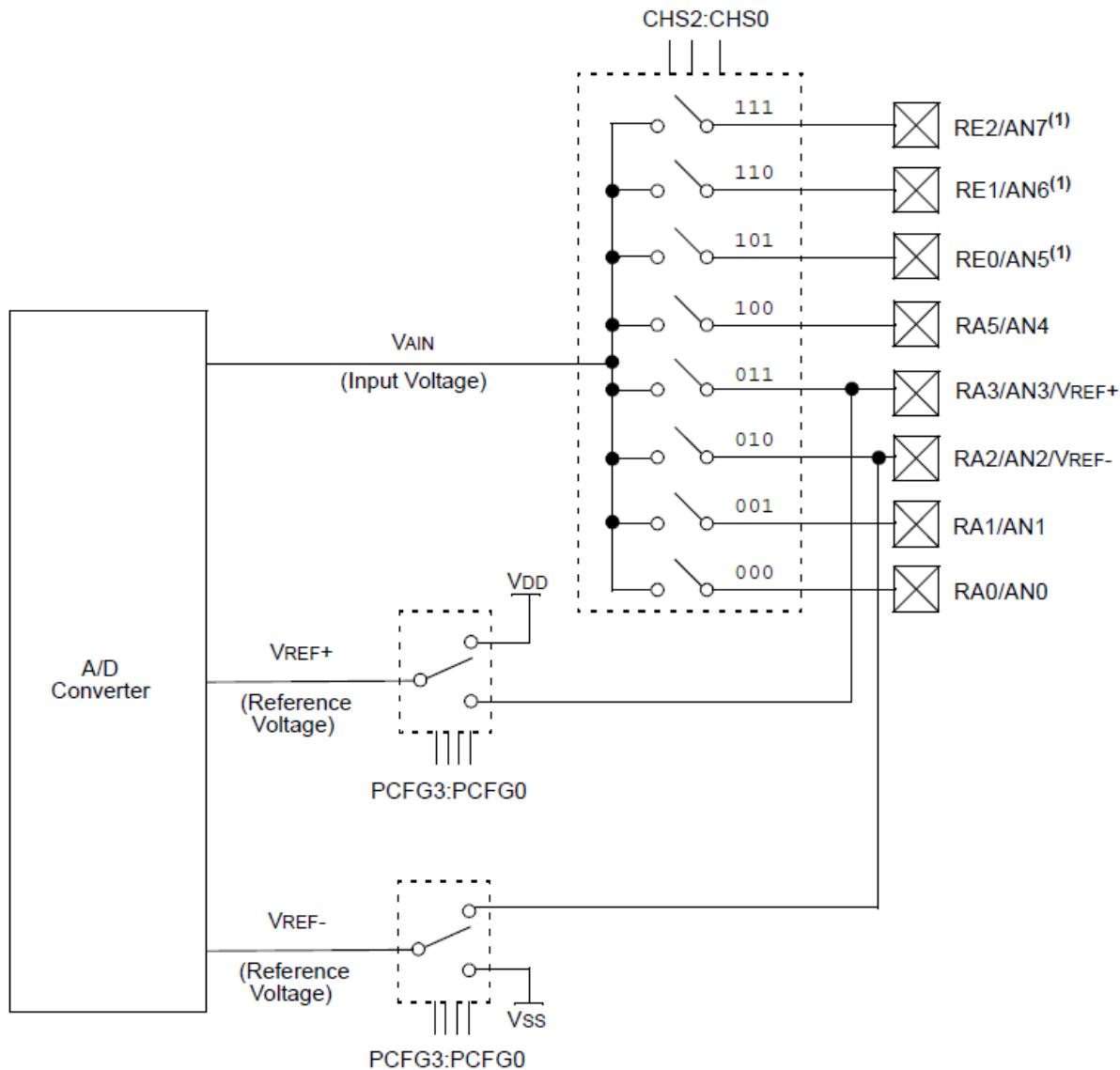


Figura 5.1. Diagrama de bloques del módulo ADC del PIC 16F877, como se observa este módulo cuenta con 8 canales de entrada, la configuración de estos canales es a través del registro ADCON1.

Los bits ADCON1 [3:0], PCFG [3:0], determina el número de ADC y canales digitales disponibles en el puerto A. En el registro ADCON1, se puede ver los tipos de configuración para cada combinación de bits PCFG [3:0].

ADCON1 también selecciona las fuentes de referencia, VREF + y VREF -. Estas tensiones de referencia especifican entre que valores el ADC espera que se sitúen la señal de entrada.



El ADC divide el rango ($V_{REF+} - V_{REF-}$) en $2^{10} = 1024$ niveles.

La salida del PIC será un número de 10 bits, y se representará en el par de registros ADRESH - ADRESL, es decir, 16 bits. El módulo ADC proporciona la flexibilidad de justificar a la derecha o izquierda el resultado de 10 bits, en el registro resultante de 16 bits. El bit de selección de formato del ADC (ADFM – A / D Format Select bit) controla esta justificación. En la figura 5.2. Se explica la operación de justificación del ADC. Los bits extras son cargados con ‘0’s. Cuando el módulo ADC no va a sobrescribir estos registros (ADC deshabilitado), estos registros se pueden usar como registros de propósito general de 8 bits.

Mientras que los bits PCFG[3:0] del ADCON1 permiten al PIC tener habilitados diferentes canales, el PIC no puede convertir más de un canal al mismo tiempo. El canal a convertir, se seleccionará a través de los bits CHS2:CHS1. Los tres bits son la representación del canal seleccionado por el ADC en ese preciso instante.

El registro ADCON0 también contiene otro bit importante para la configuración, ADON ADCON0[0]. Cuando ADON = 1, el modulo ADC es habilitado, encendido, y consume corriente. Cuando ADON = 0 el modulo estará deshabilitado y no consumirá corriente. Si el módulo ADC no está siendo usado debería apagarse mediante ADON = 0, que es el valor que se le da después de un reset.

El módulo ADC del PIC utiliza el método de las aproximaciones sucesivas para generar el resultado de 10 bits, y tal como hemos visto el ADC de aproximaciones sucesivas, se necesita un ciclo de reloj por bit como tiempo de conversión, es decir 10 ciclos de reloj.

El datasheet del PIC especifica que el conversor ADC necesita 12 TAD para tener un resultado exacto de la conversión, donde TAD es por lo menos $1,6\mu s$. El PIC usa un periodo TAD extra para preparar el ADC y otro para copiar el resultado en los registros ADRESH:ADRESL después de la conversión. Para garantizar que el TAD es mayor de $1,6\mu s$ el PIC proporciona 4 opciones para seleccionar el reloj del ADC mediante los bits ADCS1:ADCS0, ADCON0[7:6].



5 EL MÓDULO ADC

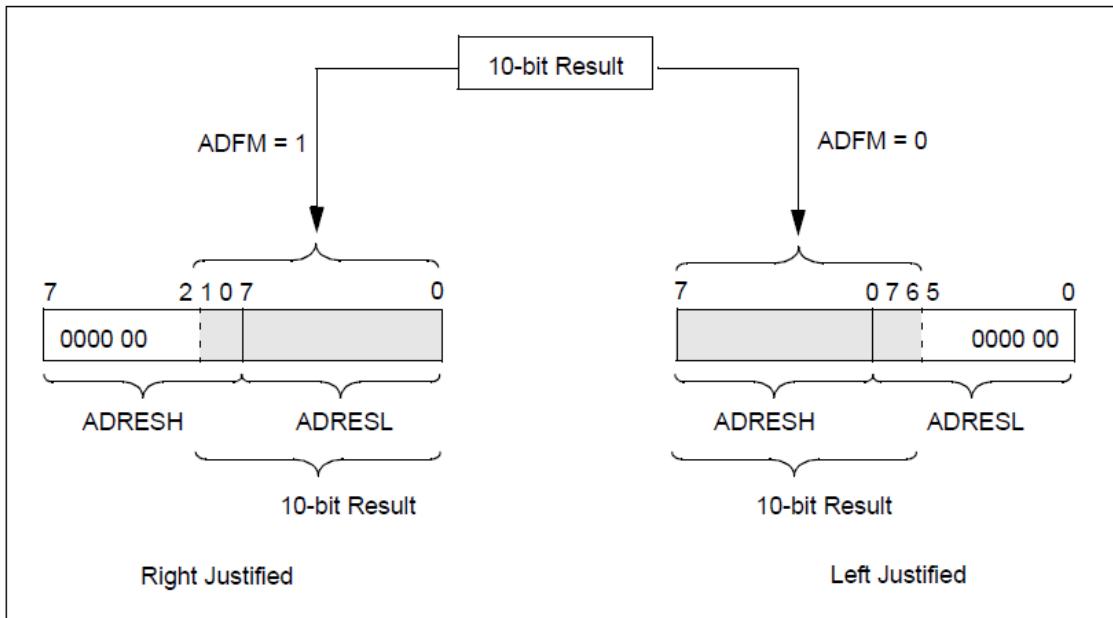


Figura 5.2. Justificación a la derecha o izquierda en los registros ADRESL y ADRESH, el resultado siempre es de 10 bits, esta justificación depende de la configuración previa.

Las frecuencias que se pueden seleccionar son las resultantes de dividir a la frecuencia del reloj FOSC entre 2, 8 ó 32, también se puede seleccionar la frecuencia del oscilador RC interno. Si el ADC es controlado por una frecuencia relacionada con la FOSC se debe seleccionar el divisor de forma que el TAD sea mayor de 1, 6 μ s, o la conversión del ADC no será exacta.

La última opción del ADC para configurar antes de poder usar el ADC son las fuentes de interrupciones ADC. El ADC del PIC genera una interrupción cuando la conversión de 10 bits se ha completado. Para activar las interrupciones del ADC primero debemos poner a '0' el bit ADIF (ADC Interrupt Flag, PIR1[6]) para prevenir una solicitud de interrupción falsa cuando habilitemos las interrupciones. También necesitamos configurar el bit ADIE (ADC Interrupt Enable, PIE1[6]) para hacer del ADC una fuente de interrupciones. La interrupción ADC es activada completamente mediante los bits de GIE / GIEH y PEIE / GIEL.



3.2. Operación del ADC

Estos son los pasos que hay que seguir para realizar una conversión ADC:

1. Configurar el módulo ADC:
 - Configurar los pines analógicos / tensiones de referencia / pines digitales (ADCON1)
 - Seleccionar el canal de entrada del ADC (ADCON0)
 - Seleccionar el reloj de conversión (ADCON0)
 - Activar el módulo ADC (ADCON0)
2. Configurar las interrupciones ADC
 - Poner el bit ADIF a '0'
 - Poner a '1' el bit ADIE
 - Poner a '1' el bit GIE
 - Esperar el tiempo requerido
3. Iniciar la conversión
 - Activar el bit GO / DONE (ADCON0)
 - Esperar a que el ciclo de conversión ADC se complete
 - Chequear si el bit GO / DONE está a '0' o el bit ADIF está a '1'
 - Esperar a que se produzca una interrupción ADC
 - Leer el resultado del par de registros (ADRESH:ADRESL), resetear el bit ADIF si es necesario
 - Para realizar la siguiente conversión ir al paso 1 o al 2 si es necesario

En resumen la configuración del módulo ADC, podría seguir el siguiente orden:

a) Configuración de los pines analógicos / digitales y la tensión de referencia

La señal de entrada a nuestro conversor ADC va a ser, como es lógico, la tensión de salida del circuito acondicionador de señal. La configuración de los pines del puerto A, como analógicos o digitales y la tensión de referencia se realiza en el registro ADCON1.

Tenemos una única señal a convertir, y nos interesa que la tensión de referencia sea la de alimentación del PIC, este es uno de los casos que se nos permite configurar, dando a los bits PCFG[3:0] los valores 1110 lograremos que todos los pines del puerto A sean digitales, menos el RA0, que será analógico, y como tensión de referencia tendremos la de alimentación del PIC.



5 EL MÓDULO ADC

b) Seleccionar el canal analógico

El puerto A consta de 8 entradas, que para el módulo ADC serán canales, la selección del canal que convertirá este periférico la haremos utilizando el registro ADCON0. La entrada de la señal será a través de AN0, los bits CHS2:CHS0 serán 000.

c) Seleccionar el reloj de la conversión

El registro ADCON1 también nos permite configurar la frecuencia con la que se realizará la conversión, esto se selecciona mediante los bits ADCS1:ADCS0 (ADCON0[7:6]). Elegimos Fosc / 8.

d) Activar el módulo ADC

Una vez que el canal ADC, las tensiones de referencia, el reloj de conversión han sido configurados, el PIC está preparado para convertir una tensión analógica en un número binario sin signo de 10 bits. El proceso de conversión empezará cuando activemos el bit GO / DONE (ADCON0[2]). Este bit también sirve para comunicar que la conversión se ha realizado, el bit GO / DONE estará a nivel alto hasta que el ADC termine su ciclo de conversión. Cuando el bit GO / DONE esté a ‘1’, el valor de 10 bits estará en los registros ADRESH:ADRESL representando el valor de tensión existente en el pin de entrada al ADC.

e) Configuración de las interrupciones

Otra forma de determinar cuando se ha realizado la conversión completa es usando las interrupciones del ADC. Este método permite al PIC continuar con otros procesos hasta que la conversión del ADC haya finalizado y se genere automáticamente una interrupción. Para ello habrá que resetear el bit ADIF que indica que la interrupción se ha realizado por el módulo AD, habilitar las interrupciones por este periférico (ADIE = 1), habilitar las interrupciones provocadas por los periféricos (PEIE = 1), habilitar las interrupciones (GIE = 1), y por fin, después de un par de ciclos de instrucción empezar con la conversión (ADGO = 1).

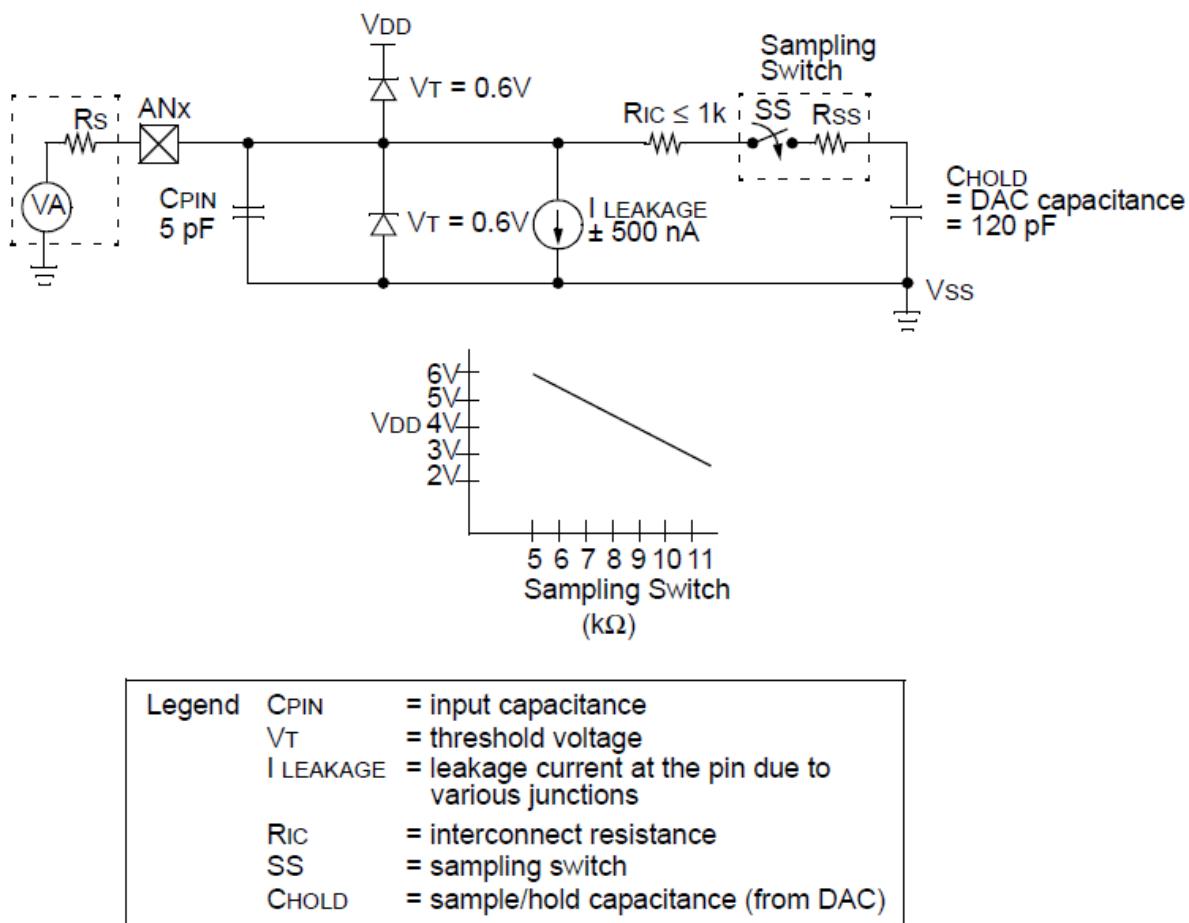


Figura 5.3. Modelo de entrada analógica.

f) Código de la ISR

Esta parte del programa se ejecuta cuando el conversor A / D ha realizado una conversión, y se lo indica al PIC activando un bit (ADIF) que informa de que se ha producido una interrupción y que esta ha sido emitida por el conversor A / D, es decir, el módulo ADC ha realizado una conversión y tiene el resultado almacenado. Lo primero que se hace es copiar el resultado del ADC de ADRESH:ADRESL para ser usado por el mismo ISR o por otra parte del programa.

Este valor procedente de la conversión, será posteriormente analizado y se hallará la temperatura que ha captado el sensor. Después se resetea el bit que ha indicado que se ha producido una interrupción en el conversor A / D, para de esta forma poder saber cuando se va a producir la próxima conversión.



5 EL MÓDULO ADC

Si la siguiente conversión ADC se tiene que realizar en otro canal de entrada, el nuevo canal de entrada deberá seleccionarse mediante los bits CHS2:CHS0. Si la siguiente conversión ADC se realiza en el mismo canal, no es necesario modificar estos bits.

Proyecto 13: Termómetro digital

A) Planteamiento del programa:

El programa consiste en implementar un termómetro digital, aplicando el módulo ADC del PIC, y una pantalla LCD para visualizar la temperatura, el rango de entrada es de 0 a 5 voltios y la temperatura va desde 0°C hasta 99°C, el sensor de temperatura puede ser el LM35 u otro sensor con esas características.

B) Esquema electrónico:

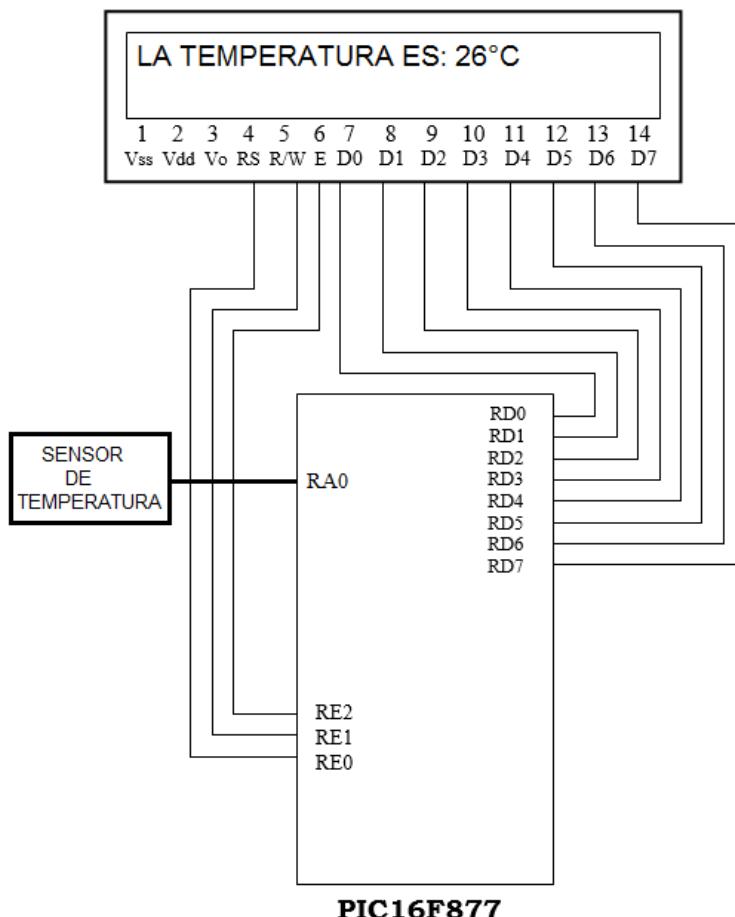


Figura 5.4. Circuito electrónico para el proyecto 13.



C) Código del programa:

```

;-----

#include"p16F877.inc"           ;incluir librerias del PIC 16F877
#include"tamper.inc"            ;incluir macros para los bancos
__config 3F31h                  ;palabra de configuración para el PIC 16F877

dato1    EQU      0x20          ;declarar la variable dato1
dato2    EQU      0x21
dato4    EQU      0x22
dato5    EQU      0x23
del_01   EQU      0x24
del_02   EQU      0x25
del_03   EQU      0x26
del_04   EQU      0x27
tmp_u    EQU      0x29
tmp_d    EQU      0x2A
adc_H    EQU      0x2D
adc_L    EQU      0x2E
adc_T    EQU      0x2F
tmpc_u   EQU      0x30
tmpc_d   EQU      0x31
dat_mot  EQU      0x34
dat_m    EQU      0x35
dat_c    EQU      0x36

ORG      0x00          ;inicio de programa en el vector reset
                    ;ir al banco1
banco1
movlw   b'00111111' ;Puerto RA0 y RA1 como entrada
movwf   TRISA
movlw   b'00000000' ;Configuración del PORTB
movwf   TRISB
movlw   b'00000000'
movwf   TRISC

;-----

```



5 EL MÓDULO ADC

```
;-----  
    clrf    TRISD      ;PORTB como salida  
    clrf    TRISE      ;PORTE como salida  
    movlw   b'10000010' ;PA como analógico y PE como digital  
    movwf   ADCON1  
    movlw   b'00000111' ;preescaler 1:256, para el retardo  
    movwf   OPTION_REG  
    movlw   b'00001000'  
    movwf   INTCON  
    banco0           ;ir al banco0  
    movlw   b'11000001'  
    movwf   ADCON0  
    movlw   0x02  
    movwf   tmp_d  
    movlw   0x01  
    movwf   tmp_u  
    clrf    PORTB  
    clrf    dat_mot  
    movlw   b'00000001'  
    movwf   dat_m  
    clrf    dat_c  
    call    lcd_msn  
repetir  call    lcd_dat  
        call    adc_dat  
        goto   repetir  
adc_dat  bcf    PIR1,ADIF  
        bsf    ADCON0,GO  
esp      btfss  PIR1,ADIF  
        goto   esp  
banco1  
    movf    ADRESL,0  
banco0  
    movwf   adc_L  
;-----
```



```
;-----  
    movf      ADRESH, 0  
    movwf     adc_H  
    movf      adc_L, 0  
    call      tabla  
    movwf     adc_T  
    movwf     tmp_u  
    swapf    adc_T, 0  
    movwf     tmp_d  
    movlw    b'00001111'  
    andwf    tmp_u, 1  
    movlw    b'00001111'  
    andwf    tmp_d, 1  
    return  
  
lcd_dat  movlw    b'10001100'  
          movwf    dat01  
          clrf     dato2  
          movlw    b'00000001'  
          movwf    dato4  
          call     enviar  
          movlw    b'00001111'  
          andwf    tmp_d, 1  
          movlw    b'00110000'  
          iorwf   tmp_d, 1  
          movf     tmp_d, 0  
          movwf    dat01  
          clrf     dato2  
          clrf     dato4  
          call     enviar  
          movlw    b'00001111'  
          andwf    tmp_u, 1  
          movlw    b'00110000'  
          iorwf   tmp_u, 1  
;-----
```



5 EL MÓDULO ADC

```
;-----  
        movf      tmp_u,0  
        movwf     dato1  
        clrf      dato2  
        clrf      dato4  
        call      enviar  
        return  
  
lcd_msn  clrf      dato2      ;para sacar datos de la tabla  
        movlw     d'5'  
        movwf      dato4      ;para discernir comandos-datos en la tabla  
lab_10   movf      dato2,0    ;sacar comandos o datos de la tabla segun el  
        call      msn_02  
        movwf      dato1  
        xorlw     0x00      ;para verificar si el ultimo dato de la  
        btfsc     STATUS,2    ;tabla ha sido enviado al LCD  
        return      ;el programa ha terminado  
        call      enviar      ;envia el dato o comando al LCD  
        incf      dato2,1    ;para sacar el siguiente dato de la tabla  
        goto      lab_10  
enviar   bcf      PORTE,2    ;deshabilita el modulo LCD  
        bsf      PORTE,1    ;LCD en modo lectura  
        bcf      PORTE,0    ;para trabajar con comandos  
        banco1  
        movlw     0xFF      ;PORTB como entrada  
        movwf      TRISD  
        banco0  
        bsf      PORTE,2    ;habilita el modulo LCD  
espera   btfsc     PORTD,7    ;espera a que el LCD este desocupado  
        goto      espera  
bcf      PORTE,2  
        banco1  
        clrf      TRISD      ;PORTB como salida  
        banco0  
;-----
```



```

;-----

        movf    dato1,0      ;carga el PORTB con un comando o dato
        movwf   PORTD
        bcf     PORTE,1      ;LCD en modo escritura
        movf    dato2,0
        xorwf   dato4,0      ;verifica si el dato a enviar es un comando
        btfss   STATUS,2      ;o es un dato del mensaje
        goto   comando       ;si es un comando se va a comando
        bsf    PORTE,0       ;para trabajar con datos
        incf   dato4,1       ;actualiza dato4
        bsf    PORTE,2       ;habilita el modulo LCD
        bcf    PORTE,2       ;deshabilita el modulo LCD
        call   delay_1       ;retardo entre caracteres a ser escritos
        return           ;como parte del mensaje

comando bsf    PORTE,2
        bcf    PORTE,2
        call   delay_1
        return

delay_1  movlw   d'1'      ;retardo aproximado de 0.33 segundos
        movwf   del_01
lab_02   movlw   d'250'
        movwf   TMR0
        bcf    INTCON,2
lab_01   btfss   INTCON,2
        goto   lab_01
        decfsz  del_01,1
        goto   lab_02
        return

delay_2  movlw   d'15'     ;retardo aproximado de 0.33 segundos
        movwf   del_02
lab_04   clrf    TMR0
        bcf    INTCON,2
lab_03   btfss   INTCON,2
;-----
```



5 EL MÓDULO ADC

```
;-----  
      goto      lab_03  
      decfsz   del_02,1  
      goto      lab_04  
      return  
  
delay_3  movlw     d'1'          ;retardo aproximado de 0.33 segundos  
          movwf    del_03  
  
lab_206  movlw     d'100'  
          movwf    TMR0  
          bcf      INTCON,2  
  
lab_205  btfss    INTCON,2  
          goto     lab_205  
          decfsz   del_03,1  
          goto     lab_206  
          return  
  
msn_02   addwf    PCL,1  
          retlw    b'00000001' ;comando CLEAR DISPLAY  
          retlw    b'00000110' ;camando ENTRY MODE SET  
          retlw    b'00001100' ;comando DISPLAY ON / OFF  
          retlw    b'00111111' ;comando FUNCTION SET  
          retlw    b'10000000' ;comando SET DDRAM ADDRESS  
          retlw    'T'        ;primer elemento del mensaje superior  
          retlw    'E'  
          retlw    'M'  
          retlw    'P'  
          retlw    'E'  
          retlw    'R'  
          retlw    'A'  
          retlw    'T'  
          retlw    'U'  
          retlw    'R'  
          retlw    'A'  
  
;-----
```



```
;-----  
        retlw    `:'  
        retlw    `'  
        retlw    `'  
        retlw    0xDF  
        retlw    `C'  
        retlw    0x00      ;indica al programa fin del mensaje  
  
ORG     0x500  
  
tabla  bsf     PCLATH,2  
          bcf     PCLATH,1  
          bsf     PCLATH,0  
          addwf   PCL,1  
          retlw   0x00  
          retlw   0x01  
          retlw   0x01  
          retlw   0x02  
          retlw   0x02  
          retlw   0x03  
          retlw   0x03  
          retlw   0x04  
          retlw   0x04  
          retlw   0x05  
          retlw   0x05  
          retlw   0x06  
          retlw   0x06  
          retlw   0x07  
          retlw   0x07  
          retlw   0x08  
          retlw   0x08  
          retlw   0x09  
          retlw   0x09  
          retlw   0x10  
          retlw   0x10  
  
;-----
```



5 EL MÓDULO ADC

```
;-----  
    retlw 0x11  
    retlw 0x11  
    retlw 0x12  
    retlw 0x12  
    retlw 0x13  
    retlw 0x13  
    retlw 0x14  
    retlw 0x14  
    retlw 0x15  
    retlw 0x15  
    retlw 0x16  
    retlw 0x16  
    retlw 0x17  
    retlw 0x17  
    retlw 0x18  
    retlw 0x18  
    retlw 0x19  
    retlw 0x19  
    retlw 0x20  
    retlw 0x20  
    retlw 0x21  
    retlw 0x21  
    retlw 0x22  
    retlw 0x22  
    retlw 0x23  
    retlw 0x23  
    retlw 0x24  
    retlw 0x24  
    retlw 0x25  
    retlw 0x25  
    retlw 0x26  
;-----
```



```
;-----  
    retlw 0x26  
    retlw 0x27  
    retlw 0x27  
    retlw 0x28  
    retlw 0x28  
    retlw 0x29  
    retlw 0x29  
    retlw 0x30  
    retlw 0x30  
    retlw 0x31  
    retlw 0x31  
    retlw 0x32  
    retlw 0x32  
    retlw 0x33  
    retlw 0x33  
    retlw 0x34  
    retlw 0x34  
    retlw 0x35  
    retlw 0x35  
    retlw 0x36  
    retlw 0x36  
    retlw 0x37  
    retlw 0x37  
    retlw 0x38  
    retlw 0x38  
    retlw 0x39  
    retlw 0x39  
    retlw 0x40  
    retlw 0x40  
    retlw 0x41  
    retlw 0x41  
;-----
```



5 EL MÓDULO ADC

```
;-----  
    retlw 0x42  
    retlw 0x42  
    retlw 0x43  
    retlw 0x43  
    retlw 0x44  
    retlw 0x44  
    retlw 0x45  
    retlw 0x45  
    retlw 0x46  
    retlw 0x46  
    retlw 0x47  
    retlw 0x47  
    retlw 0x48  
    retlw 0x48  
    retlw 0x49  
    retlw 0x49  
    retlw 0x50  
    retlw 0x50  
    retlw 0x51  
    retlw 0x51  
    retlw 0x52  
    retlw 0x52  
    retlw 0x53  
    retlw 0x53  
    retlw 0x54  
    retlw 0x54  
    retlw 0x55  
    retlw 0x55  
    retlw 0x56  
    retlw 0x56  
;-----
```



```
;-----  
    retlw 0x57  
    retlw 0x57  
    retlw 0x58  
    retlw 0x58  
    retlw 0x59  
    retlw 0x59  
    retlw 0x60  
    retlw 0x60  
    retlw 0x61  
    retlw 0x61  
    retlw 0x62  
    retlw 0x62  
    retlw 0x63  
    retlw 0x63  
    retlw 0x64  
    retlw 0x64  
    retlw 0x65  
    retlw 0x65  
    retlw 0x66  
    retlw 0x66  
    retlw 0x67  
    retlw 0x67  
    retlw 0x68  
    retlw 0x68  
    retlw 0x69  
    retlw 0x69  
    retlw 0x70  
    retlw 0x70  
    retlw 0x71  
    retlw 0x71  
;-----
```



5 EL MÓDULO ADC

```
;-----  
    retlw    0x72  
    retlw    0x72  
    retlw    0x73  
    retlw    0x73  
    retlw    0x74  
    retlw    0x74  
    retlw    0x75  
    retlw    0x75  
    retlw    0x76  
    retlw    0x76  
    retlw    0x77  
    retlw    0x77  
    retlw    0x78  
    retlw    0x78  
    retlw    0x79  
    retlw    0x79  
    retlw    0x80  
    retlw    0x80  
    retlw    0x81  
    retlw    0x81  
    retlw    0x82  
    retlw    0x82  
    retlw    0x83  
    retlw    0x83  
    retlw    0x84  
    retlw    0x84  
    retlw    0x85  
    retlw    0x85  
    retlw    0x86  
    retlw    0x86  
    retlw    0x87  
;-----
```





5 EL MÓDULO ADC

D) Simulación del programa:

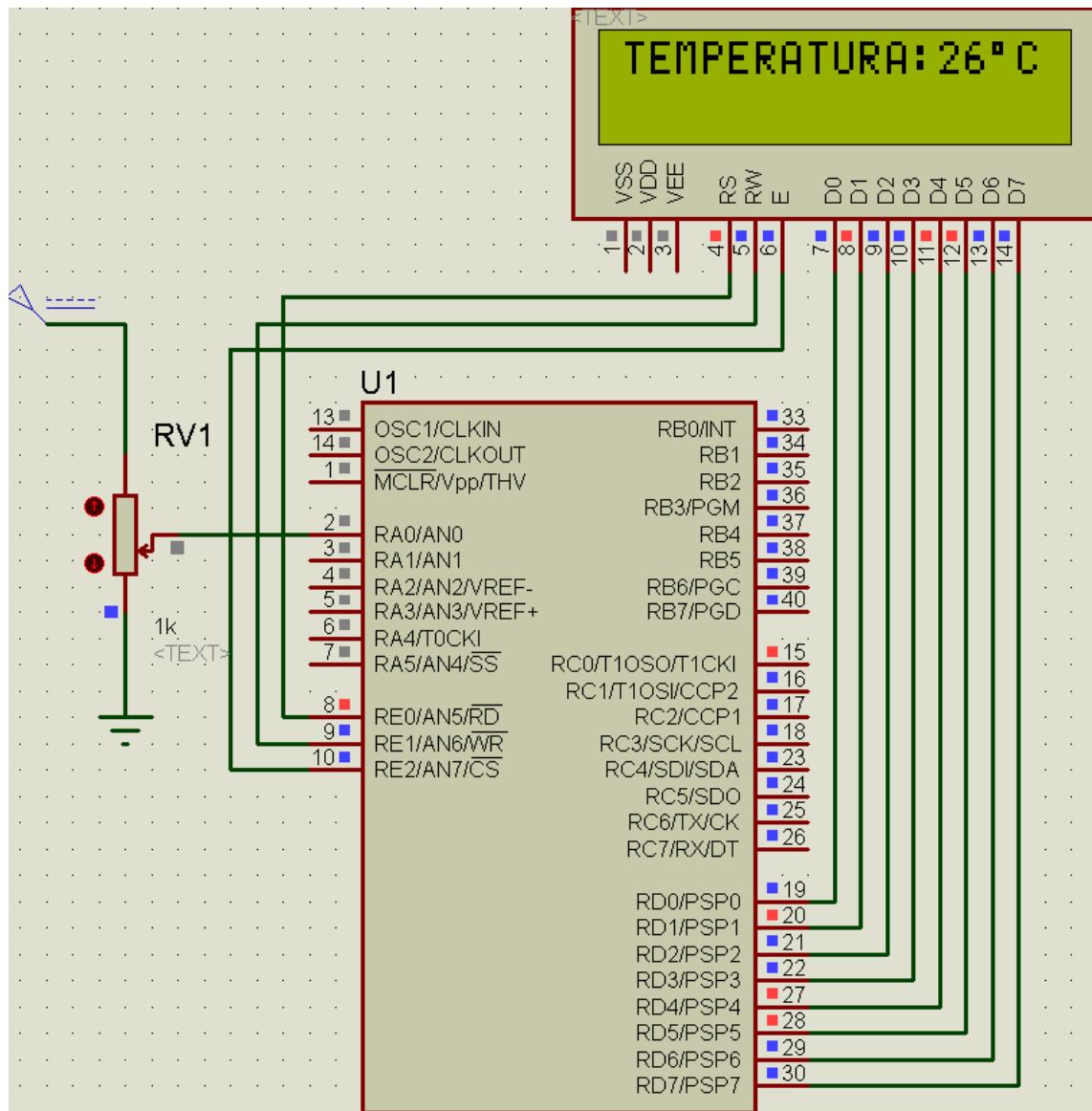


Figura 5.5. Circuito electrónico implementado en el PROTEUS, en este circuito se realizó la simulación del programa, el cual funcionó de manera normal.



Actividades

A) Responda los espacios en blanco:

- a) La mínima variación a la entrada para que se produzca un cambio a la salida del ADC se conoce como
- b) Los registros de estado y control para el modulo ADC son: y
- c) El registro es utilizado para configurar los pines asignados al ADC como digitales o analógicos.
- d) El módulo ADC puede trabajar en modo: o
.....

B) Responda “V” si es verdadero y “F” si es falso, a las siguientes afirmaciones.

- a) El registro ADRESL, siempre almacena los 8 bits menos significativos después de una conversión ()
- b) El registro ADRESL se encuentra en el banco 0 del PIC 16F877.....()
- c) La velocidad de conversión del módulo ADC depende únicamente del valor del registro SPBRG.....()
- d) Si habilitamos el puerto serie, entonces el puerto RA0 trabaja como entrada.....()

C) Colocar en orden cada una de las siguientes instrucciones:

movwf PORTB
movf ADRESH,W
bsf STATUS,5
bcf STATUS,5
movf ADRESL,W
movwf PORTD



5 EL MÓDULO ADC

D) Marque la alternativa correcta.

1. El módulo ADC utiliza dos registros, donde almacena el dato que recibe, uno de estos registros es:
a) TXREG b) ADRESL c) RCREG d) RCIF e) N.A
2. Si deseamos utilizar el puerto RE0 como entrada analógica de un PIC 16F877, el valor del registro ADCON0 vale:
a) xx011xxx b) xxx101xx c) xx110xxx d) xx100xxx e) N.A.
3. Para un ADC cuyo rango de voltaje es: 0 – 5 V y una palabra de 10 bits, después de realizar una conversión la palabra vale 0000000111b, entonces se puede afirmar que el voltaje a la entrada del ADC es:
a) 14.64mV b) 24.44mV c) 36 mV d) 50.44 mV e) N.A.





ESTE TEXTO HA SIDO ELABORADO PENSANDO EN LOS ESTUDIANTES QUE VAN A INTERACTUAR POR PRIMERA VEZ CON LOS MICROCONTROLADORES PIC, DE AHÍ SU ENFOQUE DIDÁCTICO Y 13 PROYECTOS QUE LES PERMITIRÁ APRENDER DE UNA FORMA SIMPLE LA PROGRAMACIÓN Y APLICACIÓN DE ESTOS DISPOSITIVOS, PROXIMAMENTE SE ESTARÁ TERMINANDO EL LIBRO AUTÓMATAS PROGRAMABLES PLC, TAMBIÉN CON UN ENFOQUE DIDÁCTICO, DE ESTA FORMA APORTAR A LA FORMACIÓN DE NUESTROS ESTUDIANTES.

