

Microcontroladores PIC: Diseño práctico de aplicaciones – 2da Edición

Jose María Angulo Usategui, Ignacio Angulo Martínez
ISBN: 8484124960
MGH – 1999

CONTENIDO

Prólogo XI

Capítulo 1. MICOCRONTROLADORES PROGRAMABLES: LA SOLUCIÓN ESTÁ EN UN CHIP 1

1.1. ¿Qué es un microcontrolador? 1
1.1.1. Diferencia entre microprocesador y microcontrolador 3
1.2. Arquitectura interna 5
1.2.1. El Procesador 5
1.2.2. Memoria de programa 6
1.2.3. Memoria de datos 8
1.2.4. Líneas de E/S para los controladores de periféricos 8
1.2.5. Recursos auxiliares 8
1.3. Programación de los microcontroladores 9
1.4. Instrumental de trabajo 9

Programar PIC es fácil 14

El primer programa: Aprendiendo a sumar 14
Enunciado 16
Edición del programa 17
Propuesta de trabajo 18
Manejando el simulador SIMUPIC'84: El primer contacto 18
¿Repetimos la suma? 19
La eficiencia del Ensamblador frente a la comodidad del *C* 20
Diseñar aplicaciones con PIC es fácil 22
Presentación de los periféricos más simples 22
Periféricos digitales de entrada 22
Pulsadores 23
Interruptores 23
Circuito antirrebotes 24
Entradas digitales optoacopladas 25

Capítulo 2. MICROCONTROLADORES DE 8 BITS: LOS «PIC» 27

2.1. La elección 27
2.2. La familia de los PIC 29
2.2.1. La gama enana: PIC12C(F)XXX de 8 patitas 29
2.2.2. Gama baja o básica: PIC16C5X con instrucciones de 12 bits 30
2.2.3. Gama media: PIC16CXXX con instrucciones de 14 bits 31
2.2.4. Gama alta: PIC17CXXX con instrucciones de 16 bits 32
2.3. Descripción general y aplicaciones 33

Programar PIC es fácil 37

El segundo programa de calentamiento: Explorando interruptores y encendiendo luces 37
Enunciado 39
Probando el programa con el SIMUPIC'84 41
Programando en C 41
Propuesta de un ejercicio para «subir nota» 42
Diseñar aplicaciones con PIC es fácil 42
Presentación de los periféricos más simples 42
Periféricos digitales de salida 42
Diodos led 42
Relés 43
Microrrelé 44
Salidas optoacopladas 45
Zumbador 45
Control de cargas gobernadas por corriente alterna 45
Interface RS-232 46
Motor PAP 47

Capítulo 3. EL PRIMER CONTACTO CON EL PIC16X84 49

3.1. Pequeño, pobre, pero resultón 49
3.1.1. Variantes del PIC16X8X 50
3.1.2. Diferencias del PIC16C84 con los demás PIC16X8X 52
3.2. El aspecto externo 52
3.3. La frecuencia de funcionamiento. El reloj 53
3.3.1. Tipos de osciladores 55
3.4. Reinicialización o Reset 57
3.5. El primer circuito práctico 57
3.6. Una solución al diseño de proyectos: La tarjeta de prototipos 59

Programar PIC es fácil 62

El tercer programa con diseño de hardware 62
Enunciado: Más interruptores y diodos led 62

Probando el programa con el SIMUPIC'84 65
Programando en C 65
¿Se atreve con este ejercicio? Primeros pasos con el Micro'PIC Trainer 66
Diseñar aplicaciones con PIC es fácil 67
Visualizadores con displays de 7 segmentos 67
Teclado 68
Teclado/display 69
Pantalla de cristal líquido 71

Capítulo 4. EN EL INTERIOR DEL PROCESADOR 73

4.1. Una arquitectura moderna y revolucionaria 73
4.1.1. Diferencias de la arquitectura del PIC16C84 con la de sus variantes. 76
4.2. Memoria de programa 77
4.2.1. El Contador de Programa y la Pila 77
4.3. Memoria de datos RAM 78
4.3.1. Memoria de datos RAM en el PIC16F84 y el PIC16CR84 81
4.3.2. Direccionamiento de la memoria de datos 83
4.4. El registro de Estado 85

Programar PIC es fácil 87

El cuarto programa: Contando y visualizando 87
Enunciado 87
Probando con el Simupic'84 89
Lo montamos y no funciona 90
Poniéndolo a punto 90
Programamos en C 90

Capítulo 5. LOS RECURSOS FUNDAMENTALES: TEMPORIZADORES, PUERTAS DE E/S Y EEPROM DE DATOS 93

5.1. El papel de los temporizadores 93
5.2. Temporizador/Contador TMR0 93
5.2.1. El registro OPTION 96
5.3. El Perro Guardián (WDT) 97
5.4. Las puertas de E/S 99
5.4.1. La Puerta A 100
5.4.2. La Puerta B 102
5.5. La Palabra de Configuración 102
5.5.1. Palabras de Identificación (ID) 104
5.6. La memoria EEPROM de datos 104

Programar PIC es fácil 108

El quinto programa de aprendizaje: Controlando el tiempo 108
Enunciado 108

Programa de parpadeo del diodo led 110
¡Error de ensamblado! 111
Probando con el Simupic'84 112
Programando en C 113
Montaje y experimentación en el Micro'PIC Trainer 113
Correcciones al programa 115
¿Se atreve con este ejercicio? 116

Capítulo 6. INTERRUPCIONES, RESET Y RECURSOS AUXILIARES 117

6.1. Importancia de las interrupciones 117
6.2. Causas de interrupción 118
6.2.1. El Registro de control de interrupciones INTCON 118
6.2.2. Interrupción Externa INT 122
6.2.3. Interrupción por desbordamiento del TMR0 123
6.2.4. Interrupción por cambio de estado en las líneas RB7 : RB4 de la Puerta B 124
6.2.5. Interrupción por finalización de la escritura en la EEPROM de datos 124
6.3. Reinicialización o Reset 125
6.3.1. Reset por fallo en la alimentación («Brown Out») 127
6.4. El modo de Reposo o de bajo consumo 129
6.5. Programación en serie de los PIC16X8X 129

Programar PIC es fácil 131

El sexto ejercicio práctico: Manejando interrupciones 131
Programa Parpadeo 132
Simulando con el Simupic'84 134
Programando en C 134
Le proponemos un ejercicio lleno de interrupciones 136

Capítulo 7. MANEJANDO EL REPERTORIO DE INSTRUCCIONES 139

7.1. Repertorio RISC 139
7.2. Tipos de formato 140
7.2.1. Operaciones orientadas a manejar registros de tamaño byte 141
7.2.2. Operaciones orientadas a manejar bits 142
7.2.3. Operaciones que manejan un valor inmediato o literal 143
7.2.4. Operaciones incondicionales de control de flujo del programa 143
7.2.5. Operaciones de salto condicional 143
7.3. Precisiones sobre nomenclatura y símbolos 144
7.4. Instrucciones que manejan registros 146
7.5. Instrucciones que manejan bits 147
7.6. Instrucciones de brinco (*skip*) 148
7.7. Instrucciones que manejan operandos inmediatos 149
7.8. Instrucciones de control y especiales 150

Programar PIC es fácil 153

Séptimo ejercicio: Programas con un poco de todo. Manejando el registro de trabajo W 153
Manejando igualdades y desigualdades 154
Explorando la causa de interrupción 154
Proyecto para manejar un display de 7 segmentos 155
Simulando con el SIMUPIC'84 158
Programando en C 159
Probando en el Micro'PIC Trainer 160
¿Se atreve con esto? 160

Capítulo 8. HERRAMIENTAS Y DISEÑO DE PROYECTOS 161

8.1. Fases de diseño 161
8.2. Herramientas accesibles 163
8.2.1. La cabeza 164
8.2.2. Editor de textos 164
8.2.3. Ensamblador o Compilador 164
8.2.4. Simulador software 165
8.2.5. El grabador 165
8.2.6. Sistema de desarrollo 166
8.2.7. La tarjeta de prototipos 166
8.3. Emulador en circuito 167
8.4. Introducción al diseño de proyectos: Puesta en hora 168
8.5. Un dado electrónico 173
8.6. Semáforo regulable 175
8.7. Presentador de mensajes 177
8.8. Clave de acceso 177

Apéndice A: Guía Rápida del PIC16X84 181

Apéndice B: Guía del usuario del SIMUPIC'84 191

Apéndice C: Breve descripción del Micro'PIC Trainer 203

Apéndice D : La familia PIC al completo 211

Apéndice E: Contenido del CD-ROM 215

Apéndice F: Proyectos con el PIC16F84 217

Bibliografía y direcciones de interés relacionadas con los PIC 291

Índice 293

PRÓLOGO

Los microcontroladores están conquistando el mundo. Están presentes en nuestro trabajo, en nuestra casa y en nuestra vida, en general. Se pueden encontrar controlando el funcionamiento de los ratones y teclados de los computadores, en los teléfonos y en los hornos microondas y los televisores de nuestro hogar. Pero la invasión acaba de comenzar y el nacimiento del siglo XXI será testigo de la conquista masiva de estos diminutos computadores, que gobernarán la mayor parte de los aparatos que fabricamos y usamos los humanos.

Las extensas áreas de aplicación de los microcontroladores, que se pueden considerar ilimitadas, exigirán un gigantesco trabajo de diseño y fabricación.

Aprender a manejar y aplicar microcontroladores sólo se consigue desarrollando prácticamente diseños reales. Sucede lo mismo que con cualquier instrumento musical, cualquier deporte y con muchas actividades.

El objetivo primordial que nos ha movido al escribir este libro es facilitar el camino al lector para que se entusiasme y utilice los microcontroladores. Desde el primer tema le proponemos programas para confeccionar y simular, y desde el segundo, proyectos con periféricos.

La filosofía y el método que hemos empleado nos ha dado excelentes resultados en nuestros entornos de trabajo. Un autor es profesor y enseña los microcontroladores en la Universidad, y el otro los aplica en los proyectos que dirige en una empresa. Se trata de describir un modelo simple de microcontrolador de manera lo suficientemente clara para que el lector quiera usarlo. Cuando se conoce un modelo y se han desarrollado proyectos simples, es inmediato pasar a manejar otros microcontroladores y acometer complejos diseños. En ese momento, las posibilidades sólo están limitadas por la imaginación y la capacidad de trabajo.

Hemos elegido una familia de microcontroladores: los **PIC** de **Microchip Technology Inc.** Dentro de ella seleccionamos un modelo concreto: el **PIC16X84**, con varias versiones. Las razones que nos han empujado a esta decisión es que estos microcontroladores son sencillos, modernos, rápidos, baratos, se pueden escribir los programas y borrarlos montones de veces, disponen de una excelente documentación y las herramientas de desarrollo son asequibles para todos los bolsillos.

Para dar consistencia a nuestro interés por la experimentación, el libro incluye un CD que contiene varios tutoriales, un simulador y los programas necesarios para editar, ensamblar, depurar y simular todos los ejercicios y proyectos que se proponen en el texto y que se hallan resueltos.

Además, para que el lector no encuentre una barrera en el coste de las herramientas, se han elegido aquellas que por su precio se acomodan a cualquier presupuesto.

Cada tema o capítulo consta de dos partes diferenciadas. La primera tiene un marcado carácter teórico y va salteada con ejemplos y programas resueltos. La segunda es eminentemente práctica y contiene ejercicios, programas y diseños que el lector puede comprobar con el simulador del disquete y construir con el mínimo desembolso.

En el Capítulo 1 se describen las características generales de la arquitectura, la programación, las aplicaciones y las herramientas que usan los microcontroladores. En la parte práctica se propone el primer programa que puede analizarse con el simulador. También se estudian las diferencias entre el lenguaje Ensamblador y el C. Finalmente, se comentan los esquemas de conexionado de los periféricos digitales de entrada más comunes.

El Capítulo 2 expone, en la parte teórica, las características de los microcontroladores PIC, y en la práctica, resuelve el primer diseño con interruptores y diodos led y continúa la descripción de los periféricos de salida.

En el Capítulo 3 se toma el primer contacto con el PIC16X84 y se configura su circuitería fija; también se sigue ampliando el diseño práctico y se recomienda el uso de la tarjeta de prototipos. Se completa el recorrido de los periféricos típicos estudiando los displays, los teclados y las pantallas LCD.

La moderna arquitectura del procesador que contienen los PIC se analiza en el Capítulo 4. Se proponen diseños de visualización y medida de tiempos y se confeccionan sus programas en Ensamblador y C.

Los recursos fundamentales de los PIC: temporizadores, puertas de E/S y la memoria EEPROM de datos se describen, con numerosos ejemplos, en el Capítulo 5. En su parte práctica los proyectos se destinan fundamentalmente a utilizar los temporizadores, las puertas de E/S y la EEPROM de datos.

Las interrupciones, tanto en el aspecto teórico como en el práctico, constituye el tema principal al que se destina el Capítulo 6.

El repertorio de instrucciones, analizado desde el punto de vista de su aplicación con numerosos programas de ejemplo, es el objeto de la teoría del Capítulo 7. En cuanto a prácticas, se proponen diversos caminos para el diseño de un sistema de visualización de un conjunto de entradas digitales.

El Capítulo 8 describe las herramientas más adecuadas para el desarrollo de las aplicaciones con microcontroladores y proporciona una documentada información sobre el diseño del hardware y software de varios interesantes proyectos, tales como el dado electrónico, un semáforo programable, un reloj digital, un presentador de mensajes y el control de accesos mediante una clave secreta.

Además, el libro dispone en su parte final de unos Apéndices muy útiles para el trabajo profesional. En uno se hace un resumen de las características más importantes del PIC16X84. En otro se ofrece un completo Manual de Usuario del simulador **SIMUPIC'84**, que contiene el CD que acompaña al libro. Otro apéndice está dedicado a dar una visión de la arquitectura de un sistema de desarrollo muy recomendado, el **Micro'PIC Trainer**. Hay un apéndice que recoge todos los microcontroladores de la familia PIC, con sus características más importantes. También se incluye un índice del contenido del CD, en el que se pueden encontrar todos los programas de los diseños propuestos en la obra.

Nuestro agradecimiento a **Microchip** por su constante esfuerzo en ayudar a los usuarios de sus productos. Los señores Benito Moreno y Pepe Bustamante, de **Sagitrón**, distribuidor en España de **Microchip**, han colaborado con nosotros en todo lo que hemos necesitado.

La empresa bilbaina **Microsystems Engineering** ha puesto a nuestra disposición todo tipo de herramientas, proyectos y material para desarrollar programas y diseños. También nos ha proporcionado la versión del simulador software **SIMUPIC'84**, que se incluye en el CD y que ha sido revisada en la segunda edición por Gorka Elexgaray para poder ser ejecutada en los modernos computadores dotados de los veloces *Pentium*. Quedamos muy reconocidos a Goyo y Jorge, de **Bilbao Electrónica**, por todas las facilidades que nos han dado.

En la segunda edición se han corregido erratas, se han revisado y actualizado algunos conceptos y se ha añadido el Apéndice F, que contiene una colección de proyectos de complejidad progresiva descritos detalladamente y que pueden implementarse fácilmente y con materiales muy comunes. Está basado en dos kits de prácticas que comercializa **Microsystems Engineering** (Proyectos Básicos y Proyectos Típicos).

La mayor aportación de la nueva edición es la inclusión del CD multimedia que contiene, además de todos los programas de trabajo fundamentales y los ejercicios y proyectos resueltos que aparecen en los capítulos y apéndices del libro, diversos tutoriales que, combinando la imagen y el sonido, ofrecen una explicación clara y gráfica para aprender a manejar los recursos lógicos y físicos y para la construcción paso a paso y puesta a punto del sistema de desarrollo **Micro'PIC Trainer**. Esta fantástica herramienta didáctica la ha creado el grupo de alumnos de Ingeniería Informática de **ESIDE** en la **Universidad de Deusto**, formado por Tatiana Macías, Roberto Urrutikoetxea, Sergio Zamora y Maite Zumárraga, bajo la dirección de la profesora Susana Romero.

MICROCONTROLADORES PROGRAMABLES: LA SOLUCIÓN ESTÁ EN UN CHIP

1

1.1. ¿QUÉ ES UN MICROCONTROLADOR?

Es un circuito integrado programable que contiene todos los componentes de un computador. Se emplea para controlar el funcionamiento de una tarea determinada y, debido a su reducido tamaño, suele ir incorporado en el propio dispositivo al que gobierna. Esta última característica es la que le confiere la denominación de «controlador incrustado» (*embedded controller*).

El microcontrolador es un computador **dedicado**. En su memoria sólo reside un programa destinado a gobernar una aplicación determinada; sus líneas de entrada/salida soportan el conexionado de los sensores y actuadores del dispositivo a controlar y todos los recursos complementarios disponibles tienen como única finalidad atender sus requerimientos. Una vez programado y configurado el microcontrolador solamente sirve para gobernar la tarea asignada.

Un microcontrolador es un computador completo, aunque de limitadas prestaciones, que está contenido en el chip de un circuito integrado y se destina a gobernar una sola tarea.

El número de productos que funcionan en base a uno o varios microcontroladores aumenta de forma exponencial. No es aventurado pronosticar que en el siglo XXI habrá pocos elementos que carezcan de microcontrolador. En esta línea de prospección del futuro, la empresa **Dataquest** calcula que en cada hogar americano existirá un promedio de 240 microcontroladores para el año 2000.

La industria Informática acapara gran parte de los microcontroladores que se fabrican. Casi todos los periféricos del computador, desde el ratón o el teclado hasta la impresora, son regulados por el programa de un microcontrolador. (Véase Figura 1.1.)

Los electrodomésticos de línea blanca (lavadoras, hornos, lavavajillas, etc.) y de línea marrón (televisores, vídeos, aparatos musicales, etc.) incorporan numerosos microcontroladores. Igualmente, los sistemas de supervisión, vigilancia y alarma en los edificios utilizan estos chips para optimizar el rendimiento de ascensores, calefacción, aire acondicionado, alarmas de incendio, robo, etc.

Las comunicaciones y sus sistemas de transferencia de información utilizan profusamente

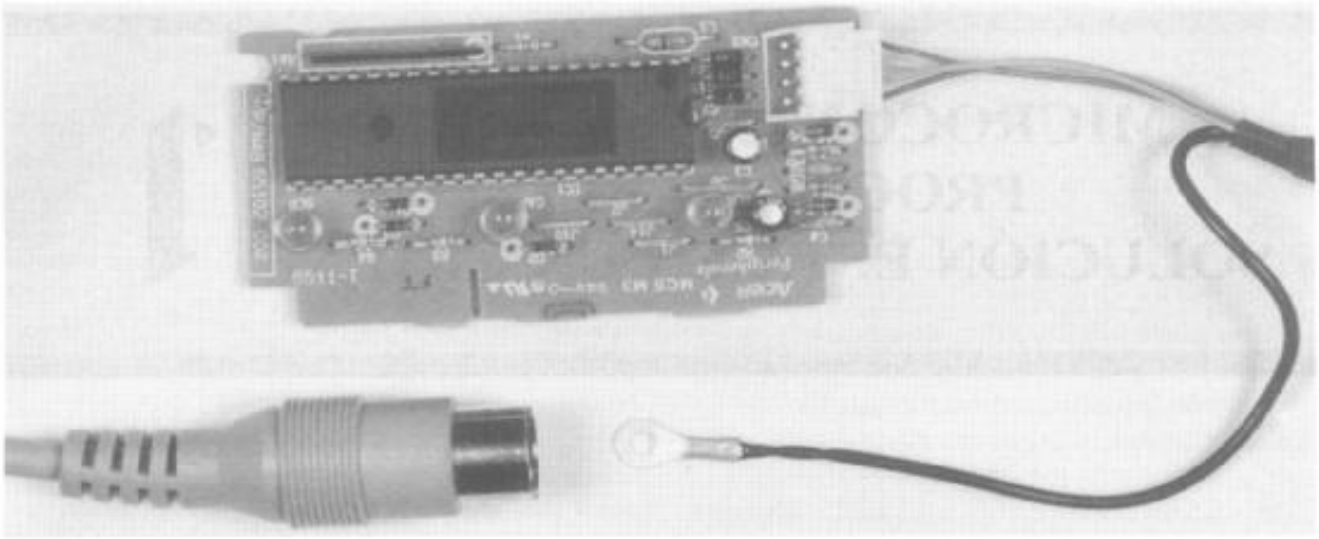


Figura 1.1. *Fotografía de la tarjeta electrónica usada para el control del teclado de un computador. El único circuito integrado es un microcontrolador de INTEL que está encargado de detectar la tecla pulsada y transferir su código ASCII.*

estos pequeños computadores incorporándolos en los grandes automatismos y en los modernos teléfonos.

La instrumentación y la electromedicina son dos campos idóneos para la implantación de estos circuitos integrados. Una importante industria consumidora de microcontroladores es

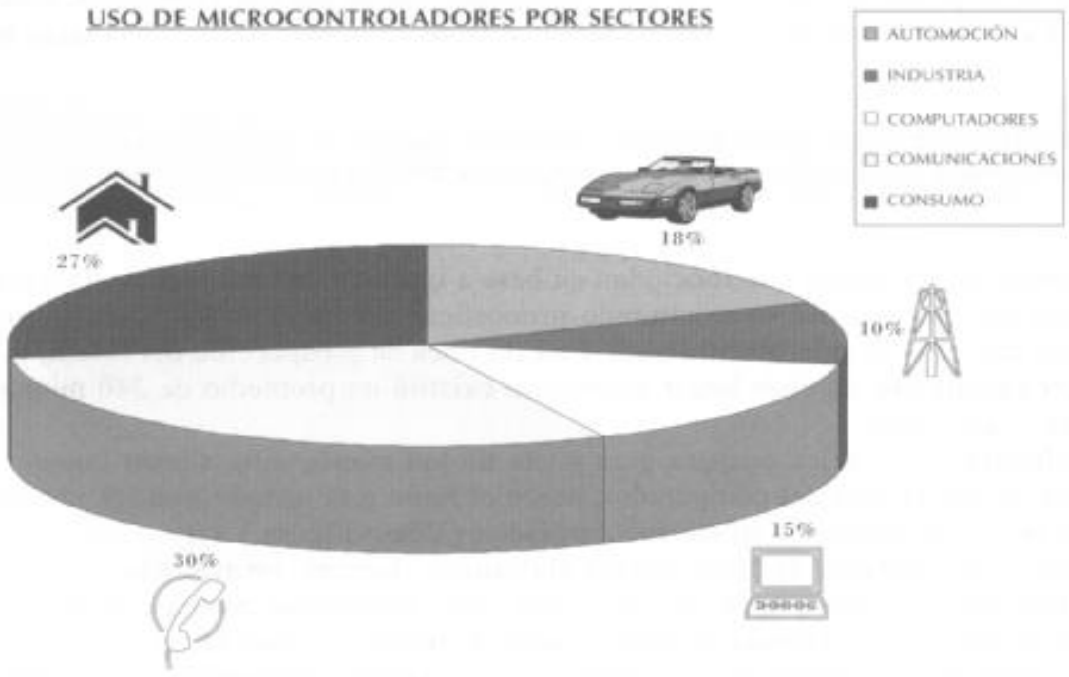


Figura 1.2. *Gráfico que muestra la distribución de la producción mundial de microcontroladores en las diversas áreas de aplicación.*

la de automoción, que los aplica en el control de aspectos tan populares como la climatización, la seguridad y los frenos ABS.

Las comunicaciones y los productos de consumo general absorben más de la mitad de la producción de microcontroladores. El resto se distribuye entre el sector de la automoción, los computadores y la industria.

1.1.1. Diferencia entre microprocesador y microcontrolador

El microprocesador es un circuito integrado que contiene la Unidad Central de Proceso (UCP), también llamada procesador, de un computador. La UCP está formada por la Unidad de Control, que interpreta las instrucciones, y el Camino de Datos, que las ejecuta.

Las patitas de un microprocesador sacan al exterior las líneas de sus buses de direcciones, datos y control, para permitir conectarle con la Memoria y los Módulos de E/S y configurar un computador implementado por varios circuitos integrados. Se dice que un microprocesador es un **sistema abierto** porque su configuración es variable de acuerdo con la aplicación a la que se destine. (Figura 1.3.)

Un microprocesador es un **sistema abierto** con el que puede construirse un computador con las características que se desee, acoplándole los módulos necesarios.
Un microcontrolador es un **sistema cerrado** que contiene un computador completo y de prestaciones limitadas que no se pueden modificar.

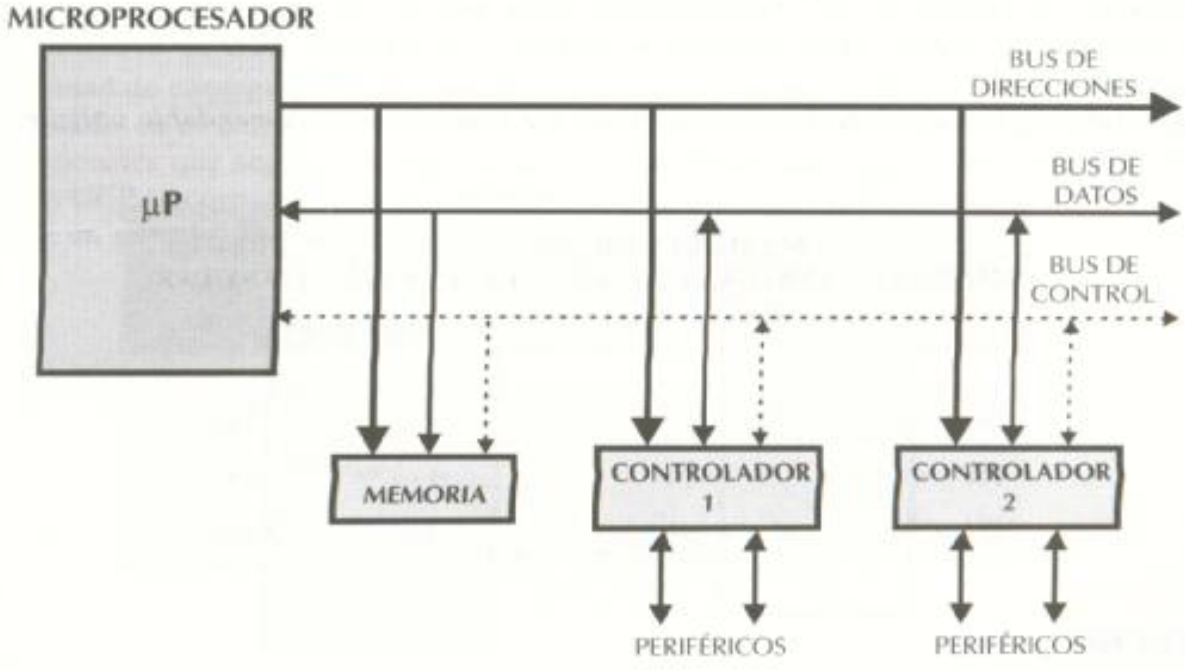


Figura 1.3. *Estructura de un sistema abierto basado en un microprocesador. La disponibilidad de los buses en el exterior permite que se configure a la medida de la aplicación.*

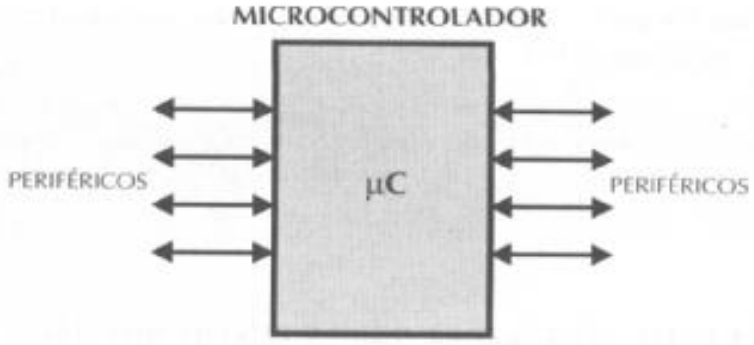


Figura 1.4. *El microcontrolador es un sistema cerrado. Todas las partes del computador están contenidas en su interior y sólo salen al exterior las líneas que gobiernan los periféricos.*

Si sólo se dispusiese de un modelo de microcontrolador, éste debería tener muy potenciados todos sus recursos para poderse adaptar a las exigencias de las diferentes aplicaciones. Esta potenciación supondría en muchos casos un despilfarro. En la práctica cada fabricante de microcontroladores oferta un elevado número de modelos diferentes, desde los más sencillos hasta los más poderosos. Es posible seleccionar la capacidad de las memorias, el número de líneas de E/S, la cantidad y potencia de los elementos auxiliares, la velocidad de funcionamiento, etc. Por todo ello, un aspecto muy destacado del diseño es la selección del microcontrolador a utilizar.

EJEMPLO

Un horno microondas se gobierna mediante un microcontrolador en el que se almacena el programa de control de 382 instrucciones, cada una de las cuales ocupa una palabra de la memoria de código. Para soportar el teclado y la pantalla LCD se necesitan 12 líneas de E/S.

¿Cuál de los tres siguientes microcontroladores será el más recomendable utilizar?

MODELO	MEMORIA DE INSTRUCCIONES (PALABRAS)	LÍNEAS E/S	PRECIO (PESETAS)
PIC16F83	512	13	380
PIC16C84	1.024	13	690
PI16C74	4.096	33	2.100

SOLUCIÓN

Para esta aplicación es suficiente el PIC16F83 y su empleo supone una importante economía.

1.2. ARQUITECTURA INTERNA

Un microcontrolador posee todos los componentes de un computador, pero con unas características fijas que no pueden alterarse.

Las partes principales de un microcontrolador son:

1. Procesador
2. Memoria no volátil para contener el programa
3. Memoria de lectura y escritura para guardar los datos
4. Líneas de E/S para los controladores de periféricos:
   a) Comunicación paralelo
   b) Comunicación serie
   c) Diversas puertas de comunicación (bus I^2C, USB, etc.)
5. Recursos auxiliares:
   a) Circuito de reloj
   b) Temporizadores
   c) Perro Guardián («watchdog»)
   d) Conversores AD y DA
   e) Comparadores analógicos
   f) Protección ante fallos de la alimentación
   g) Estado de reposo o de bajo consumo

A continuación se pasa revista a las características más representativas de cada uno de los componentes del microcontrolador.

1.2.1. El Procesador

La necesidad de conseguir elevados rendimientos en el procesamiento de las instrucciones ha desembocado en el empleo generalizado de procesadores de arquitectura **Harvard** frente a los tradicionales que seguían la arquitectura de **von Neumann**. Esta última se caracterizaba porque la UCP se conectaba con una memoria única, donde coexistían datos e instrucciones, a través de un sistema de buses. (Véase Figura 1.5.)

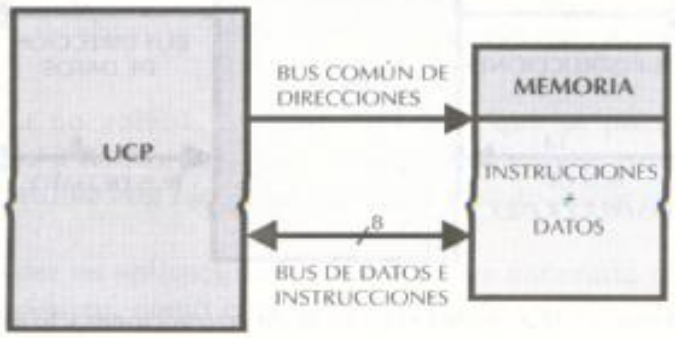


Figura 1.5. *En la arquitectura de «von Neumann» la UCP se comunicaba a través de un sistema de buses con la Memoria, donde se guardaban las instrucciones y los datos.*

En la arquitectura **Harvard** son independientes la memoria de instrucciones y la memoria de datos y cada una dispone de su propio sistema de buses para el acceso. Esta dualidad, además de propiciar el paralelismo, permite la adecuación del tamaño de las palabras y los buses a los requerimientos específicos de las instrucciones y de los datos. También la capacidad de cada memoria es diferente. (Figura 1.6.)

El procesador de los modernos microcontroladores responde a la arquitectura **RISC** (Computadores de Juego de Instrucciones Reducido), que se identifica por poseer un repertorio de instrucciones máquina pequeño y simple, de forma que la mayor parte de las instrucciones se ejecuta en un ciclo de instrucción.

Otra aportación frecuente que aumenta el rendimiento del computador es el fomento del paralelismo implícito, que consiste en la segmentación del procesador (*pipe-line*), descomponiéndolo en etapas para poder procesar una instrucción diferente en cada una de ellas y trabajar con varias a la vez.

El alto rendimiento y elevada velocidad que alcanzan los modernos procesadores, como el que poseen los microcontroladores PIC, se debe a la conjunción de tres técnicas:

— Arquitectura Harvard
— Arquitectura RISC
— Segmentación

1.2.2. Memoria de programa

El microcontrolador está diseñado para que en su memoria de programa se almacenen todas las instrucciones del programa de control. No hay posibilidad de utilizar memorias externas de ampliación.

Como el programa a ejecutar siempre es el mismo, debe estar grabado de forma permanente. Los tipos de memoria adecuados para soportar esta función admiten cinco versiones diferentes:

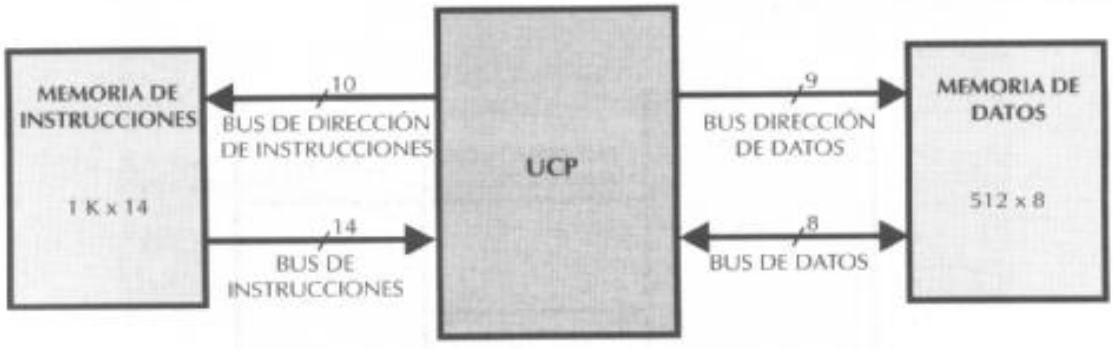


Figura 1.6. *En la arquitectura «Harvard» la memoria de instrucciones y la de datos son independientes, lo que permite optimizar sus características y propiciar el paralelismo. En la figura, la memoria de instrucciones tiene 1 K posiciones de 14 bits cada una, mientras que la de datos sólo dispone de 512 posiciones de un byte.*

1.ª ROM con máscara

En este tipo de memoria el programa se graba en el chip durante el proceso de su fabricación mediante el uso de «máscaras». Los altos costes de diseño e instrumental sólo aconsejan usar este tipo de memoria cuando se precisan series muy grandes.

2.ª EPROM

La grabación de esta memoria se realiza mediante un dispositivo físico gobernado desde un computador personal, que recibe el nombre de **grabador**. En la superficie de la cápsula del microcontrolador existe una ventana de cristal por la que se puede someter al chip de la memoria a rayos ultravioletas para producir su borrado y emplearla nuevamente. Es interesante la memoria EPROM en la fase de diseño y depuración de los programas, pero su coste unitario es elevado.

3.ª OTP (Programable una vez)

Este modelo de memoria sólo se puede grabar una vez por parte del usuario, utilizando el mismo procedimiento que con la memoria EPROM. Posteriormente no se puede borrar. Su bajo precio y la sencillez de la grabación aconsejan este tipo de memoria para prototipos finales y series de producción cortas.

4.ª EEPROM

La grabación es similar a las memorias OTP y EPROM, pero el borrado es mucho más sencillo al poderse efectuar de la misma forma que el grabado, o sea, eléctricamente. Sobre el mismo zócalo del grabador puede ser programada y borrada tantas veces como se quiera, lo cual la hace ideal en la enseñanza y en la creación de nuevos proyectos. El fabuloso PIC16C84 dispone de 1 K palabras de memoria EEPROM para contener instrucciones y también tiene algunos bytes de memoria de datos de este tipo para evitar que cuando se retira la alimentación se pierda información.

Aunque se garantizan 1.000.000 de ciclos de escritura/borrado en una EEPROM, todavía su tecnología de fabricación tiene obstáculos para alcanzar capacidades importantes y el tiempo de escritura de las mismas es relativamente grande y con elevado consumo de energía.

5.ª FLASH

Se trata de una memoria no volátil, de bajo consumo, que se puede escribir y borrar en circuito al igual que las EEPROM, pero suelen disponer de mayor capacidad que estas últimas. El borrado sólo es posible con bloques completos y no se puede realizar sobre posiciones concretas.

Son muy recomendables en aplicaciones en las que sea necesario modificar el programa a lo largo de la vida del producto, como consecuencia del desgaste o cambios de piezas, como sucede con los vehículos.

Por sus mejores prestaciones está sustituyendo a la memoria EEPROM para contener instrucciones. De esta forma **Microchip** comercializa dos microcontroladores prácticamente igua-

les, que sólo se diferencian en que la memoria de programa de uno de ellos es tipo EEPROM y la del otro tipo Flash. Se trata del PIC16C84 y el PIC16F84, respectivamente.

1.2.3. Memoria de datos

Los datos que manejan los programas varían continuamente, y esto exige que la memoria que les contiene debe ser de lectura y escritura, por lo que la memoria RAM estática (SRAM) es la más adecuada, aunque sea volátil.

Hay microcontroladores que disponen como memoria de datos una de lectura y escritura no volátil, del tipo EEPROM. De esta forma, un corte en el suministro de la alimentación no ocasiona la pérdida de la información, que está disponible al reiniciarse el programa. El PIC16C84, el PIC16F83 y el PIC16F84 disponen de 64 bytes de memoria EEPROM para contener datos.

> La memoria tipo EEPROM y la tipo Flash pueden escribirse y borrarse eléctricamente. Sin necesidad de sacar el circuito integrado del zócalo del grabador pueden ser escritas y borradas numerosas veces.

1.2.4. Líneas de E/S para los controladores de periféricos

A excepción de dos patitas destinadas a recibir la alimentación, otras dos para el cristal de cuarzo, que regula la frecuencia de trabajo, y una más para provocar el Reset, las restantes patitas de un microcontrolador sirven para soportar su comunicación con los periféricos externos que controla.

Las líneas de E/S que se adaptan con los periféricos manejan información en paralelo y se agrupan en conjuntos de ocho, que reciben el nombre de **Puertas**. Hay modelos con líneas que soportan la comunicación en serie; otros disponen de conjuntos de líneas que implementan puertas de comunicación para diversos protocolos, como el I^2C, el USB, etc.

1.2.5. Recursos auxiliares

Según las aplicaciones a las que orienta el fabricante cada modelo de microcontrolador, incorpora una diversidad de complementos que refuerzan la potencia y la flexibilidad del dispositivo. Entre los recursos más comunes se citan a los siguientes:

a) *Circuito de reloj*, encargado de generar los impulsos que sincronizan el funcionamiento de todo el sistema.
b) *Temporizadores*, orientados a controlar tiempos.
c) *Perro Guardián («watchdog»)*, destinado a provocar una reinicialización cuando el programa queda bloqueado.
d) *Conversores AD y DA*, para poder recibir y enviar señales analógicas.
e) *Comparadores analógicos*, para verificar el valor de una señal analógica.
f) *Sistema de protección ante fallos de la alimentación*.
g) *Estado de Reposo*, en el que el sistema queda «congelado» y el consumo de energía se reduce al mínimo.

1.3. PROGRAMACIÓN DE MICROCONTROLADORES

La utilización de los lenguajes más cercanos a la máquina (de bajo nivel) representan un considerable ahorro de código en la confección de los programas, lo que es muy importante dada la estricta limitación de la capacidad de la memoria de instrucciones. Los programas bien realizados en lenguaje Ensamblador optimizan el tamaño de la memoria que ocupan y su ejecución es muy rápida.

Los lenguajes de alto nivel más empleados con microcontroladores son el C y el BASIC, de los que existen varias empresas que comercializan versiones de compiladores e intérpretes para diversas familias de microcontroladores. En el caso de los PIC es muy competitivo e interesante el compilador de C **PCM** de la empresa **CCS** y el **PBASIC** de **microLab Engineering**, ambos comercializados en España por **Microsystems Engineering**.

Hay versiones de intérpretes de BASIC que permiten la ejecución del programa línea a línea y, en ocasiones, residen en la memoria del propio microcontrolador. Con ellos se puede escribir una parte del código, ejecutarlo y comprobar el resultado antes de proseguir.

El lenguaje que utilizan los profesionales para la programación de los microcontroladores es el Ensamblador, que es el más cercano a la máquina. También son frecuentes los programas en lenguaje C y en BASIC.

1.4. INSTRUMENTAL DE TRABAJO

Siempre que se diseña con circuitos integrados programables se precisan herramientas para la puesta a punto del hardware y del software.

Con referencia al software, además de los compiladores o intérpretes de los lenguajes usados, es muy interesante disponer de simuladores software, que consisten en programas que simulan la ejecución de instrucciones representando el comportamiento interno del procesador y el estado de las líneas de E/S. Como se simula por software al procesador, el comportamiento no es idéntico aunque proporciona una aproximación aceptable, especialmente cuando no es esencial el trabajo en tiempo real.

Microchip pone libremente a disposición de sus usuarios, a través de Internet (*http://www.microchip.com*), ensambladores como el **MPASM** y simuladores como el **MPSIM. Microsystems Engineering** comercializa un simulador muy completo para el PIC16X84 y varios compiladores e intérpretes de lenguaje C y BASIC (*http://www.arrakis.es/~msyseng*).

Respecto a las herramientas hardware, una indispensable es el grabador, encargado de escribir el programa en la memoria del microcontrolador. Existen grabadores muy completos capaces de trabajar con muchos modelos de diferentes familias, pero su elevado precio los aleja de los usuarios personales. Para estos últimos existen bastantes versiones de sencillos grabadores, específicos para ciertos modelos de microcontroladores, que gobernados desde un computador personal se ofrecen por un precio ligeramente superior al de un libro. (Véase Figura 1.7.)

Los fabricantes de los microcontroladores disponen de numerosas herramientas apropiadas para sus productos. (Véase Figura 1.8.)

En la puesta a punto de proyectos industriales una potentísima herramienta es el **emulador en circuito**, que consiste en un complejo dispositivo físico que, controlado por un programa

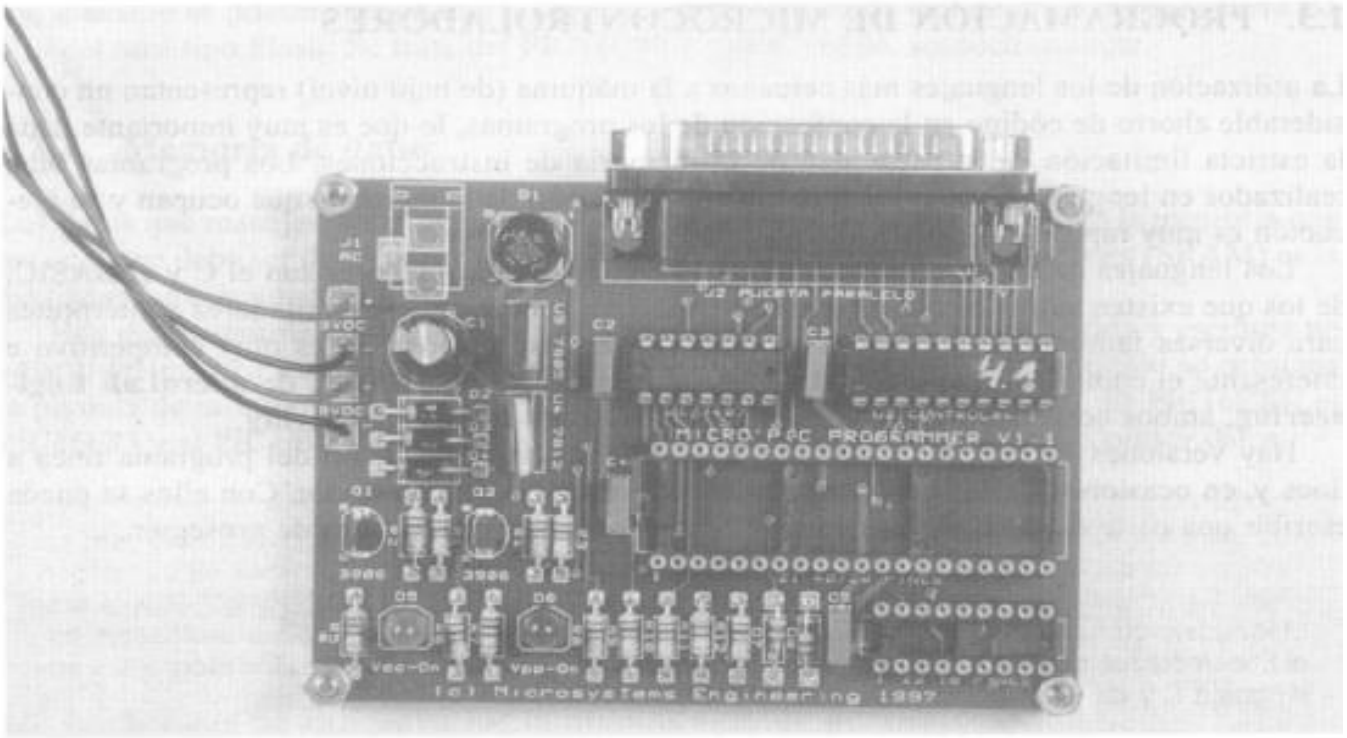


Figura 1.7. *Fotografía de un económico grabador de PIC de la gama media, el MICROPIC'PROGRAMMER, que fabrica* ***Microsystems Engineering***.



Figura 1.8. ***Microchip*** *dispone de una extensa gama de herramientas para realizar diseños con sus microcontroladores PIC.*

desde un computador personal, dispone de una «cabeza» con las mismas patitas que el microcontrolador que se trata de emular. La cabeza se introduce en el zócalo donde irá el microcontrolador con el programa definitivo en la placa de prototipo. El emulador en circuito hace funcionar al sistema como si hubiese un microcontrolador real, pero presentando en la pantalla del computador toda la información necesaria para conocer el desarrollo del programa y la actuación de los periféricos. Los resultados que así se obtienen son idénticos a los del producto final, puesto que la ejecución se realiza en tiempo real, a diferencia de los simuladores software. (Figura 1.9.)

Finalmente, reciben el nombre de **sistemas de desarrollo** equipos que mediante una combinación de hardware y de software consiguen realizar la mayoría de las fases que exige un diseño basado en microcontrolador. Como la mayoría de las herramientas, también ésta se gobierna por un programa desde un computador personal. Mantienen una buena comunicación y gestión de la edición de *programas* y su compilación o ensamblado. Tras la depuración del software y la obtención del código binario ejecutable disponen de un grabador que escribe el programa obtenido sobre la memoria de instrucciones. También puede verificar lo escrito en la memoria e incluso borrarlo, cuando es del tipo EEPROM o Flash

Una vez grabado el microcontrolador sus patitas quedan conectadas a través de *jumper* con los periféricos que hay que controlar y se procede a la ejecución en tiempo real del pro-



Figura 1.9. *Fotografía de un emulador en circuito para microcontroladores PIC. Se trata del modelo ICEPIC de la empresa* ***R.F. Solutions****.*

grama. La inclusión de los periféricos clarifica el resultado final. Mediante el programa de comunicación con el PC se puede conocer en todo momento el estado de todos los elementos del computador. Por su sencillez, potencia y economía destacamos el sistema de desarrollo **MICRO'PIC TRAINER** (Figura 1.10) de **Microsystems Engineering**, que incorpora las siguientes prestaciones:

— Grabador de PIC.
— Hardware y software de adaptación a PC.
— Programa de comunicaciones.
— Tarjeta que incluye la mayoría de los periféricos típicos: interruptores, leds, display de siete segmentos, potenciómetros para señales analógicas y pantalla LCD.
— Tutorial con fases de montaje y funcionamiento y ejercicios de experimentación y diseño de proyectos.

Para facilitar el montaje de los prototipos se comercializan diversas tarjetas de circuito impreso predefinidas. En ellas existe una zona común donde se aloja el zócalo del microcontrolador y su circuitería auxiliar (reloj, pulsador de Reset, regulación de voltaje, etc.). Hay otra zona libre para situar los periféricos propios de la aplicación, que pueden conectarse fácilmente con todas las líneas de E/S presentes en determinados puntos de la tarjeta. (Véase Figura 1.11.) Más adelante presentaremos la tarjeta para prototipos de **Microsystems Engineering** con la que se pueden desarrollar la mayoría de los proyectos y ejercicios que ofrecemos en este libro.

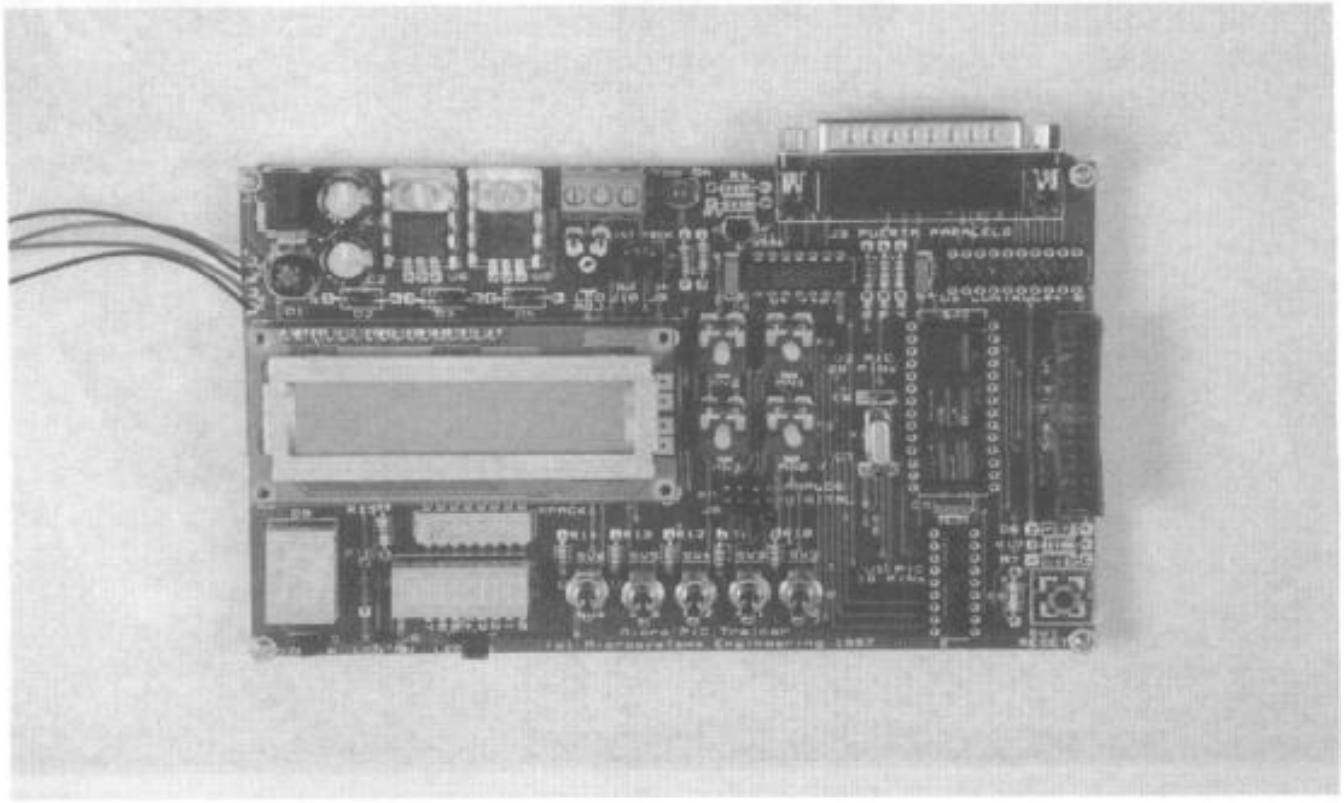


Figura 1.10. *Fotografía del sistema de desarrollo MICRO'PIC TRAINER.*

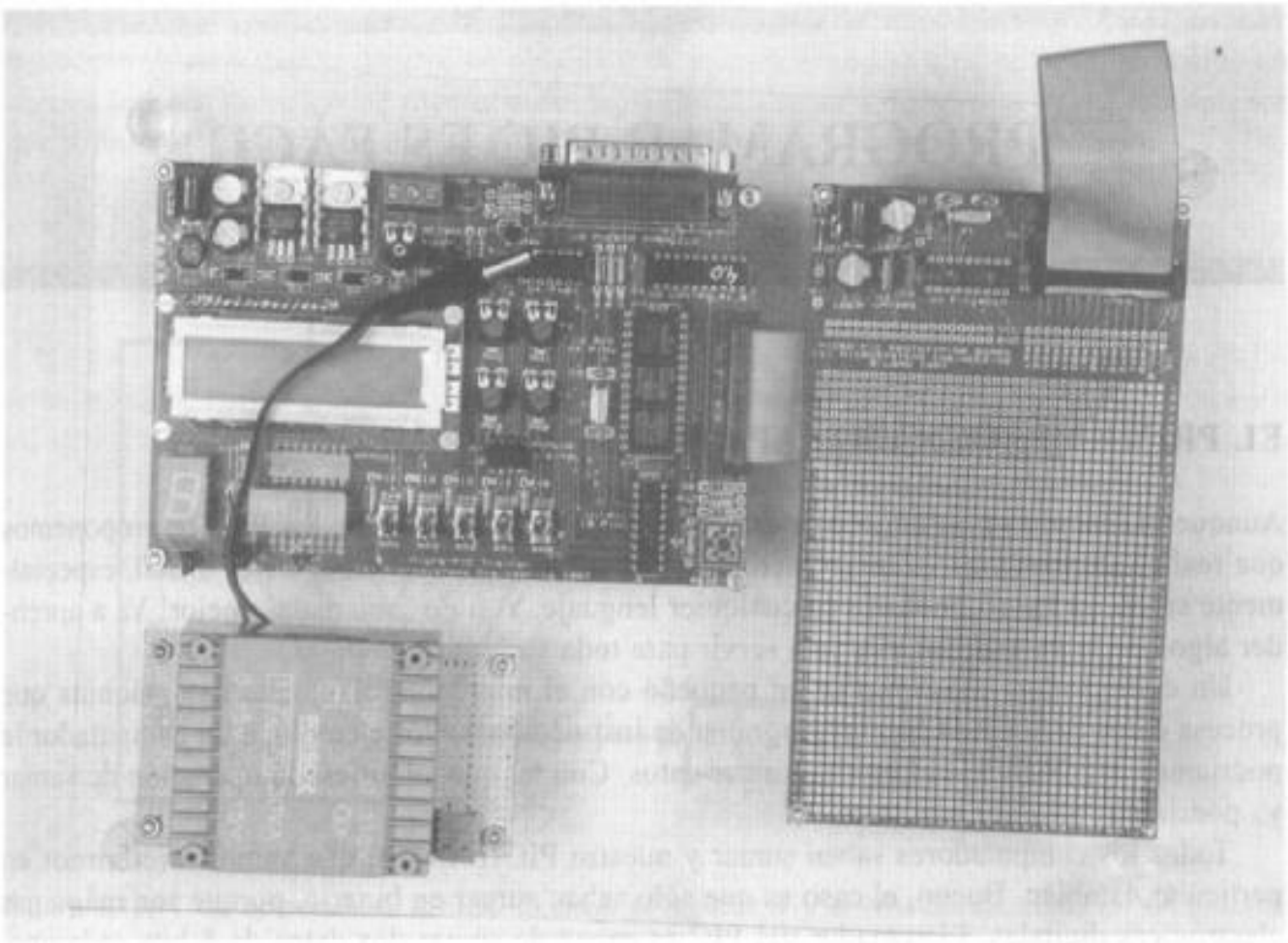


Figura 1.11. *Tarjeta para soportar prototipos basados en el PIC16X84 y conectarse al MICRO'PIC TRAINER.* (*Cortesía de* ***Microsystems Engineering***.)

PROGRAMAR PIC ES FÁCIL

EL PRIMER PROGRAMA: APRENDIENDO A SUMAR

Aunque el lector aún no conoce la estructura y la programación de los PIC, le proponemos que realice su primer programa. Queremos convencerle que esta labor no es difícil, especialmente si Vd. ya ha programado en cualquier lenguaje. Y si no sabe nada, ¡mejor! Va a aprender algo que le va a gustar y le va a servir para toda su vida.

Un computador, como sucede en pequeño con el microcontrolador, es una máquina que procesa datos de acuerdo con un programa de instrucciones. Por ejemplo, a un computador le podríamos encomendar la labor de sumar datos. Con tal que admitiese la operación de sumar ya podríamos resolver esta tarea.

Todos los computadores saben sumar y nuestro PIC16X84, al que vamos a referirnos en particular, también. Bueno, el caso es que sólo saben sumar en binario, porque son máquinas electrónicas digitales. El sumador del PIC es capaz de sumar dos datos de 8 bits cada uno. Dicho sumador tiene un registro, llamado **W** (de trabajo), en donde aloja uno de los sumandos. El otro sumando debemos decir lo que vale (literal) o en qué posición de la memoria de datos está. El resultado de la suma puede ir a ocupar W. Como los dos sumandos son de 8 bits cabe la posibilidad de que exista acarreo al sumar los dos bits de más peso y el resultado tenga 9 bits. Para resolver tal circunstancia existe un flip-flop FC, **señalizador de acarreo**, que se pone automáticamente a 1 cuando sucede este acontecimiento. En la Figura 1.12 se muestra un esquema simplificado del sumador del PIC16X84.

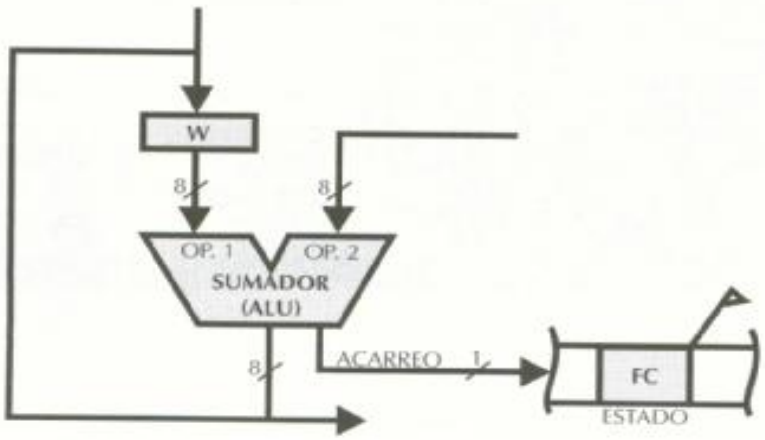


Figura 1.12. *Esquema simplificado del circuito sumador del PIC16X84.*

Además del operando que recibe la ALU desde W, precisa otro que puede ser un valor inmediato (literal) que se incluye en el código de la instrucción, o bien el valor contenido en alguno de los registros de la memoria de datos. En la Figura 1.13 se muestra el multiplexor que permite la selección de una de estas dos posibles fuentes del segundo operando.

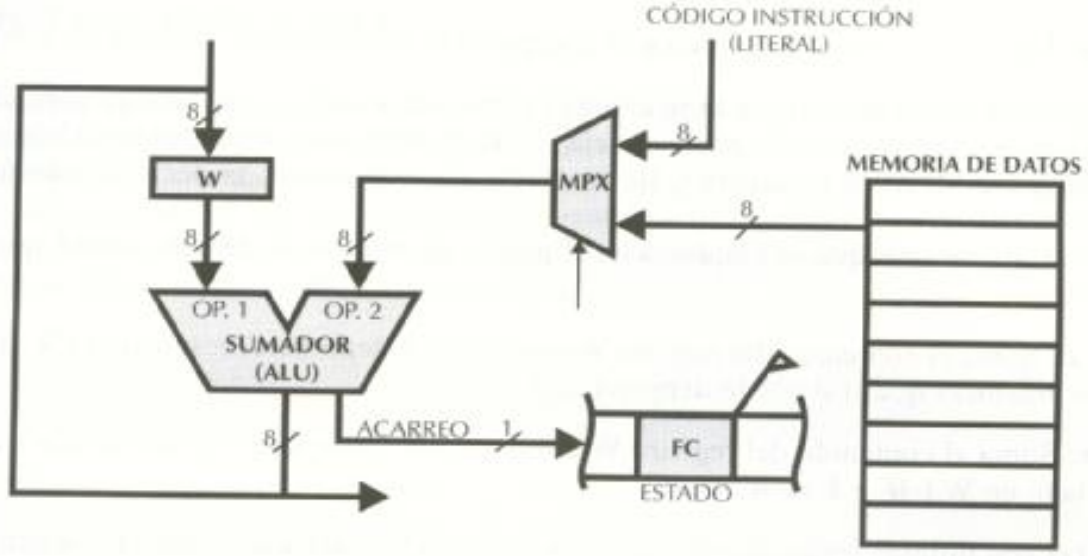


Figura 1.13. *Un operando de la ALU lo envía el registro W y el otro lo selecciona un multiplexor desde la propia instrucción (literal) o desde una de las posiciones de la memoria de datos.*

Antes de exponer el enunciado del primer programa se comenta el comportamiento de las dos instrucciones que vamos a usar:

Instrucción «mov»

Esta instrucción es típica en todos los procesadores y su misión es transferir el contenido de un registro «fuente» (f) a un registro «destino» (d). En los microcontroladores PIC todos los datos residen en posiciones de la memoria de datos y sólo el registro de trabajo W se salva de esta regla. Esto significa que *mov* puede mover tres tipos de operandos fundamentales:

1. El contenido del registro W.
2. El contenido de una posición de la memoria de datos o registro.
3. Un literal o valor inmediato que se incluye en el código máquina de la propia instrucción.

Para gestionar los posibles operandos existen tres variantes de *mov:*

***movf f,d*:** Mueve el contenido del operando fuente, que es una posición de la memoria de datos, al destino, que bien puede ser el registro W cuando d = 0, o el propio fuente cuando d = 1.

Mover el contenido de un registro fuente al registro W es una operación habitual y sencilla. No está tan claro mover el contenido del fuente al fuente, es decir, dejarlo como estaba. En este último caso hay que tener en cuenta que en esta instrucción se activa el señalizador de cero, FZ, que es un flip-flop que se pone automáticamente a 1 cuando lo que se transfiere es cero.

De esta forma la instrucción *movf f,0* realiza la transferencia $f \Rightarrow W$, pasando FZ = 1 si el contenido de $f = 0$. Sin embargo, cuando la instrucción responde al nemónico *movf f,1*, el contenido de f se deja en el mismo sitio y no cambia, pero si FZ = 1 averiguamos que dicho valor es cero.

movwf f: Mueve el contenido del registro W a la posición de la memoria de datos identificada por f. Realiza la transferencia $W \Rightarrow f$.

movlw k: Mueve el literal k incluido en el código de la instrucción al registro W, $k \Rightarrow W$.

También es frecuente usar en la práctica el nemónico *movfw f*, que es muy cómodo por la claridad con la que representa la transferencia $f \Rightarrow W$. Aunque este nemónico no existe como tal, el ensamblador MPASM lo acepta y lo interpreta correctamente, es decir, lo transforma en *movf f,0*.

La otra instrucción que se empleará en el primer ejemplo es la de sumar: ***add***, que admite dos variantes:

addwf f,d: Suma el contenido del registro W con el de f y deposita el resultado en W si el valor de d = 0, mientras que si d = 1 lo deposita en f.

addlw k: Suma al contenido del registro W el literal que acompaña a la instrucción y deposita el resultado en W ($W + k \Rightarrow W$).

En las dos últimas instrucciones de suma funcionan automáticamente tres señalizadores. El FZ, de cero, que pasa a valer 1 si el resultado de la operación es 0. El de acarreo, FC, que se pone a 1 cuando hay acarreo en la suma de los dos bits de más peso de los operandos, y finalmente el señalizador de acarreo intermedio, FDC, que adquiere el valor 1 cuando hay acarreo al sumar los dos bits que ocupan el lugar 4.º (bit 3) de los operandos. Este último señalizador es muy interesante si se trabaja con dígitos BCD.

ENUNCIADO

El ejercicio maneja tres posiciones de la memoria de datos. En la dirección 0Ch se depositará el primer operando; en la 0Dh el segundo y en la 0Eh el resultado de la suma de los dos primeros operandos. Como valor para el primer operando se va a usar 5 y como segundo operando el 2.

En el lenguaje Ensamblador MPASM para expresar los valores hexadecimales se comienza anteponiéndoles «0x». Así, las tres posiciones de la memoria de datos que utiliza este programa, expresadas en hexadecimal para el Ensamblador MPASM, se definirán como 0x0c, 0x0d y 0x0e, respectivamente. Esta última notación será la que se utilice a partir de ahora en este libro. Las diferentes formas de expresar números en el MPASM dependiendo de su base son:

BASE	REPRESENTACIÓN
DECIMAL	d'12'
HEXADECIMAL	0x0c / h'0c' / 0c / 0ch
BINARIO	b'1010'

Hay varias formas de representar los números en hexadecimal, aunque la más empleada, por ser además la representación utilizada en «C», es la de anteponer 0x al número hexadecimal. No nos debemos preocupar si se nos olvida este detalle, porque el ensamblador MPASM asume por defecto que se empleará la base hexadecimal.

EDICIÓN DEL PROGRAMA

Para editar el programa se puede usar cualquier editor que use caracteres ASCII. El más a mano es el propio del sistema operativo MS-DOS. En tal caso y si le damos como nombre a este programa PRIMERO, estando situados con el *prompt* del DOS se teclea: *EDIT PRIMERO.ASM*. El texto que existe detrás de las directivas, comandos e instrucciones precedido de «;» sólo tiene la misión de aclarar su cometido, pero es ignorado por el Ensamblador cuando traduce el programa en código máquina. Son los comentarios.

PRIMERO.ASM

```
LIST P = 16C84          ; Indica el modelo de PIC que se usa
                        ; Es una directiva del Ensamblador

                        ; Zona para etiquetas
OPERANDO1 EQU 0x0c      ; Define la posición del operando 1
OPERANDO2 EQU 0x0d      ; Define la posición del operando 2
RESULTADO EQU 0x0e      ; Define la posición del resultado

ORG0                    ; Comando que indica al Ensamblador
                        ; la dirección de la memoria de
                        ; programa donde situará la
                        ; siguiente instrucción

movlw 05                ; 5 ⇒ W (primera instrucción)
movwf OPERANDO1         ; W ⇒ operando1
movlw 02                ; 2 ⇒ W
movwf OPERANDO2         ; W ⇒ operando2
movfw OPERANDO1         ; operando1 ⇒ W
addwf OPERANDO2,0       ; W + operando2 ⇒ W
movwf RESULTADO         ; W ⇒ resultado

END                     ; Directiva de fin del programa
```

El uso de mayúsculas y minúsculas en este fragmento de código obedece a una serie de reglas o normas de estilo, comunes entre los programadores en ensamblador, que aunque no son obligatorias, facilitan la lectura del código fuente. Un resumen de las reglas empleadas es el siguiente:

- Directivas del compilador en mayúsculas
- Nombres de variables en mayúsculas
- Nemónicos en minúsculas
- Programa bien tabulado

Una vez editado el programa hay que ensamblarlo. Para ello nos situamos en el directorio del Ensamblador MPASM y se teclea MPASM. Se rellenan los datos que pide el programa y pulsando la tecla F10 se realiza el ensamblado. Obtenido el programa en código binario ejecutable se puede simular el funcionamiento del mismo utilizando un simulador software. Recomendamos el MPSIM de **Microchip** y el SIMUPIC'84 de **Microsystems Engineering**, que presentamos a continuación para que pueda editar y ejecutar programas desde el principio.

Si Vd. ha sido capaz de entender el programa y la misión de las instrucciones, de momento ya basta, pero si tiene ganas de comprobar si sus pensamientos son acertados le proponemos una cuestión.

PROPUESTA DE TRABAJO

En el programa PRIMERO.A$M sobra alguna instrucción. Si Vd. consigue realizarlo con menos instrucciones, no sólo ocupará menos palabras de la memoria, sino también se ejecutará más rápidamente.

¿Quiere intentar resolver el mismo programa pero con menos instrucciones? Si se da por vencido, o quiere conocer otro enfoque, dispone de una solución en el disquete que acompaña al libro bajo el nombre PRIMERO2.ASM.

MANEJANDO EL SIMULADOR SIMUPIC'84: EL PRIMER CONTACTO

El simulador SIMUPIC'84 es una herramienta que trata de facilitar la labor del programador de PIC. Acaba de comprobar que para realizar un programa, por muy sencillo que sea, deben utilizarse varios programas; un editor, un ensamblador y un simulador. El SIMUPIC'84 trata de integrar todas esas herramientas en un único entorno. Como habrá comprobado, este libro viene acompañado por un CD en el que se incluye una copia del programa. Pues manos a la obra, ¡vamos a probarlo!

Lo primero que se debe hacer es instalar el programa; básicamente esto se realiza siguiendo las instrucciones que sobre este apartado encontrará en el Apéndice E, en el que también se describe el contenido del CD.

Ahora hay que decirle al SIMUPIC qué editor y qué ensamblador desea utilizar. Lo más sencillo es utilizar el editor del DOS (Edit) y el ensamblador MPASM, tal y como se ha utilizado en el ejercicio anterior. Para eso se edita un fichero llamado *simupic.cfg* y se alteran los valores necesarios, aunque es muy posible que el programa funcione con la configuración por defecto.



Figura 1.14. *Configuración por defecto del SIMUPIC'84. Fichero simupic.cfg.*

Ante cualquier poblema con el programa, consulte el apéndice final del libro; en él se explica detalladamente la instalación y el funcionamiento de esta herramienta. Además, el CD contiene un fantástico tutorial multimedia para manejar rápidamente el SIMUPIC'84.

¿REPETIMOS LA SUMA?

Vamos a volver a realizar el programa anterior, pero esta vez con el SIMUPIC'84. Para eso ejecutamos el programa del simulador escribiendo desde el prompt «SIM84». El menú principal contiene todos los pasos a seguir a la hora de programar una aplicación para PIC. En primer lugar hay que asignar un nombre al programa; elegiremos la primera opción y escribiremos PRIMERO, que es el nombre que hemos dado a la aplicación. Ahora elegiremos la segunda opción, con lo que aparece la pantalla del editor y podremos escribir el código fuente.

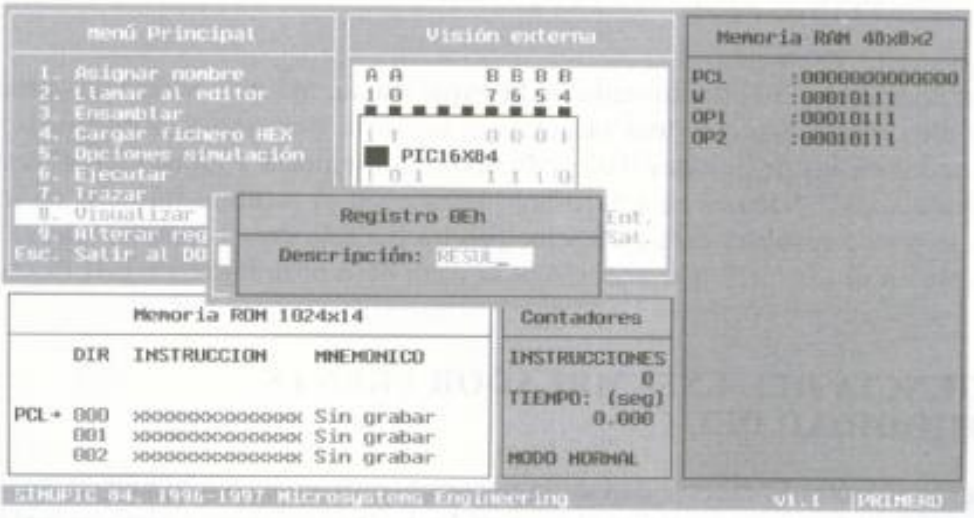


Figura 1.15. *Visualización y renombrado de registros en el SIMUPIC.*

Procedemos a ensamblarlo (opción tercera) y el SIMUPIC se encargará de llamar automáticamente al MPASM y ensamblar nuestro programa; además nos presentará los errores de ensamblado, si los hubiera.

Una vez ensamblado el código fuente del programa PRIMERO.ASM, se puede simular y comprobar que hace lo que debe. Para eso debemos cargarlo en el simulador con la cuarta opción y ejecutarlo..., pero antes debemos tener medios para comprobar que efectivamente el programa suma los contenidos de las posiciones 0x0c y 0x0d y deposita el resultado en la 0x0e. Para eso está la octava opción (visualizar registros). Seleccionaremos todos los registros que participan en la suma (W, 0x0c, 0x0d y 0x0e) y los visualizaremos; incluso es posible renombrarlos para poder seguir mejor el desarrollo de la suma.

Se pasa a simular el programa. Lo mejor será ejecutarlo paso a paso para ver qué es lo que hace cada instrucción; la opción «trazar» realiza este trabajo, y cada vez que se pulse se ejecutará una instrucción. Cuando todo el programa haya sido ejecutado se nos informa que ha finalizado, y si todo ha ido bien, aparecerá una pantalla como ésta:

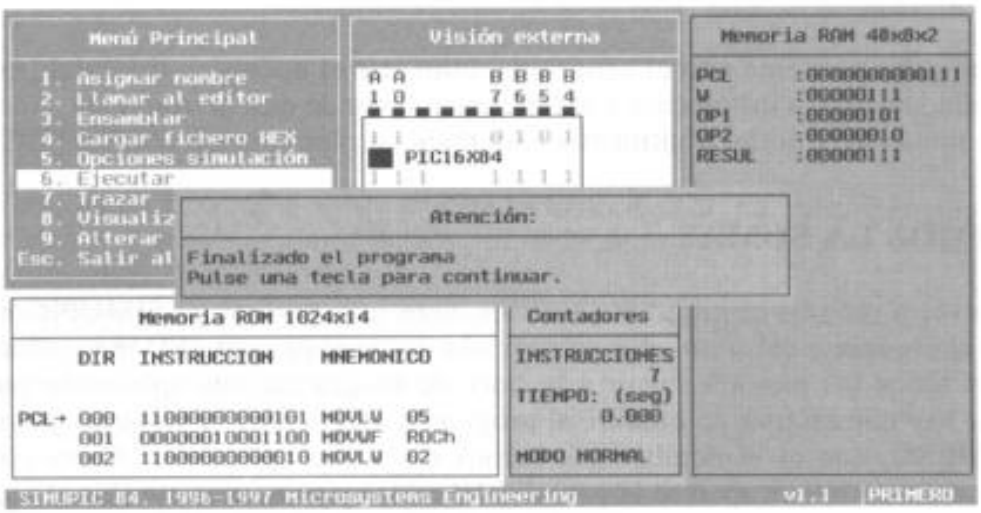


Figura 1.16. *Resultado del programa PRIMERO.ASM.*

Pero la madre de todo conocimiento es la experiencia, así que piense sus propios programas y simúlelos. De momento, una interesante variante del ejercicio podría ser sumar dos números situados en las posiciones 0x0c y 0x0d de la memoria y almacenar el resultado en la misma posición 0x0d. Además es interesante pensar en el mínimo número de instrucciones con las que se puede resolver este ejercicio; ¿podrá hacerlo con 5 instrucciones? La solución está disponible en el disquete que acompaña la obra bajo el nombre PRIMERO3.ASM.

LA EFICIENCIA DEL ENSAMBLADOR FRENTE A LA COMODIDAD DEL C

Para demostrar al lector la efectividad de la programación en ensamblador hemos realizado el mismo programa en lenguaje C: suma el contenido de dos variables y almacena el resultado en una tercera variable, llamada resultado. El programa es el siguiente.

```
/* PRIMERO.C: Programa que realiza la suma de dos */
/* operandos situados en las posiciones 0x0c y 0x0d y */
/* deposita el resultado en la posición 0x0e. */

#DEVICE PIC16C84    /* se indica el microcontrolador
                       a usar. */
void main( void )
{
int* p_op1, p_op2, p_result;  /* Declaración de los punteros
                                 a los datos que se van a
                                 utilizar. */
p_op1 = 0x0c;                 /* Se indican las direcciones de
                                 los operandos inicializando los
                                 punteros: Operando1 en 0x0c. */

p_op2 = 0x0d;                 /* Operando2 en 0x0d. */
p_result = 0x0e;              /* Resultado en 0x0e. */

*p_op1 = 5;                   /* Se inicializan los registros
                                 con sus valores */
*p_op2 = 2;                   /* correspondientes. */

*p_result = *p_op1 + *p_op2;  /* Se efectúa la suma. */
}
```

El programa ha sido compilado con el PCM de la casa **Custom Computer Services** que comercializa **Microsystems Engineering** y se ha empleado el SIMUPIC'84 para desensamblar su código y traducirlo al ensamblador. El resultado ha sido el siguiente:

```
00  movlw   00
01  movwf   0x0a
02  goto    05
03  nop
04  nop
05  clrf    0x04
06  clrf    0x03
07  movlw   0x0c
08  movwf   0x0e
09  movlw   0x0D
0A  movwf   0x0f
0B  movlw   0x0e
0C  movwf   0x10
0D  movf    0x0e,0
0E  movwf   0x04
0F  movlw   0x05
10  movwf   0x00
11  movf    0x0f,0
12  movwf   0x04
13  movlw   02
14  movwf   0x00
```

```
15  movf    0x00,0
16  movwf   0x11
17  movf    0x0e,0
18  movwf   0x04
19  movf    0x00,0
1A  movwf   0x12
1B  movf    0x0f,0
1C  movwf   0x04
1D  movf    0x00,0
1E  addwf   0x12,0
1F  movwf   0x12
20  movf    0x11,0
21  movwf   0x04
22  movf    0x12,0
23  movwf   0x00
24  sleep
25
```

¡El programa resultante está compuesto por 37 instrucciones! Más de cinco veces las empleadas para realizar el mismo trabajo en ensamblador (no se preocupe si no entiende algunos de los nemónicos, ya irá comprendiendo su significado según avance en la lectura). La flexibilidad del «C» tiene su precio en cuanto a tamaño del código y tiempo de ejecución.

DISEÑAR APLICACIONES CON PIC ES FÁCIL

PRESENTACIÓN DE LOS PERIFÉRICOS MÁS SIMPLES

Periféricos digitales de entrada

Los diseños reales utilizan diversos periféricos que hay que conectar a las patitas del microcontrolador que soportan las líneas de E/S. Los periféricos pueden trabajar con señales digitales o analógicas y pueden ser sencillos y fáciles de acoplar o complejos en su funcionamiento y su adaptación. Nuestra idea es acostumbrar al lector a que realice aplicaciones sencillas y entusiasmarle con las complejas para que él mismo se decida a realizarlas.

Personalmente propongo a mis alumnos realizar proyectos difíciles con la seguridad de que no van a funcionar y que seguramente estropearán elementos del mismo. Lo que van a aprender es algo que yo nunca les podré enseñar y, generalmente, será algo irrepetible, porque nunca volverán a caer en los mismos errores. Ellos se ríen socarronamente, pero en gran parte muerden el anzuelo y se dan cuenta de lo que les he dicho. Además, no suelo valorar mucho las experiencias que salen bien a la primera. Siempre me quejo que no han servido para nada; con ellas no han aprendido nada nuevo.

A Vd., sufrido lector, le digo lo mismo: MONTE PROYECTOS, HAGA EXPERIENCIAS, y si no funcionan, ¡¡ENHORABUENA!!, está en el buen camino y cada cosa que falle es una enseñanza valiosísima: ¡ YA SABE CÓMO NO TIENE QUE VOLVER A HACERLO! Cuando alguno se lamenta del dinero que cuesta esta filosofía de trabajo, que medite si hay alguna otra cosa que le proporcione más satisfacción y si la encuentra, QUE CAMBIE DE PROFE-

SIÓN. Los pocos días que pasamos en este mundo debemos intentar que sean felices y si nuestro trabajo y nuestro pasatiempo no nos da satisfacciones estamos dilapidando el escaso tiempo que disponemos.

Vd. como nosotros puede pasar grandes momentos montando y probando sus experimentos y plasmando en materia sus ideas. Para ello necesita un microcontrolador, una idea y unos periféricos que la lleven a buen puerto.

Cuanto más sencilla sea la idea y más simples los periféricos mayor potencial tendrá. Un simple diodo led que se encienda y un zumbador que suene cuando una persona mayor, enferma o impedida, tiene que tomar una medicina, seguramente será una idea que, bien ejecutada, tendría la posibilidad de forrar en oro al que la tenga y sepa desarrollarla.

Como primer paso vamos a describir los periféricos de entrada y salida más fáciles y usados en los proyectos, los digitales. Si son de entrada «introducen» un nivel lógico alto o bajo por la línea que se les conecta. Si son de salida «reciben» un nivel alto o bajo para activarse o desactivarse. Comenzaremos con los periféricos digitales de entrada.

PULSADORES

Estos dispositivos permiten introducir un nivel lógico en el momento que se les acciona, pasando al nivel contrario cuando se deja de hacerlo y vuelven a la posición de reposo. (Figura 1.17.)

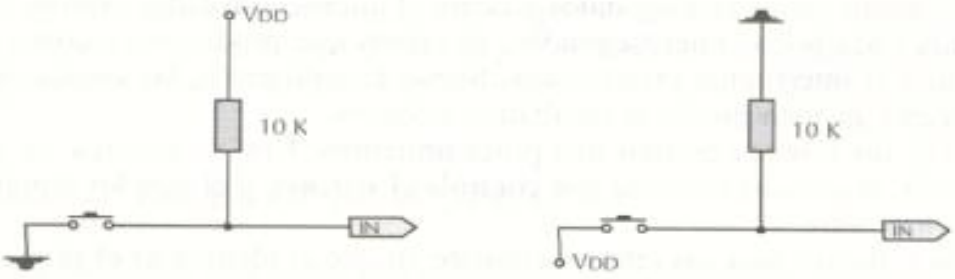


Figura 1.17. *Las dos posibles formas de conectar un pulsador.*

En el esquema izquierdo de la Figura 1.17 la línea de entrada (IN) recibe un nivel lógico alto cuando el pulsador está en reposo. Cuando se acciona se introduce un nivel lógico bajo. El pulsador de la derecha funciona al revés.

Hay multitud de detectores, finales de carrera y sensores digitales que funcionan de la misma manera que los pulsadores.

INTERRUPTORES

Los interruptores tienen dos estados estables y hay que accionarlos para cambiar de uno a otro. El interruptor admite el estado **abierto** y el estado **cerrado**. La Figura 1.18 muestra las dos formas de conectar los interruptores.

En el esquema de la izquierda de la Figura 1.18, por la línea de entrada al microcontrola-

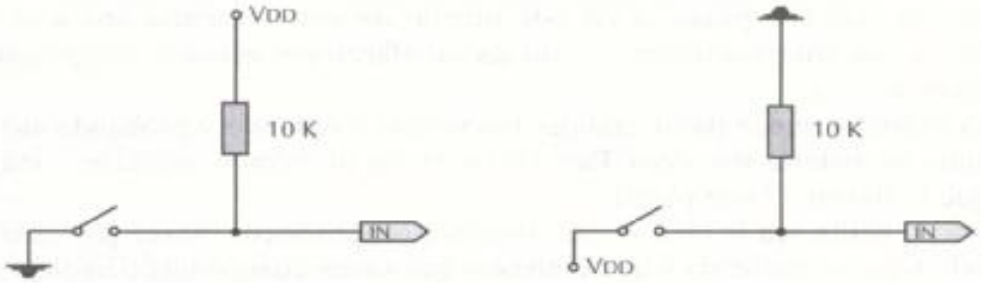


Figura 1.18. *Esquema de conexionado de los interruptores.*

dor (IN) se introduce un nivel lógico alto cuando el interruptor está abierto, mientras que cuando se cierra aplica un nivel lógico bajo. El interruptor de la derecha funciona al revés.

CIRCUITO ANTIRREBOTES

Todos los dispositivos electromecánicos originan un fenómeno denominado «rebotes». Los rebotes se producen en el instante en que se abren o se cierran las láminas metálicas del interruptor o pulsador. Hay un momento de inestabilidad en el que, aunque sólo se haya accionado el dispositivo una vez, las láminas se abren y se cierran múltiples veces en el momento de la transición. En la Figura 1.19 se presenta el efecto de los rebotes.

Los rebotes duran varios milisegundos y como el microcontrolador explora el estado de la línea de entrada cada pocos microsegundos, el efecto que producen es como si se hubiese abierto y cerrado el interruptor varias veces. Según la aplicación, las consecuencias de esta falsa interpretación pueden provocar resultados erróneos.

Para resolver los rebotes existen dos procedimientos. Uno se encarga de eliminarlos el software, es decir, el propio programa que controla el sistema, y el otro los elimina con circuitería externa o hardware.

La solución software para los rebotes consiste en que al identificar el primer flanco de la señal de entrada, el flujo de control pasa a una rutina de retardo de varios milisegundos antes de pasar a detectar si se ha producido el flanco contrario.

Los dos circuitos hardware antirrebotes se muestran en la Figura 1.20. En el de la izquier-

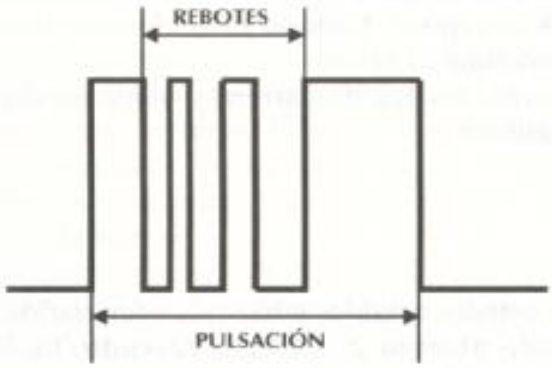


Figura 1.19. *Efecto de los rebotes.*

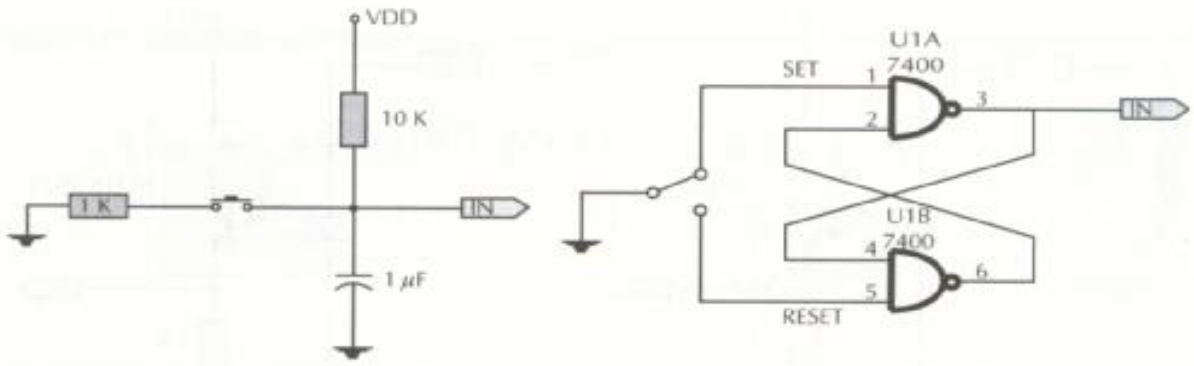


Figura 1.20. *Dos esquemas para eliminar los rebotes.*

da, cuando el pulsador está abierto el condensador está cargado y se introduce un nivel alto por la línea de entrada IN. Al activar el pulsador el condensador se descarga a través de la resistencia de 1 K y aplica un nivel lógico bajo. Al desactivar el pulsador, el nivel alto no se alcanza hasta que se cargue el condensador con la tensión positiva de alimentación a través de la resistencia de 10 K. Se puede regular el tiempo de carga modificando los valores de la resistencia y el condensador.

El circuito de la derecha de la Figura 1.20 muestra un flip-flop R-S asíncrono configurado mediante dos puertas NAND. Cuando el conmutador cierra el contacto de arriba el flip-flop se activa (Set) y, aunque haya rebotes, se mantiene activado, dado su funcionamiento. Cuando el conmutador pasa a cerrar el contacto inferior, se genera un Reset y el flip-flop se desactiva aunque existan rebotes. Una transición completa del conmutador de abajo hacia arriba y de nuevo abajo genera un impulso limpio y sin rebotes que se aplica a la línea IN.

ENTRADAS DIGITALES OPTOACOPLADAS

En bastantes ocasiones hay que aislar eléctricamente al periférico de las líneas de entrada del microcontrolador. Para este fin la señal se aplica a la línea de entrada mediante un rayo de luz. El periférico controla un emisor de luz y cuando se activa el primero, se genera un rayo luminoso. La línea de entrada IN recibe la señal de un receptor de luz, que cuando recibe un haz luminoso conduce una corriente. No existe ninguna relación eléctrica entre el periférico y la línea de entrada, sólo hay una transferencia luminosa.

En algunos dispositivos optoacopladores, tanto el emisor de luz como el receptor están encapsulados conjunta y herméticamente en un chip, siendo accesibles sólo las patitas de entrada y salida. En otros, el receptor y el emisor son independientes y pueden emplearse para detectar el paso de objetos a su través. En la Figura 1.21 se ofrecen dos esquemas de optoacopladores.

En el esquema de la izquierda de la Figura 1.21, el periférico, cuando se activa, aplica un nivel lógico alto al ánodo del diodo emisor de luz, generando el rayo luminoso. El valor de R depende de las características del diodo emisor. El receptor es un fototransistor que conduce cuando incide sobre él un rayo luminoso, produciendo un nivel lógico alto en la resistencia de 1 K que se aplica a la línea IN. Así, cuando se activa el periférico, el diodo led emite luz, conduce el fototransistor y se produce un nivel alto en la línea IN. En el esquema de la dere-

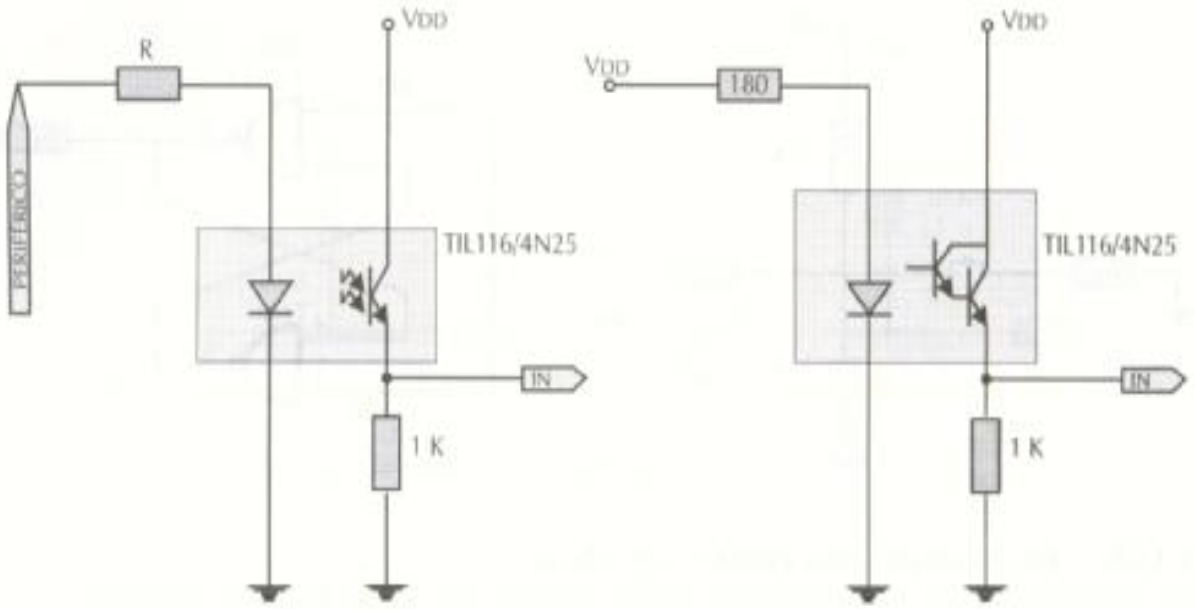


Figura 1.21. *Modelos comerciales de optoacopladores y esquemas de conexionado con las patitas de entrada del microcontrolador.*

cha, el emisor está permanentemente generando un rayo de luz, por lo que siempre hay nivel alto en la línea de entrada, al estar conduciendo siempre el fototransistor. Si algún objeto se interpone entre el emisor y el receptor se produce un nivel bajo en la línea de entrada. Funciona como un detector de paso.

MICROCONTROLADORES DE 8 BITS: LOS «PIC» 2

2.1. LA ELECCIÓN

Entre más de 50 fabricantes de microcontroladores que existen en el mundo es muy difícil seleccionar «el mejor». En realidad no existe, porque en cada aplicación son sus características específicas las que determinan el más conveniente.

Por otra parte, los autores nos hemos equivocado a menudo y no somos una garantía de acierto. Personalmente, recuerdo la época, a principios de la década de los ochenta, cuando estaba convencido que las *Memorias de Burbujas Magnéticas* revolucionarían el mundo de la Microinformática con la misma importancia que lo hicieron antes los microprocesadores. En el día de hoy cada vez hay menos profesionales que saben de su existencia.

De todas formas, como el libro lo hacemos entre dos y mi coautor está entusiasmado con los **PIC** y vive de ellos, igual esta vez tenemos más suerte.

Dejando a un lado las preferencias personales, lo cierto es que los **PIC** tienen «ángel» y una gran aceptación en la comunidad de técnicos y aficionados que trabajan con microcontroladores.

¿Cuánto durará este éxito? Hasta que salga al mercado otro producto que le robe esa imagen.

¿Cuál es la razón de la excelente acogida? Porque hay detalles que nos vuelven locos a los profesionales de la Microelectrónica y Microinformática; citamos los más importantes a nuestro juicio.

- Sencillez de manejo
- Buena información
- Precio
- Buen promedio de parámetros: velocidad, consumo, tamaño, alimentación, código compacto, etc.
- Herramientas de desarrollo fáciles y baratas
- Diseño rápido

Los **PIC** deslumbran a quienes los usamos porque:

- Su coste es comparativamente inferior al de los competidores.
- Poseen una elevada velocidad de funcionamiento.
- Tienen un juego reducido de instrucciones; 35 en la gama media.
- Los programas son compactos.
- Bajo consumo unido a un amplio rango de voltajes de alimentación.
- Muchas herramientas software se pueden recoger libremente a través de Internet desde **Microchip** (http://www.microchip.com).
- Existe una gran variedad de herramientas hardware que permiten grabar, depurar, borrar y comprobar el comportamiento de los PIC.
- La gran variedad de modelos de PIC permite elegir el que mejor responde a los requerimientos de la aplicación.

Lo cierto es que a las puertas del siglo XXI y en su corta vida, los PIC ocupan las posiciones de cabeza en el ranking mundial, compitiendo codo a codo con gigantes como **Intel** y **Motorola**. En 1990 ocupaba el puesto vigésimo y actualmente son más de 100 millones de PIC los que vende **Microchip** cada año.

Dentro de los diferentes tipos de microcontroladores los hay que procesan datos de 4, 8, 16 y 32 bits, sin embargo, el más representativo y popular es el de 8 bits, al que pertenecen los PIC, porque resulta el más flexible y eficaz para la mayoría de los diseños típicos. (Figura 2.1.)

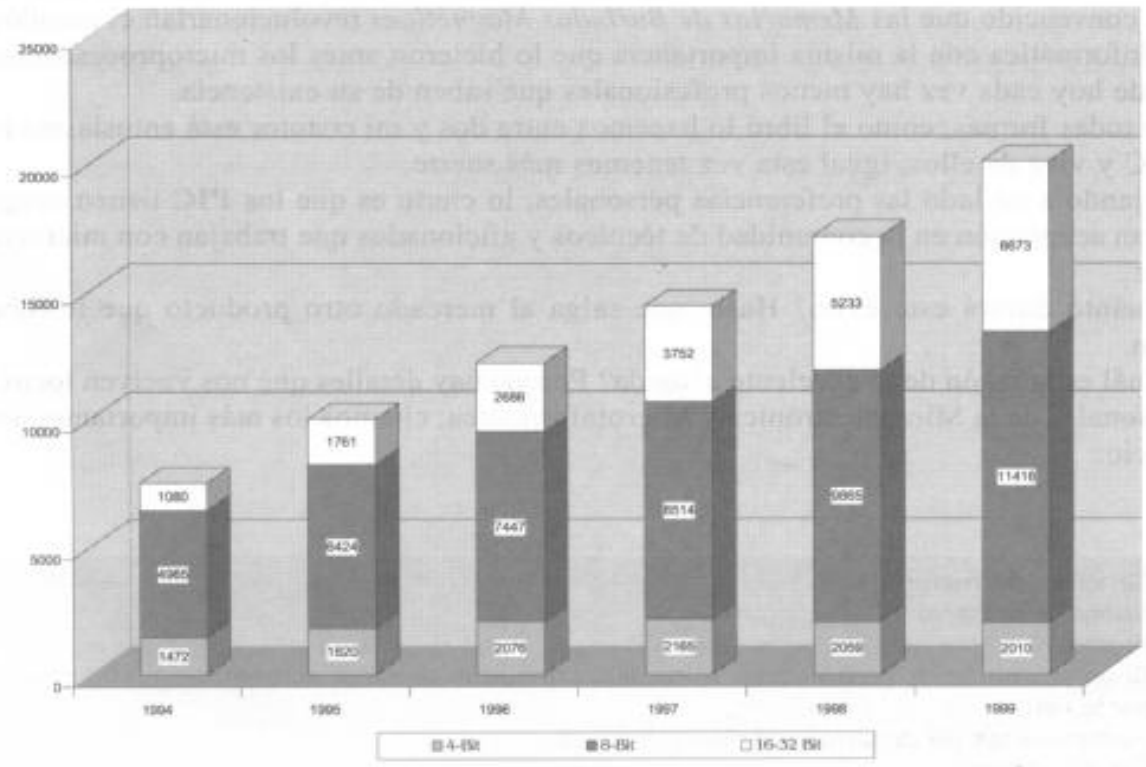


Figura 2.1. *Distribución, por segmentos, del mercado mundial de microcontroladores. Las ventas de los de 8 bits superan al resto ampliamente.*

2.2. LA FAMILIA DE LOS PIC

Una de las labores más importantes del ingeniero de diseño es la elección del modelo de microcontrolador que mejor satisfaga las necesidades del proyecto con el mínimo presupuesto.

En 1997 el fabricante de los PIC dispone de 52 versiones diferentes y cada año aumenta considerablemente su lista.

Microchip dispone de cuatro familias de microcontroladores de 8 bits para adaptarse a las necesidades de la mayoría de los clientes potenciales.

2.2.1. Gama enana: PIC12C(F)XXX de 8 patitas

Se trata de un grupo de PIC de reciente aparición que ha acaparado la atención del mercado. Su principal característica es su reducido tamaño, al disponer todos sus componentes de 8 patitas. Se alimentan con un voltaje de corriente continua comprendido entre 2,5 V y 5,5 V, y consumen menos de 2 mA cuando trabajan a 5 V y 4 MHz. El formato de sus instrucciones puede ser de 12 o de 14 bits y su repertorio es de 33 o 35 instrucciones, respectivamente. En la Figura 2.2 se muestra el diagrama de conexionado de uno de estos PIC.

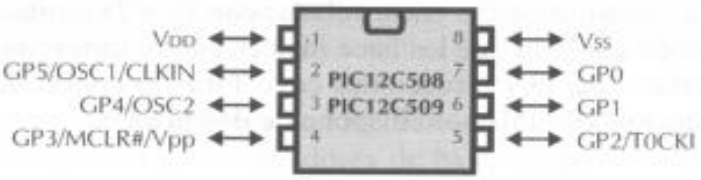


Figura 2.2. *Diagrama de conexiones de los PIC12Cxxx de la gama enana.*

Aunque los PIC enanos sólo tienen 8 patitas, pueden destinar hasta 6 como líneas de E/S para los periféricos porque disponen de un oscilador interno R-C.

En la Figura 2.3 se presentan las principales características de los modelos de esta subfamilia, que el fabricante tiene la intención de potenciar en un futuro próximo. Los modelos 12C5xx pertenecen a la gama baja, siendo el tamaño de las instrucciones de 12 bits; mientras que los 12C6xx son de la gama media y sus instrucciones tienen 14 bits. Los modelos 12F6xx poseen memoria Flash para el programa y EEPROM para los datos.

EJEMPLO

Según el esquema de la Figura 2.2, para funcionar un PIC12C508 precisa una tensión de alimentación comprendida entre 2,5 y 5,5 V, que se aplica a las patitas V_{DD} y V_{SS}. ¿Cuántas patitas quedan disponibles, como máximo, para trabajar como líneas de E/S?

SOLUCIÓN

Sólo 6, al disponer la cápsula de 8 patitas.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS	FRECUENCIA MÁXIMA	LÍNEAS E/S	ADC 8 BITS	TEMPORIZADORES	PATITAS
PIC12C508	512x12	25x8	4 MHz	6		TMR0 + WDT	8
PIC12C509	1024x12	41x8	4 MHz	6		TMR0 + WDT	8
PIC12C670	512x14	80x8	4 MHz	6		TMR0 + WDT	8
PIC12C671	1024x14	128x8	4 MHz	6	2	TMR0 + WDT	8
PIC12C672	2048x14	128x8	4 MHz	6	4	TMR0 + WDT	8
PIC12C680	512x12 FLASH	80x8 16x8 EEPROM	4 MHz	6	4	TMR0 + WDT	8
PIC12C681	1024x14 FLASH	80x8 16x8 EEPROM	4 MHz	6		TMR0 + WDT	8

Figura 2.3. *Caracteristicas de los modelos PIC12C(F)XXX de la gama enana.*

2.2.2. Gama baja o básica: PIC16C5X con instrucciones de 12 bits

Se trata de una serie de PIC de recursos limitados, pero con una de las mejores relaciones *coste/prestaciones*. Sus versiones están encapsuladas con 18 y 28 patitas y pueden alimentarse a partir de una tensión de 2,5 V, lo que les hace ideales en las aplicaciones que funcionan con pilas. Tienen un repertorio de 33 instrucciones cuyo formato consta de 12 bits. No admiten ningún tipo de interrupción y la Pila sólo dispone de dos niveles.

EJEMPLO

¿Cuántas patitas destina un PIC16C54, Figura 2.4, para líneas de E/S, sabiendo que se las denomina RAx y RBx?

SOLUCIÓN

12 patitas.

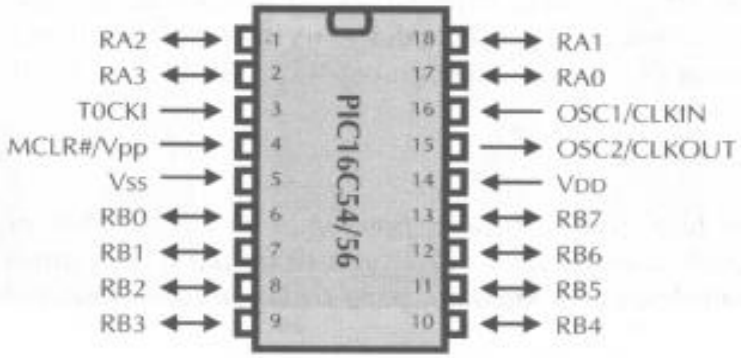


Figura 2.4. *Diagrama de patitas de los PIC de la gama baja que responden a la nomenclatura PIC16C54/56.*

2.2.3. Gama media. PIC16CXXX con instrucciones de 14 bits

Es la gama más variada y completa de los PIC. Abarca modelos con encapsulado desde 18 patitas hasta 68, cubriendo varias opciones que integran abundantes periféricos. Dentro de esta gama se halla el «fabuloso PIC16X84» y sus variantes.

El repertorio de instrucciones es de 35 de 14 bits cada una y compatible con el de la gama baja. Sus distintos modelos contienen todos los recursos que se precisan en las aplicaciones de los microcontroladores de 8 bits. También dispone de interrupciones y una Pila de 8 niveles que permite el anidamiento de subrutinas.

La gama media puede clasificarse en las siguientes subfamilias:

a) Gama media estándar (PIC16C55X);
b) Gama media con comparador analógico (PIC16C62X/64X/66X);
c) Gama media con módulos de captura (CCP), modulación de anchura de impulsos (PWM) y puerta serie (PIC16C6X);
d) Gama media con CAD de 8 bits (PIC16C7X);
e) Gama media con CAD de precisión (PIC14000);
f) Gama media con memoria Flash y EEPROM (PIC16X8X) (a esta subfamilia es a la que se dedica este libro);
g) Gama media con driver LCD (PIC16C92X).

Encuadrado en la gama media también se halla la versión PIC14C000, que soporta el diseño de controladores inteligentes para cargadores de baterías, pilas pequeñas, fuentes de alimentación ininterrumpibles y cualquier sistema de adquisición y procesamiento de señales que requiera gestión de la energía de alimentación. Los PIC14C000 admiten cualquier tecnología de las baterías como *Li Ion, NiMH, NiCd, Pb y Zinc.*

MODELO	MEMORIA PROGRAMA (x12 BITS) EPROM ROM	MEMORIA DATOS (bytes)	FRECUENCIA MÁXIMA	LÍNEAS E/S	TEMPORIZADORES	PATITAS
PIC16C52	384	25	4 MHz	4	TMR0 + WDT	18
PIC16C54	512	25	20 MHz	12	TMR0 + WDT	18
PIC16C54A	512	25	20 MHz	12	TMR0 + WDT	18
PIC16CR54A	512	25	20 MHz	12	TMR0 + WDT	18
PIC16C55	512	24	20 MHz	20	TMR0 + WDT	28
PIC16C56	1 K	25	20 MHz	12	TMR0 + WDT	18
PIC16C57	2 K	72	20 MHz	20	TMR0 + WDT	28
PIC16CR57B	2 K	72	20 MHz	20	TMR0 + WDT	28
PIC16C58A	2 K	73	20 MHz	12	TMR0 + WDT	18
PIC16CR58A	2 K	73	20 MHz	12	TMR0 + WDT	18

Figura 2.5. *Principales características de los modelos de la gama baja.*

EJEMPLO

Según la Figura 2.7, la nomenclatura de la subfamilia 16x8x diferencia a sus modelos por la letra intermedia (C, F o CR). ¿Qué significado tienen dichas letras?

SOLUCIÓN

C: *Significa que la memoria de instrucciones es EEPROM.*
F: *Indica que la memoria de instrucciones es tipo Flash.*
CR: *La memoria de instrucciones es ROM y se graba en fábrica. Sólo se usa para grandes series.*

En la Figura 2.6 se ofrece el diagrama de conexiones del PIC16C74 de 40 patitas, que es el paradigma de la variedad de recursos que pueden incluir los PIC de esta gama.

2.2.4. Gama alta: PIC17CXXX con instrucciones de 16 bits

Se alcanzan las 58 instrucciones de 16 bits en el repertorio y sus modelos disponen de un sistema de gestión de interrupciones vectorizadas muy potente. También incluyen variados controladores de periféricos, puertas de comunicación serie y paralelo con elementos externos y un multiplicador hardware de gran velocidad.

Quizás la característica más destacable de los componentes de esta gama es su *arquitectura abierta*, que consiste en la posibilidad de ampliación del microcontrolador con elementos externos. Para este fin, las patitas sacan al exterior las líneas de los buses de datos, direcciones y

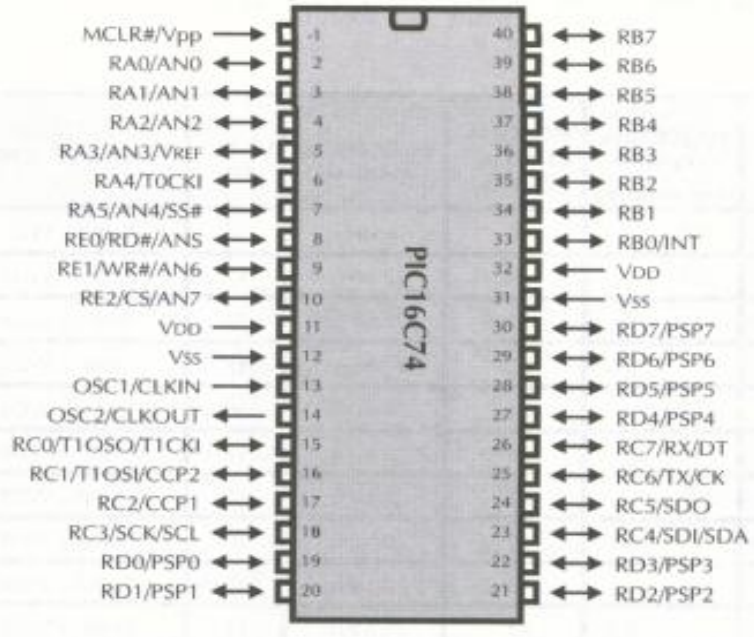


Figura 2.6. *Diagrama de patitas del PIC16C74, uno de los modelos más representativo de la gama media.*

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS		REGISTROS ESPECÍFICOS	TEMPORIZADORES	INTERRUPCIONES	E/S	RANGO VOLTAJE	PATITAS
		RAM	EEPROM						
PIC16C84	1Kx14 EEPROM	36	64	11	TMR0 + WDT	4	13	2-6	18
PIC16F84	1Kx14 FLASH	68	64	11	TMR0 + WDT	4	13	2-6	18
PIC16F83	512x14 FLASH	36	64	11	TMR0 + WDT	4	13	2-6	18
PIC16CR84	1Kx14 ROM	68	64	11	TMR0 + WDT	4	13	2-6	18
PIC16CR83	512x14 ROM	36	64	11	TMR0 + WDT	4	13	2-6	18

Figura 2.7. *Características relevantes de los modelos PIC16X8X de la gama media, que son a los que se dedica este libro.*

control, a las que se conectan memorias o controladores de periféricos. Esta filosofía de construcción del sistema es la que se empleaba en los microprocesadores y no suele ser una práctica habitual cuando se emplean microcontroladores. En la Figura 2.8 se muestran las características más relevantes de los modelos de esta gama, que sólo se utilizan en aplicaciones muy especiales con grandes requerimientos.

Con vistas al siglo XXI, **Microchip** lanzará la gama «mejorada» PIC18CXXX.

EJEMPLO

Localizar el PIC de la gama alta que tenga mayores capacidades en sus dos memorias.

SOLUCIÓN

El PIC17C756 con 16 K palabras de 16 bits cada una en la memoria de instrucciones y 902 bytes en la memoria de datos.

2.3. DESCRIPCIÓN GENERAL Y APLICACIONES

La diversidad de los modelos de PIC tiene una finalidad: *poder seleccionar el más adecuado para cada aplicación*. Por este motivo conviene tener un conocimiento fundamental y actualizado de cada gama.

En cuanto a la gama enana, la familia PIC12C5XX, se trata de un conjunto de microcontroladores CMOS encapsulados con 8 patitas. Tienen un precio muy bajo y resultan altamente competitivos.

MODELO	MEMORIA PROGRAMA	MEMORIA DATOS RAM	REGISTROS ESPECÍFICOS	TEMPORIZADORES	CAP	PWM	CAD 10 bit	INTERRUPCIONES	E/S	MULTIPLICADOR HARDWARE	PATITAS
PIC17C42A	2Kx16	232	48	4 + WDT	2	2		11	33	8x8	40/44
PIC17C43	4Kx16	454	48	4 + WDT	2	2		11	33	8x8	40/44
PIC17C44	8Kx16	454	48	4 + WDT	2	2		11	33	8x8	40/44
PIC17C752	8Kx16	454	76	4 + WDT	4	3	12	18	50	8x8	64/68
PIC17C756	16Kx16	902	76	4 + WDT	4	3	12	18	50	8x8	64/68

Figura 2.8. *Características más destacadas de los modelos PIC17CXXX de la gama alta.*

Todos los PIC enanos tienen un formato único para sus instrucciones, aunque hay modelos con 12 bits, otros con 14 y otros con 16. En este grupo los más usados son los de 12 bits, que proporcionan una extraordinaria compactación en el código de los programas, pudiéndose cifrar en el doble con respecto a otros similares.

La arquitectura RISC, tipo Harvard, unida a la segmentación del procesador, consigue que el ciclo de ejecución de una instrucción sea de 1 microsegundo cuando funciona a una frecuencia de 4 MHz. Todas las instrucciones duran este tiempo menos las de salto, que duran el doble.

Los PIC enanos son muy apreciados en las aplicaciones de control de personal, en sistemas de seguridad y en dispositivos de bajo consumo que gestionan receptores y transmisores de señales. Su pequeño tamaño los hace ideales en muchos proyectos donde esta cualidad es fundamental.

Dentro de la gama baja el PIC14000 resulta muy interesante por los recursos que incorpora: conversor A/D de media a alta resolución (de 10 a 16 bits), sensor interno de temperatura, comunicación serie I^2C y muy bajo consumo. Son aptos para el control de cargadores de baterías, monitores del estado de pilas y baterías, fuentes de alimentación ininterrumpibles, gestión del consumo de la energía de alimentación y sistemas de adquisición de datos, especialmente de temperatura.

La gama baja, representada por los PIC16C5X, tiene proyección sobre los dispositivos de alta velocidad usados en la industria de la automoción, el control de motores y los receptores/transmisores y procesadores de bajo consumo encargados de la gestión de comunicaciones. Dentro de esta gama se encuentra la subfamilia PIC16C55X, que incorpora recursos propios de la gama media. Así, dispone de Pila de 8 niveles y múltiples fuentes de interrupción.

Los microcontroladores PIC de la gama baja consiguen superar a sus competidores en velocidad, a los que aventajan en un factor cercano a 4; y también en compresión del código, que llega a ser del doble.

En la gama media hay una subfamilia, PIC16C62X, con recursos muy especiales, pero manteniendo el bajo precio y el consumo reducido, que es muy recomendable en aplicaciones relativas a la gestión de cargadores de baterías, control de sensores remotos con bajo consumo, etc.

Con diez versiones diferentes, la subfamilia PIC16C6X de la gama media se caracteriza por la incorporación de numerosos recursos (varios temporizadores, módulo de captura/comparación, puerta serie SPI e I^2C, puerta paralela PSP, etc.). Mejorando estos modelos, la serie PIC16C7X es una muestra de las posibilidades del procesador RISC funcionando a 20 MHz, con un ciclo de instrucción de 200 ns. Los componentes de esta serie son muy apreciados en los sistemas de alarma y seguridad, así como en el control remoto de sensores en automoción.

El PIC16x8x y sus variantes pertenecen a la gama media y su novedad es la inclusión de memoria EEPROM o Flash para contener el programa.

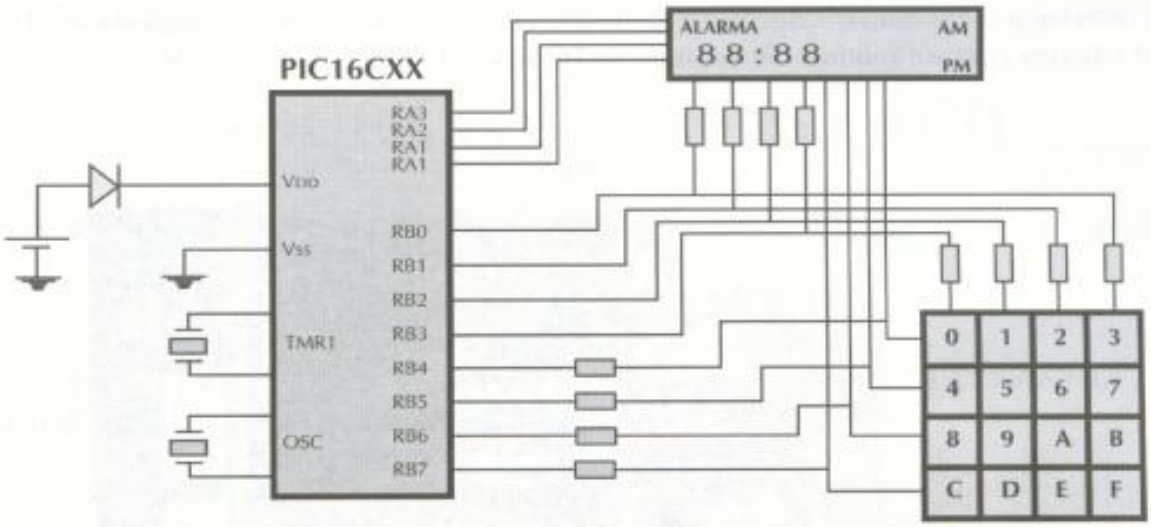


Figura 2.9. *Esquema de una aplicación típica de PIC16Cxx destinada al control de un módulo visualizador de varios dígitos y un teclado.*

La memoria EEPROM, de igual forma que la Flash, puede grabarse y borrarse eléctricamente, sin someterla a rayos ultravioleta como sucede en el borrado de las EPROM con ventana. Además, se puede realizar la grabación y el borrado en serie, lo cual posibilita la grabación de un programa, su depuración y su borrado tantas veces como se desee y manteniendo insertado el PIC en el zócalo de la aplicación. La memoria EEPROM admite hasta 1.000.000 de ciclos de escritura/borrado y almacena la información durante más de 40 años. La memoria Flash tiene un valor típico de 1.000 ciclos de escritura/borrado, pero aventaja técnicamente en varios aspectos a la EEPROM.

El hecho de poder emplear el mismo dispositivo cuando se realizan numerosos diseños o cuando se prueban muchos programas en la fase de aprendizaje resulta muy práctico y económico. Aplicaciones típicas de estos microcontroladores son el control de puertas de garaje, instrumentación, inmovilizadores de vehículos, tarjetas codificadas, pequeños sensores, etc. La grabación de los PIC16X8X en el propio circuito les hace recomendables para el almacenamiento de datos de calibración y para la modificación del programa al variar las condiciones del entorno; por ejemplo, al cambiar una pieza de un automóvil, el programa de control del motor se modifica para su adecuación.

La serie PIC16C9XX tiene como novedad la inclusión de un controlador programable de LCD, que gestiona íntegramente cualquier pantalla de cristal líquido.

Dentro de la gama alta, la subfamilia PIC17C4X, con 58 instrucciones de 16 bits, tarda en ejecutar cada una 121 nanosegundos cuando funciona a 33 MHz (las de salto tarda el doble). El núcleo del procesador está mejorado respecto al de la gama media y la Pila dispone de 16 niveles. También se han aumentado las fuentes de interrupción y se ha incorporado un multiplicador hardware de 8 x 8 bits en un ciclo, para aplicaciones que exijan un cálculo matemático intensivo. Aplicaciones industriales que requieran alta velocidad y cálculos complejos, instrumentación y telecomunicaciones, son las ideales para los componentes de la gama alta.

El PIC17C752 y el PIC17C756 son los modelos más representativos de la gama alta y están encapsulados con 64 y 68 patitas respectivamente. Tienen una elevada capacidad de memoria que alcanza los 8 K x 16 y 16 K x 16 en el área reservada a las instrucciones y 454 y 902 bytes

en la reservada a los datos, respectivamente. Poseen 12 canales para un conversor AD de 10 bits, 4 canales para un módulo de captura de 16 bits, 2 USART, bus I^2C, etc.

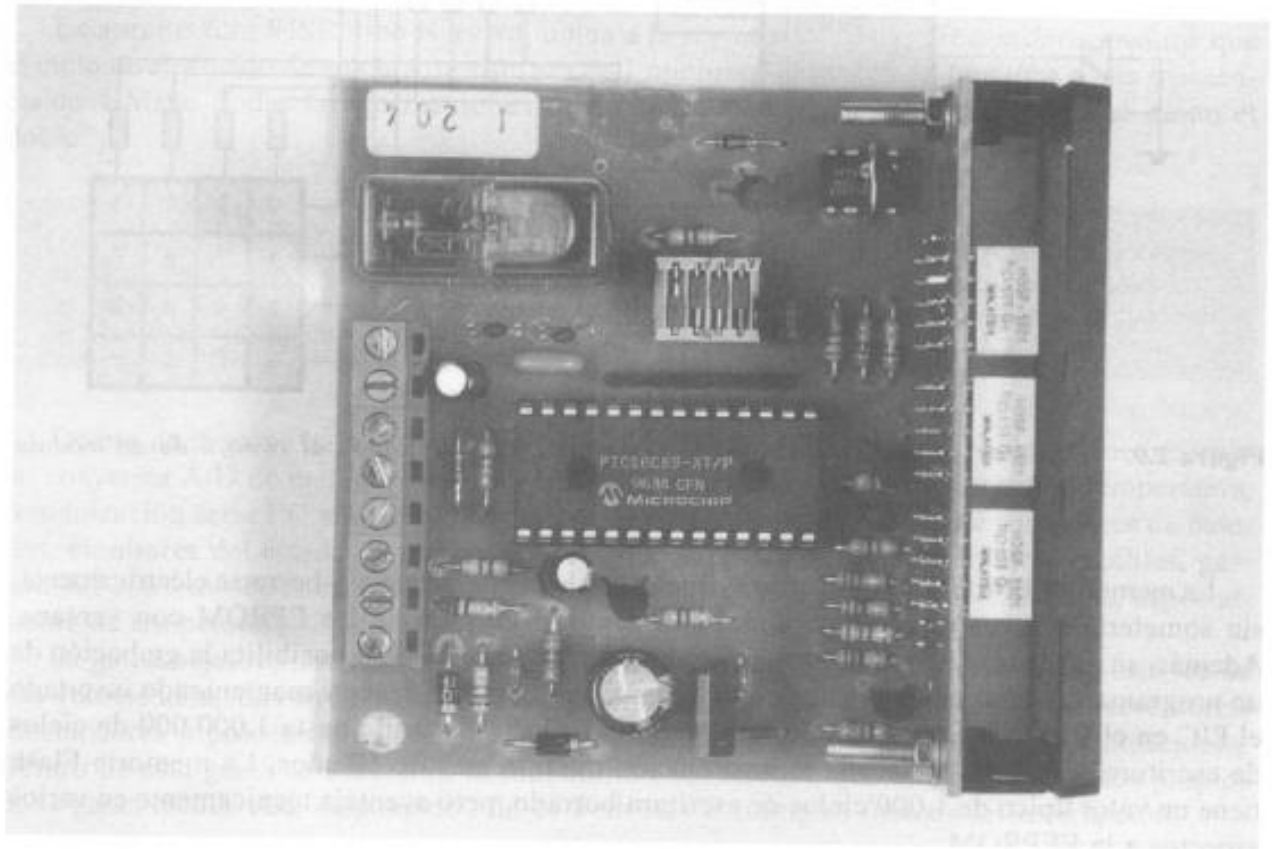


Figura 2.10. *Una típica aplicación de los microcontroladores de la gama media. Se usa un PIC16C55 para controlar un temporizador para máquinas controladas por tiempo (tipo monedero). Diseñada y fabricada por CEBEK.*

A las puertas del siglo XXI, **Microchip** presentó en el mercado mundial los nuevos microcontroladores RISC FLASH. Se trata de la serie *PIC16F87X (3, 4, 5, 6 y 7)*. Son unos modelos mucho más potentes que el *PIC16F84*, pero que mantienen sus ventajas en su grabación y borrado eléctrico.

Es una serie que encaja dentro de la gama media y que alcanza una memoria de código de hasta 8K palabras de 14 bits en memoria FLASH. Soporta hasta 386 × 8 bytes de RAM y 256 × 8 bytes de memoria de datos EEPROM. Pueden programarse sólo con +5 VDC y existe acceso a lectura y escritura de la memoria de programa.

Estos potentes microcontroladores disponen de hasta tres Timer o contadores, dos módulos de Captura, Comparación y PWM, un Convertidor Analógico/Digital de 10 bits, Canal de comunicaciones síncrono SSP con modo I2C y SPI, canal USART/SCI, puerto paralelo esclavo asíncrono de 8 bits y detección de falta de alimentación (BOR).

Los lectores que completen con éxito el primer escalón en el manejo del PIC16F84 deberán tener muy en cuenta esta potentísima familia de PIC con memoria FLASH para superar sus diseños.

PROGRAMAR PIC ES FÁCIL

EL SEGUNDO PROGRAMA DE CALENTAMIENTO: EXPLORANDO INTERRUPTORES Y ENCENDIENDO LUCES

Con el primer programa del capítulo anterior ya se conocen dos instrucciones (*mov* y *add*) y todas sus variantes. Vamos a aprender alguna nueva y hacer un segundo programa didáctico para probarlo con el SIMUPIC'84.

Se intenta manejar las líneas de E/S digitales que los microcontroladores suelen agrupar en conjuntos de 8 y que a nosotros nos gusta llamar **puertas** por la referencia que conllevan con la entrada y salida de información para los periféricos externos.

El PIC16X84 dispone de dos puertas, PUERTA A y PUERTA B, que simplificadamente denominaremos PA y PB. En este caso PA sólo posee 5 líneas (RA0-RA4) mientras que PB consta de 8 líneas (RB0-RB7). Tanto las líneas de una puerta como de otra pueden actuar de forma independiente como entrada o salida.

Todos los recursos del PIC se manejan como registros de 8 bits que están implementados físicamente como posiciones de la memoria de datos RAM. El valor de los datos que entran o salen por PA y PB están materializados en dos posiciones de la RAM, que en el caso del PIC16X84 corresponden a las direcciones 5 y 6, respectivamente.

Existen otros dos registros o posiciones de la RAM que se emplean para configurar las líneas de PA y PB como entrada o salida, de tal forma que cuando en uno de los bits de estos registros se graba un 1, la línea correspondiente de la puerta afectada actúa como entrada; mientras que si se graba un 0, actúa como salida. A estos registros de configuración de las puertas se les denomina TRISA, al que controla PA, y TRISB, al que lo hace con PB.

La memoria de datos del PIC16X84 está dividida en dos bancos: banco 0 y banco 1, y la anchura de cada posición es de 8 bits. En las posiciones 5 y 6 del banco 0 se ubican los registros PUERTAA y PUERTAB, que contienen el byte de información que ha entrado o va a salir por las líneas de dichas puertas. Con igual situación, pero en la banco 1, se hallan situados los registros TRISA y TRISB que configuran la dirección de cada línea. (Véase Figura 2.11.)

Cuando se conecta la alimentación al PIC o se reinicializa su funcionamiento mediante un Reset, se activa automáticamente el acceso al banco 0. Si se desea acceder a posiciones del banco 1 hay que poner a 1 el bit 5 de un registro llamado ESTADO y que ocupa, duplicado, dos posiciones que tienen la dirección 3 del banco 0 y del banco 1.

Si, por ejemplo, se quisiese configurar como entradas a todas las líneas de la PUERTAA y como salidas a las de la PUERTAB, habría que cargar con un 1 a todos los bits de TRISA y con un 0 a todos los bits de TRISB. Suponiendo que el bit 5 del registro ESTADO valiese 1 y se pudiese acceder al banco 1, las instrucciones necesarias para realizar la configuración de las dos puertas sería:



Figura 2.11. *Los registros PUERTAA y PUERTAB que contienen la información que entra o sale por las líneas de E/S ocupan las posiciones 5 y 6 del banco 0, mientras que TRISA y TRISB, encargados de la configuración de dichas líneas, se hallan situados en las mismas posiciones relativas pero en el banco 1.*

```
movlw 0xff    ; W ⇐ ff hex
movwf 0x05    ; W ⇒ TRISA (las líneas de PA se configuran como entradas);
movlw 0x00    ; W ⇐ 0
movwf 0x06    ; W ⇒ TRISB (las líneas de PB son salidas)
```

Cuando se comienza a trabajar por primera vez con el PIC16X84, el bit 5 del registro ESTADO se carga automáticamente con un 0, con lo que se permite el acceso a las posiciones del banco 0 de la memoria de datos. Para configurar las líneas de las puertas hay que acceder a los registros TRISA y TRISB que se hallan en el banco 1, por lo que, previamente a cargar en ellos los valores adecuados, hay que poner a 1 el bit 5 de ESTADO. Una vez configuradas las puertas, habrá que volver a poner a 0 el bit 5 del registro ESTADO para poder leer la información introducida por las líneas que funcionan como entradas o para poder enviar al exterior los bits colocados sobre las líneas de salida, cuyos valores se hallan en las posiciones 5 y 6 del banco 0.

Para resolver el nuevo ejercicio se comentan tres nuevas instrucciones:

bsf f,b: Pone a 1 el bit «b» del operando fuente «f», que es una posición de la memoria de datos o registro.

La instrucción *bsf 0x03,5* pone a 1 el bit 5 de la posición 3 de la memoria de datos, en la que se ubica el registro ESTADO.

bcf f,b: Pone a 0 el bit «b» del operando fuente «f».

La instrucción *bcf estado,5* pone a 0 el bit 5 del registro ESTADO.

goto etiqueta: Provoca un salto incondicional en la ejecución del programa hasta la instrucción que vaya precedida por el nombre de la etiqueta.

La instrucción *goto inicio* carga al PC o Contador de Programa con la dirección de la instrucción que vaya referenciada con la etiqueta *inicio*, provocando la rotura de la secuencia normal del programa.

ENUNCIADO

Se colocan tres interruptores en las líneas RA0, RA1 y RA2 de la puerta PA de un PIC16C84 y cuatro diodos led en las líneas RB0, RB1, RB2 y RB3 de PB, tal como se refleja en la Figura 2.12.

Mediante los tres interruptores se introduce un número binario de tres bits, de forma que si el interruptor está abierto mete un 1, y si está cerrado, pone la línea a tierra y mete un 0.

Se trata de confeccionar un programa con los nemónicos de las instrucciones del PIC, que llamaremos SEGUNDO.ASM y que comienza leyendo el número binario introducido por los interruptores, luego suma 2 unidades a este valor y visualiza en los diodos led el resultado binario de la operación. Un led apagado representa un 0 y encendido un 1.

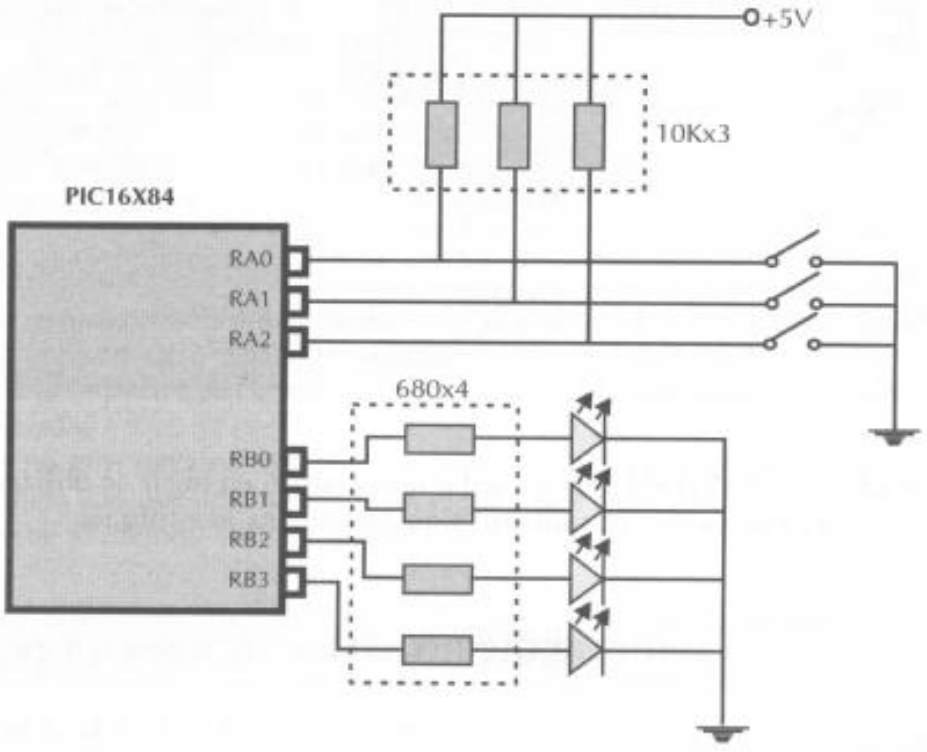


Figura 2.12. *Conexión de tres interruptores y cuatro diodos led en las líneas indicadas de las puertas PA y PB.*

SEGUNDO.ASM

```
        LIST P = 16C84      ; Comando que indica el PIC
                            ; usado

        RADIX HEX           ; Los valores se representan en
                            ; hexadecimal


        ORG 0               ; Comando que indica al
                            ; Ensamblador
                            ; la dirección de la memoria
                            ; donde se situará la
                            ; instrucción siguiente
        bsf     0x03,5      ; Pone a 1 el bit 5 de ESTADO
                            ; para direccionar el banco 1
                            ; de la memoria de datos
        movlw   0xff        ; W ⇐ ff (hex)
        movwf   0x05        ; W ⇒ TRISA
        movlw   0x00        ; W ⇐ 0
        movwf   0x06        ; W ⇒ TRISB (las líneas de PB
                            ; salidas)
        bcf     0x03,5      ; Pone a 0 el bit 5 de ESTADO
                            ; pasando a acceder al banco 0
inicio  movf    0x05,0      ; W ⇐ PUERTAA. Se introduce el
                            ; valor binario de los
                            ; interruptores
        addlw   2           ; W ⇐ W + 2
        movwf   0x06        ; W ⇒ PUERTAB. El; valor de W
                            ; sale por las líneas de PB a
                            ; los led
        goto    inicio      ; Se salta a la instrucción
                            ; precedida por la etiqueta
                            ; inicio
        END                 ; Directiva que indica el final
                            ; del programa
```

El programa SEGUNDO.ASM puede quedar más claro si en lugar de utilizar las direcciones reales en código hexadecimal se utilizan etiquetas que las identifiquen.

SEGUNDO.ASM

```
LIST P = 16C84
RADIX HEX
```

```
PUERTAA   EQU   0x05   ; La etiqueta "PUERTAA"
                       ; queda identificada con
                       ; la dirección 0x05, que
                       ; si corresponde con el
                       ; banco 0 es el valor de
                       ; PUERTAA y si es del
                       ; banco 1 con el de TRISA
PUERTAB   EQU   0x06   ; Equivalencia de la
                       ; etiqueta PUERTAB
ESTADO    EQU   0x03   ; Estado corresponde con
                       ; el valor 0x03
W         EQU   0      ; Identifica w con el
                       ; valor 0
```

```
          ORG     0
```

```
          bsf     ESTADO,5
          movlw   0xff
          movwf   PUERTAA
          movlw   0x00
          movwf   PUERTAB
          bcf     ESTADO,5
inicio    movf    PUERTAA,W
          addlw   2
          movwf   PUERTAB
          goto    inicio
          end
```

Con estos ejercicios sólo nos proponemos introducir al lector a la programación del PIC. Si ha seguido nuestro paso hasta ahora, es una persona inteligente. Si lo ve oscuro es un tío listo, porque con lo poco que hemos dicho es lógico que tenga sombras en el horizonte, pero lo peor ya ha pasado.

¡¡Todavía no sabe nada del PIC16x84 y queremos que encienda varias luces!! Este método es el que usamos con nuestros alumnos en clase y nos va muy bien, aunque, en confianza, el éxito no se debe al método sino a los alumnos.

PROBANDO EL PROGRAMA CON EL SIMUPIC'84

PROGRAMANDO EN C

Tal como se hizo en el capítulo precedente, se muestra el programa anteriormente desarrollado en ensamblador, escrito ahora en lenguaje C. El programa en cuestión es SEGUNDO.C:

```
/*SEGUNDO.C Programa que suma 2 unidades al valor que entra*/
/* por la PuertaA mediante unos interruptores y visualiza el*/
/* resultado mediante unos leds que se suponen conectados a*/
/* la PuertaB */
```

```
#include <16c84.h>          /* Dispositivo a utilizar */
#bytepuertaa=05             /* Posición de la puerta A */
#bytepuertab=06             /* Posición de la puerta B */

void main(void)
{
int valor;                  /* Valor temporal */

set_tris_a (0xff);          /* 0xff --> TRISA */
set_tris_b (0x00);          /* 0x00 --> TRISB */
do {
     valor = puertaa;       /* PuertaA --> valor */
     valor += 2;            /* se incrementa valor en 2 */
     puertab = valor;       /* valor --> puertaB */

   } while (TRUE);          /* Bucle infinito */
}
```

PROPUESTA DE UN EJERCICIO PARA «SUBIR NOTA»

Se propone un ejercicio con el que podrá comprobar los conocimientos adquiridos hasta el momento. Se trata de sumar el valor introducido mediante cuatro interruptores conectados a las líneas RA0 a RA3 de PA con el valor introducido por otros cuatro interruptores conectados a las líneas RB0 a RB3 de PB. El resultado de esta suma deberá situarse en la posición de la memoria de datos 0x0C.

Al tratarse de la suma binaria de 2 operandos de 4 bits sólo se deberán tener en cuenta los 4 bits de menos peso del registro 0x0c, aunque puede haber acarreo.

Para comprobar su programa lo mejor es que lo simule con el **Simupic'84**, pero si tiene algún problema o está algo perdido, en el disquete que acompaña al libro se encuentra una solución bajo el nombre SEGUNDO2.ASM. Su versión puede no coincidir con la propuesta aquí y, sin embargo, ser correcta. Para ir a un sitio hay muchos caminos.

DISEÑAR APLICACIONES CON PIC ES FÁCIL

PRESENTACIÓN DE LOS PERIFÉRICOS MÁS SIMPLES

Periféricos Digitales de Salida

DIODOS LED

El diodo electroluminiscente o led es uno de los dispositivos de salida digital más sencillo y económico. Según el nivel lógico que se les aplique a su cátodo o ánodo se encienden o se apagan.

En realidad todos los periféricos digitales de salida funcionan como el led. Uno de sus extremos lo tienen conectado al positivo de la alimentación o a tierra y por el otro se les aplica el nivel lógico que saca una de las líneas del microcontrolador. En el caso del diodo led, se encenderá cuando la polarización entre sus electrodos sea la directa, o sea, positivo el ánodo y negativo el cátodo.

Las líneas de los PIC pueden suministrar suficiente corriente como para encender a un diodo led, por eso se les puede conectar directamente a través de una resistencia de absorción, como muestra la Figura 2.13.



Figura 2.13. *Conexión directa de los diodos led con las líneas de salida de un PIC.*

A veces puede haber diodos led u otro tipo de carga que necesiten más corriente que la que pueden entregar las líneas del PIC, en cuyo caso es necesario introducir un transistor amplificador, como se muestra en la Figura 2.14.

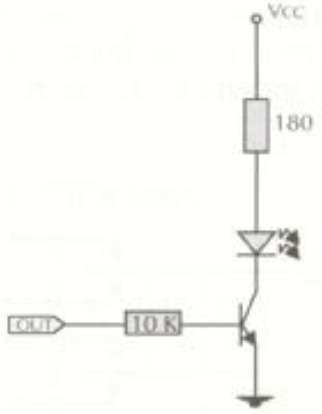


Figura 2.14. *Cuando la carga precisa más corriente que la que puede suministrar la línea de salida del PIC hay que colocar un transistor amplificador.*

RELÉS

La activación y desactivación de un relé brinda la oportunidad de poder controlar cargas mucho mayores (más corriente) porque pueden ser controladas por los contactos de dicho relé. En la Figura 2.15 se ofrece un posible esquema para el gobierno de un relé.

Cuando la línea de salida, OUT, aplica un nivel alto a la base del transistor Darlington, hace que conduzca y se active el relé. Al cerrarse los contactos de éste se controla una carga mayor. El valor de la resistencia depende del tipo de relé y del Darlington.

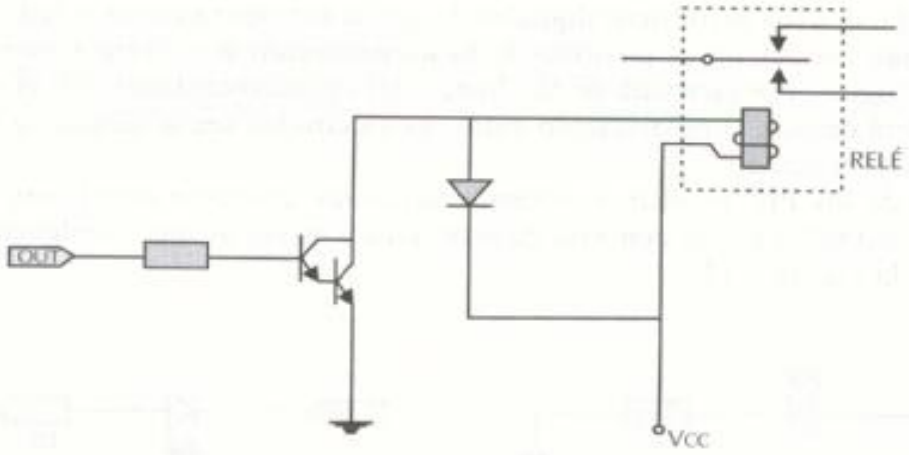


Figura 2.15. *Esquema de gobierno de un relé.*

MICRORRELÉ

Funciona como un relé, pero más pequeño y por tanto con menor intensidad para su activación y con menor posibilidad de controlar una carga grande. El circuito de conexión de un microrrelé a una línea de salida de un microcontrolador se muestra en la Figura 2.16.

El transistor PNP 2N3906 conduce y activa la bobina del microrrelé cuando sale un nivel lógico bajo por la línea OUT. El microrrelé de la figura es de doble contacto de dos posiciones, usando uno de ellos para controlar el diodo led que actúa como piloto de la activación. El otro contacto está a disposición del usuario para gobernar la carga deseada.

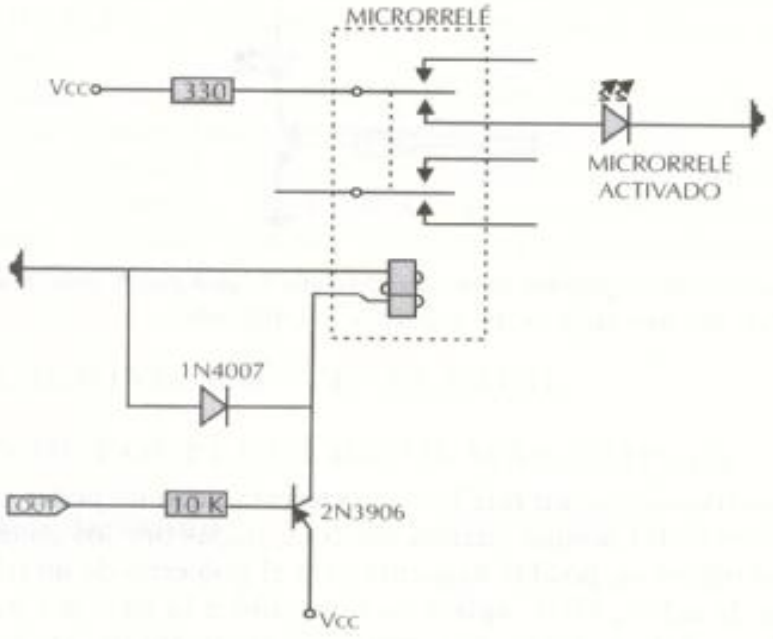


Figura 2.16. *Conexión de un microrrelé.*

SALIDAS OPTOACOPLADAS

A veces hay que aislar eléctricamente la carga a controlar de la línea de salida. Con este fin se puede usar el mismo procedimiento que para el aislamiento de las entradas, como se muestra en la Figura 2.17.

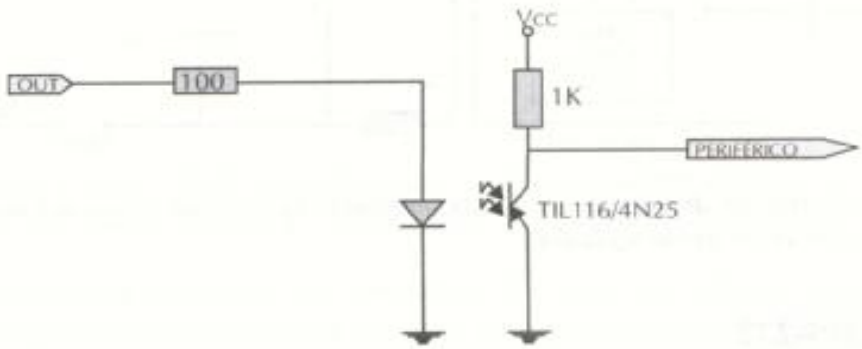


Figura 2.17. *Salidas optoacopladas, aisladas eléctricamente.*

ZUMBADOR

Se trata de un periférico que genera un zumbido sonoro cuando se activa. Hay muchos modelos y en la Figura 2.18 se muestra uno piezoeléctrico, cuya activación se produce cuando por la línea de salida aparece un nivel bajo.

Existen potentes zumbadores que precisan de una etapa previa de amplificación.

CONTROL DE CARGAS GOBERNADAS POR CORRIENTE ALTERNA

Los TRIAC son dispositivos electrónicos que dejan pasar una parte del semiciclo de la CA que se les aplica cuando a su puerta se les envía un impulso de disparo. De esta forma la potencia que entregan depende de la generación de dichos impulsos.

Se pueden controlar TRIAC con optoacopladores, tal como se muestra en la Figura 2.19.

Cuando la línea de salida OUT tiene nivel alto el encendido del fotodiodo hace conducir a TRIAC, el cual está controlando la potencia que entrega a una carga.

Conociendo la frecuencia a la que trabaja el TRIAC, por ejemplo 50 Hz, mediante soft ware se puede determinar en qué momento de cada semiciclo se genera el impulso de cebado regulando así la potencia entregada a la carga.



Figura 2.18. *Conexionado de un zumbador.*

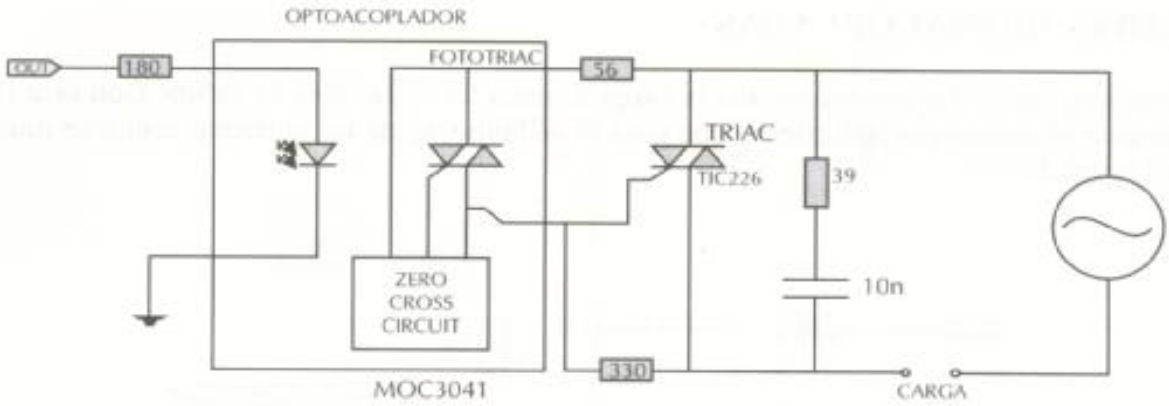


Figura 2.19. *Gobierno del disparo de un TRIAC mediante la línea de salida del microcontrolador a través de un optoacoplador.*

INTERFACE RS-232

Una aplicación verdaderamente interesante es la comunicación del microcontrolador con otros dispositivos inteligentes, como puede serlo el computador.

La norma de comunicación serie RS-232 es una de las más extendidas. Una de sus características es la de los rangos de tensiones que utiliza para representar los niveles lógicos. El nivel alto se representa con una tensión comprendida entre –3 V y –15 V, mientras que el nivel bajo utiliza el rango de +3 V a +15 V. En la Figura 2.20 se presenta un esquema para convertir los niveles lógicos proporcionados por las líneas del microcontrolador con los exigidos por la norma RS-232.

El MAX232 de la firma **MAXIM** dispone de dos canales de entrada para niveles TTL, que son el T1IN y el T2IN, con sus correspondientes salidas, R1OUT y R2OUT. Igualmente, posee dos canales de entrada para niveles RS-232 R1IN y R2IN y sus correspondientes salidas TTL, T1OUT y T2OUT. Se alimenta con +5 V.

En el esquema de la figura sólo se ha empleado uno de los dos canales disponibles. Una línea de salida del PIC (TxD) aplica la información a transmitir (TTL) por la patita T1IN del

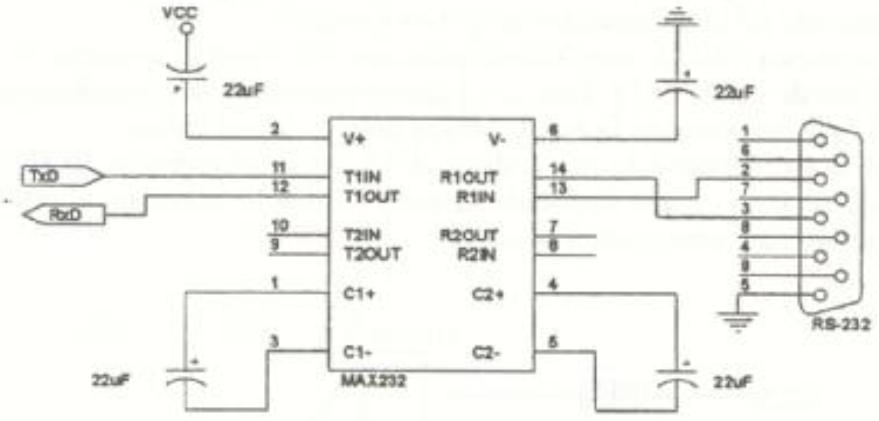


Figura 2.20. *El circuito integrado MAX232 adapta los niveles lógicos de las líneas del microcontrolador con los que exige la norma RS-232.*

MAX232. Dicha información sale por la patita R1OUT, pero convertida a niveles RS-232, enviándose por la patita 3 (TxD) de un conector estándar DB25.

La línea de recepción RS-232 (TxD) corresponde con la patita 2 del DB25. La información que se recibe se introduce por la patita R1IN del MAX232, que la convierte a niveles TTL que salen por T1OUT a otra patita del PIC.

En el interior del PIC debe existir un programa que gestione la entrada y salida de la información serie.

MOTOR PAP

Los motores paso a paso (PAP) se gobiernan mediante cuatro señales digitales. Cada vez que se introduce un nuevo patrón al motor, gira un determinado ángulo, que se denomina paso.

El PIC puede dedicar cuatro de sus líneas de E/S a sacar por ella las diversas combinaciones de niveles lógicos en los momentos precisos para gobernar el giro del motor. Mediante el programa adecuado puede controlarse con gran exactitud el posicionamiento del eje, su velocidad de giro, así como su sentido. (Figura 2.21.)

Las combinaciones binarias que deben sacar las líneas del microcontrolador dependerán de las características y especificaciones del motor. Cada combinación origina el giro de un paso, y el orden en que se aplican dichas combinaciones definen el sentido. El intervalo de tiempo que transcurre entre la salida de una combinación binaria y la siguiente determina la velocidad.

La precisión en el giro puede alcanzar un alto nivel al existir motores cuya resolución por paso es de fracciones de grado, de forma que para girar una vuelta de 360° hay que aplicar gran cantidad de combinaciones.

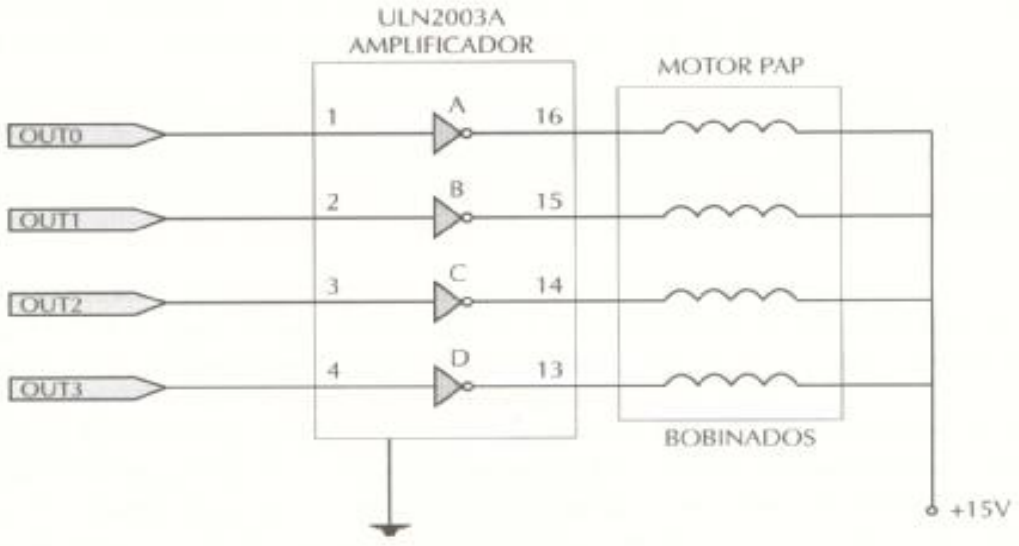


Figura 2.21. *Conexión de las 4 entradas de un motor PAP a cuatro líneas de salida del PIC. El ULN2003A es un amplificador de alta ganancia para consumos y tensiones de trabajo elevados.*

EL PRIMER CONTACTO CON EL PIC16X84



3.1. PEQUEÑO, POBRE, PERO RESULTÓN

Nuestro «maravilloso» PIC pertenece a la familia de la gama media y dentro de ella es uno de los más *pequeños;* sólo tiene 18 patitas. Además es el que dispone de menos recursos. El resto de sus parientes tiene conversores AD y DA, comparadores analógicos, tensión interna de referencia, más líneas de E/S, más capacidad en sus memorias, varios temporizadores y un largo etcétera de dispositivos de los que el PIC16X84 carece. Es «pobre» de nacimiento.

Pero nosotros hemos elegido a este PIC, en sus dos versiones C y F, porque cumple con los requisitos esenciales para enseñar a manejar los microcontroladores y comenzar a diseñar proyectos. Es práctico, es sencillo y es económico, pudiéndolo considerar como el paradigma para su empleo en todos los pequeños proyectos que realizan los aficionados, los estudiantes y quienes prefieren progresar en sus conocimientos de lo fácil a lo difícil. Creemos que sabiendo manejar un PIC, el hacerlo con todos los demás es muy fácil.

> La filosofía que hemos seguido en este libro es enseñar «bien» un modelo de PIC, el PIC16X8X, porque si lo conseguimos sabemos que va a ser inmediato manejar los restantes. Se trata del mismo perro, con el mismo collar, pero con distintas pulgas.

La razón exclusiva de su interés se debe al tipo de memoria de programa que posee. En el caso del PIC16C84 se trata de una EEPROM de 1 K palabras de 14 bits cada una. El PIC16F84 tiene la misma capacidad de memoria de instrucciones, pero de tipo Flash. Ambos disponen de 64 bytes de EEPROM como memoria de datos auxiliar y opcional.

La memoria EEPROM y la Flash son eléctricamente grabables y borrables, lo que permite escribir y borrar el programa bajo prueba manteniendo el microcontrolador en el mismo zócalo y usando el mismo dispositivo para grabar y borrar. Esta característica supone una gran ventaja con la mayoría de los microcontroladores, que tienen como memoria de programa reescribible una tipo EPROM. Se graban eléctricamente, pero para borrarlas hay que someterlas durante cierto tiempo a rayos ultravioleta, lo que implica sacar del zócalo al circuito integrado y colocarlo en un borrador de EPROM.

Microchip ha introducido la memoria Flash porque tiene mejores posibilidades de aumentar su capacidad con relación a la EEPROM. También por su mayor velocidad y menor consumo. No obstante, la EEPROM es capaz de soportar 1.000.000 de ciclos de escritura/borrado, frente a los 1.000 de la Flash.

Otra ventaja del PIC16X84 en cuanto a simplificar el proceso de escritura, borrado y reescritura de programas, tan necesario en la fase de diseño, es su sistema de grabación de datos, que se realiza en serie. Para escribir un programa en la memoria se manda la información en serie usando sólo dos patitas: la RB6 para la señal de reloj y la RB7 para los bits de datos serie.

Mostramos a continuación la tarjeta de presentación de los PIC16X84 con sus datos personales más significativos.

TARJETA DE PRESENTACIÓN DE LOS «PIC16X84»

MEMORIA DE PROGRAMA: 1 K x 14, EEPROM (PIC16C84) y Flash (PIC16F84)
MEMORIA DE DATOS RAM: 36 bytes (PIC16C84) y 68 bytes (PIC16F84)
MEMORIA DE DATOS EEPROM: 64 bytes para ambos modelos
PILA (Stack): De 8 niveles
INTERRUPCIONES: 4 tipos diferentes
JUEGO DE INSTRUCCIONES: 35
ENCAPSULADO: Plástico DIP de 18 patitas
FRECUENCIA DE TRABAJO: 10 MHz máxima
TEMPORIZADORES: Sólo uno, el TMR0. También tienen Perro Guardián (WDT)
LÍNEAS DE E/S DIGITALES: 13 (5 Puerta A y 8 Puerta B)
Corriente máxima absorbida: 80 mA Puerta A y 150 mA Puerta B
Corriente máxima suministrada: 50 mA Puerta A y 100 mA Puerta B
Corriente máxima absorbida por línea: 25 mA
Corriente máxima suministrada por línea: 20 mA
VOLTAJE DE ALIMENTACIÓN (V_{DD}): De 2 a 6 V DC
VOLTAJE DE GRABACIÓN (V_{PP}): De 12 a 14 V DC

3.1.1. Variantes del PIC16X8X

Se comercializan diferentes modelos que responden a la nomenclatura genérica PIC16X8X, atendiendo a diversas características.

1.ª Frecuencia máxima de funcionamiento.
2.ª Tipo de oscilador externo usado para generar la frecuencia de reloj.
3.ª Margen del voltaje de alimentación.

Los modelos que tienen las letras **C, F** o **CR** intermedias admiten el rango de voltaje de alimentación estándar, que es el que oscila de 4 a 4,5 V DC como mínimo, hasta 5,5 a 6 V DC

como máximo. Los modelos con las letras **LC, LF** o **LCR** intermedias admiten el margen de voltaje «extendido», que abarca desde los 2 V hasta los 6 V DC.

En la tabla de la Figura 3.1 se indican las principales características de algunos modelos, atendiendo a la frecuencia máxima y al tipo de oscilador usado.

OSCILADOR	PIC16F84-04 PIC16F83-04	PIC16F84-10 PIC16F83-10	PIC16LF84-04 PIC16LF83-04
RC	VDD: 4.0V a 6.0V IDD: 4.5 mA máx a 5.5V IPD: 14 µA máx. a 4V WDT dis Frec: 4.0 MHz máx.	VDD: 4.5V a 5.5V IDD: 1.8 mA tip. a 5.5V IPD: 1.0 µA tip. a 5.5V WDT dis Frec: 4.0 MHz máx.	VDD: 2.0V a 6.0V IDD: 4.5 mA máx. a 5.5V IPD: 7.0 µA máx. a 2V WDT dis Frec: 2.0 MHz máx.
XT	VDD: 4.0V a 6.0V IDD: 4.5 mA máx. a 5.5V IPD: 14 µA máx. a 4V WDT dis Frec: 4.0 MHz máx.	VDD: 4.5V a 5.5V IDD: 1.8 mA tip. a 5.5V IPD: 1.0 µA tip. a 5.5V WDT dis Frec: 4.0 MHz máx.	VDD: 2.0V a 6.0V IDD: 4.5 mA máx. a 5.5V IPD: 7.0 µA máx. a 2V WDT dis Frec: 2.0 MHz máx.
HS	VDD: 4.5V a 6.0V IDD: 4.5 mA tip. a 5.5V IPD: 1.0µA tip. a 4.5V WDT dis Frec: 4.0 MHz máx.	VDD: 4.5V a 5.5V IDD: 10 mA máx. a 5.5V tip. IPD: 1.0 µA tip. a 4.5V WDT dis Frec: 10 MHz máx.	NO SE USA EN EL MODO **HS**
LP	VDD: 4.0V a 6.0V IDD: 35 µA tip a 32 kHz, 3.0V IPD: 0.6µA tip. a 3.0V WDT dis Frec: 200 kHz máx.	NO SE USA EN EL MODO **LP**	VDD: 2.0V a 6.0V IDD: 32µA máx. a 32 kHz, 3.0V IPD: 7 µA máx. a 2.0V WDT dis Frec: 200 kHz máx.

Figura 3.1. *Principales características de diversos modelos de PIC16X8X.*

La buena aceptación en el mercado del «primogénito» PIC16C84 ha animado al fabricante **Microchip** a comercializar nuevas versiones. Ha diseñado cuatro nuevos modelos que se muestran en la Figura 3.2. Dos de ellos intercalan la letra **F**, que significa que la memoria EEPROM

	RELOJ	MEMORIA						PERIFÉRICOS		CARACTERÍSTICAS	
	FRECUENCIA MÁXIMA (MHz)	MEMORIA DE PROGRAMA: FLASH	MEMORIA DE PROGRAMA: EEPROM	MEMORIA DE PROGRAMA: ROM	MEMORIA RAM DE DATOS (bytes)	DATOS EEPROM (bytes)	TEMPORIZADORES	FUENTES DE INTERRUPCIÓN	PATITAS DE E/S	RANGO DE VOLTAJE (V)	ENCAPSULADOS
PIC16C84	10	—	1K	—	36	64	TMR0 + WDT	4	13	2.0-6.0	18 patitas DIP, SOIC
PIC16F84	10	1K	—	—	68	64	TMR0 + WDT	4	13	2.0-6.0	18 patitas DIP, SOIC
PIC16CR84	10	—	—	1K	68	64	TMR0 + WDT	4	13	2.0-6.0	18 patitas DIP, SOIC
PIC16F83	10	512	—	—	36	64	TMR0 + WDT	4	13	2.0-6.0	18 patitas DIP, SOIC
PIC16CR83	10	—	—	512	68	64	TMR0 + WDT	4	13	2.0-6.0	18 patitas DIP, SOIC

Figura 3.2. *Variantes del PIC16C84.*

se ha sustituido por otra tipo FLASH. Los otros dos modelos la sustituyen por memoria ROM con máscara, pero todos ellos mantienen los 64 bytes de EEPROM para datos. En el año 2000 existen los modelos PIC16F87X enormemente más potentes que el PIC16F84.

3.1.2. Diferencias del PIC16C84 con los demás modelos PIC16X8X

1.ª *La memoria de programa es de tipo Flash en los PIC16F8X y de tipo ROM en los PIC16CR8X.*
2.ª *El PIC16F84 y el PIC16CR84 tienen mayor capacidad de la RAM de datos. Disponen de 68 bytes en lugar de los 36 del PIC16C84.*
3.ª *Se ha invertido la polaridad del bit de control PWRTE, que activa y desactiva un temporizador encargado de retrasar el funcionamiento del PIC al aplicarle la alimentación.*
4.ª *Se ha incorporado un filtro a la entrada de la patita MCLR#, por lo que se requiere un pulso de mayor duración para generar el Reset.*
5.ª *Se han mejorado muchas especificaciones eléctricas en los modelos F y CR.*

3.2. EL ASPECTO EXTERNO

Nuestro fabuloso PIC está fabricado con tecnología CMOS de altas prestaciones y encapsulado en plástico con 18 patitas. (Figura 3.3.)

Se comenta brevemente la misión de cada una de las patitas.

V_{DD}: Patita por la que se aplica la tensión positiva de la alimentación.

V_{SS}: Patita conectada a tierra o negativo de la alimentación.

OSC1/CLKIN: Patita por la que se aplica la entrada del circuito oscilador externo que proporciona la frecuencia de trabajo del microcontrolador.

OSC2/CLKOUT: Patita auxiliar del circuito oscilador.

MCLR#: Esta patita es activa con nivel lógico bajo, lo que se representa con el simbolo #. Su activación origina la reinicialización o Reset del PIC.

También se usa esta patita durante la grabación de la memoria de programa para introducir por ella la tensión, V_{PP}, que está comprendida entre 12 y 14 V DC.

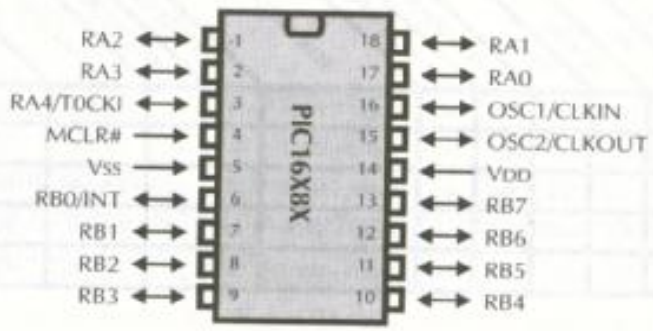


Figura 3.3. *Diagrama del conexionado de las patitas del PIC16X8X.*

RA0 - RA4: Son las 5 líneas de E/S digitales correspondientes a la Puerta A.

La línea RA4 multiplexa otra función expresada por TOCKI. En este segundo caso sirve para recibir una frecuencia externa para alimentar al temporizador interno TMR0.

RB0 - RB7: Estas 8 patitas corresponden a las 8 líneas de E/S digitales de la Puerta B.

La línea RB0 multiplexa otra función, que es la de servir como entrada a una petición externa de una interrupción, por eso se la denomina RB0/INT.

Los PIC16X8X tienen un encapsulado de 18 patitas, dos de ellas soportan la tensión de alimentación, otras dos reciben la señal del oscilador externo y otra se utiliza para generar un Reset. Las 13 patitas restantes funcionan como líneas de E/S para controlar las aplicaciones.

EJEMPLO

En el diseño de un juguete controlado por un programa de 915 instrucciones hay que construir, inicialmente, un prototipo y posteriormente una serie de 25.000 unidades. ¿Qué modelos de PIC16X8X se usarán en cada caso?

SOLUCIÓN

Para el prototipo se usará el PIC16C84 o el PIC16F84. Para la serie de 25.000 unidades se empleará el PIC16CR84, cuyo programa de instrucciones se grabará en fábrica.

3.3. LA FRECUENCIA DE FUNCIONAMIENTO. EL RELOJ

La frecuencia de trabajo del microcontrolador es un parámetro fundamental a la hora de establecer la velocidad en la ejecución de instrucciones y el consumo de energía.

Cuando un PIC16X8X funciona a 10 MHz, que es su máxima frecuencia, le corresponde un ciclo de instrucción de 400 ns, puesto que cada instrucción tarda en ejecutarse cuatro periodos de reloj, o sea, 4 × 100 ns = 400 ns. Todas las instrucciones del PIC se realizan en un ciclo de instrucción, menos las de salto, que tardan el doble.

Los impulsos de reloj entran por la patita OSC1/CLKIN y se dividen por 4 internamente, dando lugar a las señales Q1, Q2, Q3 y Q4, mostradas en la Figura 3.4. Durante un ciclo de instrucción, que comprende las 4 señales mencionadas, se desarrollan las siguientes operaciones:

Q1: Durante este impulso se incrementa el Contador de Programa.

Q4: Durante este impulso se busca el código de la instrucción en la memoria del programa y se carga en el Registro de Instrucciones.

Q2-Q3: Durante la activación de estas dos señales se produce la decodificación y la ejecución de la instrucción.

Para conseguir ejecutar cada instrucción en un ciclo de instrucción (excepto las de salto, que tardan dos), se aplica la técnica de la segmentación o *«pipe-line»*, que consiste en realizar en paralelo las dos fases que comprende cada instrucción.

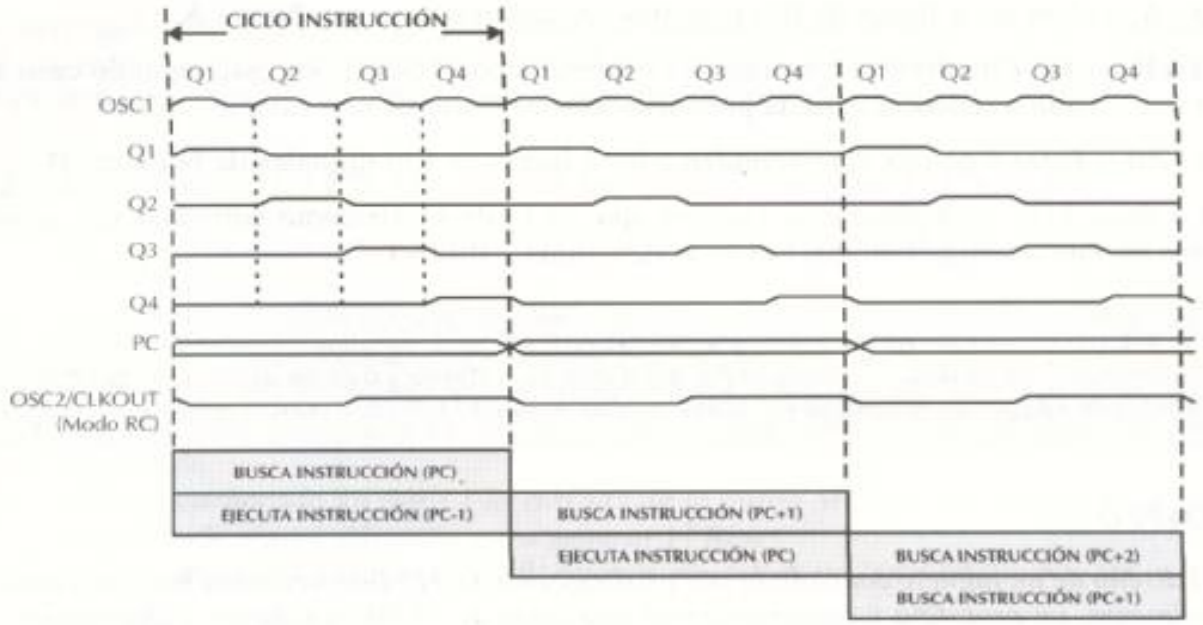


Figura 3.4. *Los impulsos del reloj externo (OSC1) se dividen por 4 formando las señales Q1, Q2, Q3 y Q4, que configuran un ciclo de instrucción.*

En realidad, cada instrucción se ejecuta en dos ciclos: en el primero se lleva a cabo la fase de búsqueda del código de la instrucción en la memoria del programa, y en el segundo se decodifica y se ejecuta (fase de ejecución). La estructura segmentada del procesador permite realizar al mismo tiempo la fase de ejecución de una instrucción y la de búsqueda de la siguiente. Cuando la instrucción ejecutada corresponde a un salto no se conoce cuál será la siguiente hasta que se complete, por eso en esta situación se sustituye la fase de búsqueda de la siguiente instrucción por un ciclo «vacío», originando que las instrucciones de salto tarden en realizarse dos ciclos de instrucción. (Figura 3.5.)

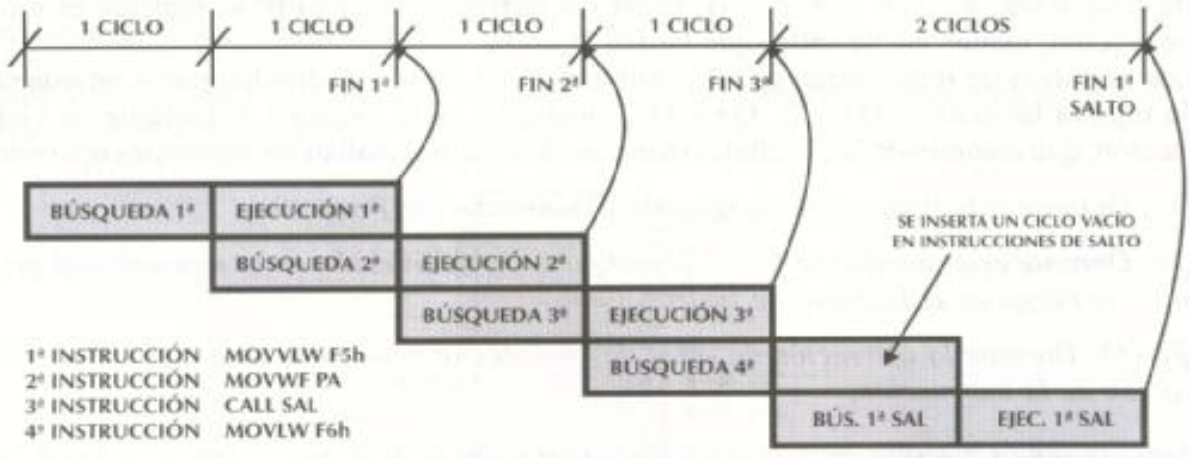


Figura 3.5. *La segmentación permite solapar en el mismo ciclo la fase de ejecución de una instrucción y la de búsqueda de la siguiente, excepto en las instrucciones de salto.*

La frecuencia de funcionamiento de un microcontrolador es un parámetro fundamental del que depende la velocidad de ejecución del programa y el consumo de energía. Cada modelo de PIC dispone de diversas versiones para distintas frecuencias.

La técnica de la segmentación unida a la arquitectura Harvard del procesador permite al PIC16C84 superar la velocidad de sus competidores directos. Así, por ejemplo, es 1,54 veces más rápido que el microcontrolador de Motorola 68HC05 cuando ambos funcionan a la misma frecuencia de 4 MHz.

EJEMPLO

Un PIC16X84 que funciona a 4 MHz ejecuta un programa de 1.000 instrucciones, de las cuales el 25 por 100 son de salto. Calcular el tiempo que tarda en ejecutarlo.

SOLUCIÓN

$T_{CICLO\ INSTRUCCIÓN} = 4 \cdot T_{RELOJ} = 4 \cdot 250\ ns = 1.000\ ns = 1\ \mu s$

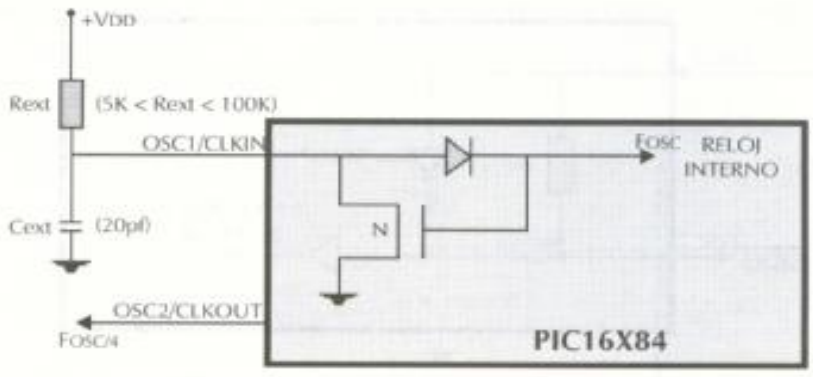
$T_{PROGRAMA} = 750 \cdot 1 + 250.\ 2 = 1.250\ \mu s$

3.3.1. Tipos de osciladores

Los PIC admiten cuatro tipos de osciladores externos para aplicarles la frecuencia de funcionamiento. El tipo empleado debe especificarse en dos bits (FOSC1 y FOSC0) de la «Palabra de Configuración», como se comenta más adelante.

Oscilador tipo «RC»

Se trata de un oscilador de bajo coste formado por una simple resistencia y un condensador. Proporciona una estabilidad mediocre de la frecuencia, cuyo valor depende de los valores de los dos elementos de la red R-C. (Figura 3.6.)



FOSCILACIÓN	Rext	Cext
625 KHz	10K	20pF
80 KHz	10K	220 pF
80 Hz	10K	0.1 µF

Figura 3.6. *Esquema del oscilador tipo RC, formado por un condensador y una resistencia externos. Se indican los valores que deben tener estos dos componentes para obtener algunas frecuencias de oscilación.*

EJEMPLO

Un PIC16F84 funcionando con una red R-C de 10 K y 220 pF, respectivamente, está ejecutando un programa de 500 instrucciones de las cuales el 20 por 100 son de salto. ¿Cuánto tarda en ejecutar el programa?

SOLUCIÓN

Según la Figura 3.6, con una red R-C de 10 K y 220 pF se trabaja a 80 KHz, de donde:

$T_{OSC} = 1s / 80 \cdot 10^3 = 12{,}5\ \mu s$

$T_{CICLO\ INSTRUCCIÓN} = 4 \cdot Tosc = 50\ \mu s$

$T_{PROGRAMA} = 400 \cdot 50 + 100 \cdot 100 = 30.000\ \mu s = 30\ ms$

Oscilador tipo «HS»

Se trata de un oscilador que alcanza una alta velocidad comprendida entre 4 y 10 MHz y está basado en un cristal de cuarzo o un resonador cerámico.

Oscilador tipo «XT»

Es un oscilador de cristal o resonador para frecuencias estándar comprendidas entre 100 KHz y 4 MHz.

Oscilador tipo «LP»

Oscilador de bajo consumo con cristal o resonador diseñado para trabajar en un rango de frecuencias de 35 a 200 KHz.

El cristal de cuarzo o el resonador cerámico se coloca entre las patitas OSC1 y OSC2. (Figura 3.7.)

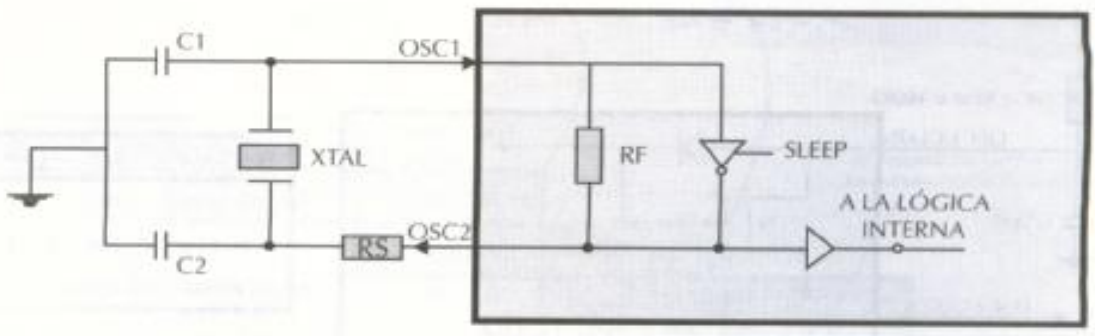


Figura 3.7. *Según el tipo de oscilador y la frecuencia de trabajo se emplean diferentes valores en los condensadores C1 y C2 que acompañan al cristal de cuarzo. Para frecuencias comprendidas entre 4 MHz y 10 MHz, los condensadores tienen una capacidad de 15 pF a 33 pF. La resistencia RS sólo es necesaria en algunas versiones tipo HS.*

EJEMPLO

Para una aplicación en la que el coste es un factor preponderante y no le afecta la precisión del tiempo, ¿cuál sería el oscilador más adecuado?

SOLUCIÓN

El oscilador R-C. El coste de una resistencia y un condensador es mínimo, aunque dejan bastante que desear la precisión y la estabilidad de la frecuencia de trabajo.

3.4. REINICIALIZACIÓN O RESET

Cuando se aplica un nivel lógico bajo en la patita MCLR# el microcontrolador reinicializa su estado. Dos acciones importantes se producen en la reinicialización o Reset:

1.ª El Contador de Programa se carga con la dirección 0, apuntando la primera dirección de la memoria de programa en donde deberá estar situada la primera instrucción del programa de aplicación.
2.ª La mayoría de los registros de estado y control del procesador toman un estado conocido y determinado.

Se puede ocasionar el Reset de varias maneras, que estudiaremos más adelante.

El circuito más simple para provocar un Reset manualmente al presionar un pulsador se muestra en la Figura 3.8.

3.5. EL PRIMER CIRCUITO PRÁCTICO

Con lo poco que sabemos ya se puede poner en marcha el PIC16X84. Conocemos la función de todas sus patitas, 13 de las cuales se emplean para adaptarse con los periféricos externos. De las 5 patitas restantes, dos (V_{DD} y V_{SS}) aplican la tensión de alimentación, otras dos (OSC1 y OSC2) soportan el cristal de cuarzo del reloj externo y la que queda (MCLR#), ya se ha expuesto cómo hay que conectarla para generar el Reset.

En los circuitos donde se usa un PIC16X84 es muy habitual utilizar como tensión de alimentación la de 5 V normalizada y como circuito de reloj externo uno del tipo XT a una

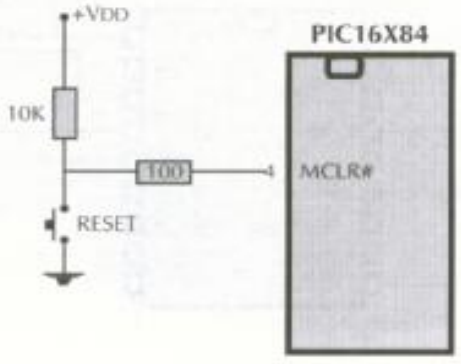


Figura 3.8. *Sencillo circuito que permite producir un Reset al presionar un pulsador externo.*

frecuencia de 4 MHz. Con esta configuración, el conexionado fijo para cualquier aplicación del PIC es el mostrado en la Figura 3.9.

Las patitas que no se hallan conectadas en la Figura 3.9 son las dedicadas a transferir información con los periféricos que utilice la aplicación. Para hacer operativo dicho esquema vamos a conectar algunos periféricos sencillos, que no por eso son los menos usados.

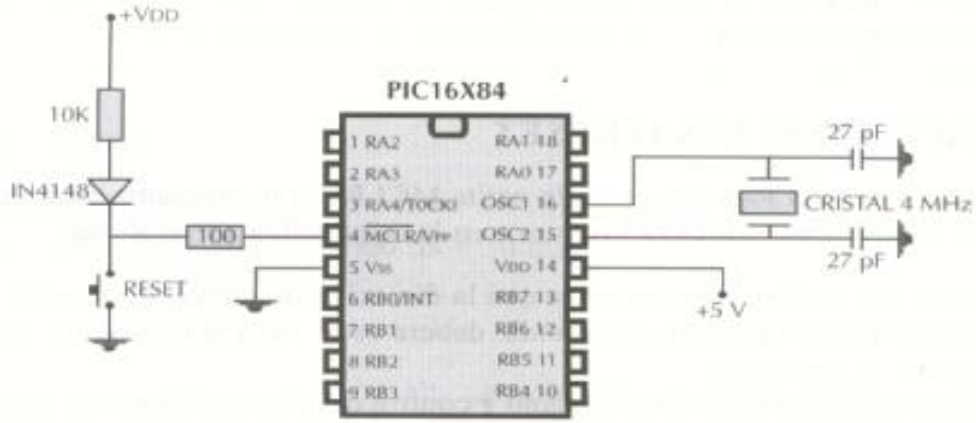


Figura 3.9. *Conexionado fijo del PIC16X84 para cualquier aplicación.*

Como periférico de entrada utilizaremos un interruptor (I1) que se conecta a la línea RA1 de la Puerta A. Como periférico de salida se colocará un diodo led (L3) en la línea RB3 de la Puerta B. Según el interruptor se abra o se cierre introducirá un nivel lógico alto o bajo. Cuando está abierto aplica a la patita RA1 un nivel lógico alto a través de la resistencia de pull-up de 10 K. Si se cierra, al conectarse con tierra, introduce un nivel bajo. (Figura 3.10.)

Aprovechando la corriente de salida de las líneas de la Puerta B, en torno a los 25 mA, conectaremos directamente el diodo led L3 a la patita RB3, a través de una resistencia de absorción de 330 Ω.

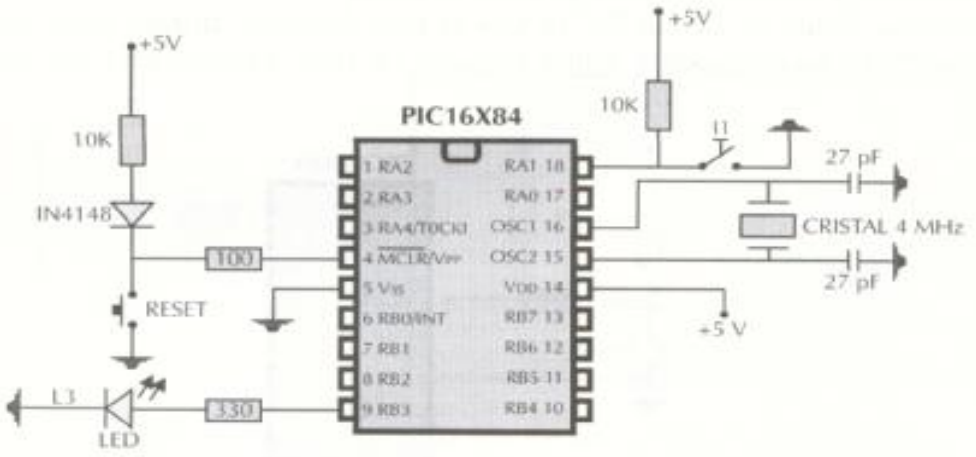


Figura 3.10. *Circuito de conexionado de un PIC16X84 en disposición de controlar un periférico de entrada, el interruptor I1, y un periférico de salida, el diodo led L3.*

3.6. UNA SOLUCIÓN AL DISEÑO DE PROYECTOS: LA TARJETA DE PROTOTIPOS

El PIC precisa una circuitería auxiliar común para cualquier aplicación, cambiando únicamente el conexionado de las líneas de E/S a los periféricos. Existe una forma muy eficaz de desarrollar el prototipo de un diseño mediante una tarjeta de circuito impreso premontada y con una zona libre para instalar y cablear los periféricos.

Una tarjeta de prototipos consta de dos zonas. En una se ubica el zócalo que alojará al microcontrolador con el programa ya grabado junto a la circuitería común (alimentación, reloj y Reset). En la otra zona sólo existe un área de agujeros perforados y estañados sobre los que se conectan los periféricos y se unen fácilmente con las líneas de E/S que quedan accesibles. En la Figura 3.11 se muestra una tarjeta de prototipos para PIC de 18 patitas, diseñada por **Microsystems Engineering**. Una ventaja adicional de esta tarjeta es que además del zócalo donde se ubica el PIC, existe otro de 18 patitas, correspondientes a las del microcontrolador, que puede conectarse por cable plano con un sistema de desarrollo o un grabador para que el *PIC16X8X resida en el propio grabador y se pueda escribir y borrar el programa hasta que* quede definitivamente a punto. Llegado ese momento se quita el cable plano del grabador y se coloca el PIC grabado en el zócalo de la tarjeta de prototipos.

En la Figura 3.12 se muestra el esquema eléctrico de la circuitería que viene montada en la tarjeta de prototipos PIC18-ME y que es la imprescindible para el funcionamiento del

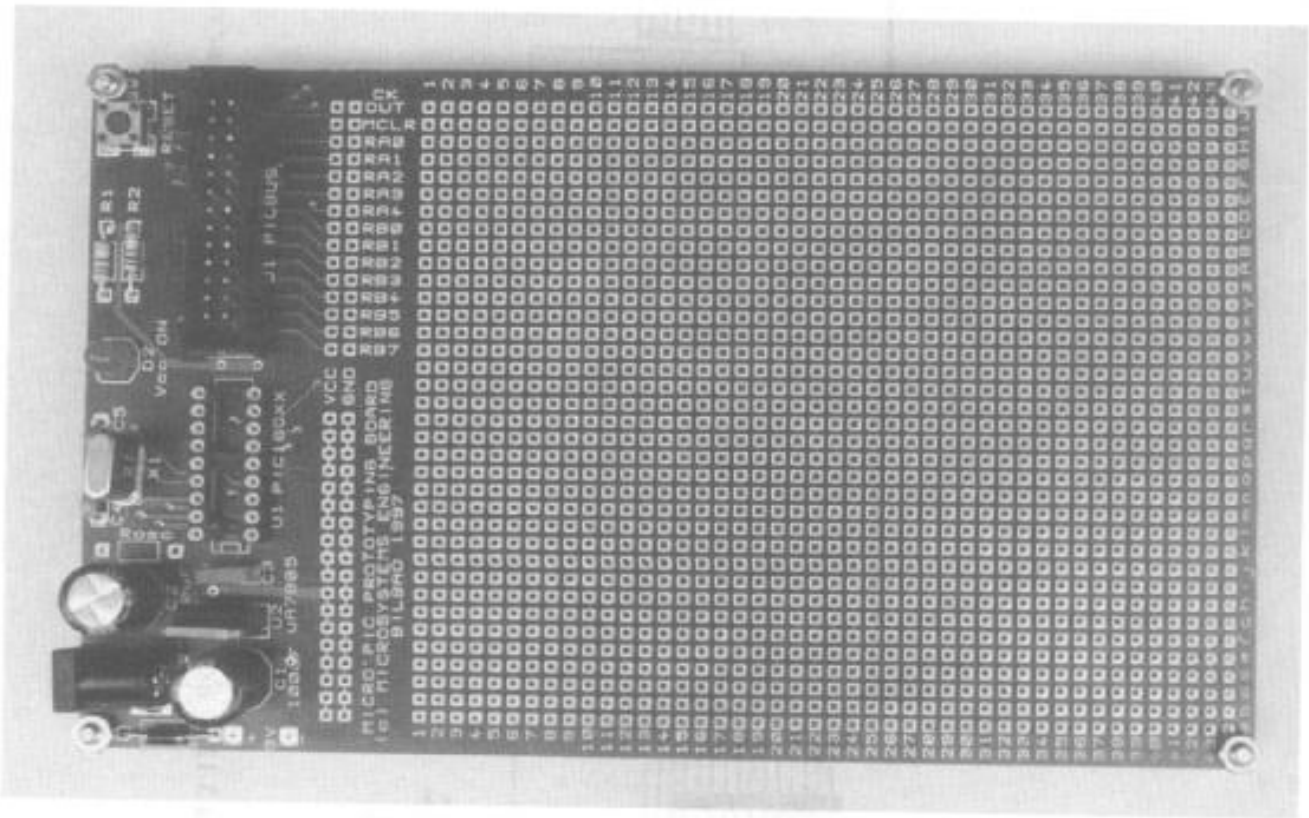


Figura 3.11. *Fotografía de la tarjeta de prototipos para los PIC de la gama media de 18 patitas, diseñada por* ***Microsystems Engineering***.

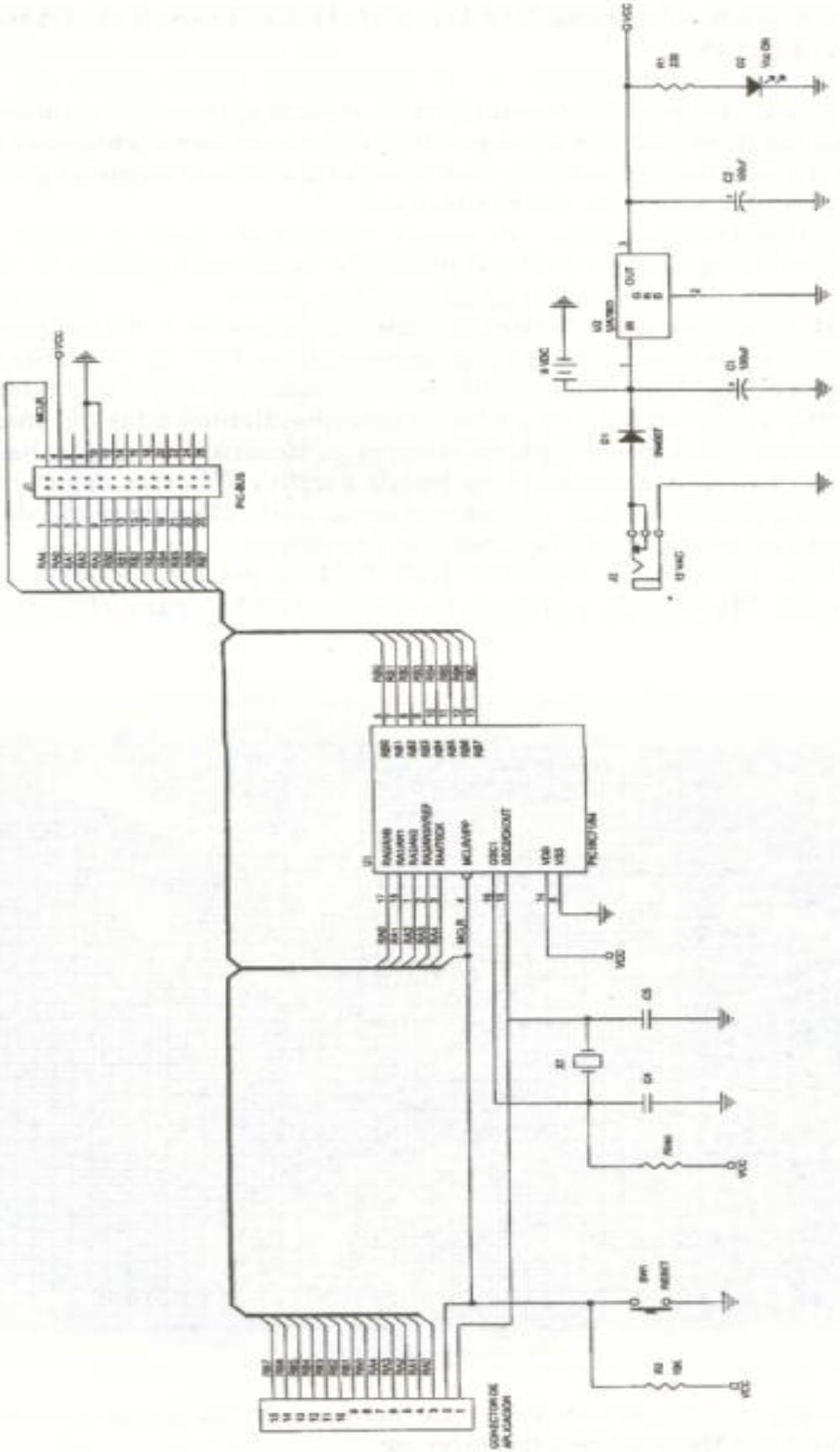


Figura 3.12. *Esquema eléctrico de la circuitería auxiliar del PIC que viene montada en la tarjeta de prototipos.*

microcontrolador. La alimentación es de 12 VAC procedente de un transformador externo y aplicada a través del conector J2. El diodo D1 rectifica la corriente alterna y el condensador C1 actúa como filtro. El regulador de tensión U2, modelo µA7805, junto con C2, proporcionan la tensión de trabajo estabilizada de + 5 V. También es posible alimentar al circuito con una pila de 9 V.

Las patitas de las Puertas A y B del PIC colocado en el zócalo de la tarjeta quedan accesibles desde el «conector de aplicación» por el que se unen con los periféricos alojados en la zona libre. Dichas patitas también están disponibles, junto a la línea de Reset MCLR#, en el conector J1 PIC-BUS, capaz de conectarse con el sistema de desarrollo Micro'PIC Trainer con un cable plano de 26 hilos.

Además de las patitas de las líneas de E/S de las Puertas, en el conector de aplicación existen dos contactos para la señal de Reset, MCLR#, y para la señal de salida del oscilador interno, CLKOUT.

La tarjeta de prototipos PIC18-ME es una herramienta rápida, compacta y segura para construir el prototipo de cualquier diseño que utiliza un PIC de 18 patitas.

PROGRAMAR PIC ES FÁCIL

EL TERCER PROGRAMA CON DISEÑO HARDWARE

Junto con un nuevo programa se intenta desarrollar la circuitería electrónica asociada para proporcionar al lector una idea completa que aclare el interfaz del hardware con el software.

En cuanto a novedades sobre instrucciones sólo vamos a emplear una desconocida y muy fácil:

***comf f,d**: Complementa el registro fuente f (cambia los 1 por 0 y viceversa) y el resultado lo deposita en el destino d. Si d = 0, el destino es W, y si d = 1, el destino es el registro fuente f.*

EJEMPLO

comf PUERTAA,0: Complementa el contenido del registro «PUERTAA» y deposita el resultado en W.

ENUNCIADO: MÁS INTERRUPTORES Y DIODOS LED

Se dispone de cinco interruptores que actúan como periféricos de entrada y se conectan a las cinco líneas de la Puerta A de un PIC16X84 funcionando a 4 MHz con oscilador externo tipo XT.

Como periféricos de salida actúan cinco diodos led unidos a las líneas RB0-RB4 de la Puerta B. También se dispone de un pulsador manual para provocar el Reset.

*Diseñar el esquema electrónico de conexionado del PIC y confeccionar un programa en lenguaje Ensamblador MPASM de **Microchip**, que explore el estado de los interruptores y que ilumine el led correspondiente al interruptor que se halle cerrado.*

En la Figura 3.13 se ofrece el esquema electrónico del conexionado del PIC con la circuitería auxiliar y con los periféricos de entrada y salida.

Al utilizar el programa Ensamblador MPASM, se comienza indicando el tipo de PIC que se usa y el sistema de numeración:

```
LIST  p=16c84 ; Se usa el PIC16C84
RADIX hex     ; Se emplea el sistema de
              ; numeración hexadecimal
```

Con objeto de no emplear en el programa las direcciones numéricas de los registros que se usan, se asigna a cada uno de ellos una etiqueta identificativa que facilita la comprensión del

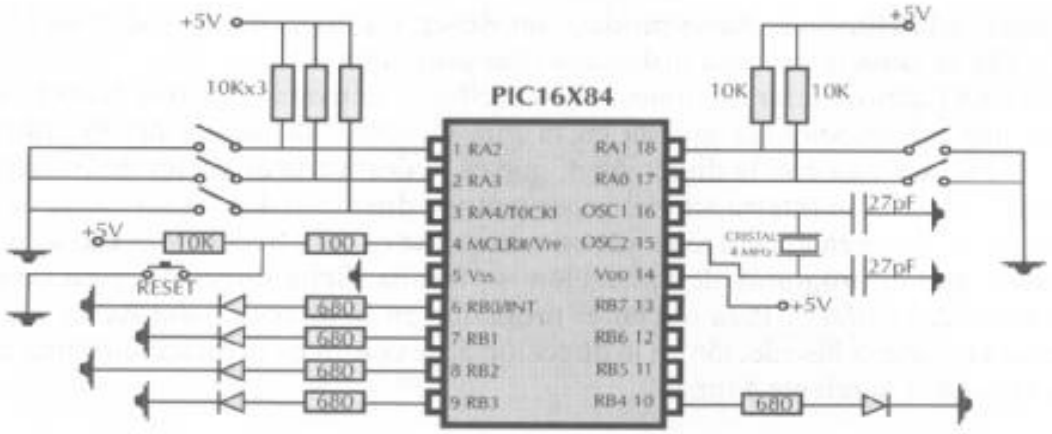


Figura 3.13. *Esquema electrónico del conexionado al PIC16X84 de la circuitería auxiliar común y los periféricos de entrada y salida del ejercicio.*

programa. Hay registros que se hallan en los dos bancos de la memoria de datos, pero otros sólo en uno. Hay que tener en cuenta que cuando se inicializa el programa tras un Reset, el procesador apunta automáticamente al banco 0 y si se desea acceder al banco 1 es preciso poner a 1 el bit 5 del registro Estado, que se halla en la dirección 3 de ambos bancos. La información que entra o sale por la Puerta A se implementa en la dirección 5 del banco 0, mientras que en esa dirección, pero del banco 1, se halla el registro de configuración de las líneas de E/S de la Puerta A. Usaremos la etiqueta «PUERTAA» para ambos registros, pero accederemos a cada uno según el banco que esté activado en ese momento. Lo mismo haremos para los dos registros de la Puerta B, a los que denominaremos «PUERTAB» y se ubican en la dirección 6.

```
; ZONA DE ETIQUETAS-----------------------------------------------

    W       EQU  0      ; Cuando el destino es W, d = 0
    f       EQU  1      ; Cuando el destino es el
                        ; registro f, d=1
    PUERTAA EQU  0x05   ; La Puerta A (datos) ocupa la
                        ; dirección 5 del banco 0 y el
                        ; registro de configuración la
                        ; dirección 5 del banco 1
    PUERTAB EQU  0x06   ; La misma etiqueta para el
                        ; registro de datos y el de
                        ; configuración de la
                        ; Puerta B
    ESTADO  EQU  0x03   ; El registro Estado ocupa
                        ; dirección 3 de los dos bancos
```

Una vez asignadas las etiquetas a los registros y elementos que maneja el programa, se pasa a delimitar el mapa de la memoria del programa.

Como tras realizarse un Reset el PC apunta la dirección 0 de la memoria de programa, en dicha posición habrá que situar la primera instrucción. Al conectarse por primera vez el mi-

crocontrolador a la alimentación se produce un Reset. La dirección 0 se destina al Vector de Reset y en ella se sitúa la primera instrucción del programa.

El PIC16X84 admite interrupciones, una de ellas al activarse la patita RB0/INT. Cuando se produce una interrupción se guarda en la Pila el contenido actual del PC (dirección de retorno) y el PC se carga con la dirección 4, que es la destinada al Vector de Interrupción. En este programa no se usan interrupciones y por tanto la dirección 4 de la memoria de programa se podría utilizar libremente, pero teniendo en cuenta que es muy frecuente el uso de interrupciones, conviene que el programa de aplicación no invada dicha dirección y en consecuencia comience en la dirección 5. Para iniciar el programa en la dirección del Reset 0 y al mismo tiempo situar la primera instrucción en la dirección 5, se coloca en la dirección 0 una instrucción de salto a la 5 de la siguiente forma:

```
ORG   0        ; El programa comienza en la
               ; dirección 0
               ; (Vector de reset)
goto  inicio   ; Se salta a la etiqueta "inicio"
ORG   5        ; Se asigna la dirección 5 a la
               ; siguiente instrucción que contiene
               ; la etiqueta inicio
```

La primera instrucción tiene como etiqueta «inicio» y el Ensamblador MPASM la situará en la dirección 5, gracias a la directiva previa *org 5*.

Para configurar las líneas de la Puerta A como entradas y las de la Puerta B como salidas hay que cargar el registro de configuración de la Puerta A con unos y el de la Puerta B con ceros. Aunque la etiqueta que hace referencia al registro de datos y al de configuración de la Puerta A es la misma «PUERTAA», el acceso a uno u otro se efectúa según el banco de la memoria de datos que se halle activo. Para acceder a los registros de configuración hay que poner a 1 el bit 5 del registro Estado.

Cuando un interruptor está cerrado introduce un nivel bajo y como se desea que el diodo led correspondiente se encienda, es necesario complementar dicho bit para que los diodos, que se encienden por nivel alto, respondan al enunciado del proyecto. Para esta operación se utiliza la instrucción de complemento.

```
inicio bsf    ESTADO,5    ; Pone a 1 el bit 5 de
                          ; Estado.
                          ; Acceso al banco 1.
       clrf   PUERTAB     ; Se configuran como
                          ; salidas las
                          ; líneas de la Puerta B.
       movlw  0xff        ; El registro W se carga
                          ; con unos.
       movwf  PUERTAA     ; Se configuran como
                          ; entradas las
                          ; líneas de la Puerta A
       bcf    ESTADO,5    ; Pone a 0 el bit 5 de
                          ; Estado.
                          ; Acceso al banco 0.
```

```
        movf    PUERTAA,W   ; Carga el registro de
                            ; datos de
                            ; Puerta A en W.
bucle   comf    PUERTAA,W   ; Complementa a 1 los
                            ; datos de la puerta A y
                            ; lo deposita en W.
        movwf   PUERTAB     ; El contenido de W se
                            ; deposita en
                            ; el registro de datos de
                            ; la Puerta B.
        goto    bucle       ; Se crea un bucle cerrado
                            ; e infinito
                END         ; Fin del programa.
```

Si Vd. es capaz de simular este programa con el simulador SIMUPIC'84 que se incluye con este libro y cuyo manejo se ofrece en un Apéndice; si, además, dispone de un sistema de desarrollo como el Micro'PIC Trainer de **Microsystems Engineering** o de un grabador de PIC para escribir este programa en uno de ellos y, finalmente, monta en una tarjeta de prototipos el esquema de la Figura 3.13 y al conectar el PIC grabado le funciona el ejercicio correctamente, ¡¡ES UN MONSTRUO!!

De todas formas, si Vd. no hace nada de lo que le hemos comentado en el párrafo anterior, pero va entendiendo todo lo que le contamos, YA SABE CÓMO FUNCIONAN LOS MICROCONTROLADORES. Pero tenga en cuenta que hasta que no haga con sus propias manos una aplicación y diseñe un programa y ponga todo ello en marcha, VD. NO SE APROVECHARÁ DE TODO LO QUE LE PUEDEN DAR LOS PIC.

PROBANDO EL PROGRAMA CON EL SIMUPIC'84

Una gran cualidad que posee el SIMUPIC'84 es la de poder simular el estado lógico que se introduce por las patitas y que actúan como líneas de entrada. Se usa la ventana «VISIÓN EXTERNA», que muestra el comportamiento de las líneas de salida ante los estímulos externos aplicados a las entradas.

En esta ocasión conviene ejecutar el programa todo seguido y no trazarlo instrucción a instrucción; de esta manera podremos hacer uso de las teclas F1 a F5 para introducir niveles lógicos por las patitas RA0 a RA4 del PIC que se simula. Cada vez que se pulsa una de estas teclas el valor digital de la entrada correspondiente voltea, es como si se accionase el interruptor correspondiente.

Durante la ejecución del programa las entradas RA0 a RA4 se ponen de color verde, mientras las salidas (de RB0 a RB7) adoptan un color rojo. Queda comprobar que el valor de las salidas RB0 a RB4 (donde realmente irán los leds) es siempre el inverso del estado de las entradas RA0 a RA4. Cada vez que se pulsa una tecla de función, el valor de la línea correspondiente de la puerta B cambia.

PROGRAMANDO EN C

El listado que se presenta a continuación es una de las posibles soluciones, realizada para el compilador PCM de la casa **Custom Computer Services**:

```
/* TERCERO.C: Lee el valor de cinco interruptores en la puerta A */
/* y saca por la puerta B el valor inverso                       */
```

```
#device pic16c84
#byte   puertaa = 05 /* Posición de la puerta A */
#byte   puertab = 06 /* Posición de la puerta B */
```

```
void main (void)
{
 int temp;
 set_tris_a( 0xFF ); /* Puerta A configurada para entrada */
 set_tris_b( 0x00 ); /* Puerta B configurada para salida  */

do {                 /* La variable temporal es necesaria porque */
temp = ~puertaa;     /* la complementación se traduce en dos
                        pasos:*/
   puertab = temp;   /* a) Copia de puertaa en temp */
}                    /* b) Complementación de temp */
                     /* De no hacerlo así los LED parpadearían */

while( 1 );          /* Repetir ininterrumpidamente */
}
```

En el cuerpo del bucle principal se aprecia otra de las desventajas del lenguaje C: *«no se conoce cuál va a ser la traducción a lenguaje máquina»*. Puede que hubiese sido más lógico hacer `"puertab = ~puertaa"`. Sin embargo esto no da resultado porque los leds se encienden y apagan constantemente (pruebe a simularlo). Sucede así porque el PCM traduce esa instrucción en el siguiente código:

```
                    movf    0x05,0
                    movwf   0x06
                    comf    0x06,1
```

De forma que primero se copia en W el contenido de la puerta A, de aquí se mueve a la puerta B y luego se complementa. Eso quiere decir que durante un ciclo de instrucción, la puerta A valdrá lo mismo que la puerta B y en el siguiente ciclo lo contrario (al complementarse). De ahí el efecto de parpadeo y la conclusión de que sólo el lenguaje ensamblador nos da un control absoluto sobre la máquina.

¿SE ATREVE CON ESTE EJERCICIO? PRIMEROS PASOS CON EL MICRO'PIC TRAINER

¿Se ha conformado con ver una simple simulación en la pantalla de su ordenador? ¿No le queda la duda de si el montaje realmente funcionaría en condiciones reales? Es nor-

mal. Por mucho que nos digan que una simulación es fiel a la realidad nunca dejará de ser software.

Vamos a comprobar que nuestro montaje funciona REALMENTE, con un PIC auténtico. Para eso utilizaremos el Micro'PIC Trainer de **Microsystems Engineering**, que nos va a permitir grabar un PIC 16X84 y ejecutar el programa de control. Además dispone de todos los periféricos necesarios para nuestro montaje, con lo que la comprobación va a ser realmente sencilla. En el Apéndice C se describe el esquema y el funcionamiento del Micro'PIC Trainer.

Existe un apéndice al final del libro en el que se explica detalladamente el uso del Micro'PIC Trainer, pero como más de uno estará ansioso por empezar, se describe brevemente su funcionamiento. En el CD encontrará un magnífico tutorial multimedia.

Conecte el Micro'PIC Trainer al puerto paralelo de un PC y ejecute en este último el programa de control PICME-TR. En la pantalla de computador seleccionamos la opción de «Borrar». Es importante comprobar que la barra de leds, la pantalla LCD y el display de siete segmentos están desconectados mediante los *jumpers* correspondientes antes de escribir cualquier cosa en el PIC, ya que estos periféricos están conectados a la puerta B, que a su vez es utilizada durante la grabación serie del PIC.

Se carga el programa TERCERO.HEX del disquete que ya está ensamblado, y se vuelca directamente en la memoria de programa del PIC mediante la opción «Programar Todo». Ya está todo listo para ejecutar el programa.

En el sistema de desarrollo Micro'PIC Trainer hay cinco interruptores que están conectados a la puerta A y que seleccionaremos cambiando de posición los *jumpers* que se encuentran sobre ellos a la posición DIGITAL. Además los *jumpers* J9 y J10 deberán estar en la posición A4 y B0 respectivamente. También se conectará la barra de leds mediante J7. En estas condiciones, al accionar el pulsador de Reset se ejecuta el programa TERCERO. Cada vez que se cierra un interruptor deberá encenderse el led correspondiente.

Este mismo montaje puede realizarse en una placa de prototipos. Necesitará grabar el PIC16X84 en el Micro'PIC Trainer o un grabador cualquiera. Y dos cosas más: «una tarde y el dinero que se gastaría en ella». Vd. elige, pero no se pierda un buen plan por ver cómo se encienden y apagan unos leds cuando acciona unos interruptores.

DISEÑAR APLICACIONES CON PIC ES FÁCIL

Visualizadores con displays de 7 segmentos

Cuando se precisa realizar presentaciones de números la solución más económica es el display de 7 segmentos. En realidad cada display consta de 8 segmentos, pues además de los 7 (a, b, c, d, e, f y g) que sirven para mostrar el número, hay un octavo para indicar el punto decimal (dp), si hay que ponerlo detrás del dígito.

En la Figura 3.14, las 8 líneas de la Puerta B actúan como salidas y sirven para iluminar cada uno de los 8 segmentos del display. Como existen 4 displays en paralelo y los segmentos de cada uno están conectados con los correspondientes de los otros 3, es necesario activar al que debe iluminar en cada momento el código que sacan las líneas de la Puerta B.

Si la línea RA0 saca un nivel alto consigue la activación del dígito D0 a través del bloqueo del transistor correspondiente. La información presente en ese momento sobre las líneas de la Puerta B corresponderá a los segmentos que deben iluminarse en dicho dígito. Después se

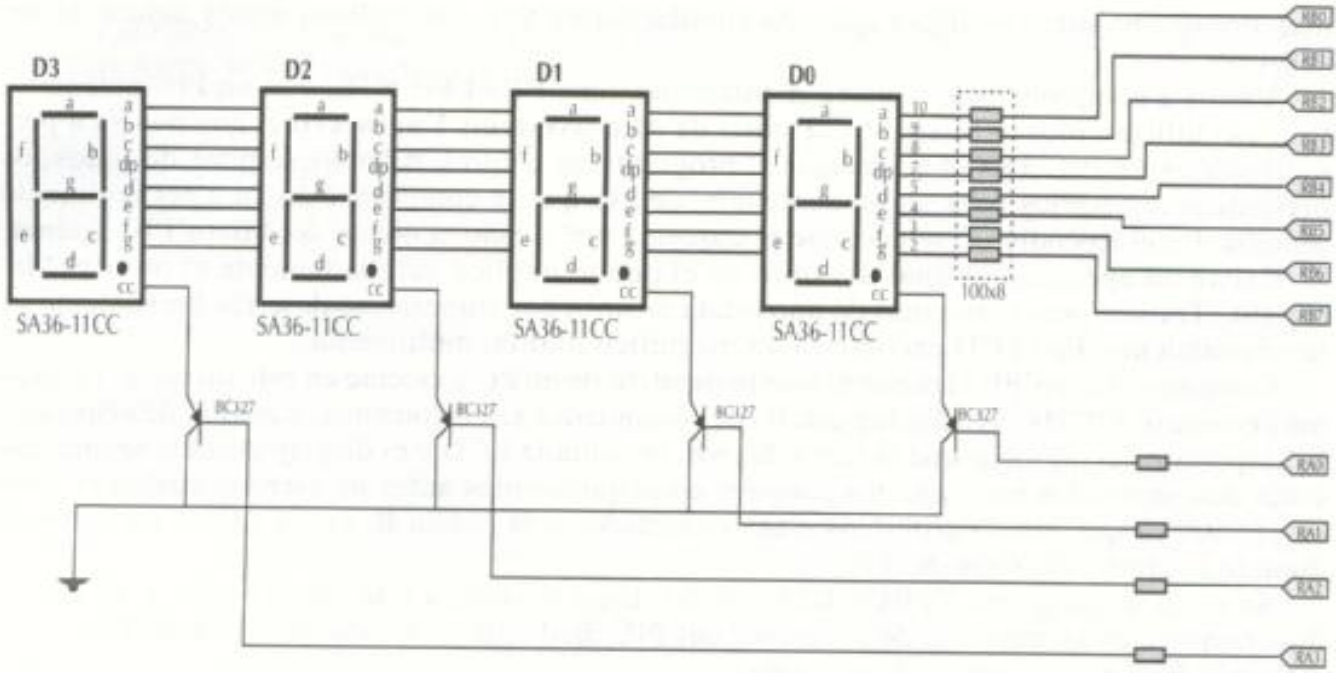


Figura 3.14. *Esquema de conexionado de un visualizador de 4 displays de 7 segmentos a 12 líneas del PIC.*

repite el proceso con los demás dígitos D1, D2 y D3. Este proceso se conoce con el nombre de «barrido» de los displays y en él se van activando secuencialmente uno a uno. Si se realiza la iluminación de cada dígito cada poco tiempo provoca la sensación óptica de que todos están iluminados, cada uno con su dígito. Se estima un tiempo de activación de cada display entre 10 y 20 ms para originar dicha sensación.

TECLADO

Desde el punto de vista eléctrico, cada tecla de un teclado es un mecanismo idéntico a un pulsador. La aportación del teclado consiste en la configuración de las teclas para que necesiten pocas líneas de entrada en la detección de la que se ha presionado.

Para disminuir las líneas necesarias para detectar la tecla pulsada, éstas se agrupan de forma matricial en filas y columnas. Con esta configuración un teclado matricial de 16 teclas sólo precisa 8 líneas del PIC para su gestión. Si cada tecla actuase como un pulsador individual se precisarían 16 líneas de E/S del microcontrolador para gestionarlas. En la Figura 3.15 se muestra el esquema de un teclado matricial, modelo ECO 16250 06 de la empresa **SECME.**

En la Figura 3.15 las 4 líneas de menos peso de la Puerta B (RB0-RB3) se configuran como salidas que aplican un patrón de estados lógicos a las 4 columnas del teclado. Las 4 líneas de más peso de la Puerta B (RB4-RB7) están configuradas como entradas y reciben los niveles lógicos que tienen las filas del teclado.

El programa que gestiona el teclado saca, secuencialmente, un nivel bajo por una de las 4 líneas de salida que se aplica a las columnas, al mismo tiempo que lee el nivel lógico introdu-

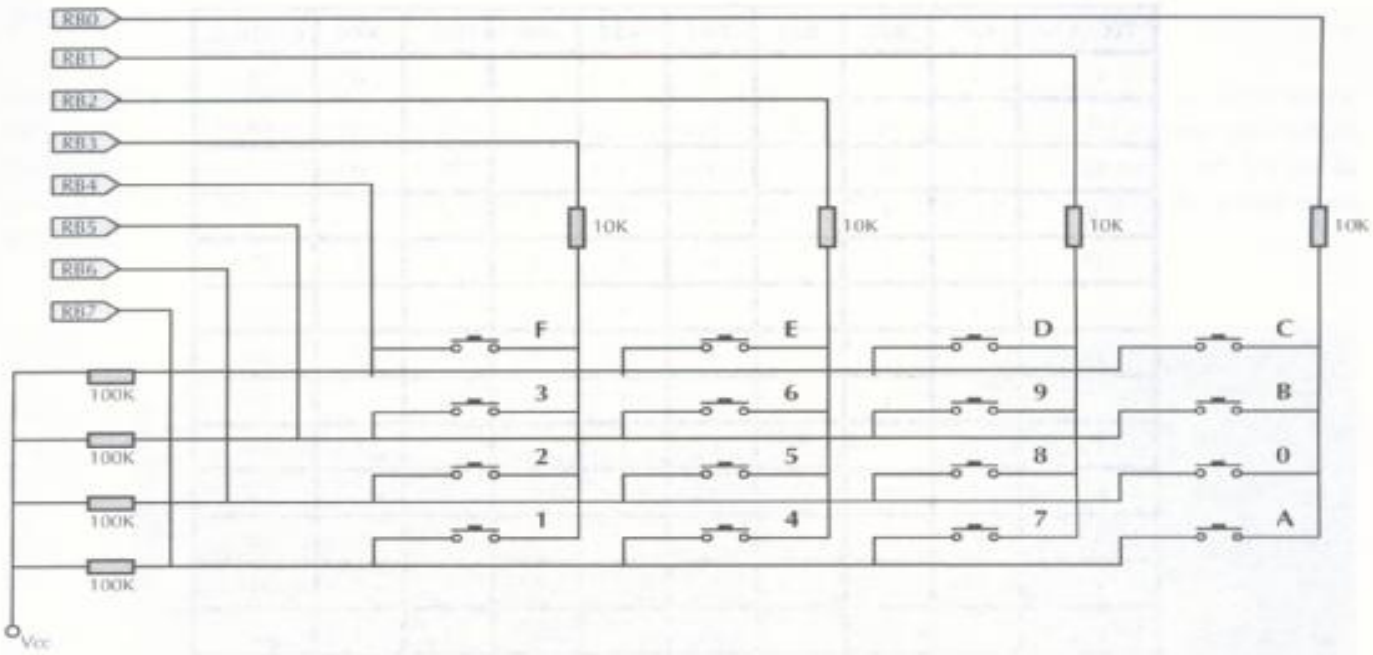


Figura 3.15. *Conexionado de un teclado matricial, modelo ECO 16250 06, con las 8 líneas de la Puerta B de un PIC.*

cido por las filas en las líneas RB4-RB7. Si ninguna de las teclas de la columna por la que se introduce el nivel bajo está pulsada, se leerá un nivel alto en las 4 filas, pasándose a activar la siguiente columna.

Si se aplica en una columna un nivel bajo y al leer las filas una de ellas se encuentra a nivel bajo, se deduce que la tecla asociada a dicha fila y dicha columna se halla presionada. Así se averigua la tecla que se presiona en cada momento.

Cada tecla tiene asociado un código binario que se muestra en la tabla de la Figura 3.16 y que se corresponde con los 4 bits que sacan las líneas RB0-RB3 y los 4 recibidos por RB4-RB7.

Los códigos de exploración de las teclas varían según los modelos y la colocación de las mismas. El programa que gestiona el teclado realiza la exploración cada cierto tiempo, que suele ser de un valor aproximado a los 20 ms.

También es el software el encargado de realizar el tratamiento oportuno cuando se pulsan varias teclas a la vez, eliminar los rebotes, implementar funciones de repetibilidad, etc.

TECLADO/DISPLAY

Una de las aplicaciones más usuales que afrontan los diseños basados en microcontroladores son las que manejan el conjunto de un teclado y un display. En la Figura 3.17 se muestra el esquema de conexionado de un display de 4 dígitos y un teclado matricial de 16 teclas.

Debe ser el programa que gestiona las líneas de E/S del microcontrolador el encargado de

TECLA Nº	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	CÓDIGO
C	1	1	1	0	1	1	1	0	EE
B	1	1	0	1	1	1	1	0	DE
0	1	0	1	1	1	1	1	0	BE
A	0	1	1	1	1	1	1	0	7E
D	1	1	1	0	1	1	0	1	ED
9	1	1	0	1	1	1	0	1	DD
8	1	0	1	1	1	1	0	1	BD
7	0	1	1	1	1	1	0	1	7D
E	1	1	1	0	1	0	1	1	EB
6	1	1	0	1	1	0	1	1	DB
5	1	0	1	1	1	0	1	1	BB
4	0	1	1	1	1	0	1	1	7B
F	1	1	1	0	0	1	1	1	E7
3	1	1	0	1	0	1	1	1	D7
2	1	0	1	1	0	1	1	1	B7
1	0	1	1	1	0	1	1	1	77

Figura 3.16. *Códigos binarios que corresponden a cada una de las 16 teclas del teclado modelo 16250 06 de SECME.*

efectuar barridos periódicos en el teclado y en el display, así como reconfigurar las líneas de E/S del PIC.

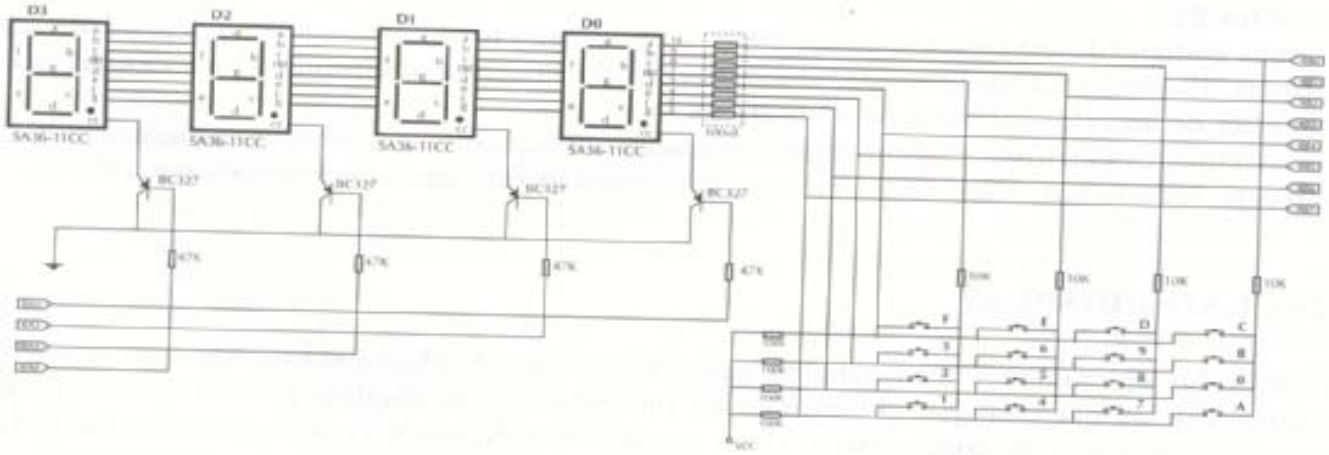


Figura 3.17. *Esquema de conexionado de un display de 4 dígitos y un teclado matricial de 16 teclas.*

PANTALLA DE CRISTAL LÍQUIDO

La pantalla de cristal líquido (LCD) es uno de los visualizadores más utilizados en la actualidad debido a las importantes ventajas que ofrece. Se trata de módulos inteligentes que están gobernados con microcontroladores. En la fotografía de la Figura 3.18 se aprecia un kit de la casa **CEBEK** que maneja una pantalla LCD, lo cual se presenta por la parte de atrás para reconocer los dos microcontroladores que la gestionan.

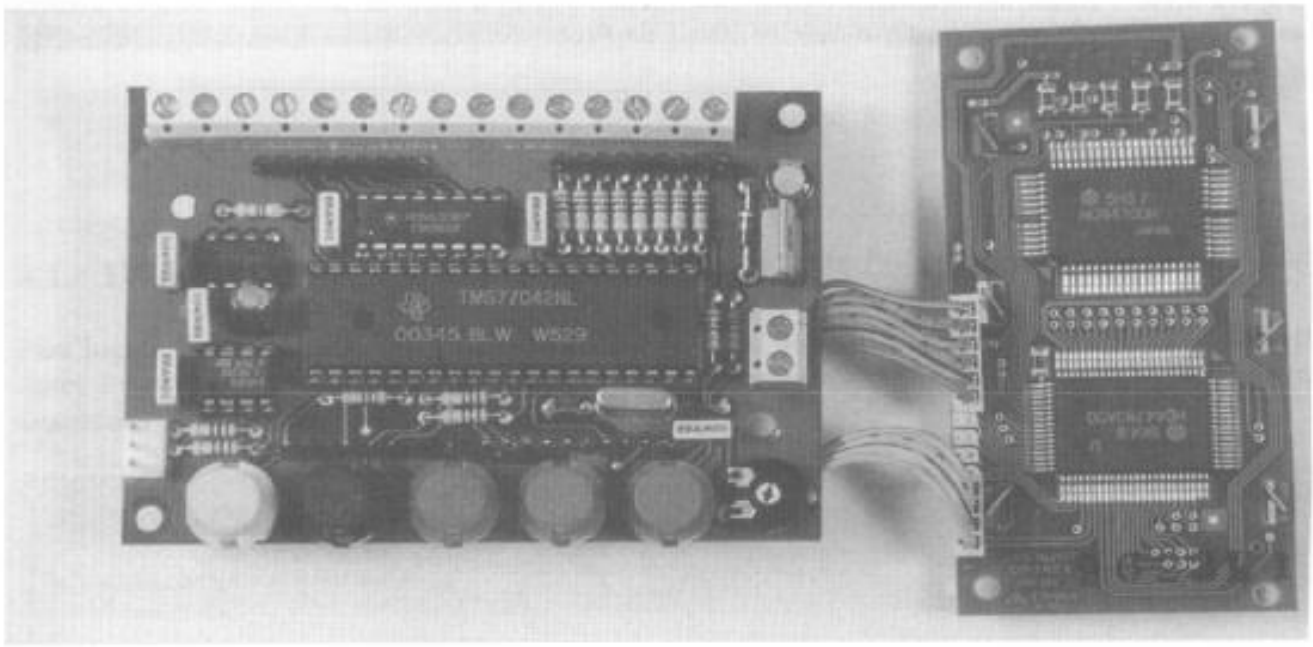


Figura 3.18. *Fotografía de un interesante kit de CEBEK que maneja una pantalla LCD, que está gestionada mediante microcontroladores especializados.*

Los mensajes se visualizan en la pantalla LCD introduciendo los correspondientes códigos ASCII de cada uno de los caracteres a visualizar. Además, un extenso juego de caracteres predefinidos en fábrica permiten usar nuevos caracteres y símbolos definidos por el usuario. A pesar de que el número de modelos comerciales es muy grande, las líneas necesarias para su conexión y control son prácticamente las mismas. En la Figura 3.19 se ofrece el esquema de conexionado de un módulo capaz de mostrar dos líneas de 16 caracteres cada una.

Los códigos ASCII de los caracteres se introducen por las 8 líneas de la Puerta B, mientras que las señales de control se aplican mediante tres de las líneas de la Puerta A. RA1 corresponde con la señal de control R/W#, que indica si el módulo LCD va a ser leído o escrito. RA2 se aplica a la entrada de activación E (Enable), con la que se indica si el módulo está activado o desactivado. Finalmente, con la línea RA0 se controla la señal RS, que permite el acceso a los distintos registros de control del módulo para establecer las condiciones de la visualización.

El programa de control residente en el PIC debe encargarse de gestionar las tres señales de control y el envío de los caracteres ASCII por la Puerta B.

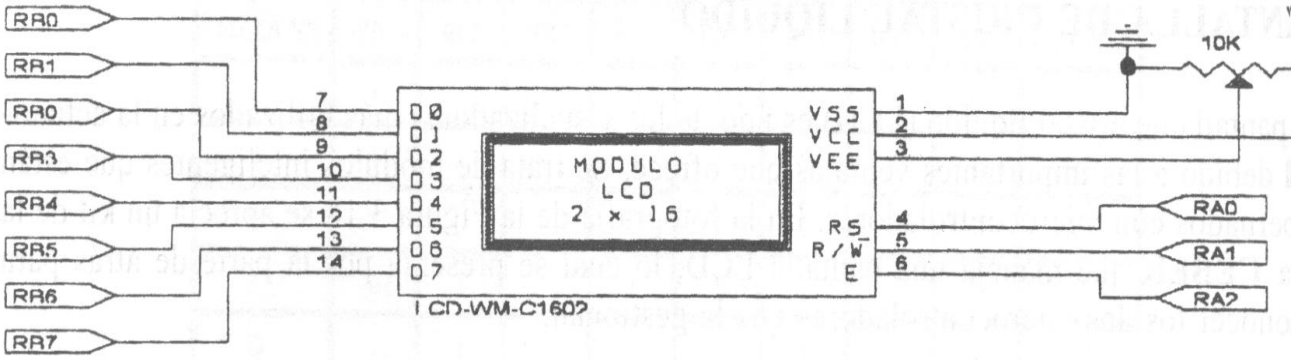


Figura 3.19. *Esquema típico de conexionado de una pantalla LCD de dos líneas de 16 caract cada una con las líneas de E/S de un PIC.*

EN EL INTERIOR DEL PROCESADOR



4.1. UNA ARQUITECTURA MODERNA Y REVOLUCIONARIA

Para lograr una compactación de código óptima y una velocidad superior a la de sus competidores los microcontroladores PIC incorporan en su procesador tres de las características más avanzadas en los grandes computadores:

- *Procesador tipo RISC*
- *Procesador segmentado*
- *Arquitectura HARVARD*

Con la incorporación de estos recursos los PIC son capaces de ejecutar en un ciclo de instrucción todas las instrucciones, excepto las de salto, que tardan el doble. Una condición imprescindible es la simetría y ortogonalidad en el formato de las instrucciones, que en el caso de los PIC de la gama media tienen una longitud de 14 bits. De esta forma se consigue una compactación en el código del programa para un PIC16X84 2,24 veces superior al de un 68HC05, funcionando a la misma frecuencia.

El juego de instrucciones se reduce a 35 y sus modos de direccionado se han simplificado al máximo.

Con la estructura segmentada se pueden realizar simultáneamente las dos fases en que se descompone cada instrucción. Al mismo tiempo que se está desarrollando la fase de ejecución de una instrucción se realiza la fase de búsqueda de la siguiente.

La arquitectura HARVARD es uno de los pilares en los que se sustenta la organización de los PIC. Gracias a ella se puede acceder de forma simultánea e independiente a la memoria de datos y a la memoria de instrucciones.

El aislamiento y diferenciación de los dos tipos de memoria permite que cada uno tenga la longitud y el tamaño más adecuados. De esta forma en el PIC16C84 la longitud de los datos es de un byte, mientras que la de las instrucciones es de 14 bits. (Véase Figura 4.1.)

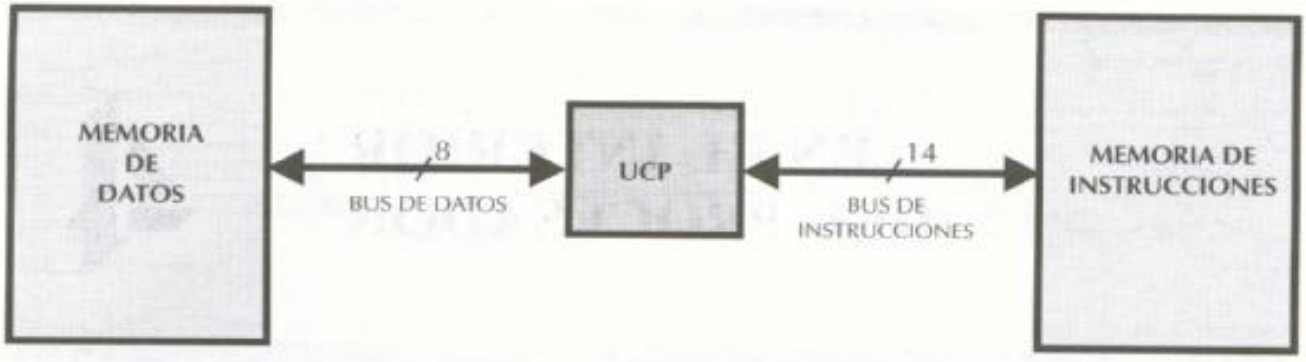


Figura 4.1. *La arquitectura HARVARD aísla la memoria de datos y la de instrucciones, permitiendo que la longitud de las palabras de cada una sea la más adecuada.*

Otra característica relevante de los PIC es el manejo intensivo del Banco de Registros, los cuales participan de una manera muy activa en la ejecución de las instrucciones. Como se muestra en la Figura 4.2, la ALU efectúa sus operaciones lógico-aritméticas con dos operandos, uno que recibe desde el registro W (*Work*), que hace las veces de «Acumulador» en los microprocesadores convencionales, y otro que puede provenir de cualquier registro o del propio código de la instrucción. El resultado de la operación puede almacenarse en cualquier registro o en W. Esta funcionalidad da un carácter completamente ortogonal a las instrucciones que pueden utilizar cualquier registro como operando fuente y destino. La memoria de datos RAM implementa en sus posiciones los registros específicos y los de propósito general.

La arquitectura interna del PIC16C84 se presenta en la Figura 4.3 y consta de siete bloques fundamentales.

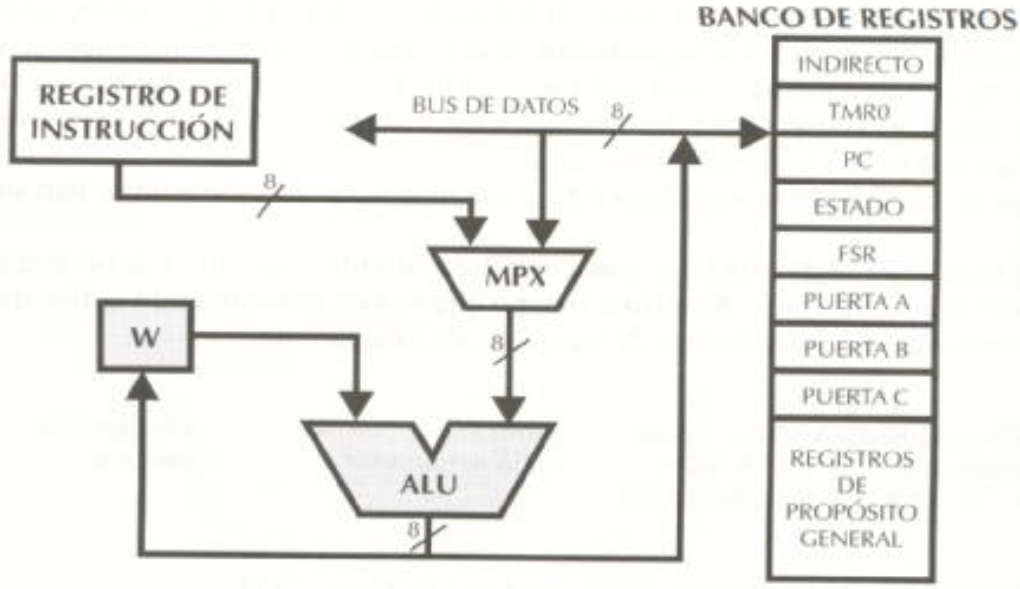


Figura 4.2. *En el esquema se muestra los diferentes elementos que proporcionan los operandos a la ALU. El resultado se puede cargar en cualquier registro o en W.*

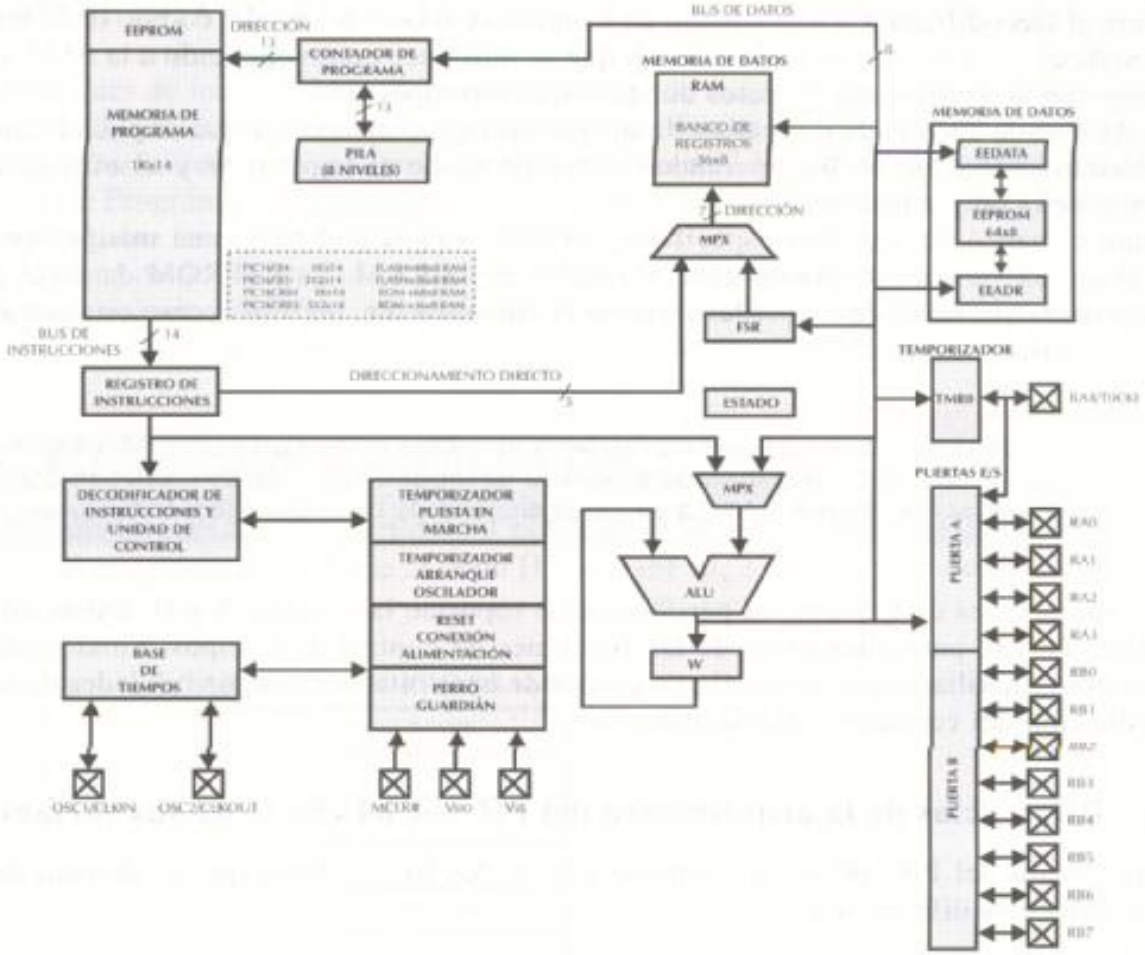


Figura 4.3. *Arquitectura interna del PIC16C84.*

1.° *Memoria de programa EEPROM de 1 K x 14 bits.*
2.° *Memoria de datos formada por dos áreas. Una RAM donde se alojan 22 registros de propósito específico (SFR) y 36 de propósito general (GPR), y otra del tipo EEPROM de 64 bytes.*
3.° *Camino de datos con una ALU de 8 bits y un registro de trabajo W del que normalmente recibe un operando y envía el resultado. El otro operando puede provenir del bus de datos o del propio código de la instrucción (literal).*
4.° *Diversos recursos conectados al bus de datos, tales como Puertas de Entrada/Salida, Temporizador TMR0, etc.*
5.° *Base de tiempos y circuitos auxiliares.*
6.° *Direccionamiento de la memoria de programa en base al Contador de Programa ligado a una Pila de 8 niveles de profundidad.*
7.° *Direccionado directo e indirecto de la memoria RAM.*

Para analizar de forma global el funcionamiento del procesador nos vamos a centrar en la ejecución de una instrucción. Todo comienza con la *fase de búsqueda*, que la inicia el Contador de Programa facilitando la dirección de la memoria de instrucciones donde se ubica. Su código binario de 14 bits se lee y se carga en el Registro de Instrucciones, desde donde se

transfiere al Decodificador y a la Unidad de Control. A veces, dentro del código de la instrucción, existe el valor de un operando (literal) que se introduce como operando a la ALU, o bien una dirección de la memoria de datos donde reside otro operando.

La ALU es la encargada de realizar la operación lógico-aritmética que implica la instrucción decodificada. Uno de los operandos lo recibe desde el registro W y el otro desde un registro o de la propia instrucción.

Tanto el banco de registros específicos, en el que cada uno tiene una misión concreta, como el de registros de propósito general residen en la RAM. La EEPROM de datos puede contener datos que no se desee perder al quitar la alimentación, pero su acceso está controlado con unos registros especiales.

El PIC16C84 tiene una zona de memoria de datos tipo RAM y otra tipo EEPROM. La primera tiene una capacidad de 22 registros de propósito especial (SFR) y 36 de propósito general (GPR), todos ellos de tamaño byte. La segunda dispone de una capacidad de 64 bytes.

Las operaciones de E/S con los periféricos las soportan las Puertas A y B. Existe un Temporizador, TMR0, para encargarse de las funciones de control de tiempos. Finalmente, hay unos circuitos auxiliares que dotan al procesador de unas interesantes posibilidades de seguridad, reducción del consumo y reinicialización.

4.1.1. Diferencias de la arquitectura del PIC16C84 con la de sus variantes

La arquitectura del PIC16C84 se mantiene para todos los modelos que se derivan de ella, con las siguientes diferencias:

PIC16F84: La memoria de programa es de 1K x 14 bits, pero de tipo Flash. La memoria de datos RAM tiene 68 registros de tamaño byte de propósito general, en lugar de 36.

PIC16CR84: La memoria de programa es de 1K x 14 bits tipo ROM y la de datos tiene iguales características que el PIC16F84.

PIC16F83: La memoria de programa es de 512 palabras de 14 bits y la RAM de datos tiene 36 bytes de registros de propósito general.

PIC16CR83: Igual que el PIC16F83, pero la memoria de instrucciones es de tipo ROM, o sea, sólo grabable en fábrica y utilizada en grandes series.

EJEMPLO

Indicar las diferencias entre el PIC16C84 y el PIC16F84 en cuanto a su estructura interna.

SOLUCIÓN

1.ª *La memoria de programa tiene una capacidad de 1K x 14 bits en ambos modelos, pero en el PIC16C84 es de tipo EEPROM y en el otro es Flash.*

2.ª *La diferencia en la memoria de datos sólo afecta a la zona de registros de propósito general de tipo RAM, que en el PIC16C84 consta de 36 bytes y en el PIC16F84 de 68 bytes.*

4.2. MEMORIA DE PROGRAMA

La arquitectura de los PIC de la gama media admite un mapa de memoria de programa capaz de contener 8.192 instrucciones de 14 bits cada una. Este mapa se divide en páginas de 2.048 posiciones. Para direccionar 8 K posiciones se necesitan 13 bits, que es la longitud que tiene el Contador de Programa. Sin embargo, el PIC16C84 sólo tiene implementadas 1 K posiciones, por lo que ignora los 3 bits de más peso del PC.

En la gama media, la verdadera aportación del PIC16C84 es la utilización de una memoria de programa del tipo EEPROM, capaz de ser escrita y borrada eléctricamente. Lo mismo sucede con el PIC16F84, cuya memoria de instrucciones es de tipo Flash.

4.2.1. El Contador de Programa y la Pila

Tal como se presenta en la Figura 4.4, el rango de direcciones que cubre el PIC16X84 en su memoria de programa llega desde la 0000 H a la 03FF H, o sea, un total de 1.024 posiciones.

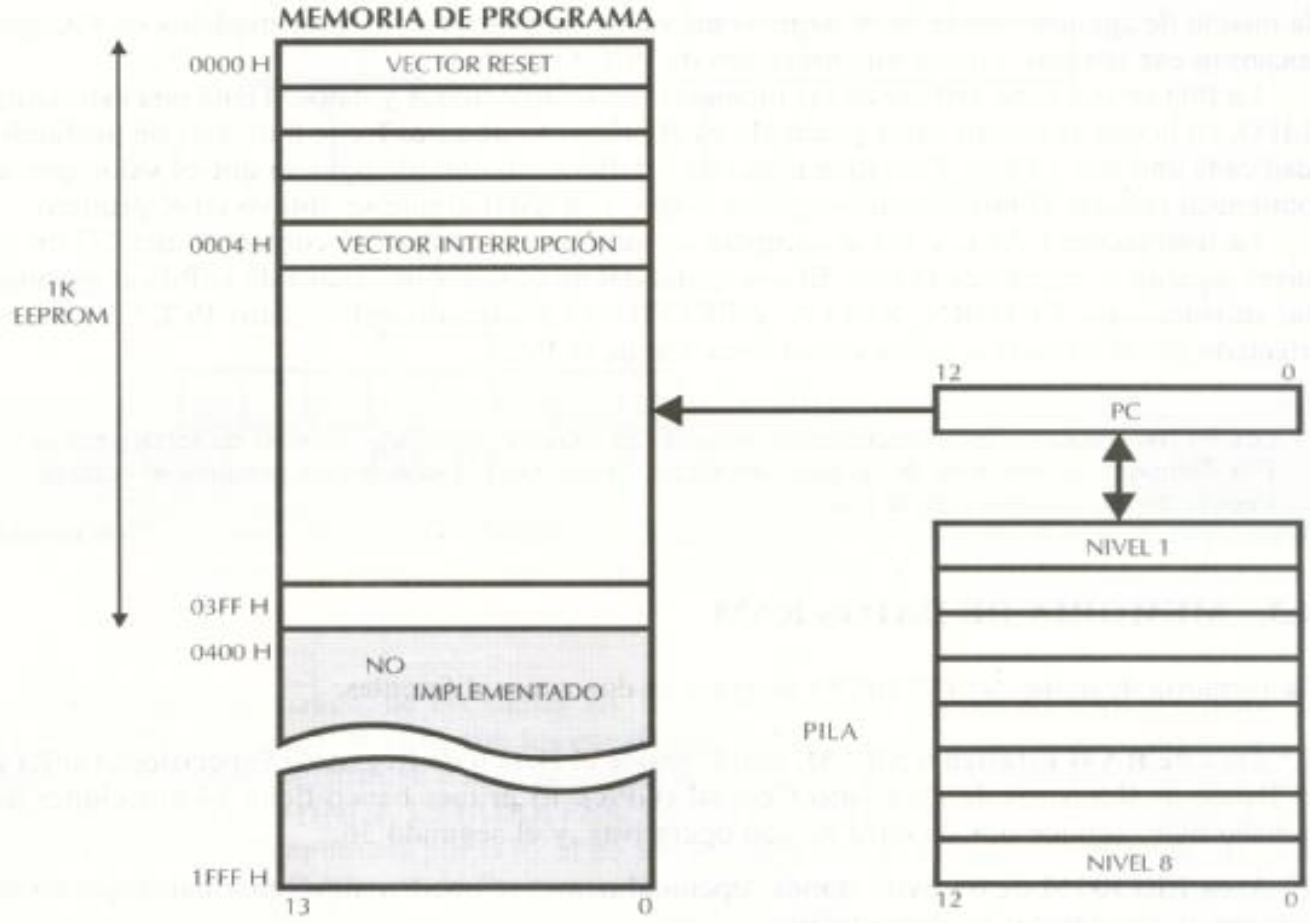


Figura 4.4. *Al tener implementado sólo 1 K posiciones la memoria de programa del PIC16X84, se ignoran los 3 bits de más peso del PC. La dirección 0000 H está reservada para el vector de Reset y la 0004 H para el vector de Interrupción.*

En el PC se ignoran los 3 bits de más peso, de forma que apuntar a la dirección 33 H es lo mismo que hacerlo a la 433 H, 833 H, C33 H, 1033 H, 1433 H o a la 1C33 H.

Los PIC16X84 tienen 1K palabras de 14 bits en la memoria de programa y aunque el PC dispone de 13 bits, en el direccionamiento de la misma sólo emplea los 10 de menos peso.

Al igual que todos los registros específicos que controlan la actividad del procesador, el Contador de Programa está implementado sobre un par de posiciones de la memoria RAM. Cuando se escribe el Contador de Programa como resultado de una operación de la ALU, los 8 bits de menos peso del PC residen en el registro PCL, que ocupa, repetido, la posición 2 de los dos bancos de la memoria de datos. Los bits de más peso, PC<12 :8>, residen en los 5 bits de menos peso del registro PCLATH, que ocupa la posición 0A H de los dos bancos de la memoria RAM.

En las instrucciones GOTO y CALL de la gama media los 11 bits de menos peso del PC provienen del código de la instrucción y los otros dos de los bits PCLATH <4 :3>. (Véase Figura 4.5.)

Con los 11 bits que se cargan en el PC desde el código de las instrucciones GOTO y CALL, se puede direccionar una página de 2 K de la memoria. Los bits restantes PC<12 :11> tienen la misión de apuntar una de las 4 páginas del mapa de memoria y, en los modelos de PIC que alcanzan ese tamaño, dichos bits proceden de PCLATH<4 :3>.

La Pila es una zona aislada de las memorias de instrucciones y datos. Tiene una estructura LIFO, en la que el último valor guardado es el primero que sale. Tiene 8 niveles de profundidad cada uno con 13 bits. Funciona como un «buffer» circular, de manera que el valor que se obtiene al realizar el noveno «desempilado» (*pop*) es igual al que se obtuvo en el primero.

La instrucción CALL y las interrupciones originan la carga del contenido del PC en el nivel superior o «cima» de la Pila. El contenido del nivel superior se saca de la Pila al ejecutar las instrucciones RETURN, RETLW y RETFIE. El contenido del registro PCLATH no es afectado por la entrada o salida de información de la Pila.

Los PIC no disponen de instrucciones específicas (*push* y *pop*) para manejar directamente la Pila. Tampoco se dispone de ningún señalizador que indique cuándo se produce el rebosamiento de los 8 niveles de la Pila.

4.3. MEMORIA DE DATOS RAM

La memoria de datos del PIC16C84 dispone de dos zonas diferentes:

1.ª **Área de RAM estática o SRAM**, donde reside el Banco de Registros Específicos (SFR) y el Banco de Registros de Propósito General (GPR). El primer banco tiene 24 posiciones de tamaño byte, aunque dos de ellas no son operativas, y el segundo 36.

2.ª **Área EEPROM** de 64 bytes donde, opcionalmente, se pueden almacenar datos que no se pierden al desconectar la alimentación.

La zona de memoria RAM se halla dividida en dos bancos (banco 0 y banco 1) de 128 bytes cada uno. En el PIC16C84 sólo se hallan implementadas físicamente las 48 primeras

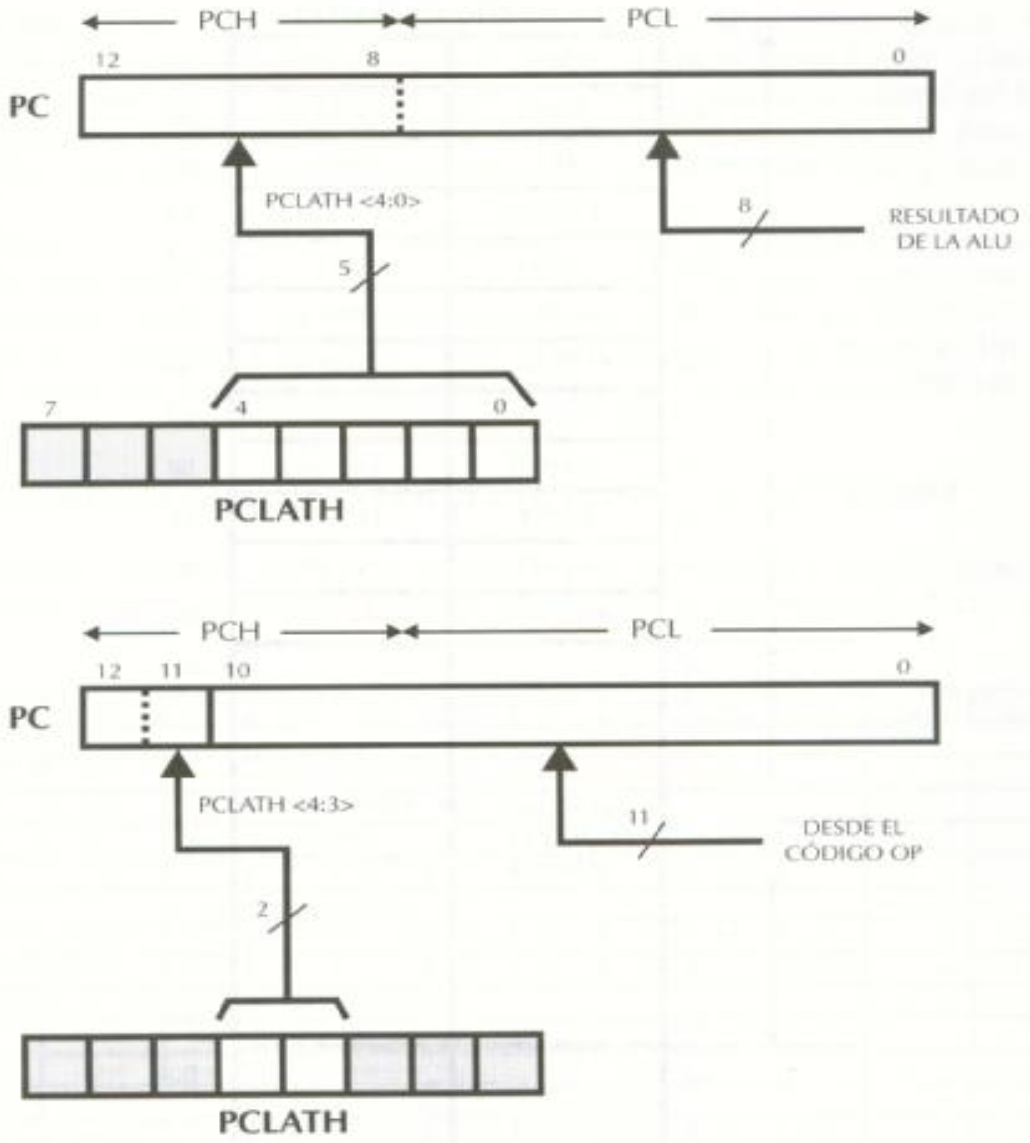


Figura 4.5. *En la figura superior se muestra cómo se carga el PC cuando una instrucción deposita en él el resultado que se obtiene de la ALU. Abajo se indica la carga del PC en las instrucciones GOTO y CALL.*

posiciones de cada banco, de las cuales las 12 primeras están reservadas a los Registros de Propósito Específico (SFR), que son los encargados del control del procesador y sus recursos. Algunos de dichos registros se hallan repetidos en la misma dirección de los dos bancos, para simplificar su acceso (INDF, ESTADO, FSR, PCLATH e INTCON). La posición apuntada por la dirección 7 H y la apuntada por la 87 H no son operativas. Los 36 registros restantes de cada banco se destinan a Registros de Propósito General y en realidad sólo son operativos los 36 del banco 0 porque los del banco 1 se mapean sobre el banco 0, es decir, cuando se apunta a un registro general del banco 1, se accede al mismo del banco 0. (Véase Figura 4.6.)

Para seleccionar el banco a acceder hay que manipular el bit 5 (RP0) del registro ESTADO. Si RP0 = 1 se accede al banco 1 y si RP0 = 0 se accede al banco 0. Tras un Reset se accede

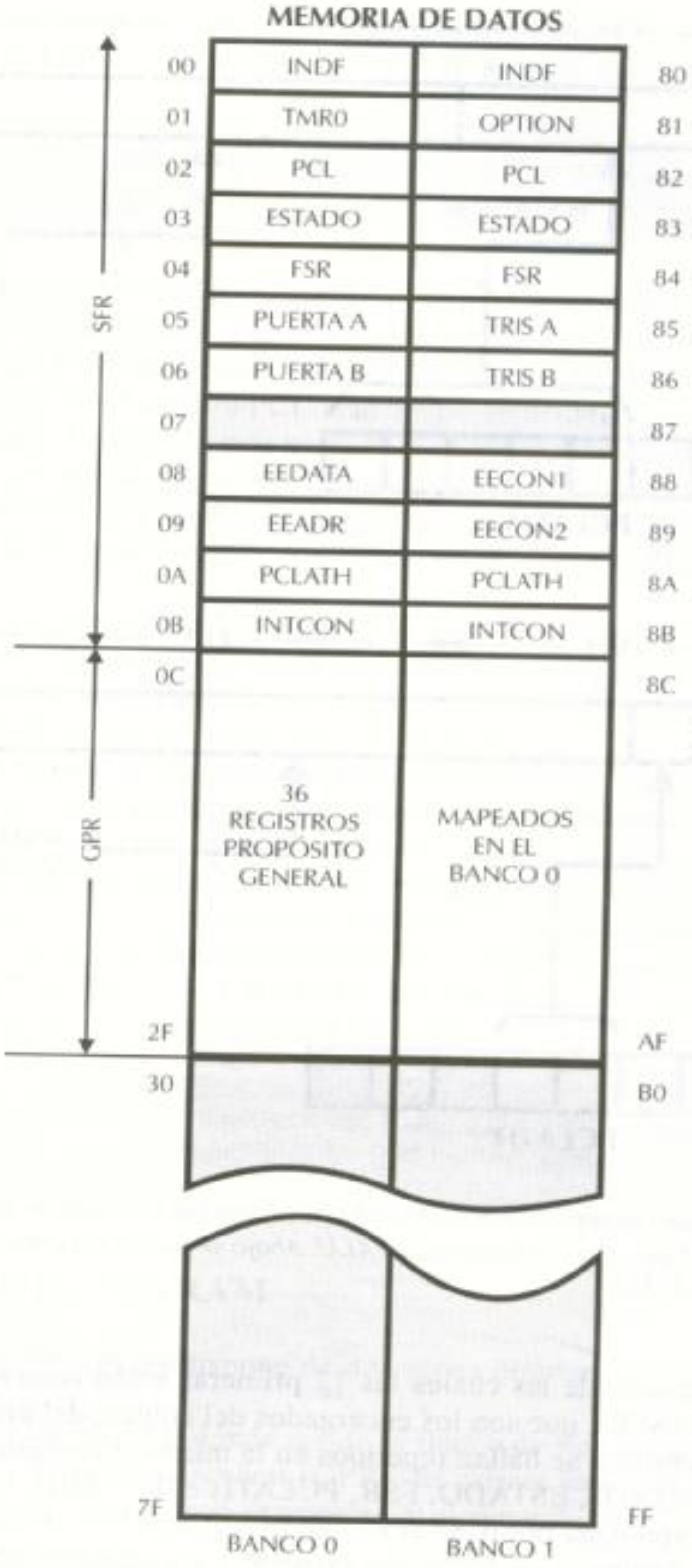


Figura 4.6. *La memoria RAM de datos se divide en dos bancos de 128 bytes cada uno, aunque sólo los 48 primeros de cada banco están implementados en el PIC16C84. Las 12 primeras posiciones de cada banco se reservan a los Registros de Propósito Especial (SFR) y las 36 restantes a los de Propósito General (GPR).*

automáticamente al banco 0. Para seleccionar un registro de propósito general no hay que tener en cuenta el estado del bit RP0, porque al estar mapeado el banco 1 sobre el banco 0, cualquier direccionamiento de un registro del banco 1 corresponde al homólogo del banco 0. En el direccionamiento directo a los registros GPR se ignora el bit de más peso, que identifica el banco y sus direcciones están comprendidas entre el valor 0x0c y 0x2f en hexadecimal.

Los registros SFR se clasifican en dos grupos. En uno se incluyen aquellos que controlan el núcleo del microcontrolador (ESTADO, OPTION, INTCON, etc.) y en el otro los que caracterizan la operatividad de los recursos auxiliares y periféricos. La Figura 4.7 presenta la estructura de estos registros y los valores que toman después de un Reset. El PIC16F83 y el PIC16CR83 tienen una memoria RAM con la misma organización que el PIC16C84.

4.3.1. Memoria de datos RAM en el PIC16F84 y el PIC16CR84

Estos dos modelos disponen como peculiaridad los 68 registros de propósito general que ocupan las direcciones comprendidas entre la 0x0c y la 0x4f en hexadecimal. (Véase Figura 4.8.)

DIR.	NOMBRE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	VALOR TRAS EL RESET	VALOR OTROS RESET
BANCO 0											
00 h	INDF	Usa el contenido de FSR para direccionar la memoria (no es un registro físico)								---- ----	---- ----
01 h	TMR0	Reloj/Contador en tiempo real de 8 bits								xxxx xxxx	uuuu uuuu
02 h	PCL	Byte de menos peso del PC								0000 0000	0000 0000
03 h	ESTADO	IRP	RP1	RP0	TO#	PD#	Z	DC	C	0001 1xxx	000q quuu
04 h	FSR	Direccionamiento indirecto con INDF								xxxx xxxx	uuuu uuuu
05 h	PUERTA A	---	---	---	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	---u uuuu
06 h	PUERTA B	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	uuuu uuuu
07 h		No implementado, se lee como '0'								---- ----	---- ----
08 h	EEDATA	Registro de datos de EEPROM								xxxx xxxx	uuuu uuuu
09 h	EEADR	Registro de direcciones de EEPROM								xxxx xxxx	uuuu uuuu
0A h	PCLATH	---	---	---	Se escribe en los 5 bits de PCH					---0 0000	---0 0000
0B h	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u
BANCO 1											
80 h	INDF	Usa el contenido de FSR para direccionar la memoria (no es un registro físico)								---- ----	---- ----
81 h	OPTION	RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111
82 h	PCL	Byte de menos peso del PC								0000 0000	0000 0000
83 h	ESTADO	IRP	RP1	RP0	TO#	PD#	Z	DC	C	0001 1xxx	000q quuu
84 h	FSR	Direccionamiento indirecto con INDF								xxxx xxxx	uuuu uuuu
85 h	TRISA	---	---	---	Configuración Puerta A					---1 1111	---1 1111
86 h	TRISB	Configuración Puerta B								1111 1111	1111 1111
87 h		No implementado, se lee como '0'								---- ----	---- ----
88 h	EECON1	---	---	---	EEIF	WRERR	WREN	WR	RD	---0 x000	---- q000
89 h	EECON2	Registro de Control EEPROM (no es un registro físico)								---- ----	---- ----
0A h	PCLATH	---	---	---	Configuración Puerta A					---0 0000	---0 0000
0B h	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	0000 000u

Figura 4.7. *Especificaciones de los registros SFR de los bancos de la memoria de datos RAM. La abreviatura x significa desconocido; u: no cambia; -: no implementado, se lee como 0, y q: valor que depende de la condición.*

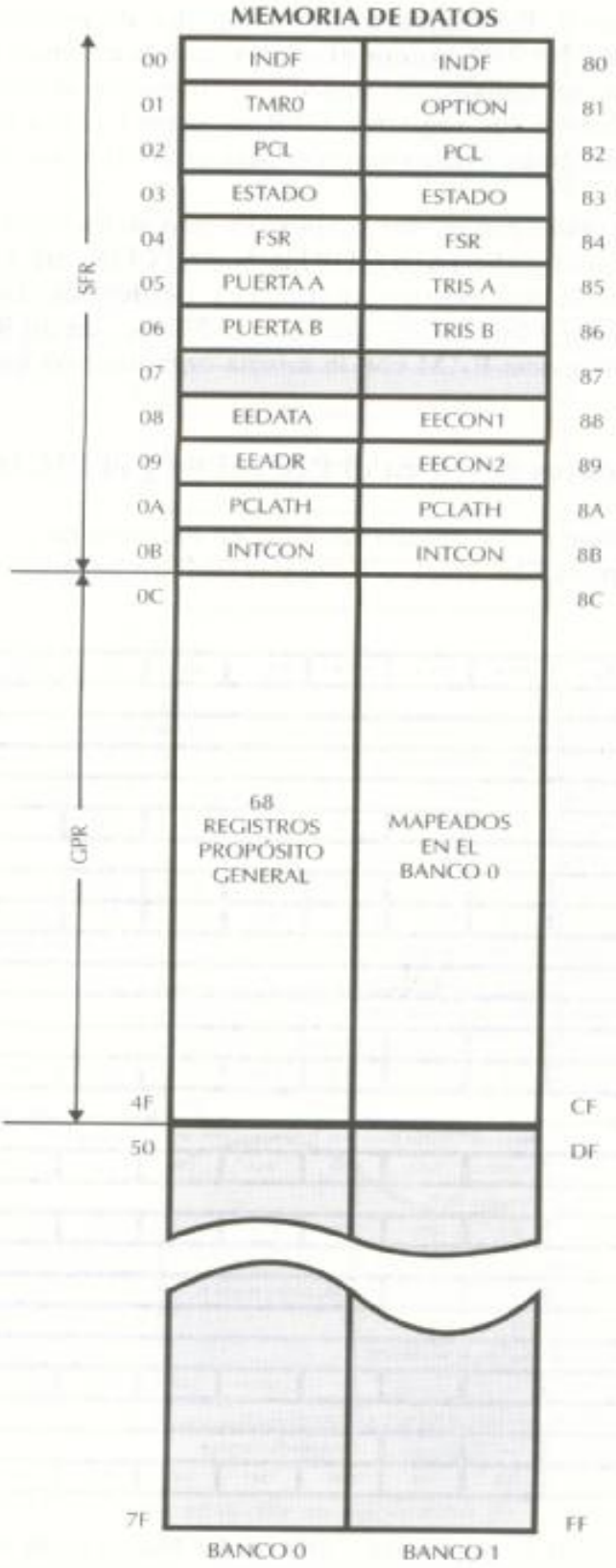


Figura 4.8. *Organización interna de la memoria de datos RAM en el PIC16F84 y el PIC16CR84.*

4.3.2. Direccionamiento de la memoria de datos

En los PIC de la gama media la memoria de datos está organizada para alojar un máximo de 4 bancos de 128 bytes cada uno. El PIC16C84 sólo tiene implementados los 48 primeros bytes de los bancos 0 y 1. En el resto de los PIC de esta familia se destinan dos bits del registro ESTADO (RP0 y RP1) para determinar el banco y otros siete para elegir una de las 128 posiciones del banco seleccionado. (Figura 4.9.)

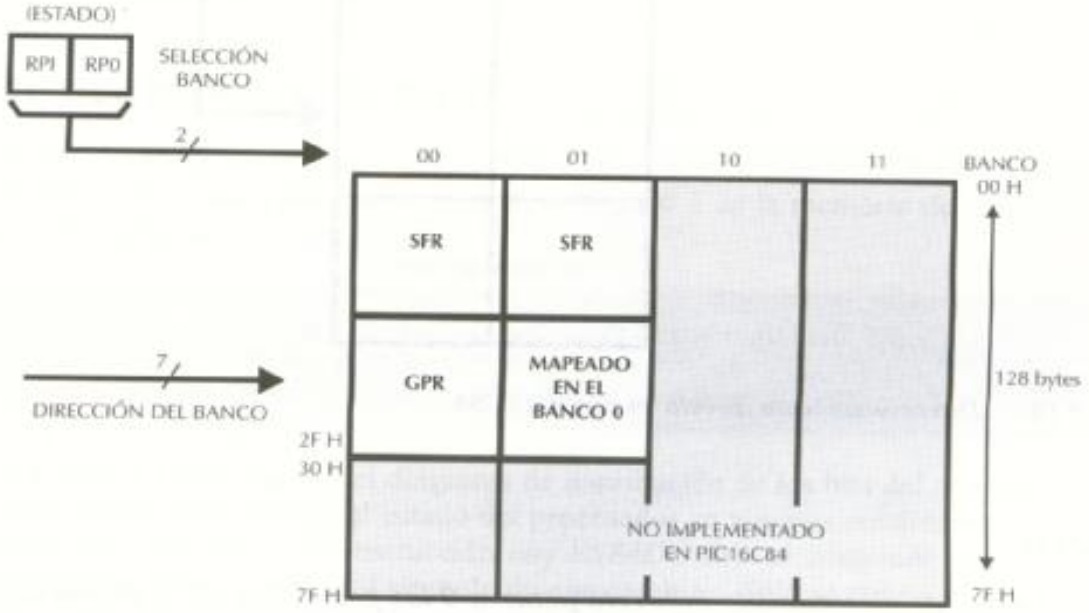


Figura 4.9. *Los PIC de la gama media tienen organizada la memoria de datos en 4 bancos de 128 posiciones cada uno. La zona sombreada no se halla implementada en el PIC16C84.*

Direccionamiento Directo

El operando que utiliza la instrucción en curso se referencia mediante su dirección, que viene incluida en el código OP de la misma, concretamente en los 7 bits de menos peso. El banco a acceder lo determinan los bits RP0 y RP1 del registro ESTADO. En el caso del PIC16C84 sólo se usa el bit RP0 al tener implementados únicamente dos bancos. (Véase Figura 4.10.)

Direccionamiento Indirecto

Este modo de direccionado se usa cuando en una instrucción se utiliza como operando el registro INDF, que ocupa la dirección 0 de ambos bancos. En realidad el registro INDF no está implementado físicamente y cuando se le hace referencia, se accede a la dirección de un banco especificada con los 7 bits de menos peso del registro FSR. El bit de más peso de FSR junto al bit IRP del registro ESTADO se encargan de seleccionar el banco a acceder, mientras que los 7 bits de menos peso apuntan a la posición. Como sólo hay dos bancos en el PIC16C84 en este modo de direccionamiento, el bit IRP = 0 siempre. (Véase Figura 4.11.)

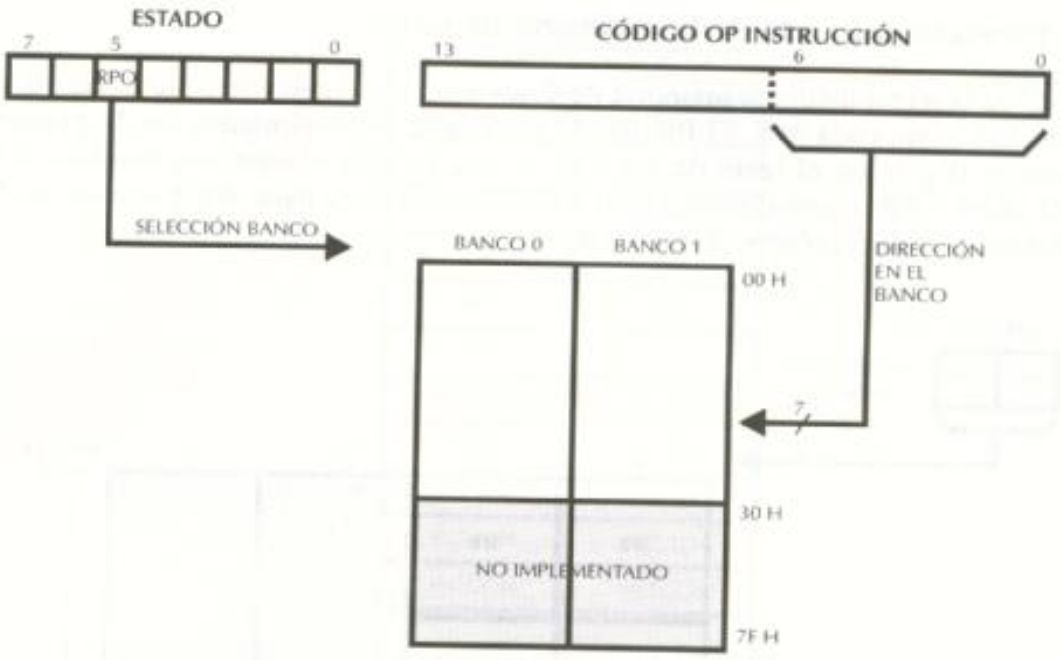


Figura 4.10. *Direccionamiento directo en el PIC16C84.*

EJEMPLO

Usando el modo de direccionado indirecto poner a 0 las posiciones de la memoria de datos comprendidas entre la dirección 0x0c y 0x27 del banco 0 en un PIC16C84.

SOLUCIÓN

En el modo de direccionado indirecto actúa como puntero el contenido del registro FSR, que en este ejercicio se irá incrementando desde 0x0c hasta el valor 0x28 H, en el que su bit 3 pasa a valer 1.

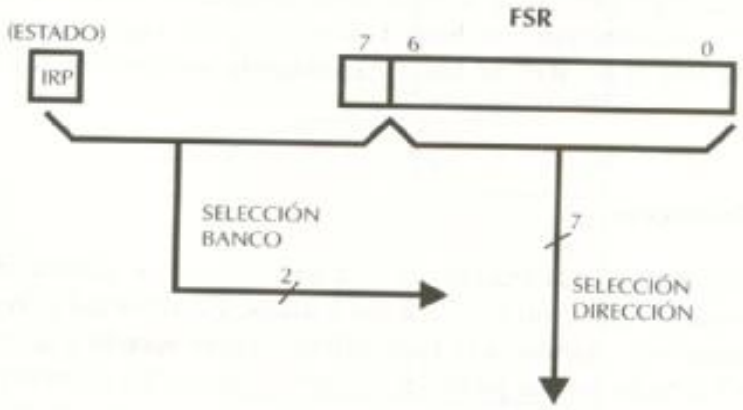


Figura 4.11. *Selección del banco y la posición en el modo de direccionamiento indirecto. Como el PIC16C84 sólo tiene dos bancos, IRP = 0 siempre.*

```
        movlw   0x0c    ; W toma el valor inicial del puntero
        movwf   FSR     ; FSR se carga con el valor de W
                        ; Se borra la dirección que apunta FSR
bucle   incf    FSR     ; Se incrementa FSR una unidad
        btfss   FSR,3   ; Si el bit 3 de FSR vale 1 se "brinca" la
                        ; siguiente instrucción ("skip")
        goto    bucle   ; Se salta a "bucle" porque vale 0 el
                        ; bit 3 de FSR
.....continúa el programa.....
```

4.4. EL REGISTRO DE ESTADO

Hasta ahora ESTADO es el registro más usado y es el momento de describirle en su totalidad. Ocupa la dirección 3 tanto del banco 0 como del banco 1 de la memoria de datos RAM. Sus bits tienen tres misiones distintas .

1.ª Se encargan de avisar las incidencias del resultado de la ALU (C, DC y Z).
2.ª Indican el estado de Reset (TO# y PD#)
3.ª Seleccionan el banco a acceder en la memoria de datos (IRP, RP0 y RP1).

En la Figura 4.12 se muestra el diagrama de distribución de los bits del registro ESTADO.

Los bits TO# y PD# indican el estado del procesador en algunas condiciones y no se pueden escribir. Por este motivo la instrucción *clrf ESTADO* deja el contenido de dicho registro con el valor *000u u1uu*, siendo *u* el símbolo de «no cambia». Sólo se ponen a 0 los tres bits de más peso, el bit Z (cero) se pone a 1 y los restantes no alteran su valor.

A continuación se describe la misión de los bits del registro ESTADO.

C: Acarreo/llevada en el bit de más peso

1: Cuando este señalizador vale 1 indica que se ha producido acarreo en el bit de más peso del resultado al ejecutar las instrucciones *addwf* y *addlw*.
0: No se ha producido acarreo.

C también actúa como señalizador de «llevada» en el caso de la instrucción de resta, como *subwf* y *sublw*. En este caso la correspondencia es inversa (si vale 1 no hay llevada y si vale 0 sí).

DC: Acarreo/llevada en el 4.º bit

Igual significado que C pero refiriéndose al 4.º bit. De interés en operaciones en BCD.



Figura 4.12. *Estructura interna del registro ESTADO. Los bits 3 y 4 sólo se pueden leer, mientras los demás también se pueden escribir.*

Z: Cero

1: El resultado de una instrucción lógico-aritmética ha sido 0.
0: El resultado de una instrucción lógica-aritmética no ha sido 0.

PD#: «Power Down»

1: Se pone automáticamente a 1 después de la conexión de la alimentación al microcontrolador o al ejecutar la instrucción *clrwdt*.
0: Se pone automáticamente a 0 mediante la ejecución de la instrucción *sleep*.

TO#: «Time Out»

1: Se pone a 1 después de la conexión de la alimentación o al ejecutarse las instrucciones *clrwdt* y *sleep*.
0: Se pone a 0 cuando se produce el desbordamiento del Perro Guardián (*Watchdog*).

RP1 - RP0: Selección de banco en direccionamiento directo

Como el PIC16X84 sólo tiene dos bancos únicamente emplea el bit RP0, de forma que cuando vale 1 se accede al banco 1 y cuando vale 0 se accede al banco 0. Después de un Reset RP0 = 0.

IRP: Selección del banco en direccionamiento indirecto

Este bit junto con el de más peso del registro FSR sirven para determinar el banco de la memoria de datos seleccionado. En el PIC16X84 al disponer de dos bancos no se usa este bit y debe programarse como 0.

PROGRAMAR PIC ES FÁCIL

EL CUARTO PROGRAMA: CONTANDO Y VISUALIZANDO

En este programa vamos a utilizar un tipo de instrucciones muy eficaces que se basan en un salto condicional. Se trata de un salto un poco especial, *sólo salta una instrucción, la siguiente a la del salto*. Nosotros le llamamos «brinco» (*skip*).

La instrucción *btfss ESTADO,4* explora el bit 4 del registro ESTADO y si vale 1 produce «brinco» (salta la siguiente instrucción). Si vale 0 se ejecuta la siguiente instrucción normalmente.

EJEMPLO

Dado el siguiente programa, indicar qué instrucción se ejecuta detrás de la primera en el caso que el bit 4 del registro ESTADO valga 1.

```
btfss   ESTADO,4  ; Explora el bit 4 del registro ESTADO
goto    VALE_0    ; Si el bit 4 vale 0 se ejecuta esta instrucción
goto    VALE_1    ; Si el bit 4 vale 1 se ejecuta esta instrucción
```

SOLUCIÓN

Se ejecuta la instrucción goto VALE_1

Dada su sencillez operativa se citan a continuación otras instrucciones que usaremos en este ejercicio.

La instrucción *clrf REGISTRO* borra o pone a 0 el contenido de REGISTRO. La instrucción *incf REGISTRO* incrementa en una unidad el contenido de REGISTRO. Su opuesta *decf, REGISTRO,* decrementa en una unidad el contenido de REGISTRO.

Finalmente, existen dos instrucciones similares a *addlw* y *addwf,* pero referidas a la resta, y son: *sublw* y *subwf.*

ENUNCIADO

Se desea confeccionar un programa para el PIC16C84 funcionando a 4 MHz, que comience poniendo a cero el contador CONTA. Después se incrementa de unidad en unidad hasta alcanzar el valor 0x5F h, momento en el que se detiene en un bucle infinito no operativo. El valor del contador en binario se visualizará con 8 diodos led conectados a las líneas de la Puerta B. Se pide:

1.º *Esquema del circuito electrónico.* (Figura 4.13.)
2.º *Organigrama.* (Figura 4.14.)
3.º *Programa en lenguaje Ensamblador MPASM.*

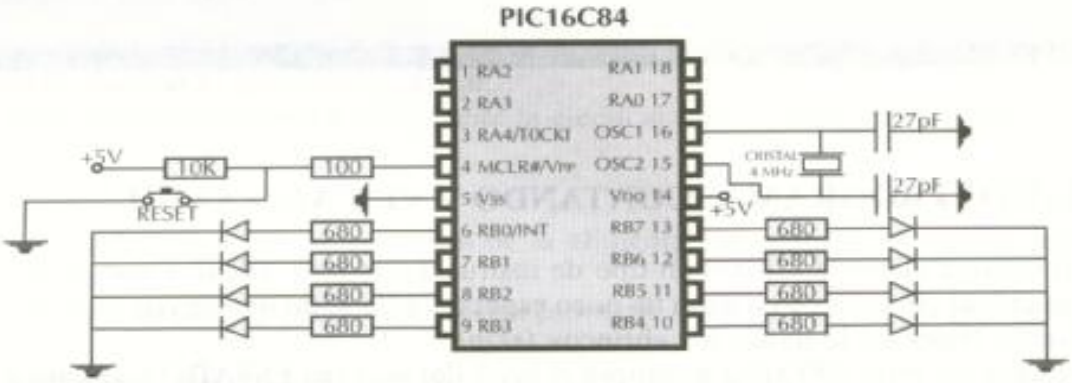


Figura 4.13. *Esquema de la circuitería auxiliar que rodea al PIC16C84 que visualiza el valor del contador en los 8 diodos led conectados a la Puerta B.*

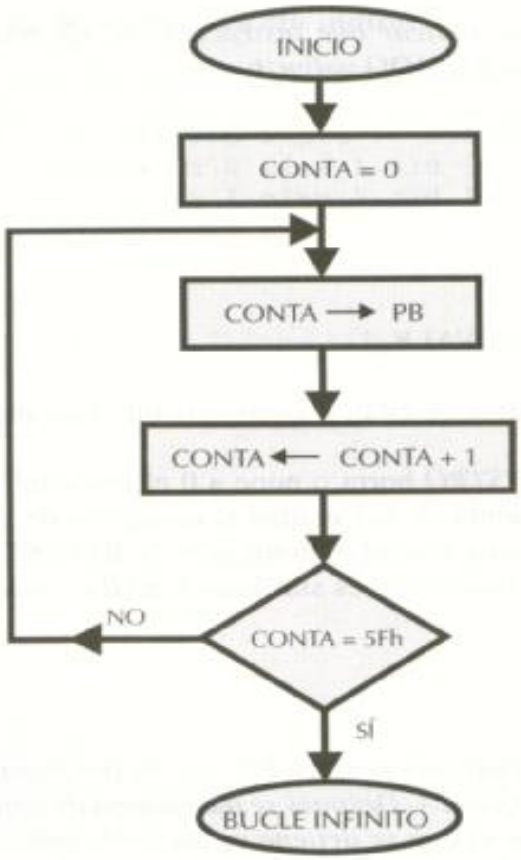


Figura 4.14. *Organigrama.*

PROGRAMA EN ENSAMBLADOR

CUARTO.ASM

```
        LIST    P=16C84
        RADIX   HEX

        W       EQU     0x00  ; Comienzo del campo de
                              ; etiquetas
        F       EQU     0x01
        PUERTAB EQU     0x06
        ESTADO  EQU     0x03
        CONTA   EQU     0x0C
        ORG     0             ; El programa comienza en la
        goto    inicio        ; dirección 0 y salta a la
        ORG     5             ; dirección 5 para sobrepasar;
                              ; el vector de interrupción

inicio  bsf     ESTADO,5      ; Selección del banco 1
        movlw   0x00          ; Se configura Puerta B
        movwf   PUERTAB       ; como salida
        bcf     ESTADO,5      ; Selección del banco 0

        clrf    CONTA         ; CONTA = 0
bucle1  incf    CONTA,F       ; CONTA + 1 → CONTA
        movf    CONTA,W       ; CONTA se carga en W
        movwf   PUERTAB       ; W se carga en el
                              ; registro de datos PB
        movlw   0x5f          ; W ← 5f
        subwf   CONTA,W       ; CONTA - W → W
        btfss   ESTADO,2      ; Explora Z y si vale 1 se
                              ; produce "brinco"
        goto    bucle1        ; Si Z = 0 se vuelve a
                              ; bucle1
bucle2  goto    bucle2        ; Si Z = 1 se produce un
                              ; bucle infinito
        end
```

PROBANDO CON EL SIMUPIC'84

Tal y como se ha hecho en los capítulos anteriores, editaremos, ensamblaremos y cargaremos en memoria el programa CUARTO.ASM. Si nos fijamos en el enunciado del ejercicio, nos damos cuenta de que en él se especifica la frecuencia de reloj a la que debe funcionar el PIC, en este caso 4 MHz. Para cambiar la frecuencia de reloj de la simulación con el **Simupic'84** se acude a la opción quinta, «Opciones de Simulación», «Parámetros de Simulación» y una vez

allí se introduce la frecuencia deseada en KHz. Por tanto, si deseamos una frecuencia de 4 MHz indicaremos 4000 KHz.

Una vez configurada la velocidad de reloj, se ejecuta el programa mediante la opción 6, fijándonos cómo efectivamente los valores de la PB representados en la ventana de visión externa se incrementan de uno en uno hasta alcanzar el valor 0x5f.

El programa de este capítulo termina en un bucle infinito, pero el lector se dará cuenta de que si deja el programa en ejecución durante el tiempo suficiente, éste volverá a su comienzo. Esto se debe al Watchdog o «Perro Guardián», que es un temporizador incorporado en el PIC que se encarga de provocar un Reset cada cierto tiempo. Esta temporización es configurable, desde un mínimo de 18 ms a un máximo de 2,3 segundos. En particular en el **Simpic'84** el timer del WDT se inicializa por defecto a 2,3 segundos. Pues bien, para evitar problemas, volveremos a la ventana de «Parámetros de Simulación» y pondremos «N» en el campo Watchdog S/N, con lo que desactivaremos el Watchdog o Perro Guardián.

LO MONTAMOS Y NO FUNCIONA

Si el lector, tal y como le aconsejamos, se ha atrevido a montar el diseño propuesto en este capítulo, es probable que se haya llevado una decepción al observar que en su montaje el valor indicado por los leds no se incrementa de uno en uno sino que llega a su valor máximo (0x5F) al instante. Esto es debido a que el tiempo en que se produce el incremento del contador y la visualización del nuevo valor es tan pequeño que no se puede distinguir a simple vista. En particular *bucle1* consta de 7 instrucciones, con una de salto, con lo que tardará en ejecutarse 8 ciclos de instrucción que suponen 8 µs, ya que a 4 MHz el ciclo de instrucción es de 1 µs.

PONIÉNDOLO A PUNTO

Para apreciar el incremento del contador hay que incluir en el programa un retardo entre cada cambio de valor. A pesar de que este retardo se podría realizar mediante la utilización del temporizador interno del PIC, hemos preferido no utilizarlo para facilitar la comprensión del programa.

Como siempre animamos al lector a que se atreva a realizar por sí mismo el programa, pero de todas formas, para comprobar su solución o si tiene algún problema, en el disquete que acompaña el libro se encuentra bajo el nombre de CUARTO2.ASM el programa que incorpora el retardo.

Bajo el nombre CUARTO2A.ASM, encontrará un programa que consigue un retardo mucho mayor, pero anidando 2 bucles.

También sería interesante preocuparse de grabar el PIC con la opción de WatchDog Timer desactivada, pues de lo contrario se producirá el fenómeno comentado en el apartado del **Simupic'84**, aunque lógicamente a mucha mayor velocidad. Es decir, el contador se incrementará y cada 2,3 segundos, que es el valor máximo de WDT y el que tiene por defecto, se provocará un Reset y el programa volverá a su comienzo.

PROGRAMAMOS EN C

Se muestra la versión en lenguaje C del ejercicio propuesto en este capítulo. Se ha usado el compilador de C **PCM** de la empresa **Custom Computer Services.**

```
/* CUARTO.C Programa que comienza poniendo a cero el */
/* contador CONTA. Después este contador se incrementa */
/* de unidad en unidad hasta alcanzar el valor 5F, momento */
/* en que entra en un bucle infinito no operativo */
```

```
#include <16c84.h>                          /* Tipo de PIC a usar *

#byte puertab=0x06
#byte conta=0x0C

void main(void)

{
set_tris_b(0x00);                           /* Configura la PB como */
                                            /* salida */

for (conta=0;conta<=0x5f;++conta)           /* Bucle que incrementa */
                                            /* CONTA de 0 a 0x5F */
    puertab=conta;
}
```

Éste es otro claro ejemplo de cómo simplifica la programación el uso de un lenguaje de alto nivel. Aunque la eficiencia del código obtenido es mucho mayor programando directamente en ensamblador.

LOS RECURSOS FUNDAMENTALES: TEMPORIZADORES, PUERTAS DE E/S Y EEPROM DE DATOS



5.1. EL PAPEL DE LOS TEMPORIZADORES

Los PIC disponen de un procesador rápido y potente. En él se incluye la memoria de programa, la de datos, la ALU, la Unidad de Control y algunos registros especiales. Si dentro de un microcontrolador únicamente existiese un procesador sólo se podrían ejecutar instrucciones lógico-aritméticas y de transferencia. Pero un microcontrolador es mucho más que un procesador, es un «computador integral», en el que además del procesador hay puertas de E/S para conectarse con periféricos, canales de comunicación, temporizadores para controlar tiempos, sistema de interrupciones capaz de detectar anomalías o sucesos especiales, sistemas de seguridad, modo de funcionamiento con bajo consumo y un largo etcétera de recursos que configuran la potencia integral de un computador.

Los PIC contienen todos los recursos posibles, aunque su fabricante no les incluye en todos los modelos, sino que los alterna para poderse ajustar óptimamente a las necesidades de cada diseño.

Los PIC16X8X contienen pocos recursos en comparación a otros familiares, pero poseen los suficientes para resolver gran parte de las aplicaciones típicas de los microcontroladores. Sobre todo disponen de una memoria de programa tipo EEPROM (PIC16C84) o Flash (PIC16F8X), que les posibilita regrabar el programa hasta su total puesta a punto.

En este capitulo se describen los recursos más importantes de los PIC16X8X, que son:

— *Temporizador/Contador TMR0*
— *Perro Guardián («Watchdog»)*
— *Puertas de E/S*
— *Memoria EEPROM de datos*

5.2. TEMPORIZADOR/CONTADOR TMR0

Una de las labores más habituales en los programas de control de dispositivos suele ser determinar intervalos concretos de tiempo, y recibe el nombre de **temporizador** (*timer*) el ele-

mento encargado de realizar esta función. También suele ser frecuente contar los impulsos que se producen en el exterior del sistema, y el elemento destinado a este fin se denomina contador.

Si las labores del temporizador o contador las asignamos al programa principal robarían mucho tiempo al procesador en detrimento de actividades más importantes. Por este motivo se diseñan recursos específicamente orientados a estas misiones.

Los PIC16X8X poseen un temporizador/contador de 8 bits, llamado TMR0, que actúa de dos maneras diferentes:

1.ª Como contador de sucesos, que están representados por los impulsos que se aplican a la patita RA4/TOCKI. Al llegar al valor FF H se desborda el contador y, con el siguiente impulso, pasa a 00 H, advirtiendo esta circunstancia activando un señalizador y/o provocando una interrupción.

2.ª Como temporizador, cuando se carga en el registro que implementa al recurso un valor inicial se incrementa con cada ciclo de instrucción (Fosc/4) hasta que se desborda, o sea, pasa de FF H a 00 H y avisa poniendo a 1 un bit señalizador y/o provocando una interrupción. (Figura 5.1.)

Para que el TMR0 funcione como contador de impulsos aplicados a la patita T0CKI hay que poner a 1 el bit T0CS, que es el que ocupa la posición 5 del registro OPTION. En esta situación, el registro TMR0, que es el ubicado en la dirección 1 del banco 0 de la memoria de datos, se incrementa con cada flanco activo aplicado en la patita T0CKI. El tipo de flanco activo se elige programando el bit T0SE, que es el que ocupa la posición 4 del registro OPTION. Si T0SE = 1, el flanco activo es el descendente, y si T0SE = 0, es el ascendente. Cuando se desea que TMR0 funcione como temporizador el bit T0CS = 0.

En realidad, los PIC16X8X y los de la gama baja disponen de dos temporizadores, el TMR0 y el Perro Guardián (*Watchdog*). El primero actúa como principal y sobre él recae el control de tiempos y el contaje de impulsos. El otro vigila que el programa no se «cuelgue», y para ello cada cierto tiempo comprueba si el programa se está ejecutando normalmente. En caso contrario, si el control está detenido en un bucle infinito a la espera de algún acontecimiento que no se produce, el Perro Guardián «ladra», lo que se traduce en un Reset que reinicializa todo el sistema.

A menudo el TMR0 y el Perro Guardián precisan controlar largos intervalos de tiempo y necesitan aumentar la duración de los impulsos de reloj que les incrementa. Para cubrir este



Figura 5.1. *Esquema simplificado de un temporizador/contador. Como temporizador se carga con un valor inicial que se va incrementando hasta el desbordamiento. Como contador va incrementando su valor con cada impulso que se le aplica. Al alcanzar el máximo valor binario se «desborda» y pasa a cero, circunstancia que indica un señalizador.*

requisito se dispone de un circuito programable denominado Divisor de frecuencia, que divide la frecuencia utilizada por diversos rangos.

Para programar el comportamiento del TMR0, el Perro Guardián (WDT) y el Divisor de frecuencia se utilizan algunos bits del registro OPTION y de la Palabra de Configuración, que se explicará más adelante. En la Figura 5.2 se proporciona un esquema simplificado de la arquitectura del circuito de control de tiempos usado en los PIC16X8X.

El Divisor de frecuencia puede usarse con el TMR0 o con el WDT. Con el TMR0 actúa como *Pre-divisor*, es decir, los impulsos pasan primero por el Divisor y luego se aplican al TMR0, una vez aumentada su duración. Con el Perro Guardián actúa después, realizando la función de *Post-divisor*. Los impulsos, que divide por un rango el Divisor de frecuencia, pueden provenir de la señal de reloj interna (Fosc/4) o de los que se aplican a la patita T0CKI.

El TMR0 se comporta como un registro de propósito especial (SFR) ubicado en la dirección 1 del banco 0 de la memoria de datos. En igual dirección, pero en el banco 1, se halla el registro OPTION.

TMR0 puede ser leído y escrito en cualquier momento al estar conectado al bus de datos. Funciona como un contador ascendente de 8 bits. Cuando funciona como temporizador conviene cargarle con el valor de los impulsos que se quiere temporizar, pero expresados en com-

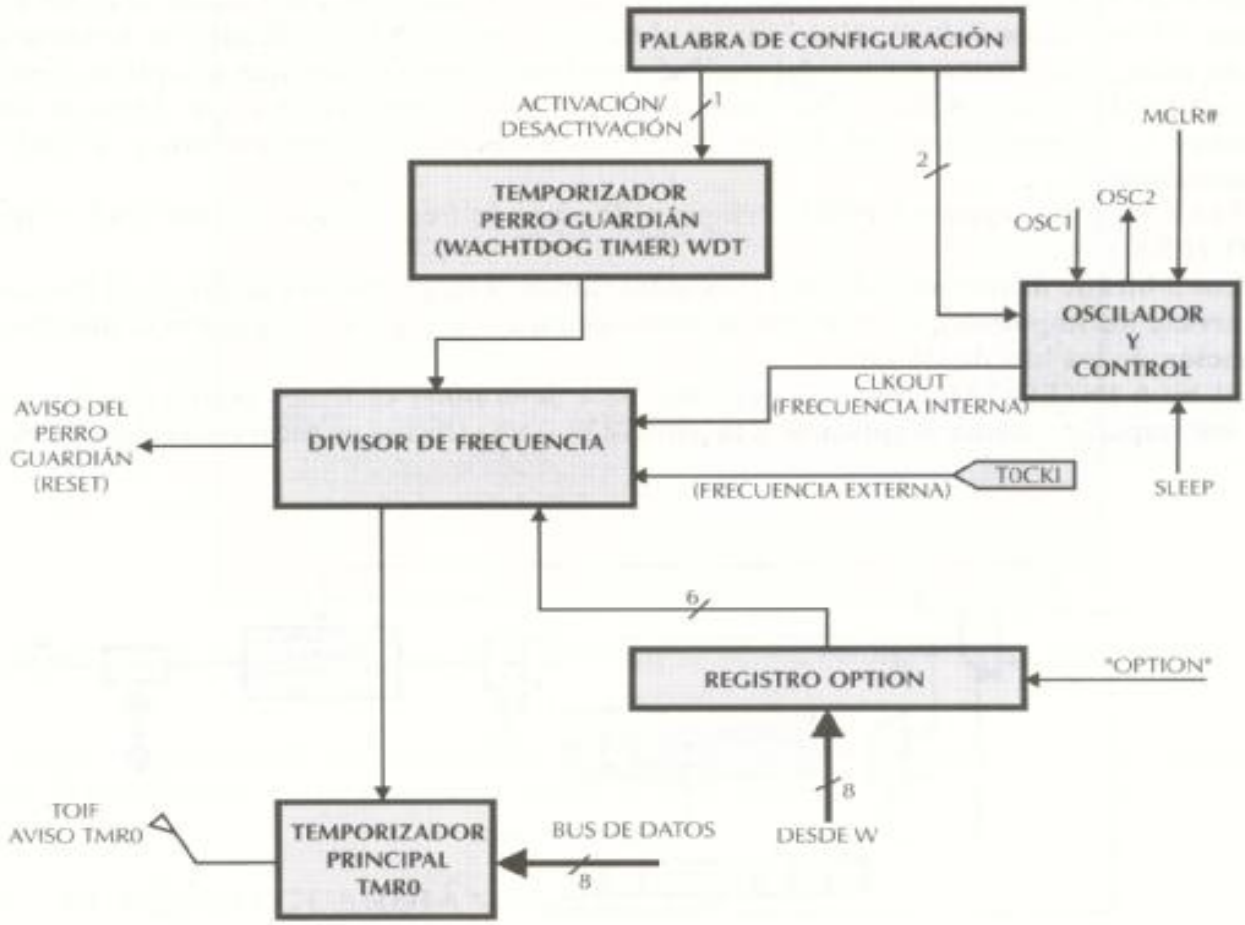


Figura 5.2. *Esquema simplificado de la sección dedicada al control de tiempos en la arquitectura de los PIC16X8X.*

plemento a 2. De esta manera, al llegar el número de impulsos deseado se desborda y al pasar por 00 H se activa el señalizador TOIF y/o se produce una interrupción.

Para calcular los tiempos a controlar con TMR0 se utilizan las siguientes fórmulas prácticas.

Temporización = 4 · Tosc · (Valor cargado en TMR0) · (Rango del Divisor)

Valor a cargar en TMR0 = Temporización/4 · Tosc · Rango del Divisor

En cualquier momento se puede leer el valor que contiene TMR0, sin detener su contaje. La instrucción adecuada al caso es *movf tmr0,w*.

En la Figura 5.3 se ofrece el esquema de funcionamiento del TMR0. Obsérvese que hay un bloque que retrasa 2 ciclos el contaje para sincronizar el momento del incremento producido por la señal aplicada en T0CKI con el que producen los impulsos internos de reloj. Cuando se escribe TMR0 se retrasan 2 ciclos su reincremento y se pone a 0 el Divisor de frecuencia.

5.2.1. El registro OPTION

La misión principal de este registro es controlar TMR0 y el Divisor de frecuencia. Ocupa la posición 81 H de la memoria de datos, que equivale a la dirección 1 del banco 1. El bit T0CS (*Timer 0 Clock Edge Select*) selecciona en el multiplexor MPX1 la procedencia de los impulsos de reloj, que pueden ser los del oscilador interno (Fosc/4) o los que se aplican desde el exterior por la patita T0CKI. El bit T0SE (*Timer 0 Clock Source Select*) elige el tipo de flanco activo en los impulsos externos. Si T0SE = 1 el flanco activo es el descendente y si T0SE = 0 el ascendente.

El bit PSA del registro OPTION asigna el Divisor de frecuencia al TMR0 (PSA = 0) o al WDT (PSA = 1).

Los 3 bits de menos peso de OPTION seleccionan el rango por el que divide el Divisor de frecuencia los impulsos que se le aplican en su entrada. La Figura 5.4 muestra la distribución y función de los bits de OPTION.

El bit 6 INTEDG (*Interrupt Edge*) sirve para determinar el flanco activo que provocará una interrupción externa al aplicarse a la patita RB0/INT. Un 1 si es ascendente y un 0 descendente.

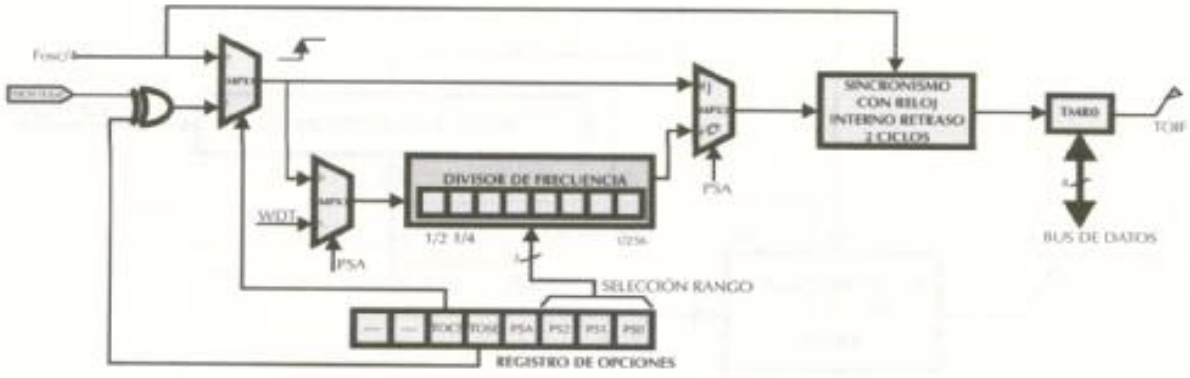


Figura 5.3. *Esquema general del funcionamiento de TMR0. Cuando se desborda se pone a 1 el señalizador TOIF.*

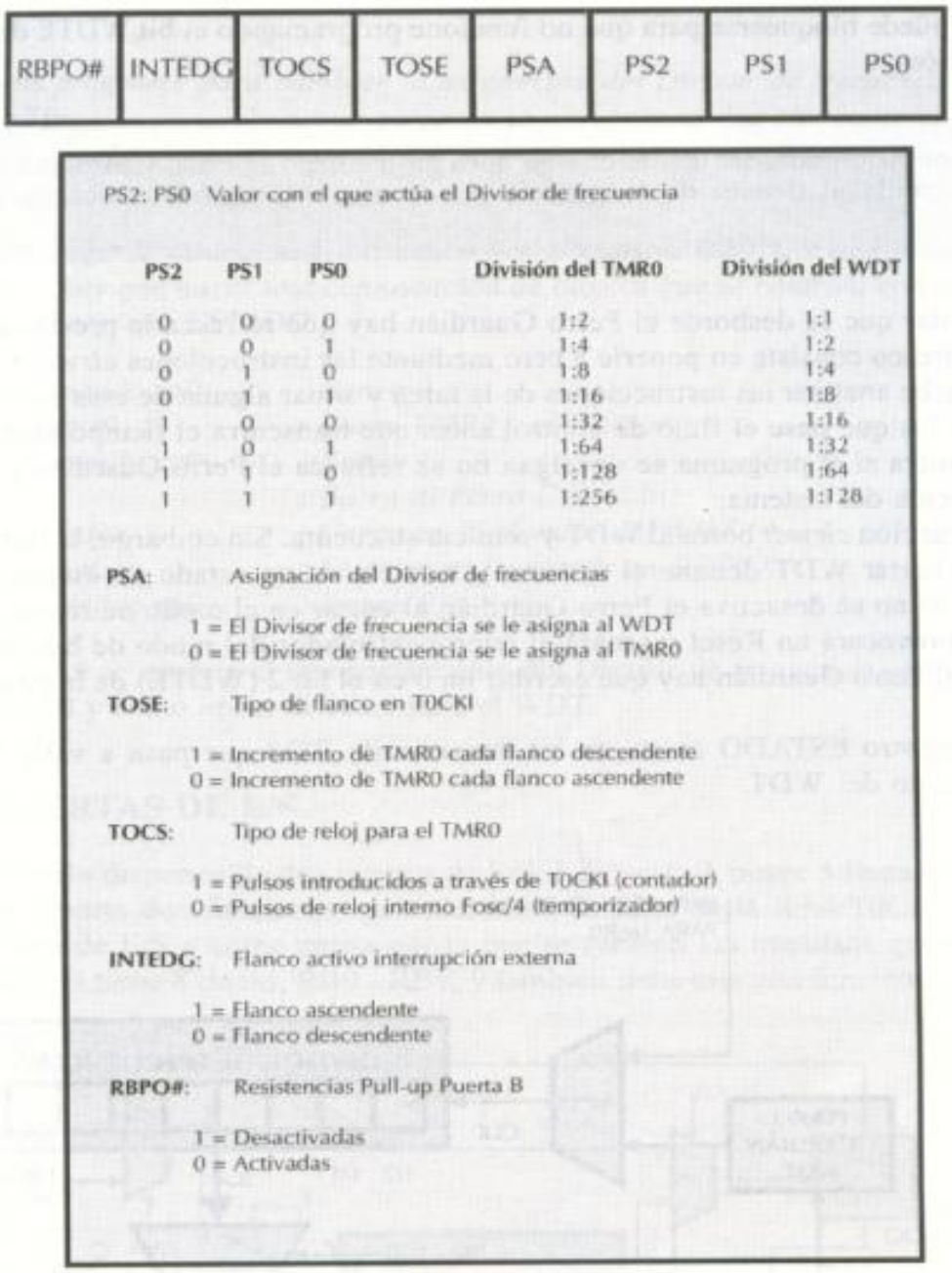


Figura 5.4. *Distribución y asignación de funciones de los bits del registro OPTION.*

El bit 7 RBPU# (*RB Pull-Up*) activa, si vale 0, o desactiva, cuando vale 1, las resistencias Pull-Up que pueden conectarse en las líneas de la Puerta B.

5.3. EL PERRO GUARDIÁN (WDT)

Se trata de un contador interno de 8 bits que origina un Reset cuando se desborda. Su control de tiempos es independiente del TMR0 y está basado en una simple red R-C. Su actuación es

opcional y puede bloquearse para que no funcione programando el bit WDTE de la Palabra de Configuración.

La temporización nominal con la que se halla programado el Perro Guardián es de 18 ms, pero utilizando el Divisor de frecuencia puede aumentarse hasta alcanzar los 2,3 segundos.

Para evitar que se desborde el Perro Guardián hay que refrescarle previamente. En realidad este refresco consiste en ponerle a cero mediante las instrucciones *clrwdt* y *sleep*. El programador debe analizar las instrucciones de la tarea y situar alguna de esas dos en sitios estratégicos por los que pase el flujo de control antes que transcurra el tiempo asignado al WDT. De esta manera si el programa se «cuelga» no se refresca el Perro Guardián y se produce la reinicialización del sistema.

La instrucción *clrwdt* borra al WDT y reinicia su cuenta. Sin embargo, la instrucción *sleep*, además de borrar WDT detiene al sistema y lo mete en un estado de «reposo» o «de bajo consumo». Si no se desactiva el Perro Guardián al entrar en el modo de reposo, al completar su contaje provocará un Reset y sacará al microcontrolador del modo de bajo consumo. Para desactivar al Perro Guardián hay que escribir un 0 en el bit 2 (WDTE) de la Palabra de Configuración.

En el registro ESTADO existe un bit denominado TO# que pasa a valer 0 después del desbordamiento del WDT.

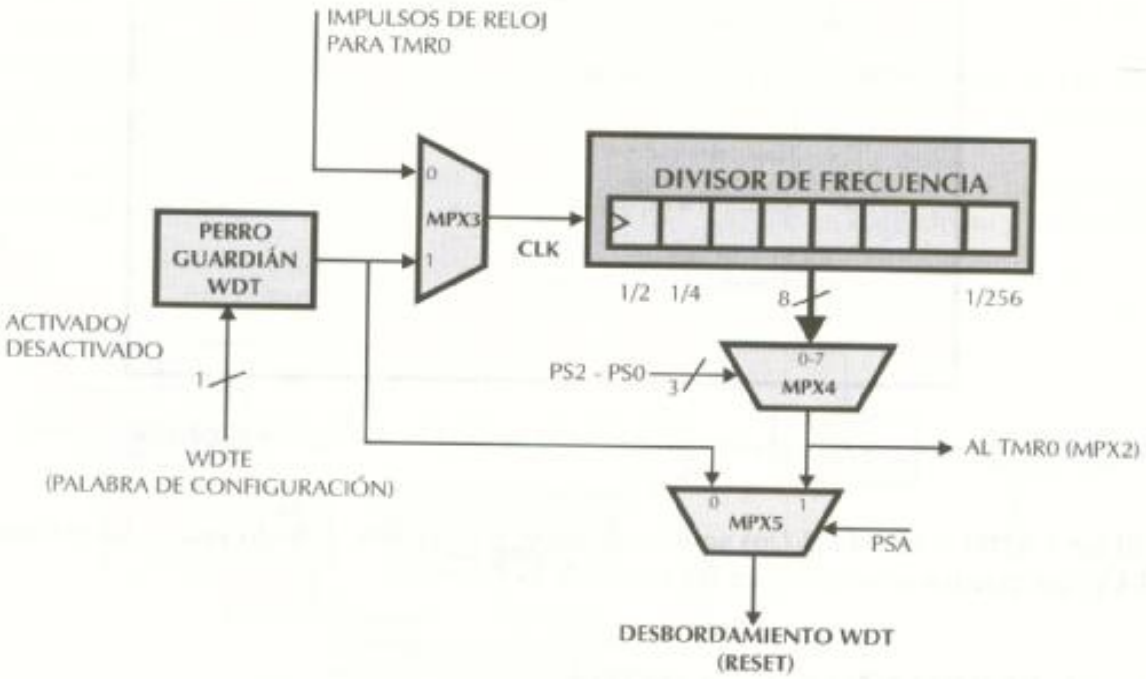


Figura 5.5. *Cuando PSA = 1 los impulsos producidos por el Perro Guardián pasan al Divisor (post-divisor). Si PSA = 0, los impulsos pasan primero por el Divisor y luego se aplican al TMR0 (pre-divisor).*

EJEMPLO

Confeccionar un programa para cambiar la asignación del Divisor de frecuencia del TMR0 al Perro Guardián.

SOLUCIÓN

Como el TMR0 ocupa la dirección 1 del banco 0 y el registro OPTION la misma dirección, pero del banco 1, hay que hacer una conmutación de bancos que se controla con el bit «rp0», que es el bit 5 del registro ESTADO.

```
bcf     estado,rp0      ; Banco 0
clr     tmr0            ; Borra TMR0 y el Divisor
bsf     estado,rp0      ; Banco 1
clrwdt                  ; Borra al Perro Guardián
movlw   valor           ; Carga en W el valor del WDT
movwf   option          ; Se asignan funciones en OPTION
bcf     estado,rp0      ; Banco 0
```

En la Figura 5.6 se aprecia el comportamiento del Divisor de frecuencia como «pre-divisor» para el TMR0 y como «post-divisor» para el WDT.

5.4. LAS PUERTAS DE E/S

Los PIC16X8X sólo disponen de dos puertas de E/S. La Puerta A posee 5 líneas, RA0 - RA4, y una de ellas soporta dos funciones multiplexadas. Se trata de la RA4/T0CKI, que puede actuar como línea de E/S o como patita por la que se reciben los impulsos que debe contar TMR0. La Puerta B tiene 8 líneas, RB0 - RB7, y también tiene una con funciones multiplexa-

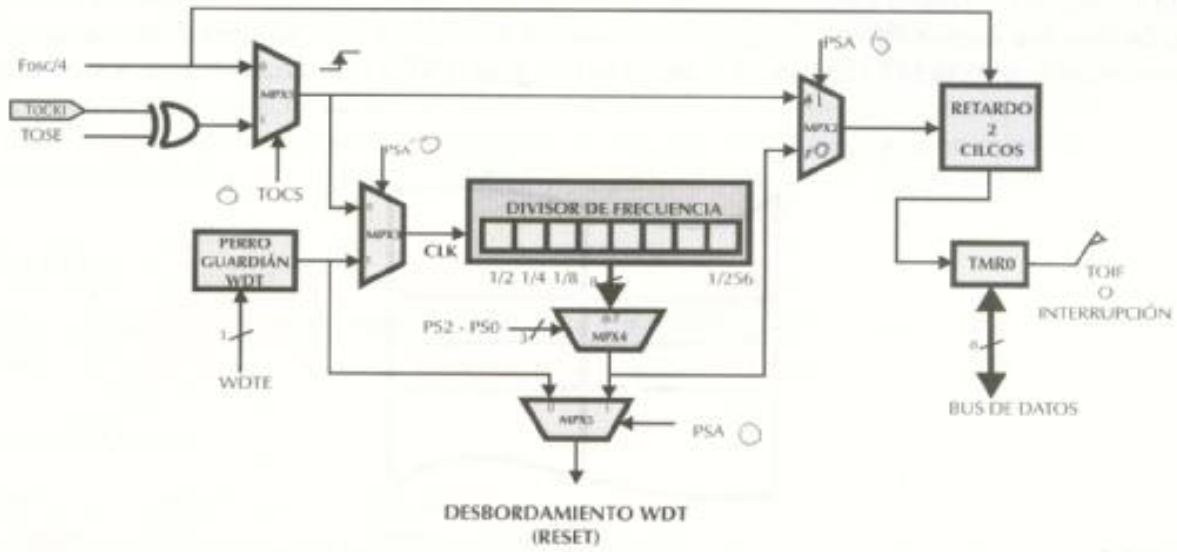


Figura 5.6. *Esquema que resalta la actuación del Divisor de frecuencia con el TMR0 y con el WDT. El bit PSA selecciona la asignación del Divisor al TMR0 o al WDT.*

das, la RB0/INT, que, además de línea típica de E/S, también sirve como patita por la que se reciben los impulsos externos que provocan una interrupción

Cada línea de E/S puede configurarse independientemente como entrada o como salida, según se ponga a 1 o a 0, respectivamente, el bit asociado del registro de configuración de cada puerta (TRISA y TRISB). Se llaman PUERTAA y PUERTAB los registros que guardan la información que entra o sale por la puerta y ocupan las direcciones 5 y 6 del banco 0 de la memoria de datos. Los registros de configuración TRISA y TRISB ocupan las mismas direcciones pero en el banco 1. (Figura 5.7.)

5.4.1. La Puerta A

Las líneas RA3 - RA0 admiten niveles de entrada TTL y de salida CMOS. La línea RA4/T0CKI dispone de un circuito Schmitt Trigger que proporciona una buena inmunidad al ruido y la salida tiene drenador abierto. RA4 multiplexa su función de E/S con la de entrada de impulsos externos para el TMR0.

Los bits del registro TRISA configuran a las líneas de la Puerta A como entradas si están a 1 y como salidas si están a 0.

En el circuito de la Figura 5.8 se muestra la adaptación de las patitas RA3 - RA0 a las señales internas del procesador.

Cuando se lee una línea de la Puerta A (instrucción *movfw puertaa*) se recoge el nivel lógico que tiene en ese momento. Las líneas cuando actúan como salidas están «lacheadas», lo que significa que sus patitas sacan el nivel lógico que se haya cargado por última vez en el registro PUERTAA. La escritura de una puerta implica la operación «lectura/modificación/escritura». Primero se lee la puerta, luego se modifica el valor y finalmente se escribe en el «latch» de salida.

De la Figura 5.8 se desprende que cuando se saca un nivel lógico por una línea de la Puerta A, primero se deposita en la línea correspondiente del bus interno de datos y se activa la señal WRITE, lo que origina el almacenamiento de dicho nivel en la báscula de datos. En esta situación, la báscula de configuración debería contener un 0 para que actuase como salida. Con estos valores la puerta OR tendría un 0 en su salida y la AND también. Estos valores producen

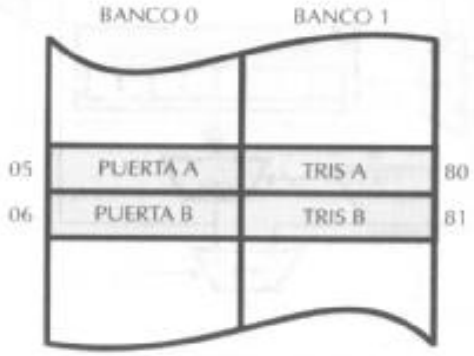


Figura 5.7. *La información que entra o sale por las puertas reside en las posiciones 5 y 6 del banco 0, mientras que TRISA y TRISB, que son los registros de configuración, ocupan las direcciones 5 y 6 del banco 1.*

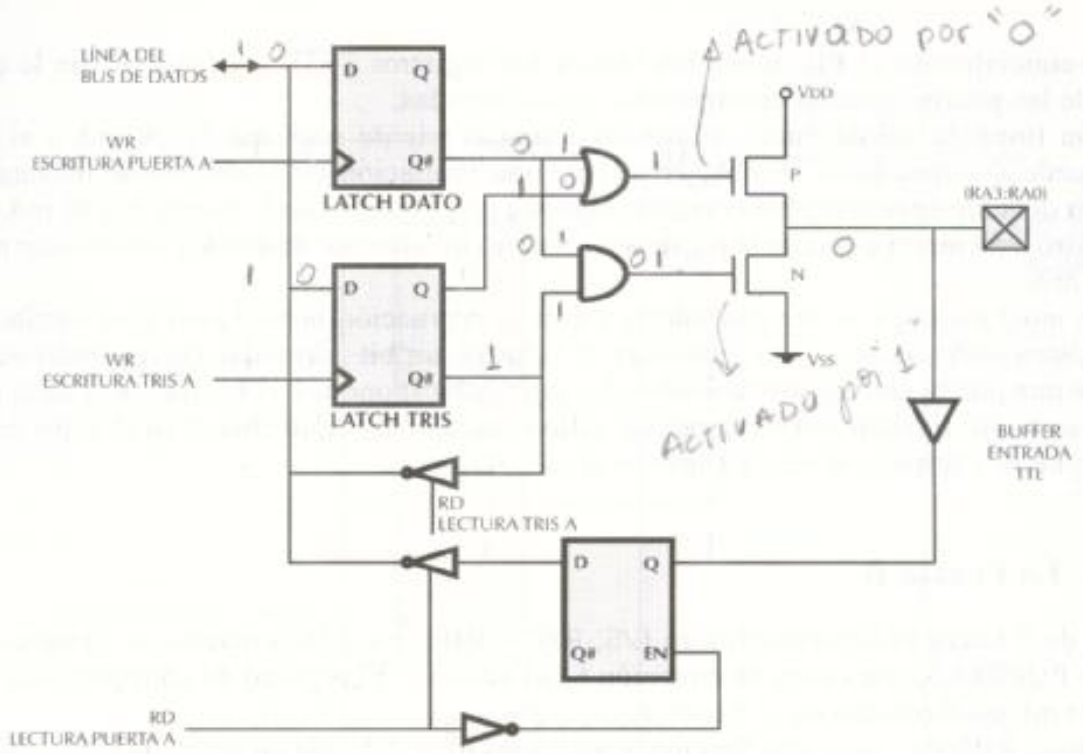


Figura 5.8. *Conexión electrónica de las patitas RA3 - RA0 con las líneas de control y del bus de datos del procesador.*

la conducción del transistor PMOS superior y el bloqueo del NMOS. Así, la patita de E/S queda conectada a la V_{DD} y tiene nivel alto. Como la línea de salida está lacheada conserva su valor hasta que no se reescriba en la báscula D.

Si una línea actúa como entrada, el nivel lógico depositado en ella desde el exterior pasa a la línea correspondiente del bus interno de datos cuando se activa la señal READ y se hace conductor el dispositivo triestado que les une. Al programarse como entrada, los dos transistores MOS de salida quedan bloqueados y la línea en alta impedancia. Téngase en cuenta que cuando se lee una línea de entrada se obtiene el estado actual que tiene su patita correspondiente y no el valor que haya almacenado en la báscula de datos. La información presente en una línea de entrada se muestrea al iniciarse el ciclo de instrucción y debe mantenerse estable durante su desarrollo.

EJEMPLO

Configurar las dos líneas de menos peso de la Puerta A como entradas y las otras tres como salidas.

SOLUCIÓN

```
bsf     estado,rp0    ; Banco 1
movlw   0x03          ; Valor configuración Puerta A
movwf   trisa         ; RA4:RA2 salidas y RA0:RA1
                      ; entradas
```

Al reinicializarse el PIC todos los bits de los registros TRIS quedan a 1, con lo que las líneas de las puertas quedan configuradas como entradas.

Cada línea de salida puede suministrar una corriente máxima de 20 mA y si es entrada puede absorber hasta 25 mA. Al existir una limitación en la disipación máxima de la potencia del chip se restringe la corriente máxima de absorción de la Puerta A a 80 mA y la de suministro a 50 mA. La Puerta B puede absorber un máximo de 150 mA y suministrar un total de 100 mA.

Con *movf puerta,w* se lee una puerta y con la instrucción *movwf puerta* se escribe. También existen instrucciones para modificar el valor de un bit particular correspondiente a una línea de una puerta con las instrucciones *bsf puerta,bit* (pone a 1 el bit indicado de la puerta) y *bcf puerta,bit*. Existen instrucciones de salto condicionales que chequean el valor de un bit de una puerta y brincan si vale 1 (*btfss*) o si vale 0 (*btfsc*).

5.4.2. La Puerta B

Consta de 8 líneas bidireccionales de E/S, RB7 - RB0, cuya información se almacena en el registro PUERTAB, que ocupa la dirección 6 del banco 0. El registro de configuración TRISB ocupa la misma dirección en el banco 1.

La línea RB0/INT tiene dos funciones multiplexadas. Además de patita de E/S, actúa como patita para la petición de una interrupción externa, cuando se autoriza esta función mediante la adecuada programación del registro INTCON, del que se hablará más adelante.

A todas las líneas de esta puerta se las permite conectar una resistencia pull-up de elevado valor con el positivo de la alimentación. Para este fin hay que programar en el registro OPTION el bit RBPU# = 0, afectando la conexión de la resistencia a todas las líneas. Con el Reset todas las líneas quedan configuradas como entradas y se desactivan las resistencias pull-up.

Las 4 líneas de más peso, RB7 - RB4, pueden programarse para soportar una misión especial. Cuando las 4 líneas actúan como entradas se las puede programar para generar una interrupción si alguna de ellas cambia su estado lógico. Esta posibilidad es muy práctica en el control de teclados. En la Figura 5.9 se muestra el esquema de conexionado entre las patitas RB7 - RB4 y las líneas correspondientes del bus de datos interno.

El estado de las patitas RB7 - RB4 en modo entrada se compara con el valor antiguo que tenían y que se había lacheado durante la última lectura de la Puerta B. El cambio de estado en alguna de esas líneas origina una interrupción y la activación del señalizador RBIF.

La línea RB6 también se utiliza para la grabación serie de la memoria de programa y sirve para soportar la señal de reloj. La línea RB7 constituye la entrada de los datos en serie.

5.5. LA PALABRA DE CONFIGURACIÓN

Se trata de una posición reservada de la memoria de programa situada en la dirección 2007 H y accesible únicamente durante el proceso de grabación. Al escribirse el programa de la aplicación es necesario grabar el contenido de esta posición de acuerdo con las características del sistema.

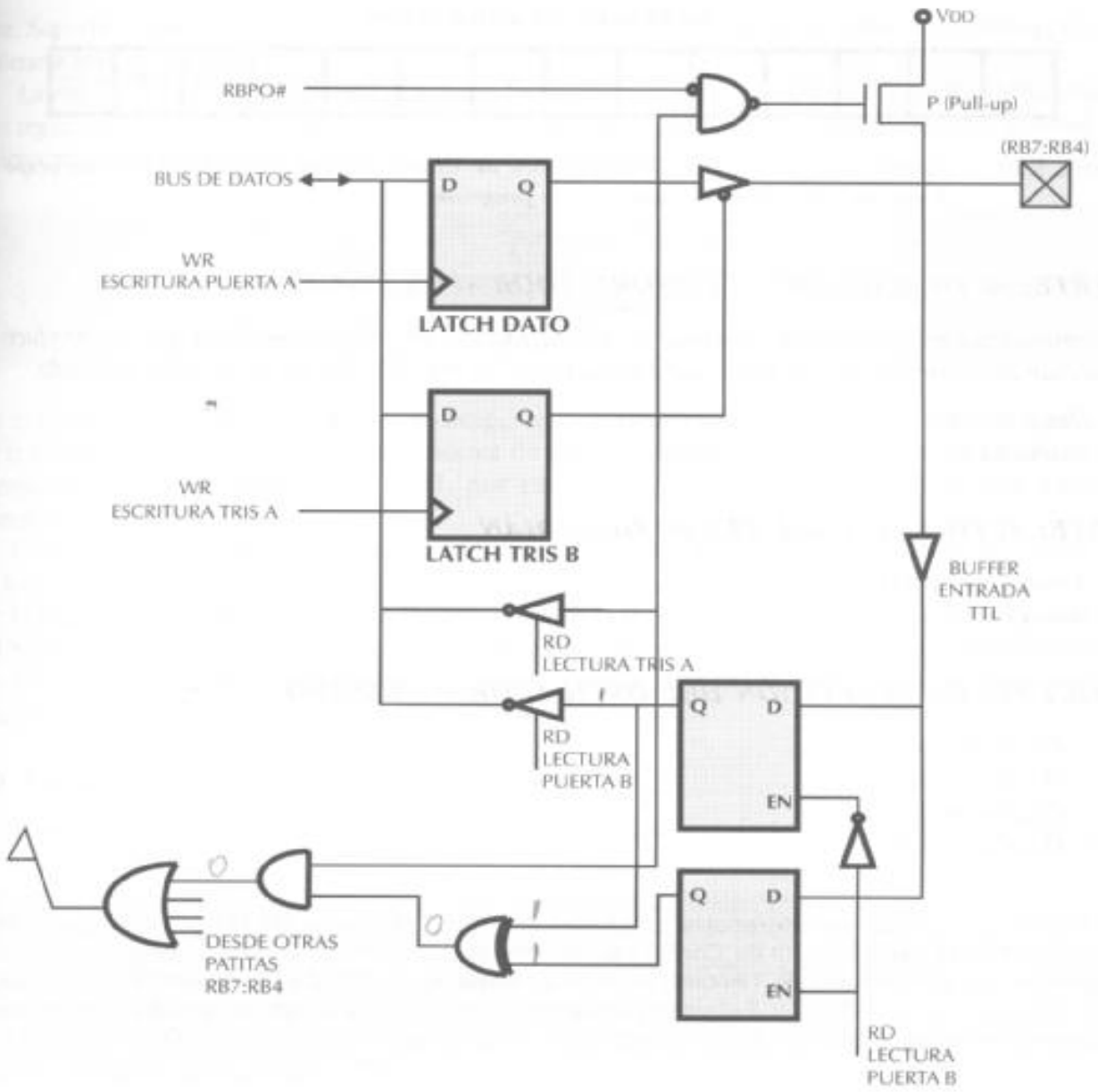


Figura 5.9. *Conexionado de las patitas RB7 - RB4 y las líneas correspondientes al bus interno de datos y las señales de control.*

En la Figura 5.10 se muestra la distribución y asignación de los 14 bits de la Palabra de Configuración de los PIC16F8X, que tienen las siguientes funciones:

CP: BITS DE PROTECCIÓN DE LA MEMORIA DE CÓDIGO

1: No protegida

0: Protegida. El programa no se puede leer, evitando copias. Tampoco se puede sobreescribir. Además evita que pueda ser accedida la EEPROM de datos y, finalmente, si se modifica el bit CP de 0 a 1, se borra completamente la EEPROM.



Figura 5.10. *Distribución de los bits de la Palabra de Configuración del PIC16F8X, que ocupa la dirección 2007 H de la memoria de programa.*

PWRTE: ACTIVACIÓN DEL TEMPORIZADOR «POWER-UP»

El temporizador «power-up» retrasa 72 ms la puesta en marcha o Reset que se produce al conectar la alimentación al PIC, para garantizar la estabilidad de la tensión aplicada.

1: Desactivado
0: Activado

WDTE: ACTIVACIÓN DEL PERRO GUARDIÁN

1: Activado el WDT
0: Desactivado

FOSC1-FOSC0: SELECCIÓN DEL OSCILADOR UTILIZADO

1-1: Oscilador RC
1-0: Oscilador HS
0-1: Oscilador XT
0-0: Oscilador LP

Una de las poquísimas diferencias que tiene el PIC16C84 con el PIC16F84 es la actuación del bit PWRTE de la Palabra de Configuración, que tiene invertida su función en ambos. En el primero cuando este bit vale 1 el temporizador de Power-Up está activado, mientras que sucede lo contrario en el segundo. Este temporizador retrasa 72 ms el funcionamiento del microcontrolador al conectarle la alimentación para asegurar la estabilidad del voltaje aplicado.

5.5.1. Palabras de Identificación (ID)

Son 4 posiciones reservadas de la memoria de programa ubicadas en las direcciones 2000 H - 2003 H que no son accesible en el funcionamiento normal del microcontrolador y sólo pueden ser leídas y escritas durante el proceso de grabación.

Sólo se utilizan los 4 bits de menos peso de cada palabra de identificación (ID), en donde se almacena un valor que puede consistir en un número de serie, códigos de identificación, numeraciones secuenciales o aleatorias, etc.

5.6. LA MEMORIA EEPROM DE DATOS

Los PIC16X8X tienen 64 bytes de memoria EEPROM de datos, donde se pueden almacenar datos y variables que interesa que no se pierdan cuando se desconecta la alimentación al siste-

ma. Soporta 1.000.000 de ciclos de escritura/borrado y es capaz de guardar la información sin alterarla más de 40 años.

La memoria EEPROM no está mapeada en la zona de la memoria de datos donde se ubican los registros SFR y GPR. Para poder leerla y escribirla durante el funcionamiento normal del microcontrolador hay que utilizar 4 registros del banco SFR:

— *EEDATA*
— *EEADR*
— *EECON1*
— *EECON2*

En el registro EEADR, ubicado en la dirección 9 del banco 0, se carga la dirección a acceder de la EEPROM de datos. Las 64 posiciones de un byte ocupan las direcciones de un mapa que comienza en 00 H y termina en 3F H, por eso los 2 bits de más peso del registro EEADR siempre valen 0.

En el registro EEDATA, ubicado en la dirección 8 del banco 0, se depositan los datos que se leen o se escriben.

El registro EECON1, que ocupa la dirección 88 H de la memoria de datos, o la dirección 8 H del banco 1, tiene misiones de control de las operaciones en la EEPROM y la distribución de sus bits se presenta en la Figura 5.11, mientras que la misión de cada uno se explica a continuación:

RD: Lectura

1: Se pone a 1 cuando se va a realizar un ciclo de lectura de la EEPROM. Luego pasa a 0 automáticamente.

WR: Escritura

1: Se pone a 1 cuando se inicia un ciclo de escritura de la EEPROM. Cuando se completa el ciclo pasa a 0 automáticamente.

WREN: Permiso de escritura

1: Permite la escritura de la EEPROM.
0: Prohíbe la escritura.

WRERR: Señalizador de error en escritura

1: Se pone a 1 cuando una operación de escritura ha terminado prematuramente.
0: La operación de escritura se ha completado correctamente.



Figura 5.11. *Distribución de los bits del registro EECON1 de los PIC16X8X.*

EEIF: Señalizador de final de operación de escritura

1: Cuando este señalizador se pone a 1 indica que la operación de escritura se ha completado con éxito. Se pone a 0 por programa.
0: La operación de escritura no se ha completado.

Un ciclo de escritura de una posición de la EEPROM de datos tiene una duración típica de 10 ms, que resulta muy larga para la velocidad del procesador. Por este motivo existen varios bits en el registro EECON1 para supervisar la completa y correcta terminación del mismo.

El registro EECON2 en realidad no está implementado físicamente. Al leerlo todos sus bits son 0. Sólo se emplea como un dispositivo de seguridad durante el proceso de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo.

Proceso de lectura

Se inicia un ciclo de lectura colocando la dirección a acceder en el registro EEADR y poniendo el bit RD = 1 en el registro EECON1. El dato leído estará disponible en el registro EEDATA en el siguiente ciclo y permanecerá en él hasta que se realice una nueva lectura o escritura en la EEPROM.

EJEMPLO

Confeccionar un programa para realizar la lectura de una posición, «lect», de la EEPROM de datos.

SOLUCIÓN

```
bcf     estado,rp0      ; Banco 0
movlw   lect            ; Dirección a leer
movwf   eeadr           ;
bsf     estado,rp0      ; Banco 1
bsf     eecon1,rd       ; Lectura
bcf     estado,rp0      ; Banco 0
movf    eedata,w        ; W se carga con valor leído en EEPROM
```

Proceso de escritura

Para escribir una posición de la EEPROM de datos el usuario debe seguir una determinada secuencia de instrucciones en las que participa el registro EECON2. Este registro, que en realidad no se halla implementado físicamente, sólo asume funciones de seguridad en el proceso, cargándose en él dos valores concretos: 55 H y AA H. La duración típica de un ciclo de escritura es de 10 ms, que es notablemente larga en comparación con la velocidad del PIC.

El ciclo de escritura comienza cargando en EEADR la dirección de la posición a escribir y en el registro EEDATA el valor a grabar.

EJEMPLO

Escribir en la dirección «escr» de la EEPROM de datos el valor dato.

SOLUCIÓN

```
bcf     estado,rp0      ; Banco 0
movlw   escr            ;
movwf   eeadr           ; Se escribe la dirección en eeadr
movlw   dato            ;
movwf   eedata          ; Se escribe el dato en eedata
bsf     estado,rp0      ; Banco 1
bsf     eecon1,wren     ; Permiso escritura
                        ; Inicio Secuencia Escritura
movlw   0x55 h          ;
movwf   eecon2          ; Se escribe 55 h en eecon2
movlw   0xaa h          ;
movwf   eecon2          ; Se escribe AA h en eecon2
bsf     eecon1,wr       ; Comienza la escritura
```

Al acabar el proceso de escritura el bit WR pasa a valer 0 automáticamente, mientras que el señalizador EEIF se pone a 1. Este último bit hay que ponerlo a 0 posteriormente mediante software.

Una buena precaución es verificar si la escritura de la EEPROM ha sido correcta, para lo cual se suele restar el dato escrito con el que existe en el registro EEDATA. Si no se ha producido error el señalizador Z pasa a valer 1.

PROGRAMAR PIC ES FÁCIL

EL QUINTO PROGRAMA DE APRENDIZAJE: CONTROLANDO EL TIEMPO

Una vez que se conoce el comportamiento del temporizador/contador TMR0 se pretende manejarlo con este ejercicio en el que se dedica a controlar un tiempo concreto, de relativamente larga duración. Dicho tiempo (8,2 ms) se usará para regular el parpadeo de un diodo led.

ENUNCIADO

En un PIC16X84 que trabaja a 1 MHz se conecta un diodo led a la línea RB7 de la Puerta B. Se desea hacer parpadear al led cada 8,2 ms. Confeccionar el programa.

SOLUCIÓN

En primer lugar vamos a explicar dos nuevas instrucciones que nos van a ser de gran utilidad, ya que nos van a permitir realizar subrutinas dentro de nuestro código. Las subrutinas desvían el flujo de ejecución de nuestro programa a otro punto, pudiendo luego retornar al punto original, para lo cual ha de salvarse el Contador de Programa en la pila. La pila del PIC16X8X tiene ocho niveles, lo que permite anidar hasta ocho subrutinas mediante las siguientes instrucciones:

call k: *Guarda la dirección de la instrucción actual en la pila y salta a la dirección k.*

return: *Retorna a la dirección que esté en la cabeza de la pila, esto es, la última que haya sido almacenada.*

Para medir los 8,2 ms se emplea el TMR0, al cual se le hace contar 16 impulsos del reloj interno divididos por 128 mediante el Divisor de frecuencia. La fórmula que indica el tiempo de la temporización es:

Temporización = 4 · Tosc · Valor cargado en TMR0 · Rango del Divisor

Temporización = 4 · 1 µs · 16 · 128 = 8,192 ms ≈ 8,2 ms

El único hardware que hay que añadir al circuito básico de funcionamiento del PIC es el que corresponde al periférico a gobernar, tal como se muestra en la Figura 5.12.

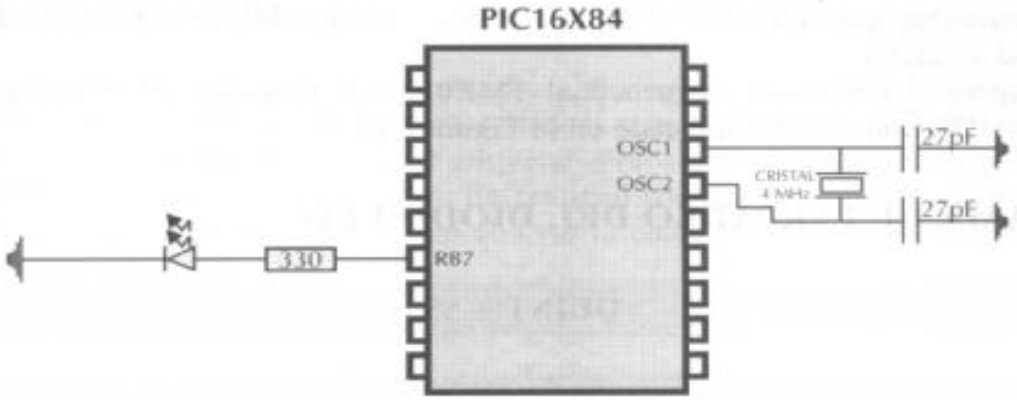


Figura 5.12. *Conexión del led a la línea RB7.*

En la Figura 5.13 se muestra el organigrama que refleja las fases operativas del programa y la subrutina RETARDO, que es la que determina el tiempo de 8,2 ms.

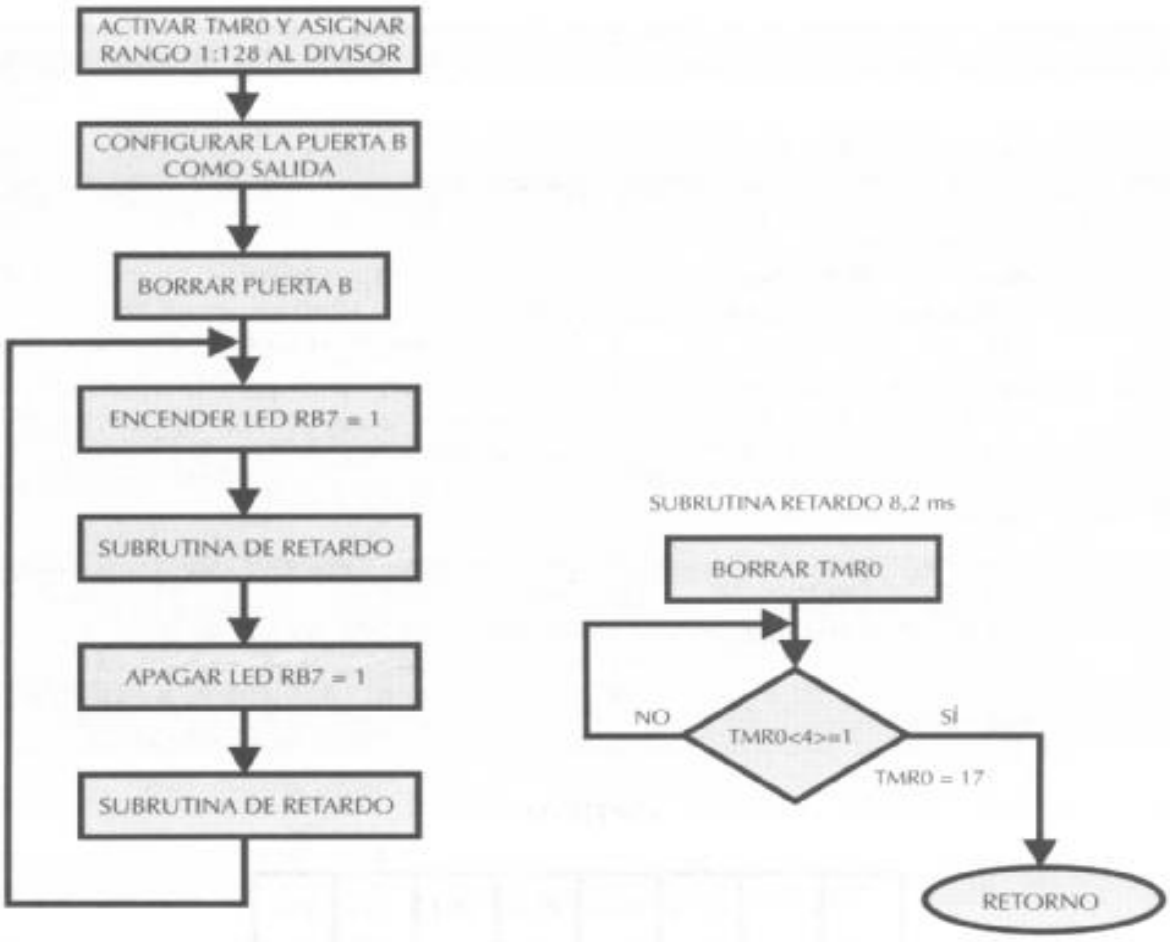


Figura 5.13. *Organigrama del programa principal para controlar el parpadeo del led y el de la subrutina RETARDO que cuenta los 8,2 ms.*

Para comprobar que el TMR0 ha llegado a $16_{10} = 0001\ 0000_2$ se explora el momento que su bit 4 pasa a valer 1.

Para asignar el Divisor de frecuencia al TMR0 con el rango de 1:128 hay que cargar al registro OPTION con el valor indicado en la Figura 5.14.

PROGRAMA DE PARPADEO DEL DIODO LED

QUINTO.ASM

```
            LIST    P=16C84
            RADIX   HEX

            PUERTAB EQU   0x06
            OPTION  EQU   0x01
            ESTADO  EQU   0x03
            TMR0    EQU   0x01

            ORG     0               ; Inicio del programa en dirección 0

            bsf     ESTADO,5        ; Banco 1
            movlw   b'11010110'     ; Valor a cargar en OPTION
            movwf   OPTION
            movlw   0x00
            movwf   PUERTAB         ; La Puerta B salida
            bcf     ESTADO,5        ; Banco 0
            clrf    PUERTAB         ; Las líneas de salida de PB a 0
parpa       bsf     PUERTAB,7       ; Enciende el led RB7 = 1
            call    retardo         ; Llamada a subrutina de RETARDO
            bcf     PUERTAB,7       ; Apaga el led, RB7 = 0
            call    retardo
            goto    parpa
retardo     clrf    TMR0            ; TMR0 = 0 y empieza su incremento
explora     btfss   TMR0,4          ; TMR0<4> = 1?
            goto    explora         ; No ha llegado TMR0 a 16(10)
            return                  ; Ha llegado TMR0 al valor 16(10)
                                    ; y retorna al programa principal
            END
```

OPTION

1	1	0	1	0	1	1	0
		TOCS	TOSE	PSA	PS2	PS1	PS0

Figura 5.14. *Valor que hay que cargar en el registro OPTION para asignar el Divisor a TMR0 y hacer que divida por 128 los impulsos del reloj interno.*

¡ERROR DE ENSAMBLADO!

Este programa da un error al ser ensamblado... ¿Por qué? Para hallar una respuesta a esta pregunta debemos recordar una de las características más conflictivas de los PIC: el empleo de varios bancos de memoria. A la hora de etiquetar direcciones de memoria hemos asignado a una dirección de memoria dos etiquetas. En realidad son dos direcciones diferentes, porque están en bancos distintos que nada tienen que ver la una con la otra, pero eso es algo que el ensamblador no sabe.

```
OPTION      EQU     0x01
TMR0        EQU     0x01
```

¿Cómo podríamos solucionar esto? En principio podríamos etiquetar OPTION como 0x81, ya que está en el banco 1 y ésta es la dirección que le correspondería si los bancos siguieran un orden de numeración absoluto. El caso es que esto tampoco funciona.

Se suelen utilizar dos «trucos» para evitar este inconveniente. El primero es etiquetar OPTION como 0x81, pero cada vez que hagamos referencia a esta dirección usamos la etiqueta OPTION^81H. Esto evita que el ensamblador dé error y hace que el programa se ensamble correctamente.

La otra solución, menos engorrosa, es la de utilizar etiquetas compartidas, esto es, una única etiqueta que se refiera a varias direcciones. Existen varias formas de hacer esto; nosotros empleamos:

```
TMR0_OPT  EQU   0x01
```

De esta manera sabremos que la parte de la etiqueta a la izquierda del subrayado se refiere a la posición de memoria 0x01 del banco 0, mientras que la parte derecha se refiere a la misma dirección, pero del banco siguiente.

El programa, utilizando etiquetas compartidas, quedaría del siguiente modo:

QUINTO.ASM

```
        LIST    P=16C84
        RADIX   HEX

        PUERTAB EQU   0x06
        ESTADO  EQU   0x03

        TMR0_OPT EQU  0x01

        ORG     0               ; Inicio del programa en dirección 0

        bsf     ESTADO,5        ; Banco 1
        movlw   b'11010110'     ; Valor a cargar en OPTION
```

```
            movwf     TMR0_OPT
            movlw     0x00
            movwf     PUERTAB       ; La Puerta B salida
            bcf       ESTADO,5      ; Banco 0
            clrf      PUERTAB       ; Las líneas de salida de PB a 0

parpa       bsf       PUERTAB,7     ; Enciende el led RB7 = 1
            call      retardo       ; Llamada a subrutina de RETARDO
            bcf       PUERTAB,7     ; Apaga el led, RB7 = 0
            call      retardo
            goto      parpa

retardo     clrf      TMR0_OPT      ; TMR0 = 0 y empieza su incremento
explora     btfss     TMR0_OPT,4    ; TMR0<4> = 1?
            goto      explora       ; No ha llegado TMR0 a 16(10)
            return                  ; Ha llegado TMR0 al valor 16(10) y
                                    ; retorna al programa principal

            END
```

PROBANDO CON EL SIMUPIC'84

A la hora de simular un programa con temporizaciones siempre es importante tener en cuenta que la simulación no se va a hacer en «tiempo real». Siempre será más lenta. Hay que tener esto presente e incluso considerar reducir proporcionalmente las temporizaciones para que la simulación «no nos desespere». En este caso no hace falta que reduzcamos el período de 8,2 ms, porque el simulador nos da una respuesta en un tiempo aceptable.

Para saber cuánto tiempo de simulación ha transcurrido (tiempo simulado, no tiempo real, que es mayor) utilizamos la ventana de contadores, pero antes deberemos ajustar la velocidad del reloj, ya que el tiempo transcurrido se calcula en base a este dato. Este parámetro se cambia desde el menú de «opciones de simulación», y viene dado en KHz. De modo que cargaremos el programa tal y como ya sabemos y ajustaremos la velocidad a 1000 KHz (1 MHz).

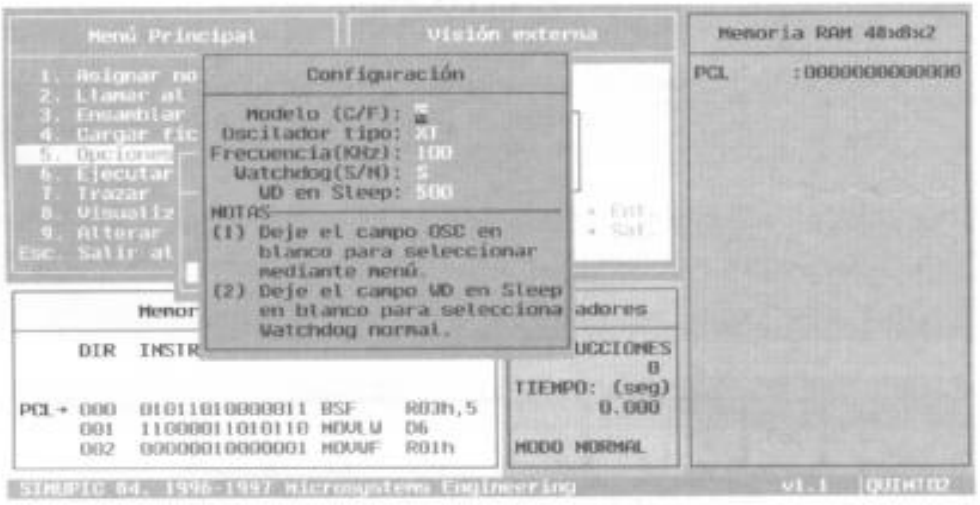


Figura 5.15. *Ajustando la velocidad de la simulación.*

PROGRAMANDO EN C

Se ofrece el mismo programa con el compilador PCM.

```
/* QUINTO.C: Programa que hace parpadear un led */
/* en RB7 cada 8,2 ms                           */
```

```
#include <16C84.H>
#use     delay( clock = 1000000 )  /* Especifica reloj de 1 MHz */
#byte    puertab = 06              /* Dirección de la puerta B */

void main( void )
{
 set_tris_b( 0x00 );
 puertab = 0;
 do{
    delay_us( 8200 );              /* Crea un bucle de espera de
                                      8200 microsegundos */

    bit_set( puertab, 7 );         /* Enciende el LED */
    delay_us( 8200 );              /* Espera */
    bit_clear( puertab, 7 );       /* Apaga el LED */

 } while( TRUE );                  /* Repetir siempre */
}
```

El programa es tan sencillo porque el compilador PCM pone a nuestro alcance librerías con funciones de uso común. De este modo el realizar una temporización es tan simple como usar la directiva `#use delay(clock=velocidad_en_hercios)` y luego emplear las funciones `delay_ms`, `delay_us` o `delay_cycles` según nos convenga.

TEMPORIZACIONES EN C

delay_us(xxx)	Temporización de xxx microsegundos
delay_ms(xxx)	Temporización de xxx nanosegundos
delay_cycles(xxx)	Pausa de xxx ciclos de instrucción (máximo 255)

MONTAJE Y EXPERIMENTACIÓN EN EL MICRO'PIC TRAINER

Existe un problema a la hora de probar este programa con el Micro'PIC Trainer, y es que nuestro programa está preparado para funcionar a 1 MHz, mientras que el cristal de cuarzo del que dispone la placa es de 4 MHz. La solución es fácil: hay que modificar nuestro programa.

Como la frecuencia de la placa es cuatro veces mayor habrá que multiplicar por cuatro el valor del contador, y esto en binario significa rotar «16» dos veces a la izquierda.

$16_{10} \cdot 4_{10} = 64_{10}$
$0001\ 0000_2 \cdot 4_{10} = 0100\ 0000_2$

De esta forma conseguimos que:

Temporización = 4 · Tosc · Valor a contar por TMR0 · Rango del Divisor

Temporización = 4 · 0,25 ms · 64 · 128 = 8.192 ms = 8,2 ms

Hay que comprobar el bit 6 en lugar del bit 4, con lo que el nuevo programa queda así:

QUINTO3.ASM

```
            LIST      P=16C84
            RADIX     HEX

            PUERTAB   EQU   0x06
            ESTADO    EQU   0x03
            TMR0_OPT  EQU   0x01

            ORG       0               ; Inicio del programa en dirección 0

            bsf       ESTADO,5        ; Banco 1
            movlw     b'11010110'     ; Valor a cargar en OPTION
            movwf     TMR0_OPT
            movlw     0x00
            movwf     PUERTAB         ; La Puerta B salida
            bcf       ESTADO,5        ; Banco 0
            clrf      PUERTAB         ; Las líneas de salida de PB a 0
parpa       bsf       PUERTAB,7       ; Enciende el led RB7 = 1
            call      retardo         ; Llamada a subrutina de RETARDO
            bcf       PUERTAB,7       ; Apaga el led, RB7 = 0
            call      retardo
            goto      parpa
retardo     clrf      TMR0_OPT        ; TMR0 = 0 y empieza su incremento
explora     btfss     TMR0_OPT,6      ; TMR0<6> = 1?
            goto      explora         ; No ha llegado TMR0 a 64₁₀

            return                    ; Ha llegado TMR0 al valor 64₁₀
                                      ; y retorna al programa principal
            END
```

Al ensamblar este programa y cargarlo en el Micro'PIC Trainer, ¿parpadea el led? La respuesta es NO. El led no parpadea porque una temporización de 8,2 ms es inapreciable al ojo

humano. Si se fija mucho notará que el led se ilumina con una intensidad ligeramente inferior a la del led de alimentación; es el único efecto apreciable a simple vista.

CORRECCIONES AL PROGRAMA

Vamos a tratar de conseguir la mayor temporización posible utilizando el Divisor de frecuencia con rango 256 y el TMR0 con su valor máximo. De esta manera, a 4 MHz obtendremos:

```
Temporización = 4 · Tosc · Valor cargado en TMR0 · Rango del Divisor
Temporización = 4 · 0,25 µs · 255 · 255 = 65,025 ms
```

Ya no sirve controlar el cuarto ni el sexto bit del Timer0, porque los necesitamos todos. Aunque vigilásemos el séptimo bit, éste se pondría a 1 al llegar a 128 y no a 255. ¿Qué bit debemos vigilar si queremos saber cuándo se desborda el Timer0? Usamos el señalizador que nos indica cuándo se produce el desbordamiento del temporizador. Está en el registro «INTCON», dirección 0x0b y 0x8b (se encuentra en los dos bancos). En concreto es el bit T0IF, que es el bit 2.

QUINTO4.ASM

```
        LIST    P=16C84
        RADIX   HEX

PUERTAB EQU     0x06
ESTADO  EQU     0x03
TMR0_OPT EQU    0x01

INTCON  EQU     0x0B

        ORG     0               ; Inicio del programa en dirección 0

        bsf     ESTADO,5        ; Banco 1
        movlw   b'11010111'     ; Valor a cargar en
                                ; OPTION: Prescaler x255
        movwf   TMR0_OPT
        movlw   0x00
        movwf   PUERTAB         ; La Puerta B salida
        bcf     ESTADO,5        ; Banco 0
        clrf    PUERTAB         ; Las líneas de salida de PB a 0
parpa   bsf     PUERTAB,7       ; Enciende el led RB7 = 1
        call    retardo         ; Llamada a subrutina de RETARDO
        bcf     PUERTAB,7       ; Apaga el led, RB7 = 0
        call    retardo
        goto    parpa
```

```
retardo   clrf      TMR0_OPT     ; TMR0 = 0 y empieza su incremento
explora   btfss     INTCON,2     ; TMR0<6> = 1?
          goto      explora      ; No ha llegado TMR0 a 255(10)
          bcf       INTCON,2     ; Pone el flag a 0 por software
          return                 ; Ha llegado TMR0 al valor 255(10)
                                 ; y retorna al programa principal
          END
```

El señalizador T0IF debe ser borrado por software; de ahí la instrucción bcf INTCON,2 antes de retornar de la subrutina de espera.

¿SE ATREVE CON ESTE EJERCICIO?

Supongamos que se desea una temporización de un segundo. Ya hemos visto que a 4 MHz, utilizando el mayor valor para el Divisor de frecuencias y para el temporizador, lo máximo que hemos conseguido ha sido una temporización de 65 ms; valor muy lejano al deseado.

Una posibilidad estaría en reducir la frecuencia de funcionamiento del PIC, pero vamos a fijarla en 4 MHz, que es la del cristal que lleva el Micro'PIC Trainer. Le proponemos una solución en el disquete bajo el nombre QUINTO5.ASM. Una pista: si consigue temporizar 8,192 ms (cosa que ya hemos hecho) 122 veces, tendrá 999,424 ms, prácticamente un segundo.

INTERRUPCIONES, RESET Y RECURSOS AUXILIARES



6.1. IMPORTANCIA DE LAS INTERRUPCIONES

Las llamadas a subrutinas mediante la instrucción CALL son desviaciones del flujo de control del programa originadas por instrucciones, por lo que se consideran síncronas. Se producen cada vez que se ejecuta dicha instrucción.

Las interrupciones son desviaciones del flujo de control del programa originadas asíncronamente 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 patita 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. En ambos casos se detiene la ejecución del programa en curso, se salva la dirección actual del PC en la Pila y se carga el PC con una dirección, que en el caso de CALL viene acompañando a la propia instrucción, y en el caso de una interrupción es una dirección «reservada» de la memoria de código, llamada Vector de Interrupción.

En los PIC16X8X el Vector de Interrupción se halla situado en la dirección 0004 H, 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.

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 valores guardados y se restaura completamente el estado del procesador. Algunos procesadores salvan estos registros en la Pila, pero los PIC no disponen de instrucciones para meter (*push*) y sacar (*pop*) información de la Pila, utilizando para este fin registros de propósito general de la memoria de datos.

Los PIC16X8X pueden ser interrumpidos por 4 causas diferentes, pero todas ellas desvían el flujo de control a la dirección 0004 H, 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 en la RSI de los PIC16X8X es que estos microcontroladores poseen un bit GIE (*Global Interrupt Enable*) que cuando vale 0 prohíbe todas las 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 interrupción 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. En la Figura 6.1 se muestra un organigrama de las fases más importantes que se desarrollan durante el proceso de ejecución de una interrupción.

6.2. CAUSAS DE INTERRUPCIÓN

Los PIC16X8X tienen 4 causas o fuentes posibles de interrupción:

1.ª *Activación de la patita RB0/INT*
2.ª *Desbordamiento del temporizador TMR0*
3.ª *Cambio de estado en una de las 4 patitas de más peso (RB7:RB4) de la Puerta B*
4.ª *Finalización de la escritura en la EEPROM de datos*

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 0004 H, 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.

6.2.1. El Registro de Control de Interrupciones INTCON

La mayor parte de los señalizadores y bits de permiso de las fuentes de interrupción en los PIC16X8X están implementados sobre los bits del registro INTCON, que ocupa la dirección 0B H del banco 0, hallándose duplicado en el banco 1. (Véase Figura 6.2.)

GIE: Permiso Global de Interrupciones

1: Permite la ejecución de todas las interrupciones, cuyos bits de permiso individuales también las permitan.
0: Prohíbe todas las interrupciones.

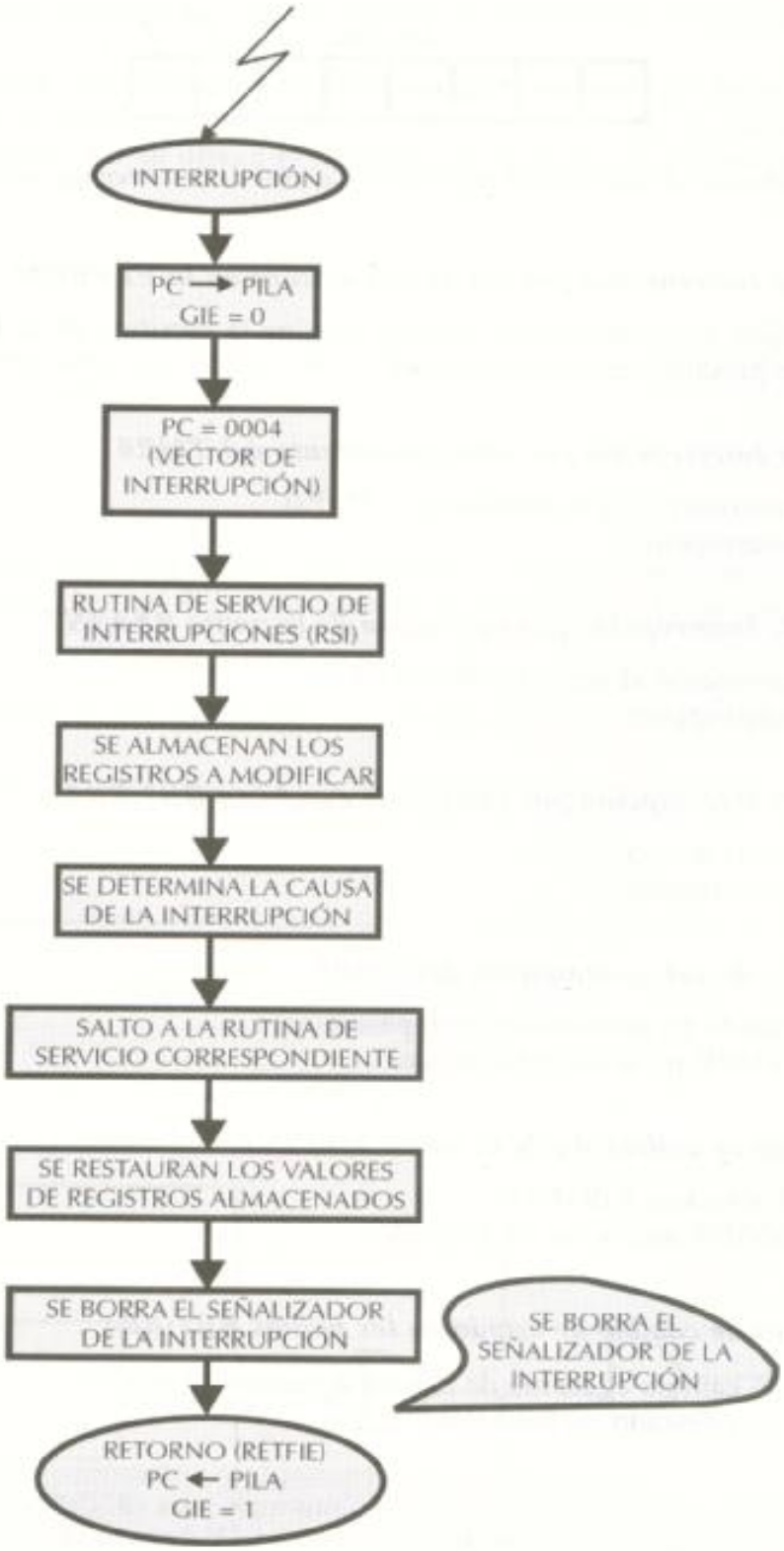


Figura 6.1. *Organigrama de las operaciones principales de una interrupción en los PIC16X8X. El señalizador de la fuente de interrupción hay que borrarlo antes del retorno para que no se repita la misma interrupción.*



Figura 6.2. *Distribución de los bits del registro de Control de las Interrupciones en los PIC16X8X.*

EEIE: Permiso de Interrupción por fin de la escritura en la EEPROM

1: Permite se origine una interrupción cuando termina la escritura de la EEPROM de datos.
0: Prohíbe que se produzca esta interrupción.

T0IE: Permiso de Interrupción por sobrepasamiento del TMR0

1: Permite una interrupción al desbordarse el TMR0.
0: Prohíbe esta interrupción.

INTE: Permiso de Interrupción por activación de la patita RB0/INT

1: Permite la interrupción al activarse RB0/INT.
0: Prohíbe esta interrupción.

RBIE: Permiso de interrupción por cambio de estado en RB7:RB4

1: Permite esta interrupción.
0: Prohíbe esta interrupción.

T0IF: Señalizador de sobrepasamiento del TMR0

1: Se pone a 1 cuando ha ocurrido el sobrepasamiento.
0: Indica que el TMR0 no se ha desbordado.

INTF: Señalizador de activación de la patita RB0/INT

1: Se pone a 1 al activarse RB0/INT.
0: Indica que RB0/INT aún no se ha activado.

RBIF: Señalizador de cambio de estado en las patitas RB7:RB4

1: Pasa a 1 cuando cambia el estado de alguna de estas 4 líneas.
0: No ha cambiado el estado de RB7:RB4.

Cuando GIE = 0 no se acepta ninguna de las interrupciones. Si GIE = 1 sólo se aceptan aquellas fuentes de interrupción cuyo bit de permiso se lo consienta.

Siempre que se produzca una interrupción por cualquier causa, GIE = 0 y el PC se carga con el valor 0004 H, que es el Vector de Interrupción. Para conocer qué causa ha provocado la interrupción se exploran los señalizadores, tres de los cuales se ubican en el registro INTCON

y el cuarto, EEIF, que se pone a 1 cuando finaliza la escritura de la EEPROM, se halla en el bit 4 del registro EECON1.

Los señalizadores 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 6.3 se ofrece el esquema de la lógica de control que origina la interrupción.

EJEMPLO

Confeccionar un programa que explore de forma continua las 4 fuentes de interrupción para averiguar cuál se ha activado.

SOLUCIÓN

Se usa la instrucción btfss registro,n, *que explora el bit «n» de «registro» y si vale 1 se salta una instrucción, o sea, el PC se incrementa en 2 unidades. A esta operación se denomina brinco.*

```
*inicio del servicio de interrupción por activación de RB0/INT

int         btfss intcon, intf    ; Explora el bit intf de
                                  ; intcon y si vale 1 hay brinco
            goto puertab          ; Salta a puertab

*inicio del servicio de interrupción por cambio en RB7 : RB4

puertab     btfss intcon,rbif     ; Explora el bit "rbif" de
                                  ; intcon y si vale 1 se produce brinco
            goto tmr0
```

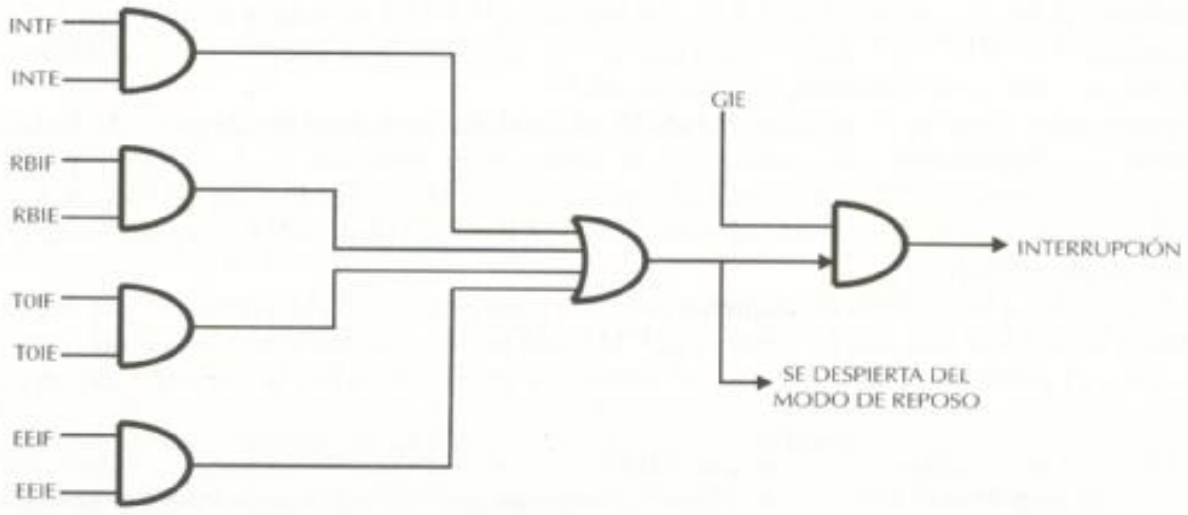


Figura 6.3. *Lógica de control para la generación de una interrupción en los PIC16X8X.*

```
* inicio del servicio de interrupción por desbordamiento de TMR0

tmr0        btfss intcon,t0if     ; Explora el bit toif y si
                                  ; vale 1 brinco
            goto eeprom

* inicia el servicio de interrupción por fin de la escritura en la EEPROM

eeprom      btfss eecon1,eeif     ; Explora el bit eeif del
                                  ; registro eecon1 y si
                                  ; vale 1 hay brinco
            goto int              ; Vuelve al principio
```

EJEMPLO

Confeccionar un programa que asegure que todas las interrupciones se hallan prohibidas.

SOLUCIÓN

Se comprueba que el bit GIE = 0. Se utiliza la instrucción que examina un bit y si vale 1 produce un brinco o un salto de la instrucción siguiente.

```
bucle   bcf intcon,gie      ; GIE = 0
        btfsc intcon,gie    ; Si GIE = 0, entonces hay brinco
        goto bucle
        (continúa)
```

6.2.2. Interrupción Externa INT

Esta fuente de interrupción es sumamente importante para atender acontecimientos externos en tiempo real. Cuando ocurre alguno de ellos activa la patita RB0/INT y se hace una petición de interrupción. Entonces, de forma automática, el bit INTF = 1 y, si el bit de permiso INTE = 1, se autoriza el desarrollo de la interrupción.

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

El procesador explora el señalizador INTF al final del primer ciclo de reloj de cada ciclo de instrucción. Recuérdese que cada ciclo de instrucción constaba de 4 ciclos de reloj: Q1, Q2, Q3 y Q4. Al terminar Q1 se exploran los señalizadores produciéndose un período de latencia de 3 o 4 ciclos de instrucción desde el momento que hay un señalizador activado hasta que se inicializa la interrupción.

En la Figura 6.4 se muestra un esquema práctico para generar una interrupción por activación en la patita RB0/INT. Los dos inversores del 74LS04 realimentados constituyen un filtro antirebotes para el pulsador, y la red R-C y el último inversor controlan la duración del impulso.

Si INTE = 1 y se aplica un flanco activo en RB0/INT, el señalizador INTF se pone a 1 automáticamente y se solicita una interrupción, que es aceptada cuando GIE = 1. Antes de regresar al programa principal hay que borrar INTF, puesto que en caso contrario al ejecutar la instrucción de retorno RETFIE se volvería a desarrollar el mismo proceso de interrupción.

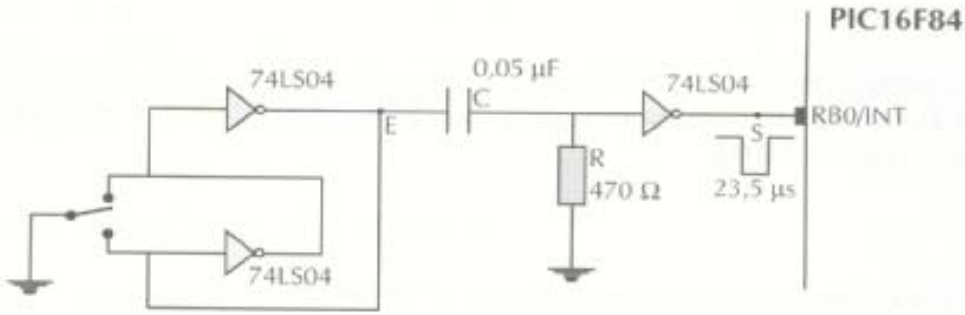


Figura 6.4. *Circuito para generar impulsos de duración determinada. En el esquema Tiempo = 470 · 0,05 = 23,5 µs.*

EJEMPLO

Confeccionar un programa para permitir que se acepte la interrupción por activación de la patita RB0/INT al aplicarla un flanco descendente.

SOLUCIÓN

El programa deberá encargarse de que los bits del registro INTCON tomen los siguientes valores: INTE = GIE = 1 y T0IE = RBIE = EEIE = 0. También deberá asegurarse de que inicialmente, antes de producirse la interrupción, el señalizador INTF valga 0.

```
bsf     estado,rp0      ; Selección del banco 1
bcf     option,intdeg   ; INTDEG = 0 para flanco descendente
bcf     estado,rp0      ; Selección banco 0
bsf     intcon,gie      ; GIE = 1
bsf     intcon,inte     ; INTE = 1
bcf     intcon,rbie     ; RBIE = 0
bcf     intcon,eeie     ; EEIE = 0
bcf     intcon,intf     ; INTF = 0
```

Se podía haber cargado todos los bits del registro INTCON con una instrucción.

6.2.3. Interrupción por desbordamiento del TMR0

Cuando TMR0 se desborda y pasa del valor FF H al 00 H, el señalizador T0IF se pone automáticamente a 1. Si, además, el bit de permiso de la interrupción del TMR0 T0IE = 1 y el bit de Permiso Global de Interrupciones GIE = 1, se produce una interrupción.

Si no se recarga el TMR0 cuando se desborda, sigue contando desde 00 H a FF H. En cualquier momento se puede leer y escribir este registro, pero cada vez que se escribe se pierden dos ciclos de reloj para la sincronización.

Cuando se carga inicialmente TMR0 con el valor N_{10}, cuenta 256 – N impulsos, siendo el tiempo que tarda en hacerlo el que expresa la siguiente fórmula:

Temporización = 4 · Tosc · (256 – N_{10}) · Rango del Divisor de Frecuencia

EJEMPLO

¿Cada cuánto tiempo producirá TMR0 una interrupción si se le carga continuamente con el valor 40_{10}, el Divisor de frecuencia está programado en el rango 1:8 y el microcontrolador dispone de un cristal de 1 MHz?

SOLUCIÓN

Para averiguar el tiempo de retardo se aplica la fórmula:

$$Temporización = 4 \cdot Tosc \cdot (256 - N_{10}) \cdot Rango\ del\ Divisor\ de\ Frecuencia$$
$$Temporización = 4 \cdot 1\ \mu s \cdot (256 - 40) \cdot 8 = 6{,}192\ \mu s$$

6.2.4. Interrupción por cambio de estado en las líneas RB7 : RB4 de la Puerta B

Esta interrupción está diseñada específicamente para detectar la pulsación de una tecla correspondiente a un teclado matricial que se explora con 4 líneas de E/S. Para esta función se destinan las líneas RB7 : RB4 de la Puerta B, que cada vez que cambia el estado lógico de una de ellas fuerza al señalizador RBIF a ponerse a 1, y si los bits de permiso RBIE = GIE = 1 se autoriza la interrupción.

6.2.5. Interrupción por finalización de la escritura en la EEPROM de datos

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 = 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 55 H y el AA H. 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 trozo 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.

```
bsf     estado,rp0      ; Selecciona el banco1
bcf     intcon,gie      ; GIE = 0. Prohíbe interrupciones
bsf     intcon,eeie     ; Permite la interrupción de la EEPROM
bsf     eecon1,wren     ; Permiso de escritura de la EEPROM
movlw   0x55
movwf   eecon2          ; Se escribe en eecon2 el dato 55 h
movlw   0xaa
movw    eecon2          ; Se escribe aa h en eecon2
bsf     eecon1,wr       ; Comiena 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 PIC16CR8X, 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.

6.3. REINICIALIZACIÓN O RESET

Los PIC16X8X tienen cinco causas que provocan la reinicialización del sistema, que consiste en cargar al PC con el valor 000 H (Vector de Reset) y poner el estado de los bits de los registros específicos (SFR) con un valor conocido.

1.ª Conexión de la alimentación. POR («Power on Reset»)
2.ª Activación de la patita MCLR# («Master Clear Reset») en funcionamiento normal
3.ª Activación de la patita MCLR# en estado de Reposo
4.ª Desbordamiento del Perro Guardián en funcionamiento normal
5.ª Desbordamiento del Perro Guardián en estado de Reposo

En la Tabla 6.1 se presenta el estado lógico que adquieren los bits de los registros SFR de la memoria de datos cuando se provoca un Reset por una de las cinco causas posibles.

En la Figura 6.5 se muestra el esquema electrónico de los PIC16X8X para la generación del Reset. La patita MCLR# dispone de un flitro interno para eliminar los ruidos y los impulsos muy pequeños.

El temporizador PWRT («Power-up Timer») activa una salida al cabo de un cierto tiempo tras la conexión de la alimentación, que se aplica a una entrada de la AND4, encargada de controlar el Reset del Flip-flop que controla la generación interna del Reset del sistema. Sólo es

REGISTRO	DIRECCIÓN	CONEXIÓN DE LA ALIMENTACIÓN	DESBORDAMIENTO PERRO GUARDIÁN MODO NORMAL	DESBORDAMIENTO PERRO GUARDIÁN MODO REPOSO	MCLR# MODO NORMAL	MCLR# MODO REPOSO
W	——	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
INDIR	00 h	-----	-----	-----	-----	-----
RTCC	01 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PC	02 h	0000 h	0000 h	PC + 1	0000 h	0000 h
STATUS	03 h	0001 1xxx	0000 1uuu	uuu0 0uuu	000u uuuu	0001 0uuu
FSR	04 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT A	05 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PORT B	06 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TRIS A	85 h	—1 1111	—1 1111	—u uuuu	—1 1111	—1 1111
TRIS B	86 h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
OPTION	81 h	1111 1111	1111 1111	uuuu uuuu	1111 1111	1111 1111
EEDATA	08 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EEADR	09 h	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
EECON 1	88 h	—0 0000	—0 ?000	—u uuuu	—0 ?000	—0 ?000
EECON 2	89 h	-----	-----	-----	-----	-----
PCLATH	0A h	—0 0000	—0 0000	—u uuuu	—0 0000	—0 0000
INTCON	0B h	0000 000x	0000 000u	uuuu uuuu	0000 000u	0000 0000

u = No cambia x = Indeterminado —— = No Implementado ? = Depende de otras condiciones

Tabla 6.1. *Valor que adquieren los bits de los registros SFR y el W después de producirse un Reset por una de las cinco causas posibles.*

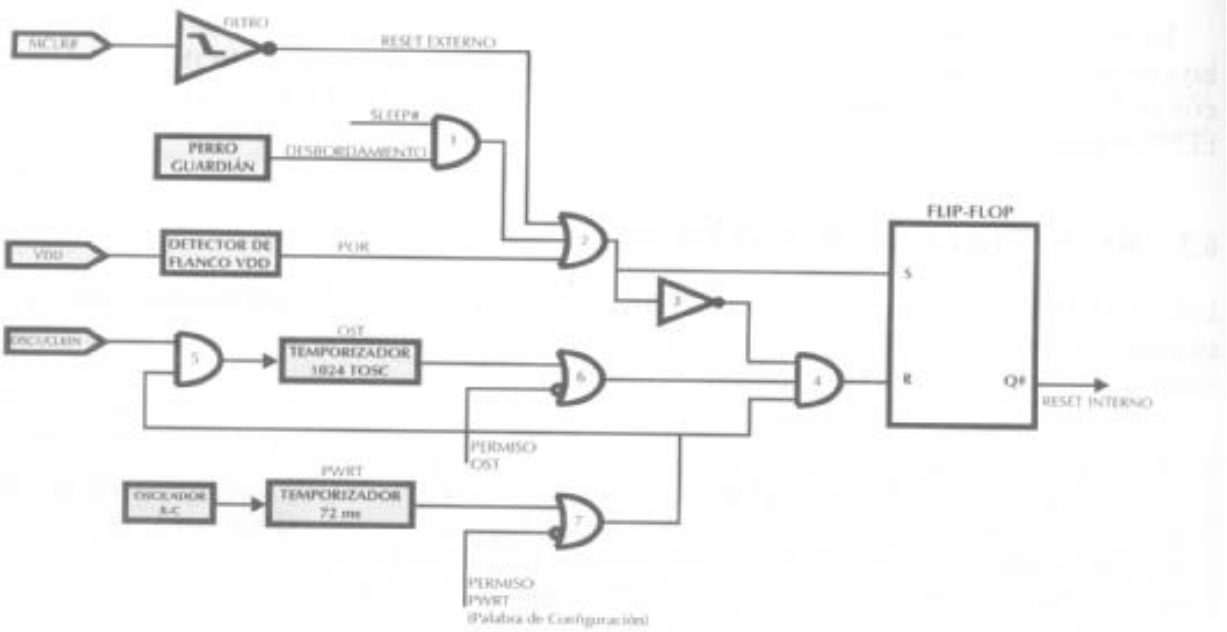


Figura 6.5. *Esquema electrónico para la generación interna del Reset.*

válida la salida de PWRT si el bit de permiso del PWRT está activo a nivel bajo (PWRTE# = 0), dado que también se aplica a la OR7 de la Figura 6.5. El bit PWRTE reside en la Palabra de Configuración y una de las pocas diferencias que tenía el PIC16C84 con el PIC16F84 era que aquél tenía como nivel activo del bit PWRTE el alto, mientras que este último tiene el bajo.

El Reset del Flip-flop final se produce cuando la AND4 saca nivel alto, lo que requiere que sus tres entradas tengan dicho nivel, y eso supone:

a) Que no haya peticiones de Reset y la puerta OR2 tenga su salida a nivel bajo.
b) Que si está activo PWRT finalice su retardo de 72 ms.
c) Que también finalice la temporización del oscilador OST, que retarda 1.024 · Tosc, tras completarse el retardo de PWRT.

El temporizador PWRT proporciona un retardo fijo de 72 ms y sus impulsos de reloj los genera un oscilador R-C propio. Este tiempo garantiza la estabilización del voltaje de alimentación V_{DD}.

El temporizador OST (*Oscillator Start-up Timer*) proporciona un retardo de 1.024 · Tosc (Periodo de los impulsos aplicados en la patita OSC1/CLKIN). Sirve para asegurar que el cristal de cuarzo o resonador cerámico empleado en los osciladores tipo XT, LP o HS esté estabilizado y en marcha. OST comienza a funcionar cuando termina el retardo de PWRT debido a la conexión de la salida de la OR7 con la AND5.

La activación de la entrada Set del Flip-flop se consigue cuando se activa la patita MCLR#, cuando se desborda el Perro Guardián o cuando se detecta un flanco ascendente en la patita V_{DD} (POR).

En la Figura 6.6 se ofrece un cronograma de las principales señales que participan en la generación del Reset y en el que se aprecia la secuencia de los retardos T_{PWRT} y T_{OST}.

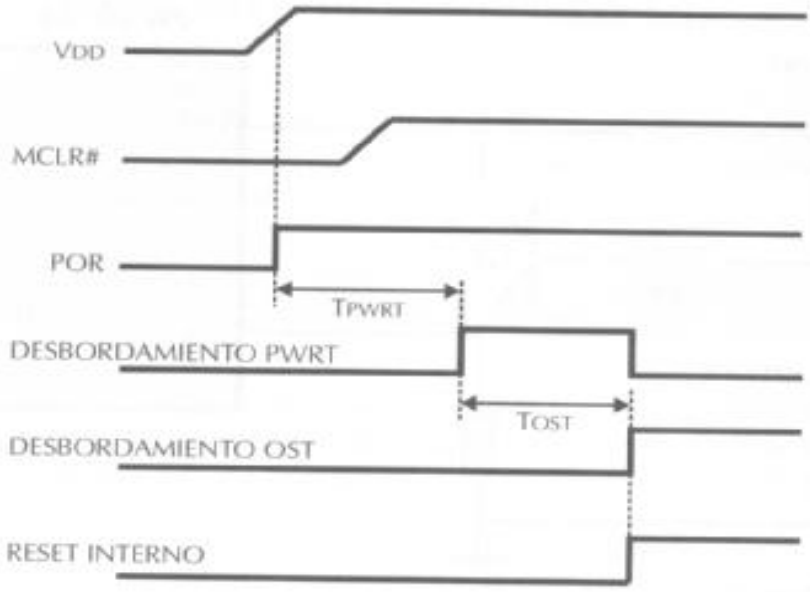


Figura 6.6. *Cronograma de las principales señales que intervienen en el Reset para uno de los posibles casos en que MCLR# no cambia de estado con* V_{DD}.

En el registro ESTADO hay dos bits que indican las condiciones en las que se ha originado el Reset. Se trata de TO# (*Timer Out*) y PD# (*Power Down*). (Figura 6.7.)

6.3.1. Reset por fallo en la alimentación («Brown Out»)

Se produce un fallo en la alimentación cuando el voltaje de alimentación V_{DD} desciende por debajo del valor mínimo, sin llegar a cero, y luego se recupera. En esta situación es preciso provocar un Reset.

Para generar un Reset en un PIC16X8X cuando hay un fallo en la alimentación hay que colocar un circuito externo de protección, como los dos que se muestran en las Figuras 6.8 y 6.9.

TO#	PD#	CONDICIÓN DE RESET
1	1	POR (RESET POR CONEXIÓN VDD)
0	1	DESBORDAMIENTO WDT EN FUNCIONAMIENTO NORMAL
0	0	DESBORDAMIENTO WDT EN ESTADO DE REPOSO
1	1	ACTIVACIÓN MCLR# EN FUNCIONAMIENTO NORMAL
1	0	ACTIVACIÓN MCLR# EN REPOSO

Figura 6.7. *Condiciones de Reset reflejadas por el estado de los bits TO# y PD# del registro ESTADO.*

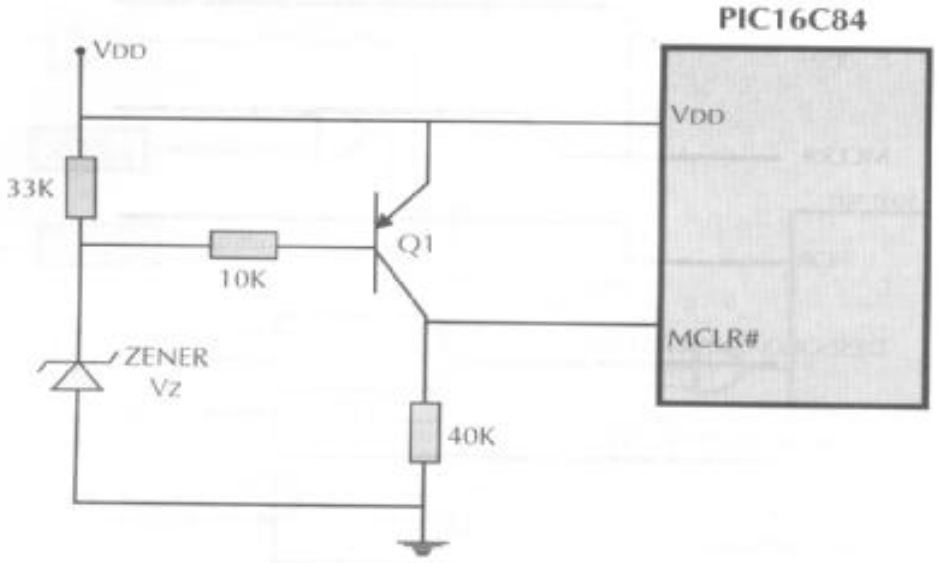


Figura 6.8. *Cuando V_{DD} desciende por debajo del valor Vz + 0,7 V, se produce un Reset por la activación de la patita MCLR#.*

La patita MCLR# puede conectarse directamente con V_{DD}, pero debe colocarse una resistencia de valor superior a 100 Ω cuando se conecta con tierra, para evitar los problemas que puede originar la corriente que circula.

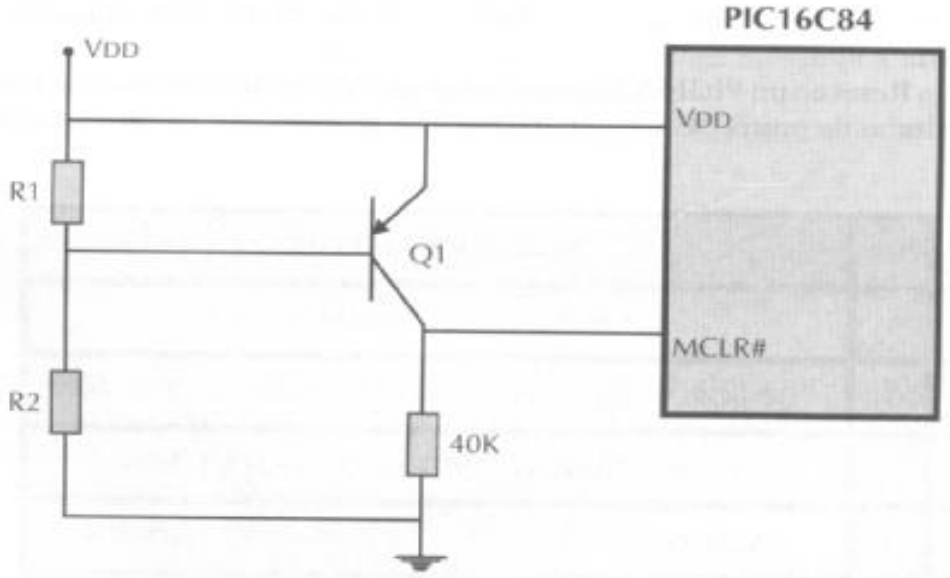


Figura 6.9. *El transistor Q1 se bloquea y activa el Reset al pasar a nivel bajo la patita MCLR#, cuando el valor V_{DD} desciende por debajo de $V_{DD} \cdot R1 / (R1 + R2) = 0,7$ V.*

6.4. EL MODO DE REPOSO O DE BAJO CONSUMO

Este modo de funcionamiento de los PIC está caracterizado por el reducido consumo de energía que requiere y está muy recomendado en aquellas aplicaciones en las que hay largos periodos de espera hasta que se produzca algún suceso asíncrono, como la pulsación de una tecla. En dichos periodos el procesador está inactivo.

El consumo típico del PIC es de 2 mA, aproximadamente, reduciéndose a menos de 10 µA en el modo Reposo, lo que permite alimentarle con una pequeña pila durante casi dos años.

Para entrar en el modo de Reposo hay que ejecutar la instrucción SLEEP y se produce una situación muy especial de funcionamiento que parece como si el sistema se ha «congelado», requiriendo el mínimo suministro de energía para mantener el estado del procesador sin ninguna actividad.

En Reposo la patita T0CKI se conecta a V_{DD} o a tierra, para eliminar la entrada de impulsos externos al TMR0. Por otra parte, como se detiene el oscilador principal que genera los impulsos Tosc, también se para TMR0. Las patitas de E/S mantienen el estado anterior al modo de Reposo y las que no se hallan conectadas a periféricos y actúan como entradas de alta impedancia se aconseja conectarlas a V_{DD} o a tierra para evitar posibles fugas de corriente. La patita MCLR# debe conectarse a nivel alto. Sin impulsos de reloj el procesador se congela y deja de ejecutar instrucciones hasta que se «despierte» y salga de ese estado.

Si el Perro Guardián continúa activo en el modo de Reposo, al entrar en él se borra, pero sigue funcionando. Los bits del registro de ESTADO PD# y TO# toman los valores 0 y 1, respectivamente.

Para salir del estado de Reposo (despertar) existen tres alternativas:

1.ª Activación externa de MCLR# para provocar un Reset.
2.ª Desbordamiento del Perro Guardián si quedó operativo en el modo de Reposo.
3.ª Generación de una interrupción. En este caso, como TMR0 está parado sólo pueden producirse los otros tres tipos de interrupción.

Cuando se despierta el PIC se desarrolla la secuencia del oscilador OST, que retarda 1.024 Tosc para estabilizar la frecuencia de trabajo, y luego se pasa a ejecutar la siguiente instrucción a *sleep* (PC + 1).

Los bits TO# y PD# se emplean para conocer la causa del Reset que despierta al sistema. PD# = 0 cuando se ejecuta la instrucción SLEEP. TO# = 0 cuando se desborda el Perro Guardián.

6.5. PROGRAMACIÓN EN SERIE DE LOS PIC16X8X

Los PIC que estamos estudiando permiten que se grabe en su memoria de código el programa sobre el mismo circuito de la aplicación. Esto permite a los fabricantes de productos finales construir la tarjeta electrónica con el microcontrolador sin grabar y realizar la escritura del programa justo cuando se realiza la venta. Así se puede aportar el firmware más reciente que se disponga.

La programación en serie de los PIC16X8X requiere 5 líneas, dos de las cuales se destinan a soportar el reloj de la transmisión (RB6) y los bits de datos serie (RB7), mientras que las otras tres soportan V_{DD}, tierra y la tensión especial de programación V_{PP}, que oscila entre 12 y 14 V. (Figura 6.10.)

El microcontrolador pasa al modo de programación/verificación colocando a nivel bajo las líneas RB6 y RB7 y aplicando la tensión de programación V_{PP}. A partir de ese momento por RB6 se transmiten los impulsos de reloj y por RB7 se transfieren los bits de datos en serie.

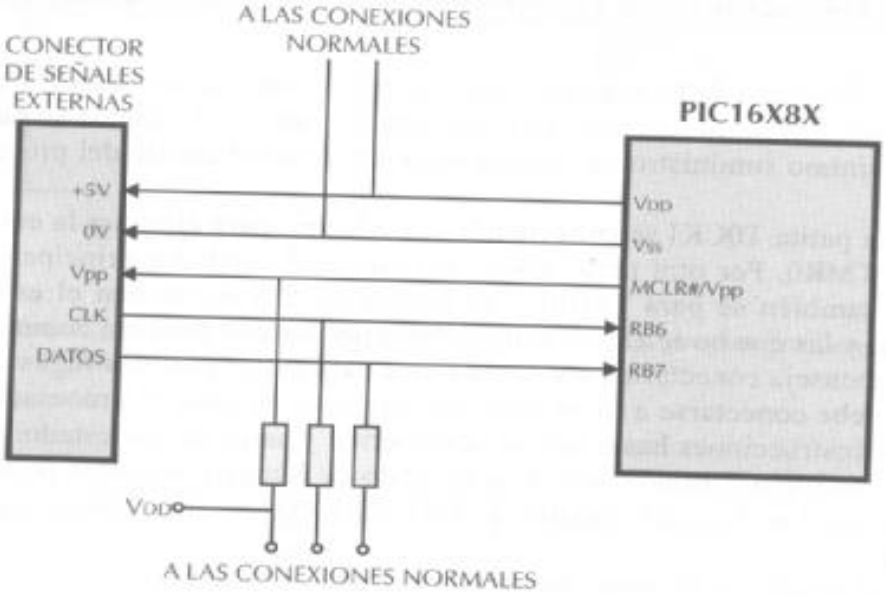


Figura 6.10. *Tipico esquema de conexionado de un PIC16X8X para su grabación del programa en serie, sobre el circuito final de la aplicación.*

PROGRAMAR PIC ES FÁCIL

Las interrupciones constituyen un recurso esencial en la resolución de las aplicaciones reales y los PIC16X8X disponen de 4 fuentes capaces de producir interrupción:

1.ª *Flanco activo sobre la patita RB0/INT*
2.ª *Desbordamiento del TMR0*
3.ª *Cambio de estado en una de las patitas RB7 : RB4*
4.ª *Finalización de un ciclo de escritura en la EEPROM de datos*

La interrupción provocada por la aplicación de un flanco en la patita RB0/INT es muy eficaz para controlar cualquier suceso asíncrono externo. La interrupción producida por el cambio de estado en una de las 4 patitas de más peso de la Puerta B está diseñada para controlar periféricos específicos, como el teclado. La que controla el fin de la escritura en la EEPROM de datos es muy necesaria cuando se guarda información en esta memoria no volátil, porque el tiempo de escritura es relativamente largo (10 ms) y variable. Finalmente, la interrupción más empleada es la que origina el desbordamiento del temporizador/contador TMR0, porque con ella se puede controlar con mucha exactitud un período de tiempo, sin desviar la atención del procesador.

El ejercicio que se presenta en este capítulo dispone de una tarea principal muy simple y de otra auxiliar basada en el cambio de estado de un periférico cada cierto tiempo. Existe un retardo que se controlará con la interrupción producida por TMR0 y así el procesador se dedicará a la tarea principal, excepto cuando le avise el TMR0, instante en el que se desviará temporalmente a atender al otro periférico.

EL SEXTO EJERCICIO PRÁCTICO: MANEJANDO INTERRUPCIONES

Se trata de confeccionar un programa en lenguaje ensamblador para un PIC16X84, cuyo oscilador interno se halla controlado por un cristal de cuarzo funcionando a 4 MHz. Conectados a las líneas RA0 y RA1 de la Puerta A existen dos interruptores, A y B, los cuales tienen que ser explorados continuamente y reflejar su estado (abiertos o cerrados) sobre dos diodos led, A' y B', conectados a las líneas RB0 y RB1 de la Puerta B.

Además existe otro diodo led en la línea RB7 que está parpadeando continuamente, pasando cada segundo por los estados de encendido y apagado.

SOLUCIÓN

En la Figura 6.11 se ofrece el esquema general de conexionado del PIC16X84 que responde a las especificaciones del ejercicio.

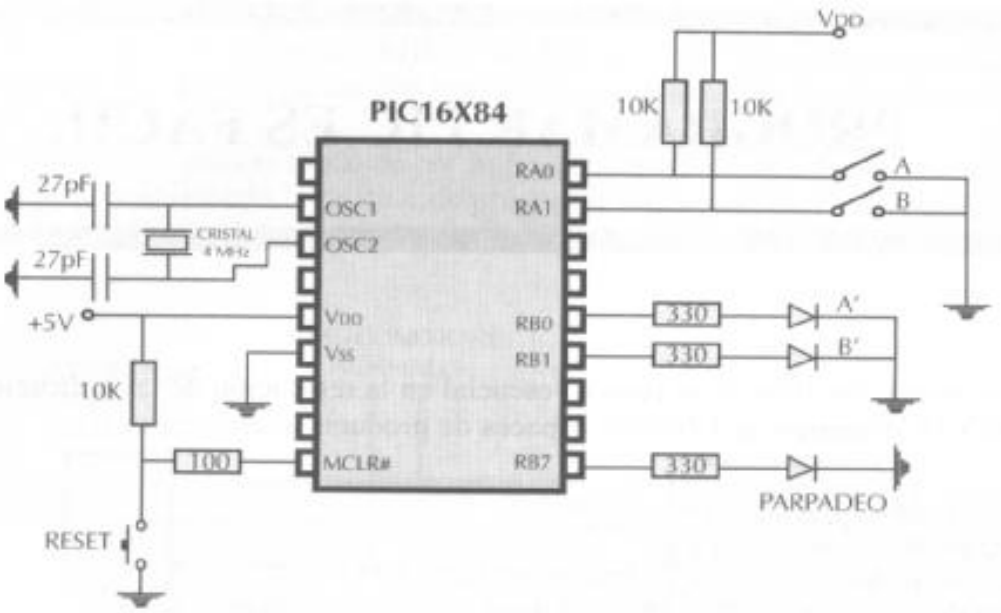


Figura 6.11. *Esquema electrónico de conexionado del PIC16X84.*

Para controlar el retardo de 1 segundo para el parpadeo del led conectado en la linea RB7, se generará una interrupción cada dicho tiempo por desbordamiento del TMR0.

Inicialmente se cargará al TMR0 con el valor 12_{10}, con lo que el desbordamiento se producirá al cabo de 244 (256 – 12) impulsos aplicados a dicho temporizador. Dichos impulsos serán los del oscilador interno Tosc, pero divididos por 256 en el Divisor de frecuencia, con lo cual la temporización total será:

```
Temporización = 4 · Tosc · Valor TMR0 · Rango Divisor Frecuencia
Temporización = 4 · 250 ns · 244 · 256 = 62,4 ms
```

Como no se alcanza el retardo de 1 segundo se emplea un contador auxiliar CONTA que al cargarse con un valor de 16_{10} y decrementarse una unidad cada 62,4 ms, cuando llegue a 0 conseguirá, aproximadamente, el tiempo buscado (62,4 ms x 16).

En el programa que resuelve el ejercicio se ha utilizado como instrucción clave la *btfss registro,n*, que explora el valor del bit *n* de *registro* y si vale 1 se salta la siguiente instrucción brinco, o sea, el PC se incrementa en 2 unidades. La instrucción inversa es *btfsc registro,n*, que realiza el brinco si el bit examinado vale 0.

PROGRAMA PARPADEO

SEXTO.ASM

```
        LIST    P=16C84
        RADIX   HEX
```

```
        W       EQU   0
        F       EQU   1
        TMR_OPT EQU   0x01   ; TMR0 en banco 0 OPTION en banco 1
        ESTADO  EQU   0x03
        PUERTAA EQU   0x05   ; PA en banco 0 TRISA en banco1
        PUERTAB EQU   0x06   ; PB en banco 0 TRISB en banco1
        INTCON  EQU   0x0B
        CONTA   EQU   0x10   ; Contador auxiliar
```

```
        ORG     0            ; Vector de Reset
        goto    inicio
        ORG     4            ; Vector de Interrupción
        goto    inter        ; Salta a comienzo de rutina
                             ; de interrupción
        ORG 5
```

```
inicio  bsf     ESTADO,5     ; Selección del banco 1
        clrf    PUERTAB      ; Configura PUERTA B como salida
        movlw   b'00000011'  ; Configura RA0, RA1 como entradas
        movwf   PUERTAA
        movlw   b'00000111'
        movwf   TMR_OPT
        bcf     ESTADO,5     ; Banco 0
        movlw   b'10100000'  ; Se permite interrupción del
                             ; TMR0 y la global (GIE)
        movwf   INTCON
        movlw   0x10
        movwf   CONTA        ; Se carga CONTA con 16 decimal
        movlw   0x0c
        movwf   TMR_OPT      ; Se carga TMR0 con 12 decimal
bucle   btfsc   PUERTAA,0    ; Explora RA0 y brinco si 0
                             ; salta a RA0_1
        goto    ra0_1
        bcf     PUERTAB,0    ; Si RA0 = 0 RB0 = 0
        goto    ra1x         ; A explorar RA1
ra0_1   bsf     PUERTAB,0    ; Si RA0 = 1 se saca por RB0 un 1
ra1x    btfsc   PUERTAA,1    ; Examina ra1 y brinco si 0
        goto    ra1_1        ; Salta si RA1 = 1
        bcf     PUERTAB,1    ; Si RA1 = 0, RB1 = 0
        goto    bucle2
ra1_1   bsf     PUERTAB,1    ; Si RA1 = 1, RB1 = 1
bucle2  goto    bucle        ; Bucle indefinido, se sale
                             ; por la interrupción
```

```
inter   decfsz  CONTA,1      ; RSI. Decrementa CONTA y
                             ; brinco si vale 0
        goto    seguir
conta_0 movlw   0x10         ; Si CONTA = 0 se carga
        movwf   CONTA
        btfsc   PUERTAB,7    ; Si RB7 = 0, brinco
        goto    rb7_1
        bsf     PUERTAB,7    ; Si RB7 = 0, se invierte
```

```
            goto    seguir
rb7_1       bcf     PUERTAB,7     ; Si RB7 = 1, se invierte
seguir      movlw   b'10100000'   ; Se restaura INTCON por desactivar
                                  ; las interrupciones el procesador
            movwf   INTCON
            movlw   0x0c
            movwf   TMR_OPT       ; Se recarga TMR0 con 12
            retfie
            end
```

Si Vd. ha «seguido» nuestras explicaciones sobre el comportamiento de las interrupciones y especialmente la del TMR0, es muy posible que entienda la mayor parte del programa del parpadeo. Es muy difícil que lo comprenda todo, porque siempre hay algo que está mal. Comience a pensar seriamente que aquello que no encaja con sus esquemas puede estar equivocado o tratarse de una errata. Si nos la comunica, le escribiremos agradeciendo su gentileza y trataremos de suprimirla en la próxima edición.

SIMULANDO CON EL SIMUPIC'84

En este capítulo no vamos a incorporar ninguna novedad sustancial en el manejo del **Simupic'84.** Tal y como ya estamos acostumbrados, editaremos, ensamblaremos y cargaremos en memoria el programa de este capítulo, SEXTO.ASM. Como el enunciado del programa nos indica que el PIC debe funcionar a una frecuencia de 4 MHz, debemos ir a la ventana de «Parámetros de simulación» y allí establecer la frecuencia de simulación en 4000 KHz. Aprovecharemos también para desactivar el WDT.

En SEXTO.ASM existen, por una parte, 2 interruptores conectados a RA0 y RA1 que encienden 2 leds que están conectados a RB0 y RB1, y por otra parte un led conectado a RB7 que parpadea cada segundo. Mediante las teclas de función podemos introducir valores por RA0 y RA1 y ver las reacciones de RB0 y RB1 en la ventana de «visión externa».

Una característica muy útil del **Simupic'84** que no ha sido aún comentada es la «Ventana de Contadores». En esta ventana se muestran 2 contadores, uno de tiempo y otro de instrucciones. Ambos contadores pueden ser puestos a cero simultáneamente mediante la opción «Poner a cero contadores» de la ventana «Opciones de simulación». Lógicamente, al comenzar el programa ambos contadores están a cero. Mediante el uso de estos contadores se puede llevar a cabo cualquier temporización, y en nuestro caso comprobar que el led conectado a RB7 parpadea efectivamente cada segundo. Pero el mayor problema con el que nos encontramos a la hora de programar y simular temporizaciones con **Simupic'84**, o con cualquier otro simulador software en general, es el problema de la velocidad de simulación, ya que ésta es muchas veces inferior a la de un PIC real.

Junto con los contadores de tiempo e instrucciones, en la ventana de contadores también nos encontramos con un indicador que muestra el modo de funcionamiento actual de PIC, es decir, si se encuentra en modo de ejecución normal o en modo de Reposo.

PROGRAMANDO EN C

La principal novedad de este programa con respecto a los anteriores, programados también en lenguaje C, es la incorporación de la rutina que maneja la interrupción por desbordamiento del

Timer 0 y las directivas del compilador para la gestión de las interrupciones. El programa en cuestión es el siguiente:

```
/* SEXTO.C programa que controla 2 leds conectados a RB0 */
/* RB1 que reflejan el estado de 2 interruptores conectados */
/* a RA0 y RA1. */
/* Por otra parte, conectado a RB/ hay un led que parpadea */
/* cada segundo. */
```

```
#include <16c84.h>                    /* Selección del PIC a usar */
#use delay(clock= 4000000)            /* Reloj de 4 MHz */
#BYTEpuertab = 6
#define máximo 16                     /* Constante: máximo = 16 */

byte conta;
```

```
/* SERVICIO DE LA INTERRUPCIÓN   */
#INT_RTCC                                       /* Interrupción por
                                                   desbordamiento
                                                   del timer */

clock_isr(){
short valor;
short novalor;
valor = bit_test(puertab,7);                    /* El bit valor es igual
                                                   al contenido de
                                                   RB7 */

novalor = !valor;
      if (--conta==0) {
            output_bit (PIN_B7,novalor);        /* Voltea RB7 */
            delay_us (5);
            conta = máximo;
      }
}
```

```
/* PROGRAMA PRINCIPAL */

main()      {
      short ra0,ra1;
      set_tris_b(0x00);                         /* Se inicializa PB
      conta=máximo;
      set_rtcc(0);                              /* Se inicializa el
                                                   timer A cero*/
      setup_counters(rtcc_internal,
                     rtcc_div_256);             /* Divisor = 256*/
      enable_interrupts(rtcc_zero);             /* Se permite int */
                                                /* del tmr0 */
      enable_interrupts(GLOBAL);                /* Se activa GIE */
```

```
        /* Rutina de control de los leds e interruptores: */
        do {
             ra0 = input(PIN_A0);
             ra1 = input(PIN_A1);

             output_bit (PIN_B0,ra0);
             output_bit (PIN_B1,ra1);
        }
             WHILE (TRUE);                    /* Bucle infinito */
}
```

LE PROPONEMOS UN EJERCICIO LLENO DE INTERRUPCIONES

El siguiente programa controla una alarma conectada a las cuatro puertas de un coche. Se utiliza la interrupción por cambio de estado en una de las patitas RB7 : RB4 y la interrupción por flanco activo sobre la patita RB0/INT.

El diseño práctico de esta aplicación sería algo complicado, por lo que nuestro objetivo será únicamente mostrar al lector el programa en ensamblador para que se familiarice con el uso de interrupciones. El programa se llama ALARMA.ASM.

```
; ALARMA.ASM Un PIC 16C84 controla la alarma de un coche:
; Conectados a RB4 - RB7 hay 4 sensores que controlan cada
; uno una puerta del coche. Cuando una de las puertas del
; coche es abierta, su sensor manda un 1 por su línea
; respectiva cambiando ésta de estado, con lo que se provoca
; una interrupción y comienzan a sonar 2 zumbadores
; conectados a RA0 y RA1.
; Para detener la alarma, el usuario debe mandar una señal
; infrarroja a un sensor, que cuando la detecta activa la
; patita RB0 con lo que se produce una interrupción, los
; zumbadores se paran y el programa vuelve a su comienzo.
```

```
        LIST    P=16C84
        RADIX   HEX
```

```
W       EQU     0
F       EQU     1
ESTADO  EQU     0x03
PUERTAA EQU     0x05
PUERTAB EQU     0x06
INTCON  EQU     0x0B
```

```
        ORG     0
        goto    inicio
        ORG     4
        goto    inter
        ORG     5

inicio  bsf     ESTADO,5      ; Banco 1
        movlw   b'00000000'
        movwf   PUERTAA       ; PUERTAA salidas
        movlw   b'11111111'
        movwf   PUERTAB       ; PUERTAB entradas
        bcf     ESTADO,5      ; Banco 0
        clrf    PUERTAA
        clrf    PUERTAB
        movlw   b'10011000'   ; Se activan GIE, Int externa
                              ; y Int por cambio de PB
        movwf   INTCON
bucle   goto    bucle
```

```
inter   btfss   INTCON,0      ; Explora flag de int por
                              ; cambio RB4 RB7
        goto    parar
        goto    alarma
alarma  clrf    PUERTAB
        movlw   b'10011000'   ; Se activan GIE, Int externa
        movwf   INTCON        ; y Int por cambio de PB
buzzer  bsf     PUERTAA,0
        NOP
        bcf     PUERTAA,0
        bsf     PUERTAA,1
        NOP
        bcf     PUERTAA,1
        goto    buzzer
parar   clrf    PUERTAA
        bcf     PUERTAB,0
        movlw   b'10011000'   ; Se activan GIE, Int externa
                              ; y Int por cambio de PB
        movwf   INTCON
        goto    bucle
        end
```

Para animarle a trabajar con este programa le advertimos que una de las actividades que más usan los controladores es la que se dedica a las alarmas.

MANEJANDO EL REPERTORIO DE INSTRUCCIONES

7

7.1. REPERTORIO RISC

Todos los modelos de microcontroladores PIC responden a la arquitectura RISC, que significa «Computador de Juego de Instrucciones Reducido». No sólo implica que el número de instrucciones máquina que es capaz de interpretar y ejecutar el procesador es pequeño, como sucede en los PIC16X8X, que consta de 35, sino también que posee las siguientes características:

1.ª *Las instrucciones son simples y rápidas*

La falta de complejidad en la operación que realizan las instrucciones de los procesadores RISC permite que sean ejecutadas, mayoritariamente, en un solo ciclo de instrucción. Los PIC tardan en ejecutar todas las instrucciones un ciclo, excepto las de salto, que tardan el doble.

2.ª *Las instrucciones son ortogonales*

Apenas tienen restricciones en el uso de operandos. Cualquier instrucción puede usar cualquier operando.

3.ª *La longitud de las instrucciones y los datos es constante*

Todas las instrucciones tienen la misma longitud, 14 bits en los PIC16X8X, y todos los datos también, un byte. La arquitectura Harvard del procesador aísla la memoria de instrucciones de la de datos, pudiendo tener diferente tamaño sus palabras.

EJEMPLO

Un PIC16F84 funcionando con un cristal de cuarzo de 10 MHz tarda 600 µs en ejecutar un programa que tiene el 50 por 100 de las instrucciones de salto.

a) *¿De cuántas instrucciones consta el programa?*
b) *¿Cuántas posiciones de la memoria de código ocupa?*

SOLUCIÓN

Todas las instrucciones tardan en ejecutarse un ciclo de instrucción, equivalente a 4 ciclos de reloj, excepto las de salto, que tardan el doble.

CLK = Fosc = $1/10 \cdot 10^6$ s = 100 ns

Ciclo Instrucción = $4 \cdot$ Tosc = 400 ns

$T_{PROGRAMA} = 600\ \mu s = X \cdot 400\ ns + 2 \cdot X \cdot 400\ ns$;

X = 500

Luego hay 500 instrucciones normales y 500 de salto, en total 1.000 instrucciones.

a) *1.000 instrucciones tiene el programa.*
b) *El programa ocupa 1.000 posiciones de la memoria de código.*

7.2. TIPOS DE FORMATO

Las instrucciones de los PIC de la gama media, entre los que se encuentran los modelos PIC16X8X, tienen 14 bits de longitud. Dicho formato se divide en diferentes campos de bits, cada uno de los cuales referencia a operandos o elementos que maneja la instrucción en la operación que realiza en el procesador. Se describen dichos campos:

a) *Campo del código OP*

Los bits de este campo sirven para definir la operación que realiza la instrucción.

b) *Campo de los operandos fuente (f) y destino (d)*

Estos campos de bits definen los registros que actúan como operandos en la instrucción. Suelen referenciar la dirección que ocupan en la memoria de datos.

c) *Campo de operando inmediato o literal (k)*

Es un campo de bits que contiene el valor de un operando inmediato.

d) *Campo que referencia a un bit (b)*

Suele ser un campo de 3 bits que indica la posición de un bit concreto dentro de un registro de 8 bits.

e) *Campo de la dirección del salto*

En las instrucciones de salto CALL y GOTO hay un campo de bits que contiene la dirección de la siguiente instrucción que hay que ejecutar. Dicho campo de bits se carga en el PC en las instrucciones de salto incondicional.

Para estudiar los diversos formatos que admiten las instrucciones del repertorio de los PIC16X8X se clasifican en cinco grandes grupos, atendiendo al tipo de operación que desarrollan.

1.° *Operaciones orientadas a manejar registros de tamaño byte*
2.° *Operaciones orientadas a manejar bits*
3.° *Operaciones que manejan valores inmediatos*
4.° *Operaciones incondicionales de control del flujo del programa*
5.° *Operaciones de salto condicional*

7.2.1. Operaciones orientadas a manejar registros de tamaño byte

El formato de las instrucciones que corresponden a este grupo se divide en tres campos:

1. Campo del Código OP de 6 bits
2. Campo de la dirección del operando fuente (f) de 7 bits
3. Campo que define el operando destino (d) de 1 bit

Las instrucciones de este grupo tienen como sintaxis: *nemónico f,d*. Cuando d = 1 el registro destino coincide con el fuente. Por ejemplo, si se desea incrementar el contenido de un registro, *reg1*, y guardar el nuevo valor en el mismo, se usa la instrucción : *inc reg1,1*

EJEMPLO

¿Qué utilidad puede tener una instrucción que mueve el contenido de un registro a él mismo (coincide el registro fuente con el destino)? movf reg1,1.

SOLUCIÓN

La instrucción movf f,d *pone a 1 el señalizador Z cuando lo transferido es cero. Por eso dicha instrucción puede servir para saber si el contenido es cero o no lo es.*

EJEMPLO

Hallar el formato de la instrucción incf f,d, que utiliza como operando fuente el registro que ocupa la dirección 0x07 de la memoria de datos y como registro destino él mismo (d = 1). Se sabe que los 6 bits del Código OP valen : 001010.

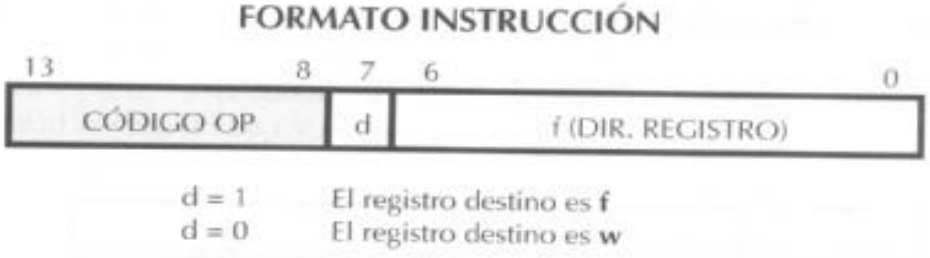


Figura 7.1. *Formato de las instrucciones orientadas a manejar operandos que son registros de un byte. El operando fuente (f) se representa con los 7 bits de la dirección que ocupa el registro en la memoria de datos. El destino puede ser el mismo operando fuente (d = 1) o el registro W (d = 0).*

SOLUCIÓN

Campo del Código OP (6 bits) : 001010
Campo del destino (1 bit) : 1
Campo operando fuente (7 bits) : 0000111

FORMATO DE LA INSTRUCCIÓN *incf 0x07,1* : 001010 1 0000111

7.2.2. Operaciones orientadas a manejar bits

El formato al que responden las instrucciones de este grupo tiene tres campos y se muestra en la Figura 7.2.

1. Campo del Código OP de 4 bits
2. Campo de la dirección del registro fuente (7 bits)
3. Campo de la posición del bit en el registro (3 bits)

EJEMPLO

Al hacer un Reset el procesador coloca el bit 5 (rp0) del registro ESTADO a cero, con lo que comienza referenciando al banco 0 de la memoria de datos de un PIC16X8X. Para referenciar al banco 1 hay que poner a 1 dicho bit:

a) *Indicar la instrucción que permite dicha conmutación de bancos.*
b) *Si el Código OP de la instrucción bsf es 0101 y el registro ESTADO ocupa la dirección 0x03 del banco 0 y 1 de la memoria de datos, hallar el formato de la instrucción del apartado anterior.*

SOLUCIÓN

a) *bsf ESTADO,5*

b) *Campo del Código OP : 0101*
Campo de la posición del bit : 101
Campo dirección registro : 0000011

FORMATO INSTRUCCION *bsf ESTADO,5:* 0101 101 0000011

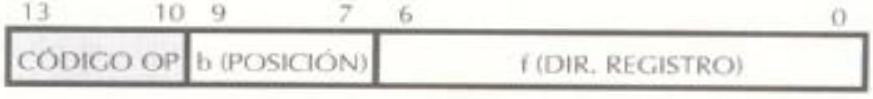


Figura 7.2. *Formato de las instrucciones que realizan operaciones que manipulan un bit. Los 3 bits del campo «b» indican la posición que ocupa el bit afectado dentro del registro fuente cuya dirección la expresa el campo «f».*

7.2.3. Operaciones que manejan un valor inmediato o literal

El formato correspondiente a este grupo de instrucciones sólo tiene dos campos:

1. Campo del Código OP con 6 bits
2. Campo del valor inmediato (k) con 8 bits

EJEMPLO

Se desea cargar en el registro W el valor inmediato 0x33:

a) *¿Qué instrucción se puede emplear?*
b) *Formato de la instrucción, sabiendo que el Código OP de la instrucción movlw es 1100xx (x : indiferente).*

SOLUCIÓN

a) *movlw 0x33*
b) *Campo del Código OP : 1100xx*
Campo del inmediato : 00110011

FORMATO DE LA INSTRUCCIÓN *movlw 0x33* : 1100xx 00110011

7.2.4. Operaciones incondicionales de control del flujo del programa

Este tipo de instrucciones afectan al contenido del Contador de Programa (PC) y sirven para romper la secuencia ordenada de las instrucciones del programa. Las instrucciones de este grupo tienen un formato con sólo dos campos:

1. Campo del Código OP de 3 bits.
2. Campo de la dirección del salto que se carga en el PC de 11 bits.

7.2.5. Operaciones de salto condicional

Los PIC disponen de unas pocas instrucciones «muy efectivas» que cuando se cumple una condición dan un brinco (*skip*). Llamamos brinco a un saltito muy pequeño, concretamente sólo se salta una instrucción, la que hay detrás de la condicional. La condición es el estado de un bit de un registro o la puesta a cero de un registro tras un decremento o un incremento.

La instrucción *decfsz f,d* responde al grupo de instrucciones que manejan registros de un byte. En este caso, si d = 1, el destino es el propio registro fuente y lo que hace la instrucción es decrementar el contenido de *f* y dejar el resultado en *f*. Pero, además, si el nuevo valor de *f*



Figura 7.3. *Formato de las instrucciones de salto incondicional tipo CALL y GOTO. El literal (k) es el valor de la nueva dirección que se carga en el PC.*

es cero se produce un brinco. Si no es cero el flujo de control continúa con la siguiente instrucción.

El formato de las instrucciones de «brinco» es el mismo que el de las operaciones que manipulan registros de un byte, y se muestra en la Figura 7.4.

EJEMPLO

Indicar el formato de la instrucción condicional que decrementa el registro que ocupa la dirección 0x04 de la RAM de datos de un PIC16F84, deposita el resultado en W y brinca en el caso que dicho resultado valga cero. El campo del Código OP correspondiente a la instrucción decfsz es 001011.

SOLUCIÓN

La instrucción que hay que utilizar es decfsz 0x04,0.

FORMATO DE LA INSTRUCCIÓN *decfsz 0x04,0* : 001011 0 0000100

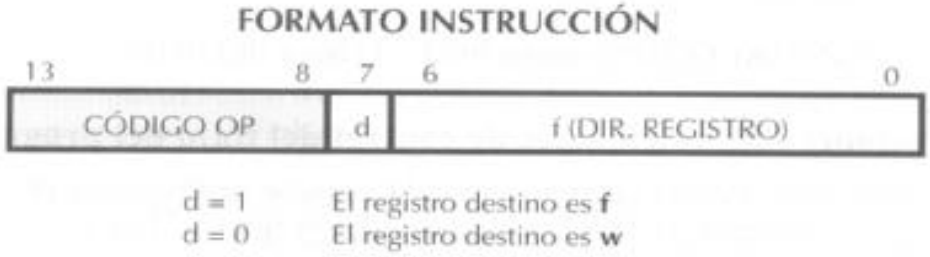


Figura 7.4. *Formato de las instrucciones de brinco condicional, que coincide con el de las instrucciones que manejan registros de un byte.*

7.3. PRECISIONES SOBRE NOMENCLATURA Y SÍMBOLOS

Hemos tomado la decisión de utilizar la nomenclatura y los símbolos que emplea MICROCHIP para su lenguaje Ensamblador MPASM. Recordamos los más específicos.

SÍMBOLO	SIGNIFICADO
0xhh	*Es la forma que se usa en el lenguaje MPASM para referenciar a los números hexadecimales de dos dígitos (hh).*
f	*Representa la dirección en la memoria RAM de datos del registro fuente. Tiene un tamaño de 7 bits, con un direccionamiento de 128 posiciones comprendidas entre la dirección 0x00 y la 0x7f.*
d	*Es un bit que conforma el campo del formato de una instrucción que indica el registro destino. Si d = 0 es W, y si d = 1 es f.*

SÍMBOLO	SIGNIFICADO (*continúación*)
k	*Campo que contiene un valor inmediato, que puede ser un operando (8 bits) o una dirección para el PC (11 bits).*
x	*Valor indeterminado de un bit. Puede ser 1 o 0.*
( )	*Contenido.*
< >	*Campo de un bit de un registro. Por ejemplo: ESTADO <5>.*
∈	*En el conjunto de Ejemplo: d ∈ [0,1].*
[ ]	*Opciones.*

EJEMPLO

Utilizar los símbolos y las abreviaturas usadas en el programa Ensamblador MPASM para expresar las características de la instrucción que realiza la operación lógica AND entre el contenido del registro W y el de otro registro f.

SOLUCIÓN

SINTAXIS	*andwf f,d*	*(Si d = 0 el destino es W)*
OPERANDOS	*0 ≤ f ≤ 127* *d ∈ [0, 1]*	
OPERACIÓN	*(W) and (f) → dest* *Si d = 0, dest = W* *Si d = 1, dest = f*	
FORMATO	*000101 d ffffff*	

PALABRAS QUE OCUPA EN LA MEMORIA DE CÓDIGO : 1

CICLOS DE INSTRUCCIÓN PARA LA EJECUCIÓN : 1

EJEMPLO

Si el contenido del registro W es 0xf0 y el del registro ESTADO es 0x0f, ¿con qué valores quedan cargados ambos registros tras ejecutar la instrucción andwf ESTADO,1?

SOLUCIÓN

Como el operando destino es ESTADO (d = 1), quedan:

W = 0xf0
ESTADO = 0x00

7.4. INSTRUCCIONES QUE MANEJAN REGISTROS

Responden a la sintaxis *nemónico f,d*, siendo *f* y *d* los dos operandos fuente y destino que se hallan implementados por registros de 8 bits de la memoria de datos.

El registro *f* viene referenciado por la dirección de 7 bits que ocupa, mientras que el destino sólo por 1, que si vale 0 es el W y si vale 1 es el fuente. En la Tabla 7.1 se muestran las instrucciones de este grupo con sus características más interesantes.

EJEMPLO

Si se supone que inicialmente valen cero los registros W y los que ocupan las direcciones 0x01 y 0x02 de la memoria de datos, ¿qué valor contendrán después de ejecutar el siguiente programa?

```
incf    0x01,0
incf    0x01,1
comf    0x02,1
iorwf   0x02,0
xorwf   0x01,1
```



INSTRUCCIONES QUE MANEJAN REGISTROS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
ADDWF f,d	Suma W y f	1	00 0111 dfff ffff	C,DC,Z
ANDWF f,d	AND W con f	1	00 0101 dfff ffff	Z
CLRF f	Borra f (Pone todos los bits a 0)	1	00 0001 1fff ffff	Z
CLRW ---	Borra w	1	00 0001 0xxx xxxx	Z
COMF f,d	Complementa f (Invierte)	1	00 1001 dfff ffff	Z
DECF f,d	Decrementa f	1	00 0011 dfff ffff	Z
INCF f,d	Incrementa f	1	00 1010 dfff ffff	Z
IORWF f,d	OR entre W y f	1	00 0100 dfff ffff	Z
MOVF f,d	Mueve f	1	00 1000 dfff ffff	Z
MOVWF f	Mueve W y f	1	00 0000 1fff ffff	---
NOP ---	No opera	1	00 0000 0xx0 0000	---
RLF f,d	Rota f a la izqda. a través del acarreo	1	00 1101 dfff ffff	C
RRF f,d	Rota f a la dcha. a través del acarreo	1	00 1100 dfff ffff	C
SUBWF f,d	Resta W a f	1	00 0010 dfff ffff	C,DC,Z
SWAPF f,d	Intercambia nibbles	1	00 1110 dfff ffff	---
XORWF f,d	XOR de W con f	1	00 0110 dfff ffff	Z

Tabla 7.1. *Principales características de las instrucciones de los PIC16X8X que manejan como operandos registros de 8 bits. Se han excluido las de salto condicional que tienen el mismo formato.*

SOLUCIÓN

(W) = 0xff
(0x01) = 0xfd
(0x02) = 0xff

EJEMPLO

Existe una instrucción con la que se rota a la derecha, a través del señalizador de acarreo C, el registro que ocupa la dirección 0x04 de la memoria de datos y deposita el resultado en W. Se pide:

a) *Nemónico.*
b) *Formato de la instrucción.*
c) *Si inicialmente C = 1 y (0x04) = 0x00, ¿qué valor se cargará en W?*

SOLUCIÓN

a) *rrf 0x04,0*
b) *001100 0 0000100*
c) *W = b'10000000'*

EJEMPLO

Si el registro ESTADO contiene el valor 0xf5 y se ejecuta la instrucción swap ESTADO,0, ¿qué valor contendrán ESTADO y W?

SOLUCIÓN

La instrucción swap *intercambia los «nibbles» entre sí. En un registro de 8 bits un nibble corresponde a los 4 bits de más peso y el otro a los de menos peso.*

$$(f<3:0>) \rightarrow (dest<7:4>)$$
$$(f<7:4>) \rightarrow (dest<3:0>)$$

Como el registro destino es el W, quedarán cargados:

ESTADO : 0xf5
W : 0x5f

7.5. INSTRUCCIONES QUE MANEJAN BITS

Sólo hay dos instrucciones en este grupo, pero son muy flexibles. Una de ellas pone a 1 (*bsf*) cualquier bit de un registro, mientras que la otra lo pone a 0 (*bcf*). (Véase Tabla 7.2.)

En el formato de las instrucciones que manejan bits se representa con 3 bits la posición que ocupa en el registro el bit *b*. El registro se especifica mediante su dirección de 7 bits.

INSTRUCCIONES QUE MANEJAN BITS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
BCF f,b	Borra bit de f	1	01 00bb bfff ffff	---
BSF f,b	Pone a 1 el bit f	1	01 01bb bfff ffff	---

Tabla 7.2. *Características más importantes de las dos instrucciones que manejan un bit determinado de un registro.*

EJEMPLO

Si, inicialmente, el registro W = 0xff y el registro OPTION = 0x00, ¿con qué valores quedarán cargados tras ejecutar el siguiente programa?

```
bsf     OPTION,2
comf    OPTION,0
swapf   OPTION,1
```

SOLUCIÓN

W = b'1111 1011'
OPTION = b'0100 0000'

EJEMPLO

Hallar el formato de la instrucción bsf 0x08,3.

SOLUCIÓN

FORMATO DE LA INSTRUCCIÓN *bsf 0x08,3 :* 0101 011 0001000

7.6. INSTRUCCIONES DE BRINCO *(SKIP)*

Sólo hay cuatro instrucciones de salto condicional en los PIC de la gama media. Dos de ellas testan un bit de un registro y según valga 1 o 0, brincan o no. Recuérdese que un brinco es un «saltito» pequeño, sólo se salta la instrucción siguiente a la condicional. Las otras dos instrucciones incrementan o decrementan un registro y la posibilidad del brinco se efectúa si con esa operación el valor del registro ha llegado a cero. Cuando estas instrucciones no brincan porque no se cumple la condición, tardan 1 ciclo de instrucción en ejecutarse. Caso que brinquen tardan el doble. (Véase Tabla 7.3.)

EJEMPLO

Se trata de averiguar el valor del bit 4 del registro ESTADO. Si vale 1 se borra o se pone a 0 el registro W y si dicho bit vale 0, se pone a 1 ese bit.

INSTRUCCIONES DE "BRINCO"									
SINTAXIS		OPERACIÓN	CICLOS	FORMATO 14 bits					SEÑALIZADORES
BTFSC	f,d	Explora un bit de f y brinca si vale 0	1(2)	01	10bb	bfff	ffff		---
BTFSS	f,d	Explora un bit de f y brinca si vale 1	1(2)	01	11bb	bfff	ffff		---
DECFSZ	f,d	Decrementa f y si es 0, brinca	1(2)	00	1011	dfff	ffff		---
INCFSZ	f,d	Decrementa f y si es 1, brinca	1(2)	00	1111	dfff	ffff		---

Tabla 7.3. *Características más relevantes de las cuatro instrucciones condicionales de brinco.*

SOLUCIÓN

```
btfss    ESTADO,4  ; Si ESTADO<4> = 1, hay brinco
bsf      ESTADO,4  ; Como ESTADO<4> = 0, se pone a 1
clrw               ; Como ESTADO<4> = 1, se borra W
```

EJEMPLO

Decrementar el contenido del registro FSR hasta que valga cero y entonces borrar W.

SOLUCIÓN

```
bucle    decfsz FSR,1
         goto   bucle
         clrw
```

7.7. INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS

Se trata de media docena de instrucciones que realizan una operación con un valor inmediato de 8 bits que se proporciona dentro del formato de la instrucción, el cual sólo tiene dos campos: el del Código OP (6 bits) y el del operando inmediato (8 bits). (Tabla 7.4.)

INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS									
SINTAXIS		OPERACIÓN	CICLOS	FORMATO 14 bits					SEÑALIZADORES
ADDLW	k	Suma inmediata con W	1	11	111x	kkkk	kkkk		C,DC,Z
ANDLW	k	AND inmediato con W	1	11	1001	kkkk	kkkk		Z
IORLW	k	OR inmediato con W	1	11	1000	kkkk	kkkk		Z
MOVLW	k	Mueve a W un valor inmediato	1	11	00xx	kkkk	kkkk		---
SUBLW	k	Resta W de un inmediato	1	11	110x	kkkk	kkkk		C,DC,Z
XORLW	K	OR exclusiva con W	1	11	1010	kkkk	kkkk		Z

Tabla 7.4. *Características más importantes de las instrucciones que manejan operandos inmediatos (k).*

EJEMPLO

Indicar el valor del registro W después de ejecutar el siguiente programa:

```
clrw
movwf   OPTION
bsf     OPTION,5
comf    OPTION,1
btfsc   OPTION,1
swap    OPTION,0
```

SOLUCIÓN

(W) = b'1111 1101'

EJEMPLO

Averiguar el contenido de W después de ejecutar el siguiente programa:

```
clrw
addlw   0x55
andlw   0xff
iorlw   0x0f
xorlw   0xaf
```

SOLUCIÓN

(W) = 0xf0

7.8. INSTRUCCIONES DE CONTROL Y ESPECIALES

En este grupo se incluyen las instrucciones que rompen la secuencia normal del programa porque alteran el contenido del PC y también las instrucciones especiales.

La instrucción de salto incondicional *goto* carga en el PC la dirección de la nueva instrucción. La instrucción *call* de Llamada a Subrutina, antes de cargar el PC con la dirección de la instrucción a saltar, salva la dirección de partida guardando en la cima de la Pila el valor actual del PC. De esta manera, al retornar de la subrutina se saca de la Pila la dirección de regreso del programa principal.

Para realizar un retorno de una subrutina se pueden emplear dos instrucciones. La más habitual es *return*, que se limita a extraer de la cima de la Pila el valor que carga en el PC. Otra más compleja es *retlw k*, que, además de hacer lo mismo que *return*, carga en W el valor inmediato *k* que contiene. Es decir, devuelve un parámetro desde la subrutina.

Para el final de las interrupciones hay otra instrucción cuyo nemónico es *retfie*. La operatividad de esta instrucción consiste en cargar en el PC el contenido de la cima de la Pila y poner el bit GIE = 1, pues al comenzar la interrupción este bit se pone automáticamente a 0 para evitar que cuando se atiende una interrupción se produzca otra. GIE es el bit de permiso de todas las interrupciones.

En cuanto a las instrucciones especiales, se han incluido dos en este grupo: *clrwdt* y *sleep*. La primera pone a 0 el contenido del Perro Guardián, es decir, lo refresca o lo reinicializa. El Perro Guardián si se desborda (pasa de 0xff a 0x00) provoca un Reset. La instrucción *clrwdt* hay que colocarla estratégicamente en ciertos puntos del programa para evitar la reinicialización.

La instrucción *sleep* introduce al procesador en un modo de funcionamiento que se llama de Reposo o de Bajo Consumo. Detiene al oscilador y el procesador queda congelado, no ejecutando instrucciones y manteniendo el mismo valor las Puertas de E/S. También pone los bits PD# = 0 y TO# = 1 y borra al Perro Guardián y al Divisor de frecuencia. (Tabla 7.5.)

EJEMPLO

Si un PIC16F84 funciona con un cristal de 4 MHz y el Perro Guardián está programado para que se desborde cada 18 ms, calcular cada cuántas instrucciones como máximo deberá refrescarse al Perro con la instrucción clrwdt.

SOLUCIÓN

Fosc = 1/4 · 10^6 = 250 ns
Ciclo Instrucción = 4 · Fosc = 1.000 ns = 1 µs
18 ms = 18.000 µs

Luego habrá que incluir una instrucción clrwdt cada 18.000 normales como máximo. Las instrucciones de salto cuentan el doble.

EJEMPLO

Explicar las diferencias entre las tres instrucciones de «retorno».

SOLUCIÓN

return: *Retorna de una subrutina al programa principal. Carga al PC con el contenido de la cima de la Pila.*

retlw k: *Hace lo mismo que «return» y, además, carga en W el literal k.*

retfie: *Carga al PC* con el contenido de la cima de la Pila y pone GIE = 1.

En la Tabla 7.6 se presentan las principales características de las 35 instrucciones de los PIC de la gama media, agrupadas por funciones, tal como se han descrito.

INSTRUCCIONES DE CONTROL Y ESPECIALES				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
CALL k	Llamada a subrutina	2	10 0kkk kkkk kkkk	TO#,PD#
CLRWDT	Borra o refresca el Perro Guardian	1	00 0000 0110 0100	—
GOTO k	Salto incondicional	2	10 1kkk kkkk kkkk	—
RETFIE	Retorno de Interrupción (GIE=1)	2	00 0000 0000 (00)	—
RETLW k	Retorno subrutina y carga W=k	2	11 01xx kkkk kkkk	—
RETURN	Retorno de subrutina	2	00 0000 0000 1000	—
SLEEP	Pasa al modo de reposo	1	00 0000 0110 0011	TO#,PD#

Tabla 7.5. *Principales características de las instrucciones de control del flujo del programa y de las especiales.*

INSTRUCCIONES QUE MANEJAN REGISTROS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
ADDWF f,d	Suma W y f	1	00 0111 dfff ffff	C,DC,Z
ANDWF f,d	AND W con f	1	00 0101 dfff ffff	Z
CLRF f	Borra f (Pone todos los bits a 0)	1	00 0001 1fff ffff	Z
CLRW —	Borra w	1	00 0001 0xxx xxxx	Z
COMF f,d	Complementa f (Invierte)	1	00 1001 dfff ffff	Z
DECF f,d	Decrementa f	1	00 0011 dfff ffff	Z
INCF f,d	Incrementa f	1	00 1010 dfff ffff	Z
IORWF f,d	OR entre W y f	1	00 0100 dfff ffff	Z
MOVF f,d	Mueve f	1	00 1000 dfff ffff	Z
MOVWF f	Mueve W y f	1	00 0000 1fff ffff	—
NOP —	No opera	1	00 0000 0xx0 0000	—
RLF f,d	Rota f a la izqda. a través del acarreo	1	00 1101 dfff ffff	C
RRF f,d	Rota f a la dcha. a través del acarreo	1	00 1100 dfff ffff	C
SUBWF f,d	Resta W a f	1	00 0010 dfff ffff	C,DC,Z
SWAPF f,d	Intercambia nibbles	1	00 1110 dfff ffff	—
XORWF f,d	XOR de W con f	1	00 0110 dfff ffff	Z
INSTRUCCIONES QUE MANEJAN BITS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
BCF f,b	Borra bit de f	1	01 00bb bfff ffff	—
BSF f,b	Pone a 1 el bit f	1	01 01bb bfff ffff	—
INSTRUCCIONES DE "BRINCO"				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
BTFSC f,d	Explora un bit de f y brinca si vale 0	1(2)	01 10bb bfff ffff	—
BTFSS f,d	Explora un bit de f y brinca si vale 1	1(2)	01 11bb bfff ffff	—
DECFSZ f,d	Decrementa f y si es 0, brinca	1(2)	00 1011 dfff ffff	—
INCFSZ f,d	Decrementa f y si es 1, brinca	1(2)	00 1111 dfff ffff	—
INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
ADDLW k	Suma inmediata con W	1	11 111x kkkk kkkk	C,DC,Z
ANDLW k	AND inmediato con W	1	11 1001 kkkk kkkk	Z
IORLW k	OR inmediato con W	1	11 1000 kkkk kkkk	Z
MOVLW k	Mueve a W un valor inmediato	1	11 00xx kkkk kkkk	—
SUBLW k	Resta W de un inmediato	1	11 110x kkkk kkkk	C,DC,Z
XORLW k	OR exclusiva con W	1	11 1010 kkkk kkkk	—
INSTRUCCIONES DE CONTROL Y ESPECIALES				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
CALL k	Llamada a subrutina	2	10 0kkk kkkk kkkk	TO#,PD#
CLRWDT	Borra o refresca el Perro Guardian	1	00 0000 0110 0100	—
GOTO k	Salto incondicional	2	10 1kkk kkkk kkkk	—
RETFIE	Retorno de interrupción (GIE=1)	2	00 0000 0000 1001	—
RETLW k	Retorno subrutina y carga W=k	2	11 01xx kkkk kkkk	—
RETURN	Retorno de subrutina	2	00 0000 0000 1000	—
SLEEP	Pasa al modo de reposo	1	00 0000 0110 0011	TO#,PD#

Tabla 7.6. *Principales características de las 35 instrucciones de los PIC de la gama media, agrupadas según su función y en orden alfabético.*

PROGRAMAR PIC ES FÁCIL

SÉPTIMO EJERCICIO: PROGRAMAS CON UN POCO DE TODO MANEJANDO EL REGISTRO DE TRABAJO W

Siendo W el registro de trabajo por excelencia y el más usado en todos los programas, resulta curioso que no disponga de algunas instrucciones que existen para realizar algunas operaciones con los otros registros. Se presentan algunas soluciones a estas carencias.

Incrementar W

Como W no tiene una instrucción específica para realizar esta operación, se puede conseguir el mismo resultado utilizando *addlw 0x01*.

Decrementar W

Se recomienda emplear la instrucción *addlw 0xff*.

Complementar o invertir W

Para este fin puede usarse la instrucción *xorlw 0xff*.

Hallar el complemento a 2 de W

Primero se realiza el complemento a 1 y luego se suma 1:

```
xorlw 0xff
addlw 0x01
```

Poner a 0 varios bits de W sin alterar el resto

Si, por ejemplo, se desea poner a 0 los 4 bits de menos peso de W sin modificar los restantes, puede usarse la instrucción *andlw 0xf0*.

Poner a 1 varios bits de W sin alterar el resto

Se usa la operación lógica OR poniendo en el operando inmediato un 1 en aquellos bits que se desea poner a 1, mientras que los restantes se ponen a 0. Por ejemplo, si se desea poner a 1 los 6 bits de más peso de W sin modificar los otros dos, puede usarse la instrucción *iorlw 0xfc*.

MANEJANDO IGUALDADES Y DESIGUALDADES

Comprobar la igualdad del contenido de dos registros

Para llamar a la subrutina IGUAL en un programa cuando se cumple que el contenido del REG1 es igual al del REG2 se puede usar las siguientes instrucciones:

```
movf    REG1,0      ; REG1 se carga en W
xorwf   REG2,0      ; REG1 xor REG2 se carga en W
btfsc   ESTADO,2    ; Si ESTADO<2> = Z = 0, brinca
call    IGUAL       ; Z = 1 y se llama a IGUAL
```

Comprobar que un registro tiene un contenido mayor que otro

Se proponen las instrucciones que proporcionan un salto a la subrutina MAYOR cuando el contenido del REG1 es mayor que el del REG2.

```
movf    REG1,0      ; REG1 a W
subwf   REG2,0      ; REG1 - REG2 a W
btfss   ESTADO,0    ; Si ESTADO<0> = C = 0, brinca
call    MAYOR       ; Como C = 1, REG1 > REG2
```

Comprobar que un registro tiene un contenido menor que otro

Con las siguientes instrucciones se salta a la subrutina MENOR cuando el contenido del REG1 es menor que el del REG2.

```
movf    REG2,0      ; REG2 a W
subwf   REG1,0      ; REG2 - REG1 a W
btfss   ESTADO,0    ; Si ESTADO<0> = C = 1, brinca
call    MENOR       ; C = 0, luego REG1 < REG2
```

EXPLORANDO LA CAUSA DE LA INTERRUPCIÓN

Los PIC16X8X disponen de 4 fuentes o causas que provocan una interrupción. Las cuatro mandan a la dirección 0004 H de la memoria de código el flujo de control (Vector de Interrupción), por eso hay que comenzar averiguando cuál de las cuatro ha sido la causante de la interrupción para atenderla con su correspondiente rutina. Las mencionadas causas son:

1.ª Interrupción Externa por activación de la patita RB0/INT (Señalizador INTF).
2.ª Desbordamiento del TMR0 (Señalizador T0IF).
3.ª Cambio de estado en RB 7:4 (Señalizador RBIF).
4.ª Fin de escritura en EEPROM (Señalizador EEIF).

Los señalizadores son bits que se ponen a 1 automáticamente en cuanto se produce alguna de las causas de interrupción. Como se aprecia en la Figura 7.5, tres de estos señalizadores se ubican en el registro INTCON y el cuarto en el EECON1.

Al iniciarse en el Vector de Interrupción la Rutina de Servicio de la Interrupción hay que averiguar la causa explorando los cuatro señalizadores, para saltar a la rutina especifica, que llamaremos EXTERNA, TIMER, PUERTAB y EEPROM, según que se hallen a 1 los señalizadores INTF, T0IF, RBIF y EEIF, respectivamente. Se ofrece un programa que explora a los señalizadores en el orden comentado y al encontrar al primero que valga 1 envía el flujo de control a su rutina de tratamiento.

```
btfsc   INTCON,1   ; Si INTF = 0, brinca
call    INTE       ; Como INTF = 1, se llama a INTE
btfsc   INTCON,2   ; Si T0IF = 0, brinca
call    TIMER      ; Como T0IF = 1, se llama a TIMER
btfsc   INTCON,0   ; Si RBIF = 0, brinca
call    PUERTAB    ; Como RBIF = 1, se llama a PUERTAB
bsf     ESTADO,5   ; Selección del banco 1 donde está EECON1
btfsc   EECON1,4   ; Si EEIF = 0, brinca
call    EEPROM     ; Como EEIF = 1, se llama a EEPROM
```

PROYECTO PARA MANEJAR UN DISPLAY DE 7 SEGMENTOS

Con este ejercicio se pretende manejar todo el conjunto de instrucciones del repertorio, diseñar el hardware y confeccionar el software para controlar un visualizador de 7 segmentos desde varios interruptores.

A las 3 líneas de menos peso de la Puerta A de un PIC16X84 se conectan 3 interruptores por los que se introduce un número binario comprendido del 000 al 111.

A las líneas de la Puerta B se conectan los segmentos de un display, que como se aprecia en el esquema de la Figura 7.6, son 7 más el del punto decimal, dp.

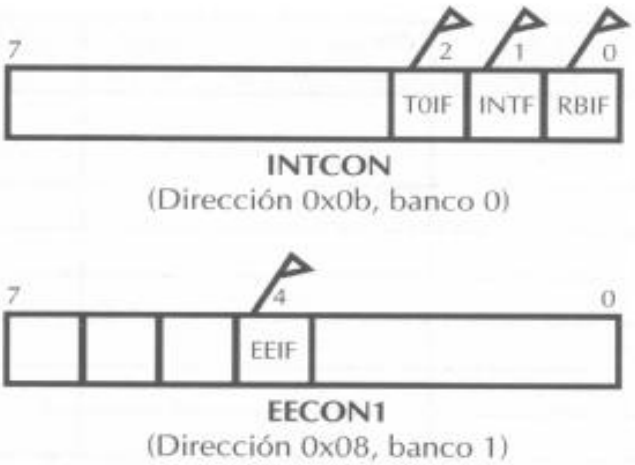


Figura 7.5. *Los cuatro señalizadores de las causas de interrupción se ubican en los registros INTCON y EECON1.*

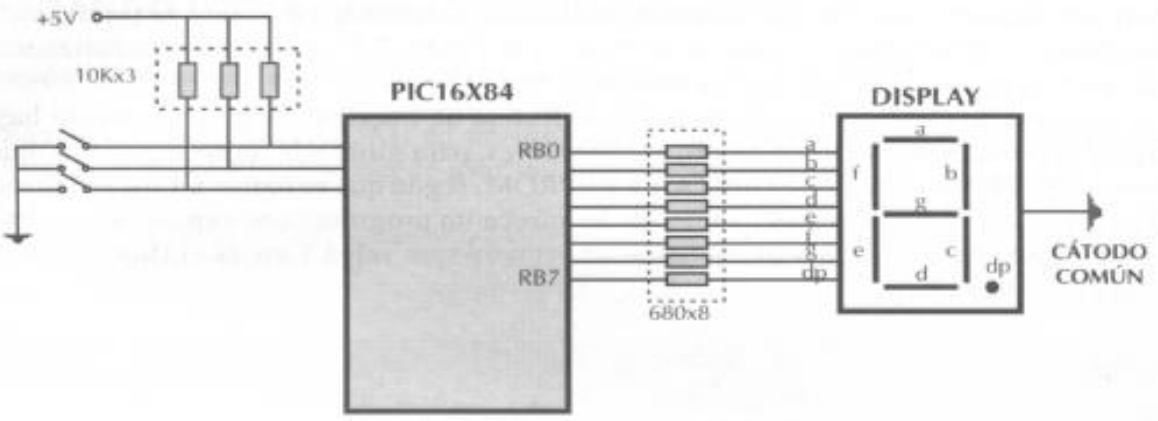


Figura 7.6. *Esquema del conexionado de los interruptores que actúan como periféricos de entrada y el display de 7 segmentos como periférico de salida. El display es de cátodo común, lo que supone que los segmentos se iluminan cuando la línea de la Puerta B les introduce un nivel alto.*

El ejercicio consiste en visualizar sobre el display el número decimal correspondiente al binario que se introduce por los tres interruptores.

SOLUCIÓN

Vamos a dividir el programa en dos partes. En la primera realizaremos el tratamiento del display de 7 segmentos y en la segunda el de los interruptores.

Para visualizar en el display los 8 números decimales del 0 al 7, correspondientes a las 8 combinaciones binarias de los interruptores, se utiliza una tabla de conversión de códigos. El código de 7 segmentos es el que tienen que sacar las 8 líneas de la Puerta B para que se ilumine el número decimal. (Tabla 7.7.)

BINARIO (INTERRUPCIONES)	DECIMAL (A VISUALIZAR)	CÓDIGO DE 7 SEGMENTOS BINARIO (RB7-RB0)	CÓDIGO DE 7 SEGMENTOS HEXADECIMAL
000	0	0011 1111	3F
001	1	0000 0110	06
010	2	0101 1011	5B
011	3	0100 1111	4F
100	4	0110 0110	66
101	5	0100 1101	6D
110	6	0111 1101	7D
111	7	0000 0111	07

Tabla 7.7. *Tabla de conversión del código binario de los tres interruptores al código de 7 segmentos que se aplica al display a través de las líneas de la Puerta B.*

Si se conoce el número decimal que hay que visualizar y dicho valor se carga en W, se llama a la subrutina DISPLAY que contiene los 8 valores ordenados de la Tabla 7.7. Dichos valores están implementados mediante instrucciones del tipo *retlw k*, siendo *k* el valor correspondiente al código de 7 segmentos. *retlw k* retorna al programa principal y carga en W el valor *k*. Una vez realizada esta operación bastará sacar W por la Puerta B para iluminar el número decimal.

El programa que se propone parte del supuesto de que, inicialmente, en W se encuentra el número decimal a visualizar.

```
        call   display  ; Llama a la tabla "display"
        movwf  PUERTAB  ; Al retornar de la subrutina el valor
                        ; de W se saca por la Puerta B
bucle   goto   bucle    ; bucle infinito
```

```
; TABLA DE CONVERSIÓN-----------------------------------------

display addwf  PCL,f    ; pcl + W → W
        .               ; El pcl se incrementa con el valor
                        ; de W proporcionando un salto
        retlw  0x3F     ; Retorna con el valor del código del 0

        retlw  0x06     ; Retorna con el código del 1
        retlw  0x5b     ; Retorna con el código del 2
        retlw  0x4f     ; Retorna con el código del 3
        retlw  0x66     ; Retorna con el código del 4
        retlw  0x6d     ; Retorna con el código del 5
        retlw  0x7d     ; Retorna con el código del 6
        retlw  0x07     ; Retorna con el código del 7

        END
```

El módulo anterior realizaba el tratamiento del display; falta explorar los 3 interruptores para averiguar el valor decimal a representar. Se supone que un interruptor cerrado *equivale* a 1 y abierto a 0. Por esta razón habrá que invertir los niveles introducidos por RA2:0 y después convertir a decimal el código binario. Para efectuar esta conversión se supone que el bit de menos peso corresponde con el valor invertido que se introduce por RA0. Si dicho valor vale 1 el peso de este dígito es 1. Si RA1 = 1, el peso de este digito corresponde a 2_{10}, y si RA2 = 1, el peso de este bit corresponde a 4_{10}.

Se propone a continuación la cabecera del programa del proyecto y la parte correspondiente a la exploración de los interruptores y el cálculo del número decimal correspondiente que se carga en W, desde donde ya puede enlazarse con el módulo del DISPLAY expuesto anteriormente.

```
        LIST    P=16C84
        RADIX   HEX

        w       EQU     0
        f       EQU     1
        PCL     EQU     02
        PUERTAA EQU     05
        PUERTAB EQU     06
```

```
          ESTADO  EQU     03
          VALOR   EQU     12      ; Registro auxiliar
          ORG     0
          goto    inicio
          ORG     5               ; Para saltar el Vector de
                                  ; Interrupción
inicio    bsf     ESTADO,5        ; Selección del banco 1
          clrw
          movwf   PUERTAB         ; Puerta B salida
          movlw   0xff
          movwf   PUERTAA         ; Puerta A entrada
          bcf     ESTADO,5        ; Selección banco 0
explora   movf    PUERTAA,w       ; Se cargan los
                                  ; interruptores en W
          movwf   VALOR           ; Se usa un registro
                                  ; auxiliar
          comf    VALOR,f         ; Invertir los niveles de
                                  ; interuptores
          clrw
          btfss   VALOR,0         ; Si RA0 = 1, brinca
          goto    bit_1           ; Salta a explorar el
                                  ; bit_1
          addlw   0x01            ; Si RA0 = 1, se suma 1
bit_1     btfss   VALOR,1         ; Si RA1 = 1, brinca
          goto    bit_2           ; Salta a explorar el
                                  ; bit_2
          addlw   0x02            ; Si RA1 = 1, se suman 2
bit_2     btfss   VALOR,2         ; Si RA2 = 1, brinca
          goto    visual          ; Salta al módulo del
                                  ; display
          addlw   0x04            ; Si RA2 0 1, se suman 4
visual    call    display         ; Salta al módulo del
                                  ; display
```

Al ejecutarse este módulo se consigue que el registro de trabajo W quede cargado con el valor que debe visualizar el display de 7 segmentos, por cuyo motivo la última instrucción llama a la subrutina *display*. En realidad, como el PIC opera en binario, el valor que contiene W es el mismo que el que se obtendría explorando los 3 interruptores e invirtiendo su valor. Sin embargo se ha realizado este inútil proceso (convertir el valor binario en el mismo valor binario) para manejar instrucciones muy usuales.

Lo que perseguimos es que intente probar el programa, y de lo que nos alegraríamos es que no le funcionase, porque trampas, si no las hay, aparecen solas. Así que a trabajar, porque si logra implementar el hardware y el software de este ejercicio y le funcionan correctamente, ¡ES UN ARTISTA!

SIMULANDO CON EL SIMUPIC'84

Para simular este programa haremos uso de la ventana de «visión externa» del SIMUPIC, de la introducción de señales mediante las teclas de función, de la Tabla 7.7 de este libro y de una gran dosis de paciencia.

El simulador no dispone de un display de siete segmentos, de modo que tendremos que comprobar que la salida en binario es correcta (por eso necesitamos la Tabla 7.7). Se carga el programa SEPTIMO.ASM y se modifica el contenido de la Puerta A. (Figura 7.7.)

Se introduce el valor binario que se desee en los tres bits de menos peso de la Puerta A y se ejecuta el programa

PROGRAMANDO EN C

Este programa puede confeccionarse en C de la siguiente manera:

```
/* SÉPTIMO.C: Programa que explora el contenido de tres    */
/* interruptores en RA0, RA1 y RA2 y muestra el resultado */
/* en un display de siete segmentos en RB                 */
```

```
#include <16C84.H>
#byte puertaa = 05  /* Posición de la puerta A */
#byte puertab = 06  /* Posición de la puerta B */

void main( void )

{
int temp = 0, tabla[8]={ 0x3F, 0x06, 0x5B, 0x4F,
                         0x66, 0x6D, 0x7D, 0x07 };
int índice = 0;
set_tris_a( 0xFF );  /* Puerta A configurada para entrada */
set_tris_b( 0x00 );  /* Puerta B configurada para salida */
temp = ~puertaa;     /* Se complementa la entrada de PA */
if ( bit_test(temp, 0 ) ) índice += 1;
if ( bit_test(temp, 1 ) ) índice += 2;
if ( bit_test(temp, 2 ) ) índice += 4;
puertab = tabla[índice];
}
```

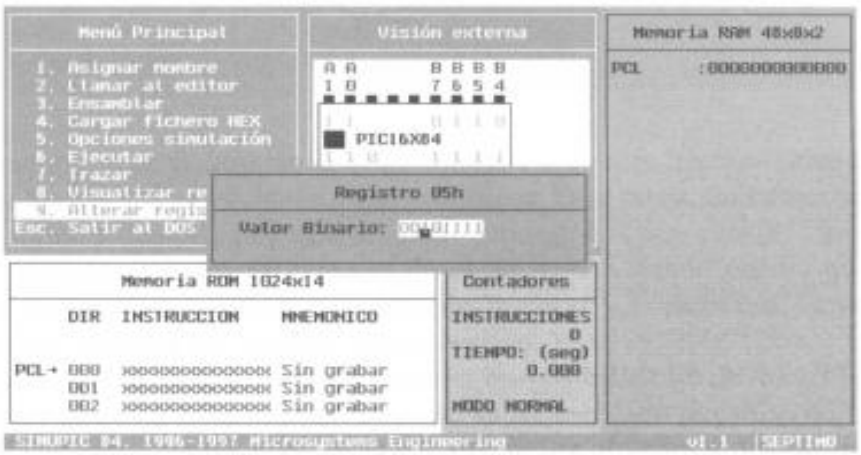


Figura 7.7. *Modificación de los valores de los registros con el SIMUPIC.*

Hemos utilizado un *array* tradicional en lugar de una tabla que haya que recorrer mediante saltos, método que resulta bastante más sencillo pero que consume memoria RAM. Por otra parte, el cálculo del valor decimal introducido por la puerta A se sigue haciendo mediante una serie de condiciones y sumas similar a la programada anteriormente en ensamblador.

PROBANDO EN EL MICRO'PIC TRAINER

Para probar nuestro programa con el Micro'PIC Trainer se procede de forma parecida al SIMUPIC, pero usando el display de la propia placa. Cargar el programa en el PIC de la placa y conectar el display de siete segmentos mediante el *jumper* correspondiente. El display deberá visualizar el número que se esté introduciendo por los interruptores RA0, RA1 y RA2.

¿Se atreve con esto?

Ahora que sabemos manejar un display de siete segmentos, sabemos hacer temporizaciones y no tenemos ningún problema a la hora de leer interruptores, le proponemos un ejercicio que combina todo.

Se trata de confeccionar un programa que lea tres interruptores en la Puerta A, situados en RA0, RA1 y RA2, y que introducen un 0 al estar cerrados y un 1 al estar abiertos. Contar en un display conectado a la puerta B (tal y como se muestra en la Figura 7.6) los valores desde el 0 al introducido por los interruptores de la Puerta A a intervalos de 500 ms. Ejemplo: si se introduce el valor «011», en el display deberá aparecer «0-1-2-3-0-1-2-3...», con una pausa de 500 ms entre números. Suponer que el PIC va a funcionar a una frecuencia de 4 MHz y realizar la temporización mediante interrupciones. Tampoco estaría de más que utilizase el «Perro Guardián». Existe una solución a este ejercicio en el disquete bajo el nombre SEPTIMO2.ASM.

HERRAMIENTAS Y DISEÑO DE PROYECTOS



8.1. FASES DE DISEÑO

En el despertar del siglo XXI asistimos a una impresionante carrera mundial por incorporar microcontroladores a los productos de mayor consumo para mejorar la imagen, las prestaciones, el tamaño, el consumo y el precio.

Cada nuevo diseño nace con una idea y termina con el prototipo que la implementa. Durante ese proceso suceden ordenadamente una serie de etapas que hay que cubrir para llegar a un final feliz. En el desarrollo de cada fase se utilizan un conjunto de herramientas hardware y software, que según su potencia reducen más o menos el tiempo que se consume en completarla. Pero, aunque sean muy valiosas dichas herramientas, no hay que olvidar que, al igual que pasa con la mayoría de las cosas en nuestro mundo, hay dos pilares fundamentales:

1.º La IDEA

Sólo la imaginación del diseñador pone límites a un proyecto.

2.º El TIEMPO

La voracidad del mercado es tan agobiante que cuando nos piden hoy un proyecto, nos exigen que lo hayamos entregado ayer.

Con este panorama de trabajo se deben manejar excelentes herramientas que sean capaces de *mejorar la idea inicial en el mínimo tiempo*. Hasta hace poco, únicamente las grandes empresas podían soportar los grandes desembolsos económicos que exigía la adquisición de esas herramientas. Actualmente, los fabricantes de *chips* se van dando cuenta que para venderlos por cientos de millones de unidades tienen que facilitar y popularizar su empleo y poner a disposición de los usuarios los medios que necesitan.

Microchip se ha esforzado siempre en atender las posibilidades de los profesionales particulares, y puede ser que ésta haya sido la razón de haber escalado puestos en el ranking mundial de venta de microcontroladores durante la década de los noventa y ocupar uno de los puestos de cabeza. La disponibilidad de herramientas eficaces y económicas justifica la espectacular

aceptación de los PIC. Cualquier aficionado, estudiante, diseñador particular o pequeña empresa puede adquirir el instrumental requerido para el desarrollo de proyectos con PIC y cientos de empresas de todo el mundo compiten por mejorar, ampliar y abaratar todo tipo de herramientas.

Para analizar las prestaciones y funciones que deben soportar las herramientas de desarrollo, se presentan las fases típicas de un proyecto basado en un microcontrolador y que se muestran gráficamente en el organigrama de la Figura 8.1. Se ha supuesto que no se dispone de una herramienta muy potente y cara llamada «emulador en circuito» y que se comenta más adelante.



Figura 8.1. *Organigrama que representa las sucesivas fases del diseño de un proyecto basado en microcontrolador, en el caso de no disponer de un «emulador en circuito».*

Tanto el organigrama de la Figura 8.1 como la tabla de la Figura 8.2 se han hecho pensando en personas particulares que no dispongan de la herramienta de trabajo por excelencia, el «emulador en circuito», cuyo coste suele ser superior a los 1.000 dólares. No obstante, con las herramientas que se proponen se pueden alcanzar las mismas metas, de manera más artesanal y con igual precisión y eficacia.

La Figura 8.2 contiene una tabla que describe las características fundamentales de cada fase de un diseño con microcontrolador, cuando no se dispone de un «emulador en circuito».

8.2. HERRAMIENTAS ACCESIBLES

Se describen modelos comerciales de las herramientas hardware y software empleadas en los diseños «humildes», que pueden adquirir muchas personas a nivel particular.

	FUNCIÓN	TRABAJO A REALIZAR	HERRAMIENTA NECESARIA
1ª	LA IDEA	PENSAR	LA CABEZA
2ª	EDICIÓN PROGRAMA FUENTE	EDITAR EL PROGRAMA	EDITOR
3ª	ENSAMBLAR O COMPILAR	Traducir el programa en lenguaje fuente a código binario ejecutable	ENSAMBLADOR O COMPILADOR
4ª	SIMULACIÓN DEL SOFTWARE	Simular el comportamiento del programa	SIMULADOR SOFWARE
5ª	DEPURACIÓN	Corregir los defectos del programa	TODAS LAS ANTERIORES
6ª	GRABACIÓN DEL µC	Grabar el programa depurado en la memoria de instrucciones del microcontrolador	GRABADOR
7ª	SIMULAR EL HARDWARE Y EL SOFTWARE EN TIEMPO REAL	Conectar al µC grabado los periféricos fundamentales y analizar su comportamiento	SISTEMA DE DESARROLLO
8ª	DEPURACIÓN	Corregir errores en el programa y el hardware	LAS ANTERIORES
9ª	MONTAJE DEL PROTOTIPO	Construir un prototipo completo con el µC grabado y todos sus periféricos	PLACA DE PROTOTIPOS
10ª	DEPURACIÓN	Corregir errores hardware y software en el prototipo	TODAS LAS ANTERIORES
11ª	PRODUCTO FINAL	MONTAJE DEFINITIVO	EL SOLDADOR

Figura 8.2. *Tabla que muestra algunas características de cada fase de un proyecto y las herramientas que se utilizan.*

8.2.1. La cabeza

Es una herramienta que tenemos todos, pero es la más importante. Con ella se puede obtener la clave del éxito, que es la idea, y también puede suplir y resolver carencias y dificultades que se presentan en todos los diseños. Se usa en todas las fases y de su rendimiento depende todo el trabajo.

Cuide bien su cabeza, porque con sólo una buena idea que le proporcione puede ser suficiente.

8.2.2. Editor de textos

Concebida la estructura básica de la aplicación, se puede atacar el desarrollo del software y del hardware. Con un equipo de especialistas estas dos partes se atacarían en paralelo, pero si el equipo es Vd., como pasa a menudo, no se preocupe. No tendrá que compartir éxitos y fracasos.

Le recomendamos que si está solo comience confeccionando el software. Tendrá que escribir el programa de control de la tarea, para lo cual precisará de un editor de textos que trabaje con caracteres ASCII. El más a mano es el propio editor del sistema operativo MS-DOS de su computador. Teclee EDIT, ejecute y estará dentro de dicho editor.

8.2.3. Ensamblador o Compilador

En el editor tiene que ir tecleando el programa usando las instrucciones del lenguaje que elija. Así creará el programa fuente. Dicho programa hay que traducirlo a código binario o código máquina para cargarlo en la memoria de instrucciones del microcontrolador y así poder ejecutarlo el procesador. A este programa ejecutable por el microcontrolador se llama programa objeto.

En los diversos ejercicios que vamos presentando en todos los capítulos del libro hemos tratado de resolverlos con lenguaje Ensamblador y con C, para comparar los resultados. También hay lenguajes muy fáciles, como el BASIC, que podría utilizar si no es un experto en Informática. El lenguaje que le recomendamos es el Ensamblador; con él obtendrá el mínimo código y así necesitará menos capacidad de memoria y menos tiempo para su ejecución. Pero Vd. es libre.

Microchip ha creado un lenguaje y el correspondiente programa traductor MPASM, que es el que hemos utilizado en este libro.

También para los PIC la empresa **Parallax** dispone del programa Ensamblador **MPASM**, cuyos nemónicos son similares a los utilizados por **Intel**.

Si trabaja con un lenguaje de alto nivel, como el C, tiene varias opciones de adquirir un compilador que lo traduzca a código máquina. Destaca el compilador de C creado por **Byte Craft**, cuyo coste ronda los 1.000 dólares. Otro más barato y muy eficaz es el de la empresa **CCS**, que distribuye en España **Microsystems Engineering**.

En EE.UU. hay bastantes empresas que ofrecen compiladores e intérpretes del lenguaje BASIC, entre las que se distinguen **Parallax**, **microEngineering Labs** y **VersaTech**. En Europa este lenguaje no tiene tanta aceptación.

8.2.4. Simulador software

Es un programa que reproduce por software el comportamiento del microcontrolador en la ejecución de un programa, presentando en la pantalla del PC el estado de todos los registros y recursos. No funciona en tiempo real al estar implementada la simulación con software. Tampoco puede adaptarse con periféricos externos.

Su empleo no exige ningún hardware y es muy práctico en la primera fase de depuración del programa. Elimina muchos errores que agilizan el desarrollo posterior.

Los simuladores software no están recomendados en aplicaciones en las que el tiempo sea un parámetro estricto y determinante, así como en aquellas que tengan mucha dependencia con el mundo exterior.

Microchip tiene el simulador **MPSIM** para los PIC y **Microsystems Engineering** ha creado el simulador **SIMUPIC'84**, sólo aplicable a los PIC16X84, pero de un valor didáctico y profesional extraordinarios. Su bajo coste lo hace muy recomendable en formación y diseño con esos modelos concretos de PIC. En la Figura 8.3 se muestra una pantalla del **SIMUPIC'84**, en la que puede apreciarse que el Menú Principal dispone de todas las fases para la confección del software. Tiene ventanas en las que se visualizan los estados de registros, posiciones de memoria, el programa ensamblado y desensamblado, así como la presentación del estado lógico que tiene cada una de las patitas del PIC16X84.

8.2.5. El grabador

El programa objeto, o sea, en código máquina ejecutable, hay que escribirlo en la memoria de instrucciones del microcontrolador. Para realizar esta operación se necesita un «grabador».

Un grabador consiste en una tarjeta electrónica que soporta varios zócalos con diferente número de patitas, en los que se introducen diversos tipos de microcontroladores. Dispone de circuitos auxiliares y de estabilización de la alimentación. La grabación se controla mediante un programa de comunicación desde un PC que se adapta al grabador por el puerto paralelo.

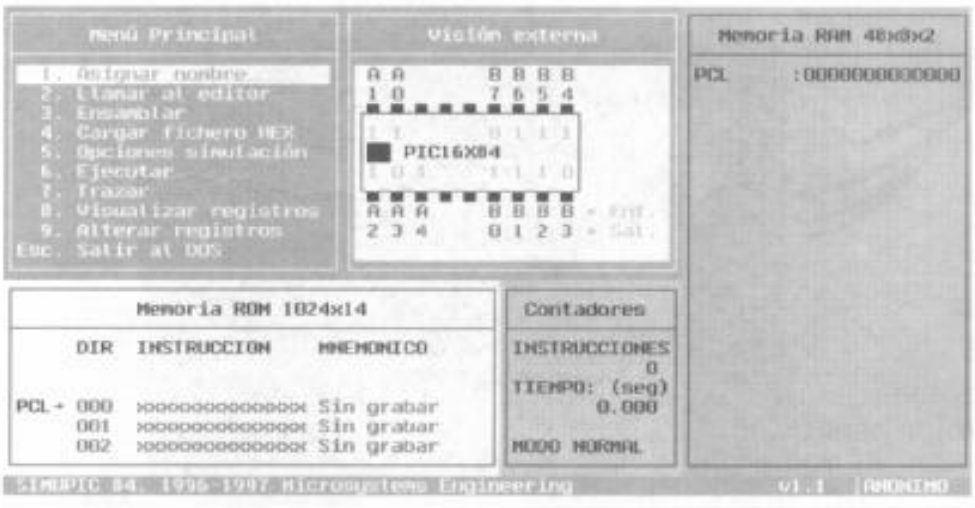


Figura 8.3. *Pantalla principal del simulador SIMUPIC'84. Se trata de un excelente, didáctico y económico simulador software para los PIC16X84.*

El grabador **Micro'PIC Programmer** es un buen exponente de este tipo de herramientas y es capaz de grabar todos los PIC de la gama media de 18, 28 y 40 patitas. (Figura 8.4.)

8.2.6. Sistema de desarrollo

Es una herramienta completa, muy práctica y económica que sustituye de alguna forma la carencia del emulador en circuito. Consiste en una tarjeta de circuito impreso controlada por el puerto paralelo de un PC y que aloja un grabador de microcontroladores, circuito estabilizador de tensiones de alimentación y gran parte de los periféricos más usados en las aplicaciones con microcontroladores. Su idea de aplicación es grabar inicialmente el programa en la memoria del microcontrolador y después conectarle los periféricos disponibles para ejecutar el programa en tiempo real y con total exactitud. Los errores hardware y software pueden resolverse sobre la misma tarjeta y volver a ejecutar el programa hasta su correcto funcionamiento.

Con un precio muy atractivo el **Micro'PIC Trainer** de **Microsystems Engineering** contiene todo lo necesario para desarrollar un proyecto con un PIC de la gama media de 18 o 28 patitas. Posee un grabador de PIC, que en caso de ser tipo PIC16X84 también es borrador, un circuito de estabilización de las tensiones utilizadas y un conjunto de periféricos (interruptores, pulsador, potenciómetros, leds, display de 7 segmento y pantalla LCD), con los que se puede editar, depurar, ensamblar, grabar, comprobar el funcionamiento real, borrar y repetir el proceso hasta tener a punto el proyecto. (Véase Figura 8.5.)

8.2.7. Tarjeta de prototipos

Consiste en una placa de circuito impreso donde viene montado el zócalo del microcontrolador, la circuitería auxiliar y la estabilización de la alimentación. Además, dispone de una

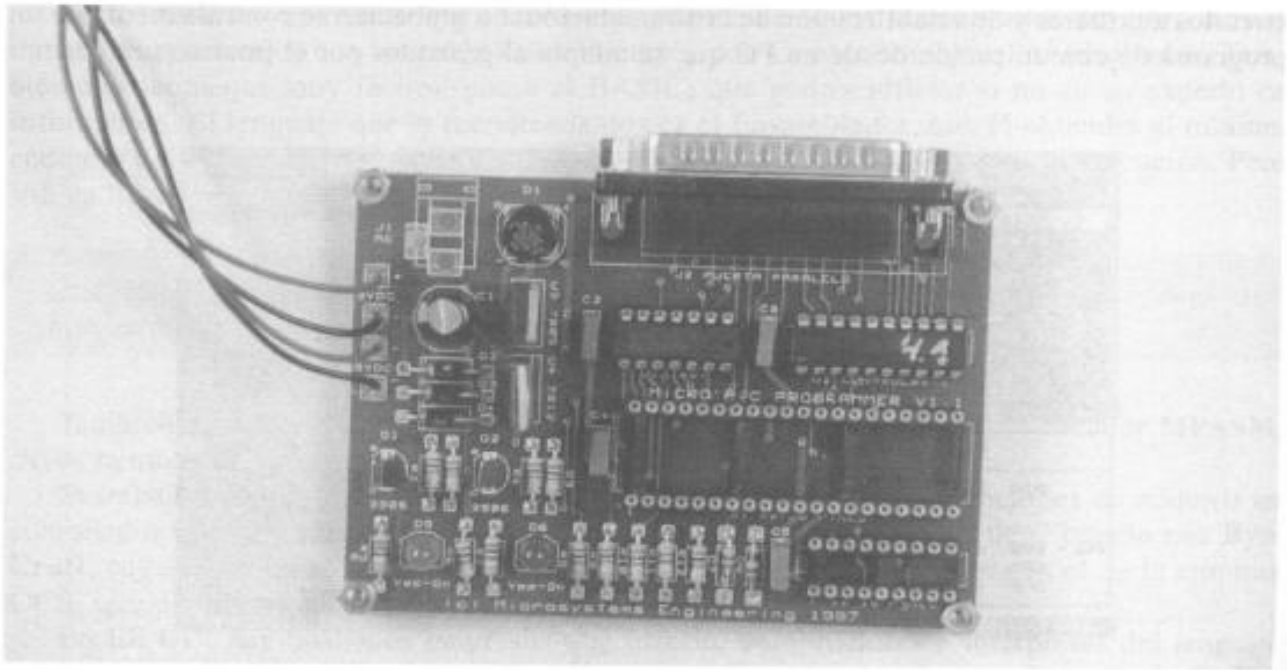


Figura 8.4. *Fotografía del grabador Micro'PIC Programmer diseñado por* ***Microsystems Engineering.***

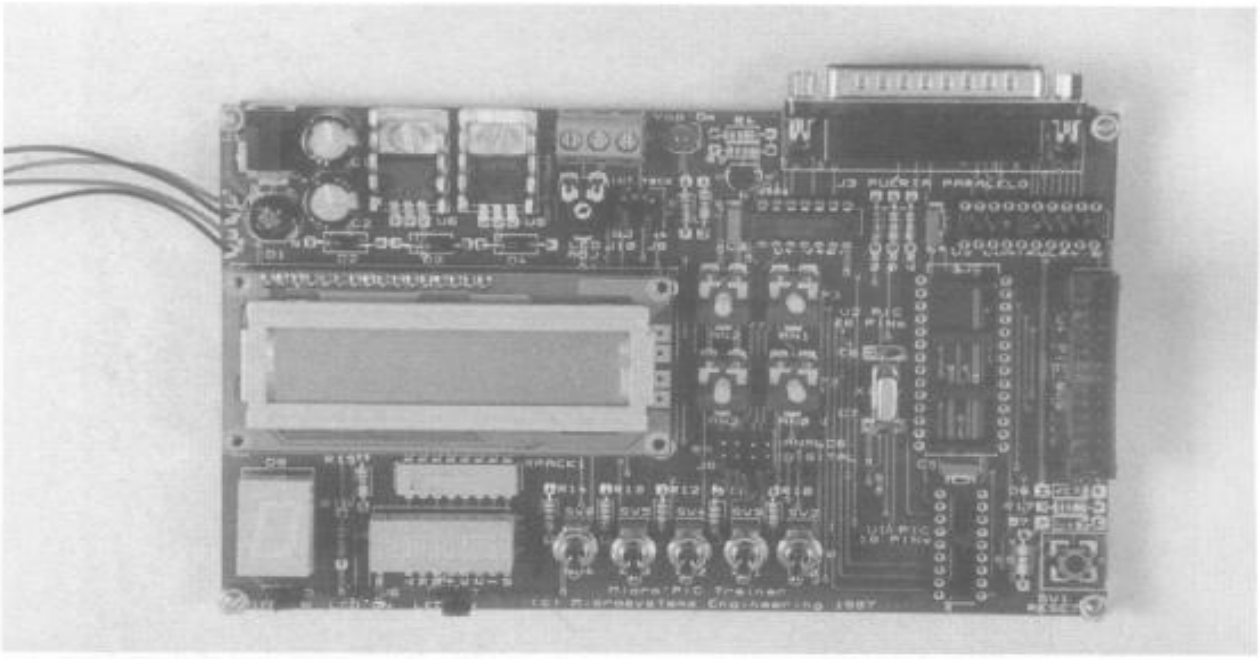


Figura 8.5. *Fotografía del sistema de desarrollo Micro'PIC Trainer.*

zona agujereada para soportar y conectar los periféricos específicos de la aplicación. Insertando el microcontrolador con el programa grabado en su zócalo queda completado el prototipo definitivo.

Confeccionado y depurado el programa, grabado el microcontrolador y comprobado el comportamiento real de gran parte del software mediante los periféricos del sistema de desarrollo, se pasa al montaje del prototipo. Una parte de la circuitería del prototipo siempre es igual, porque es la que corresponde a la alimentación y la auxiliar que acompaña al microcontrolador (cristal y Reset). Lo que cambian son los periféricos a controlar y que se conectan con las patitas de E/S. Para ellos se destina el área libre agujereada.

En la Figura 8.6 se muestra una fotografía de la tarjeta de prototipos **PIC18-ME** de **Microsystems Engineering**, que es la que hemos usado en la implementación de los ejercicios presentados en este libro. Está montada sobre un pupitre de metacrilato que también contiene el Micro'PIC Trainer y un transformador de alimentación, configurando una utilísima herramienta de laboratorio.

8.3. EMULADOR EN CIRCUITO

Es una herramienta cara pero muy potente, que permite realizar un seguimiento completo y rápido de todas las fases de un proyecto. Con ella se reproduce el comportamiento del microcontrolador, pero ahora al emplearse hardware y software se consigue una similitud total de las funciones. Se trabaja en tiempo real y se pueden conectar los periféricos externos a controlar para analizar los resultados de forma idéntica a la que se produciría si fuese el microcontrolador emulado el que estuviese ejecutando el programa de la aplicación.

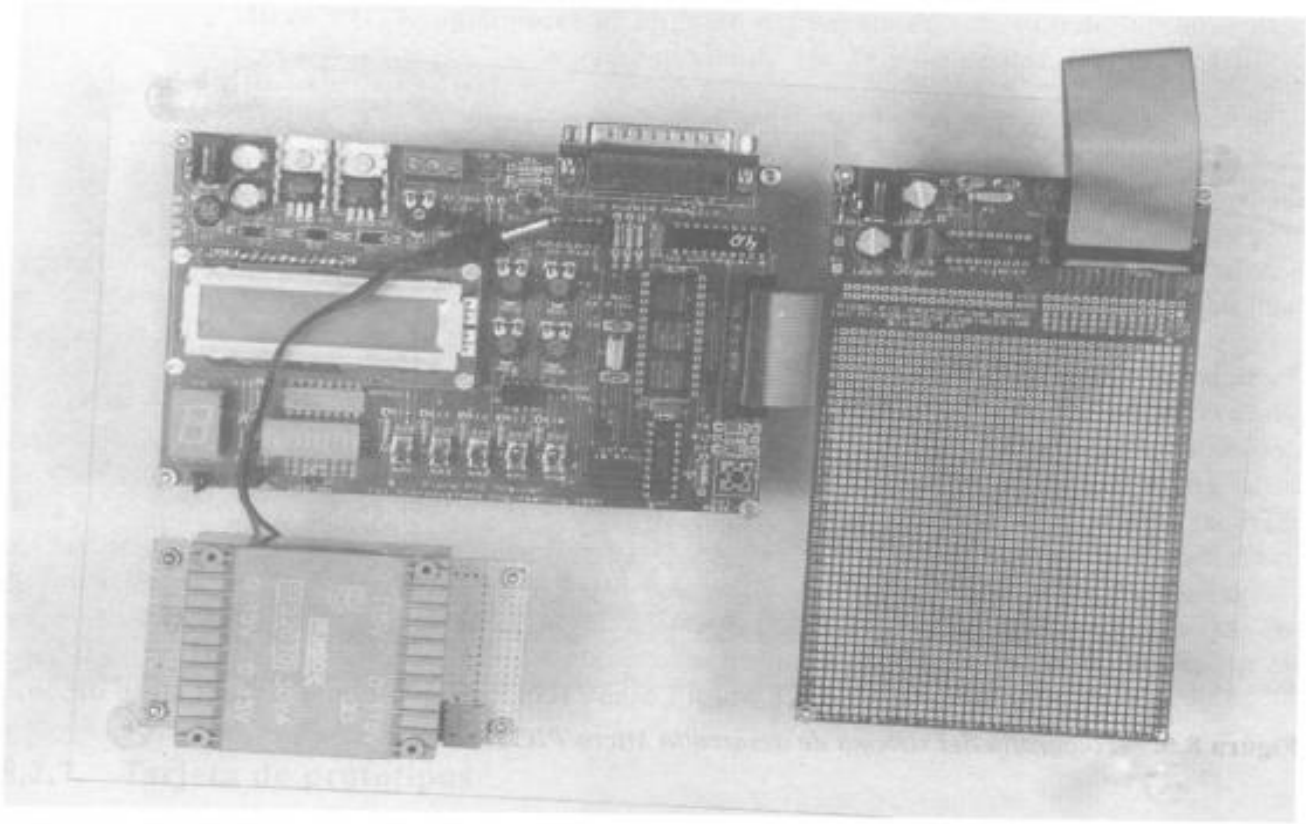


Figura 8.6. *Fotografía de la tarjeta de prototipos PIC18-ME sobre un pupitre de metacrilato que soporta al Micro'PIC Trainer y un transformador de alimentación.*

Es un equipo electrónico controlado mediante un programa desde un PC y del que sale una sonda que reemplaza al microcontrolador en la tarjeta de prototipos. Las ayudas que ofrece al diseñador son excelentes al integrar todos los programas que se precisan, permitiendo una depuración potente del hardware y del software asociados al proyecto.

En la Figura 8.7 se muestra la fotografía del emulador en tiempo real **ICEPIC-Jnr-5X** de la empresa **R. F. Solutions**, que es capaz de emular a los PIC12C5XX y PIC16C5X hasta una frecuencia de 20 MHz. La Figura 8.8 presenta una pantalla de trabajo de dicho emulador en la que se recoge una amplia y valiosa información para la depuración del proyecto.

8.4. INTRODUCCIÓN AL DISEÑO DE PROYECTOS: PUESTA EN HORA

Presentamos algunos proyectos que hemos considerado suficientes para abrir la mente y estimular la voluntad para que cualquier persona se decida a construir el primero.

Para no tener que emplear más que un microcontrolador siempre se realizarán con un PIC16X84 y el montaje se hará sobre una tarjeta de prototipos **PIC18-ME**, de forma que si se anima y hace varios precise de los mínimos componentes y no se gaste mucho dinero. Para

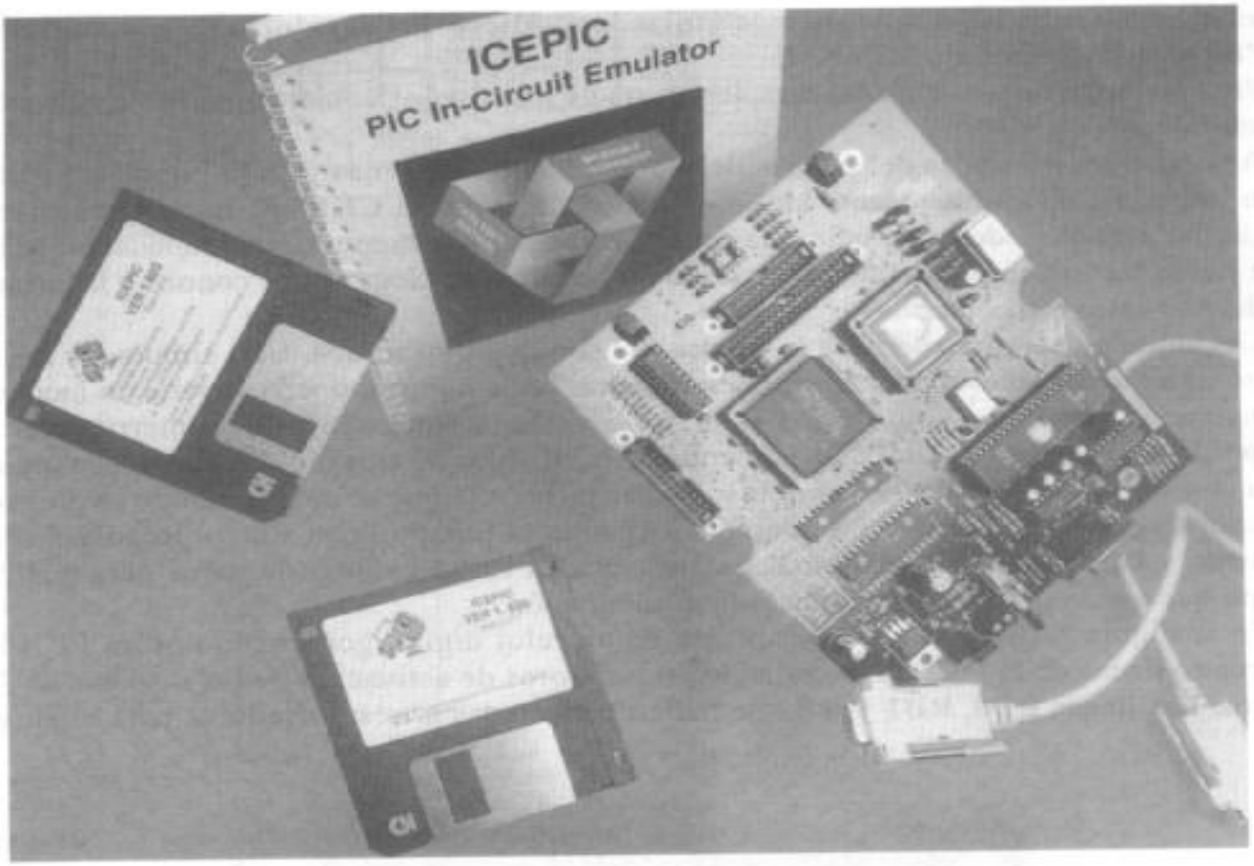


Figura 8.7. *Fotografía del hardware, software y documentación que componen el emulador en circuit ICEPIC-Jnr-5X de* ***R. F. Solutions***.

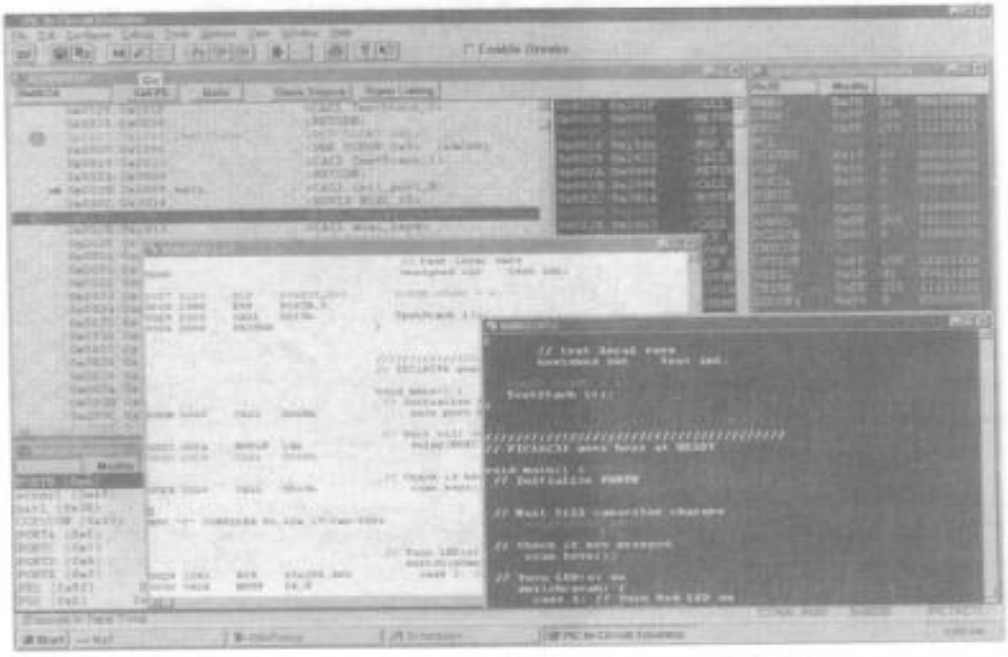


Figura 8.8. *Pantalla de trabajo del emulador ICEPIC-Jnr.*

soportar las fases de los diseños hemos utilizado el **Micro'PIC Trainer**, el simulador **SIMU-PIC'84** y el Ensamblador **MPASM**.

Para poner de relieve algunas características de los diseños comenzamos poniendo en hora un reloj digital.

Aplicaciones derivadas del control del tiempo hay muchísimas y en la Figura 8.9 se muestra la fotografía de un interesante kit comercial de la empresa **CEBEK**, que es un temporizador digital para monederos de máquinas que funcionan por tiempo. Cada moneda que se introduce añade cierto tiempo de funcionamiento de la máquina. Seguro que conoce y ha manejado muchos de estos «sacacuartos».

Las entradas del monedero de la Figura 8.9 son dos pulsadores. Uno simula las monedas que se introducen y el otro es la puesta en marcha. Cada nueva moneda supone un incremento del tiempo de funcionamiento cuyo valor se selecciona con cuatro microinterruptores. Hay dos periféricos de salida. Uno consiste en un visualizador de tres dígitos que puede presentar hasta un tiempo máximo de 999 minutos. El otro es un relé que se activa cuando se presiona el pulsador de puesta en marcha y se desactiva cuando el tiempo llega a cero. Regula el funcionamiento y la parada de la máquina. Existe una EEPROM externa de datos para guardar el tiempo que queda útil si se desconecta la alimentación.

En la Figura 8.10 se ofrece el esquema de un reloj digital gobernado por un PIC16C54. Las cuatro líneas de la Puerta A regulan los transistores de activación de los displays de 7 segmentos. Las líneas RB0, RB1 y RB2 se hallan conectadas a tres pulsadores para el ajuste de

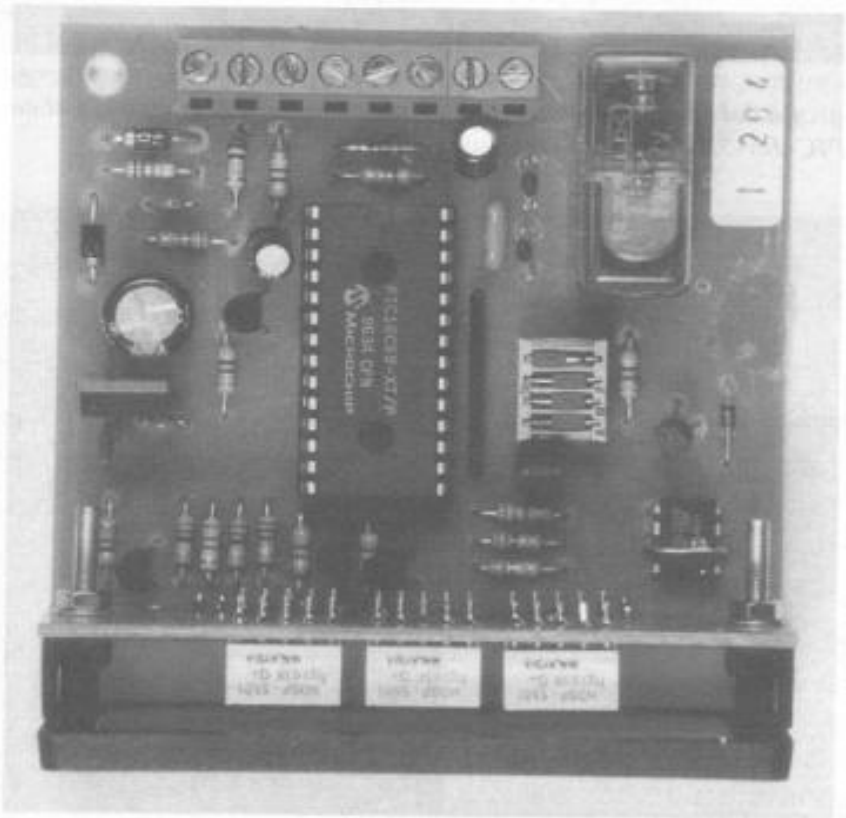


Figura 8.9. *Fotografía de un temporizador digital para máquinas que funcionan con monedas, construido con un PIC16C55 y comercializado por CEBEK.*

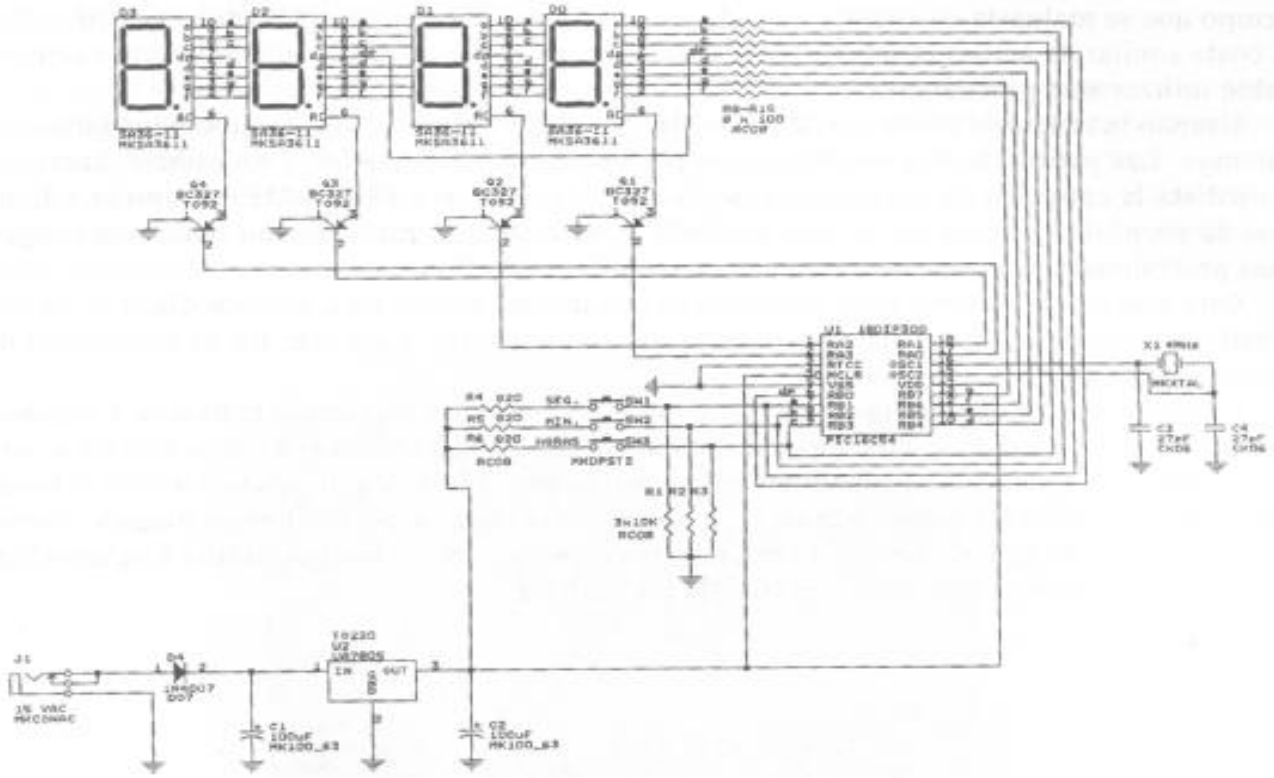


Figura 8.10. *Esquema electrónico de un reloj digital basado en un PIC16C54.*

horas, minutos y segundos. También RB1 y RB2, junto con las 5 líneas de más peso de la Puerta B, excitan a los segmentos de los displays.

Para que aprecie las ventajas de usar una tarjeta de prototipos hemos montado el reloj digital sobre una placa de propósito general, que se muestra en la Figura 8.11. El enorme

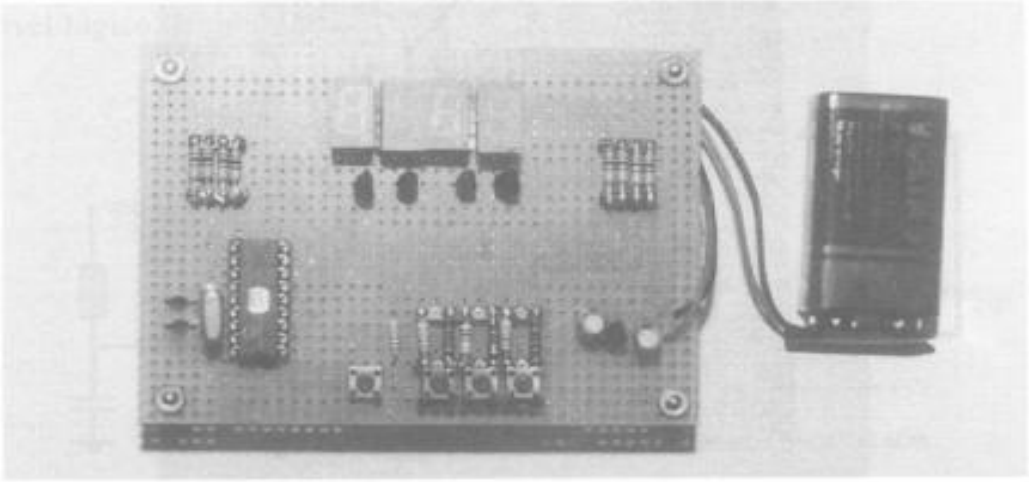


Figura 8.11. *Montaje del reloj digital sobre una tarjeta de circuito impreso de propósito general. Tiempo, coste, fallos y presentación hacen que este procedimiento sea poco interesante.*

tiempo que se malgasta en situar y conectar los componentes, los posibles errores de montaje, el coste similar de ambas tarjetas y la mediocre presentación del prototipo no hacen recomendable utilizar este procedimiento.

Usando la tarjeta de prototipos **PIC18-ME** se reducen al mínimo el tiempo y los fallos del montaje. Las patitas de E/S del PIC están perfectamente señalizadas y accesibles, haciendo inmediata la conexión de los periféricos. El coste de la tarjeta **PIC18-ME** es similar a la de una de propósito general del mismo acabado. Finalmente, la presentación tiene una imagen más profesional y el riesgo de averías es nulo. (Figura 8.12.)

Otra ventaja de las tarjetas de prototipo es que pueden usarse para muchos diseños, ya que finalizado uno de ellos pueden dessoldarse los componentes. Con este fin es recomendable montar los periféricos con rapinado.

En este proyecto el módulo principal consiste en conseguir un tiempo patrón de 1 segundo que vaya incrementando el contador de segundos. Al alcanzar el valor de 60 se pondrá a cero el contador de segundos y se incrementará el de minutos. El de minutos pasará a cero al llegar a 60 e incrementará las horas, y éstas pasarán a cero al llegar a 24. No hemos juzgado interesante incluir el programa, pero si le interesa puede solicitarlo a **Microsystems Engineering**, que también comercializa el kit y el PIC16C54 grabado.

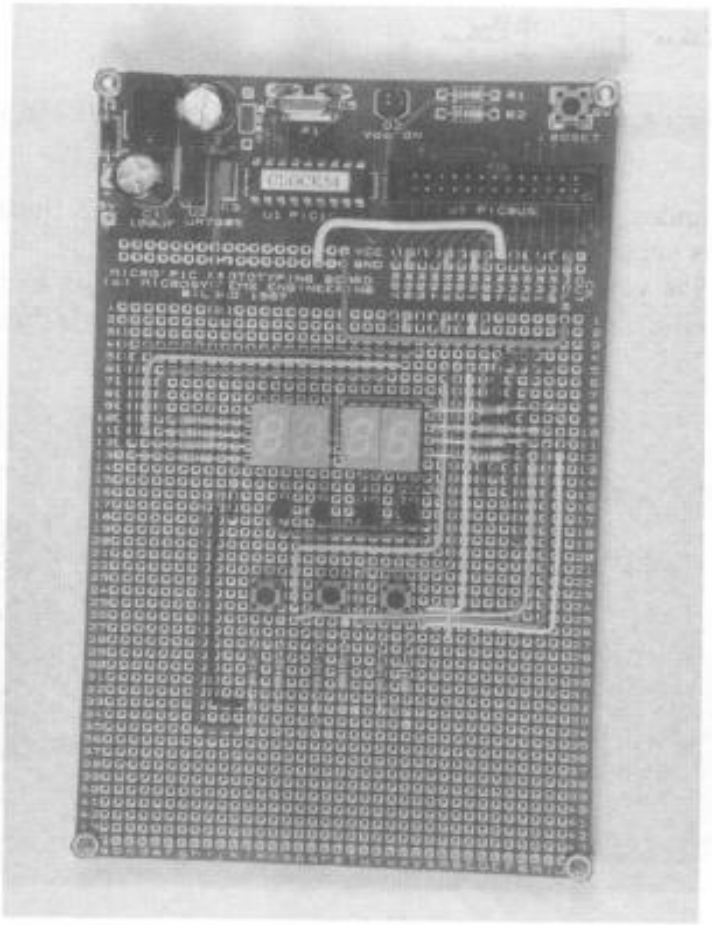


Figura 8.12. *Fotografía del reloj digital montado sobre una tarjeta de prototipos PIC18-ME.*

8.5. UN DADO ELECTRÓNICO

Se describe la construcción de un dado electrónico que simula las seis caras de los reales y que también puede «tirarse» para sacar un número aleatorio del 1 al 6. Para la interpretación electrónica usaremos un display de 7 segmentos sobre el que se visualizará el número de cada tirada y, también, un pulsador que indique el momento de la tirada. En la Figura 8.13 se muestra la fotografía de un kit comercial basado en el PIC16C54, que implementa esta aplicación sobre una cajita de baquelita que contiene la pila de 9 V que precisa su alimentación.

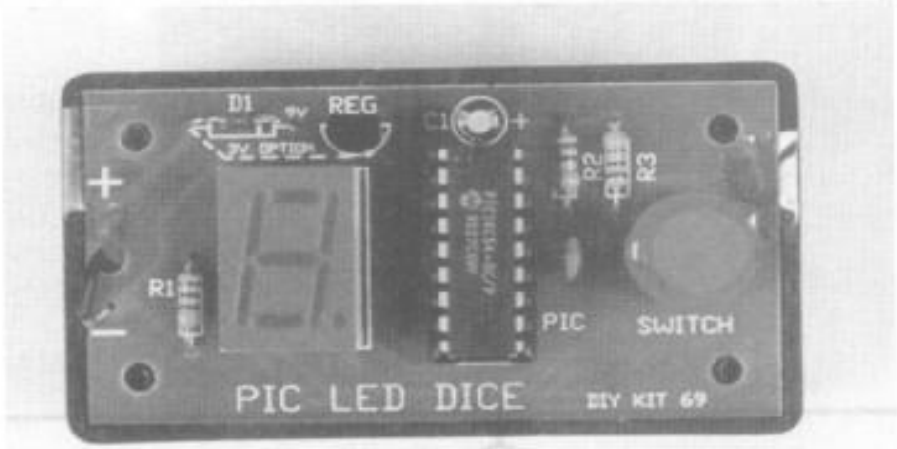


Figura 8.13. *Fotografía del dado electrónico comercializado en kit (DIY KIT 69). El display de 7 segmentos y el pulsador de tirada se controlan con un PIC16C54.*

La Puerta B del PIC, que en la fase de diseño y en la comprobación en el Micro'PIC Trainer puede ser un PIC16X84, se configura como salida y sus líneas se aplican a los segmentos del display como se refleja en la Figura 8.14. La línea de menos peso de la Puerta A recibe el nivel lógico del pulsador de tirada.

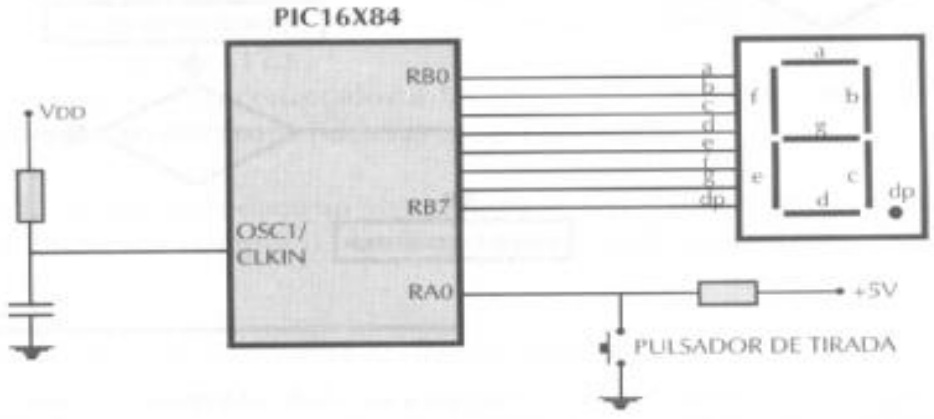


Figura 8.14. *Conexión del PIC a los periféricos que configuran el dado.*

En la Figura 8.15 se presenta el organigrama resumido para el programa del dado.

En la Figura 8.16 se ofrece una fotografía de la implementación especial del dado sobre una tarjeta de prototipos PIC18-ME de **Microsystems Engineering**. Lo más llamativo es la sustitución del display de 7 segmentos por un presentador que simula al display a base de numerosos leds. El programa lo encontrará en el disquete.

En los juegos de azar hay jugadores de ventaja que hacen trampas. Una podría consistir en modificar el programa para que cada cierto número de veces salga un número dado, o incluso para generar una secuencia de números preestablecida. Vd. no debe emplear sus conocimientos para lucrarse de forma poco honesta.

Hay infinidad de juegos que están esperando su aportación para que se jueguen más, sean

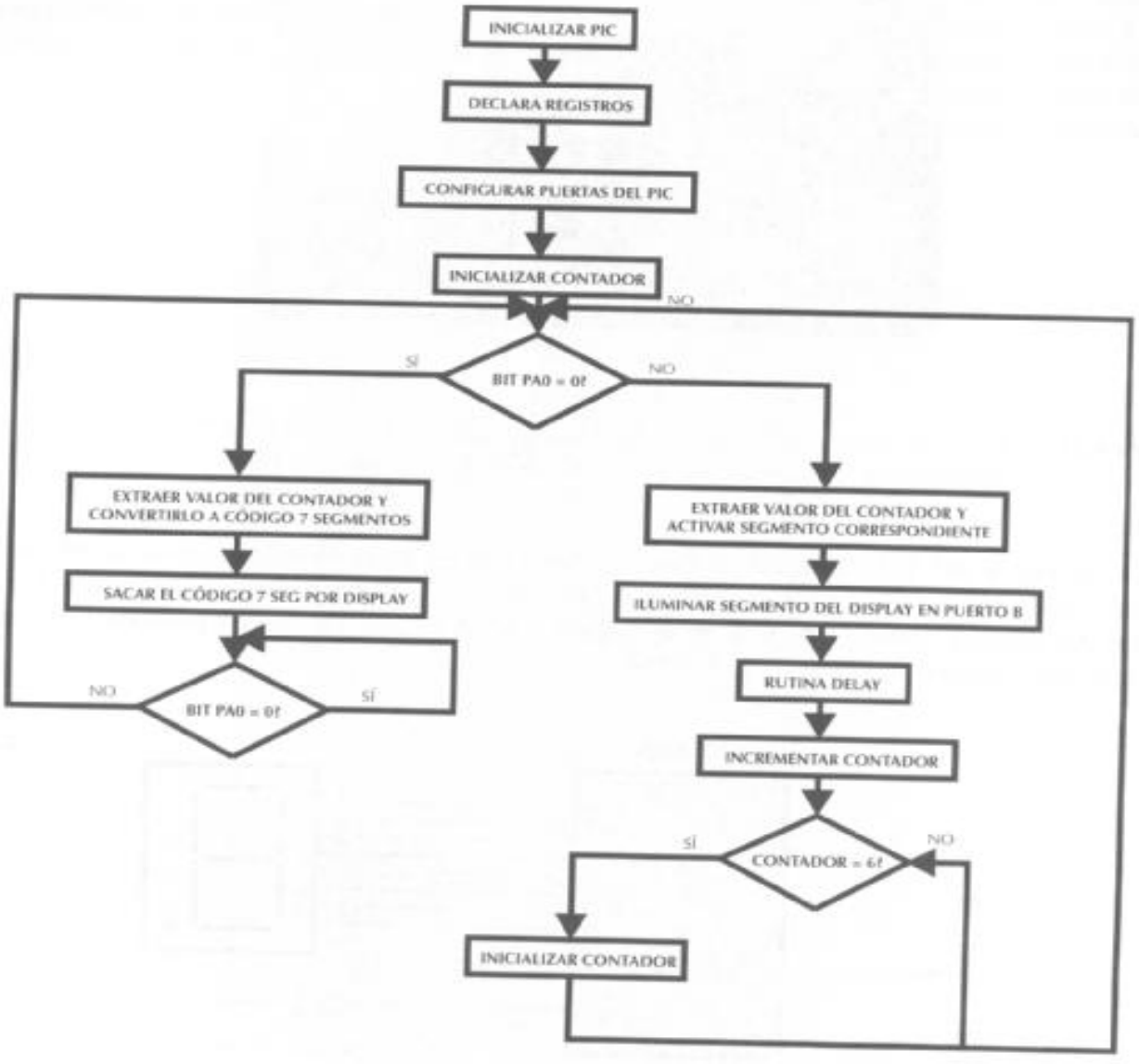


Figura 8.15. *Organigrama general para el programa del dado electrónico. Cuando el pulsador no está activado (RA0 = 1) se van iluminando sucesivamente los segmentos del display para dar cierta vistosidad al juego.*

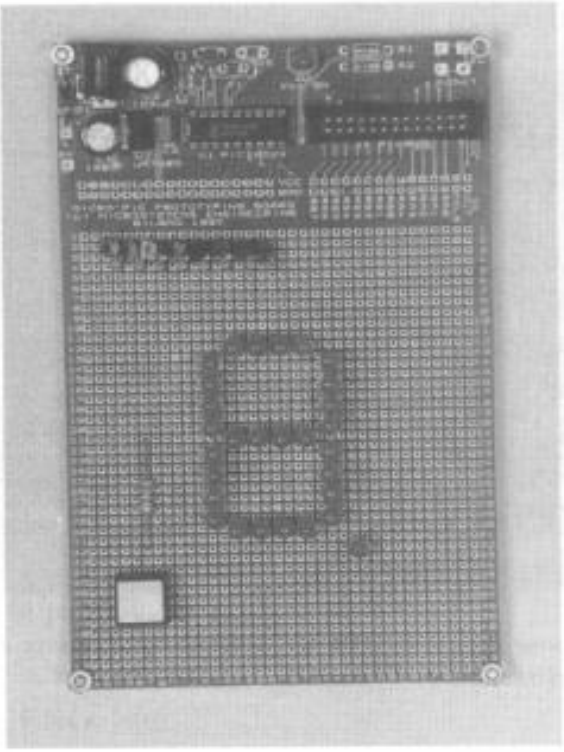


Figura 8.16. *Fotografía del dado electrónico montado sobre una tarjeta de prototipos PIC18-ME.*

más atractivos y hagan disfrutar más a la gente. Las quinielas, la lotería primitiva, la bonoloto, las damas, la oca, el mus y el ajedrez le necesitan. Ayúdelos; además de entretenerse puede entretener a los demás y, de paso, ganar algún premio por su trabajo.

8.6. SEMÁFORO REGULABLE

Este sencillo proyecto va a permitir manejar tiempo y usar la memoria de datos EEPROM. Se trata de un semáforo corriente con tres luces: roja, verde y ámbar, que están simuladas por tres diodos led de dichos colores. La luz ámbar se enciende durante un segundo en la transición de las otras dos. (Véase Figura 8.17.)

Mediante tres pulsadores, conectados a las tres líneas de menos peso de la Puerta A, se regula el tiempo que estarán encendidas las luces roja y verde.

Interruptor RA0: Cuando introduce un nivel lógico alto el sistema funciona en modo AJUSTE y se pueden modificar los tiempos de las luces. Si introduce un nivel lógico bajo funciona en modo NORMAL.

Interruptor RA1: Con nivel alto regula el tiempo de la luz roja y con nivel bajo el de la verde.

Interruptor RA2: Con nivel alto incrementa el tiempo de la luz seleccionada y con nivel bajo se decrementa.

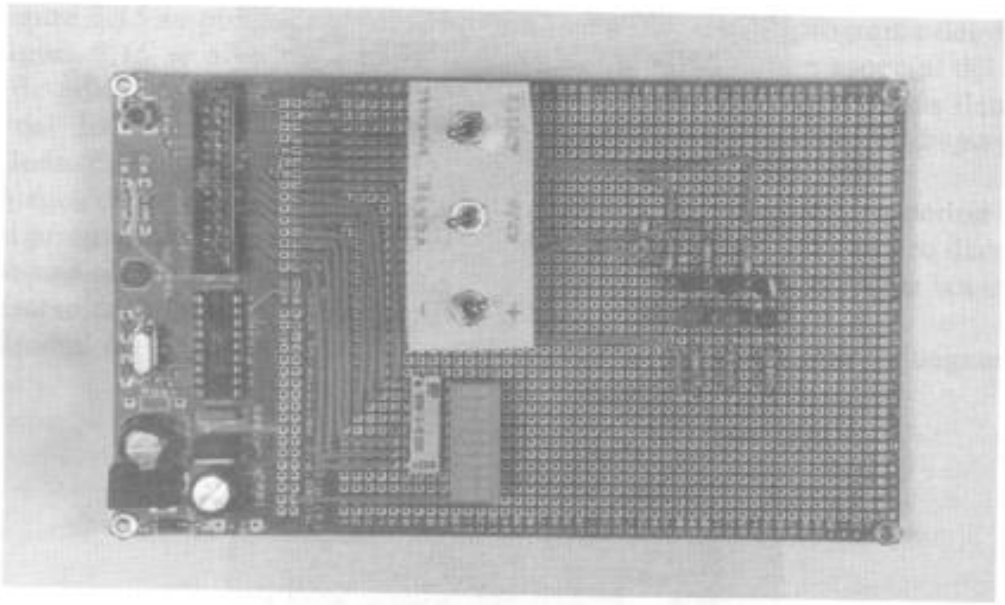


Figura 8.17. *Fotografía de una tarjeta PIC18-ME sobre la que se han montado todos los periféricos que conlleva el semáforo.*

Las 8 líneas de la Puerta B se conectan a 8 diodos led integrados en una barra que tienen la misión de visualizar el número de segundos, de 1 a 8, que está encendida la luz bajo prueba. (Figura 8.18.)

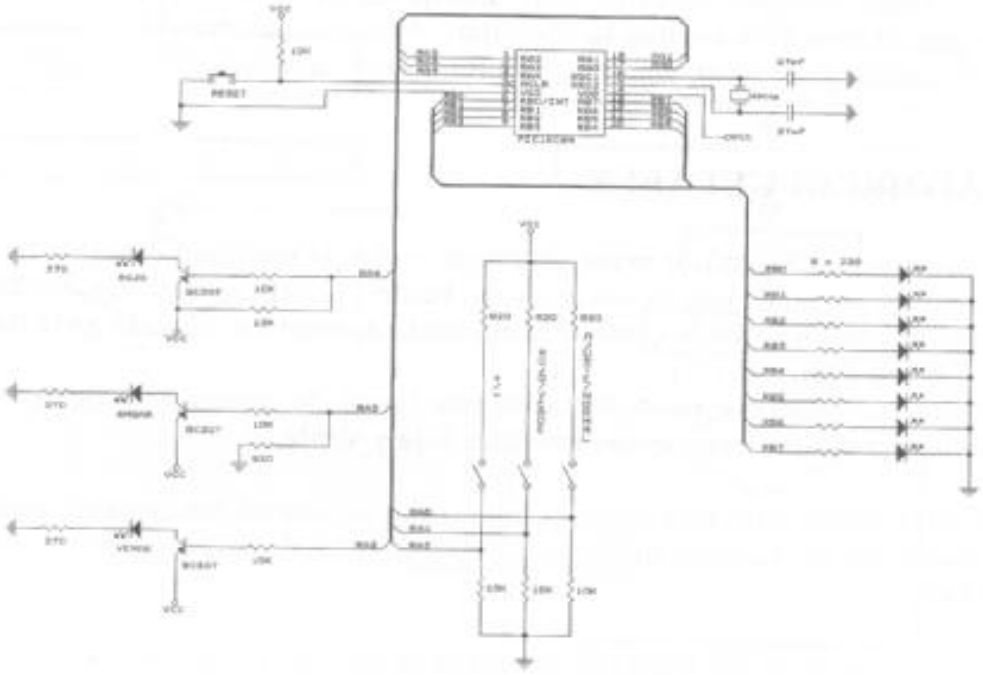


Figura 8.18. *Esquema de un conexionado de un PIC16C84 con los periféricos que implementa el semáforo.*

La duración establecida para el encendido de las luces roja y verde se registran en posiciones de la memoria de datos EEPROM para no realizar una nueva reprogramación cuando se desconecta la alimentación.

El programa de esta aplicación lo encontrará en el disquete del libro.

8.7. PRESENTADOR DE MENSAJES

Es un buen ejercicio para experimentar el manejo de pantallas LCD, que en este proyecto se usan para visualizar diversos mensajes previamente grabados. Concretamente, se pueden visualizar hasta 8 mensajes diferentes de acuerdo con el código binario que introduzcan tres interruptores, conectados a las tres líneas de menos peso de la Puerta A. La Figura 8.19 muestra el aspecto de esta aplicación montada sobre una tarjeta de prototipos PIC18-ME y la Figura 8.20 el esquema de conexionado.

El programa de esta aplicación, debidamente comentado, se halla en el disquete, donde también podrá encontrar el programa HOLA, que contiene las principales rutinas básicas que se necesitan para manejar la pantalla LCD.

8.8. CLAVE DE ACCESO

Como último proyecto se describe una aplicación que emplea a los dos periféricos más clásicos en el control de los microcontroladores: teclado y pantalla LCD.

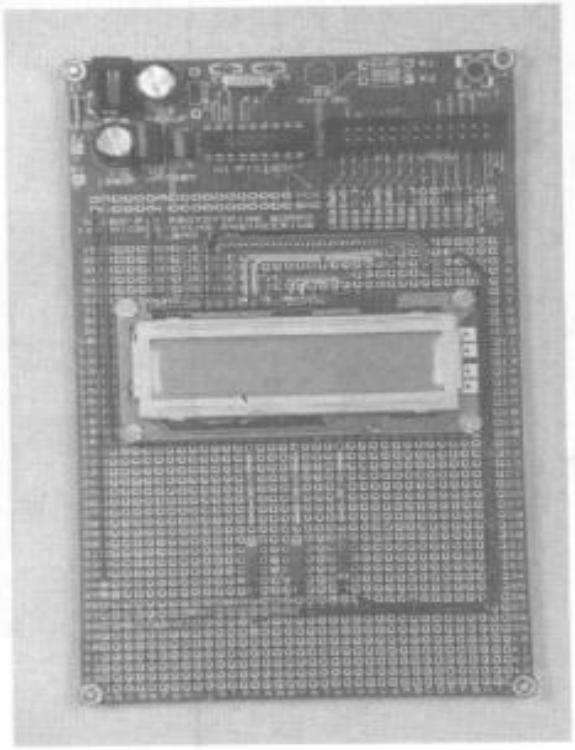


Figura 8.19. *Fotografía del presentador de mensajes implementado sobre una tarjeta PIC18-ME.*

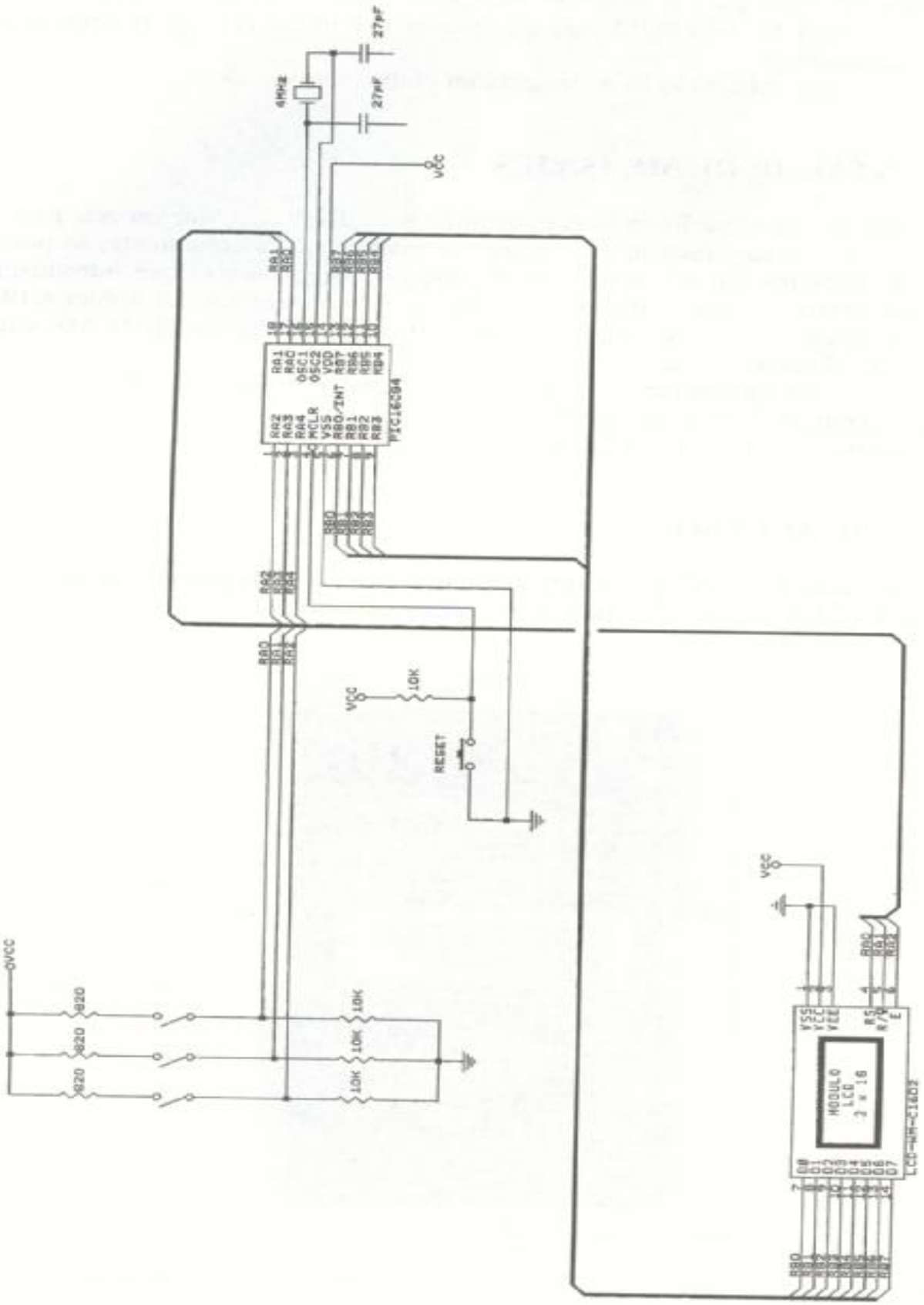


Figura 8.20. *Esquema de conexionado del presentador de mensajes.*

Para controlar el acceso a un recinto se debe pulsar sobre un teclado la tecla A seguida de otras cuatro con diferentes dígitos, que constituyen la clave. Si la clave es correcta se activa durante un segundo un relé, conectado a la línea RA4, que abre la puerta.

Se dan tres oportunidades para introducir la clave correcta. Además, se utiliza un zumbador piezoeléctrico que genera un sonido «beep» tras la pulsación de cada tecla.

Con la tecla C se puede cambiar la clave y la pantalla LCD genera una serie de mensajes durante la operación. La clave queda registrada en la memoria EEPROM de datos de forma permanente.

La Figura 8.21 muestra la disposición de los periféricos sobre la tarjeta PIC18-ME y la Figura 8.22 ofrece el esquema de conexionado de los mismos con el PIC.

El programa de la clave se ha incluido en el disquete.

Ahora le toca a Vd. intentar repetir estos diseños o, ¡mejor aún!, crear otros nuevos. Si en algo le podemos ayudar nos brindamos a hacerlo. Nuestras direcciones de correo electrónico son *jmangulo@inf.deusto.es* y *msyseng@arrakis.es*.

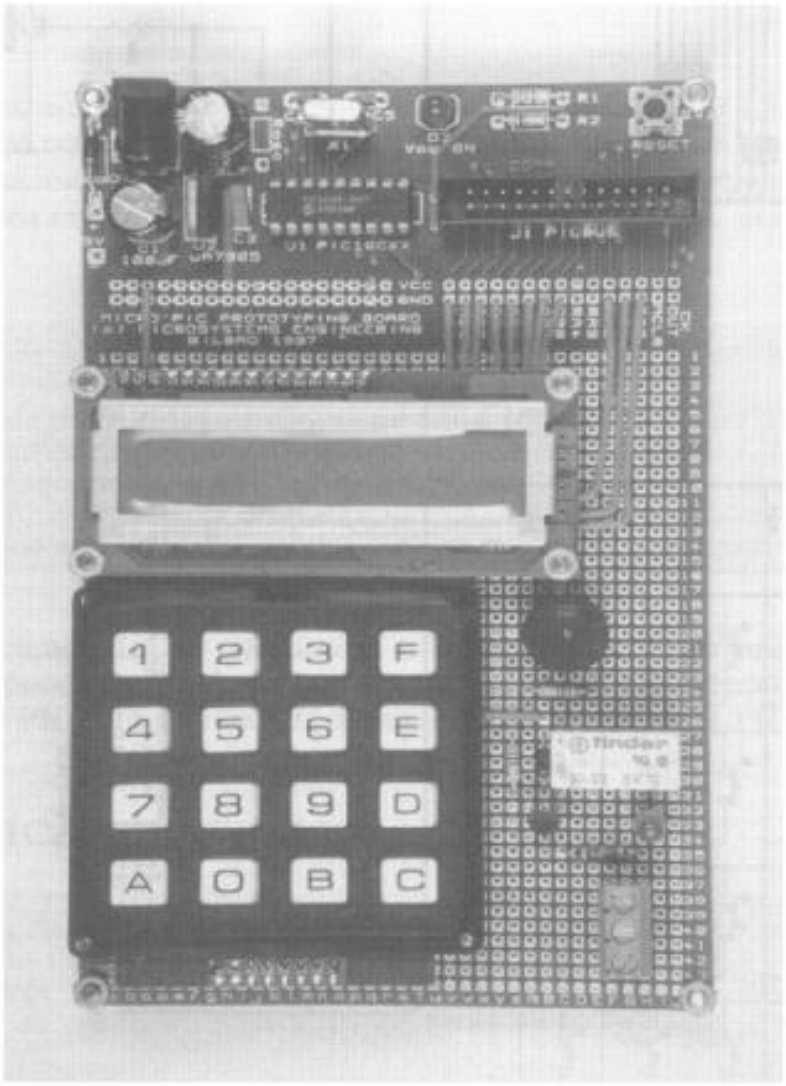


Figura 8.21. *Fotografía del sistema de control de acceso montado sobre una tarjeta PIC18-ME*

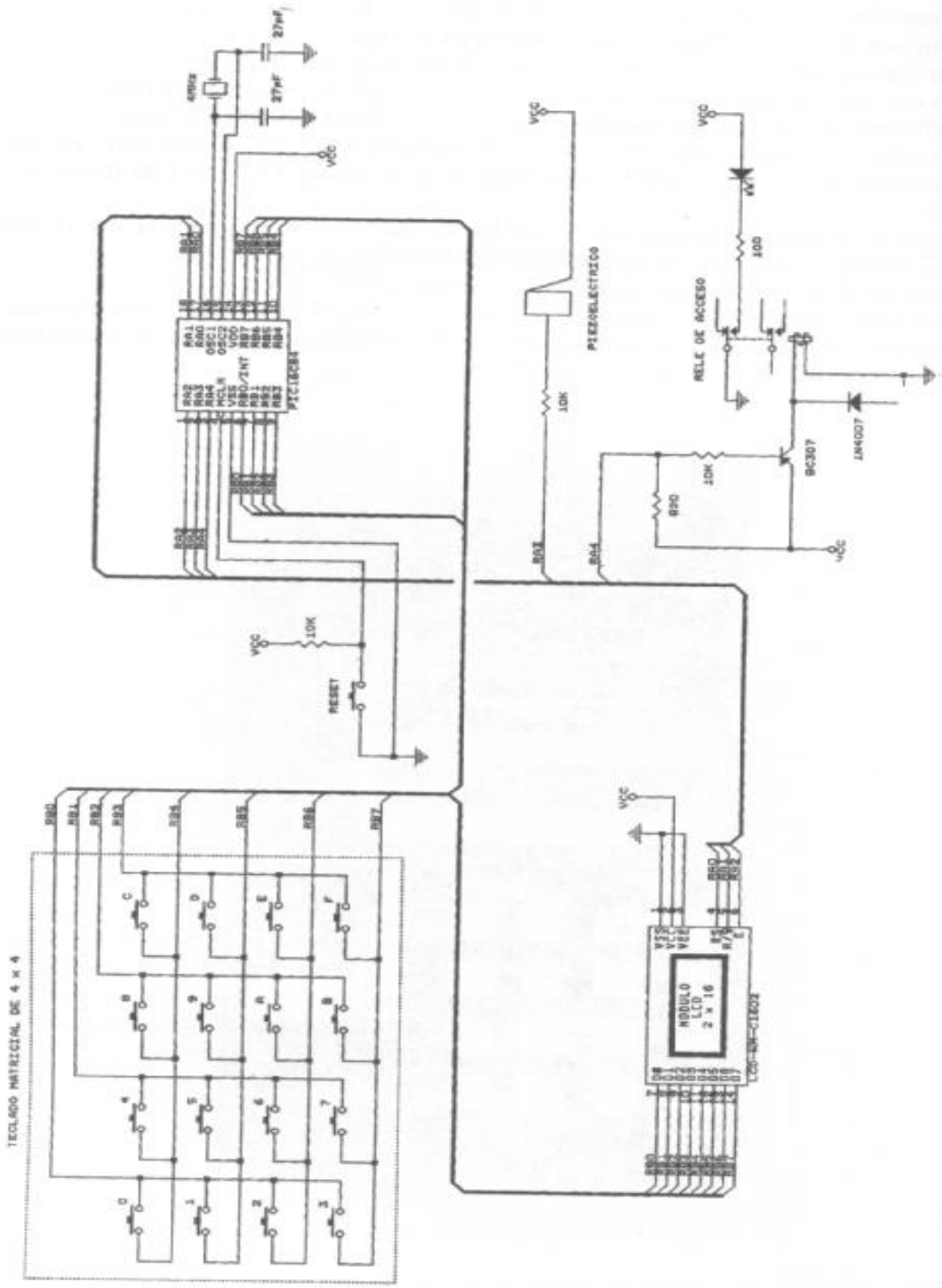


Figura 8.22. *Esquema de conexionado del sistema de control de acceso mediante una clave secreta.*

GUÍA RÁPIDA DEL PIC16X84



INTRODUCCIÓN

Todos los programas, ejercicios y proyectos de este libro pueden ser implementados con un PIC16X84, bien sea la versión EEPROM con el PIC16C84, o la versión Flash con el PIC16F84. La razón de esta elección es que sobre su memoria de instrucciones se pueden grabar y borrar los programas. Ambos modelos de microcontroladores son idénticos, excepto en los siguientes detalles:

1.° *La memoria de instrucciones del PIC16C84 es de tipo EEPROM, mientras que la del PIC16F84 es Flash.*
2.° *La memoria de datos RAM del PIC16C84 tiene 36 bytes y la del PIC16F84 tiene 68.*
3.° *El nivel activo del bit PWRTE está invertido en ambos modelos. Este bit permite o prohíbe el funcionamiento de un temporizador (POWER-UP) que actúa retrasando el funcionamiento del PIC al producirse un Reset por conexión de la alimentación.*
4.° *En el PIC16F84 se han mejorado algunas características eléctricas.*

Presentamos las características más usadas en la programación y manejo de estos modelos en las aplicaciones, para que al trabajar con ellos se localicen rápidamente los datos precisos. Sabiendo manejar el PIC16F84 es muy fácil pasar a usar los potentes PIC16F87X.

CARACTERÍSTICAS GENERALES

- *Memoria de programa de 1 K x 14. El PIC16C84 de tipo EEPROM y el PIC16F84 de tipo Flash.*
- *Memoria de datos EEPROM de 64 bytes.*
- *Memoria RAM de propósito general de 36 bytes en el PIC16C84 y de 68 bytes en el PIC16F84.*
- *13 líneas de E/S con control individual de dirección.*
- *Frecuencia máxima de funcionamiento de 10 MHz.*

- *Cuatro fuentes de interrupción:*
  a) *Activación de la patita RB0/INT.*
  b) *Desbordamiento del TMR0.*
  c) *Cambio de estado en alguna patita RB7-RB4.*
  d) *Fin de la escritura de la EEPROM de datos.*
- *Temporizador/Contador TMR0 programable de 8 bits.*
- *Perro Guardián.*
- *Encapsulado de 18 patitas.*

DIAGRAMA DE CONEXIONES

La distribución y denominación de las patitas del PIC16X84 se presenta en la Figura A.1.

FUNCIÓN DE LAS PATITAS

OSC1/CLKIN: Entrada externa de los impulsos de reloj o conexión con el cristal de cuarzo.

OSC2/CLKOUT: Salida de Fosc/4 en modo oscilador R-C o conexión con el cristal de cuarzo.

VPP/MCCLR#: En modo grabación por esta patita se introduce la tensión VPP (12 - 14 V DC). En funcionamiento normal esta patita soporta la petición de Reset.

RA0 - RA3: Líneas de E/S de la Puerta A.

RA4/T0CKI: Línea de E/S de la Puerta A o entrada de impulsos de reloj para TMR0.

RB0/INT: Línea de E/S de la Puerta B o petición de interrupción.

RB1 - RB7: Líneas de E/S de la Puerta B.

MEMORIA DE PROGRAMA

Tiene implementadas 1 K palabras de 14 bits cada una, ocupando las direcciones comprendidas entre la 0x0000 y la 0x03FF. Los modelos de la gama media alcanzan 8 K de capacidad máxima. (Véase Figura A.2.)

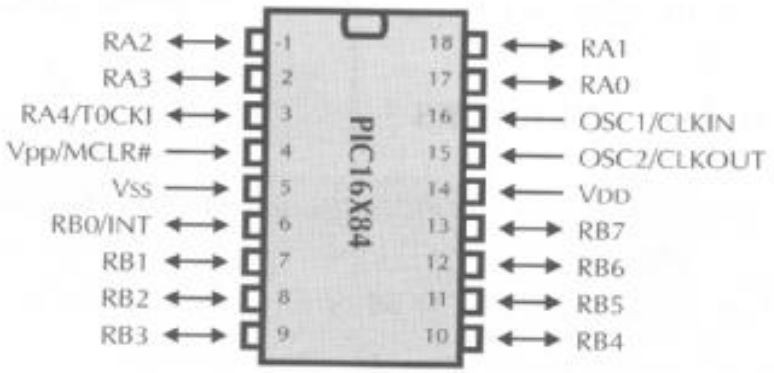


Figura A.1. *Distribución y denominación de las patitas del PIC16X84.*

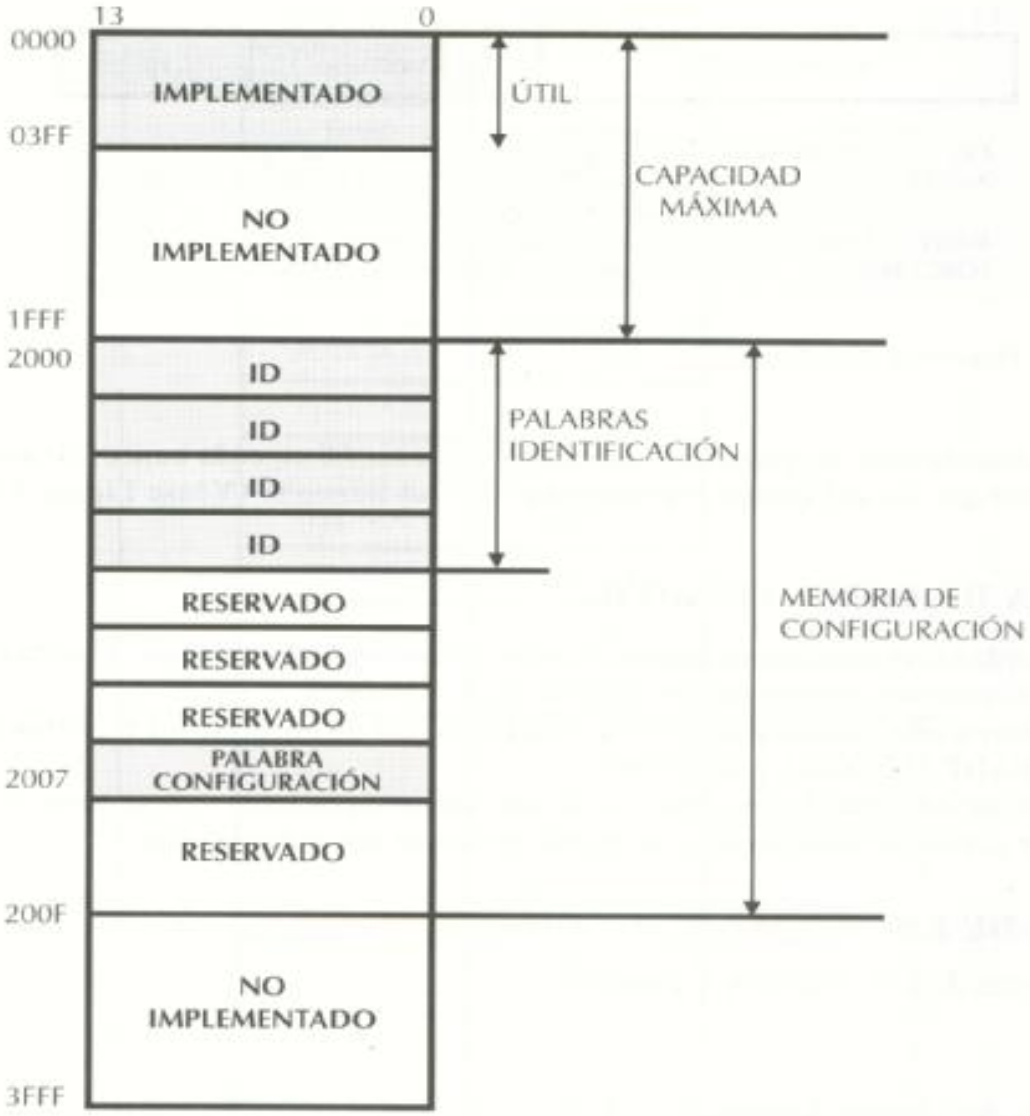


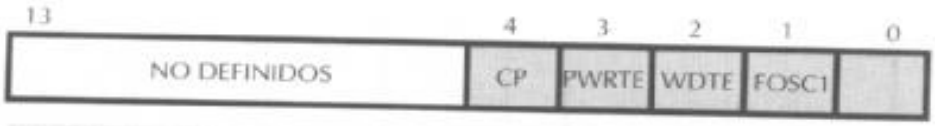
Figura A.2. *Organización de la memoria de instrucciones de los PIC16X84.*

Las posiciones ID de Identificación sirven para que el usuario grabe en sus 4 bits de menos peso los códigos que desee.

La Palabra de Configuración sólo tiene 5 bits válidos y sus símbolos y misiones se muestran en la Figura A.3.

MEMORIA DE DATOS RAM

Consta de dos bancos de 128 bytes cada uno. En el PIC16C84 sólo están implementadas las 48 primeras direcciones de cada banco. En las 12 primeras se ubican los registros de propósito específico (SFR) y en las 36 siguientes los registros de propósito general (GPR). Sin embargo, los 36 registros de propósito general del Banco 1 se mapean sobre el Banco 0, dejando reducidos a 36 los registros operativos de este tipo. En el PIC16F84 la organización de la RAM es



CP: Protección de código (CP = 0 activado)
PWRTE: Control de temporizador "POWER-UP" (En el PIC16C84 PWRTE = 1 activado, en el PIC16F84 al revés)
WDTE: Control del Perro Guardián (WDTE = 1, activado)
FOSC1-FOSC0: Selección oscilador: 00:LP, 01:XT, 10:HS, 11:R-C

Figura A.3. *Palabra de Configuración.*

similar, pero los registros de propósito general llegan a ser 68 en cada banco, siendo efectivos sólo 68, puesto que los del Banco 1 se mapean sobre el Banco 0. (Véase Figura A.4.)

MEMORIA DE DATOS EEPROM

Los datos guardados en esta memoria no se pierden al quitar la alimentación. Consta de 64 bytes que ocupan direcciones comprendidas entre la 0x00 y la 0x3F.

Para acceder a esta memoria en lectura o escritura hay que utilizar los registros de control: EEDATA, EEADR, EECON1 y EECON2.

El proceso de escritura de una posición de esta memoria es lento y dura unos 10 ms. Cuando finaliza se activa un señalizador y se puede provocar una interrupción.

PUERTAS DE E/S

Hay dos puertas de E/S: Puerta A y Puerta B.

Puerta A

- *Consta de 5 líneas denominadas RA4 - RA0.*
- *RA3 - RA0 actúan como líneas de E/S digitales configurables de forma independiente.*
- *RA4/T0CKI, además de línea de E/S, puede funcionar como entrada de los impulsos de reloj para el TMR0.*
- *Si a un bit de TRISA se pone a 1, la línea de E/S correspondiente de la Puerta A actúa como entrada y si se le pone a 0 funciona como salida.*
- *Al leer una entrada se lee el estado actual de la patita.*
- *Las líneas de salida están lacheadas y en ellas se mantiene el último valor que han sacado.*
- *Tras un Reset todos los bits de TRISA se ponen a 1 y las líneas de la Puerta A quedan configuradas como entradas.*
- *Cada línea de la Puerta A suministra una corriente máxima de 20 mA y puede absorber hasta 25 mA. La corriente máxima total que puede absorber la Puerta A es de 80 mA y la que puede suministrar de 50 mA.*

Puerta B

- *Cuando las líneas de la Puerta B se configuran como entradas es posible acoplarlas una resistencia «pull-up» al positivo, poniendo a cero el bit 7 (RBPU) del registro OPTION.*

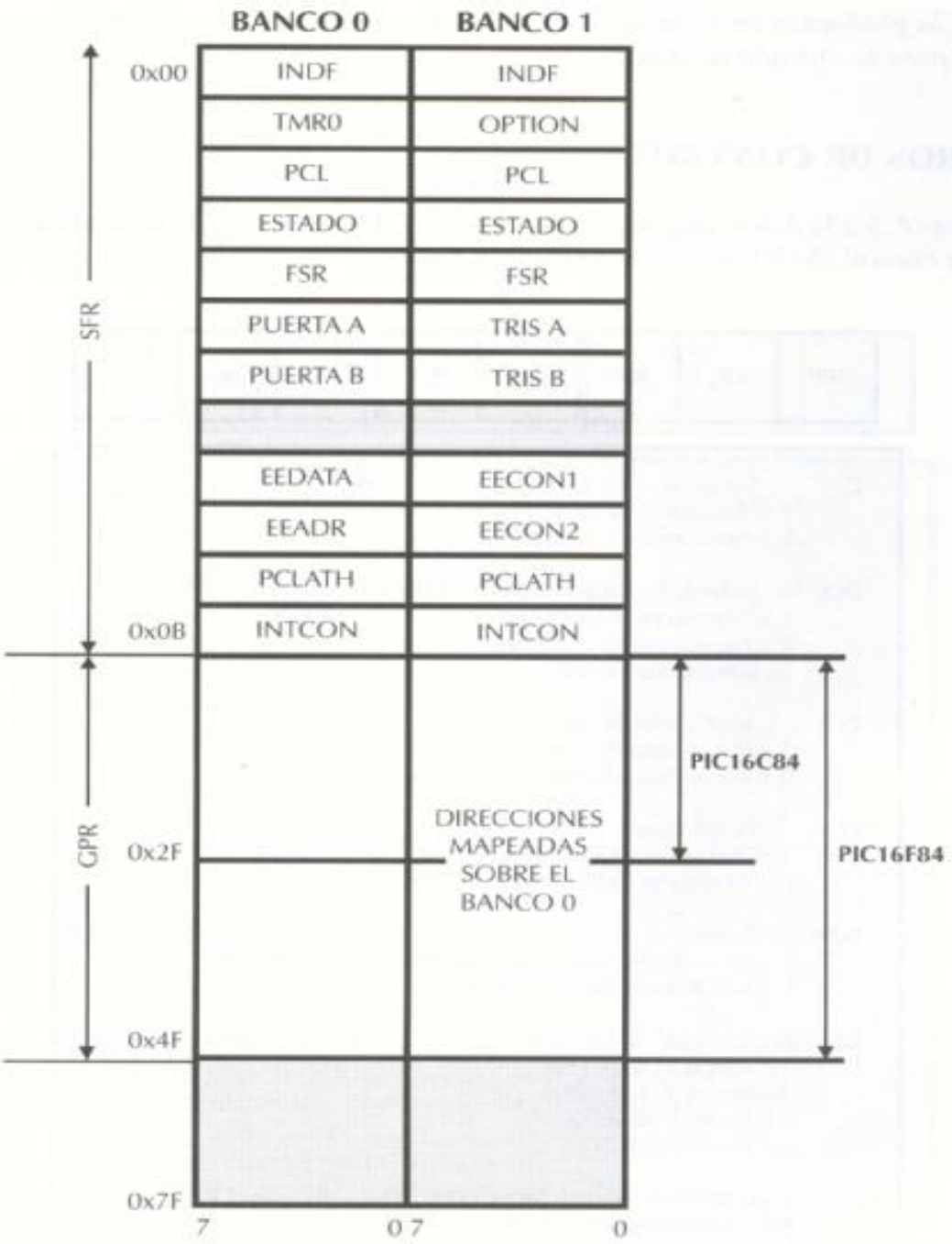


Figura A.4. *Organización de la memoria de datos RAM. En el PIC16C84 sólo hay 36 registros de propósito general, mientras que el PIC16F84 hay 68.*

- *La Puerta B puede absorber una corriente máxima de 150 mA y suministrar un total de 100 mA.*
- *RB0/INT también puede actuar como línea de petición de interrupción.*
- *RB7 - RB4 pueden programarse para soportar una misión especial cuando están configuradas como entradas. Si cambia el estado lógico de una de ellas se provoca una interrupción.*

- *Para la grabación en serie del PIC se usa la patita RB6 para los impulsos de reloj y la RB7 para la entrada de datos.*

REGISTROS DE CONTROL

En las figuras A.5 a la A.9 se expone gráficamente la distribución y la misión de los bits de los registros de control (SFR) más importantes, que son:

IRP	RP1	RP0	TO#	PD#	Z	DC	C

C: Señalizador de acarreo en el octavo bit
1 = Acarreo en la suma y no en la resta
0 = No acarreo en la suma y sí en la resta

DC: Señalizador de acarreo en el 4º bit de menos peso
1 = Acarreo en la suma
0 = No acarreo en la suma
En la resta lo contrario

Z: Señalizador de cero
1 = El resultado de una operación es cero
0 = El resultado de una operación es distinto de cero

PD#: Power Down
1 = Tras conectar VDD o ejecutar "CLRWDT"
0 = Al ejecutar la instrucción "SLEEP"

TO#: Timer Out
1 = Tras conectar VDD o ejecutar "CLRWDT" o "SLEEP"
0 = Al rebasar el tiempo del "WDT"

RP<1:0>: Selección de banco para el direccionamiento directo
11: Banco 3 (180-1FFh)
10: Banco 2 (100-17Fh)
01: Banco 1 (80-FFh)
00: Banco 0 (00-7Fh)

Cada banco es de 128 bytes. El PIC16C84 sólo usa el RP0 y RP1 debe mantenerse a cero

IRP: Selección de bancos para el direccionamiento indirecto
0 = Bancos 0 y 1 (00-FFh)
1 = Bancos 2 y 3 (100-1FFh)
El IRP no se usa en el PIC16C84 y debe mantenerse a cero

GIE: Activación global de interrupciones
1 = Concedido el permiso de interrupciones
0 = No hay posibilidad de interrupciones

Figura A.5. *Registro de ESTADO.*

a) Registro de ESTADO (Figura A.5)
b) Registro OPTION (Figura A.6)
c) Registros de Direccionamiento Directo e Indirecto (Figura A.7)
d) Registro de control EECON1 para la EEPROM (Figura A.8)
e) Registro de control de las interrupciones INTCON (Figura A.9)

RBPO#	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

PS2: PS0 Rango con el que actúa el Divisor de frecuencia

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

PSA: Asignación del Divisor de frecuencias
1 = El Divisor de frecuencia se le asigna al WDT
0 = El Divisor de frecuencia se le asigna al TMR0

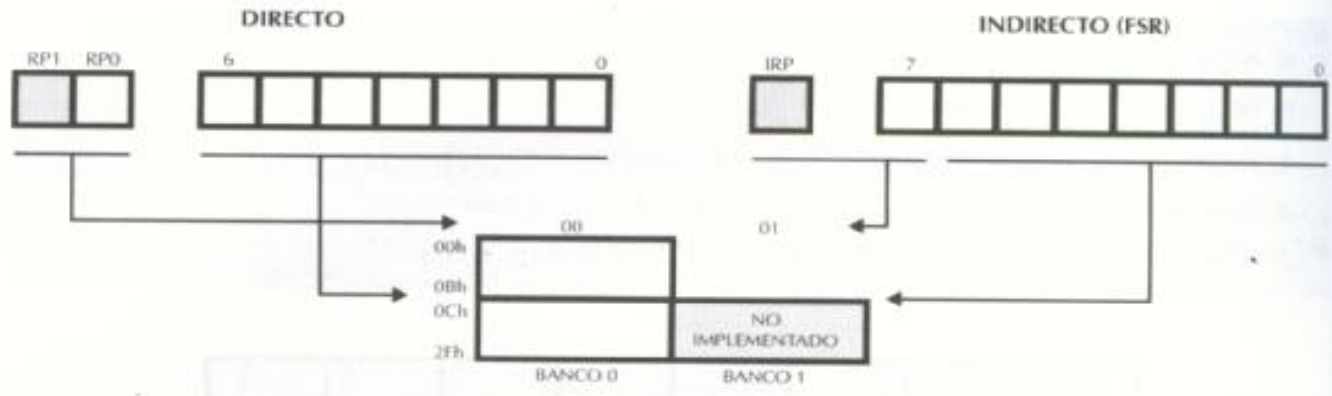
TOSE: Tipo de flanco en T0CKI
1 = Incremento de TMR0 cada flanco descendente
0 = Incremento de TMR0 cada flanco ascendente

TOCS: Tipo de reloj para el TMR0
1 = Pulsos introducidos a través de T0CKI (contador)
0 = Pulsos de reloj interno Fosc/4 (temporizador)

INTEDG: Flanco activo interrupción externa
1 = Flanco ascendente
0 = Flanco descendente

RBPO#: Resistencias Pull-up Puerta B
1 = Desactivadas
0 = Activadas

Figura A.6. *Registro OPTION.*



DIRECTO

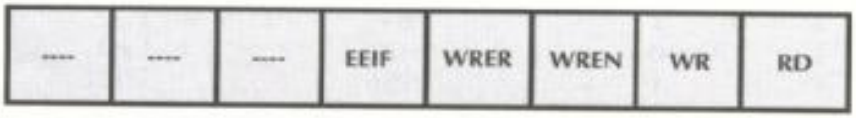
Los 7 bits de menos peso que forman el código OP de la instrucción, seleccionan una de entre 128 posiciones diferentes de la memoria de datos. Los bits <6:5> del registro de estado (RP1:RP0) seleccionan el banco.

Para acceder al banco 1 es obligatorio poner previamente a 1 el bit RP0.

INDIRECTO

Si como operando de una instrucción se accede al registro INDF (posición 0 área de datos), se accede realmente a la posición que en ese momento indique el registro FSR (posición 4 del área de datos). Los 7 bits de menos peso del (<6:0>) seleccionan la posición; el bit <7> del FSR, junto con el bit IRP del registro de Estado (bit <7>), seleccionan el banco. En el PIC16C84 sólo están implementados los bancos 0 y 1, por lo que el bit IRP debe valer "0"

Figura A.7. *Registros de Direccionamiento Directo e Indirecto.*



Bits 7-5: No implementados. Se leen como 0

EEIFE: Señalizador de interrupción de la operación de escritura
1 = La escritura ha terminado
0 = La escritura no se ha terminado o no ha comenzado

WRERR: Señalizador de error
1 = La escritura ha terminado prematuramente
0 = La escritura se ha completado

WREN: Activación de escritura
1 = Permite escribir
0 = Prohíbe escribir en la memoria de datos EEPROM

WR: Control de escritura
1 = Inicia un ciclo
0 = Prohibición de comienzo de ciclo de escritura en la memoria de la EEPROM

RD: Control de lectura
1 = Comienzo de lectura
0 = Prohibición de comienzo de lectura

Figura A.8. *Registro de control EECON1 para la EEPROM de datos.*

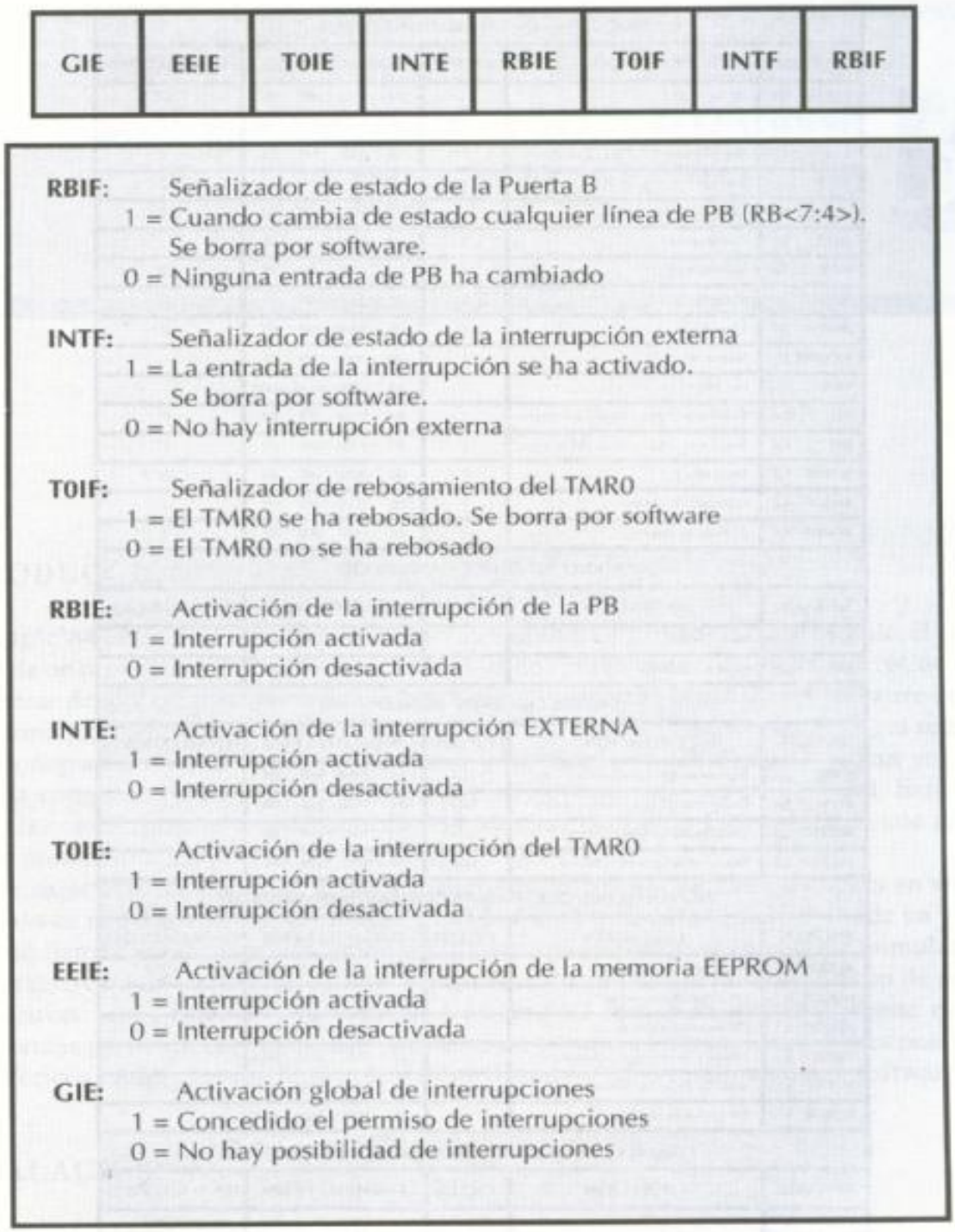


Figura A.9. *Registro de control de las interrupciones INTCON.*

JUEGO DE INSTRUCCIONES

En la Figura A.10 se ofrece el conjunto de las 35 instrucciones que componen el repertorio del PIC16X84.

INSTRUCCIONES QUE MANEJAN REGISTROS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
ADDWF f,d	Suma W y f	1	00 0111 dfff ffff	C,DC,Z
ANDWF f,d	AND W con f	1	00 0101 dfff ffff	Z
CLRF f	Borra f (Pone todos los bits a 0)	1	00 0001 1fff ffff	Z
CLRW —	Borra w	1	00 0001 0xxx xxxx	Z
COMF f,d	Complementa f (Invierte)	1	00 1001 dfff ffff	Z
DECF f,d	Decrementa f	1	00 0011 dfff ffff	Z
INCF f,d	Incrementa f	1	00 1010 dfff ffff	Z
IORWF f,d	OR entre W y f	1	00 0100 dfff ffff	Z
MOVF f,d	Mueve f	1	00 1000 dfff ffff	Z
MOVWF f	Mueve W y f	1	00 0000 1fff ffff	—
NOP —	No opera	1	00 0000 0xx0 0000	—
RLF f,d	Rota f a la izqda. a través del acarreo	1	00 1101 dfff ffff	C
RRF f,d	Rota f a la dcha. a través del acarreo	1	00 1100 dfff ffff	C
SUBWF f,d	Resta W a f	1	00 0010 dfff ffff	C,DC,Z
SWAPF f,d	Intercambia nibbles	1	00 1110 dfff ffff	—
XORWF f,d	XOR de W con f	1	00 0110 dfff ffff	Z
INSTRUCCIONES QUE MANEJAN BITS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
BCF f,b	Borra bit de f	1	01 00bb bfff ffff	—
BSF f,b	Pone a 1 el bit f	1	01 01bb bfff ffff	—
INSTRUCCIONES DE "BRINCO"				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
BTFSC f,d	Explora un bit de f y brinca si vale 0	1(2)	01 10bb bfff ffff	—
BTFSS f,d	Explora un bit de f y brinca si vale 1	1(2)	01 11bb bfff ffff	—
DECFSZ f,d	Decrementa f y si es 0, brinca	1(2)	00 1011 dfff ffff	—
INCFSZ f,d	Decrementa f y si es 1, brinca	1(2)	00 1111 dfff ffff	—
INSTRUCCIONES QUE MANEJAN OPERANDOS INMEDIATOS				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
ADDLW k	Suma inmediata con W	1	11 111x kkkk kkkk	C,DC,Z
ANDLW k	AND inmediato con W	1	11 1001 kkkk kkkk	Z
IORLW k	OR inmediato con W	1	11 1000 kkkk kkkk	Z
MOVLW k	Mueve a W un valor inmediato	1	11 00xx kkkk kkkk	—
SUBLW k	Resta W de un inmediato	1	11 110x kkkk kkkk	C,DC,Z
XORLW k	OR exclusiva con W	1	11 1010 kkkk kkkk	—
INSTRUCCIONES DE CONTROL Y ESPECIALES				
SINTAXIS	OPERACIÓN	CICLOS	FORMATO 14 bits	SEÑALIZADORES
CALL k	Llamada a subrutina	2	10 0kkk kkkk kkkk	TO#,PD#
CLRWDT	Borra o refresca el Perro Guardian	1	00 0000 0110 0100	—
GOTO k	Salto incondicional	2	10 1kkk kkkk kkkk	—
RETFIE	Retorno de Interrupción (GIE=1)	2	00 0000 0000 1001	—
RETLW k	Retorno subrutina y carga W=k	2	11 01xx kkkk kkkk	—
RETURN	Retorno de subrutina	2	00 0000 0000 1000	—
SLEEP	Pasa al modo de reposo	1	00 0000 0110 0011	TO#,PD#

Figura A.10. *Repertorio de instrucciones del PIC16X84.*

GUÍA DEL USUARIO DEL SIMUPIC'84



INTRODUCCIÓN

El **Simupic'84** es un programa cuyo objetivo es simular, en la medida de lo posible, el comportamiento de un microcontrolador PIC16X84, en sus dos posibles versiones, ya sea 16C84 o 16F84.

A pesar de que compite en prestaciones con muchos otros simuladores software del mercado, la principal característica de este programa es su facilidad de manejo. Se logra reunir en un mismo programa unas elevadas prestaciones y un diseño sencillo, que facilitan en todo momento la comprensión de las operaciones que se llevan a cabo en un PIC real. Esto se logra, entre otras cosas, gracias a recursos como la Ventana de Visión Externa, en la que se muestra en todo momento el estado de las líneas de E/S del PIC.

Otra característica que hace fácil el manejo de este simulador es que opera en un entorno de trabajo en el que están integradas todas las herramientas para el desarrollo de un programa. Se puede llamar al Editor, al Ensamblador y, una vez obtenido el ejecutable, simularlo.

Al tratarse de un simulador software, lógicamente, la velocidad de ejecución de un programa es varias veces inferior a la velocidad en un PIC real. **Simupic'84** dispone de algunas herramientas particulares, como los contadores de tiempo e instrucciones. No es posible el uso de periféricos reales, ya que la simulación es llevada a cabo totalmente por software.

INSTALACIÓN

La instalación de este programa en el disco duro es muy sencilla y basta seguir las instrucciones que se indican en el Apéndice E, el cual también describe el contenido del CD que acompaña a este libro. Le recomendamos que siga el tutorial multimedia sobre el SIMUPIC'84.

El editor y ensamblador utilizados son el EDIT.COM del DOS y el MPASM de Microchip. Si se desean utilizar otros, se han de especificar en el fichero de configuración **Simupic.cfg.**

LA VENTANA DE MENÚ PRINCIPAL

En esta ventana se presentan las siguientes opciones:

1. Asignar Nombre
2. Llamar al editor
3. Ensamblar
4. Cargar fichero HEX
5. Opciones de simulación
6. Ejecutar
7. Trazar
8. Visualizar registros
9. Alterar registros

ESC Salida al DOS

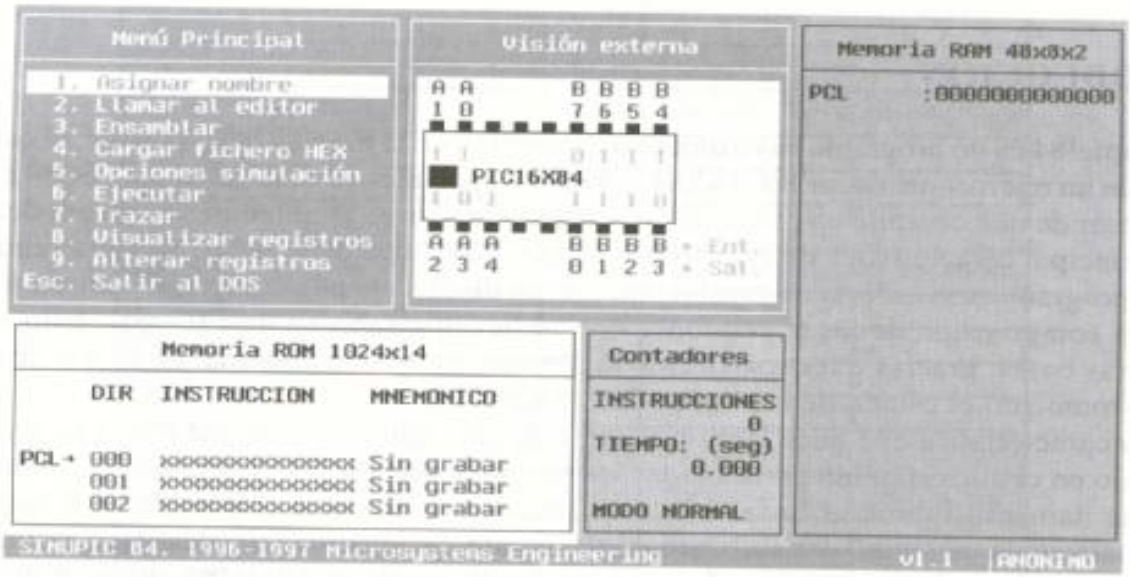


Figura B.1. *Pantalla principal del simulador SIMUPIC'84.*

Para seleccionar una de estas opciones se debe situar sobre la deseada la barra sombreada y a continuación pulsar <INTRO>. Otra forma de selección es mediante el ratón o simplemente pulsando el número de la opción elegida.

A continuación se explica la misión de cada una de las opciones:

1. Asignar Nombre

La función de esta opción es dar nombre al proyecto que se va a realizar. No es necesario indicar la extensión, y en caso de hacerlo, el programa no la tendrá en cuenta, tomando por defecto la extensión ASM para el fichero fuente y HEX para el ejecutable en formato Intel generado por el ensamblador.

2. Llamar al editor

Mediante esta opción se llama el editor de textos indicado en el fichero de configuración simupic.cfg en su línea:

«editor = prog.exe»

Especificando la ruta, se puede utilizar cualquier editor de textos del mercado para confeccionar el código fuente del programa, aunque se recomienda el EDIT.COM del DOS.

3. Ensamblar

Llama al ensamblador especificado en el fichero de configuración simupic.cfg en la línea:

«ensamblador = prog.exe»

Así se obtiene el fichero ejecutable.

4. Cargar fichero HEX

Carga en la memoria de programa el fichero *.HEX en formato Intel, generado por el ensamblador.

5. Opciones de simulación

Da paso al siguiente menú:

1. *Reset en el PIC:*
   Se genera en el PIC el Reset producido por MCLR (fuente externa de Reset). El PCL se pone a cero y si existía algún *breakpoint*, se mantiene.

2. *Poner a 0 contadores:*
   Pone a cero los contadores de tiempo e instrucciones de la ventana de contadores. Mediante el uso de estos contadores se puede llevar un completo control de las temporizaciones realizadas.

3. *Examinar programa:*
   Permite moverse a lo largo del programa para examinarlo, establecer un *breakpoint* o asignar a PCL una determinada dirección de la memoria de programa.

   En la pantalla de la página siguiente aparece una ventana de ayuda en la que se especifican las teclas para realizar cada una de estas acciones. Con ESC se descartan los cambios realizados y pulsando <INTRO> se almacenan.

4. *Parámetros de simulación:*
   Se especifica el modelo de PIC (16C84 o 16F84), la frecuencia y el tipo de reloj, el estado del Perro Guardián (activo o inactivo) y la duración del período de SLEEP.

   Otra característica reseñable es que debido a la lentitud de la simulación se puede alterar la duración del período de SLEEP para evitar que el Perro Guardián genere un Reset.

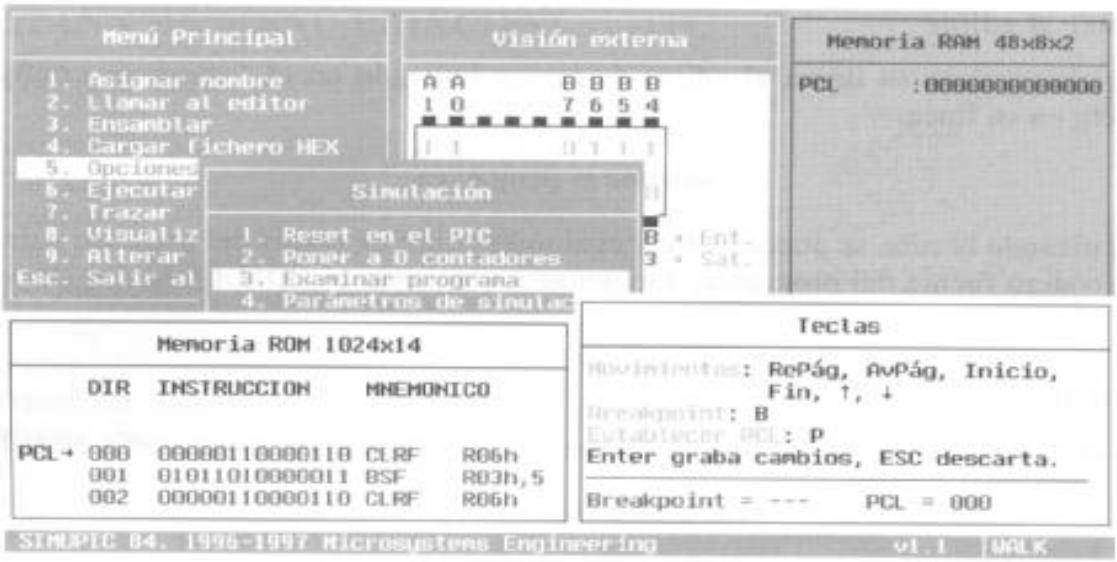


Figura B.2. *Ventana que permite examinar el programa de la memoria de instrucciones así como la asignación de puntos de ruptura (breakpoint).*

Admite indicar el número de instrucciones NOP que se ejecutarán antes de que se produzca el Reset o determinar la duración del SLEEP para que sea la misma que en un PIC real.

6. Ejecutar

Esta opción ejecuta el programa que contiene la memoria de instrucciones del simulador. El programa se ejecutará desde la posición actual del PCL hasta que encuentre un *breakpoint* o se pulse la tecla de final de ejecución (ESC).

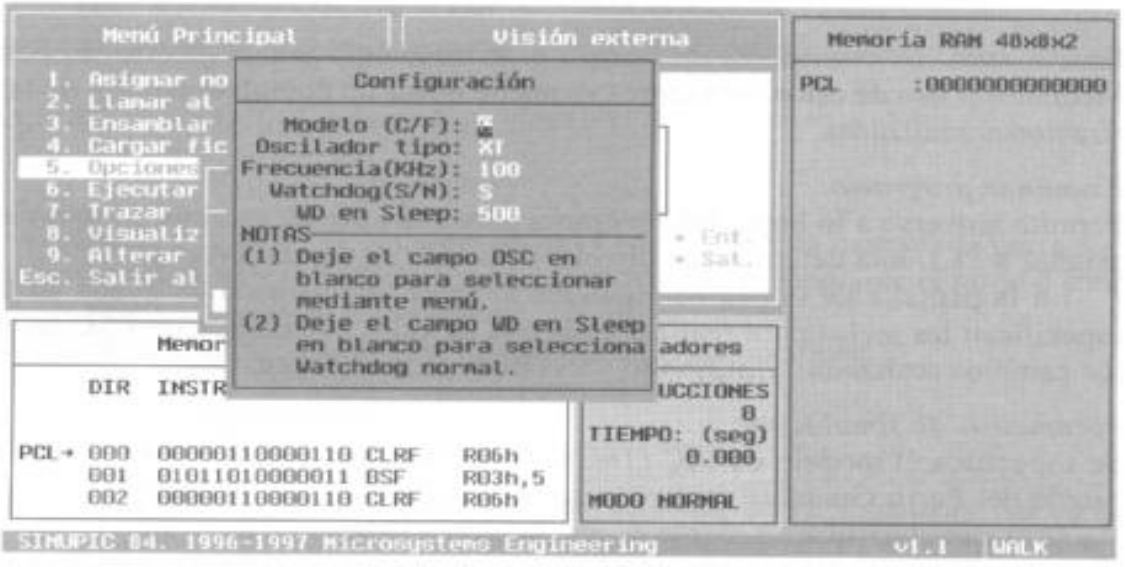


Figura B.3. *Ventana de configuración del simulador.*

Durante la ejecución del programa, tanto los contadores de tiempo como los registros visualizados son actualizados dinámicamente.

Una característica muy interesante que incorpora **Simupic'84** es la posibilidad de poder introducir estímulos a las líneas de E/S, durante la ejecución del programa:

F1-F5: Voltean el valor actual de las líneas RA0 a RA4.
Shift+F1 – Shift+F8: Voltean el valor actual de las líneas RB0 a RB7.

7. Trazar

Ejecuta la instrucción apuntada en ese momento por el PCL. Durante el trazado de un programa no es posible introducir estímulos a las líneas de E/S, pero es posible alterar los valores de los registros 5 (Puerta A) y 6 (PuertaB) mediante la opción «alterar registros», que se explicará más adelante.

Durante el trazado de un programa los *breakpoints* son ignorados.

8. Visualizar registros

Se visualizan en la pantalla los registros que se desee, ya sean especiales, de propósito general o de la EEPROM. El PCL es visualizado por defecto.

Se puede asignar al registro visualizado la etiqueta que se desee, facilitando de esta manera la comprensión del programa

9. Alterar registros

Se altera el valor binario de los registros que se especifique.

Para seleccionar un registro, ya sea para visualizarlo o alterarlo, se usan las siguientes opciones.

- PCL: Contador de Programa.
- TRISA TRISB: Configuraciones de las líneas de E/S.
- OPTION: Registro OPTION del PIC (dir 81h).
- EEPROM: Una dirección de la memoria EEPROM mediante dirección hexadecimal.

Una vez seleccionado el registro, se puede definir una etiqueta descriptiva de 8 caracteres para facilitar la identificación del registro.

Salir al DOS

Da la posibilidad de salir temporalmente al DOS o terminar el programa. En el primer caso tecleando EXIT se regresa a **Simupic'84** en el punto en que se abandonó

AYUDA

Mediante la tecla de función F1 se muestra en pantalla la ayuda para la función que se esté realizando en ese momento.

OTRAS VENTANAS

- *VENTANA DE VISIÓN EXTERNA*

  En esta ventana se muestra el diagrama de patitas gráficamente. Las líneas de E/S configuradas como entradas se muestran en color verde, mientras que las configuradas como salidas se muestran en color rojo. El valor de las líneas (0-1) se actualiza durante la ejecución y trazado de los programas.

- *VENTANA DE VISUALIZACIÓN DE REGISTROS*

  Es una ventana de color azul y está situada a la derecha de la pantalla. En ella se muestran los registros que han sido seleccionados previamente.

- *VENTANA DE MEMORIA ROM*

  En esta ventana se muestra el contenido de la memoria de programa del PIC. Cada línea muestra la dirección de memoria, el código de instrucción en binario y el nemónico correspondiente.

- *VENTANA DE CONTADORES*

  Se presentan los contadores de tiempo e instrucciones y el estado en que se encuentra el PIC, normal o SLEEP.

EJEMPLO PRÁCTICO

Se ofrece un programa muy sencillo cuyo único fin es ilustrar el manejo práctico del simulador.

```
; EJEMPLO.ASM Programa de ejemplo de uso del simulador
; SIMUPIC'84. El programa consiste en sacar por 4 leds
; conectados a las cuatro líneas de menos peso de PB el
; valor introducido por 4 interruptores conectados a las
; 4 líneas de menos peso de PA.

        LIST    P=16C84         ; Modelo de PIC a utilizar
        RADIX   HEX

ESTADO  EQU  0x03       ; Zona de etiquetas
PUERTAA EQU  0x05
PUERTAB EQU  0x06

        ORG     0
        bsf     ESTADO,5        ; Selección de banco 1
        movlw   0xff
        movwf   PUERTAA         ; Se configura PA como
                                ; entradas
```

```
        movlw   0x00
        movwf   PUERTAB         ; se configura PB como
                                ; salidas
        bcf     ESTADO,5        ; Selección de banco 0
bucle   movf    PUERTAA,0       ; Bucle infinito que mueve el
        movwf   PUERTAB         ; contenido de PA a PB.
        goto    bucle
        end
```

Mediante la opción «Asignar Nombre» se da un nombre al programa, en este caso EJEMPLO. No es necesario indicar la extensión. Se llama al editor de texto usando la opción 2 del menú principal. Una vez editado el programa fuente se obtiene el programa ejecutable, tras ensamblarlo mediante la opción 3. Realizada esta operación se carga en memoria de programa el fichero ejecutable con la opción «Cargar fichero HEX».

La numeración de las opciones del menú principal no es arbitraria sino que se corresponde con el orden de los pasos que se supone deben seguirse hasta obtener el programa ejecutable y poder simularlo.

Cargado en memoria el programa ejecutable y antes de pasar a simularlo, se debe acceder a la ventana de «Opciones de simulación», y dentro de ella a «Parámetros de simulación», para configurar las diferentes posibilidades de simulación, como el modelo de PIC a simular, la frecuencia de reloj, etc., que en este caso dejaremos con sus valores por defecto, es decir, PIC16C84 a una frecuencia de 100 KHz y con el Perro Guardián activado.

Finalmente, se simula el programa, ejecutándolo mediante la opción sexta del menú principal. Se puede ver el estado de las líneas de E/S en la ventana de visión externa y cambiar el estado de PA, que está configurada como entrada, mediante las teclas de función.

Para tener un control total sobre los pasos que está realizando el programa, conviene visualizar los registros implicados (W, Puerta A y Puerta B) y trazar el programa mediante la opción 7.

EJERCICIOS VARIADOS

WATCHDOG.ASM

Este programa muestra el funcionamiento del Perro Guardián o Watchdog. Para comprobar la temporización del Watchdog se va a utilizar una característica específica de **Simupic'84:** Al llegar a una posición de la memoria de programa que no ha sido grabada se muestra el mensaje «programa finalizado». De esta forma, el programa comienza en la dirección 10h, inicializa el WDT con una temporización de 36 ms y entra en un bucle infinito del que sólo sale al producirse Reset por WDT, condición que pone el PCL a cero, apuntando una posición no grabada. Esto ocurrirá a los 36 ms de iniciarse el programa. Mediante la ventana de contadores se puede comprobar esta temporización. Para poner a cero los contadores de tiempo e instrucciones se accede a la ventana de «Opciones de simulación» y se selecciona la opción «Poner a cero contadores».

```
; El objetivo de este programa es mostrar en la práctica el
; funcionamiento del Watchdog en el PIC 16C84.
; El programa entra en un bucle infinito del que sale tras
; cumplirse la temporización asignada al Watchdog, que en
; este caso serán 36 ms, es decir,2 veces la temporización
; nominal.
```

```
LIST    p=16C84
STATUS  EQU     03H
OPT     EQU     01H
; Programa principal:
        ORG     10
        bsf     STATUS,5        ;Banco 1
        movlw   b'00001001'
        movwf   OPT             ; Se asigna el Divisor al
                                ; watchdog con un rango
                                ; 1:2.
        clrwdt                  ; Se inicializa el WDT
bucle:  goto    bucle
        END
```

TIMER.ASM

A continuación se presenta un programa que muestra de forma práctica el uso del TMR0. El objetivo de este programa es lograr una temporización de 100 ms con una frecuencia de reloj de 100 KHz. Para ello, y tras realizar los cálculos pertinentes, inicializamos el TMR0 con un valor 100 usando el Divisor de Frecuencia con rango 16. Al igual que en el ejercicio anterior, utilizaremos los contadores de tiempo e instrucciones para comprobar la temporización.

```
; Programa que muestra el funcionamiento del timer:
; Realiza una temporización de 100 ms
list    P=16C84                 ; Tipo de PIC
intcom  equ     0BH
tmr_opt equ     01H
t0if    equ     02h
status  equ     03h
        org     0
        movlw   b'00000000'
        movwf   intcom          ; Desactiva interrupciones
        bsf     status,5        ; Banco 1
        movlw   b'00000011'     ; Divisor asignado al
                                ; TMR0 con un factor de 16
        movwf   tmr_opt
        bcf     status,5        ; Banco 0
        movlw   b'01100100'     ; Se inicializa TMR0 con
                                ; 100
        movwf   tmr_opt
bucle   btfsc   intcom,t0if     ; Si el bit t0if del intcon
                                ; es 1
```

```
        goto    fin             ; indica que el timer se ha
                                ; sobrepasado y entonces
        goto    bucle           ; se salta a fin. Si no
                                ; se vuelve al bucle
fin
        End
```

TMR0INT.ASM

El objetivo del programa que se presenta a continuación es similar al del programa anterior. En este caso se trata de realizar una temporización de 150 ms utilizando una frecuencia de reloj de 100 KHz, pero utilizando la interrupción de desbordamiento del TMR0. Tras realizar los cálculos pertinentes, obtenemos que se debe inicializar el timer con 22 y el Divisor con rango 16. Para el tratamiento de la interrupción únicamente se pone el W a cero y se espera a que el WDT reinicie el PIC.

```
; EJEMPLO de interrupciones en el PIC16C84 programado con
; SIMUPIC.
; Se trata de realizar una temporización de 150ms utilizando
; el reloj de 100KHz de un PIC

LIST    p=16C84                 ; Tipo de PIC que se usa

STATUS          EQU     03H
INTCON          EQU     0BH
TR0_OPT         EQU     01H     ; TIMER0 en banco 0, OPTION
                                ; en banco 1
        ORG     0               ; Vector de RESET
        goto    EMPIEZA
        ORG     4               ; Vector de interrupciones
        goto    TRATINT

EMPIEZA bsf     STATUS,5        ; Banco 1
        movlw   b'00000011'     ; Divisor rango 1:16 asignado
                                ; TMR0
        movwf   TR0_OPT
        movlw   b'10100000'     ; Se permite la interrupción
                                ; por TMR0
        movwf   INTCON          ; No es necesario volver a
                                ; cambiar de banco
                                ; porque INTCON está  en los
                                ; dos
        bcf     STATUS,5        ; Banco 0
        movlw   d'22'
        movwf   TR0_OPT         ; Se carga TMR0 con el
                                ; valor 22 decimal
BUCLE   goto    BUCLE           ; Bucle indefinido hasta
                                ; que se produzca la
                                ; interrupción
```

```
TRATINT   ; Aquí se incluirán los tratamientos de las
          ; interrupciones. De momento nos limitaremos a poner W
          ; a 0 (para que se vea que se ha producido la
          ; interrupción) y esperar a que el WatchDog reinicie
          ; el PIC.

          movlw   0
FIN       goto    FIN

          END
```

D_EEPROM.ASM

Se propone un ejemplo de uso de la EEPROM de datos que incorpora el PIC16C84. Esta memoria es muy útil a la hora de almacenar datos que no se desea perder tras interrumpir la alimentación. Cuenta con el inconveniente que su tiempo de escritura es muy alto y, además, variable, por lo que las escrituras sobre esta memoria se controlan mediante el uso de interrupciones.

El **Simupic'84** detiene el programa cuando encuentra alguna de estas tres condiciones de error relacionadas con escrituras sobre la EEPROM:

- *Lectura durante la escritura:* Se produce este error cuando se trata de leer o escribir en la memoria EEPROM antes de que haya terminado el proceso de escritura anterior.
- *Escritura o lectura en una posición inexistente del mapa de memoria de la EEPROM:* Cuando se trata de acceder a una posición de dirección mayor que 3Fh (los dos bit de más peso de EEADR no estaban a 0)
- *Orden de lectura y escritura simultáneas:* Esta condición de error es en realidad una variante del primer tipo de error, que consiste en poner WR y RD a 1 simultáneamente.

El programa ilustra la correcta utilización de la EEPROM de datos:

```
; Ejemplo comprobado con el SIMUPIC consistente en almacenar
; en las N primeras posiciones de la EEPROM el contenido de
; las direcciones 10h-(10+N-1)h de la memoria RAM
; El valor de N está en la dirección 0F del primer banco de
; memoria. Después de escribir la cadena se comprueba que la
; escritura es correcta y deja W a 0 si ha sido así, y a un
; valor diferente de 0 si no lo es.
; NOTA
; Funciona correctamente si N=0

LIST      p=16C84

DATA_CON1 EQU     08h
ADDR_CON2 EQU     09h
STATUS    EQU     03h
N         EQU     0Fh
INICIO    EQU     10h
```

```
INDF    EQU     00h
FSR     EQU     04h
INTCON  EQU     0Bh

        ORG     0
        goto    INI
        ORG     4
        retfie                  ; Interrupción sin
                                ; tratamiento
INI     bcf     STATUS,5        ; Banco 0
        movf    N,0             ; Se guarda en W el
                                ; contador
        movwf   ADDR_CON2
        andwf   ADDR_CON2,0     ; Se refrescan los flags
        btfsc   STATUS,2        ; Si FZ=1 se salta a FIN

        goto    FIN

        ; Inicialmente en ADDR está el número de
        ; bytes a escribir + 1 (de N+1 a 1)
        ; Se escriben en orden inverso

ESCRIBE decf    ADDR_CON2,1     ; DirEEPROM en Addr
        movf    ADDR_CON2,0     ; DirEEPROM en W
        addlw   INICIO          ; INICIO+DirEEPROM=
                                ; DirRAM (en W)
        movwf   FSR             ; Direccionamiento
                                ;indirecto
        movf    INDF,0          ; En W dato a escribir
        movwf   DATA_CON1       ; Dato a escribir en EE
                                ;DATA

        ; Secuencia de escritura

        bcf     INTCON,7        ; GIE=0
        bsf     STATUS,5        ; Banco 1
        movlw   h'55'
        movwf   ADDR_CON2
        movlw   h'AA'
        movwf   ADDR_CON2
        movlw   b'11000000'     ;
        movwf   INTCON          ; GIE=1 EEIE=1
        bsf     DATA_CON1,2     ; WREN=1
        bsf     DATA_CON1,1     ; Orden de escritura

WAIT    sleep   ; Se espera el fin de la escritura
                ; El WDT siempre tarda más en rebosar
                ; que la escritura en completarse
        bcf     STATUS,5        ; Banco 0
        clrw
        addwf   ADDR_CON2,0     ; ADDR_CON2=0 ?
        btfss   STATUS,2        ; Si FZ=1 se salta a
                                ; COMPROB
        goto    ESCRIBE
```

```
COMPROB   ; Inicio de la comprobación de la escritura

          movf     N,0                ; Se guarda en W el
                                      ; contador
          movwf    ADDR_CON2

          ; Inicialmente en ADDR está el número de
          ; bytes a leer + 1 (de N+1 a 1)
          ; Se escriben en orden inverso

LEE       decf     ADDR_CON2,1        ; DirEEPROM en Addr
          movf     ADDR_CON2,0        ; DirEEPROM en W
          addlw    INICIO             ; DirRAM en W
          movwf    FSR                ; Direccionamiento
                                      ; indirecto
          bsf      STATUS,5           ; Banco 1
          bsf      DATA_CON1,0        ; Orden de lectura
          bcf      STATUS,5           ; Banco 0
          ; comprobación
          movf     INDF,0             ; Dato de la RAM en W
          subwf    DATA_CON1,0        ; DATA-W -> W
          btfsc    STATUS,2           ; Si FZ=1 todo ha ido
                                      ; bien
          goto     FIN                ; Si no Error y W <> 0

          ; Siguiente
          clrw
          addwf    ADDR_CON2,0        ; ADDR_CON2=0 ?
          btfss    STATUS,2           ; Si FZ=1 se salta a FIN
          goto     LEE                ; Lee el siguiente

FIN       sleep
          goto     FIN

          END
```

BREVE DESCRIPCIÓN DEL «MICRO'PIC TRAINER»



INTRODUCCIÓN

La práctica totalidad de los ejercicios propuestos en este libro se pueden realizar con el **Micro'PIC Trainer (µPIC Trainer)** de **Microsystems Engineering**, que nos ha permitido incluir un breve resumen de uno de los capítulos del Manual del Usuario que acompaña a dicho sistema de desarrollo. Sólo se describe el esquema electrónico y el funcionamiento del sistema, que creemos puede ser de interés para los lectores como ejercicio de aplicación.

El sistema **µPIC Trainer** consiste en un equipo completo para la evaluación de aplicaciones basadas en los microcontroladores PIC de gama media de **MICROCHIP.** Dispone de una serie de periféricos básicos de E/S con los que se puede verificar el funcionamiento de una aplicación, así como la circuitería necesaria para la grabación de todos los modelos de microcontroladores PIC de gama media de 18 y 28 patitas. Un tutorial multimedia de montaje y manejo está contenido en el CD que acompaña al libro.

Se denomina «sistema de desarrollo» el equipo físico y el conjunto de programas que conforman una herramienta capaz de desarrollar todas las operaciones necesarias para diseñar un proyecto basado, en este caso, en un microcomputador programable, que al estar incorporado en un circuito integrado recibe el nombre de **microcontrolador**.

Las características del sistema son las siguientes:

1. *Alimentación desde un transformador de 12VAC o desde dos pilas de 9VDC. El circuito de estabilización está incluido en la placa. Con él se obtiene la tensión general de +5V (Vcc) y la de grabación de +13.8 V.*
2. *Soporta cualquier microcontrolador de gama media tanto de 18 como de 28 patitas. La frecuencia de trabajo por defecto es de 4MHz, que se puede modificar cambiando el cristal de cuarzo.*
3. *Generación automática de RESET (POR) y manual, mediante pulsador accionado por el usuario.*
4. *5 entradas digitales, implementadas mediante 5 conmutadores.*
5. *4 entradas de tensión analógica, variables mediante potenciómetros.*
6. *8 salidas digitales conectadas a 8 indicadores luminosos tipo LED y/o a display de 7 segmentos. Mediante jumpers se selecciona uno u otro.*
7. *Conexión de un módulo de visualización tipo LCD de 2 X 16 (dos líneas de 16 caracteres cada una) y control de contraste mediante potenciómetro.*

8. *Tanto los periféricos de entrada como los de salida se pueden desactivar o no mediante sus respectivos jumpers. Se desconectan aquellos que en una determinada aplicación no se empleen, con objeto de reducir la carga eléctrica en las líneas de E/S del PIC.*
9. *Conector de expansión PIC-BUS donde todas las señales de E/S del microcontrolador, bajo prueba, están disponibles para ser empleadas por otros periféricos distintos a los que dispone µPIC Trainer.*
10. *La señal RA4/T0CKI se puede emplear como entrada o salida digital o entrada de pulsos de reloj para el timer 0, desde el exterior.*
11. *La señal RB0/INT se puede emplear como entrada o salida digital o entrada de interrupción externa.*
12. *Circuito grabador «en circuito». Permite, con el software de control necesario, grabar el PIC que en ese momento esté insertado en cualquiera de los zócalos. Es especialmente útil para el modelo 16C84, que al disponer de memoria EEPROM de programa, se puede grabar y regrabar tantas veces como sea necesario sin necesidad de retirarlo del zócalo. Lo mismo sucede con el 16F84, con memoria Flash.*
13. *Led indicador de presencia de la tensión Vpp de grabación.*
14. *Conector DB25 para conexión con el canal paralelo de cualquier ordenador personal.*
15. *Software en castellano para realizar todas las operaciones propias de grabación: Lectura, grabación, verificación, borrado, edición, etc.*
16. *Todos los componentes que forman µPIC Trainer van sobre una placa de circuito impreso de tipo profesional. El conjunto se suministra completamente montado o en forma de Kit para que sea el propio usuario quien lo ensamble.*

En la Figura C.1 se presenta el esquema eléctrico de conexiones del sistema de evaluación **µPIC Trainer**. En los siguientes apartados se irán explicando las diferentes secciones que lo componen. En el CD existe un tutorial muy didáctico que muestra el montaje, paso a paso, de este sistema de desarrollo.

LA FUENTE DE ALIMENTACIÓN

Se encarga de obtener las tensiones estabilizadas de trabajo: alimentación general de +5V (Vcc) y la tensión de grabación de +13.8V.

La tensión de entrada se aplica mediante 2 pilas de 9V o bien desde un transformador de 12VAC conectado a J1. En este caso la tensión alterna se rectifica mediante el puente de diodos D1 y se filtra mediante los condensadores C1 y C2, para obtener una continua sin estabilizar.

Sea como fuere, la tensión continua así obtenida se estabiliza a +5V mediante el regulador U6, –uA7805, y a +13.8V mediante el regulador uA7812 junto con los tres diodos D2-D4, 1N4007.

EL MICROCONTROLADOR

El sistema µPIC Trainer está diseñado para trabajar con cualquier modelo de microcontrolador PIC de la gama media encapsulado con 18 o 28 patitas. Ésta es la razón de la existencia de dos zócalos para soportar ambos tipos.

Según el esquema de la Figura C.1, las diferencias básicas entre los modelos de 18 y 28 patitas consiste en que los primeros disponen de una puerta A de 5 líneas (RA0-RA4) y una puerta B de 8 (RB0-RB7), mientras que los segundos disponen de una puerta A de 6 líneas (RA0-RA5), una puerta B de 8 (RB0-RB7) y una puerta C de 8 líneas (RC0-RC7).

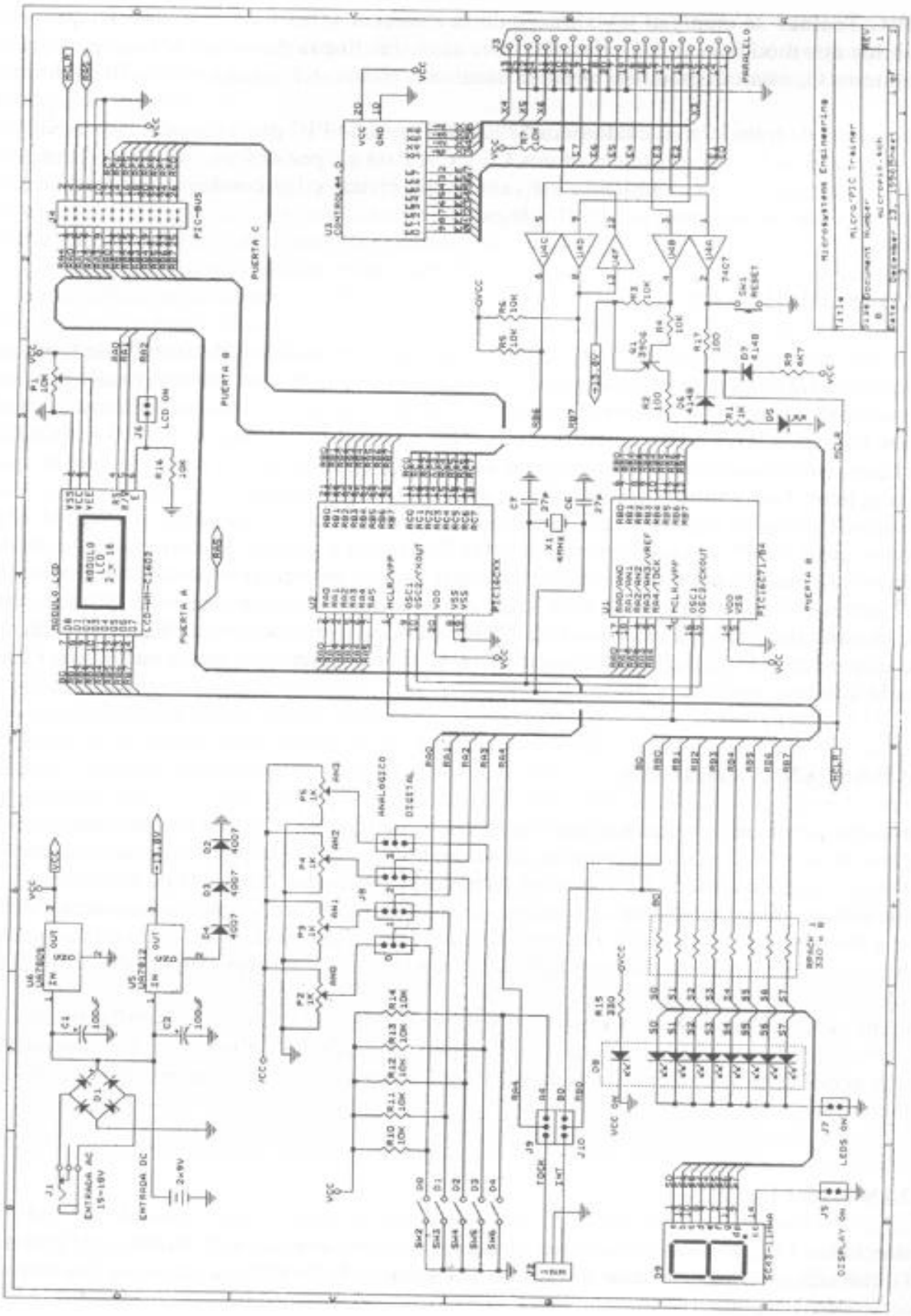


Figura C.1. *Esquema eléctrico de µPIC Trainer.*

En **µPIC Trainer** se emplean las 5 líneas de la Puerta A y las 8 de la Puerta B, que son comunes a los dos modelos de PIC. En cualquier caso, las líneas de estas puertas, junto con las de la puerta C, están disponibles para el usuario a través del conector PIC-BUS de expansión.

La frecuencia de trabajo para cualquiera de los dos tipos de PIC queda establecida mediante el cristal de cuarzo X1 y los condensadores C6 y C7. Ésta es, por defecto, de 4MHz, aunque el usuario puede modificarla cambiando los valores del cristal y los condensadores según las características técnicas del modelo de PIC elegido.

ENTRADAS DIGITALES

Están formadas por 5 interruptores (SW2-SW6) conectados a las líneas RA0-RA4 de la puerta A, capaces de introducir niveles lógicos «1» y «0» por esas líneas. Dichas líneas pueden programarse para actuar de diferentes formas. Así, RA0-RA3 pueden actuar, además, como entradas analógicas y RA4 como entrada exterior de pulsos de reloj para el TMR0 (T0CKI).

Estas líneas tienen asociados unos jumpers con los que se selecciona el tipo de entrada que se les va a aplicar. Los jumpers J8(0)-J8(3) seleccionan individualmente si por RA0-RA3 se van a introducir entradas digitales con los interruptores SW2-SW5, entradas analógicas con los potenciómetros P2-P5 o, simplemente, si estas líneas van a quedar desconectadas tanto de los interruptores como de los potenciómetros y poder así usarse con otros periféricos distintos.

Con el jumper J9 se selecciona si RA4 actúa como entrada digital procedente del interruptor SW6, entrada de señal de reloj para el TMR0 (T0CKI), que se aplica desde el exterior mediante el conector J2(1), o simplemente, si esta línea va a quedar desconectada y libre, poder usarla con otro periférico distinto.

ENTRADAS ANALÓGICAS

Están formadas por 4 resistencias variables o potenciómetros (P2-P5), que se alimentan con la tensión general de +5V. Según se mueva el eje de cualquiera de estos potenciómetros, se obtiene una tensión variable entre 0 y 5V por los terminales centrales (cursores) de los mismos.

La tensión variable presente en cualquiera de los 4 cursores de los potenciómetros P2-P5 va a parar a los jumpers J8(0)-J8(3), respectivamente. Si cualquiera de ellos está en la posición «analógico», la línea correspondiente (RA0-RA3) recibirá dicha tensión para su posterior procesamiento.

Conviene recordar que aunque a nivel de hardware una o más líneas se configuren como entradas analógicas o digitales mediante los jumpers J8(0)-J8(3), el software de control del PIC debe ir acorde con las mismas, programando adecuadamente las características de dichas líneas de entrada.

SALIDAS DIGITALES

Están conectadas a la puerta B y consisten, por una parte, en una barra de diodos leds luminosos (D8) que representan el estado lógico de las señales RB0-RB7, y por otra, un display alfanumérico de 7 segmentos.

Estos periféricos están conectados entre sí en paralelo a la Puerta B. Para evitar un exceso de consumo en dicha puerta, tanto los leds como el display disponen de los jumpers J7 y J5, respectivamente, con los que se les puede desconectar si la aplicación en curso no hace uso de cualquiera de ellos.

El pack de resistencias RPACK 1 contiene 8 resistencias de 330Ω que actúan como resistencias de absorción o limitación.

La barra de diodos luminosos D8 contiene 10 diodos tipo led. Los 8 primeros, S0-S7, representan, en caso de estar habilitados (J7), el estado de las señales RB0-RB7; el noveno led no se emplea y el décimo sirve para indicar la presencia de tensión de +5V (Vcc ON).

La línea RB0 puede actuar como entrada de interrupción externa INT. Dicha señal de entrada se aplica por el conector J2(3). Con el jumper J10 la señal RB0 se conecta a los leds y al display, a la entrada de interrupción INT o, simplemente, se deja sin conexión.

Los periféricos conectados a la Puerta B pueden deshabilitarse mediante sus respectivos jumpers. De esta forma no tienen efecto alguno sobre esa puerta, por lo que las señales RB0-RB7 disponibles en el conector de expansión PIC-BUS pueden ser empleadas por otro tipo de periféricos en otras configuraciones.

EL MÓDULO LCD

Se trata de un módulo de visualización alfanumérico de cristal líquido, capaz de presentar 2 líneas con 16 caracteres cada una.

Aplicándole los códigos necesarios se puede provocar diferentes efectos de visualización como parpadeo, scroll, activación de un cursor, etc. Incluso es posible la generación de nuevos caracteres definidos por el usuario.

Las 8 líneas de datos D0-D7 están conectadas con las 8 líneas de la Puerta B (RB0-RB7). Esta puerta, a veces, actúa como salida del PIC y entrada hacia el módulo. Por ella se le aplican los diferentes códigos de control para realizar diferentes efectos de visualización, así como los códigos ASCII de los caracteres a visualizar. En otras ocasiones la Puerta B debe actuar como entrada hacia el PIC, ya que a su través el módulo LCD devuelve códigos indicando su estado interno, el contenido del buffer de memoria interna, etc.

El módulo está conectado a las líneas RA0, RA1 y RA2 de la Puerta A del PIC. Estas líneas actúan como salida y se emplean para enviar las siguientes señales de control al módulo:

RS = 0	Selección del registro de instrucciones
RS = 1	Selección del registro de datos
R/W = 0	Ciclo de escritura sobre el módulo LCD
R/W = 1	Ciclo de lectura del módulo LCD
E = 0	Módulo LCD desactivado
E = 1	Módulo LCD activado

Hay que destacar que la señal E está conectada con RA2 a través del jumper J6. Si dicho jumper está cerrado, RA2 controlará la habilitación o no del módulo LCD. Por el contrario, si estuviera abierto, la señal E queda conectada a tierra (nivel «0») a través de R16. De esta forma el módulo está siempre desconectado y las líneas tanto de la Puerta A (RA0-RA2) como

de la B (RB0-RB7) pueden ser empleadas por otros periféricos, ya sean los que están disponibles en **µPIC Trainer** o cualquier otro conectado al conector de expansión PIC-BUS.

El módulo LCD es un periférico de visualización muy utilizado en aplicaciones reales. Permite una cómoda, atractiva y versátil forma de visualización alfanumérica e incluso gráfica.

EL CIRCUITO GRABADOR

Se trata de una de las secciones más interesantes del sistema **µPIC Trainer**. Su importancia radica en la posibilidad de realizar la grabación del PIC «*en circuito*». Esto significa que el PIC va a poder ser manipulado (leído, grabado, borrado, etc.) directamente alojado en el zócalo de trabajo.

Esta ventaja es aún más relevante si se emplea el modelo de PIC16C84. Este modelo dispone de memoria EEPROM como memoria de programa. Esta memoria, a pesar de no ser volátil, puede grabarse y borrarse casi un número indefinido de veces. Así pues, este PIC, insertado en su correspondiente zócalo, va a poder ser grabado con múltiples programas y aplicaciones, al tiempo que directamente se van ejecutando y evaluando sobre **µPIC Trainer**. Lo mismo pasa con el PIC16F84.

El circuito grabador permite grabar todos los modelos de 18 y 28 patitas de la gama media de PIC. Mediante el conector J3 se conecta al canal paralelo de cualquier ordenador personal tipo PC/XT/AT. El programa de control PICME-TR, totalmente en castellano e incluido en el kit **µPIC Trainer**, se encarga de gestionar todas las operaciones, tales como leer el contenido del PIC (si no está protegido), grabar, verificar, borrar (sólo el 16X84), establecer la palabra de configuración, etc. En el CD-ROM que acompaña al libro se incluye el programa PICME-TR y un tutorial multimedia de manejo.

La conexión entre el PC y **µPIC Trainer** se realiza a través del canal paralelo. El circuito integrado U3-CONTROL#4.0 se encarga de verificar el hardware de **µPIC Trainer** y la comunicación entre éste y el PC.

La lectura/grabación de los PIC de la gama media se realiza en serie. Los bits de datos se leen o escriben secuencialmente, uno tras otro, a través de la señal RB7 y son aplicados al canal paralelo mediante los amplificadores U4D y U4F, para su posterior tratamiento por el software de control.

La señal RB6 actúa como entrada de reloj para sincronizar la transferencia de los bits de datos durante el proceso de lectura/grabación del PIC. Esta señal de reloj es generada por el software de control y se aplica por el canal paralelo a través del amplificador U4C.

El software de control también genera una señal que, mediante U4B, se aplica al transistor de conmutación Q1. Este transistor al activarse envía los 13.8V generados desde la fuente de alimentación a la patita MCLR/VPP del PIC que se está leyendo o grabando. Al mismo tiempo el diodo led D5 se ilumina indicando actividad en el circuito grabador de **µPIC Trainer**.

Finalmente, a través de U4A, el software de control genera la señal MCLR, que resetea y reinicia al microcontrolador. Normalmente esto ocurre al finalizar cualquier operación de lectura y/o grabación, con lo que el PIC se lanza inmediatamente a ejecutar el programa que contenga en su memoria. Cabe indicar que, en cualquier caso y en cualquier momento, el usuario puede provocar el Reset manualmente accionando el pulsador RESET SW1.

Dado que las señales que se emplean para la lectura / grabación de un PIC son RB6 y RB7 y éstas a su vez están conectadas a los leds, display y módulo LCD, ***es importante*** desconectar estos periféricos mediante los jumpers J7, J5 y J6, respectivamente, para que no supongan carga alguna a dichas señales y alteren la información leída o grabada.

EL CONECTOR DE EXPANSIÓN

Se trata de un conector de 26 contactos en el que están disponibles todas las señales de las diferentes puertas de E/S, además de las de alimentación y MCLR. Esto permite conectar futuras expansiones desarrolladas por **Microsystems Engineering** o por el propio usuario, manejando nuevos periféricos. Esta posibilidad tiene especial interés en la conexión de la placa de prototipos PIC18-ME, donde pueden alojarse los periféricos de la aplicación, mientras que se depura el software desde el sistema de desarrollo.

Para que no haya conflicto entre los periféricos conectados al PIC-BUS y los ya existentes en **µPIC Trainer**, es necesario desconectar, mediante sus correspondientes jumpers, aquellos que vayan a ser reemplazados.

Una completa información técnica del sistema de desarrollo µPIC Trainer, con numerosos ejercicios prácticos, puede hallarse en el Manual de Usuario de esta excepcional herramienta diseñada por **Microsystems Engineering** (Gral. Concha, 39 - 48012 Bilbao).

En la Figura C.2. se muestra una fotografía del Entrenador de Laboratorio **Micro'PIC Lab**, compuesto por el **µPIC Trainer** conectado a la tarjeta de prototipos **PIC18-ME**, que se alimentan con un transformador especial y van montados sobre un bastidor de metacrilato.

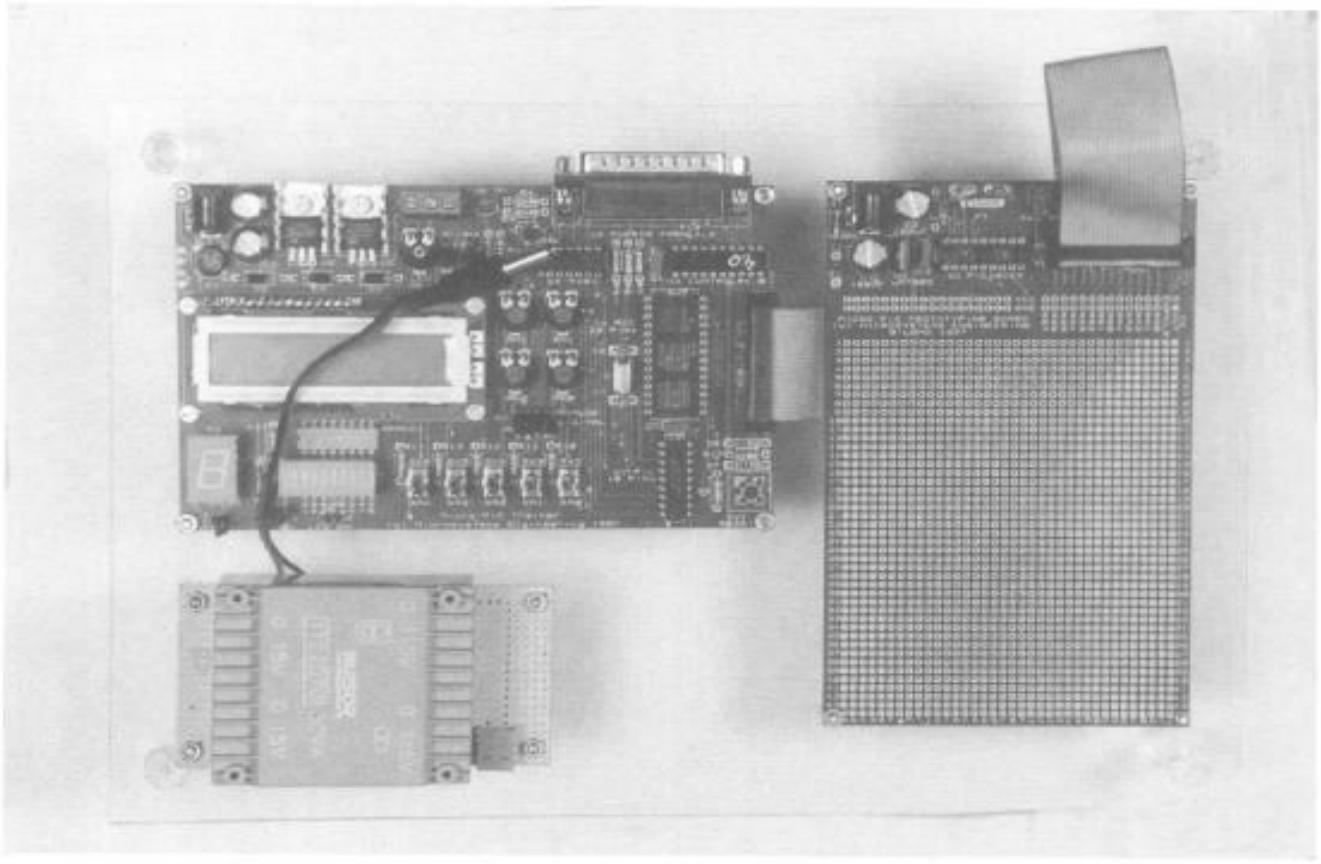


Figura C.2. *Fotografía del Entrenador de Laboratorio Micro'PIC Lab que dispone del µPIC Trainer, la tarjeta de prototipos PIC18-ME y un transformador especial, montados sobre un bastidor de metacrilato.*

LA FAMILIA PIC AL COMPLETO



(CORTESÍA DE **MICROCHIP TECHNOLOGY INC.**)

CURRENT PRODUCTS

PICmicro® 8-BIT MICROCONTROLLER FAMILY PRODUCTS

CURRENT PRODUCTS

Product	Program Memory Bytes	Program Memory OTP/FLASH Words	Program Memory ROM Words	E²PROM Memory Bytes	RAM Bytes	Analog 8-Bit ADC Channel	Analog Comparators	Analog Other	Digital I/O Ports	Digital Serial I/O	Digital PWM 10-Bits	Digital Timers	System Max. Speed MHz	System ICSP™	Other Features	Package
PIC12CXXX — 400ns Instruction Execution, 33/35 Instructions, 8-Pin Package, 4MHz Internal Oscillator, 4/5 Oscillator Selections																
PIC12C508	768	512x12	—	—	25	—	—	—	6	—	—	1-8 bit, 1-WDT	4	Yes	25mA source/sink per I/O	8P, 8SM, 8JW
PIC12C508A	768	512x12	—	—	25	—	—	—	6	—	—	1-8 bit, 1-WDT	4	Yes	25mA source/sink per I/O	8P, 8SM, 8JW, 8SN
PIC12C509	1536	1024x12	—	—	41	—	—	—	6	—	—	1-8 bit, 1-WDT	4	Yes	25mA source/sink per I/O	8P, 8SM, 8JW
PIC12C509A PIC12CR509A	1536 1536	1024x12 —	— 1024x12	— —	41 41	— —	— —	— —	6 6	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	4 4	Yes —	25mA source/sink per I/O 25mA source/sink per I/O	8P, 8SM, 8JW, 8SN 8P, 8SM, 8SN
PIC12CE518	768	512x12	—	16	25	—	—	—	6	—	—	1-8 bit, 1-WDT	4	Yes	25mA source/sink per I/O	8P, 8SM, 8JW, 8SN
PIC12CE519	1536	1024x12	—	16	41	—	—	—	6	—	—	1-8 bit, 1-WDT	4	Yes	25mA source/sink per I/O	8P, 8SM, 8JW, 8SN
PIC12C671	1792	1024x14	—	—	128	4	—	—	6	—	—	1-8 bit, 1-WDT	10	Yes	25mA source/sink per I/O	8P, 8SM, 8JW
PIC12C672	3584	2048x14	—	—	128	4	—	—	6	—	—	1-8 bit, 1-WDT	10	Yes	25mA source/sink per I/O	8P, 8SM, 8JW
PIC12CE673	1792	1024x14	—	16	128	4	—	—	6	—	—	1-8 bit, 1-WDT	10	Yes	25mA source/sink per I/O	8P, 8JW
PIC12CE674	3584	2048x14	—	16	128	4	—	—	6	—	—	1-8 bit, 1-WDT	10	Yes	25mA source/sink per I/O	8P, 8JW
PIC16C5X — 200ns Instruction Execution, 33 Instructions, 4/5 Oscillator Selections																
PIC16C52	576	384x12	—	—	25	—	—	—	12	—	—	1-8 bit	4	—	10mA source/sink per I/O	18P, 18SO
PIC16C54 PIC16CR54A	768 768	512x12 —	— 512x12	— —	25 25	— —	— —	— —	12 12	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	— —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS 18P, 18SO, 20SS
PIC16C54A	768	512x12	—	—	25	—	—	—	12	—	—	1-8 bit, 1-WDT	20	—	20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS
PIC16C54C PIC16CR54C	768 768	512x12 —	— 512x12	— —	25 25	— —	— —	— —	12 12	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	— —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS 18P, 18SO, 20SS
PIC16C55	768	512x12	—	—	24	—	—	—	20	—	—	1-8 bit, 1-WDT	20	—	20mA source and 25mA sink per I/O	28P, 28JW, 28SP, 28SO, 28SS
PIC16C55A	768	512x12	—	—	24	—	—	—	20	—	—	1-8 bit, 1-WDT	20	—	20mA source and 25mA sink per I/O	28P, 28JW, 28SP, 28SO, 28SS
PIC16C56	1536	1024x12	—	—	25	—	—	—	12	—	—	1-8 bit, 1-WDT	20	—	20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS
PIC16C56A PIC16CR56A	1536 1536	1024x12 —	— 1024x12	— —	25 25	— —	— —	— —	12 12	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	— —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS 18P, 18SO, 20SS
PIC16C57	3072	2048x12	—	—	72	—	—	—	20	—	—	1-8 bit, 1-WDT	20	—	20mA source and 25mA sink per I/O	28P, 28JW, 28SP, 28SO, 28SS
PIC16CR57B	3072	—	2048x12	—	72	—	—	—	20	—	—	1-8 bit, 1-WDT	20	—	20mA source and 25mA sink per I/O	28P, 28SP, 28SO, 28SS
PIC16C57C PIC16CR57C	3072 3072	2048x12 —	— 2048x12	— —	72 72	— —	— —	— —	20 20	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	— —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	28P, 28JW, 28SP, 28SO, 28SS 28P, 28SP, 28SO, 28SS
PIC16C58A PIC16CR58A	3072 3072	2048x12 —	— 2048x12	— —	73 73	— —	— —	— —	12 12	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	— —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS 18P, 18SO, 20SS
PIC16C58B PIC16CR58B	3072 3072	2048x12 —	— 2048x12	— —	73 73	— —	— —	— —	12 12	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	— —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18JW, 18SO, 20SS 18P, 18SO, 20SS
PIC16C505	1536	1024x12	—	—	72	—	—	—	12	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 4MHz internal oscillator	14P, 14JW, 14SL
PIC16HV540*	768	512x12	—	—	25	—	—	BOD	12	—	—	1-8 bit, 1-WDT	20	—	8 high voltage (15V) I/Os, 4 deep stack, 5 I/O with wake up on Change, Extended Voo (3.5 to 15 V)	18P, 18JW, 18SO, 20SS
PIC16CXXX — 4-12 Interrupts, 200ns Instruction Execution, 35 Instructions, 4/5 Oscillator Selections, Upwardly Compatible with PIC16C5X/PIC12CXXX																
PIC14C000	7168	4096x14	—	—	192	8 SLAC	2	Bandgap VREF, Prog. Reference Generator	20	I²C/SMB	—	1-16 bit, 1-8 bit, 1-WDT	20	Yes	25mA source/sink, internal oscillator, temperature sensor	28SP, 28SO, 28SS, 28JW
PIC16C554	896	512x14	—	—	80	—	—	—	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C558	3584	2048x14	—	—	128	—	—	—	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C61	1792	1024x14	—	—	36	—	—	—	13	—	—	1-8 bit, 1-WDT	20	Yes	20mA source and 25mA sink per I/O	18P, 18SO, 18JW
PIC16C62A PIC16CR62	3584 3584	2048x14 —	— 2048x14	— —	128 128	— —	— —	BOD BOD	22 22	I²C/SPI I²C/SPI	1 1	1-16 bit, 2-8 bit, 1-WDT 1-16 bit, 2-8 bit, 1-WDT	20 20	Yes —	25mA source/sink per I/O, CCP 25mA source/sink per I/O, CCP	28SP, 28SO, 28SS, 28JW 28SP, 28SO, 28SS
PIC16C620	3584	2048x14	—	—	128	—	—	BOD	22	I²C/SPI	1	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, CCP	28SP, 28SO, 28SS, 28JW
PIC16C63	7168	4096x14	—	—	192	—	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP	28SP, 28SO, 28JW
PIC16C63A PIC16CR63	7168 7168	4096x14 —	— 4096x14	— —	192 192	— —	— —	BOD BOD	22 22	USART/I²C/SPI USART/I²C/SPI	2 2	1-16 bit, 2-8 bit, 1-WDT 1-16 bit, 2-8 bit, 1-WDT	20 20	Yes —	25mA source/sink per I/O, 2 CCP 25mA source/sink per I/O, 2 CCP	28SP, 28SO, 28SS, 28JW 28SP, 28SO, 28SS

PICmicro® 8-BIT MICROCONTROLLER FAMILY PRODUCTS

Product	Program Memory			E²PROM Memory Bytes	RAM Bytes	Analog			Digital				System		Other Features	Packages
	Bytes	OTP/FLASH Words	ROM Words			8-Bit ADC Channel	Comparators	Other	I/O Ports	Serial I/O	PWM 10-Bits	Timers	Max. Speed MHz	ICSP™		
PIC16CXXX — 4-12 Interrupts, 200ns Instruction Execution, 35 Instructions, 4/5 Oscillator Selections, Upwardly Compatible with PIC16C5X/PIC12CXXX (continued)																
PIC16C64A	3584	2048x14	—	—	128	—	—	BOD	33	I²C/SPI	1	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, CCP	40P, 40JW, 44L, 44PQ, 44PT
PIC16C64	3584	—	2048x14	—	128	—	—	BOD	33	I²C/SPI	1	1-16 bit, 2-8 bit, 1-WDT	20	—	25mA source/sink per I/O, Parallel Slave Port, CCP	40P, 44L, 44PQ, 44PT
PIC16C65A	7168	4096x14	—	—	192	—	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, 2 CCP	40P, 40JW, 44L, 44PQ, 44PT
PIC16C65B	7168	4096x14	—	—	192	—	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, 2 CCP	40P, 40JW, 44L, 44PQ, 44PT
PIC16C65	7168	—	4096x14	—	192	—	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	—	25mA source/sink per I/O, Parallel Slave Port, 2 CCP	40P, 44L, 44PQ, 44PT
PIC16C66	14336	8192x14	—	—	368	—	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP	28SP, 28SO, 28JW
PIC16C67	14336	8192x14	—	—	368	—	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP, Parallel Slave Port	40P, 40JW, 44L, 44PQ, 44PT
PIC16C620	896	512x14	—	—	80	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C620A PIC16CR620A	896 896	512x14 —	— 512x14	— —	96 96	— —	2 2	BOD, Prog. VREF BOD, Prog. VREF	13 13	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	20 20	Yes —	25mA source/sink per I/O 25mA source/sink per I/O	18P, 18SO, 20SS, 18JW 18P, 18SO, 20SS
PIC16C621	1792	1024x14	—	—	80	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C621A	1792	1024x14	—	—	96	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C622	3584	2048x14	—	—	128	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C622A	3584	2048x14	—	—	128	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16CE623	896	512x14	—	128	96	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16CE624	1792	1024x14	—	128	96	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16CE625	3584	2048x14	—	128	128	—	2	BOD, Prog. VREF	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16F627*	1792 (FLASH)	1024 x 14 (FLASH)	—	128	224	—	2	BOD, Prog. VREF	16	USART/SCI	1	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 4MHz internal clock oscillator, CCP	18P, 18SO, 20SS
PIC16F628*	3584 (FLASH)	2048 x 14 (FLASH)	—	128	224	—	2	BOD, Prog. VREF	16	USART/SCI	1	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 4MHz internal clock oscillator, CCP	18P, 18SO, 20SS
PIC16C642	7168	4096x14	—	—	176	—	2	BOD, Prog. VREF	22	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	28SP, 28SO, 28JW
PIC16C662	7168	4096x14	—	—	176	—	2	BOD, Prog. VREF	33	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	40P, 40JW, 44L, 44PQ, 44PT
PIC16C710	896	512x14	—	—	36	4	—	BOD	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C71	1792	1024x14	—	—	36	4	—	—	13	—	—	1-8 bit, 1-WDT	20	Yes	20mA source/sink per I/O	18P, 18SO, 18JW
PIC16C711	1792	1024x14	—	—	68	4	—	BOD	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C715	3584	2048x14	—	—	128	4	—	BOD	13	—	—	1-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O	18P, 18SO, 20SS, 18JW
PIC16C72	3584	2048x14	—	—	128	5	—	BOD	22	I²C/SPI	1	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, CCP	28SP, 28SO, 28JW, 28SS
PIC16C72A PIC16CR72	3584 3584	2048x14 —	— 2048x14	— —	128 128	5 5	— —	BOD BOD	22 22	I²C/SPI I²C/SPI	1 1	1-16 bit, 2-8 bit, 1-WDT 1-16 bit, 2-8 bit, 1-WDT	20 20	Yes —	25mA source/sink per I/O, CCP 25mA source/sink per I/O, CCP	28SP, 28SO, 28SS, 28JW 28SP, 28SO, 28SS
PIC16C73A	7168	4096x14	—	—	192	5	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP	28SP, 28SO, 28JW
PIC16C73B	7168	4096x14	—	—	192	5	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP	28SP, 28SO, 28JW, 28SS
PIC16C74A	7168	4096x14	—	—	192	8	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, 2 CCP	40P, 40JW, 44L, 44PQ, 44PT
PIC16C74B	7168	4096x14	—	—	192	8	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, 2 CCP	40P, 40JW, 44L, 44PQ, 44PT
PIC16C76	14336	8192x14	—	—	368	5	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP	28SP, 28SO, 28JW
PIC16C77	14336	8192x14	—	—	368	8	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, 2 CCP	40P, 40JW, 44L, 44PQ, 44PT
PIC16C773*	7168	4096x14	—	—	256	6 (12-bit)	—	Prog. BOD & VREF & LVD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, 2 CCP, Master I²C/SPI, 9-bit address	28SP, 28SO, 28SS, 28JW
PIC16C774*	7168	4096x14	—	—	256	10 (12-bit)	—	Prog. BOD & VREF & LVD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, Parallel Slave Port, 2 CCP, Master I²C/SPI, 9-bit address	40P, 40JW, 44L, 44PT
PIC16F83 PIC16CR83	896 (FLASH) 896	512x14 (FLASH) —	— 512x14	64 64	36 36	— —	— —	— —	13 13	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	10 10	Yes —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18SO 18P, 18SO
PIC16F84 PIC16CR84	1792 (FLASH) 1792	1024x14 (FLASH) —	— 1024x14	64 64	68 68	— —	— —	— —	13 13	— —	— —	1-8 bit, 1-WDT 1-8 bit, 1-WDT	10 10	Yes —	20mA source and 25mA sink per I/O 20mA source and 25mA sink per I/O	18P, 18SO 18P, 18SO
PIC16F84A*	1792 (FLASH)	1024x14 (FLASH)	—	64	68	—	—	—	13	—	—	1-8 bit, 1-WDT	10	Yes	25mA source/sink per I/O	18P, 18SO, 20SS

PICmicro® 8-BIT MICROCONTROLLER FAMILY PRODUCTS

Product	Program Memory			E²PROM Memory Bytes	RAM Bytes	Analog			Digital				System		Other Features	Packages
	Bytes	OTP/FLASH Words	ROM Words			8-Bit ADC Channel	Comparators	Other	I/O Ports	Serial I/O	PWM 10-Bits	Timers	Max. Speed MHz	ICSP™		
PIC16CXXX — 4-12 Interrupts, 200ns Instruction Execution, 35 Instructions, 4/5 Oscillator Selections, Upwardly Compatible with PIC16C5X/PIC12CXXX (continued)																
PIC16F873*	7168 (FLASH)	4096 x 14 (FLASH)	—	128	192	5 (10-bit)	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, CCP, Master I²C/SPI, 9-bit address	28SP, 28SO
PIC16F874*	7168 (FLASH)	4096 x 14 (FLASH)	—	128	192	8 (10-bit)	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, CCP, Parallel Slave Port, Master I²C/SPI, 9-bit address	40P, 44L, 44PQ, 44PT
PIC16F876*	14336 (FLASH)	8192 x 14 (FLASH)	—	256	368	5 (10-bit)	—	BOD	22	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, CCP, Master I²C/SPI, 9-bit address	28SP, 28SO
PIC16F877*	14336 (FLASH)	8192 x 14 (FLASH)	—	256	368	8 (10-bit)	—	BOD	33	USART/I²C/SPI	2	1-16 bit, 2-8 bit, 1-WDT	20	Yes	25mA source/sink per I/O, CCP, Parallel Slave Port, Master I²C/SPI, 9-bit address	40P, 44L, 44PQ, 44PT
PIC16C923	7168	4096x14	—	—	176	—	—	—	52	I²C/SPI	1	1-16 bit, 2-8 bit, 1-WDT	8	Yes	25mA source/sink per I/O, CCP, LCD module, static, 1/2, 1/3, 1/4 multiplex	64SP, 68L, 64PT
PIC16C924	7168	4096x14	—	—	176	5	—	—	52	I²C/SPI	1	1-16 bit, 2-8 bit, 1-WDT	8	Yes	25mA source/sink per I/O, CCP, LCD module, static, 1/2, 1/3, 1/4 multiplex	64SP, 68CL, 68L, 64PT
PIC17CXXX — 120ns Instruction Execution Including Multiply, 58 Instructions, 4 Oscillator Selections, Externally expandable to 64Kx16 Program Memory, Upwardly Compatible with PIC16CXX/PIC16C5X/PIC12CXXX																
PIC17C42A	4096	2048x16	—	—	232	—	—	—	33	USART	2	2-16 bit, 2-8 bit, 1-WDT	33	—	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 2 Capture, 1 cycle 8x8 multiply	40P, 40JW, 44L, 44PQ, 44PT
PIC17CR42	4096	—	2048x16	—	232	—	—	—	33	USART	2	2-16 bit, 2-8 bit, 1-WDT	33	—	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 2 Capture, 1 cycle 8x8 multiply	40P, 44L, 44PQ, 44PT
PIC17C43	8192	4096x16	—	—	454	—	—	—	33	USART	2	2-16 bit, 2-8 bit, 1-WDT	33	—	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 2 Capture, 1 cycle 8x8 multiply	40P, 40JW, 44L, 44PQ, 44PT
PIC17CR43	8192	—	4096x16	—	454	—	—	—	33	USART	2	2-16 bit, 2-8 bit, 1-WDT	33	—	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 2 Capture, 1 cycle 8x8 multiply	40P, 44L, 44PQ, 44PT
PIC17C44	16384	8192x16	—	—	454	—	—	—	33	USART	2	2-16 bit, 2-8 bit, 1-WDT	33	—	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 2 Capture, 1 cycle 8x8 multiply	40P, 40JW, 44L, 44PQ, 44PT
PIC17C752	16384	8192x16	—	—	678	12 (10-bit)	—	BOD	50	USART (2)/I²C/SPI	3	2-16 bit, 2-8 bit, 1-WDT	33	Yes	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 4 Capture, 1 cycle 8x8 multiply, Master I²C/SPI, 9-bit address	68L, 64PT
PIC17C756	32768	16384x16	—	—	902	12 (10-bit)	—	BOD	50	USART (2)/I²C/SPI	3	2-16 bit, 2-8 bit, 1-WDT	33	Yes	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 4 Capture, 1 cycle 8x8 multiply, Master I²C/SPI, 9-bit address	68CL, 68L, 64PT
PIC17C756A*	32768	16384x16	—	—	902	12 (10-bit)	—	BOD	50	USART (2)/I²C/SPI	3	2-16 bit, 2-8 bit, 1-WDT	33	Yes	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 4 Capture, 1 cycle 8x8 multiply, Master I²C/SPI, 9-bit address	68CL, 68L, 64PT
PIC17C762	16384	8192x16	—	—	678	16 (10-bit)	—	BOD	66	USART (2)/I²C/SPI	3	2-16 bit, 2-8 bit, 1-WDT	33	Yes	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 4 Capture, 1 cycle 8x8 multiply, Master I²C/SPI, 9-bit address	84L, 80PT
PIC17C766	32768	16384x16	—	—	902	16 (10-bit)	—	BOD	66	USART (2)/I²C/SPI	3	2-16 bit, 2-8 bit, 1-WDT	33	Yes	20mA source and 35mA sink per I/O, 2 I/O with 60mA sink, 4 Capture, 1 cycle 8x8 multiply, Master I²C/SPI, 9-bit address	84CL, 84L, 80PT

Abbreviation:

ADC = Analog-to-Digital Converter
BOD = Brown out Detection
CAP = Capture
CCP = Capture/Compare/PWM
DAC = Digital-to-Analog Converter
E2 = EEPROM (Reprogrammable)
I²C = Inter-Integrated Circuit Bus
LVD = Low Voltage Detection
PWM = Pulse Width Modulator
SLAC = Slope A/D Converter, up to 16 bits
SMB = System Management Bus
SPI = Serial Peripheral Interface
USART = Universal Synchronous/Asynchronous Receiver/Transmitter
Vref = Voltage Reference
WDT = Watchdog Timer

*Contact Microchip Technology for availability date.

CONTENIDO DEL CD-ROM



Para el correcto funcionamiento del CD-ROM que acompaña a la obra es necesario un PC con los siguientes requisitos:

- Tarjeta de sonido.
- Tarjeta gráfica.
- Altavoces.

La primera vez que se ejecute la aplicación es necesaria la instalación previa del visor **Quicktime30.exe**, que se encuentra contenido en el directorio raíz del CD-ROM (para las referencias en este apéndice se supone Unidad **D:**). Una vez hecho esto se ejecutará el programa **Tutorial.exe**, contenido en **D:\Tutorial Multimedia**.

Desde la pantalla principal de este programa se podrán hacer las siguientes operaciones:

- **Instalar SIMUPIC:** crea el directorio **C:\Tutorial\Simupic** y copia en él el simulador SIMUPIC, el ensamblador MPASM y otros programas complementarios. Crea además el directorio **C:\Tutorial\Proyectos**, donde copia programas de aplicación, dispuestos en los siguientes directorios:

  - **C:Ejemplos:** incluye los ejemplos de uso del simulador SIMUPIC'84, de los que se habla en el anexo acerca de este simulador.
  - **C:Libro:** incluye todos los ejercicios de la sección «Programar PIC es fácil» de cada uno de los capítulos, así como los proyectos propuestos en el Capítulo 8.
  - **C:Básicos:** incluye los cinco Proyectos Básicos resueltos en el Apéndice F.
  - **C:Típicos:** incluye los tres Proyectos Típicos resueltos en el Apéndice F.

- **Instalar PICME-TR:** crea el directorio **C:\Tutorial\Picme-Tr** y copia en él el programa de grabación PICME-TR. Dicho programa necesitará la tarjeta de desarrollo µPIC' TRAINER, conectada al PC mediante un cable paralelo y provista de un microcontrolador PIC para ser operativo.

- **Tutorial:** este tutorial, mediante el uso del sonido, imagen, vídeo y animaciones, instruirá al usuario en los siguientes aspectos:

- **Simupic**: se mostrará el funcionamiento de cada una de las opciones de este simulador.
- **MicroPic Trainer:**
  - **Montaje:** paso a paso, se mostrará cómo identificar cada componente y montarlo en el sito y forma adecuados de la tarjeta de desarrollo.
  - **PICME-TR:** se enseñarán los pasos necesarios para la correcta grabación de un microcontrolador PIC.
  - **Simulación:** se incluyen dos vídeos, uno mostrando la ejecución de un programa y otro sobre las clases de montaje de la tarjeta MicroPic'Trainer.

PROYECTOS CON EL PIC16F84



INTRODUCCIÓN

Ante las numerosas peticiones, consultas y recomendaciones que nos han enviado muchos lectores particulares, profesores y profesionales durante la primera edición de esta obra, los autores nos hemos visto obligados a añadir este apéndice, el cual describe, de la manera más sencilla que hemos sido capaces, una colección de proyectos que pueden realizarse con un material fácil de encontrar, sencillo de manejar y muy económico. Pretendemos que esta aportación anime a todos los indecisos y les permita coger el «vuelo». Una vez estén en el aire, ya se arreglarán ellos solos, por la cuenta que les tiene.

Para desarrollar las implementaciones físicas de los proyectos hemos usado como soporte principal el PIC16F84 y la placa de prototipos **PIC18-ME** de la empresa ***Microsystems Engineering***, a la que agradecemos su inestimable colaboración. Dicha empresa comercializa un kit de Proyectos Básicos y otro de Proyectos Típicos que contienen todos los elementos empleados en los diseños, lo que permitirá su fácil disponibilidad a los lectores que residan en zonas que carezcan de establecimientos electrónicos especializados o no estén muy versados en el tema. Con la finalidad de conseguir que los materiales de estos kits puedan utilizarse en todas las aplicaciones propuestas, y en otras que acometa el usuario, se recomienda una placa *Protoboard* para el montaje sin soldadura. Se recomienda además el uso del sistema de desarrollo **MicroPic'Trainer**, ya descrito en un apéndice anterior, para la puesta a punto del software.

La primera parte de este apéndice está destinada a describir, con todo lujo de detalles, cinco proyectos sencillos que emplean los componentes del Kit de Proyectos Básicos, resueltos íntegramente y acompañado cada uno de ellos de una pequeña variación sin resolver. En la segunda parte se exponen otros tres proyectos, también resueltos y con una variación, que utilizan unos periféricos más versátiles y típicos en aplicaciones con microcontroladores, como son el teclado y la pantalla LCD, y que forman parte del Kit de Proyectos Típicos. Ya en la tercera parte, se propone una colección de proyectos para que los resuelva el lector empleando los mismos materiales y con distintos grados de complejidad. A continuación se expone el índice de este anexo:

PRIMERA PARTE: PROYECTOS BÁSICOS RESUELTOS Y COMENTADOS

1.º *Lista de materiales necesarios.*
2.º *Control de un dispositivo desde dos puntos.*
3.º *Semáforo programable.*
4.º *Dado electrónico.*
5.º *Temporizador programable.*
6.º *Alarma.*

SEGUNDA PARTE: PROYECTOS TÍPICOS RESUELTOS Y COMENTADOS

1.º *Lista de materiales necesarios.*
2.º *Teclado musical.*
3.º *Mensaje de bienvenida sobre LCD.*
4.º *Escritura de mensajes de teclado sobre LCD.*

TERCERA PARTE: COLECCIÓN DE PROYECTOS PROPUESTOS

1.º *Ejercicio combinacional de manejo de E/S digitales.*
2.º *Otro ejercicio combinacional.*
3.º *Control de un torno.*
4.º *Automatismo de control de un taladro.*
5.º *Sistema antirrebotes.*
6.º *Manejo del TMR0.*
7.º *Contador ascendente/descendente.*
8.º *Control de la puerta de un garaje.*
9.º *Generación de un número aleatorio.*
10.º *Presentación de estados lógicos.*
11.º *Control de acceso a un local.*
12.º *Contador de miles de piezas.*
13.º *Sistema de control de acceso mediante clave.*

PRIMERA PARTE: PROYECTOS BÁSICOS RESUELTOS Y COMENTADOS

LISTA DE MATERIALES NECESARIOS

Materiales comunes a todos los Proyectos Básicos

1.º *Tarjeta de prototipos PIC18-ME montada, de Microsystems Engineering, y un transformador de alimentación 220/15VAC-0,8 A.*
2.º *Protoboard para el montaje sin soldadura.*
3.º *Microcontrolador PIC16F84.*
4.º *Sistema de desarrollo Micro'PIC Trainer para grabar y borrar el PIC, y depurar los programas.*
5.º *Cables de un solo hilo de cobre preestañado, de 0,8 mm de diámetro.*

Materiales específicos para el Proyecto Básico 1

1.º Diodo LED rojo 5 mm.
2.º Conjunto de cuatro microrruptores DIP-SWITCH (sólo se usan dos microrruptores).
3.º Resistencia de 330Ω y ¼ W.
4.º Dos resistencias de 10KΩ y ¼ W.

Se recomienda leer el funcionamiento y conexionado de los periféricos que utiliza este proyecto (diodo LED y microrruptores) en la sección del Capítulo 1 titulada «Diseñar aplicaciones con PIC es fácil».

Esquema eléctrico

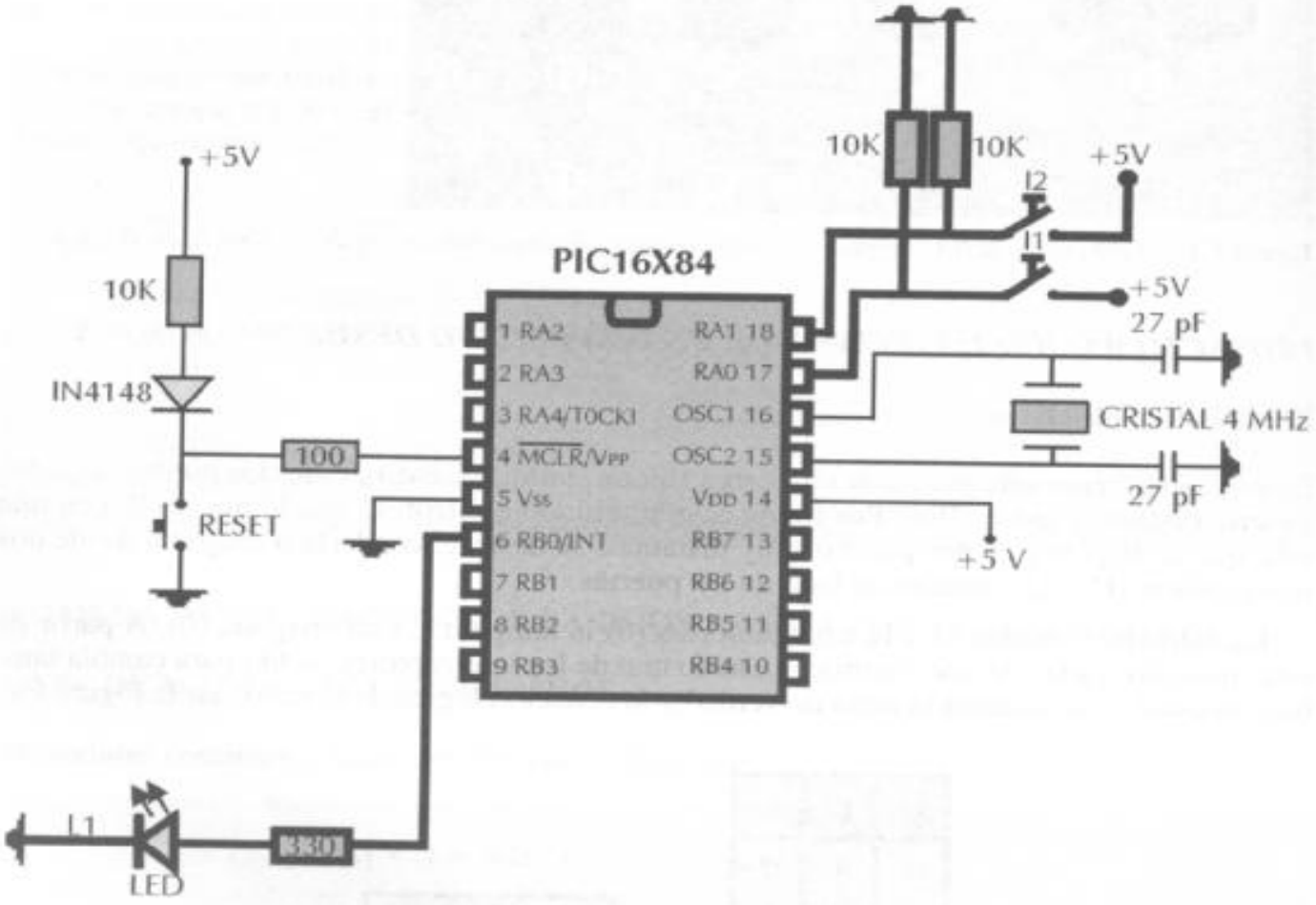


Figura F.3. *Esquema eléctrico de la práctica. Los componentes dibujados con línea fina se hallan premontados en la tarjeta de prototipos PIC18-ME.*

Foto del montaje

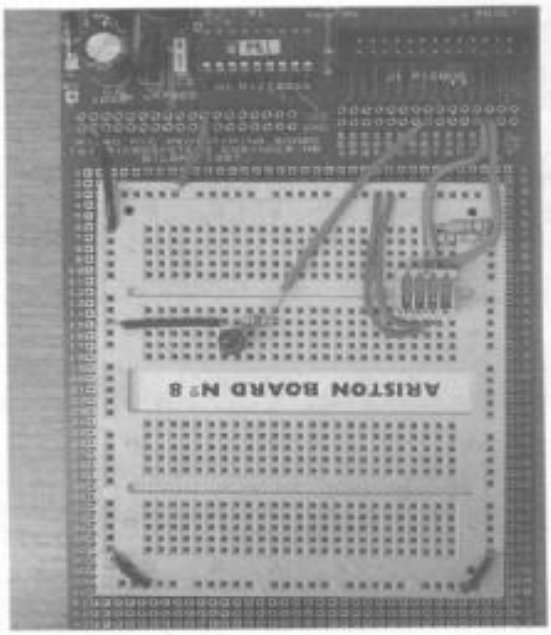


Figura F.4. *Fotografía del montaje sobre la placa PIC18-ME.*

Organigrama

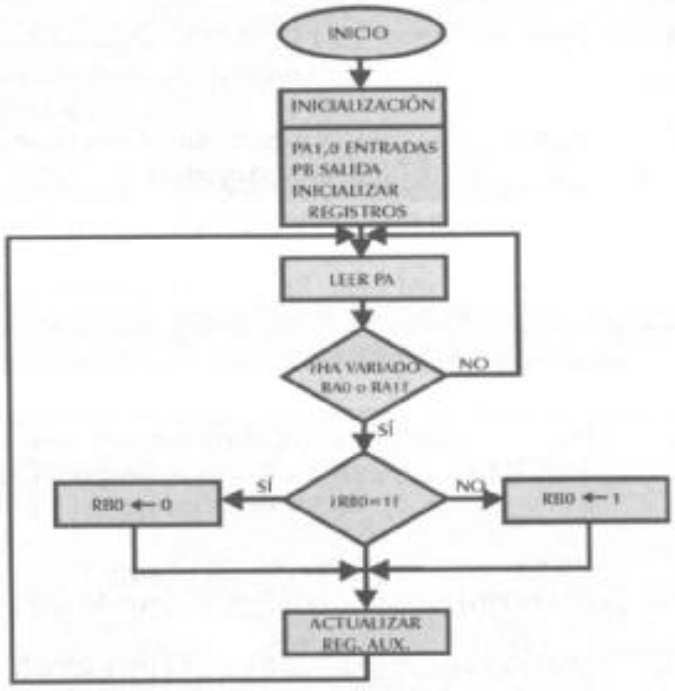


Figura F.5. *Organigrama del proyecto.*

El programa comentado

PROYECTO BÁSICO 1
«CONTROL DE UN DISPOSITIVO DESDE DOS PUNTOS»

Se desea controlar una lámpara, en este caso un diodo LED, desde dos interruptores, de forma que cuando los dos interruptores estén abiertos el LED esté apagado. A partir de esta situación, cada vez que cambie el estado de un interruptor, el estado del LED báscula (si está apagado, se enciende, y viceversa).

DIRECTIVAS DEL PROGRAMA

```
LIST    P=16F84     ; Se emplea el PIC16F84
RADIX   HEX         ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
W       EQU   0x00  ; Registro de destino W : d=0
F       EQU   0x01  ; Registro de destino F : d=1
ESTADO  EQU   0x03  ; El registro ESTADO ocupa la dirección 3 de los
                    ; dos bancos
PUERTAA EQU   0x05  ; LaPUERTAA ocupa la dirección 5 del banco 0
                    ; y su registro de configuración la dirección 5 del
                    ; banco1
PUERTAB EQU   0x06  ; La PUERTAB ocupa la dirección 6 del banco 0
                    ; y su registro de configuración la dirección 6 del
                    ; banco1
AUX     EQU   0x0C  ; Registro auxiliar en el que se carga el estado de
                    ; los interruptores
```

PROGRAMA PRINCIPAL

```
        ORG     0           ; El programa comienza en la dirección 0 y
        goto    INICIO      ; salta a la dirección 5 para sobrepasar el
        ORG     5           ; vector de interrupción

INICIO  bsf     ESTADO,5    ; Selección del banco 1
        movlw   b'00000011' ; Se configura parte de la PUERTAA como
        movwf   PUERTAA     ; entrada
        movlw   b'00000000' ; Se configura la PUERTAB como
        movwf   PUERTAB     ; salida
```

```
        bcf     ESTADO,5    ; Selección del banco 0
        clrf    PUERTAA     ; Se inicializan las dos puertas a
        clrf    PUERTAB     ; ceros
        clrf    AUX         ; Se inicializa el registro auxiliar a ceros

BUCLE   movf    PUERTAA,W   ; Se exploran los interruptores de la PUERTAA
        xorwf   AUX,W       ; Se comprueba si el estado de los
        btfsc   ESTADO,2    ; interruptores ha cambiado, para lo cual se
                            ; detecta si se ha activado el bit 2 del registro
                            ; estado (Z)
        goto    BUCLE       ; Si no ha cambiado, se sigue explorando hasta
                            ; que uno cambie

LUZ     btfss   PUERTAB,0   ; Se comprueba si el LED está encendido para
                            ; saber si hay que apagarlo o encenderlo
        goto    PUESTA1     ; Se llama a la rutina Puesta1, queenciende el
                            ; LED
        goto    PUESTA0     ; Se llama a la rutina Puesta0, que apaga el LED

PUESTA1 bsf     PUERTAB,0   ; Se pone a 1 el bit 0 de la PUERTAB (LED)
        goto    ACTUAL
PUESTA0 bcf     PUERTAB,0   ; Se pone a 0 el bit 0 de la PUERTAB

ACTUAL  movf    PUERTAA,W   ; Se actualiza el estado de AUX, para comprobar
                            ; luego si los interruptores han cambiado de valor
        movwf   AUX
        goto    BUCLE       ; Salto a bucle para empezar de nuevo

        END
```

Propuesta de un nuevo proyecto

Para comenzar a diseñar con microcontroladores, y una vez que ha sido capaz de poner en marcha el proyecto que hemos descrito, le proponemos un ejercicio para que lo implemente usted.

Se dispone de tres interruptores, I1, I2 e I3, con los que se desea controlar un diodo LED «L», de tal forma que debe encenderse cuando dos, «y sólo dos», de los interruptores estén cerrados. En todas las demás situaciones el diodo LED debe estar apagado.

PROYECTO BÁSICO 2: SEMÁFORO PROGRAMABLE

Enunciado y objetivos

Se trata de simular el funcionamiento de las tres luces de un semáforo (verde, ámbar y roja) mediante tres diodos LED (L1, L2 y L3) de entre los diez de que dispone una barra DIL. El control del encendido y apagado de los LED se realiza mediante los microrruptores I1 e I2 de los cuatro de que dispone un conjunto DIP-SWITCH.

I2	I1	L1(V) Rb1	L2(A) Rb3	L3(R) Rb5
0	0	5	2	5
0	1	5	2	10
1	0	8	2	8
1	1	10	2	5

Figura F.6. *Tabla de verdad en la que se indican los segundos de encendido de los LED.*

Según el estado de los microrruptores, se desarrolla una secuencia concreta de tiempo de encendido de los tres LED que simulan las luces del semáforo, de acuerdo con la tabla de la Figura F.6.

La finalidad primordial de este ejercicio es la de manejar el temporizador TMR0 para que produzca retardos de diversa duración.

Materiales específicos para el Proyecto Básico 2

1.º *Barra DIL de 10 diodos LED (sólo se usan tres LED).*
2.º *Conjunto de ocho resistencias RPACK SIL de 330Ω y ¼ W (sólo se usan tres).*
3.º *Dos resistencias de 10KΩ y ¼ W.*
4.º *Conjunto de cuatro microrruptores DIP-SWITCH (sólo se usan cos microrruptores).*

Esquema eléctrico

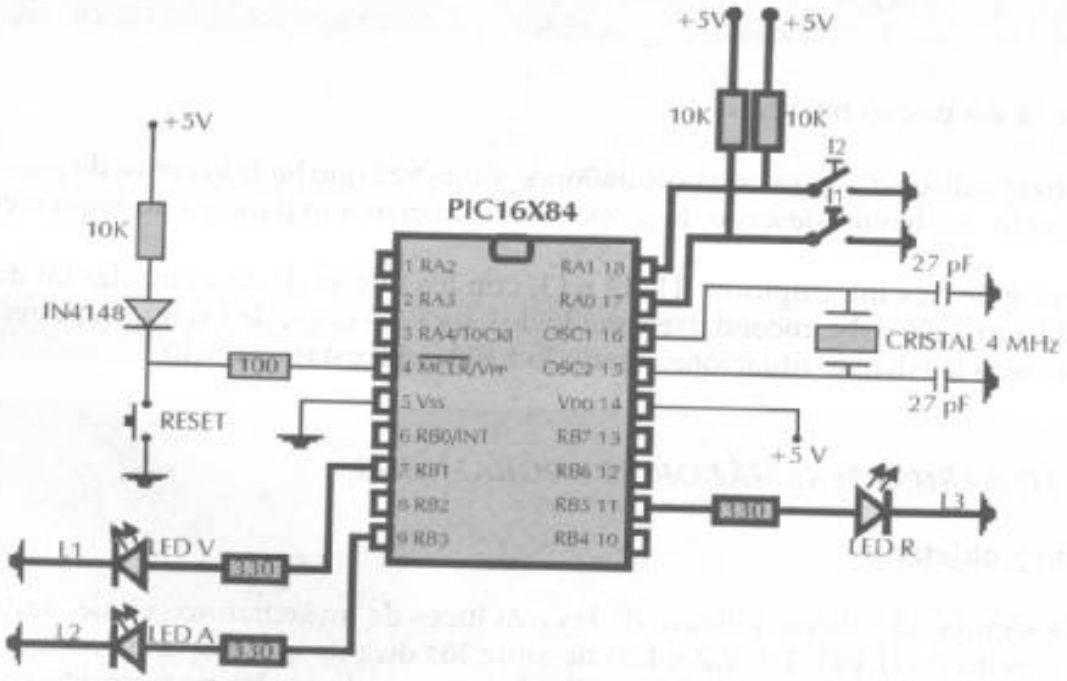


Figura F.7. *Esquema eléctrico.*

Como se puede apreciar en la Figura F.7, los microrruptores I1 e I2 se conectan a las líneas de entrada RA0 y RA1 de la PUERTAA a través de sendas resistencias de 10KΩ. En este caso son activos por nivel bajo. Los diodos LED L1, L2 y L3 se conectan a las líneas de salida RB1, RB3 y RB5 de la PUERTAB a través de resistencias de 330Ω. Los demás elementos del esquema están presolados en la tarjeta de prototipos PIC18-ME.

Organigrama

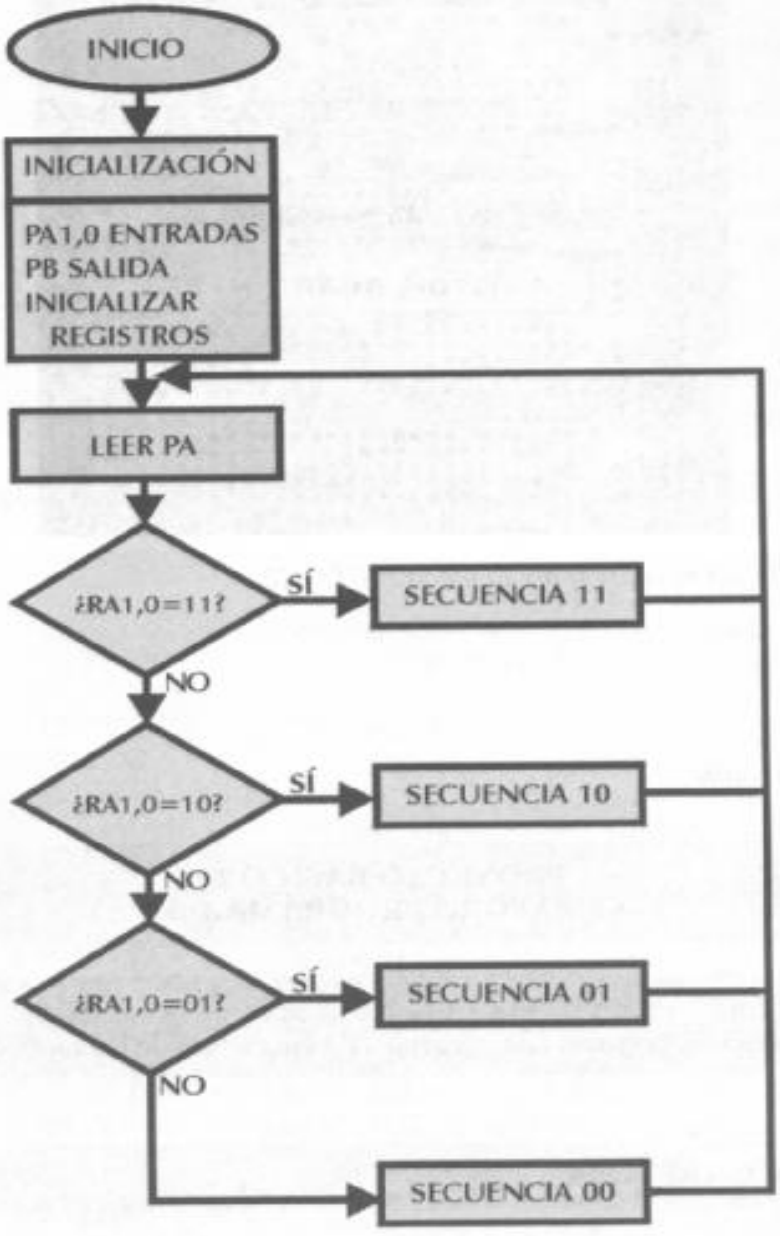


Figura F.8. *Organigrama del funcionamiento del semáforo.*

Foto del montaje

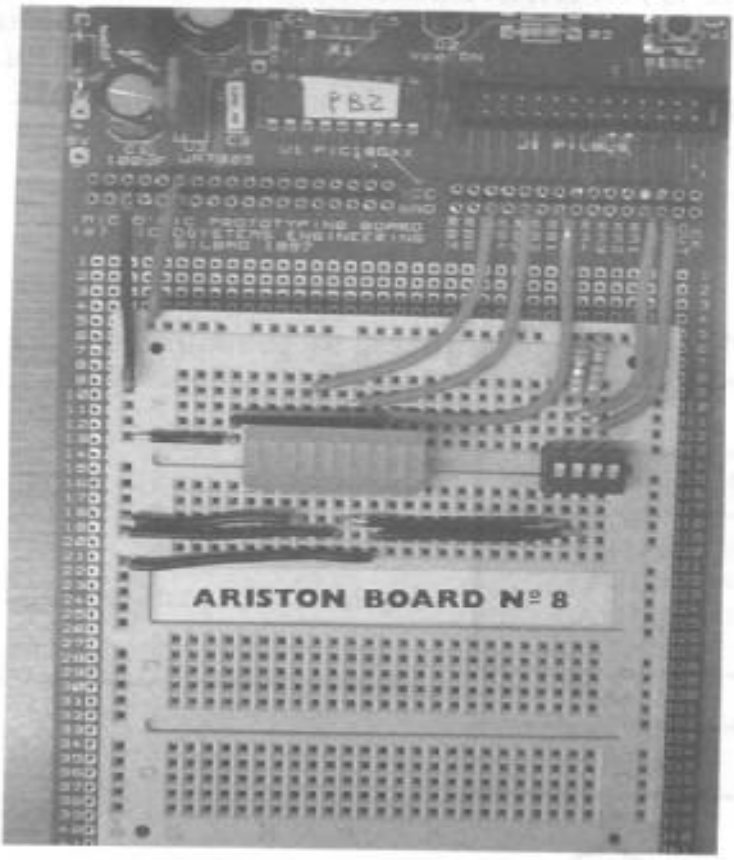


Figura F.9. *Fotografía del montaje sobre la placa PIC18-ME.*

El programa comentado

```
;*****************************************************************
;                    PROYECTO BÁSICO 2
;                «SEMÁFORO PROGRAMABLE»
;*****************************************************************
;
; Se simula un semáforo con tres diodos LED. Se dispone de dos interruptores que, según l
; combinación que adopten, originan una secuencia de encendido de las luces del semáforo.
;
;*****************************************************************
;
; DIRECTIVAS DEL PROGRAMA
;
;*****************************************************************
;
        LIST    P=16F84         ; Se emplea el PIC16F84
        RADIX   HEX             ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
W           EQU     0x00    ; Registro de destino W : d=0
F           EQU     0x01    ; Registro de destino F : d=1
Z           EQU     0x02    ; Bit 2 del registro ESTADO
TMR0_OPT    EQU     0x01    ; El registro TMR0 ocupa la dirección 1
                            ; del banco 0 y el OPTION la dirección 1
                            ; del banco 1
ESTADO      EQU     0x03    ; El registro ESTADO ocupa la dirección
                            ; 3 de los dos bancos
INTCON      EQU     0x0B    ; El registro INTCON ocupa la dirección
                            ; B de los dos bancos
PUERTAA     EQU     0x05    ; La PUERTAA ocupa la dirección 5 del
                            ; banco 0 y su registro de configuración
                            ; la dirección 5 del banco 1
PUERTAB     EQU     0x06    ; La PUERTAB ocupa la dirección 6 del
                            ; banco 0 y su registro de configuración
                            ; la dirección 6 del banco 1
AUX         EQU     0x0C    ; Registro auxiliar en el que se carga el
                            ; estado de los interruptores
TEMPO1      EQU     0x0E    ; Registro auxiliar para las rutinas de
                            ; DELAY
```

PROGRAMA PRINCIPAL

```
            ORG     0           ; El programa comienza en la dirección 0 y
            goto    INICIO      ; salta a la dirección 5 para sobrepasar el
            ORG     5           ; vector de interrupción

INICIO      bsf     ESTADO,5    ; Selección del banco 1
            movlw   b'00000011' ; Se configuran RA0 y RA1 como
            movwf   PUERTAA     ; entradas
            clrf    PUERTAB     ; Se configura la PUERTAB como salida
            movlw   b'00000111'
            movwf   TMR0_OPT    ; Se asigna el preescaler al TMR0
            bcf     ESTADO,5    ; Selección del banco 0
            clrf    PUERTAA     ; Se inicializan las dos puertas a
            clrf    PUERTAB     ; ceros
BUCLE       call    DELAY2S     ; Se deja tiempo para que el usuario programe el
                                ; semáforo
            movf    PUERTAA,W   ; Se pasa el contenido de la PUERTAA a un registro
            movwf   AUX         ; auxiliar. Dicho registro habrá que pasarlo a W antes
                                ; de cada operación XORLW, ya que la anterior habrá
                                ; cambiado su valor
            xorlw   b'00000011' ; Se hacen sucesivas máscaras para saber qué
                                ; combinación se ha introducido
            btfsc   ESTADO,Z
            goto    TRAT_1_1
```

```
        movf    AUX,W
        xorlw   b'00000010'
        btfsc   ESTADO,Z
        goto    TRAT_1_0
        movf    AUX,W
        xorlw   b'00000001'
        btfsc   ESTADO,Z
        goto    TRAT_0_1
        goto    TRAT_0_0        ; Si no es ninguna combinación anterior, tiene que ser
                                ; la correspondiente a I2=0 I1=0
```

RUTINA PARA LA COMBINACIÓN I2=1 I1=1

```
TRAT_1_1    bsf     PUERTAB,1   ; Se enciende el LED de RB1 (verde)
            call    DELAY10S    ; Se espera los segundos adecuados según I1 e I
            bcf     PUERTAB,1   ; Se apaga el LED verde
            bsf     PUERTAB,3   ; Se enciende el LED de RB3 (ámbar)
            call    DELAY2S     ; Se espera los segundos adecuados
            bcf     PUERTAB,3   ; Se apaga el LED ámbar
            bsf     PUERTAB,5   ; Se enciende el LED de RB5 (rojo)
            call    DELAY5S     ; Se espera los segundos adecuados
            bcf     PUERTAB,5   ; Se apaga el LED rojo
            goto    FIN
```

RUTINA PARA LA COMBINACIÓN I2=1 I1=1

```
TRAT_1_0    bsf     PUERTAB,1
            call    DELAY8S
            bcf     PUERTAB,1
            bsf     PUERTAB,3
            call    DELAY2S
            bcf     PUERTAB,3
            bsf     PUERTAB,5
            call    DELAY8S
            bcf     PUERTAB,5
            goto    FIN
```

RUTINA PARA LA COMBINACIÓN I2=0 I1=1

```
TRAT_0_1    bsf     PUERTAB,1
            call    DELAY5S
            bcf     PUERTAB,1
            bsf     PUERTAB,3
```

```
                call    DELAY2S
                bcf     PUERTAB,3
                bsf     PUERTAB,5
                call    DELAY10S
                bcf     PUERTAB,5
                goto    FIN
```

RUTINA PARA LA COMBINACIÓN I2=0 I1=0

```
TRAT_0_0        bsf     PUERTAB,1
                call    DELAY5S
                bcf     PUERTAB,1
                bsf     PUERTAB,3
                call    DELAY2S
                bcf     PUERTAB,3
                bsf     PUERTAB,5
                call    DELAY5S
                bcf     PUERTAB,5
                goto    FIN

FIN             goto    BUCLE           ; Bucle para repetir el tratamiento completo
```

RUTINA AUXILIAR DE DELAY

```
DELAY10         bcf     INTCON,2        ; Borra el flag de estado del TMR0
                movlw   0xD9
                movwf   TMR0_OPT        ; Carga del TMR0 para que cuente 39

DELAY10_1       btfss   INTCON,2        ; Espera el desbordamiento del TMR0
                goto    DELAY10_1
                decfsz  TEMPO1,F
                goto    DELAY10
                return
```

RUTINA DE DELAY DE UN SEGUNDO

```
DELAY1S         movlw   0x64
                movwf   TEMPO1          ; Carga la variable TEMPO1, que será
                                        ; decrementada
                call    DELAY10         ; en la rutina a la que se llama
                return
```

```
;RUTINA DE DELAY DE DOS SEGUNDOS

DELAY2S   call    DELAY1S
          call    DELAY1S
          return

;RUTINA DE DELAY DE CINCO SEGUNDOS

DELAY5S   call    DELAY2S
          call    DELAY2S
          call    DELAY1S
          return

;RUTINA DE DELAY DE OCHO SEGUNDOS

DELAY8S   call    DELAY5S
          call    DELAY2S
          call    DELAY1S
          return

;RUTINA DE DELAY DE DIEZ SEGUNDOS

DELAY10S  call    DELAY5S
          call    DELAY5S
          return

          END
```

Propuesta de un nuevo proyecto

Para continuar con el manejo del TMR0 y el control de tiempos se propone un programa que maneje ocho diodos LED de la barra DIL usada en el Proyecto Básico 2.

Los ocho diodos LED (L0-L7) de menos peso de la barra DIL se conectan a las ocho líneas (RB0-RB7) de la PUERTAB. Los LED se irán encendiendo secuencialmente durante un tiempo de 0,1 segundos de izquierda a derecha, o viceversa, según que el estado lógico que introduzca el microrruptor I0 conectado a la línea RA0 de la PUERTAA sea un «0» o un «1» res-

pectivamente. Para realizar este experimento se precisan los mismos componentes que en el Proyecto Básico 2.

PROYECTO BÁSICO 3: DADO ELECTRÓNICO

Enunciado y objetivos

Este proyecto, ya descrito de forma general anteriormente, trata de simular un juego de azar. Un dado típico tiene seis caras, en cada una de las cuales viene marcado un número de puntos de 1 a 6.

El dado es sustituido por un display de siete segmentos, que será el encargado de visualizar un número aleatorio comprendido entre el 1 y el 6 cada vez que se presione un pulsador.

Materiales específicos para el Proyecto Básico 3

1.º Display de siete segmentos 0'43" C.C. de cátodo común.
2.º Conjunto de ocho resistencias RPACK SIL de 330Ω y $^1/_4$ W (no se utiliza una).
3.º Pulsador tipo tecla C.I. blanca 12x12.
4.º Resistencia de 10KΩ y $^1/_4$ W.
5.º Resistencia de 1KΩ y $^1/_4$ W.

Foto del montaje

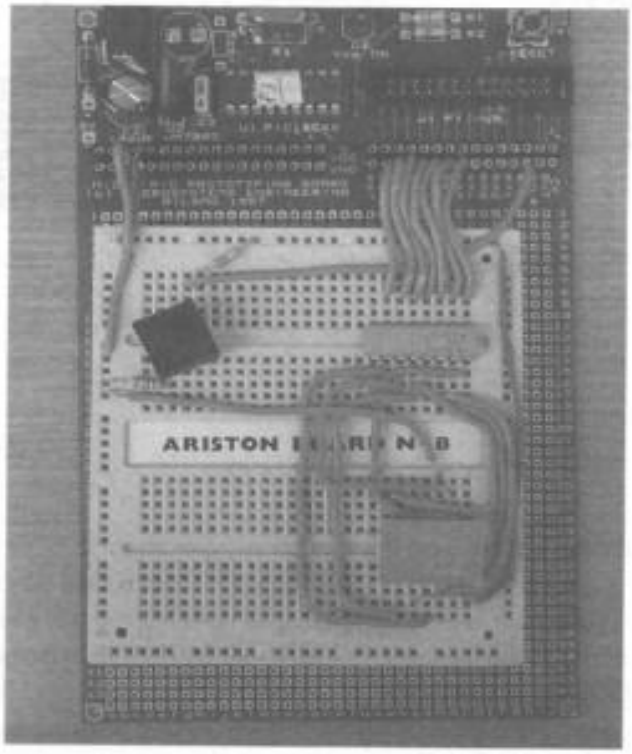


Figura F.10. *Fotografía del montaje sobre la placa PIC18-ME.*

En la Figura F.10 se muestra una foto del montaje de los periféricos del dado sobre una Protoboard y su conexionado con las líneas de E/S del PIC instalado en la tarjeta de prototipos PIC18-ME.

Esquema eléctrico

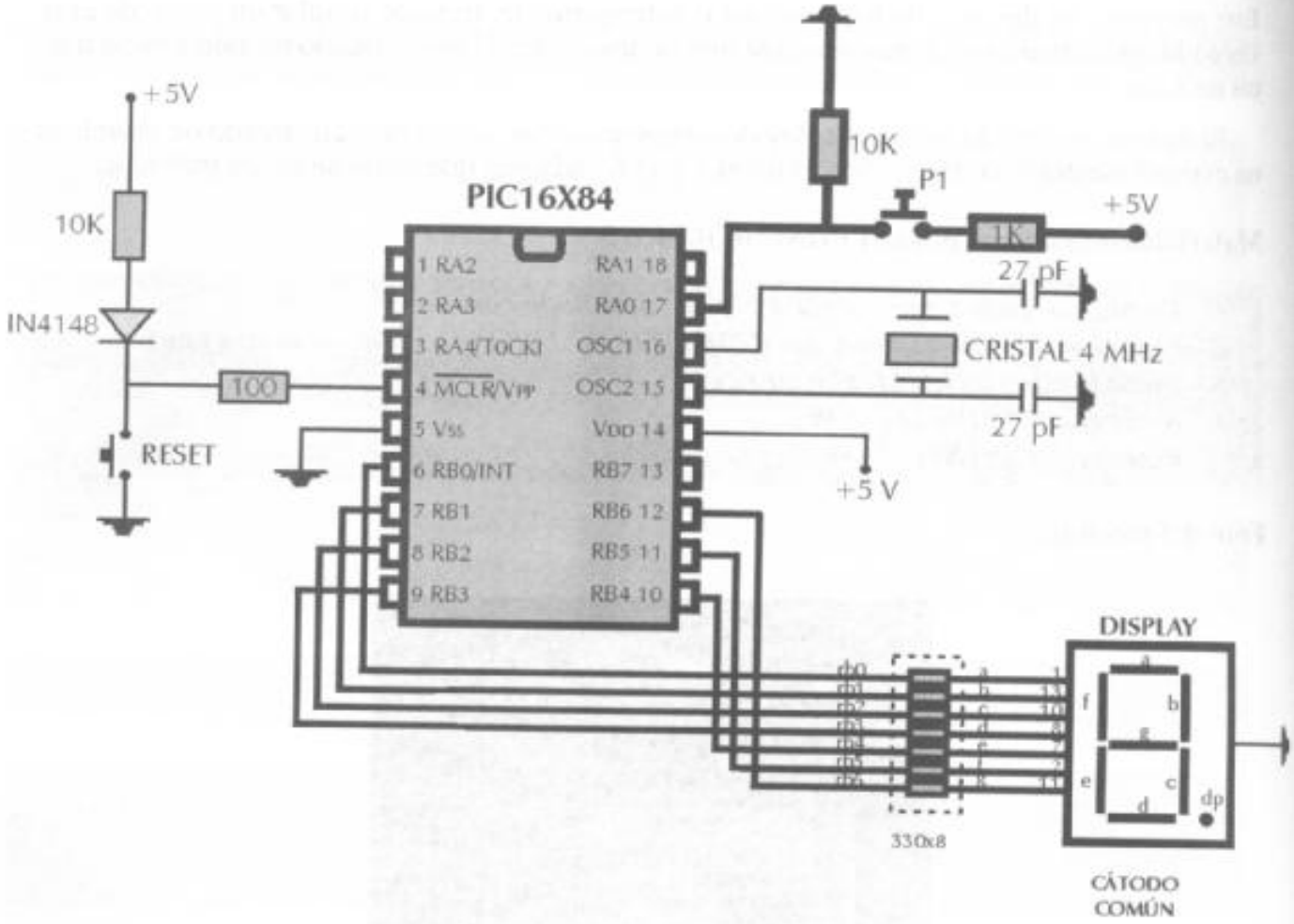


Figura F.11. *Esquema eléctrico.*

En el esquema de la Figura F.11 se muestra la conexión de las siete líneas de menos peso de la PUERTAB (RB0-RB6) a los segmentos del display a través de siete de las ocho resistencias de 330Ω contenidas en el pack. La línea de entrada RA0 soporta el nivel lógico que introduce el pulsador. Los demás componentes del circuito se hallan premontados en la tarjeta de prototipos. El pulsador introducirá un 1 lógico al activarse, aunque podría conectarse para introducir un 0 al ser pulsado. Por su parte, el display de siete segmentos también se activa por nivel alto, ya que es de cátodo común. Para que se activara por ceros tendríamos que cambiar el display por uno de ánodo común.

Organigrama



Figura F.12. *Organigrama del funcionamiento del dado.*

El programa comentado

PROYECTO BÁSICO 3
«DADO ELECTRÓNICO»

Este programa simula el juego del dado. Un dado típico tiene seis caras, en cada una de las cuales viene dibujado un número del 1 al 6. Cada vez que se presiona un pulsador se produce un número aleatorio entre 1 y 6, el cual se visualiza en un display de siete segmentos.

DIRECTIVAS DEL PROGRAMA

```
        LIST    P=16F84     ; Se emplea el PIC16F84
        RADIX   HEX         ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
        W       EQU     0x00    ; Registro de destino W : d=0
        F       EQU     0x01    ; Registro de destino F : d=1
        Z       EQU     0x02    ; Bit 2 del registro ESTADO
        PC      EQU     0x02    ; El registro PC ocupa la dirección 2 de
                                ; los dos bancos
        ESTADO  EQU     0x03    ; El registro ESTADO ocupa la dirección
                                ; 3 de los dos bancos
        PUERTAA EQU     0x05    ; La PUERTAA ocupa la dirección 5 del
                                ; banco 0 y su registro de configuración
                                ; la dirección 5 del banco 1
        PUERTAB EQU     0x06    ; La PUERTAB ocupa la dirección 6 del
                                ; banco 0 y su registro de configuración
                                ; la dirección 6 del banco 1
        RESUL   EQU     0x0E    ; Registro auxiliar para los números
                                ; aleatorios
```

PROGRAMA PRINCIPAL

```
        ORG     0           ; El programa comienza en la dirección 0 y
        goto    INICIO      ; salta a la dirección 5 para sobrepasar el
        ORG     5           ; vector de interrupción

INICIO  bsf     ESTADO,5    ; Selección del banco 1
        movlw   b'00000001' ; Se configura RA0 como entrada
        movwf   PUERTAA
        clrf    PUERTAB     ; Se configura la puerta B como salida
        bcf     ESTADO,5    ; Selección del banco 0
        movlw   0x01        ; Se inicializa la primera vez el valor de la variable
        movwf   RESUL       ; auxiliar sobre la que se hallará el número aleatorio

BUCLE   call    RANDOM      ; Se llama a la rutina que crea un número aleatorio
        call    DEC7SEG     ; Se llama a la rutina que transforma el número a 7
                            ; segmentos. Deja el resultado en W
        movwf   PUERTAB     ; Se muestra el resultado en el display
        goto    BUCLE
```

```
RUTINA DE VISUALIZACIÓN

DEC7SEG   addwf   PC,F     ; Mediante esta instrucción, el PC apunta a la
                           ; dirección de la tabla que se corresponde con el Nº a
                           ; visualizar en el display 7 segmentos. Toma el valor
                           ; hexadecimal de W y deja el valor 7 segmentos en W
          retlw   79       ; Este primer desplazamiento nunca se va a dar, ya
                           ; que se correspondería con el 0, y el dado sólo
                           ; dispone de números entre el 1 y el 6. El valor que
                           ; mostraría sería una E (Error)
          retlw   06
          retlw   5B
          retlw   4F
          retlw   66
          retlw   6D
          retlw   7D

RUTINA PARA GENERACIÓN DE NÚMERO ALEATORIO

RANDOM    btfsc   PUERTAA,0  ; Se utiliza la presión del pulsador para parar el
                             ; contaje y obtener así el número aleatorio
          goto    VOLVER     ; Si se ha pulsado, se vuelve al punto de llamada
          movlw   b'00000110' ; Se hace una comparación de RESUL con 6, por si
          subwf   RESUL,W    ; se tiene que volver a empezar de 1.
          btfsc   ESTADO,Z   ; Si RESUL=6, se inicializa; si no, se incrementa
          clrf    RESUL
          incf    RESUL,F
          goto    RANDOM     ; Se repite el bucle hasta que se pulse (orden de
                             ; parada)
VOLVER    movf    RESUL,W    ; Sobre W queda el último número antes de la
                             ; pulsación
          return

          END
```

Propuesta de un nuevo proyecto

Se dispone de un display de siete segmentos y de un conjunto de cuatro microrruptores usados en proyectos anteriores.

Se trata de visualizar en el display un dígito hexadecimal del 0 al F, según el valor binario en que se hallen los cuatro microrruptores. De esta forma, si los cuatro microrruptores introducen nivel alto, el display visualizará la F; si introducen nivel bajo, visualizará el 0, etc.

PROYECTO BÁSICO 4: TEMPORIZADOR PROGRAMABLE

Enunciado y objetivos

Se propone diseñar un temporizador que controle un tiempo medido en segundos, hasta un máximo de siete.

Para programar el temporizador se dispone de tres de los cuatro microrruptores del pack usado en proyectos anteriores. Según su posición, determinan un valor binario, que puede alcanzar el valor siete cuando los tres introducen un nivel alto. El tiempo seleccionado lo visualiza un display de siete segmentos, que se va decrementando hasta llegar a cero. También existe un pulsador para poner en marcha la cuenta del tiempo.

Como periféricos de salida, existe, además del display, un diodo LED rojo y un zumbador acústico, el primero de los cuales se ilumina y el otro genera un pitido mientras dura el período de tiempo a regular.

Materiales específicos para el Proyecto Básico 4

1.º Display de siete segmentos 0'43" C.C. de cátodo común.
2.º Conjunto de ocho resistencias RPACK SIL de 330Ω y 1/4 W (no se utiliza una).
3.º Pulsador tipo tecla C.I. blanca 12x12.
4.º Zumbador miniatura C.I. P=7,5.
5.º Diodo LED rojo 5 mm.
6.º Cinco resistencias de 10KΩ y 1/4 W.
7.º Resistencia de 1KΩ y 1/4 W.
8.º Resistencia de 330Ω y 1/4 W.
9.º Conjunto de cuatro microrruptores DIP-SWITCH (no se utiliza uno).

Esquema eléctrico

La PuertaB está configurada como salida y sus siete líneas de menos peso se asignan al control del display de siete segmentos a través de siete resistencias de 330Ω. La PuertaA actúa como entrada y recibe el estado lógico de los microrruptores que programan el tiempo. Obsérvese que en el esquema de la Figura F.13 se han instalado cuatro microrruptores en lugar de los tres que son necesarios en este ejercicio. La razón ha sido dejar preparado el montaje para el proyecto que se propone al final del tema.

En nuestro caso, los tres microrruptores válidos serán los conectados a las tres líneas de menos peso de la PuertaA (RA0, RA1 y RA2). El microrruptor de RA3 no tiene significado y su valor se pone a 0 por software cuando se lee la PuertaA.

Finalmente, la línea de salida RB7 soporta simultáneamente al diodo LED rojo y al zumbador.

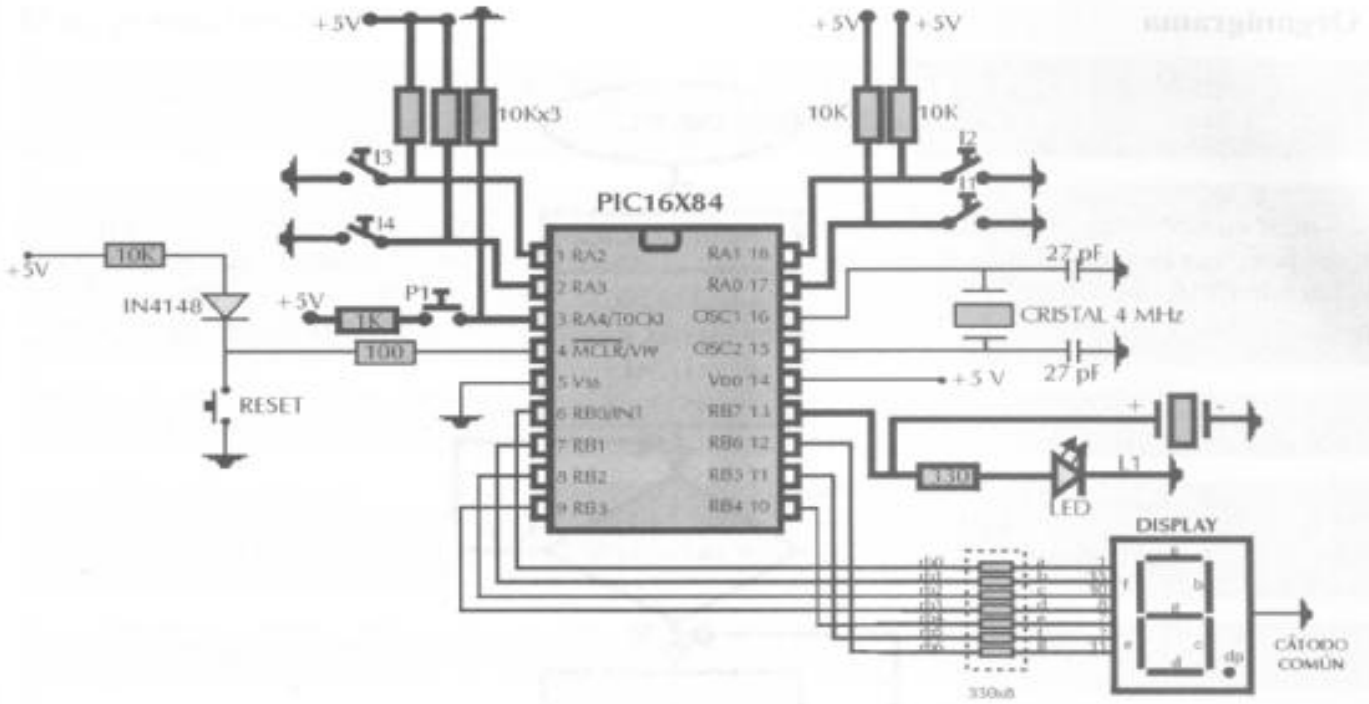


Figura F.13. *Esquema eléctrico del temporizador programable. Se ha colocado un microrruptor extra para resolver fácilmente otro proyecto propuesto posteriormente.*

Foto del montaje

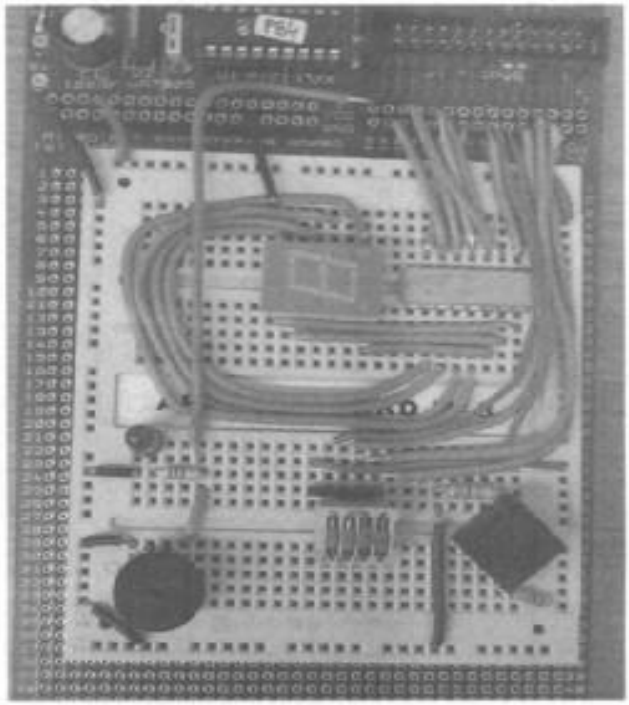


Figura F.14. *Fotografía del montaje sobre la placa PIC18-ME.*

Organigrama

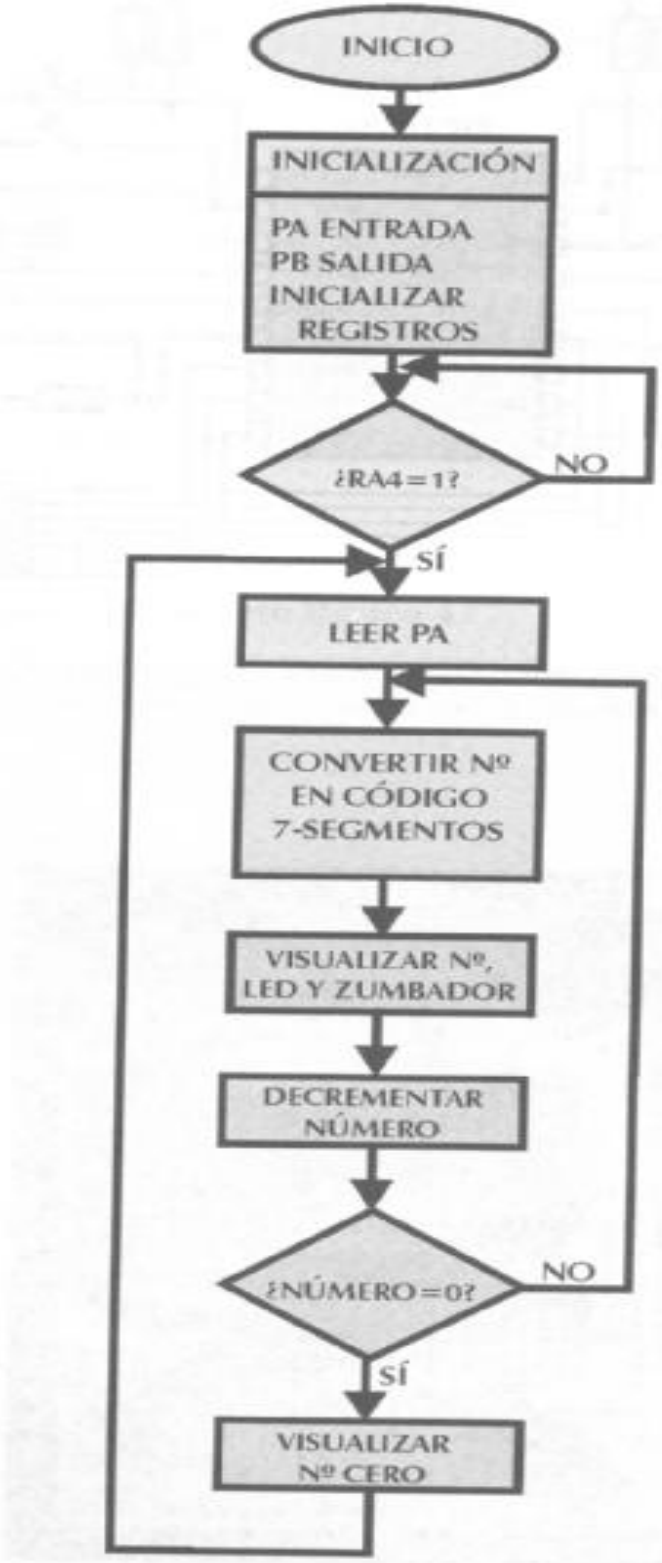


Figura F.15. *Organigrama.*

El programa comentado

PROYECTO BÁSICO 4
«TEMPORIZADOR PROGRAMABLE»

Con tres microrruptores se selecciona un valor entre el 0 y el 7 (binario), que representa un tiempo en segundos y que será visualizado por el display de siete segmentos. Durante el tiempo que dura la temporización, un LED rojo permanece activado, así como un zumbador. Al finalizar, se desactivan ambos elementos y el display se pone a 0.

DIRECTIVAS DEL PROGRAMA

```
LIST      P=16F84     ; Se emplea el PIC16F84
RADIX     HEX         ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
W         EQU   0x00  ; Registro de destino W : d=0
F         EQU   0x01  ; Registro de destino F : d=1
Z         EQU   0x02  ; Bit 2 del registro ESTADO

TMR0_OPT  EQU   0x01  ; El registro TMR0 ocupa la dirección 1
                      ; del banco 0 y el OPTION la dirección 1
                      ; del banco 1
PC        EQU   0x02  ; El registro PC ocupa la dirección 2 de
                      ; los dos bancos
ESTADO    EQU   0x03  ; El registro ESTADO ocupa la dirección
                      ; 3 de los dos bancos
INTCON    EQU   0x0B  ; El registro INTCON ocupa la dirección
                      ; B de los dos bancos
PUERTAA   EQU   0x05  ; La PUERTAA ocupa la dirección 5 del
                      ; banco 0 y su registro de configuración
                      ; la dirección 5 del banco 1
PUERTAB   EQU   0x06  ; La PUERTAB ocupa la dirección 6 del
                      ; banco 0 y su registro de configuración
                      ; la dirección 6 del banco 1
AUX       EQU   0x0C  ; Registro auxiliar en el que se carga el
                      ; estado de los interruptores
TEMPO1    EQU   0x0E  ; Registro auxiliar para las rutinas de
                      ; DELAY
```

PROGRAMA PRINCIPAL

```
          ORG   0     ; El programa comienza en la dirección 0 y
```

```
        goto    INICIO      ; salta a la dirección 5 para sobrepasar el
        ORG     5           ; vector de interrupción

INICIO  bsf     ESTADO,5    ; Selección del banco 1
        movlw   0xFF        ; Se configura la PUERTAA como
        movwf   PUERTAA     ; entrada
        clrf    PUERTAB     ; Se configura la PUERTAB como salida
        movlw   b'00000111'
        movwf   TMR0_OPT    ; Se asigna el preescaler al TMR0
        bcf     ESTADO,5    ; Selección del banco 0
        clrf    PUERTAA     ; Se inicializan las dos puertas a
        clrf    PUERTAB     ; ceros

BUCLE   btfss   PUERTAA,4   ; Se explora el valor del pulsador, que será el que indique
                            ; el comienzo del contaje de tiempo. Los microrruptores se
                            ; encontrarán ya en una determinada posición
        goto    BUCLE       ; Se continúa en el bucle hasta que se pulse
        movf    PUERTAA,W   ; Se lee el valor de los microrruptores
        andlw   b'00000111' ; Se pone a 0 con una máscara los bits que no interesan
        movwf   AUX         ; Sobre la variable AUX se llevará el contaje
        movf    AUX,F       ; Hay que controlar que el valor puede ser 0, con lo
        btfsc   ESTADO,Z    ; cual no hay que meterse en un bucle que lo
        goto    CERO        ; decremente, sino sacarlo directamente

CONTAJE movf    AUX,W       ; La rutina DEC7SEG necesita en W el valor hex.
        call    DEC7SEG     ; Se carga en W el valor a visualizar por el display
        iorlw   b'10000000' ; Se modifica sólo el bit 7 de W, que es el que se
                            ; corresponderá con RB7 (LED y zumbador), y que la
                            ; rutina de decodificación binario-7segm. ha dejado a 0
        movwf   PUERTAB     ; Se visualiza el número por el display y se activan el
                            ; LED y el zumbador
        call    DELAY1S     ; Se espera un segundo
        decfsz  AUX,F       ; Se decrementa el valor que se va mostrando
        goto    CONTAJE     ; Contará y visualizará en descendente hasta 1

CERO    movf    AUX,W
        call    DEC7SEG     ; Se traduce el valor 0
        movwf   PUERTAB     ; No hace falta activar el LED y el zumbador
        call    DELAY1S
        goto    BUCLE
```

RUTINA DE VISUALIZACIÓN

```
DEC7SEG addwf   PC,F        ; Con esta instrucción, el PC apunta a la dirección de la
                            ; tabla correspondiente al Nº a visualizar en el display
        retlw   3F          ; Retorno con carga de W con un inmediato
```

```
        retlw   06
        retlw   5B
        retlw   4F
        retlw   66
        retlw   6D
        retlw   7D
        retlw   07

RUTINA AUXILIAR DE DELAY

DELAY10     bcf     INTCON,2        ; Borra el flag de estado del TMR0
            movlw   0xD9
            movwf   TMR0_OPT        ; Carga del TMR0 para que cuente 39

DELAY10_1   btfss   INTCON,2        ; Espera el desbordamiento del TMR0
            goto    DELAY10_1
            decfsz  TEMPO1,F
            goto    DELAY10
            return

RUTINA DE DELAY DE UN SEGUNDO

DELAY1S     movlw   0x64
            movwf   TEMPO1          ; Carga la variable TEMPO1, que será decrementada
            call    DELAY10         ; en la rutina a la que se llama
            return

            END
```

Propuesta de un nuevo proyecto

En el esquema eléctrico de la Figura F.13 se introdujo un cuarto microrruptor, el cual no tenía ningún significado en dicho ejercicio.

Se propone ampliar el tiempo de contaje del temporizador para que sea capaz de alcanzar hasta 15 segundos. Para ello se usarán los cuatro microrruptores, que introducirán un valor hexadecimal entre 0 y F.

Las funciones del display, el LED rojo y el zumbador seguirán siendo las mismas que en el caso anterior, es decir, el display se irá decrementando (en hexadecimal) a medida que transcurre el tiempo programado, y el comienzo y el fin se señalizarán con el diodo y el zumbador.

PROYECTO BÁSICO 5: ALARMA

Enunciado y objetivos

Se trata de simular una alarma doméstica, para lo cual se colocan tres detectores en posibles puntos de entrada a una vivienda y avisadores acústicos y visuales. Cada sensor provoca el encendido de un diodo LED diferente cuando se activa y un pitido por un zumbador, significando que la alarma ha saltado.

Los detectores están implementados por dos microrruptores y un optoacoplador. Un optoacoplador es un sensor que manda una determinada señal (0 o 1 lógico) cuando existe haz luminoso entre el emisor y el detector, y la señal contraria cuando dicho haz se corta. Los tres diodos LED se encontrarán en la barra de LED usada en experiencias anteriores. Un tercer microrruptor servirá para conectar y desconectar la alarma.

Materiales específicos para el Proyecto Básico 5

1.° *Un optoacoplador MCA81, H21A1 o equivalente.*
2.° *Barra DIL de diez diodos LED (sólo se usan tres LED).*
3.° *Conjunto de ocho resistencias RPACK SIL de 330Ω y $^1/_4$ W (sólo se usan tres).*
4.° *Zumbador miniatura C.I. P=7,5.*
5.° *Conjunto de cuatro microrruptores DIP-SWITCH (no se utiliza una).*
6.° *Tres resistencias de 10KΩ y $^1/_4$ W.*

Esquema eléctrico

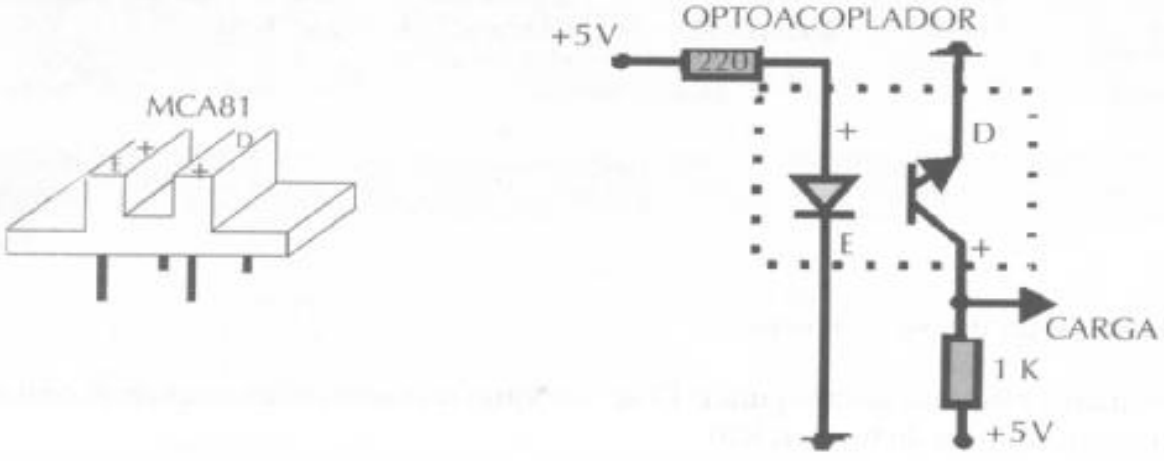


Figura F.16. *Esquema interno y de conexionado del optoacoplador MCA81.*

La Figura F.16 muestra el aspecto externo y la configuración interna de un optoacoplador estándar. Las resistencias no pertenecen al optoacoplador, sino que deben utilizarse en su conexionado.

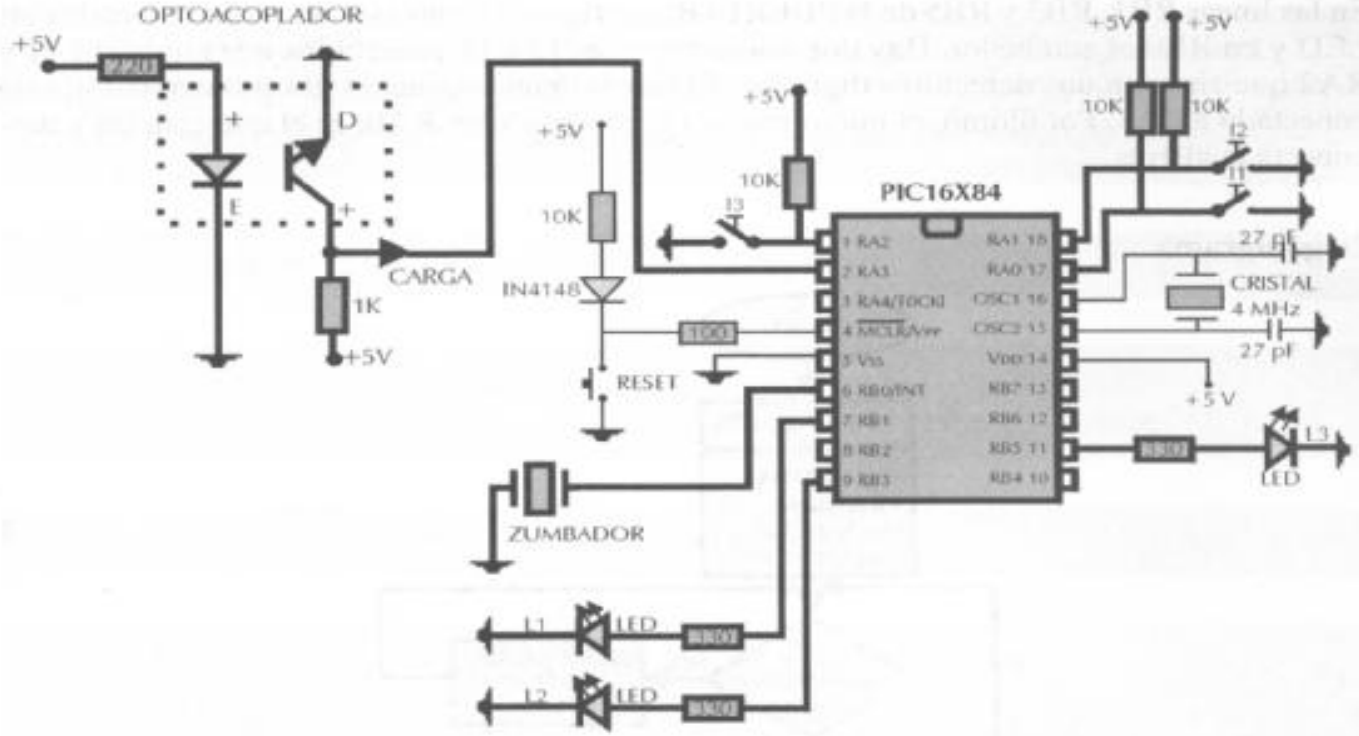


Figura F.17. *Esquema eléctrico de la alarma.*

Foto de montaje

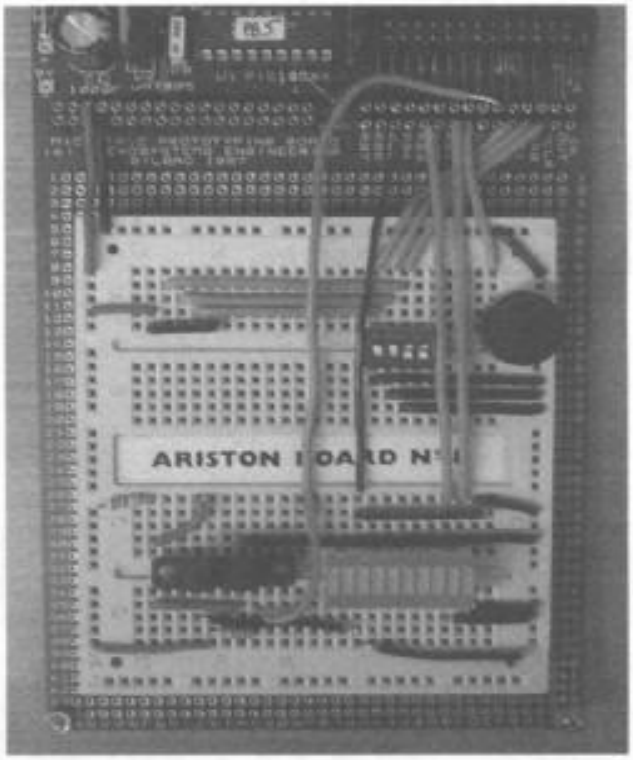


Figura F.18. *Fotografía del montaje sobre la placa PIC18-ME.*

En las líneas RB1, RB3 y RB5 de la PUERTAB, configurada como salida, están colocados los LED y en RB0 el zumbador. Hay dos microrruptores I2 e I3 conectados a las líneas RA1 y RA2 que simulan dos detectores digitales. El tercer detector, que es el optoacoplador, está conectado a RA3. Por último, el microrruptor I1, conectado en RA0, es el que conecta y desconecta la alarma.

Organigrama

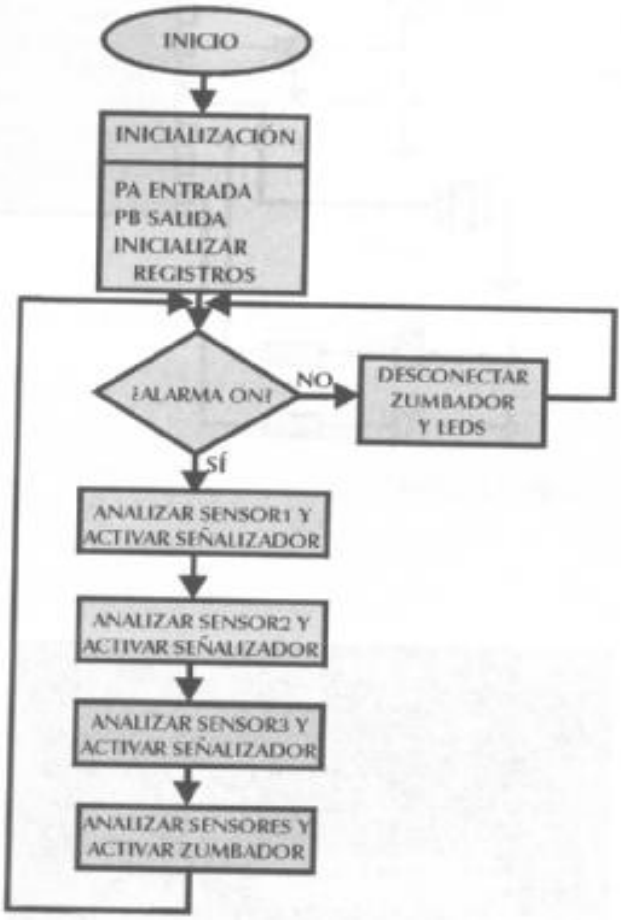


Figura F.19. *Organigrama.*

El programa comentado

PROYECTO BÁSICO 5
«ALARMA»

Se tienen tres detectores digitales, cada uno de los cuales provocará la activación de un LED, diferente en cada caso. Uno de los detectores será un optoacoplador y los otros dos se simularán mediante dos microrruptores.

El programa consiste en poner en marcha un zumbador cuando la alarma salte por causa de la activación de cualquiera de los detectores. También se enciende el LED correspondiente al detector.

La alarma sólo funcionará cuando el microrruptor I1 de conexión esté a nivel alto.

DIRECTIVAS DEL PROGRAMA

```
        LIST    P=16F84     ; Se emplea el PIC16F84
        RADIX   HEX         ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
        W       EQU   0x00  ; Registro de destino W : d=0
        F       EQU   0x01  ; Registro de destino F : d=1
        Z       EQU   0x02  ; Bit 2 del registro ESTADO
        ESTADO  EQU   0x03  ; El registro ESTADO ocupa la dirección
                            ; 3 de los dos bancos
        PUERTAA EQU   0x05  ; La PUERTAA ocupa la dirección 5 del
                            ; banco 0 y su registro de configuración
                            ; la dirección 5 del banco 1
        PUERTAB EQU   0x06  ; La PUERTAB ocupa la dirección 6 del
                            ; banco 0 y su registro de configuración
                            ; la dirección 6 del banco 1
```

PROGRAMA PRINCIPAL

```
        ORG     0           ; El programa comienza en la dirección 0 y
        goto    INICIO      ; salta a la dirección 5 para sobrepasar el
        ORG     5           ; vector de interrupción

INICIO  bsf     ESTADO,5    ; Selección del banco 1
        movlw   0xFF        ; Se configura la PUERTAA como
        movwf   PUERTAA     ; entrada
        clrf    PUERTAB     ; Se configura la PUERTAB como salida
        bcf     ESTADO,5    ; Selección del banco 0
        clrf    PUERTAA     ; Se inicializan las dos puertas a
        clrf    PUERTAB     ; ceros

BUCLE   btfss   PUERTAA,0   ; Se explora el valor del microrruptor que indica si la
                            ; alarma está o no conectada
```

```
        goto    DESCON      ; Si está desconectada, habrá que desactivar los LED
                            ; y el zumbador por si estaban activados de una
                            ; anterior pasada por el bucle
        call    SENS1       ; Se comprueba el estado de los tres sensores y se
                            ; hace el tratamiento que corresponda con los LED
        call    SENS2
        call    SENS3
        call    ZUMB        ; Si alguno de los sensores está activado, el
                            ; zumbador deberá pitar    ; si no, no
        goto    BUCLE

DESCON  clrf    PUERTAB
        goto    BUCLE
```

RUTINA DE TRATAMIENTO DEL SENSOR 1

```
SENS1   btfss   PUERTAA,1   ; Si está a 1, hay que activar el LED
                            ; correspondiente
        goto    DESACT1     ; Si no, hay que desactivarlo
        goto    ACT1

DESACT1 bcf     PUERTAB,1
        goto    VOLVER1

ACT1    bsf     PUERTAB,1
        goto    VOLVER1

VOLVER1 return
```

RUTINA DE TRATAMIENTO DEL SENSOR 2

```
SENS2   btfss   PUERTAA,2   ; Si está a 1, hay que activar el LED
                            ; correspondiente
        goto    DESACT2     ; Si no, hay que desactivarlo
        goto    ACT2

DESACT2 bcf     PUERTAB,3
        goto    VOLVER2

ACT2    bsf     PUERTAB,3
        goto    VOLVER2

VOLVER2 return
```

```
RUTINA DE TRATAMIENTO DEL SENSOR 3

SENS3       btfss   PUERTAA,3   ; Si está a 1, hay que activar el LED
                                ; correspondiente
            goto    DESACT3     ; Si no, hay que desactivarlo
            goto    ACT3

DESACT3     bcf     PUERTAB,5
            goto    VOLVER3

ACT3        bsf     PUERTAB,5
            goto    VOLVER3

VOLVER3     return

RUTINA DE TRATAMIENTO DEL ZUMBADOR

ZUMB        movf    PUERTAA,W
            andlw   b'00001110' ; Con esta máscara, se consigue que sobre W quede
                                ; sólo el valor de los sensores. Si ninguno está
                                ; activo, W contendrá 0 y el flag Z se pondrá a 1.
            btfss   ESTADO,Z    ; Si está a 1, hay que desactivar el zumbador.
            goto    ACTZ        ; Si no, hay que activarlo
            goto    DESACTZ

DESACTZ     bcf     PUERTAB,0
            goto    VOLVERZ

ACTZ        bsf     PUERTAB,0
            goto    VOLVERZ

VOLVERZ     return

            END
```

Propuesta de un nuevo proyecto

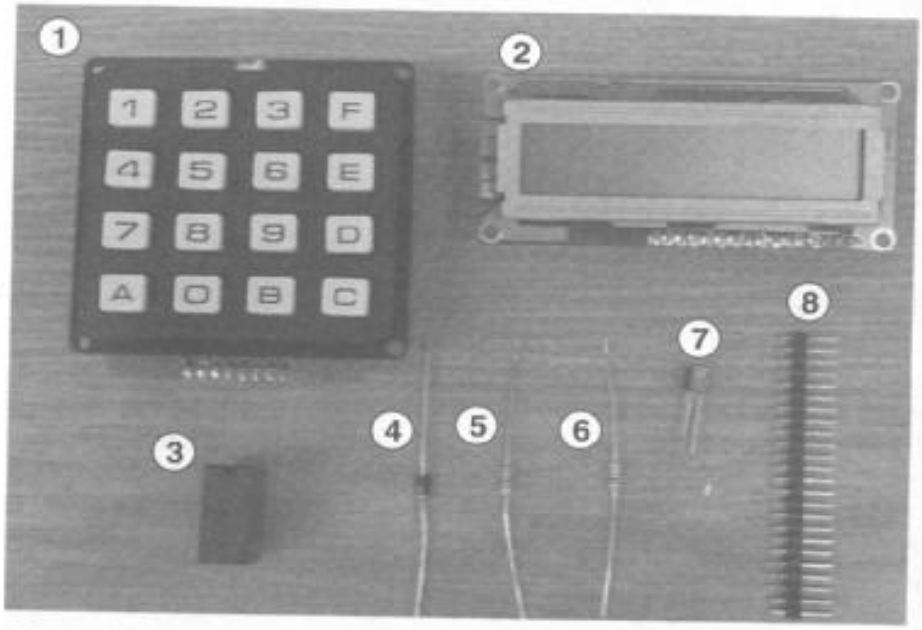
Sin modificar el hardware de la alarma, confeccionar un nuevo programa que opere de forma similar, pero con los siguientes cambios:

1.º La activación de los LED no es permanente, sino que parpadean cada segundo.
2.º El zumbador sólo genera un pitido cuando es el optoacoplador el que se ha activado.
3.º Cuando uno de los detectores se activa, el primer LED (RB1) parpadea. Si se activan dos de los detectores, parpadean los dos primeros LED (RB1 y RB3), y si están activados los tres detectores, parpadean los tres LED.

SEGUNDA PARTE: PROYECTOS TÍPICOS RESUELTOS Y COMENTADOS

LISTA DE MATERIALES NECESARIOS

En la fotografía de la Figura F.20 se muestran los componentes utilizados en los Proyectos Típicos. Se supone que adicionalmente se tienen los componentes que se usaban en los Proyectos Básicos.



1. Teclado
2. LCD
3. Microrelé
4. Diodo
5. R 820. Gris-rojo-marrón
6. R 10K. Marrón-negro-naranja
7. Transistor C307B
8. Tira de Pines

Figura F.20. *Materiales utilizados en los Proyectos Típicos.*

Materiales comunes a todos los Proyectos Típicos

1.º Protoboard para el montaje sin soldadura.
2.º Un microcontrolador PIC16F84.
3.º Sistema de desarrollo Micro'PIC Trainer para grabar y borrar el PIC y depurar los programas.
4.º Cables de un solo hilo de cobre preestañado de 0,8 mm de diámetro.

PROYECTO TÍPICO 1: TECLADO MUSICAL

Introducción. Instalación y manejo del teclado

Un periférico muy habitual para introducir información al sistema por parte del usuario es el teclado. Nos referimos al teclado matricial de 16 teclas que se incluye en el kit y que están distribuidas en cuatro filas y cuatro columnas (Figura F.21). Los ocho terminales del teclado, cuatro para las filas y cuatro para las columnas, se conectarán a las ocho líneas de E/S de la PuertaB del PIC. Cuando se pulsa una tecla, queda en contacto una fila con una columna. Si ninguna tecla está presionada, las filas están desconectadas de las columnas.



Figura F.21. *Estructura y conexionado interno del teclado matricial de 16 teclas distribuidas en cuatro filas y cuatro columnas.*

Para explorar el teclado y averiguar en cada momento cuál es la tecla que se halla pulsada, se sigue el siguiente método. Se va a trabajar con el nivel lógico alto actuando como nivel inactivo, siendo el nivel bajo al que consideraremos activo. Por los terminales de las filas del teclado se introducen cuatro niveles lógicos, uno activo (bajo) y los otros tres inactivos (altos). Si existe alguna tecla pulsada en la fila por la que se ha introducido el nivel bajo, dicho nivel saldrá por la columna correspondiente con la que haga contacto. En consecuencia, leyendo los estados lógicos de los terminales de las columnas (RB0-RB3) averiguaremos si hay alguna tecla pulsada en la fila por la que se ha introducido el nivel bajo, puesto que si no hay ninguna en todas las columnas se leerá nivel alto inactivo. Procediendo secuencialmente a introducir el nivel bajo por cada una de las cuatro filas y a leer los niveles de salida en las columnas, se podrá determinar la tecla pulsada en cada momento.

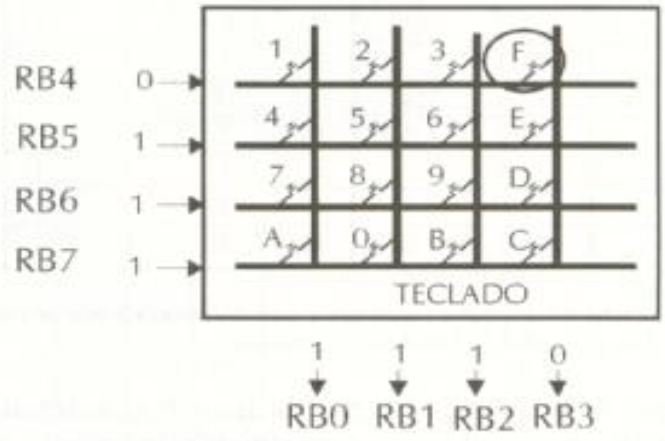


Figura F.22. *Al introducir a las filas el código 1110 por las líneas de salida de la PUERTAB (RB7-RB4), se obtiene en las filas conectadas a las líneas de entrada de PB (RB3-RB0) el código 0111 si la tecla F está presionada.*

Por ejemplo, si por las líneas RB7-RB4 se aplica a las filas los niveles lógicos 1110, el nivel bajo quedará soportado por la primera fila, la unida a la línea RB4. Si en esta situación se halla pulsada la tecla F, al leer los estados de las columnas presentes sobre las líneas RB3-RB0, por la línea RB3 saldrá el nivel bajo, mientras que por las restantes habrá nivel alto. Se habrá obtenido en las columnas el código 0111 (RB3-RB0). En resumen, se ha introducido el código 1110 a las filas por las líneas RB7-RB4 y en las líneas RB3-RB0 se lee el código 0111 que se corresponde con la tecla F.

El programa de exploración del teclado irá introduciendo por las cuatro líneas de más peso de PB los cuatro códigos correspondientes a la introducción del nivel bajo en cada caso por una de ellas, o sea, los códigos 0111, 1011, 1101 y 1110. Al introducir cada uno de estos códigos se explora el nivel en las columnas conectadas a las líneas de menos peso de PB. Cuando sale un nivel bajo por una columna, conociendo en ese momento por qué fila se ha introducido el nivel bajo, se averigua la tecla presionada por la intersección de dicha fila con dicha columna. Cuando en la exploración no hay ninguna tecla pulsada se suele generar un código concreto que representa esa posibilidad.

En la Figura F.23 se muestra la tabla correspondiente a la pulsación de cada una de las 16 teclas y el código hexadecimal que existe con los niveles lógicos que se introducen por las filas y los que se obtienen en las columnas.

TECLA	FILAS (ENTRADAS)				COLUMNAS (SALIDAS)				CÓDIGO HEX
	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	
0	0	1	1	1	1	1	0	1	7D
1	1	1	1	0	1	1	1	0	EE
2	1	1	1	0	1	1	0	1	ED
3	1	1	1	0	1	0	1	1	EB
4	1	1	0	1	1	1	1	0	DE
5	1	1	0	1	1	1	0	1	DD
6	1	1	0	1	1	0	1	1	DB
7	1	0	1	1	1	1	1	0	BE
8	1	0	1	1	1	1	0	1	BD
9	1	0	1	1	1	0	1	1	BB
A	0	1	1	1	1	1	1	0	7E
B	0	1	1	1	1	0	1	1	7B
C	0	1	1	1	0	1	1	1	77
D	1	0	1	1	0	1	1	1	B7
E	1	1	0	1	0	1	1	1	D7
F	1	1	1	0	0	1	1	1	E7

Figura F.23. *Tabla que muestra los códigos binarios y hexadecimales que se obtienen con la pulsación de cada una de las 16 teclas del teclado matricial.*

Cuando no hay ninguna tecla pulsada en una fila aparecen cuatro niveles altos en las cuatro columnas, por estar desconectadas al aire y corresponderlas estado de alta impedancia, que en lógica TTL es equivalente al nivel alto. Para asegurar dicho nivel alto, las líneas de la PUERTAB pueden conectarse al positivo de la alimentación mediante unas resistencias de *pull-up* internas.

Esta conexión puede realizarse por software, poniendo a 0 el bit 7 (RBPU#) del registro OPTION del PIC16F84 (en el PIC16C84 este bit tiene programación inversa). Se recomienda colocar cuatro resistencias de 2K2, como se muestra en la Figura F.24.

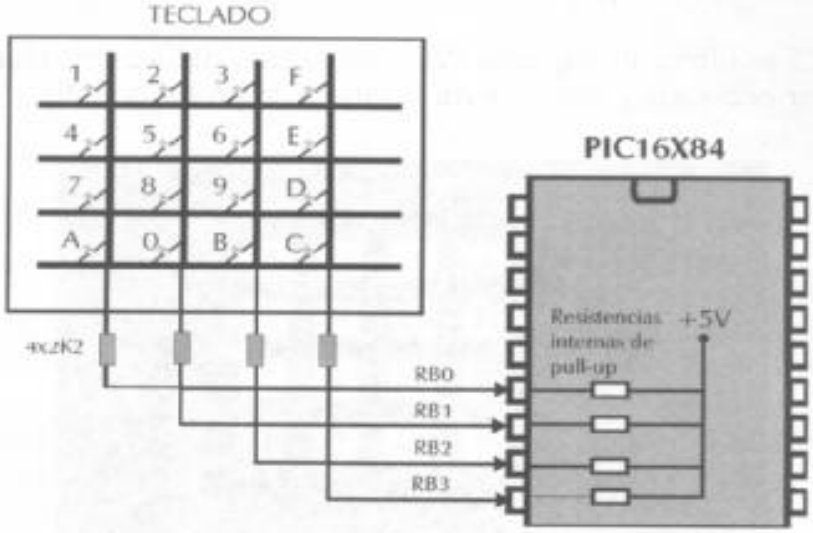


Figura F.24. *Las líneas de la PUERTAB del PIC16F84 pueden conectarse al positivo a través de unas resistencias internas de* pull-up *por software. Así, cuando no hay ninguna tecla pulsada existe un nivel lógico alto.*

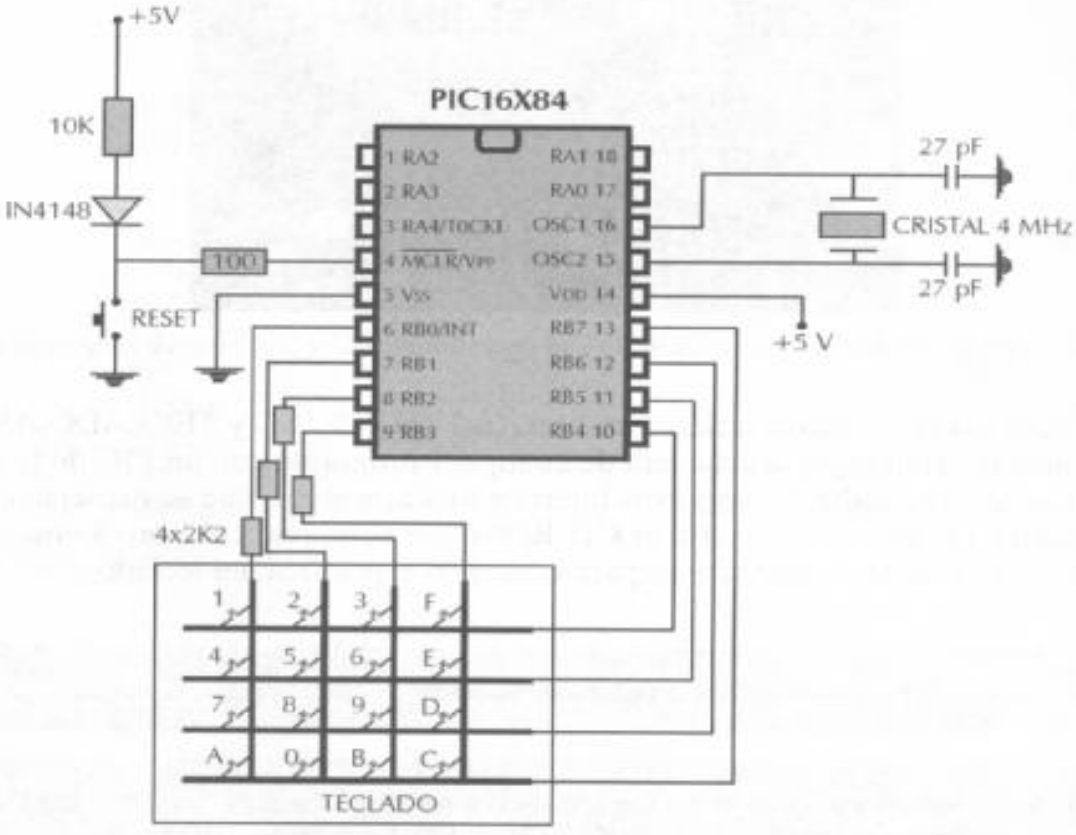


Figura F.25. *Conexionado del teclado con las líneas de la PUERTAB del PIC16F84.*

Otra precaución que hay que tener con los teclados es la introducción de pequeños retardos en la exploración de las teclas pulsadas para evitar los «rebotes». Un retardo adecuado para eliminar los rebotes es de 20 ms, que se puede conseguir utilizando el timer TMR0 al cargarle con el valor $78_{10} = 4E_{16}$, y asignar al divisor de frecuencias el rango de 256.

En la Figura F.25 se ofrece el esquema de conexionado del teclado con el PIC16F84, con la circuitería auxiliar necesaria y que soporta la tarjeta de prototipos PIC18-ME.

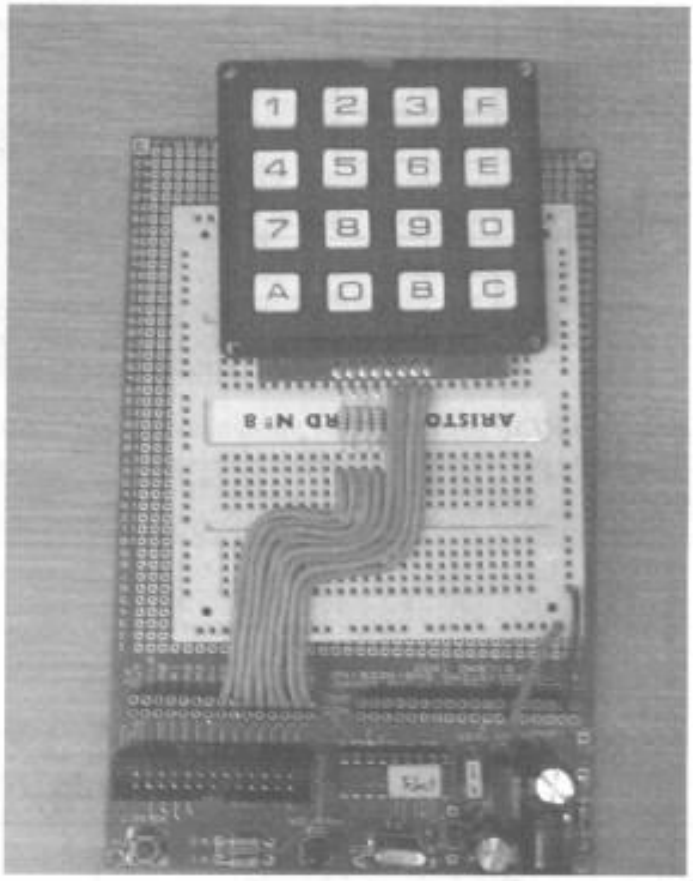


Figura F.26. *Fotografía del montaje del teclado y su conexionado sobre la tarjeta de prototipos PIC18-ME.*

Para el manejo del teclado se utilizan las librerías P16CXX.INC y TECLADO.ASM. Se recomienda usar la primera en la cabecera de cualquier programa con un PIC de la gama media. En ella se definen todos los registros internos fundamentales que se manejan en el código que resuelve las aplicaciones. En el CD-ROM que acompaña al libro se incluye dicha rutina. TECLADO.ASM recoge rutinas para la gestión específica del teclado.

TECLADO.ASM

Conjunto de rutinas para el manejo del teclado. No es un programa en sí mismo. Para utilizarlo hay que colocar en el programa principal: KEY_VAR EQU dir_inicio_del_bloque.

BLOQUE DE ETIQUETAS

```
CBLOCK    KEY_VAR                    ; Inicio de las variables
                  TECLA              ; Retorno del código de tecla
                  KEY_1              ; Nº de filas a explorar
                  KEY_2              ; Temporal de código
                  KEY_DELAY_1        ; Variable de temporización
                  KEY_DELAY_2        ; Variable de temporización
ENDC
```

RUTINA KEY_SCAN: Rutina de exploración del teclado. La variable «Tecla» retorna con el código de la tecla pulsada o el código 0x80 si no se pulsa ninguna.

```
KEY_SCAN    bsf     STATUS,RP0              ; Selecciona página 1
            movlw   b'00001111'
            movwf   PORTB                   ; RB7-RB4 salidas, RB3-RB0 entradas
            bsf     OPTION_REG,NOT_RBPU     ; Activa cargas pull-up
            bcf     STATUS,RP0              ; Selecciona página 0
            movlw   4
            movwf   KEY_1                   ; Nº de columnas a explorar
            movlw   b'01111111'
            movwf   TECLA                   ; Columna a activar

KEY_SCAN_1  movf    TECLA,W
            movwf   PORTB                   ; Activa fila
            nop
            movf    PORTB,W
            movwf   KEY_2
            subwf   TECLA,W                 ; Lee las columnas
            btfss   STATUS,Z                ; ¿Hay alguna pulsada?
            goto    KEY_SCAN_2              ; Si hay alguna pulsada
            bsf     STATUS,C                ; No hay ninguna en esa fila
            rrf     TECLA,F                 ; Selecciona siguiente fila
            decfsz  KEY_1,F                 ; Salta si se han terminado las filas
            goto    KEY_SCAN_1
            movlw   0x80
            goto    VOLVER                  ; Retorna código 0x80 (no hay pulsación)

KEY_SCAN_2  movlw   .100                    ; Bucle de temporización de unos 20 ms
            movwf   KEY_DELAY_1             ; para evitar el rebote de los pulsadores

KEY_SCAN_3  clrf    KEY_DELAY_2
            clrwdt

KEY_SCAN_4  decfsz  KEY_DELAY_2,F
            goto    KEY_SCAN_4
            decfsz  KEY_DELAY_1,F
```

```
                goto    KEY_SCAN_3
                movf    TECLA,W
                movwf   PORTB           ; Tras la temporización se lee nuevamente
                nop                     ; si la tecla es la misma. Así se
                movf    PORTB,W         ; evitan los rebotes.
                subwf   KEY_2,W
                btfss   STATUS,Z        ; ¿Es la misma?
                goto    KEY_SCAN_1      ; No, seguir con la exploración
                movf    KEY_2,W         ; Sí, guardar en variable de salida TECLA el
VOLVER          movwf   TECLA           ; valor recogido
                return                  ; Fin de exploración
```

RUTINA KEY_HEX: Convierte el código de tecla que haya en la variable «Tecla» a Hex. (0-F). El resultado se devuelve en la variable «Tecla». Usa como rutina auxiliar Key_tabla.

```
KEY_TABLA       movf    KEY_1,W
                addwf   PCL,F           ; Calcula desplazamiento
                retlw   0x7D            ; 0
                retlw   0xEE            ; 1
                retlw   0xED            ; 2
                retlw   0xEB            ; 3
                retlw   0xDE            ; 4
                retlw   0xDD            ; 5
                retlw   0xDB            ; 6
                retlw   0xBE            ; 7
                retlw   0xBD            ; 8
                retlw   0xBB            ; 9
                retlw   0x7E            ; A
                retlw   0x7B            ; B
                retlw   0x77            ; C
                retlw   0xB7            ; D
                retlw   0xD7            ; E
                retlw   0xE7            ; F
KEY_HEX         movf    TECLA,W
                movwf   KEY_2           ; Almacena el código temporalmente
                clrf    KEY_1           ; Contador HEX a 0
KEY_HEX_2       call    KEY_TABLA       ; Busca código en la tabla
                subwf   KEY_2,W         ; Compara con el de la tecla
                btfsc   STATUS,Z        ; ¿Coincide ?
                goto    KEY_HEX_1       ; Sí
                incf    KEY_1,F         ; No, incrementa contador HEX
                goto    KEY_HEX_2
KEY_HEX_1       movf    KEY_1,W
                movwf   TECLA           ; Carga contador HEX en la variable de salida
                return
```

RUTINA HEX_7SEG: Rutina de conversión de HEX (0-F) a siete segmentos. El acumulador W contiene, durante la llamada, el código HEX y, al retornar, el correspondiente a siete segmentos.

```
HEX_7SEG    andlw   b'00001111'
            addwf   PCL,F           ; Desplazamiento sobre la tabla
            retlw   b'00111111'     ; 0
            retlw   b'00000110'     ; 1
            retlw   b'01011011'     ; 2
            retlw   b'01001111'     ; 3
            retlw   b'01100110'     ; 4
            retlw   b'01101101'     ; 5
            retlw   b'01111101'     ; 6
            retlw   b'00000111'     ; 7
            retlw   b'01111111'     ; 8
            retlw   b'01100111'     ; 9
            retlw   b'01110111'     ; A
            retlw   b'01111100'     ; B
            retlw   b'00111001'     ; C
            retlw   b'01011110'     ; D
            retlw   b'01111001'     ; E
            retlw   b'01110001'     ; F
```

Una vez vistas las rutinas que se usan con el teclado y su funcionamiento básico se pasa a realizar el primero de los proyectos con teclado.

Enunciado y objetivos

Utilizando las librerías TECLADO.ASM y P16CXX.INC, se trata de realizar un programa que genere un pitido por el zumbador piezoeléctrico conectado a RA3, cuya duración dependerá del valor de la tecla pulsada.

La tecla «0» producirá un pitido de una duración determinada, dicha duración se irá incrementando con el valor de cada tecla hasta llegar a la «F», cuyo pitido será el de duración mayor.

Materiales específicos para el Proyecto Típico 1

1.º Teclado matricial de 16 teclas hexadecimales (0-F).
2.º Cuatro resistencias de 2K2Ω y ¼ W.
3.º Zumbador miniatura C.I. P=7,5.

Esquema eléctrico

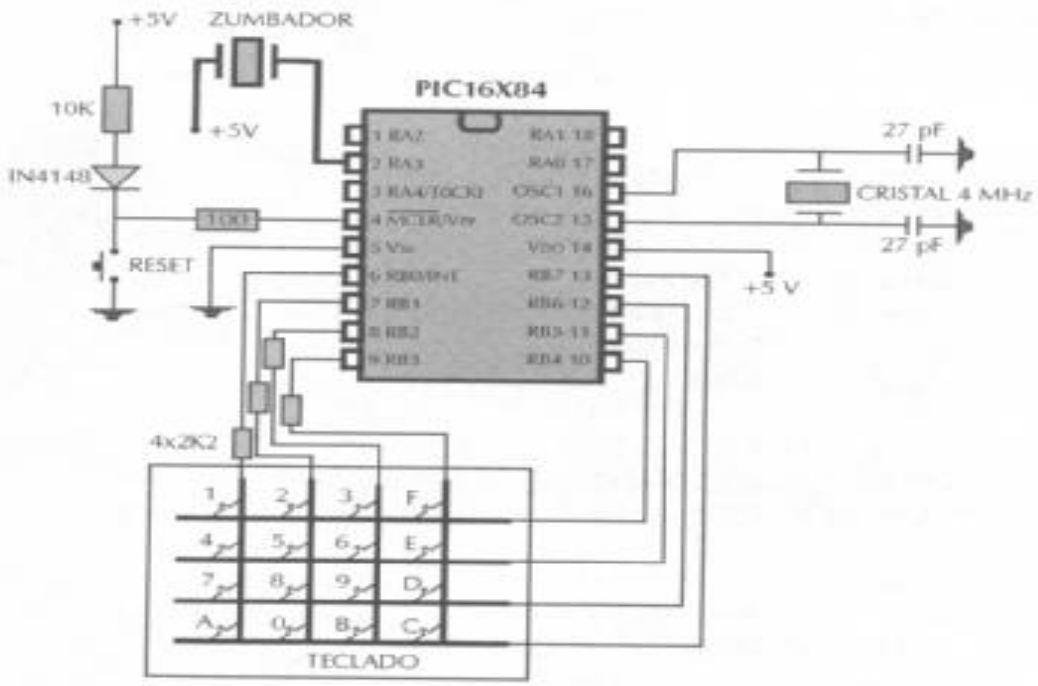


Figura F.27. *Esquema de conexionado.*

Foto del montaje

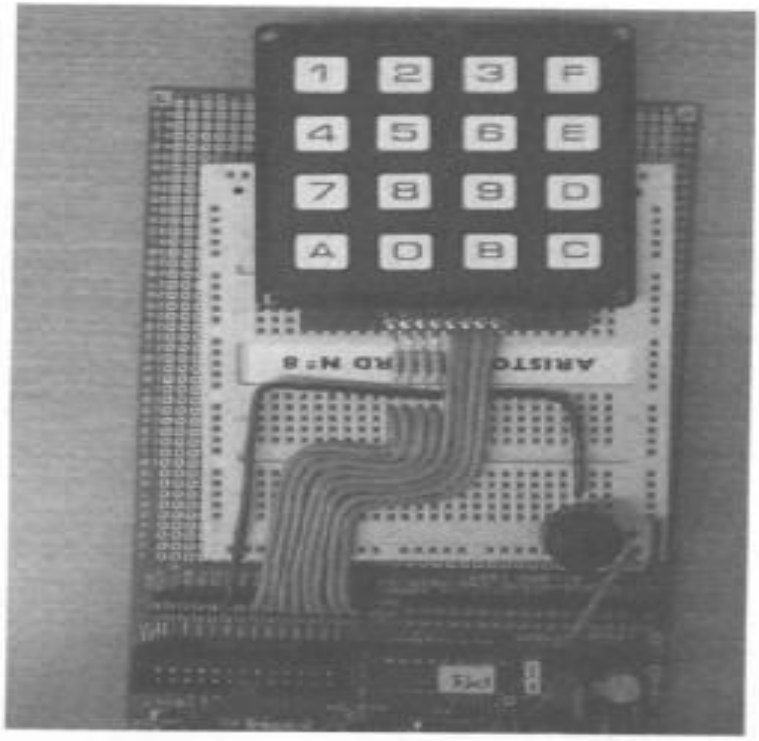


Figura F.28. *Fotografía del montaje sobre la placa PIC18-ME.*

Organigrama

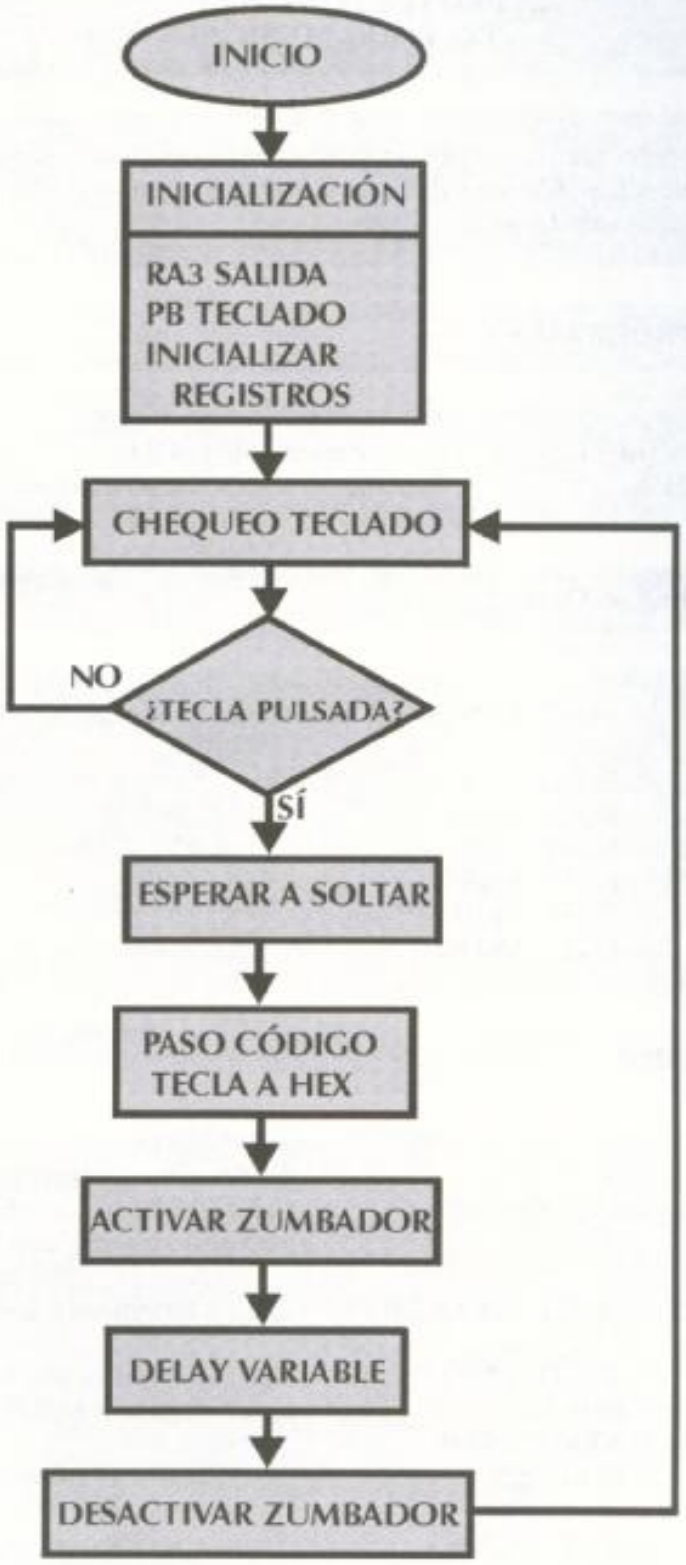


Figura F.29. *Organigrama del proyecto.*

El **programa comentado**

PROYECTO TÍPICO 1
«TECLADO MUSICAL»

Se desea realizar un programa que genere un pitido por el zumbador piezoeléctrico conectado a RA3, cuya duración dependa del valor de la tecla pulsada, siendo la «0» la tecla de pitido más corto y la «F» la de pitido más largo.

DIRECTIVAS DEL PROGRAMA

```
LIST        P=16F84      ; Se emplea el PIC 16F84
RADIX       HEX          ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
        INCLUDE     «16CXX.INC»     ; Etiquetas genéricas para los PIC 16CXX

        TEMP_1      EQU  0x0C
        TEMP_2      EQU  0x0D
        DELAY_1     EQU  0x0E
        DELAY_2     EQU  0x0F
        DELAY_3     EQU  0x10
        KEY_VAR     EQU  0x11
```

PROGRAMA PRINCIPAL

```
            ORG     0               ; El programa comienza en la dirección 0 y
            goto    INICIO          ; salta a la dirección 5 para sobrepasar el
            ORG     5               ; vector de interrupción

            INCLUDE «TECLADO.ASM»   ; Coloca las rutinas tras el vector de interrupción

INICIO      bsf     STATUS,RP0      ; Selección del banco 1
            clrf    PORTA           ; Se configura la PUERTAA como salida
            bcf     STATUS,RP0
            bsf     PORTA,3         ; Se desactiva el zumbador

NO_HAY      call    KEY_SCAN        ; Se hace un chequeo del teclado
            movlw   0x80
```

```
        subwf   TECLA,W     ; Sobre TECLA quedará la tecla pulsada o 0x80
        btfsc   STATUS,Z
        goto    NO_HAY      ; No se ha pulsado ninguna tecla
        call    KEY_OFF     ; Sí se ha pulsado. Se espera a que se suelte
        call    KEY_HEX     ; Se pasa el valor a hexadecimal
        movf    TECLA,W     ; Ese valor servirá para determinar la duración del beep
        addlw   0x01        ; Se suma 1 para que el «0» no sea el de mayor temporización
        bcf     PORTA,3     ; Se activa el zumbador (por nivel bajo)
        call    DELAY_V
        bsf     PORTA,3     ; Se desactiva el zumbador
        goto    NO_HAY      ; Se vuelve a empezar
```

RUTINA KEY_OFF: Esta rutina espera que la tecla recién pulsada sea soltada. Debe usarse justo después de llamar a la rutina KEY_SCAN.

```
KEY_OFF     movf    TECLA,W
            movwf   TEMP_1      ; Guarda temporalmente la tecla
KEY_OFF_NO  call    KEY_SCAN    ; Mira si se ha soltado
            movlw   0x80
            subwf   TECLA,W
            btfss   STATUS,Z
            goto    KEY_OFF_NO  ; Bucle mientras no se suelte
            movf    TEMP_1,W    ; Ya se ha soltado
            movwf   TECLA       ; Se repone la tecla
            return
```

RUTINA DE DELAY VARIABLE: Esta rutina realiza una temporización que depende del valor del acumulador en el momento en que se le llama.

```
DELAY_V     movwf   DELAY_1
            clrf    DELAY_3
            clrf    DELAY_2

DELAY_LOOP  decfsz  DELAY_3,F
            goto    DELAY_LOOP
            decfsz  DELAY_2,F
            goto    DELAY_LOOP
            decfsz  DELAY_1,F
            goto    DELAY_LOOP
            return

            END
```

Propuesta de un nuevo proyecto

Se propone una modificación sobre el programa anterior. Esta vez, la pulsación de la tecla «0» debe generar el pitido más largo y la de la tecla «F» el más corto.

Se recomienda introducir una rutina de conversión y su llamada correspondiente como único cambio en el programa.

PROYECTO TÍPICO 2: LCD

Introducción: instalación y manejo del LCD

La pantalla de cristal líquido, LCD, es un periférico de salida visualizador muy potente, flexible, eficaz y económico, siendo los microcontroladores los dispositivos más adecuados para optimizar sus prestaciones y control. En realidad, el módulo LCD tiene incrustado un microcontrolador específico para regular su funcionamiento.

Las pantallas LCD más comunes presentan un número variable de caracteres formados por una matriz de 5 × 7 pixels en una o varias líneas. Nos referiremos al módulo LCD con microcontrolador incrustado, modelo 44780 de *Hitachi*, que tiene la posibilidad de visualizar dos líneas de 16 caracteres cada una. Dispone de 14 terminales de conexionado, cuya misión es la siguiente:

1.º Ocho terminales (D0-D7) reciben los caracteres ASCII a representar, así como ciertos códigos de control que regulan los efectos de visualización. También por ellos, el módulo LCD envía información sobre su estado interno.
2.º Por los terminales Vdd y Vss se aplica la alimentación de +5 V y tierra respectivamente.
3.º El terminal Vo regula el contraste de la pantalla de cristal líquido al aplicarle, mediante un potenciómetro, una tensión variable comprendida entre 0 y +5 V.
4.º Los tres terminales restantes controlan las funciones principales del módulo:
   - *E: Señal de activación. Si E = 0, el módulo está desactivado y no funcionan las restantes señales.*
   - *R/W#: Lectura/Escritura. Si R/W# = 0, se escribe el módulo, y si R/W# = 1, es leído.*
   - *RS: Selección de registro de control (0) o de datos (1). Por D0-D7 se transfiere información de control-comandos o datos de caracteres a escribir.*

Para controlar un módulo LCD con un PIC16F84 se recomienda conectar las líneas de la PUERTAB (RB7-RB0) a las líneas D7-D0. Las líneas de la PUERTAB deben ser bidireccionales y deberán configurarse como Entrada o Salida según el sentido de la información. Las tres líneas de menos peso de la PUERTAA (RA0, RA1 y RA2) se configuran como salidas y se conectan a las líneas RS, R/W# y E, respectivamente (Figura F.30).

El módulo LCD responde a una serie de comandos con los que se pueden gobernar sus distintas opciones de trabajo y que puede enviarle el PIC por las líneas D7-D0 cuando RS = 0 y R/W# = 0, para trabajar en modo escritura. En la Figura F.31 se muestra una tabla con los comandos y los códigos correspondientes.

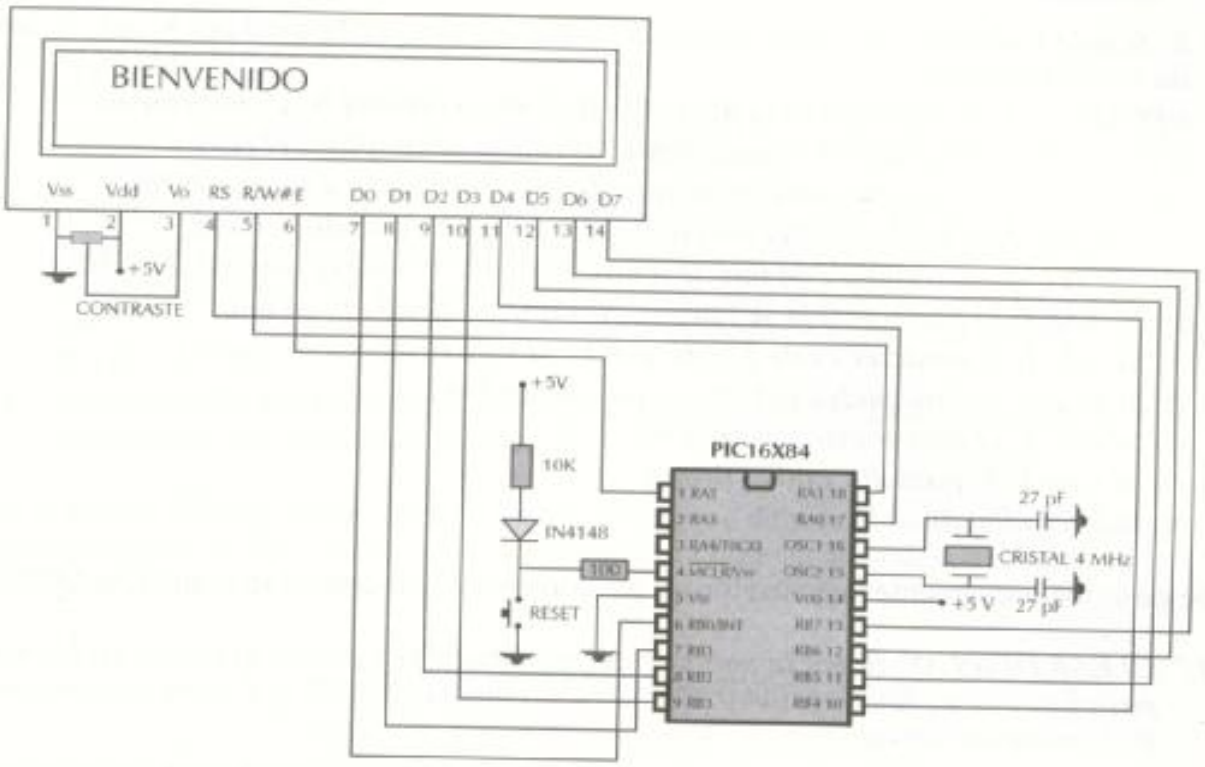


Figura F.30. *Conexionado del módulo LCD a las líneas de E/S del PIC 16F84. Debido al escaso efecto del potenciómetro de contraste, en la práctica no se tiene en cuenta.*

COMANDO	RS	R/W'	E	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Tiempo Ejecución
CLEAR DISPLAY	0	0	1	0	0	0	0	0	0	0	1	1,64 ms
HOME	0	0	1	0	0	0	0	0	0	1	X	1,64 ms
ENTRY MODE SET	0	0	1	0	0	0	0	0	1	I/D	S	40 ms
DISPLAY ON/OFF	0	0	1	0	0	0	0	1	D	C	B	40 ms
CURSOR DISPLAY SHIFT	0	0	1	0	0	0	1	S/C	R/L	X	X	40 ms
FUNCTION SET	0	0	1	0	0	1	DL	N	F	X	X	40 ms
SET CGRAM ADDRESS	0	0	1	0	1	DIRECCIÓN DE LA CGRAM						40 ms
SET DDRAM ADDRESS	0	0	1	1	DIRECCIÓN DE LA DDRAM							40 ms
READ BUSY FLAG $ ADDRESS	0	0	1	BF	DIRECCIÓN DE LA CGRAM O DDRAM							40 ms
WRITE DATA TO CG O DD	0	0	1	CÓDIGO ASCII PARA LA RAM								40 ms
READ DATA TO CG O DD	0	0	1	CÓDIGO ALMACENADO EN RAM								40 ms

Figura F.31. *Códigos de los comandos a los que responde el módulo LCD.*

En la tabla de la Figura F.31 existen diversos bits representados por una abreviatura que, según su valor, seleccionan una opción de trabajo:

- *S: Si vale 1, desplaza la visualización cada vez que se escribe un dato. Si vale 0, funciona en modo normal.*
- *I/D: Si vale 1, se incrementa la dirección del cursor; si vale 0, se decrementa.*
- *S/C: Si vale 1, desplaza la visualización; si vale 0, se desplaza el cursor.*
- *R/L: Si vale 1, el desplazamiento es a la derecha; si vale 0, a la izquierda.*
- *BF: Si vale 1, el módulo LCD está ocupado; si vale 0, está disponible.*
- *DL: Si vale 1, se trabaja con bus de datos de 8 bits; si vale 0, con bus de 4 bits.*
- *N: Si vale 1, la presentación se hace en dos líneas; si vale 0, en una.*
- *F: Si vale 1, el carácter es de 5 × 10 pixels; si vale 0, de 5 × 7.*
- *B: Si vale 1, hay parpadeo del cursor (si está en ON).*
- *C: Si vale 1, el cursor está activo (ON).*
- *D: Si vale 1, la pantalla está activada.*
- *X: Indeterminado.*

Se describe brevemente la actuación de los comandos que controlan al módulo LCD:

1.º *CLEAR DISPLAY: 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 autoincremento de la posición del cursor.*

2.º *HOME: Pone el cursor en la dirección 0. No varía el contenido de la memoria DDRAM que guarda los datos y que queda direccionada desde la posición 0.*

3.º *ENTRY MODE SET: Establece la dirección del movimiento del cursor (I/D); si pone a 1 el bit S, desplaza la visualización cada vez que se escribe un dato. Si S = 0, la presentación es la normal.*

4.º *DISPLAY ON/OFF CONTROL: Activa o desactiva al display (D) y al cursor (C) y determina si éste parpadea o no.*

5.º *CURSOR DISPLAY SHIFT: Mueve el cursor y desplaza la visualización sin cambiar el contenido de la memoria DDRAM.*

6.º *FUNCTION SET: Establece el número de líneas del interfaz con el bus de datos, siendo habitual que sea de 8 bits, con lo que DL = 1. Especifica el número de líneas de caracteres, que para que sean dos se debe poner N = 1, y el formato del carácter, que es de 5 × 7 pixels, si F = 0.*

7.º *SET CGRAM ADDRESS: El módulo LCD tiene definidos los caracteres ASCII, pero, además, permite que el usuario pueda definir un máximo de nueve caracteres nuevos. Éstos se guardan en la memoria CGRAM de 64 bytes. Cada carácter está especificado con un número de bytes comprendido entre 6 y 16, según su complejidad. Dichos bytes se almacenan en las sucesivas posiciones de la CGRAM. Con este comando se indica la dirección de la CGRAM a partir de la cual se irán almacenando los bytes que definen al nuevo carácter. Tras ejecutar este comando, todos los datos que se lean o escriban posteriormente lo hacen desde esa posición de la CGRAM.*

8.º *SET DDRAM ADDRESS: 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 90h-8Fh, y los del segundo desde la C0h a la CFh.*

9.º *READ BUSY FLAG ADDRESS: Se trata de un comando para la lectura del flag BUSY que indica si todavía se está ejecutando un comando previo en el módulo LCD. Además, proporciona la dirección de la CGRAM o DDRAM que se haya utilizado la última vez.*

10.º *WRITE DATA TO CGRAM o DDRAM: Se escriben en la DDRAM los datos (caracteres ASCII) que se quieren visualizar. También se escriben en la CGRAM los bytes de los nuevos caracteres creados por el usuario. Se usa una memoria u otra según haya sido la instrucción de direccionamiento previa, que hará que se refiera a la CGRAM o a la DDRAM.*

11.º *READ DATA TO CGRAM o DDRAM: Igual que el comando anterior, pero en esta ocasión en una operación de lectura.*

En la Figura F.32 se ofrece el juego de caracteres que admite el módulo LCD. Las posiciones marcadas con «CGRAM(n)» están destinadas a contener los nuevos caracteres gráficos definibles por el usuario.



Figura F.32. *Juego de caracteres ASCII que admite el módulo LCD y ocho posiciones marcadas con CGRAM(n) para que el usuario defina nuevos caracteres.*

Cuando se aplica adecuadamente la tensión de alimentación al módulo LCD, se ejecuta automáticamente la siguiente secuencia de inicialización :

1.º Se ejecuta el comando CLEAR DISPLAY borrando la pantalla. El *flag* BUSY se mantiene activado 15 ms hasta que se completa esta secuencia.

2.º Se ejecuta el comando FUNCTION SET, que establece el interfaz con el bus de datos a 8 bits normalmente (DL = 1), y el número de líneas (bit N).

3.º Se ejecuta el comando DISPLAY ON/OFF CONTROL, que hace que el display quede en OFF (D = 0), también el cursor (C = 0) y sin parpadeo (B = 0).

4.º Se ejecuta el comando ENTRY MODE SET, que establece la dirección de movimiento del cursor con autoincremento (I/D = 1) y el modo de visualización normal, sin desplazamiento (S = 0).

Si la conexión de la alimentación no reúne todas 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 15 ms, para completar dicha secuencia de inicialización.

LCD_CXX.ASM

El conjunto de rutinas que se presentan a continuación permiten realizar las tareas básicas de control del módulo de visualización LCD. Se emplean con los PIC 16cxx. En el programa principal se deberá reservar memoria para el bloque de variables que utiliza el LCD del modo:

```
LCD_VAR EQU dir_inicio_del_bloque
```

BLOQUE DE ETIQUETAS

```
#define ENABLE          bsf PORTA,2  ; Activa E
#define DISABLE         bcf PORTA,2  ; Desactiva
#define LEER            bsf PORTA,1  ; Pone LCD en Modo RD
#define ESCRIBIR        bcf PORTA,1  ; Pone LCD en Modo WR
#define OFF_COMANDO     bcf PORTA,0  ; Desactiva RS (modo comando)
#define ON_COMANDO      bsf PORTA,0  : Activa RS (modo datos)

CBLOCK  LCD_VAR
        LCD_TEMP_2  ; Inicio de las variables. Será la primera dirección libre disponible
        LCD_TEMP_1
ENDC
```

RUTINA UP_LCD: Con esta rutina se configura el PIC para que trabaje con el LCD.

```
UP_LCD    bsf     STATUS,RP0    ; Banco 1
          clrf    PORTB         ; RB <0-7> salidas digitales
          clrf    PORTA         ; RA <0-4> salidas digitales
          bcf     STATUS,RP0    ; Banco 0
          OFF_COMANDO           ; RS=0
          DISABLE               ; E=0
          return
```

RUTINA LCD_BUSY: Con esta rutina se chequea el estado del *flag* BUSY del módulo LCD, que indica, cuando está activado, que el módulo aún no ha terminado el comando anterior. La rutina espera a que se complete cualquier comando anterior antes de retornar al programa principal, para poder enviar un nuevo comando.

```
LCD_BUSY  LEER                  ; Pone el LCD en Modo RD
          bsf     STATUS,RP0
          movlw   H'FF'
          movwf   PORTB         ; PUERTAB como entrada
          bcf     STATUS,RP0    ; Selecciona el banco 0
          ENABLE                ; Activa el LCD
          nop

L_BUSY    btfsc   PORTB,7       ; Chequea bit de Busy
          goto    L_BUSY
          DISABLE               ; Desactiva LCD
          bsf     STATUS,RP0
          clrf    PORTB         ; PUERTAB salida
          bcf     STATUS,RP0
          ESCRIBIR              ; Pone LCD en modo WR
          return
```

RUTINA LCD_E: Se trata de una pequeña rutina que se encarga de generar un impulso de 1µs (para una frecuencia de funcionamiento de 4 Mhz) por la patita de salida de la PUERTAA RA2, que se halla conectada a la señal E (Enable) del módulo LCD. Con esta rutina se pretende activar al módulo LCD.

```
LCD_E     ENABLE                ; Activa E
          nop
          DISABLE               ; Desactiva E
          return
```

RUTINA LCD_DATO: Es una rutina que pasa el contenido cargado en el registro W, el cual contiene un carácter ASCII, a la PUERTAB, para visualizarlo por el LCD o escribirlo en la CGRAM.

```
LCD_DATO  OFF_COMANDO                ; Desactiva RS (modo comando)
          movwf     PORTB            ; Valor ASCII a sacar por PORTB
          call      LCD_BUSY         ; Espera a que se libere el LCD
          ON_COMANDO                 ; Activa RS (modo dato)
          call      LCD_E            ; Genera pulso de E
          return
```

RUTINA LCD_REG: Rutina parecida a la anterior, pero el contenido de W ahora es el código de un comando para el LCD, que es necesario pasar también a la PUERTAB para su ejecución.

```
LCD_REG   OFF_COMANDO                ; Desactiva RS (modo comando)
          movwf     PORTB            ; Código de comando
          call      LCD_BUSY         ; ¿LCD libre?
          call      LCD_E            ; Sí. Genera pulso de E
          return
```

RUTINA LCD_INI: Esta rutina se encarga de realizar la secuencia de inicialización del módulo LCD de acuerdo con los tiempos dados por el fabricante (15 ms). Se especifican los valores de DL, N y F, así como la configuración de un interfaz de ocho líneas con el bus de datos del PIC, y dos líneas de 16 caracteres de 5 × 7 pixels.

```
LCD_INI   movlw     b'00111000'
          call      LCD_REG          ; Código de instrucción
          call      LCD_DELAY        ; Temporiza
          movlw     b'00111000'
          call      LCD_REG          ; Código de instrucción
          call      LCD_DELAY        ; Temporiza
          movlw     b'00111000'
          call      LCD_REG          ; Código de instrucción
          call      LCD_DELAY        ; Temporiza
          return
```

RUTINA BORRA_Y_HOME: Borra el display y retorna el cursor a la posición 0.

```
BORRA_Y_HOME  movlw   b'00000001'    ; Borra LCD y Home.
              call    LCD_REG
              return
```

RUTINA DISPLAY_ON_CUR_OFF: Control del display y cursor. Activa el display y desactiva el cursor.

```
DISPLAY_ON_CUR_OFF    movlw b'00001100'      ;LCD on, cursor off.
                      call  LCD_REG
                      return
```

RUTINA LCD_DELAY: Se trata de un rutina que implementa un retardo o temporización de 5 ms. Utiliza dos variables, llamadas LCD_TEMP_1 y LCD_TEMP_2, que se van decrementando hasta alcanzar dicho tiempo.

```
LCD_DELAY     clrwdt
              movlw   10
              movwf   LCD_TEMP_1
              clrf    LCD_TEMP_2

LCD_DELAY_1   decfsz  LCD_TEMP_2,F
              goto    LCD_DELAY_1
              decfsz  LCD_TEMP_1,F
              goto    LCD_DELAY_1
              return
```

Enunciado y objetivos

Con ayuda de las librerías LCD_CXX.ASM y P16CXX.INC realizar un programa en el que aparezca un mensaje de bienvenida sobre el LCD.

Materiales específicos para el Proyecto Típico 2

1.º Pantalla LCD

Organigrama



Figura F.33. *Organigrama del proyecto.*

Esquema eléctrico

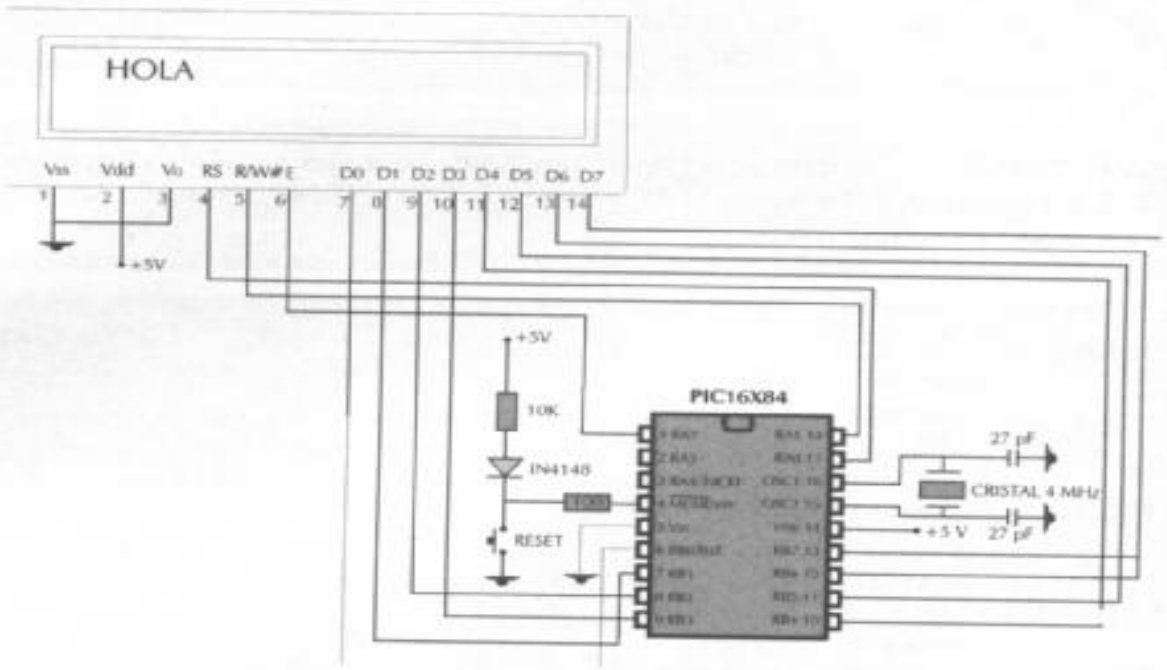


Figura F.34. *Esquema de conexionado.*

Esquema del montaje

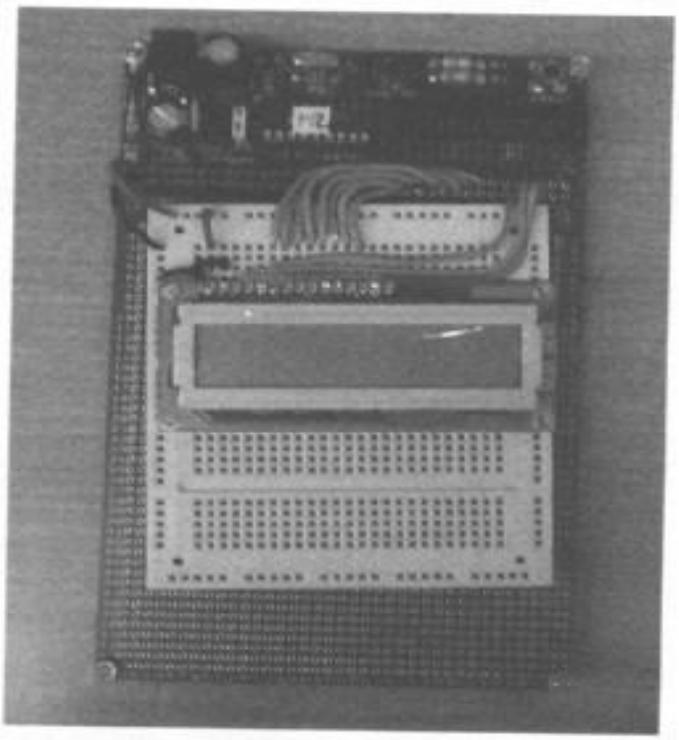


Figura F.35. *Fotografía del montaje sobre la placa PIC18-ME.*

El programa comentado

```
;*****************************************************************
;                    PROYECTO TÍPICO 2
;              «MENSAJE DE BIENVENIDA SOBRE LCD»
;*****************************************************************
;  Programa que muestra la palabra «HOLA» en la línea 1 del LCD.
;*****************************************************************
;  DIRECTIVAS DEL PROGRAMA
;*****************************************************************
        LIST    p=16F84         ; Se emplea el PIC 16F84
        RADIX   HEX             ; Sistema de numeración hexadecimal
;*****************************************************************
;  ETIQUETAS DEL PROGRAMA
;*****************************************************************
        INCLUDE "PIC16CXX.INC" ; Etiquetas genéricas para los PIC 16CXX

RESUL   EQU 0x0C
TEMPO1  EQU 0x0D
OFFSET  EQU 0x0E
OPCION  EQU 0x81^80H
LCD_VAR EQU 0x0F
;*****************************************************************
;  PROGRAMA PRINCIPAL
;*****************************************************************
        ORG     0       ; El programa comienza en la dirección 0 y salta a la
        goto    INICIO  ; dirección 5 para sobrepasar el vector de interrupción
        ORG     5

        INCLUDE "LCD_CXX.ASM"

INICIO  call    UP_LCD          ; Configura el PIC para el uso del LCD
        bsf     STATUS,RP0      ; Selección del banco 1
        movlw   b'10000111'
        movwf   OPCION          ; Asigna preescaler al TMR0
        bcf     STATUS,RP0
        call    LCD_INI         ; Inicializa el LCD

INICIO_1 call   BORRA_Y_HOME
        call    DISPLAY_ON_CUR_OFF
        movlw   0x80                    ; Primera posición de primera fila
```

```
        call    LCD_REG
        movlw   b'00000110'     ; Entry Mode Set. Inc de cursor
        call    LCD_REG
        clrf    OFFSET          ; Dirección del mensaje
        call    MENS

BUCLE   goto    BUCLE           ; El programa acaba en un bucle infinito
```

RUTINA MENS: Esta rutina saca mensajes por la pantalla LCD, siendo el primer carácter aquel que está en la posición apuntada por OFFSET, y debiendo ser el último 0x00.

```
MENS    clrf    RESUL           ; Inicializa contador de caracteres

MENS_1  movf    OFFSET,W        ; Coge el desplazamiento
        addwf   RESUL,W         ; Calcula el carácter a coger
        call    DATOS           ; Coge el carácter
        iorlw   0               ; Actualiza flags
        btfsc   STATUS,2        ; Comprueba si es el último
        return                  ; Si es el último, vuelve
        call    LCD_DATO        ; Visualiza el carácter
        incf    RESUL,F         ; Incrementa contador de caracteres
        movlw   d'10'
        movwf   TEMPO1
        goto    MENS_1
```

TABLA CON LOS DATOS A SACAR POR EL LCD

```
DATOS   addwf   PCL,1
        retlw   'H'             ; OFFSET = 0x00
        retlw   'O'
        retlw   'L'
        retlw   'A'
        retlw   0x00
```

```
        END
```

Propuesta de un nuevo proyecto

En el programa anterior se ha probado una forma típica de sacar mensajes, pero esto depende del código que se envíe a LCD_REG, según las variantes de la tabla de la Figura F.31. Ahora se trata de probar otras combinaciones y meterlas como rutinas independientes (MENSAJE_PARPADEO, MENSAJE_PRIMERA_FILA,...) en el fichero LCD_CXX.ASM.

Se puede intentar, además, crear caracteres nuevos y almacenarlos en la CGRAM, para mostrarlo con una nueva rutina MOSTRAR_CGRAM.

PROYECTO TÍPICO 3

Enunciado y objetivos

Mediante las librerías TECLADO.ASM, LCD_CXX.ASM y P16CXX.INC, realizar un programa que visualice sobre el LCD las teclas que se vayan pulsando. Cada pulsación irá acompañada de un «beep» de 0,25 segundos sobre el zumbador piezoeléctrico conectado a RA3.

Materiales específicos para el Proyecto Típico 3

1.º Teclado de 16 teclas hexadecimales (0-F).
2.º Cuatro resistencias de 2K2Ω y $^1/_4$ W.
3.º Zumbador miniatura C.I. P = 7,5.
4.º Pantalla LCD.

Foto del montaje

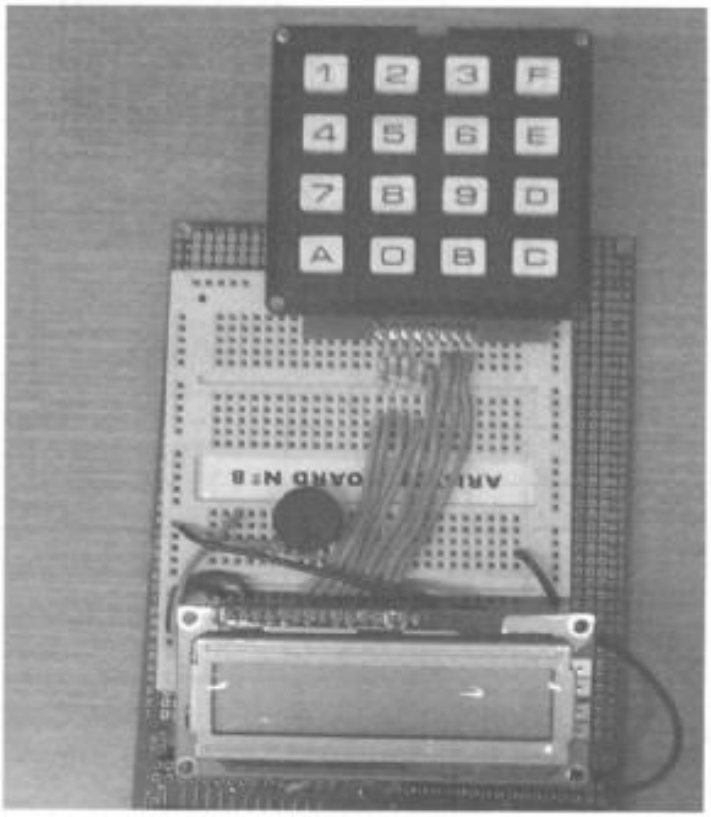


Figura F.36. *Fotografía del montaje sobre la placa PIC18-ME.*

Esquema eléctrico

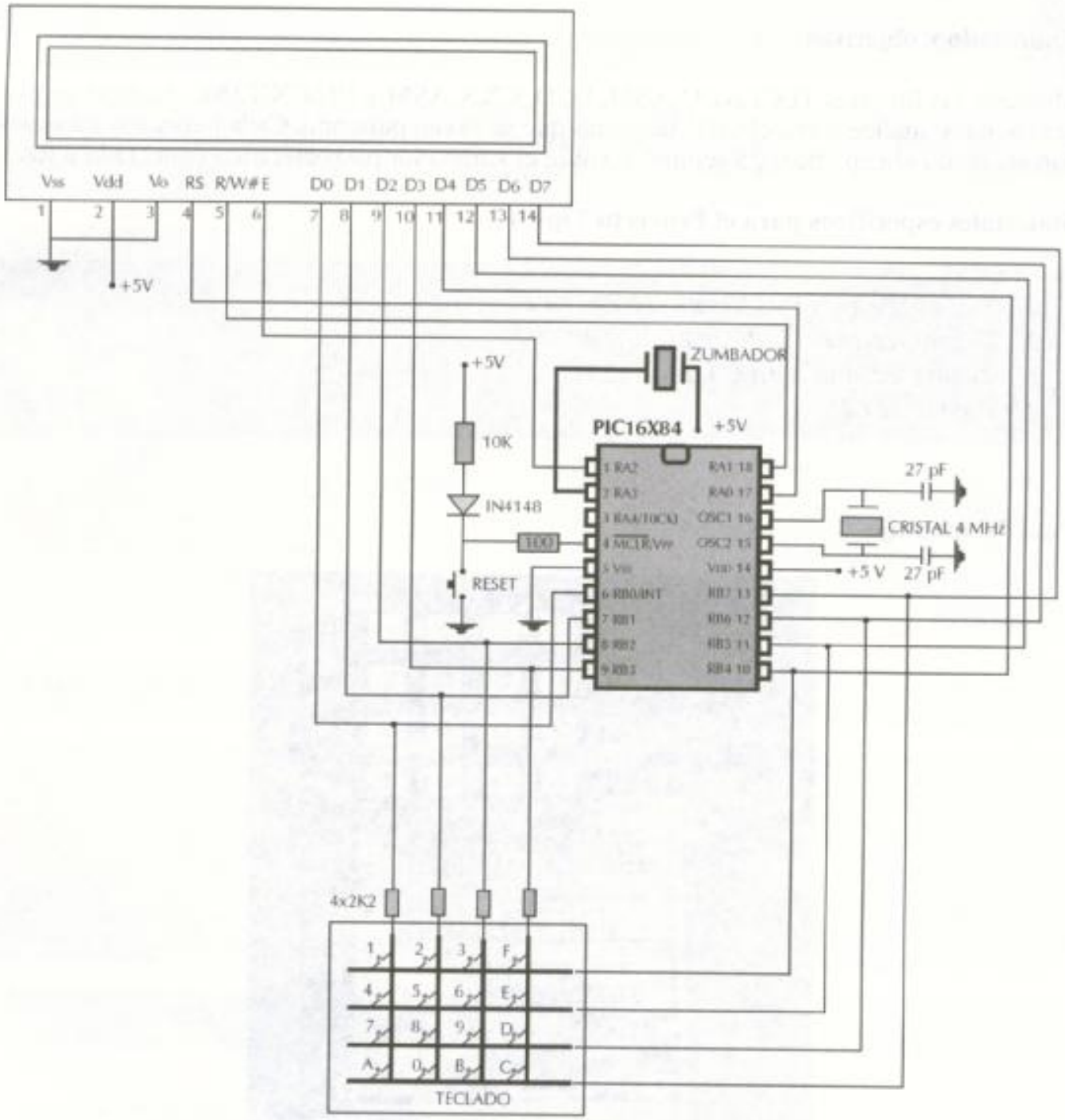


Figura F.37. *Esquema de conexionado del teclado, LCD y zumbador.*

Organigrama

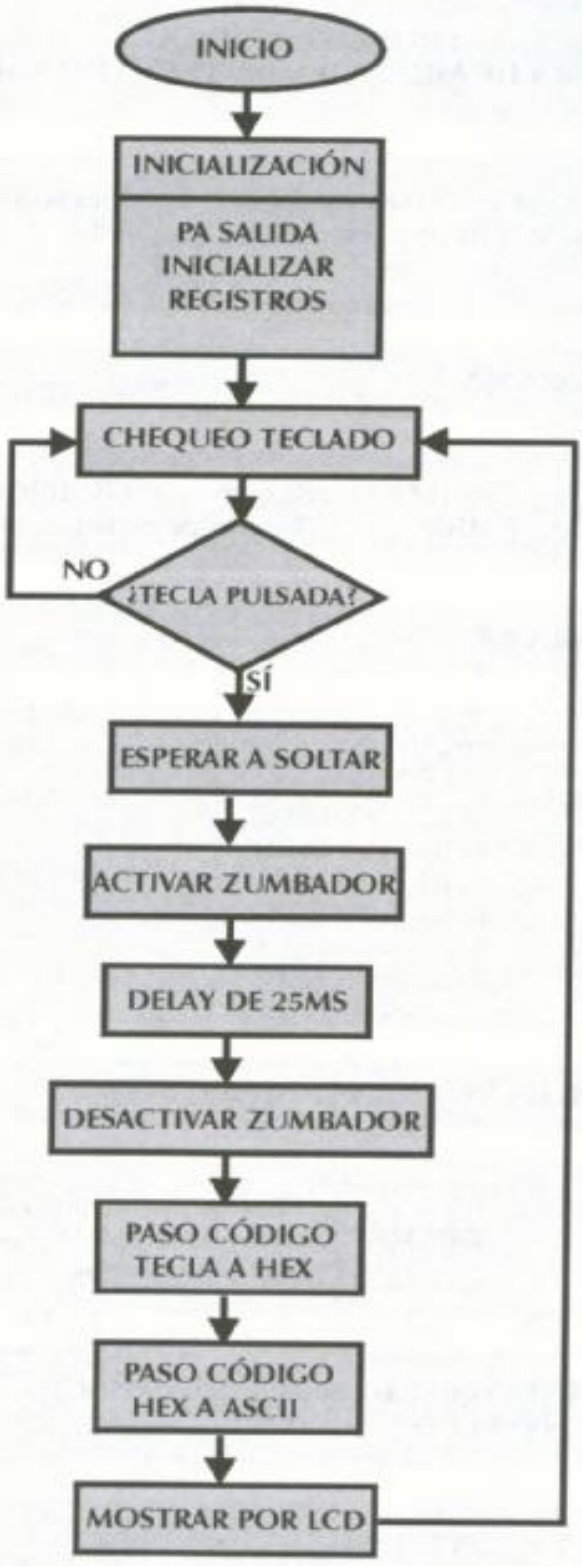


Figura F.38. *Organigrama del proyecto.*

El programa comentado

PROYECTO TÍPICO 3
«ESCRITURA DE MENSAJES DE TECLADO SOBRE LCD»

Se desea realizar un programa que visualice sobre el LCD las teclas que se vayan pulsando. Cada pulsación irá acompañada de un «beep» de 0,25 segundos sobre el zumbador piezoeléctrico conectado a RA3.

DIRECTIVAS DEL PROGRAMA

```
        LIST    P=16F84   ; Se emplea el PIC 16F84
        RADIX   HEX       ; Sistema de numeración hexadecimal
```

ETIQUETAS DEL PROGRAMA

```
        INCLUDE "PIC16CXX.INC"

TEMP_1  EQU     0x0C
DELAY_1 EQU     0x0D
DELAY_2 EQU     0x0E
DELAY_3 EQU     0x0F
LCD_VAR EQU     0x10
KEY_VAR EQU     0x12
```

POSICIONAMIENTO DE LA RUTINA DE INICIO

```
        ORG     0         ; El programa comienza en la dirección 0 y
        goto    INICIO    ; salta a la dirección 5 para sobrepasar el
        ORG     5         ; vector de interrupción
```

RUTINA HEX_ASCII: Rutina que pasa el valor HEXADECIMAL que contiene W a valor ASCII, para ser mostrado por el LCD.

```
HEX_ASCII   addwf   PCL,1
            retlw   '0'
```

```
        retlw '1'
        retlw '2'
        retlw '3'
        retlw '4'
        retlw '5'
        retlw '6'
        retlw '7'
        retlw '8'
        retlw '9'
        retlw 'A'
        retlw 'B'
        retlw 'C'
        retlw 'D'
        retlw 'E'
        retlw 'F'
```

PROGRAMA PRINCIPAL

```
        INCLUDE "LCD_CXX.ASM"
        INCLUDE "TECLADO.ASM"

INICIO  call    UP_LCD
        call    LCD_INI
        call    DISPLAY_ON_CUR_OFF
        bsf     STATUS,RP0          ; Selección del banco 1
        clrf    PORTA               ; PUERTAA como salida
        movlw   b'10000111'
        movwf   OPTION_REG
        bcf     STATUS,RP0
        bsf     PORTA,3             ; Desactivación del zumbador

NO_HAY  call    KEY_SCAN            ; Se hace un chequeo del teclado
        movlw   0x80
        subwf   TECLA,W             ; Sobre TECLA quedará la tecla pulsada o 0x80
        btfsc   STATUS,Z
        goto    NO_HAY              ; No se ha pulsado ninguna tecla
        call    KEY_OFF             ; Sí se ha pulsado. Se espera a que se suelte
        bcf     PORTA,3             ; Se genera un "beep"
        movlw   d'25'
        movwf   TEMP_1
        call    DELAY10             ; Se crea una temporización de 25 ms
        bsf     PORTA,3             ; Se desactiva zumbador
        call    KEY_HEX             ; Se pasa a valor hexadecimal
        call    HEX_ASCII           ; Se pasa el valor hexadecimal a código ASCII
        movwf   TECLA               ; Se guarda temporalmente en TECLA
        call    UP_LCD
        call    BORRA_Y_HOME
        call    DISPLAY_ON_CUR_OFF
        movlw   0x81                ; Primer carácter de primera línea
```

```
        call    LCD_REG
        movf    TECLA,W
        call    LCD_DATO        ; Se saca por el LCD
        goto    NO_HAY          ; Se vuelve a empezar
```

RUTINA DE DELAY DE 10 MS: Esta rutina utiliza el TMR0. Es una temporización de 10 ms que se repite tantas veces como indique la variable TEMPO1.

```
DELAY10     bcf     INTCON,2    ; Borra flag del TMR0
            movlw   0xD8
            movwf   TMR0        ; Carga el TMR0 con valor 39

DELAY10_1   btfss   INTCON,2    ; Espera rebosamiento del TMR0
            goto    DELAY10_1
            decfsz  TEMP_1,F
            goto    DELAY10
            return
```

RUTINA DE DELAY DE UN SEGUNDO: Se basa en la rutina de 10 ms, repitiéndola 100 veces.

```
DELAY1S     movlw   0x64
            movwf   TEMP_1
            call    DELAY10
            return
```

RUTINA KEY_OFF: Esta rutina espera que la tecla recién pulsada sea soltada. Debe usarse justo después de llamar a la rutina KEY_SCAN.

```
KEY_OFF         movf    TECLA,W
                movwf   TEMP_1      ; Guarda temporalmente la tecla

KEY_OFF_NO      call    KEY_SCAN    ; Mira si se ha soltado
                movlw   0x80
                subwf   TECLA,W
                btfss   STATUS,Z
                goto    KEY_OFF_NO  ; Bucle mientras no se suelte
                movf    TEMP_1,W    ; Ya se ha soltado
                movwf   TECLA       ; Se repone la tecla
                return
```

RUTINA DE DELAY VARIABLE: Esta rutina realiza una temporización que depende del valor del acumulador en el momento en que se le llama.

```
DELAY_V         movwf   DELAY_1
                clrf    DELAY_3
                clrf    DELAY_2

DELAY_LOOP      decfsz  DELAY_3,F
                goto    DELAY_LOOP
                decfsz  DELAY_2,F
                goto    DELAY_LOOP
                decfsz  DELAY_1,F
                goto    DELAY_LOOP
                return

                END
```

Propuesta de un nuevo proyecto

En el programa anterior, cuando se mostraba una tecla en el LCD, se hacía siempre en la misma posición, esto es, una tecla se sobrescribía «encima» de otra. Se trata de modificar ahora el programa para que las teclas vayan saliendo seguidas (como en una calculadora). Cuidado, porque la línea no es infinita, y llegará un momento en que habrá que borrarla o pasar a la de abajo.

TERCERA PARTE: COLECCIÓN DE PROYECTOS PROPUESTOS

OBJETIVOS

Se presenta en este tema una colección de proyectos que se pueden programar en el PIC16F84 y comprobar con el sistema de desarrollo Micro'PIC Trainer. Una vez que se asegure su buen funcionamiento se recomienda montarlos en la Tarjeta de Prototipos con la ayuda de la protoboard y de los materiales ya usados en los Proyectos Básicos y Típicos anteriores.

Estos proyectos son de diferente estilo y complejidad, y su finalidad es la de conseguir que se alcance una experiencia profesional en el diseño de proyectos basados en microcontroladores. El lector puede a su vez plantearse sus propios proyectos y, por qué no, llevarlos a la práctica en un entorno real.

EJERCICIO COMBINACIONAL DE MANEJO DE E/S DIGITALES

Montar el hardware necesario y confeccionar el programa correspondiente para controlar dos diodos LED, L0 y L1 desde el interruptor I0, de forma que L0 refleja el nivel lógico que introduce I0 y L1 dicho estado invertido (Figura F.39).

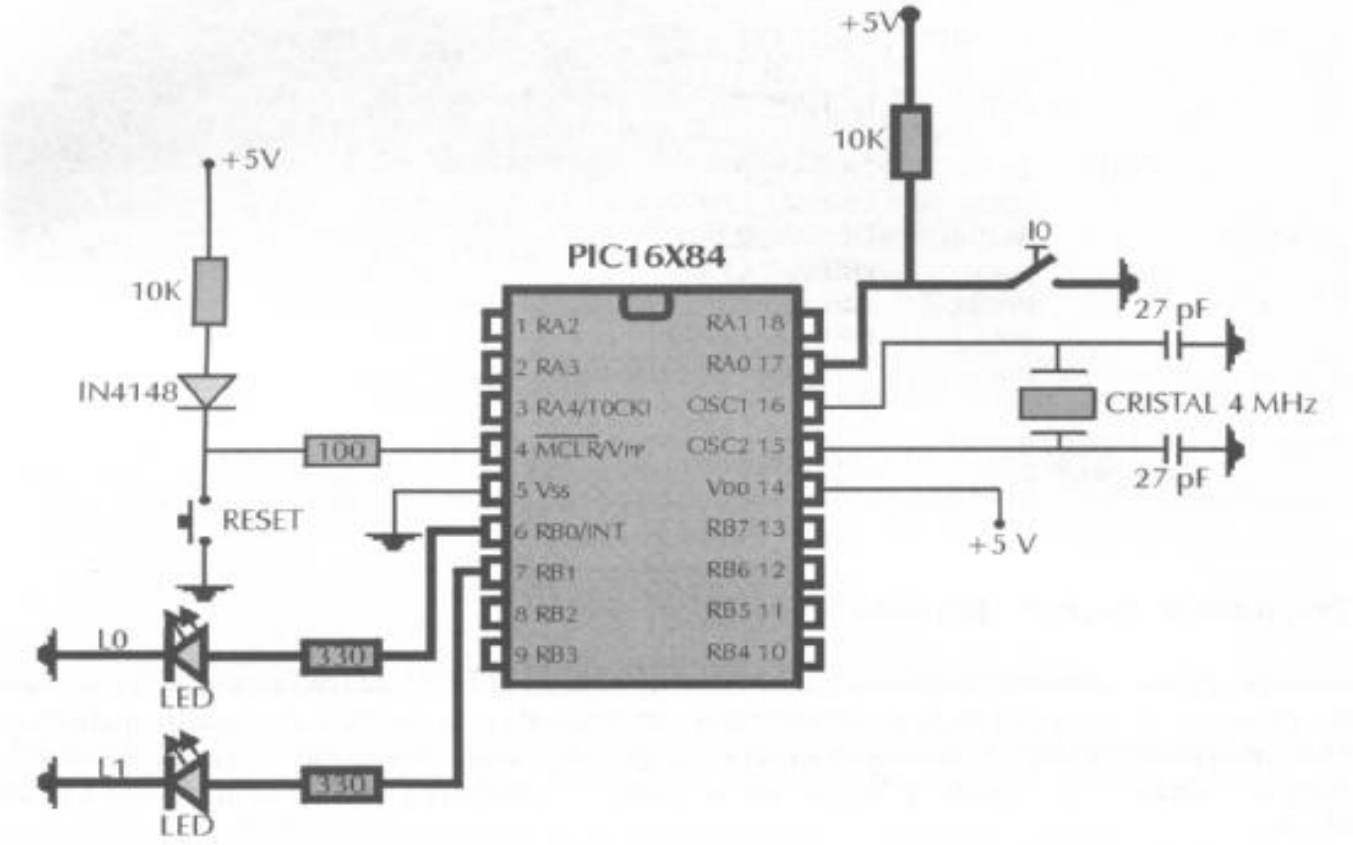


Figura F.39. *Esquema simplificado de los periféricos.*

OTRO EJERCICIO COMBINACIONAL

Mediante dos interruptores, I0 e I1, controlar el estado de ocho diodos LED, de los que dispone una barra DIL, según la siguiente tabla de verdad y esquema de conexionado.

I1	I2	L7	L6	L5	L4	L3	L2	L1	L0
0	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	0	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0	0

Figura F.40. *Tabla de verdad.*

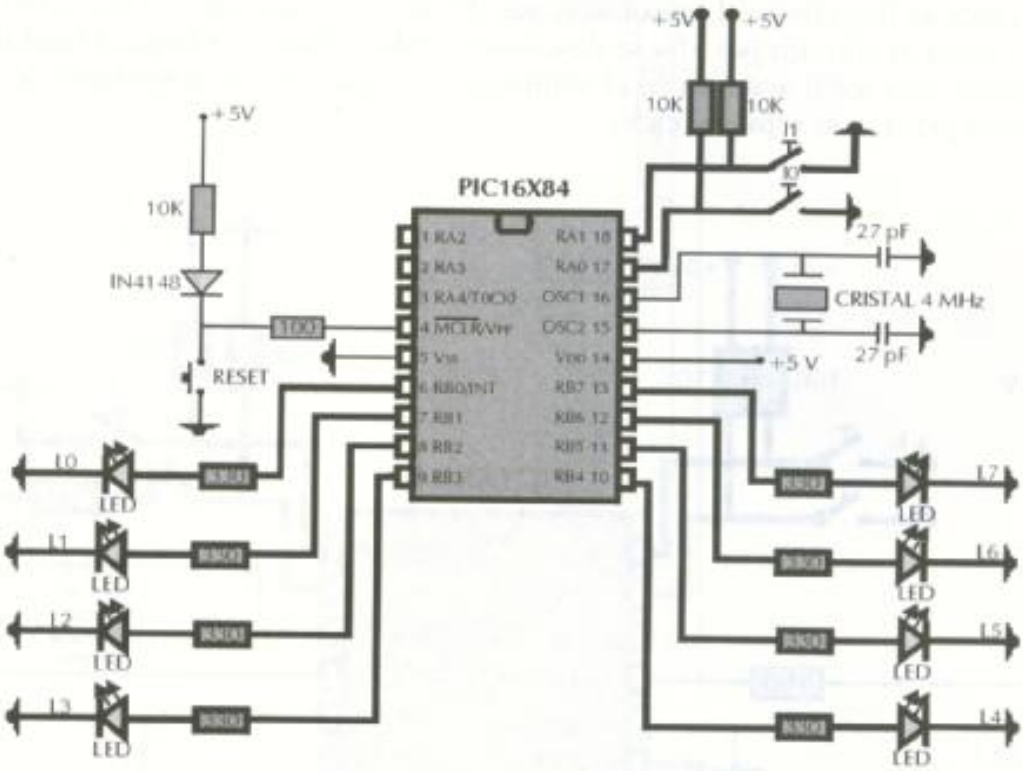


Figura F.41. *Conexión de los periféricos.*

CONTROL DE UN TORNO

La secuencia de funcionamiento del torno se inicia cuando se pulsa «I», que activa el cilindro «V», el cual, en su avance, empuja la pieza a tornear hasta que acciona el sensor «b», que pone en funcionamiento el motor «M» (Figura F.42).

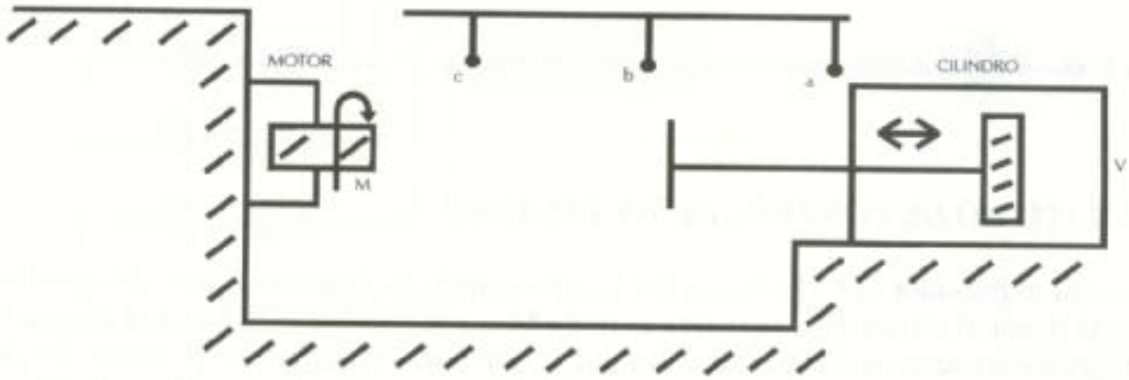


Figura F.42. *Boceto del proyecto.*

En el avance se llega hasta el sensor «c», que detiene el avance de V, e inicia su retroceso. Al volver a pasar el cilindro por «b» se desconecta «M». Cuando se llega al final de carrera «a», se produce una señal acústica en el zumbador «A» para avisar al operario de que coloque una nueva pieza, y se repite el ciclo.

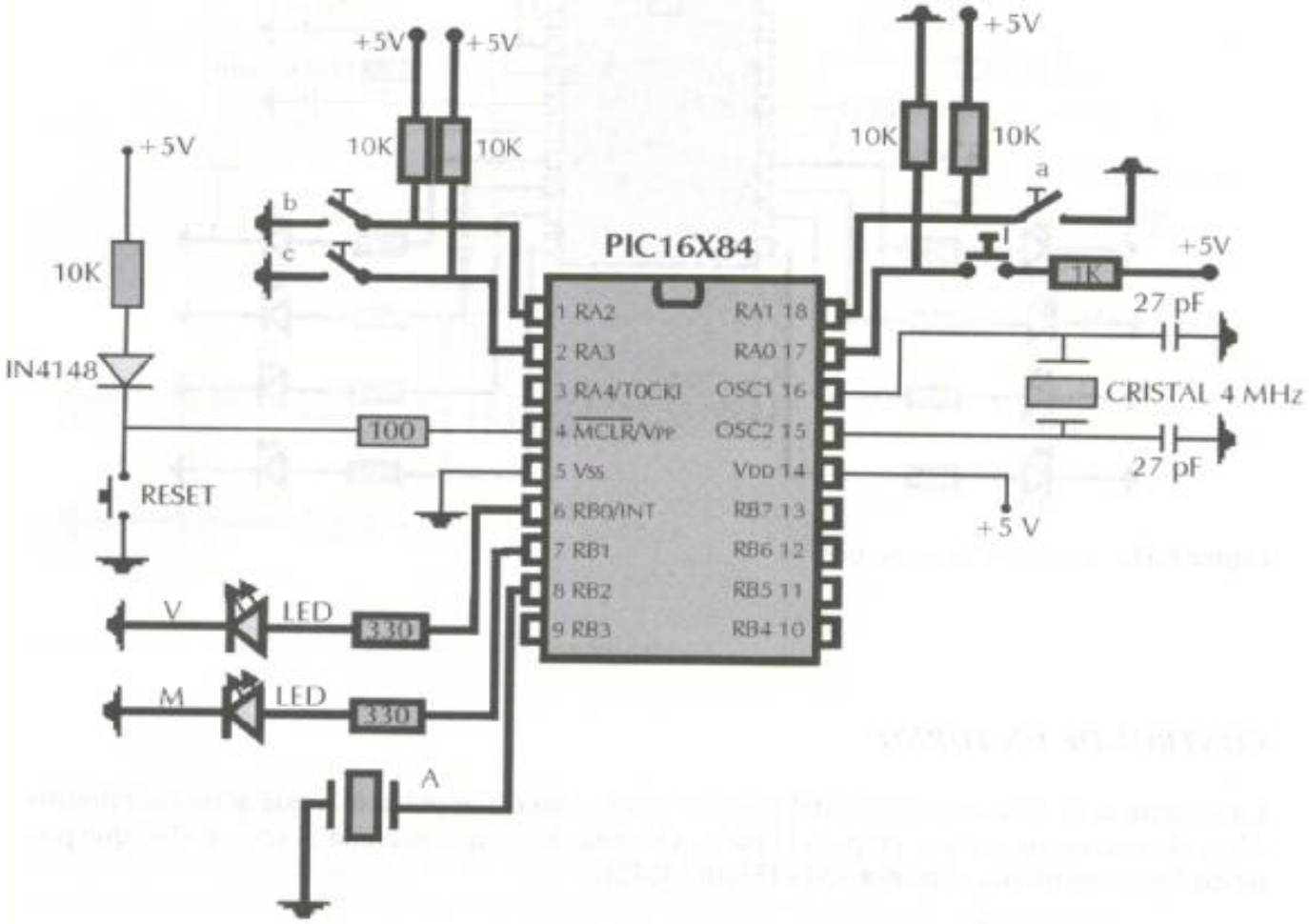


Figura F.43. *Esquema de funcionamiento del torno y conexión de periféricos al PIC16F84.*

AUTOMATISMO DE CONTROL DE UN TALADRO

Al accionar el pulsador «I», el cabezal del taladro realiza un descenso rápido de aproximación «BR». Al llegar al sensor «b», se activa el relé «M», que hace girar la broca al mismo tiempo que se ejecuta un descenso lento «BL» durante el taladrado. Al activarse el sensor «c», se considera el final del taladrado de la pieza y se inicia una subida rápida «SR» del cabezal, siguiendo activado «M». Cuando se alcanza el sensor «a», finaliza la subida rápida «SR», se para «M» y se activa una señal acústica «A» para avisar al operario del final de la operación.

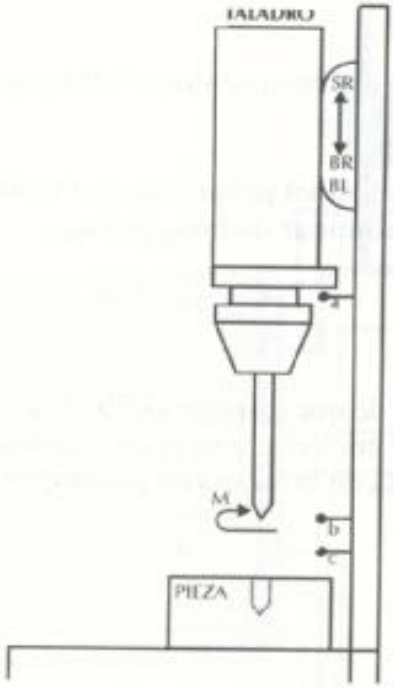


Figura F.44. *Boceto del taladro.*

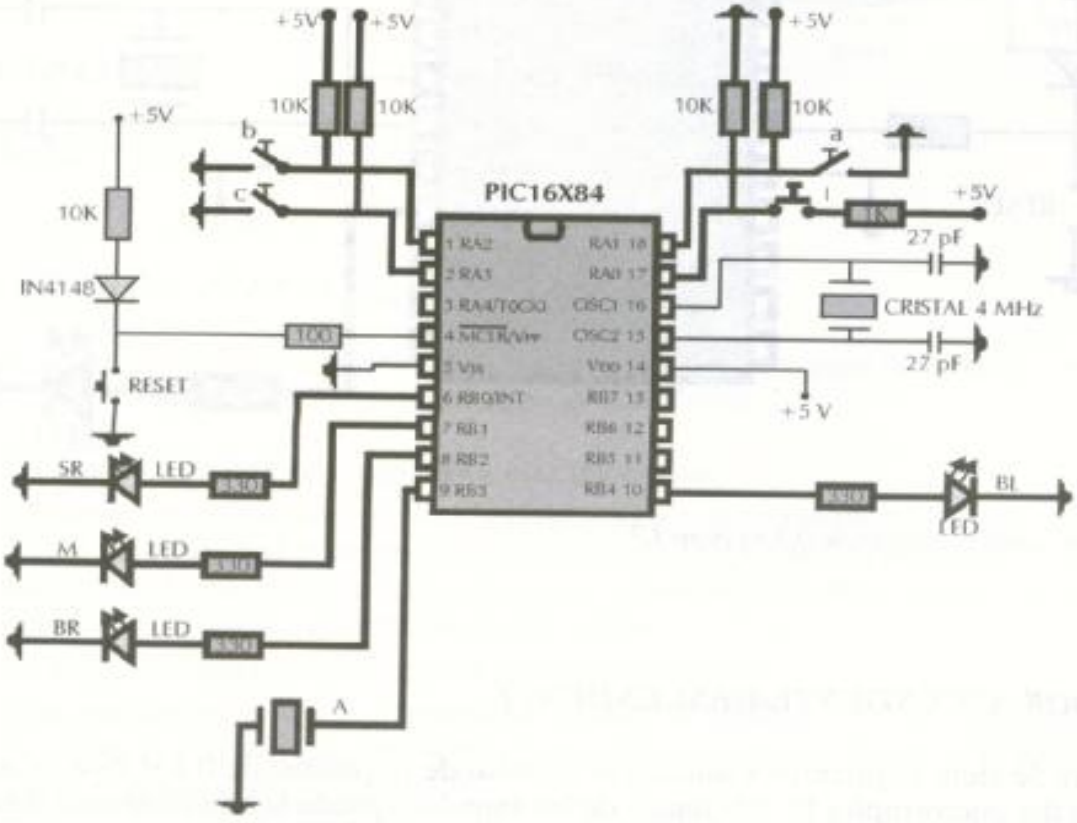


Figura F.45. *Esquema de funcionamiento del taladro y conexión de periféricos al PIC.*

SISTEMA ANTIRREBOTES

Los periféricos de entrada de tipo electromecánico (interruptores, pulsadores, etc.) producen «rebotes» de efectos indeseables.

Colocar en los lugares apropiados del programa que controla el taladro del ejercicio anterior unos retardos de 20 ms para eliminar dichos rebotes.

MANEJO DEL TMR0

Utilizando el temporizador TMR0 de que dispone el PIC16F84, junto a su divisor de frecuencia, conseguir que el diodo LED L7 se encienda y se apague con una intermitencia de 0,5 segundos. Se supone que el PIC que se inserta en la tarjeta de prototipos PIC18-ME funciona a 4MHz.

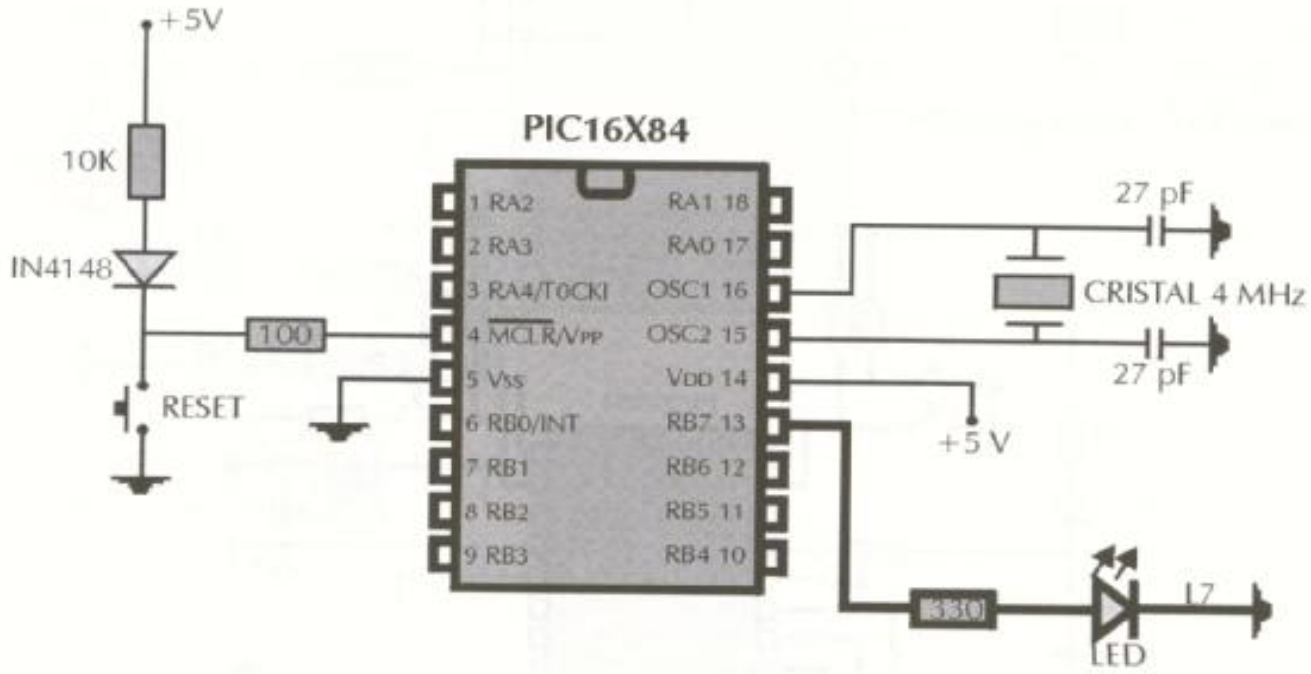


Figura F.46. *Intermitencia de 0,5 sg para L7.*

CONTADOR ASCENDENTE/DESCENDENTE

Un display de siete segmentos visualiza el número de impulsos, de 0 a 9, que se introducen por medio del microrruptor I0. El contaje de los impulsos puede ser ascendente o descendente, según el estado que introduce el microrruptor I1. Si I1=0, el contaje es ascendente; en caso contrario es descendente. Colocar retardos antirrebotes en la exploración de los interruptores.

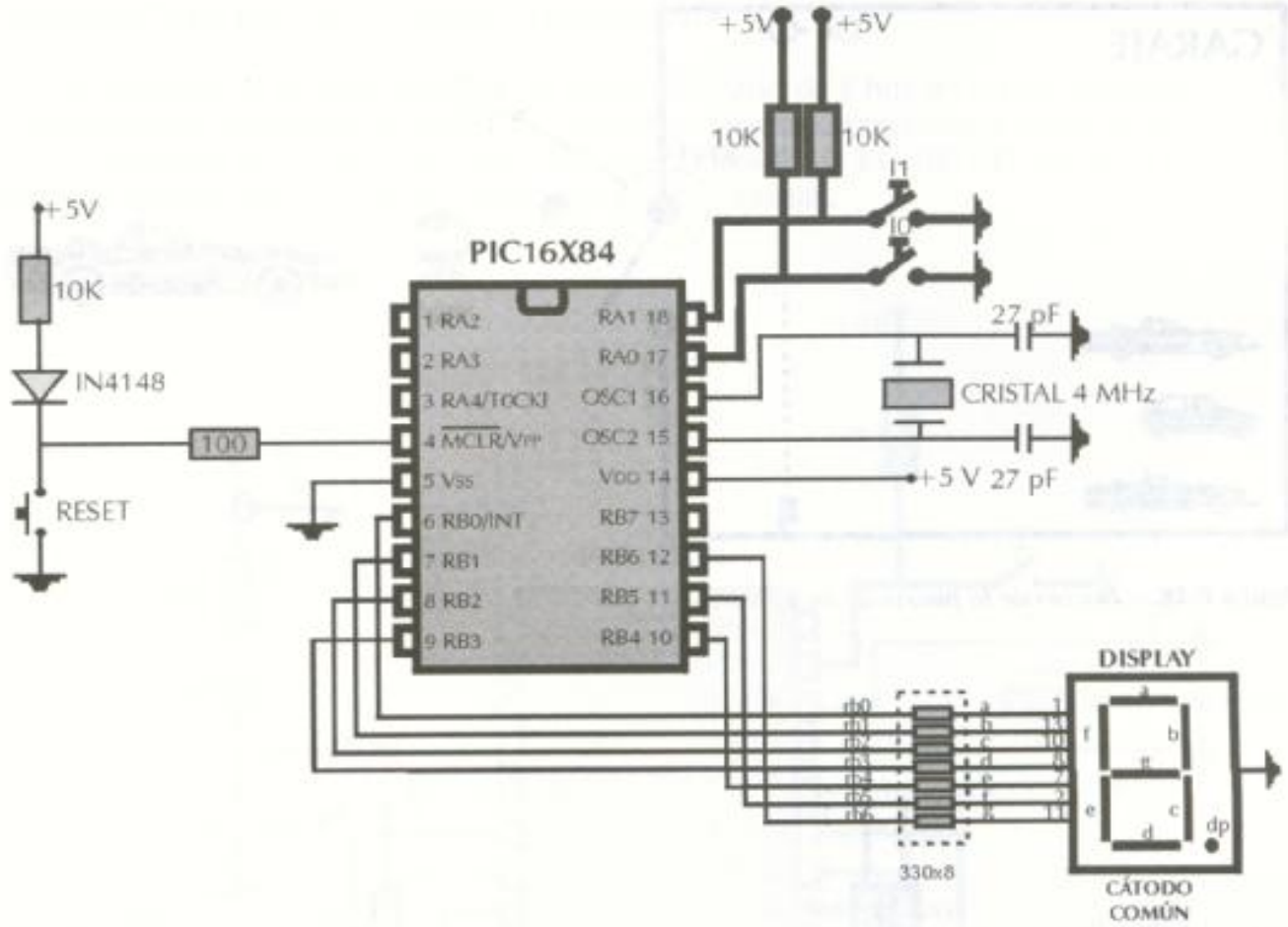


Figura F.47. *Contador ascendente/descendente.*

CONTROL DE LA PUERTA DE UN GARAJE

Se desea controlar automáticamente la secuencia de apertura para la entrada y salida de vehículos en un garaje, que tiene las siguientes características:

1.º En reposo, el semáforo rojo «SR» está encendido.

2.º Al entrar un vehículo se acciona la llave «AP» y se activa un relé «MA», simulado mediante un LED, que pone en marcha el motor de apertura de la puerta. Cuando se abre totalmente la puerta y se activa el sensor «PA», se desactiva «MA», se ilumina el semáforo verde «SV», y se apaga el rojo «SR».

3.º La situación anterior se mantiene durante 5 segundos.

4.º Tras ese tiempo, se apaga el semáforo verde y se enciende el rojo, al mismo tiempo que se activa el relé «MC», que pone en marcha el motor de cierre de la puerta, hasta retornar a la situación de reposo.

5.º Si en la fase de cierre hay algún vehículo en la entrada (sensor «DP»), se inicia una nueva secuencia de apertura (Figura F.48).

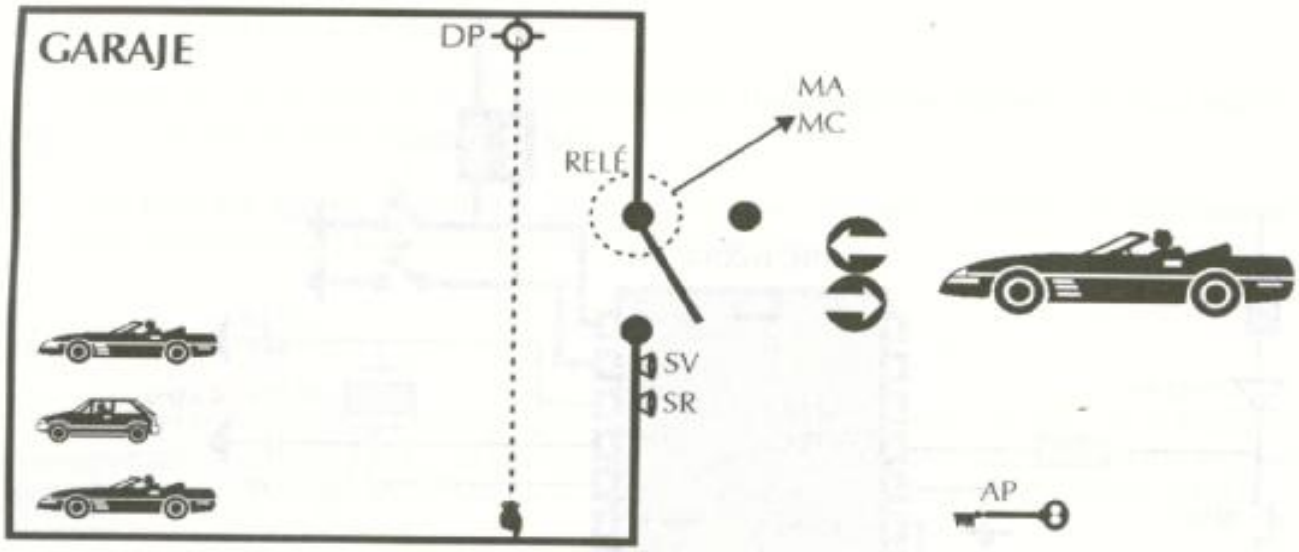


Figura F.48. *Boceto de la puerta de un garaje.*

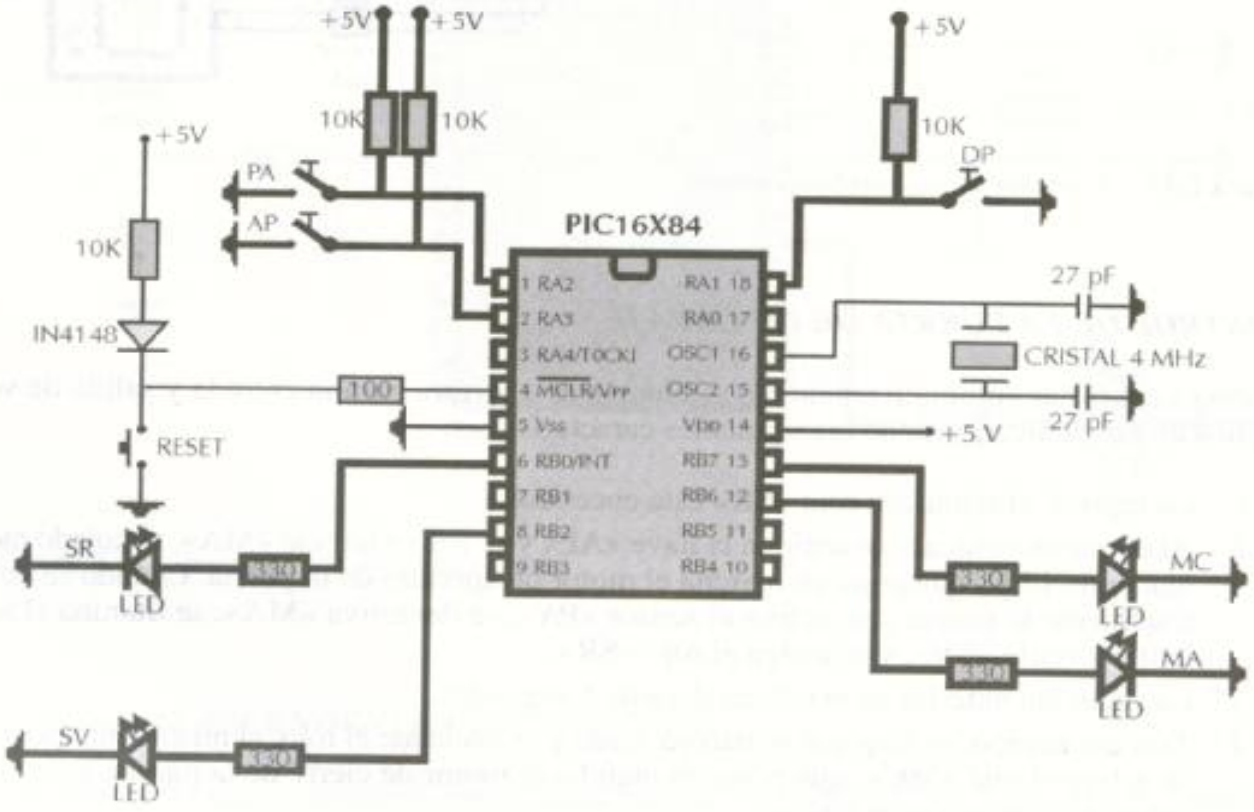


Figura F.49. *Control de la puerta de un garaje.*

GENERACIÓN DE UN NÚMERO ALEATORIO

En este ejercicio se propone generar un número binario de 8 bits de forma aleatoria (00-FF). El número debe generarse al aplicar un impulso (cierre/apertura) con el interruptor I0, y quedará visualizado en los ocho diodos LED conectados en la PUERTAB. La visualización se mantiene durante tres segundos y luego los LED se apagan.

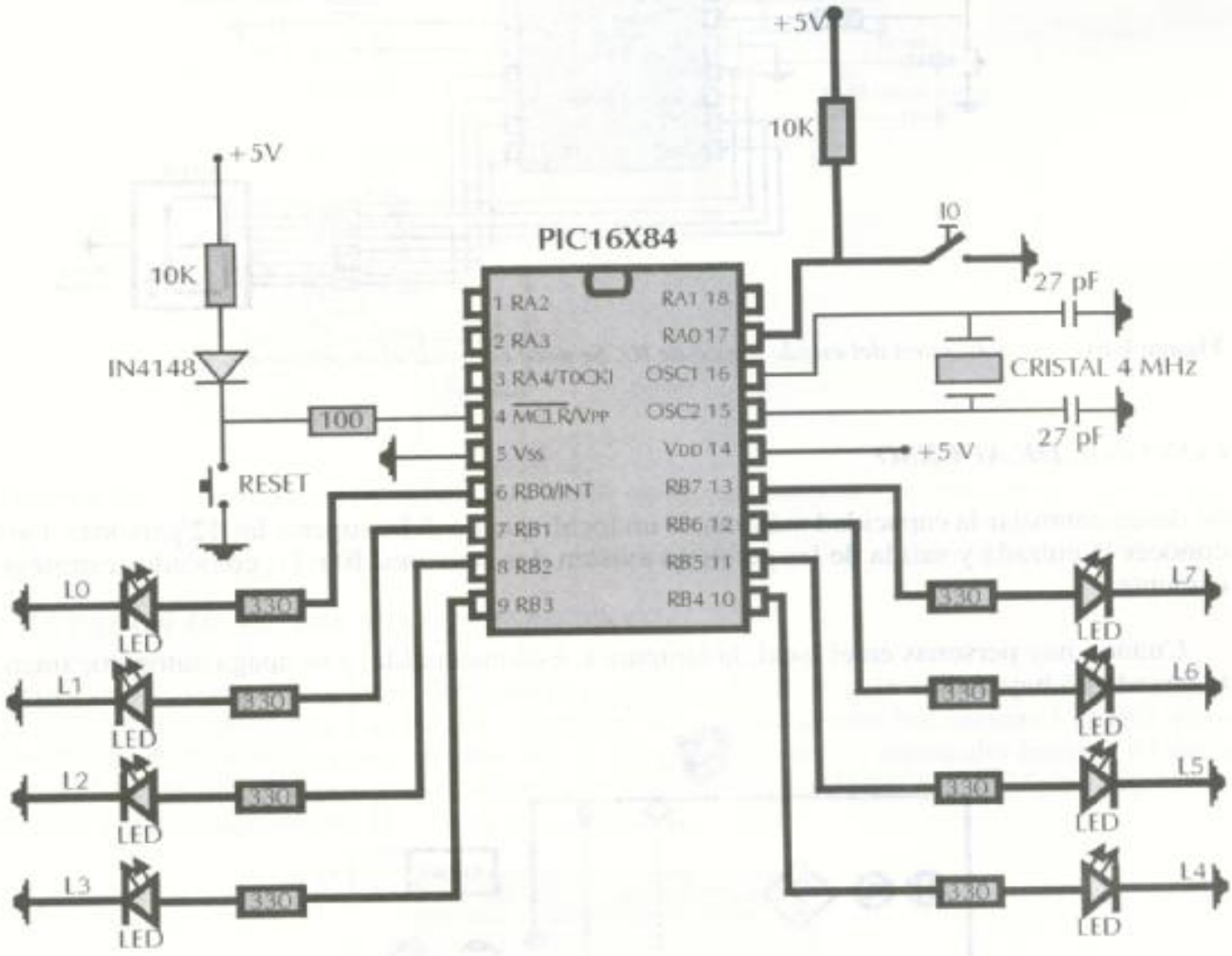


Figura F.50. *Cuando se produce un impulso en IO se genera un número binario aleatorio.*

PRESENTACIÓN DE ESTADOS LÓGICOS

Sobre el display de siete segmentos conectado en la PUERTAB, se visualiza el estado lógico del interruptor I0, mediante un 0 o un 1. El interruptor I1 enciende o no el punto decimal del display según esté cerrado o abierto.

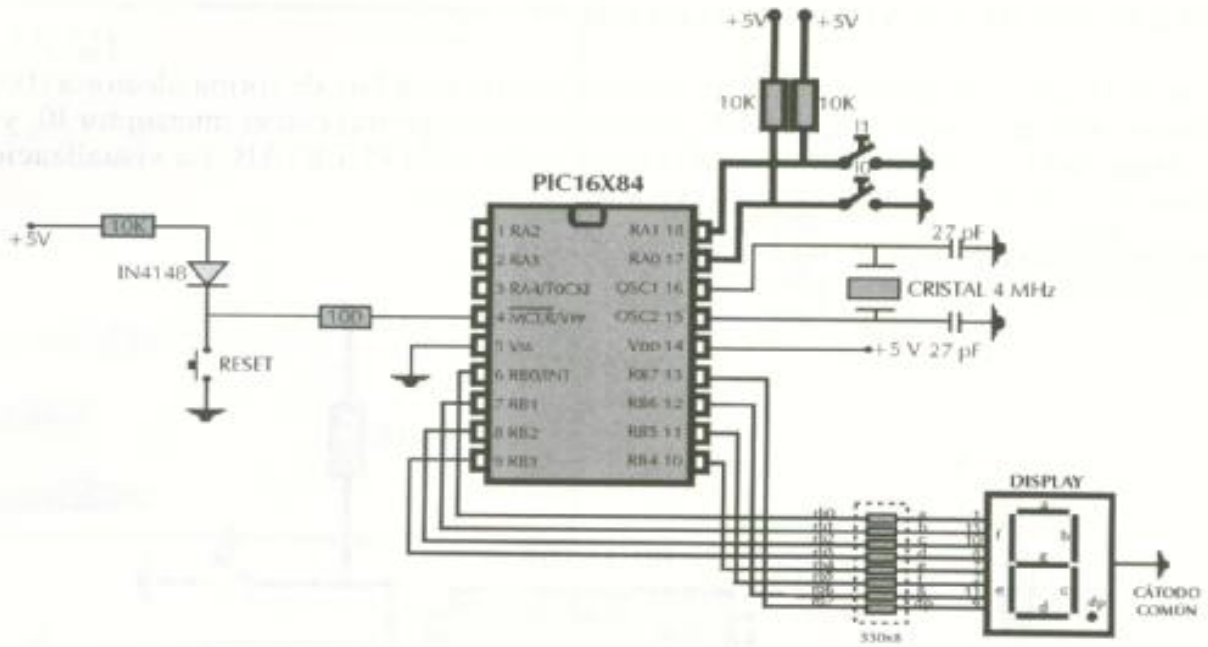


Figura F.51. *Visualización del estado lógico de IO. Se debe conectar el segmento dp.*

CONTROL DE ACCESO

Se desea controlar la capacidad máxima de un local, que no debe superar las 12 personas. Para conocer la entrada y salida de las personas existen dos sensores, I0 e I1, colocados estratégicamente.

Cuando hay personas en el local, la lámpara L está encendida, y se apaga automáticamente cuando no hay nadie.

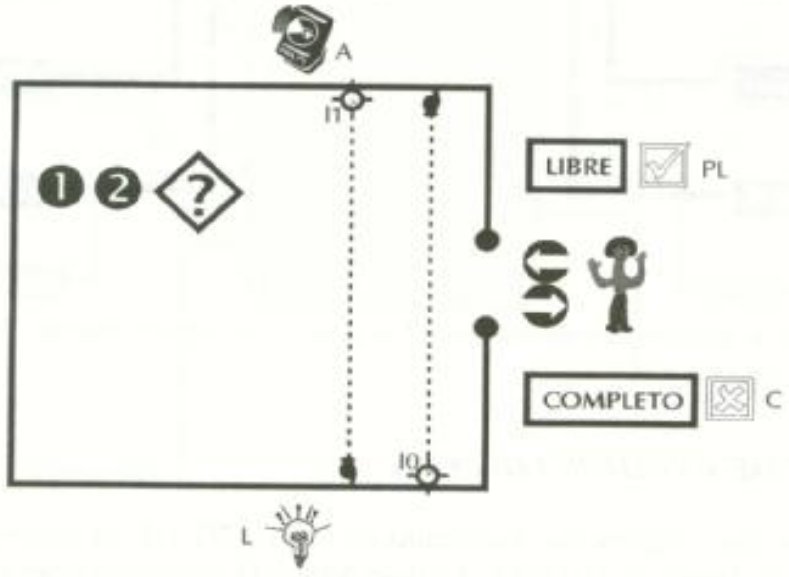


Figura F.52. *Boceto del proyecto.*

Si dentro del local hay menos de 12 personas, se ilumina en el exterior un cartel «PL» que indica PLAZAS LIBRES. Si hay 12 personas, se enciende el cartel «C» de COMPLETO.

Si estando COMPLETO el local alguien intenta entrar, se activa una alarma acústica «A».

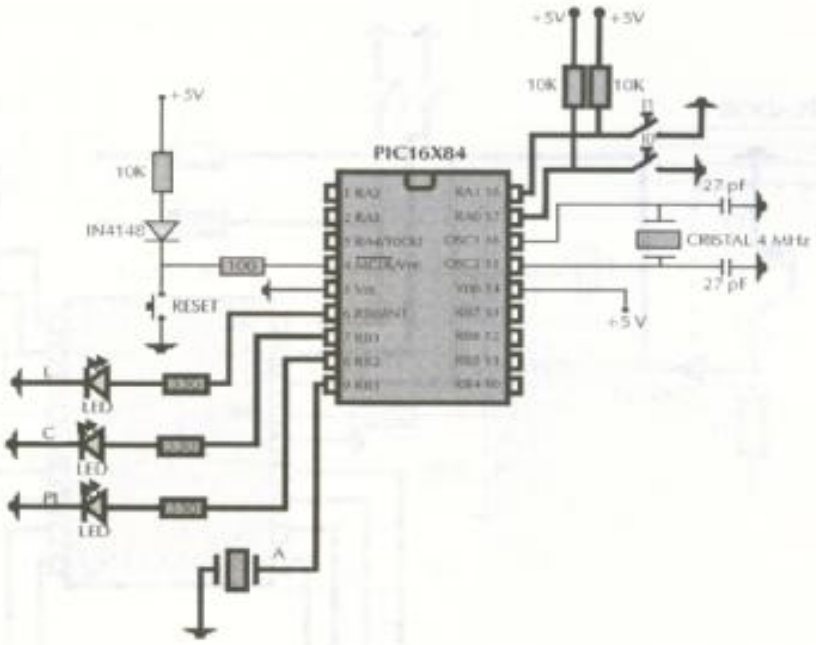


Figura F.53. *Control de acceso de personal. I0 e I1 pueden ser sustituidos por pulsadores u optoacopladores.*

CONTADOR DE HASTA MILES DE PIEZAS

Se dispone de un optoacoplador destinado a generar un impulso cada vez que pasa entre su detector y emisor una pieza. Las piezas o impulsos del optoacoplador los cuenta el TMR0, pero dividiendo dicho contaje entre un valor del divisor de frecuencia o preescaler interno. El valor entre el que divide el preescaler el número de impulsos depende del estado lógico que introduzcan los interruptores I0, I1 e I2, según la siguiente tabla.

I2	I1	I0	FACTOR DIVISIÓN
0	0	0	1:2
0	0	1	1:4
0	1	0	1:8
0	1	1	1:16
1	0	0	1:32
1	0	1	1:64
1	1	0	1:128
1	1	1	1:256

Figura F.54. *Tabla de verdad que indica el factor de división del preescaler según el valor de los interruptores.*

El contaje de los impulsos dividido entre el factor establecido por los interruptores se visualiza en ocho diodos LED.

El contador se resetea cuando I3=1.

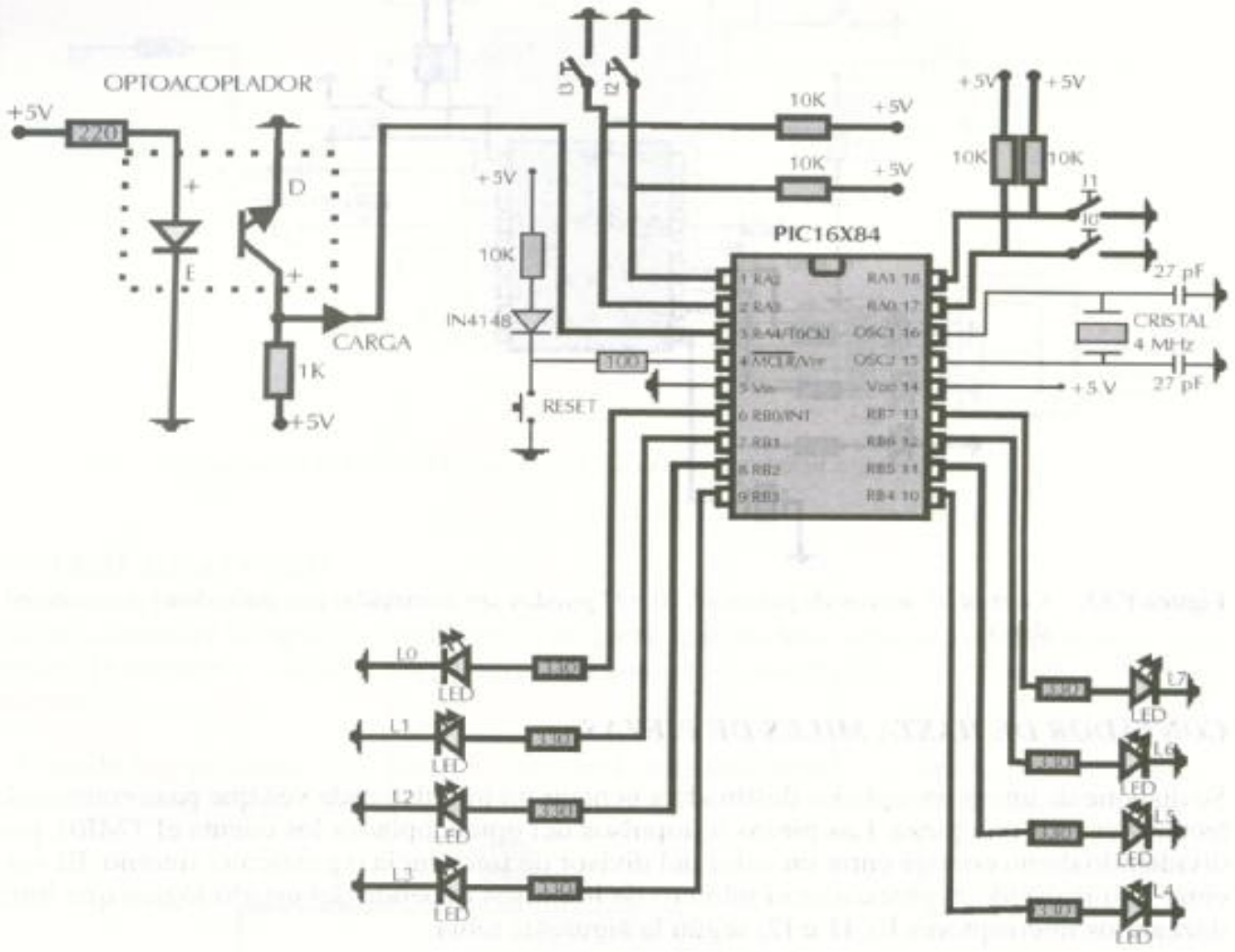


Figura F.55. *Visualización del contaje binario de impulsos del optoacoplador.*

CONTROL DE ACCESO POR CLAVE

Se pretende realizar el control de acceso a un local mediante la validación de una clave almacenada en la EEPROM de datos. Se utiliza un teclado para la introducción de la clave y el LCD para visualizar los mensajes de «Introducir clave», «acceso denegado» o «acceso permitido». Cuando el acceso sea denegado por no coincidir la clave introducida con la almacenada, se activará un zumbador durante 2 sg. Por el contrario, cuando el acceso sea permitido, se activará un relé que se supone conectado a la puerta de entrada.

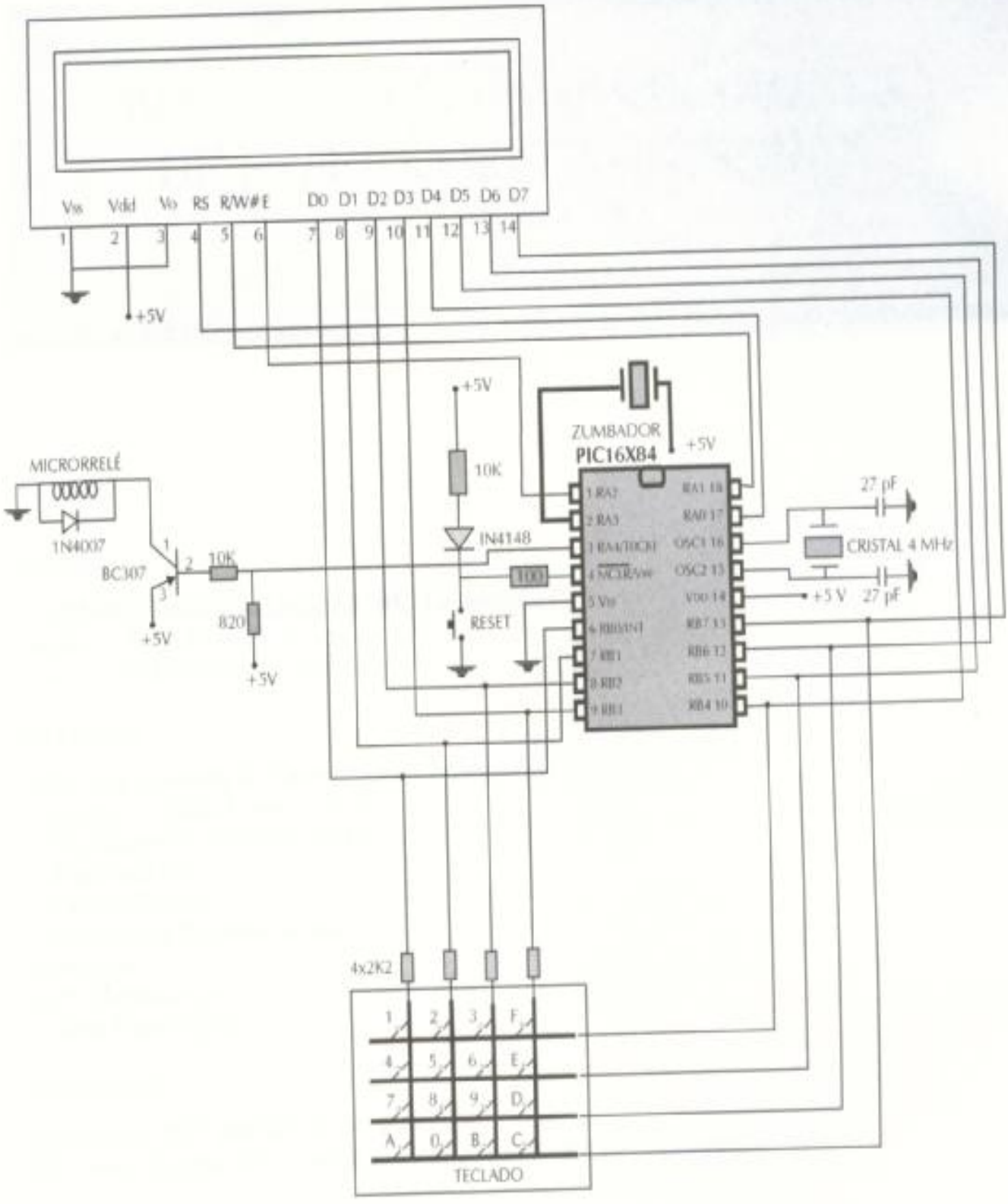


Figura F.56. *Esquema de conexionado de un control de acceso por clave.*

BIBLIOGRAFÍA Y DIRECCIONES DE INTERÉS RELACIONADAS CON LOS PIC

LIBROS

<<MICROCONTROLADORES PIC. La solución en un chip.>>
Angulo, J. M.ª; Martín, E., y Angulo, I.
Editorial ITP Paraninfo, 2.ª edición, 1999.

REVISTAS

<<Revista Española de Electrónica>>, Ediciones REDE.
<<Elektor>>, Edita Larpress, S. A.
<<Radiorama>>, Ediciones REDE.
<<Electronics>>
<<Circuit Cellar>>
<<Electrónica Práctica Actual>>
<<Resistor>>
<<PC Hardware>>
<<Todo Electrónica>>

EMPRESAS

<<MICROCHIP TECHNOLOGY INC.>>
2355 West Chandler Blv., Chandler, AZ 85224-6199, EE.UU. Tel: 602.786.7200.

<<SAGITRON>>
Corazón de María, 80-28002 MADRID (España). Tel: 91-4169261.

<<MICROSYSTEMS ENGINEERING>>
Gral. Concha, 39-48012 BILBAO (España). Tel: 94-4223263

INTERNET

MICROCHIP:	http://www.microchip.com
MICROSYSTEMS ENGINEERING:	http://www.arrakis.es/~msyseng
PARALLAX, INC:	http://www.parallaxinc.com
ITU TECHNOLOGIES:	http://www.itutech.com
SAGITRON:	http://www.sagitron.es

ÍNDICE

ALU, 74
Antirebotes, 34
Arquitectura, 75
ASCII, 71
Automoción, 34

Bajo consumo, 129
Banco de Registros, 74
Brinco, 87, 143
Brown Out, 127
Búsqueda, 75
BYTE CRAFT, 164

CCS, 9, 21, 164
CEBEK, 71, 170
Clave de acceso, 177
Código OP, 140
Contador, 95
    de Programa, 57, 77
Control de acceso, 179
Controlador incrustado, 1
Conversor AD, 31, 36
CP, 103

Dado, 173
DARLINGTON, 43
DATAQUEST, 1
Direccionamiento
    directo, 83, 188
    indirecto, 83, 188
Display 7 segmentos, 67, 70, 155
Divisor de frecuencia, 95

EDIT, 164
Editor, 17, 164, 193
EEADR, 106
EECON1, 106
EEDATA, 106
EEIF, 106
EEPROM, 7, 104, 184
Ejecutar, 194
Emulador en circuito, 9, 167
EPROM, 7

Flash, 7
Flip-flop, 25
Formato, 140, 142
FOSC1-FOSC0, 104
Fototransistor, 26
Frecuencia, 53

Gama
    Enana, 29
    Baja, 30
    Media, 31
    Alta, 32
Garage, 35
GIE, 118
Grabador, 165

HARVARD, 5, 74

INTEDG, 96
INTEL, 2, 28
Instrucciones
addlw k, 149
addwf f,d, 146
andlw k, 149
andwf f,d, 146
bcf f,b, 148
bsf f,b, 148
btfsc f,d, 149
btfss f,d, 149
call k, 151
clrf f, 146
clrw, 146
clrwdr, 151
comf f,d, 146
decf f,d, 146
decfsz f,d, 149
goto k, 151
incf f,d, 146
incfsz f,d, 149
iorlw k, 149
iorwf f,d, 146
mov f,d, 146
movlw k, 149
movwf f, 146
nop, 146
retfie, 151
retlw k, 151
return, 151
rlf f,d, 146
rrf f,d, 146
sleep, 151
sublw k, 149
subwf f,d, 146
swap f,d, 146
xorlw k, 149
xorwf f,d, 146
Instrumentación, 35
Interrupciones, 117, 131, 154
INT, 122
TMR0, 123
RB7:RB4, 124
EEPROM, 124
Interruptor, 23
ITU TECHNOLOGIES, 217

jumper, 11, 67

LCD, 71, 177
Led, 39, 42
Lenguaje
BASIC, 9
C, 9, 20, 66, 134
Ensamblador, 9, 20, 133, 164

MAX 232, 46
Mensajes, 177
Microcontrolador, 1
MICROCHIP, 7, 9, 28
MICROMINT, 217
Micro'PIC Programmer, 166
Microprocesador, 3
Micro'PIC Trainer, 12, 67, 167, 211
Microrelé, 44
MICROSYSTEMS ENGINEERING, 9, 10, 211, 217
MICROENGINEERING LABS, 217
Motor PAP, 47
Motores, 34
MOTOROLA, 55
MPASM, 9, 144
MPSIM, 9,18

NEUMANN, 5
Nomenclatura, 144

Optoacopladas, 25, 45
Organigrama, 88, 109, 119
Oscilador
RC, 55
HS, 56
XT, 56
LP, 56
OST, 126

Palabra de Configuración, 102
Palabra de Identificación, 104
PARALLAX, 164, 217

Perro Guardián, 8, 97
PIC, 27, 29
PIC16C84, 76, 182, 213
PIC16CR83, 76, 213
PIC16CR84, 76, 182, 213
PIC16F84, 76, 182, 213
PIC16F83, 76, 213
PIC18-ME, 59, 167, 172
Pila, 77
Power Down, 86
Programación serie, 129
PSA, 96
Puerta, 37
    de Entrada/Salida, 99, 184
    A, 100, 184
    B, 102, 184
PWRT, 126
PWRTE, 104

RAM, 78, 183
RBPU#, 102
Relé, 43
Reinicialización, 57, 125
Registro
    ESTADO, 85, 186
    FSR, 83
    INTCON, 118, 189
    OPTION, 96, 187
    PCH, 79
    PCL, 79
    W, 74, 153
Repertorio de instrucciones, 139, 190
Reposo, 129
RISC, 6
R.F. SOLUTIONS, 11, 168
ROM, 7

SAGITRON, 217
Segmentación, 6, 154
Semáforo, 175
Señalizador
    de acarreo, 14, 85
    de cero, 16, 86
    de acarreo intermedio, 16, 85
Símbolos, 144
Simulación
    opciones, 193
SIMUPIC'84, 16, 65, 89, 191
Sistema
    abierto, 3
    cerrado, 3
    de seguridad, 34
    de desarrollo, 11
    de bajo consumo, 34
Skip, 87, 143
Sleep, 129

T0CS, 94
T0CKI, 94
T0SE, 94
Teclado, 68, 70
Temporizador, 93
Tiempo real, 112
Time Out, 86
TMR0, 93
Trazar, 195

Ultravioleta (rayos), 7, 49

Véctor
    de Reinicialización, 64
    de Interrupción, 64
Ventanas, 196

Zumbador, 44