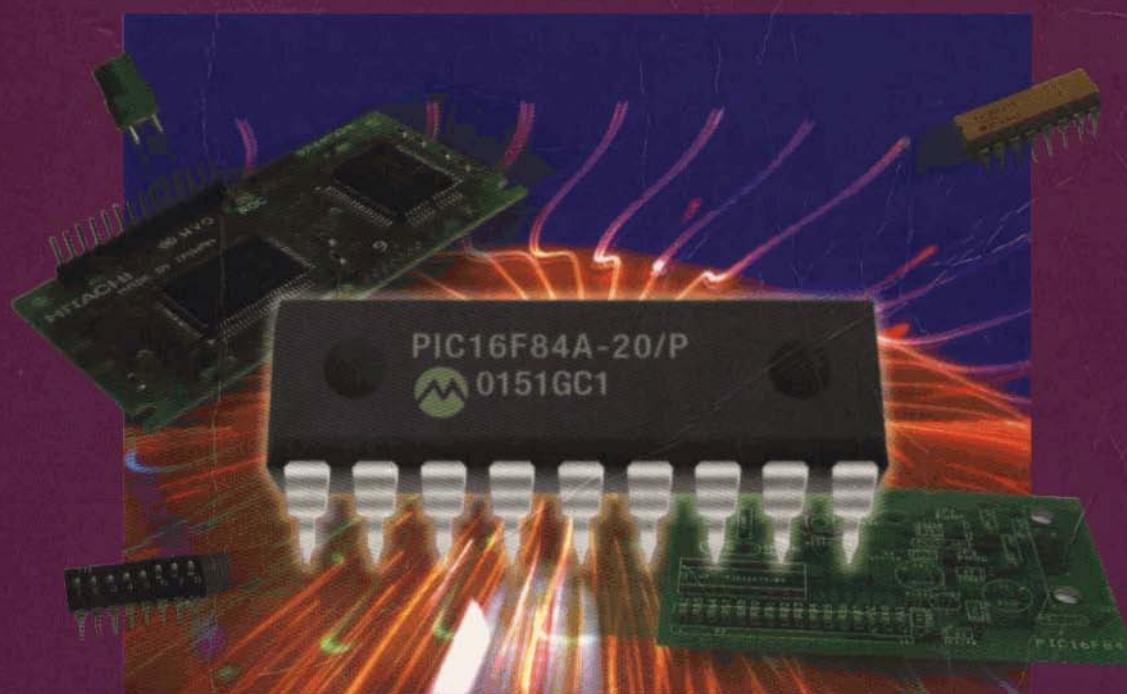


# Microcontrolador PIC16F84

## Desarrollo de proyectos



Enrique Palacios  
Fernando Remiro  
Lucas J. López



Incluye CD-ROM

Alfaomega  Ra-Ma®

# Microcontrolador **PIC16F84**

## Desarrollo de proyectos

Este libro pretende introducir al lector en la realización de proyectos de circuitos electrónicos construidos con el popular microcontrolador **PIC16F84**. Para lograrlo se muestra un elevado número de ejercicios resueltos que el lector podrá desarrollar fácilmente con medios a su alcance.

Tanto los aficionados sin grandes conocimientos de electrónica, pero con inquietud suficiente para montar sencillos trabajos con microcontroladores, como los estudiantes de carreras técnicas de electrónica y los estudiantes de Ingeniería Industrial, Telecomunicaciones o Informática, encontrarán de gran utilidad esta obra para la realización de sus primeros proyectos.

El libro resulta eminentemente práctico ya que contiene más de 160 ejercicios resueltos con sus programas y esquemas, siendo muchos de ellos proyectos clásicos, como termómetros, relojes, calendarios, cerraduras electrónicas, control de displays, termostatos, temporizadores, alarmas, sirenas, comunicación con la computadora, juegos, control de motores, microrobots, etc.

El software utilizado es de libre distribución y los circuitos emplean componentes que pueden adquirirse fácilmente en cualquier tienda de productos electrónicos. Para el desarrollo de cualquiera de los proyectos planteados no se precisa de grandes medios materiales, por lo que realizarlos resulta sencillo, económico y ameno, además, se incluye un CD-ROM que contiene el software necesario, las soluciones a los ejercicios y notas técnicas.

ISBN 970-15-1033-X



9 789701 510339



**Alfaomega Grupo Editor**

# **Microcontrolador PIC16F84**

## **Desarrollo de proyectos**

# **Microcontrolador PIC16F84**

## **Desarrollo de proyectos**

Enrique Palacios Municio  
Fernando Remiro Domínguez  
Lucas J. López Pérez

**Alfaomega**  **Ra-Ma®**

**Microcontrolador PIC16F84. Desarrollo de proyectos**  
© Enrique Palacios Municio, Fernando Remiro Domínguez  
y Lucas J. López Pérez

**ISBN 84-7897-600-0, edición original publicada por RA-MA Editorial,  
MADRID, España. Derechos reservados © RA-MA Editorial**

**MARCAS COMERCIALES:** RA-MA ha intentado a lo largo de este libro distinguir las marcas registradas de los términos descriptivos, siguiendo el estilo de mayúsculas que utiliza el fabricante, sin intención de infringir la marca y sólo en beneficio del propietario de la misma.

**Primera edición: Alfaomega Grupo Editar, México, agosto 2004**

**© 2004 ALFAOMEGA GRUPO EDITOR, S.A. de C.V.  
Pitágoras 1139, Col. Del Valle, 03100 México, D.F.**

**Miembro de la Cámara Nacional de la Industria Editorial Mexicana  
Registro No. 2317**

**ISBN 970-15-1033-X**

**Derechos reservados.**

**Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.**

**NOTA IMPORTANTE**

**La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos, han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR, S.A. de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.**

**Edición autorizada para venta en México y todo el continente americano**

**Impreso en México • Printed in Mexico**

*A mi mujer, Maribel, por su apoyo, cariño y paciencia.  
A mis hijos, Cristina y Enrique, mi mayor fuente de alegrías y satisfacciones.  
A mis padres, Enrique y Sagrario, en reconocimiento a su sacrificio.*

**Enrique**

*A mis padres, hijo y demás personas importantes de mi vida.*

**Lucas**

*A mis dos mujeres, Natalia con la que decidí compartir mis días y Mónica que me alegra cada día desde que nació y que a sus tres años no dejo de querer ayudarme a escribir y programar microcontroladores y periféricos.*

**Fernando**

# **CONTENIDO**

---

PRÓLOGO.....	XXI
--------------	-----

## **CAPITULOS**

1. MICROCONTROLADOR PIC16F84.....	1
2. PERIFÉRICOS BÁSICOS.....	9
3. GRABACIÓN DE MICROCONTROLADORES PIC .....	21
4. ORGANIZACIÓN DE LA MEMORIA .....	37
5. ARQUITECTURA INTERNA.....	47
6. ENSAMBLADOR .....	57
7. MPLAB .....	77
8. PROGRAMACIÓN ELEMENTAL .....	101
9. SALTOS.....	119
10. SUBRUTINAS .....	141
11. MANEJO DE TABLAS.....	157
12. SUBRUTINAS DE RETARDO.....	169
13. LCD.....	187
14. EEPROM DE DATOS.....	213
15. TIMER 0 .....	223
16. OTROS RECURSOS .....	235
17. INTERRUPCIONES. LECTURA DE ENTRADAS.....	253

18. INTERRUPCIÓN POR DESBORDAMIENTO DEL TIMER 0.....	271
19. TECLADO MATRICIAL.....	289
20. COMUNICACIÓN CON ORDENADOR .....	305
21. BUS I2C.....	331
22. 24LC256, MEMORIA EEPROM EN BUS I2C.....	345
23. DS1624, TERMÓMETRO EN BUS I2C .....	359
24. DS1307, RELOJ CALENDARIO EN BUS I2C.....	371
25. SAA1064, CONTROLADOR DE DISPLAY .....	397
26. PCF8574, EXPANSOR DE BUS I2C.....	409
27. PCF8591, ADC Y DAC EN BUS I2C.....	423
28. BUS DE UNA LÍNEA .....	441
29. MOTORES DE CORRIENTE CONTINUA.....	467
30. MOTORES PASO A PASO .....	481
31. SERVOMOTORES DE RADIOCONTROL .....	505
32. SENsoRES PARA MICROROBÓTICA .....	515
33. CONSTRUCCIÓN DE UN MICROROBOT .....	543

## APÉNDICES

A. CARACTERÍSTICAS TÉCNICAS DEL PIC16F84A.....	565
B. REPERTORIO DE INSTRUCCIONES .....	569
C. CONSTANTES Y OPERADORES.....	587
D. PRINCIPALES DIRECTIVAS DEL ENSAMBLADOR MPASM.....	589
E. REGISTROS ESPECIALES .....	601
F. GRABADOR T20-SE.....	611
G. CÓDIGO ASCII .....	613
H. DIRECCIONES DE INTERNET .....	615
I. CONTENIDO DEL CD-ROM .....	617
ÍNDICE ALFABÉTICO .....	619

## ÍNDICE

---

<b>PRÓLOGO .....</b>	<b>XXI</b>
<b>Capítulo 1: MICROCONTROLADOR PIC16F84.....</b>	<b>1</b>
1.1 Microcontroladores PIC .....	1
1.2 Alimentación de un PIC16F84 .....	2
1.3 Puertos de entrada/salida.....	3
1.4 Oscilador.....	4
1.4.1 Oscilador XT .....	5
1.4.2 Oscilador RC .....	5
1.4.3 Osciladores HS y LP .....	5
1.4.4 Utilizando una señal de reloj externa.....	6
1.5 Reset.....	7
1.6 Montaje del entrenador.....	8
<b>Capítulo 2: PERIFÉRICOS BÁSICOS .....</b>	<b>9</b>
2.1 Diodo LED.....	9
2.2 Interruptores y pulsadores.....	11
2.3 Entradas digitales con optoacopladores.....	11
2.4 Display de siete segmentos .....	12
2.5 Controlando cargas a 230 V.....	13
2.5.1 Control con relé .....	14
2.5.2 Control con relé miniatura en cápsula DIL.....	16
2.5.3 Control mediante fototriac .....	16
2.5.4 Control de potencia con triac .....	18
2.6 Zumbador.....	19

<b>Capítulo 3: GRABACIÓN DE MICROCONTROLADORES PIC .....</b>	<b>21</b>	6.5 El código.....
3.1 Grabación de un microcontrolador .....	21	6.5.1 E.....
3.2 Grabadores .....	22	6.5.2 G.....
3.3 Software de grabación IC-Prog.....	24	6.5.3 O.....
3.4 Grabación con medios reducidos .....	25	6.5.4 G.....
3.5 Proceso de grabación .....	26	6.5.5 P.....
3.6 Buffer de almacenamiento de programas .....	31	6.6 Constantes.....
3.7 IC-Prog trabajando bajo Windows 2000 o XP .....	31	6.7 Operaciones.....
3.8 Errores frecuentes en la programación .....	32	6.8 El reporte.....
3.9 Prácticas de laboratorio .....	34	6.9 Instrucciones.....
6.9.1 C.....		
6.9.2 G.....		
6.9.3 D.....		
6.9.4 I.....		
6.9.5 R.....		
6.9.6 S.....		
6.9.7 F.....		
6.9.8 L.....		
6.9.9 M.....		
6.9.10 P.....		
6.9.11 T.....		
6.9.12 V.....		
6.9.13 W.....		
6.9.14 X.....		
6.9.15 Y.....		
6.9.16 Z.....		
<b>Capítulo 4: ORGANIZACIÓN DE LA MEMORIA .....</b>	<b>37</b>	6.9.17 A.....
4.1 Arquitectura interna del PIC16F84.....	37	6.9.18 B.....
4.2 Organización de la memoria .....	37	6.9.19 C.....
4.3 Memoria de programa .....	39	6.10 Instrucciones.....
4.4 El contador de programa (PC) .....	39	6.10.1 C.....
4.5 Memoria de datos .....	41	6.10.2 D.....
4.6 Diferencias entre el PIC16F84A y el PIC16C84 .....	41	6.11 Instrucciones.....
4.7 Registros del SFR .....	42	6.12 Configuración.....
4.8 Registros relacionados con los puertos .....	42	6.13 Directivas.....
4.9 Registro PCL y contador de programa .....	42	6.13.1 C.....
4.10 Registro de trabajo W .....	43	6.13.2 D.....
4.11 Registro de estado o STATUS .....	43	6.13.3 E.....
4.12 Estado de los registros tras un reset .....	44	6.13.4 F.....
4.13 Registro de configuración .....	45	6.13.5 G.....
6.13.6 H.....		
<b>Capítulo 5: ARQUITECTURA INTERNA .....</b>	<b>47</b>	<b>Capítulo 7:</b>
5.1 Microprocesador y Microcontrolador .....	47	7.1 Entorno.....
5.2 Arquitectura de Von Neumann .....	48	7.2 Primeros.....
5.3 Arquitectura Harvard .....	49	7.3 Ensamblado.....
5.4 Procesador segmentado .....	50	7.4 Fichero.....
5.5 Procesador RISC .....	51	7.5 Ventana.....
5.6 Arquitectura ortogonal .....	52	7.5.1.....
5.7 Puertos .....	53	7.5.2.....
5.8 Puerto A .....	53	7.5.3.....
5.9 Puerto B .....	56	7.5.4.....
7.5.5.....		
7.5.6.....		
<b>Capítulo 6: ENSAMBLADOR .....</b>	<b>57</b>	7.6 Simulación.....
6.1 Lenguaje máquina .....	57	7.7 Simulación.....
6.2 Lenguaje ensamblador .....	58	7.8 Simulación.....
6.3 Programa ensamblador .....	58	7.9 Grabación.....
6.4 Ficheros resultantes del ensamblado .....	59	

21	6.5 El código fuente.....	59
21	6.5.1 Etiquetas.....	60
22	6.5.2 Código de operación.....	61
24	6.5.3 Operando.....	61
25	6.5.4 Comentarios.....	61
26	6.5.5 Normas de estilo para escribir un archivo fuente.....	62
31	6.6 Constantes numéricas y alfanuméricas.....	63
31	6.7 Operadores aritméticos.....	64
32	6.8 El repertorio de instrucciones.....	64
34	6.9 Instrucciones de carga .....	66
37	6.9.1 clrw.....	66
37	6.9.2 clrf f.....	67
37	6.9.3 movlw k .....	67
37	6.9.4 movf f,d.....	67
39	6.9.5 movwf f.....	67
39	6.10 Instrucciones de bit.....	68
39	6.10.1 bcf f,b .....	68
41	6.10.2 bsf f,b .....	68
41	6.11 Instrucción "goto k" .....	68
42	6.12 Configurar las líneas de los puertos.....	69
42	6.13 Directivas .....	71
42	6.13.1 END .....	71
43	6.13.2 EQU .....	72
43	6.13.3 ORG .....	72
44	6.13.4 __ CONFIG .....	73
45	6.13.5 LIST P=16F84A .....	73
47	6.13.6 INCLUDE <P16F84A.INC>.....	73
47	<b>Capítulo 7: MPLAB .....</b>	<b>77</b>
48	7.1 Entorno MPLAB .....	77
49	7.2 Primeros pasos con MPLAB IDE.....	78
50	7.3 Ensamblado del programa.....	84
51	7.4 Fichero hexadecimal resultante.....	86
52	7.5 Ventanas de visualización .....	87
53	7.5.1 Ventana de visualización de la memoria de programa .....	88
53	7.5.2 Ventana Disassembly .....	88
56	7.5.3 Ventana de visualización de los registros del SFR .....	88
57	7.5.4 Ventana de contenido de la memoria RAM.....	90
57	7.5.5 Ventana personalizada Watch.....	90
57	7.5.6 Línea de estado .....	91
58	7.6 Simulación básica.....	91
58	7.7 Simulación mediante Breakpoints y Traza.....	93
59	7.8 Simulación de entradas.....	95
59	7.9 Grabación con el archivo hexadecimal.....	96

7.10 Fichero listable.....	98	9.3 Saltos.....	9.3 Saltos.....
7.11 Prácticas de laboratorio .....	99		9.1.....
			9.1.....
			9.1.....
<b>Capítulo 8: PROGRAMACIÓN ELEMENTAL.....</b>	<b>101</b>	<b>9.4 Controladores.....</b>	<b>9.4 Controladores.....</b>
8.1 Instrucciones de suma .....	101		9.1.....
8.1.1 addlw k.....	102		9.1.....
8.1.2 addwf f,d .....	102		9.1.....
8.2 Instrucciones de resta .....	103	9.5 Lazos.....	9.5 Lazos.....
8.2.1 sublw k.....	103		9.1.....
8.2.2 subwf f,d .....	103		9.1.....
8.3 Incrementar y decrementar.....	104		9.1.....
8.3.1 decf f,d .....	104	9.6 Programación.....	9.6 Programación.....
8.3.2 incf f,d .....	104	9.7 Diagramas.....	9.7 Diagramas.....
8.4 Instrucciones lógicas .....	104	9.8 Módulos.....	9.8 Módulos.....
8.4.1 andlw k.....	105		9.1.....
8.4.2 andwf f,d .....	105		9.1.....
8.4.3 comf f,d.....	105	9.9 Controladores.....	9.9 Controladores.....
8.4.4 iorlw k.....	105	9.10 Salidas.....	9.10 Salidas.....
8.4.5 iorwf f,d.....	106	9.11 Salidas.....	9.11 Salidas.....
8.4.6 rlf f,d.....	106	9.12 Prácticas.....	9.12 Prácticas.....
8.4.7 rrf f,d .....	106		
8.4.8 swapf f,d.....	107	<b>Capítulo 9: SALTOS.....</b>	<b>10.1 Saltos.....</b>
8.4.9 xorlw k .....	107		10.2 Saltos.....
8.4.10 xorwf f,d.....	107		10.3 Saltos.....
8.5 Instrucción “sleep” .....	108	10.4 Instrucciones.....	10.4 Instrucciones.....
8.6 Algunas instrucciones útiles.....	109	10.5 Ejemplos.....	10.5 Ejemplos.....
8.7 Herramientas.....	111	10.6 Verificadores.....	10.6 Verificadores.....
8.7.1 Ensambladores y compiladores .....	111	10.7 Líneas.....	10.7 Líneas.....
8.7.2 Simuladores software .....	111	10.8 Diagramas.....	10.8 Diagramas.....
8.7.3 Emuladores .....	113	10.9 Sistemas.....	10.9 Sistemas.....
8.7.4 Grabadores o programadores .....	113	10.10 Proyectos.....	10.10 Proyectos.....
8.7.5 Sistemas de desarrollo .....	113	10.11 Prácticas.....	10.11 Prácticas.....
8.8 Proyectos con medios reducidos.....	114		
8.9 Desarrollo de proyectos sencillos .....	115	<b>Capítulo 11: TABLAS.....</b>	<b>11.1 Tablas.....</b>
8.10 Prácticas de laboratorio .....	117		11.1.....
<b>Capítulo 9: SALTOS.....</b>	<b>119</b>	11.2 Mapas.....	11.2 Mapas.....
9.1 Saltos condicionales .....	119		11.1.....
9.2 Saltos en función de un bit .....	120		11.1.....
9.2.1 Instrucción “btfscc f,b” .....	120		11.1.....
9.2.2 Instrucción “btfsf f,b”.....	120		11.1.....

98	9.3 Saltos en función de un registro.....	121
99	9.3.1 Instrucción "decfsz f,d".....	121
	9.3.2 Instrucción "incfsz f,d".....	121
101	9.4 Comparación de registros.....	122
	9.4.1 Comprobar que un registro vale 0 .....	122
101	9.4.2 Comprobar igualdad entre dos registros.....	122
102	9.4.3 Comprobar que un registro es mayor o menor que otro .....	123
102	9.4.4 Programa ejemplo.....	123
103	9.5 Lazos o bucles.....	125
103	9.5.1 Lazo de repetición infinita.....	125
103	9.5.2 Lazo con condición de testeо.....	126
104	9.5.3 Lazo que se repite un número conocido <i>de</i> veces.....	126
104	9.6 Programación y algoritmo.....	127
104	9.7 Diagramas de flujo .....	127
104	9.8 Más directivas importantes .....	130
105	9.8.1 CBLOCK y ENDC.....	130
105	9.8.2 #DEFINE .....	130
105	9.9 Conversión de binario natural a BCD.....	131
105	9.10 Salto indexado .....	134
105	9.11 Salto indexado descontrolado .....	136
106	9.12 Prácticas de laboratorio .....	138
106	<b>Capítulo 10: SUBRUTINAS.....</b>	<b>141</b>
107	10.1 Subrutinas.....	141
107	10.2 Subrutinas anidadas.....	143
108	10.3 La pila .....	145
109	10.4 Instrucciones "call" y "return".....	146
111	10.5 Ejemplo de utilización de las subrutinas .....	147
111	10.6 Ventajas de las subrutinas .....	149
111	10.7 Librería de subrutinas .....	149
113	10.8 Directiva "INCLUDE" .....	149
113	10.9 Simulación de subrutinas en MPLAB .....	154
113	10.10 Programación estructurada.....	155
114	10.11 Practicas de laboratorio .....	156
115	<b>Capítulo 11: MANEJO DE TABLAS.....</b>	<b>157</b>
119	11.1 Tablas de datos en memoria de programa.....	157
	11.1.1 Instrucción "retlw" .....	157
	11.1.2 Directiva "DT" .....	159
119	11.2 Más directivas.....	160
120	11.2.1 MESSG .....	160
120	11.2.2 ERROR .....	161
120	11.2.3 IF y ENDIF .....	161

11.3 Gobierno de un display de 7 segmentos.....	162
11.3 Prácticas de laboratorio.....	166
<b>Capítulo 12: SUBRUTINAS DE RETARDO .....</b>	<b>169</b>
12.1 Ciclo máquina.....	169
12.2 Medir tiempos con MPLAB .....	171
12.3 Instrucción "nop" .....	171
12.4 Retardos mediante lazo simple .....	172
12.5 Retardos mediante lazos anidados.....	174
12.6 Librería con subrutinas de retardos.....	176
12.7 Rebotes en los pulsadores .....	181
12.8 Prácticas de laboratorio.....	184
<b>Capítulo 13: LCD.....</b>	<b>187</b>
13.1 Visualizador LCD.....	187
13.2 Patillaje.....	188
13.3 DDRAM .....	189
13.4 Caracteres definidos en la CGROM .....	191
13.5 Modos de funcionamiento.....	191
13.6 Comandos de control.....	192
13.7 Conexión de LCD mediante 4 bits .....	193
13.8 Librería de subrutinas.....	194
13.9 Visualización de caracteres.....	201
13.10 Visualización de valores numéricos.....	202
13.11 Conexión de LCD mediante 8 bits .....	203
13.12 Visualización de mensajes fijos .....	204
13.13 Visualización de mensajes en movimiento .....	208
13.14 Prácticas de laboratorio.....	209
<b>Capítulo 14: EEPROM DE DATOS.....</b>	<b>213</b>
14.1 Memoria EEPROM de datm .....	213
14.2 Registro EECON1 .....	215
14.3 Librería de subrutinas.....	216
14.4 Lectura de la EEPROM de datos.....	217
14.5 Escritura en la EEPROM de datos.....	217
14.6 Directiva "DE" .....	218
14.7 Ventana "EEPROM" en el MPLAB .....	218
14.8 Programa ejemplo.....	218
14.9 Bloquear un circuito .....	221
14.10 Prácticas de laboratorio.....	222

162	<b>Capítulo 15: TIMER 0 .....</b>	<b>223</b>
166	15.1 El Timer 0 (TMR0) .....	223
<b>169</b>	15.2 TMR0 como contador .....	224
	15.3 TMR0 como temporizador.....	224
169	15.4 El TMR0 es un registro del SFR.....	225
171	15.5 Divisor de frecuencia (Prescaler).....	225
171	15.6 Bits de configuración del TMR0 .....	226
172	15.6.1 Del registro INTCON.....	226
174	15.6.2 Del registro OPTION .....	226
176	15.7 Ejemplo del TMR0 como contador .....	227
181	15.8 Ejemplo del TMR0 como temporizador.....	229
184	15.9 Prácticas de laboratorio .....	233
<b>187</b>	<b>Capítulo 16: OTROS RECURSOS .....</b>	<b>235</b>
187	16.1 El Watchdog (WDT) .....	235
188	16.2 Modo de bajo consumo o “SLEEP” .....	237
189	16.3 Direccionamiento indirecto.....	241
191	16.4 Macros.....	243
191	16.5 Resistencias de Pull-Up del Puerto B .....	249
192	16.6 Prácticas de laboratorio .....	250
193		
194	<b>Capítulo 17: INTERRUPCIONES. LECTURA DE ENTRADAS.....</b>	<b>253</b>
201	17.1 Técnica Polling .....	253
202	17.2 Interrupciones .....	255
203	17.3 Funcionamiento de una interrupción.....	257
204	17.4 Flags relacionados con interrupciones.....	258
208	17.4.1 Del registro INTCON.....	258
209	17.4.2 Del registro OPTION .....	259
<b>213</b>	17.5 Instrucción “retfie” .....	259
	17.6 Interrupción externa INT.....	260
213	17.7 Registros alterados por la interrupción .....	261
215	17.8 Averiguar la causa de la interrupción .....	264
216	17.9 Fases de una interrupción.....	264
217	17.10 Interrupción RBI.....	267
217	17.11 Prácticas de laboratorio .....	268
218		
<b>218</b>	<b>Capítulo 18: INTERRUPCIÓN POR DESBORDAMIENTO DEL                   TIMER 0 .....</b>	<b>271</b>
218	18.1 Interrupción producida por el TMR0 .....	271
221	18.2 Temporizaciones exactas .....	273
222	18.3 Temporizaciones largas.....	274
	18.4 Temporizador digital .....	276
	18.5 Prácticas de laboratorio .....	286

<b>Capítulo 19: TECLADO MATRICIAL .....</b>	<b>289</b>
19.1 Teclado hexadecimal.....	290
19.2 Conexión de un teclado a un PIC16F84.....	290
19.3 Algoritmo de programación.....	292
19.4 Librería de subrutinas .....	293
19.5 Ejemplo de aplicación .....	298
19.6 Cerradura electrónica .....	299
19.7 Prácticas de laboratorio .....	303
<b>Capítulo 20: COMUNICACIÓN CON ORDENADOR.....</b>	<b>305</b>
20.1 Puerto serie RS232 .....	305
20.2 El Baudio .....	307
20.3 Niveles lógicos RS232 .....	307
20.4 Formato de un byte.....	308
20.5 MAX232 .....	309
20.6 Conexión puerto RS232 y PIC16F84.....	310
20.7 Librería de subrutinas para RS232 .....	312
20.8 El HyperTerminal .....	315
20.9 Programa ejemplo.....	318
20.10 Librería RS232_MEN.INC.....	319
20.11 Sistema de monitorización .....	322
20.12 Sistema de gobierno desde ordenador .....	324
20.13 Prácticas de laboratorio .....	328
<b>Capítulo 21: BUS I2C .....</b>	<b>331</b>
21.1 El bus I2C .....	331
21.2 Hardware del bus I2C.....	333
21.3 Transferencia de un bit por la línea SDA .....	334
21.4 Condiciones de START y STOP .....	334
21.5 Transferencia de datos.....	335
21.6 Formato de una transferencia de datos .....	336
21.7 Tipos de formatos de transferencia.....	337
21.8 Temporización .....	338
21.9 Conexión de bus I2C a un PIC16F84.....	339
21.10 Librería de subrutinas para bus I2C.....	340
21.11 Dispositivos I2C .....	343
<b>Capítulo 22: 24LC256, MEMORIA EEPROM EN BUS I2C.....</b>	<b>345</b>
22.1 Memoria EEPROM serie 24LC256 .....	345
22.2 Paginación de la memoria 24LC256 .....	346
22.3 Direccionamiento como esclavo .....	347
22.4 Conexión de una 24LC256 a un PIC16F84 .....	347
22.5 Escritura en la memoria 24LC256.....	348

.....	289	22.6 Lectura de la memoria 24LC256 .....	349
.....	290	22.7 Librería de subrutinas .....	350
.....	290	22.8 Ejemplo típico de aplicación .....	352
.....	292	22.9 Grabación de datos mediante el IC-Prog .....	354
.....	293	22.10 Visualización de mensajes largos .....	355
.....	298	22.11 Control de muchos mensajes .....	356
.....	299	<b>Capítulo 23: DS1624, TERMÓMETRO EN BUS I2C.....</b>	<b>359</b>
.....	303	.....	
.....	305	23.1 El sensor de temperatura DS1624.....	359
.....	305	23.2 Direccionamiento como esclavo .....	360
.....	307	23.3 Lectura de la temperatura .....	360
.....	307	23.4 Registro de control .....	362
.....	308	23.5 Comandos .....	363
.....	308	23.6 Librería de subrutinas .....	364
.....	309	23.7 Termómetro digital .....	367
.....	310	<b>Capítulo 24: DS1307, RELOJ CALENDARIO EN BUS I2C.....</b>	<b>371</b>
.....	312	.....	
.....	315	24.1 El reloj-calendario DS1307 .....	371
.....	318	24.2 Conexión de un DS1307 a un PIC16F84 .....	372
.....	319	24.3 Registros del DS1307 .....	374
.....	322	24.4 Registro de control .....	375
.....	324	24.5 Escritura en el DS1307 .....	376
.....	328	24.6 Lectura del DS1307 .....	376
.....	331	24.7 Librería de subrutinas .....	377
.....	331	24.8 Programa del reloj calendario digital .....	379
.....	333	<b>Capítulo 25: SAA1064, CONTROLADOR DE DISPLAY .....</b>	<b>397</b>
.....	334	.....	
.....	334	25.1 SAA1064, controlador de display .....	397
.....	335	25.2 Circuito típico para modo estático .....	397
.....	336	25.3 Circuito típico para modo dinámico .....	400
.....	337	25.4 Direccionamiento como esclavo .....	400
.....	338	25.5 Registros internos .....	401
.....	339	25.6 Escritura en el SAA1064 .....	403
.....	340	25.7 Programa ejemplo .....	403
.....	343	25.8 Termómetro de visualización en displays .....	404
.....	345	<b>Capítulo 26: PCF8574, EXPANSOR DE BUS I2C.....</b>	<b>409</b>
.....	345	.....	
.....	346	26.1 El expansor de bus I2C PCF8574 .....	409
.....	346	26.2 Direccionamiento como esclavo .....	410
.....	347	26.3 Escritura en el PCF8574 .....	411
.....	347	26.4 Lectura del PCF8574 .....	411
.....	348	26.5 Librería de subrutinas .....	412
.....	348	26.6 Interrupción .....	413

26.7 Conexión entre PCF8574 y PIC16F84.....	414	30.5 Co
26.8 Ejemplo de programa .....	415	30.6 Di
26.9 Constitución interna del puerto.....	416	30.7 Pa
26.10 Teclado hexadecimal en bus I2C .....	417	30.8 Co
<b>Capítulo 27: PCF8591, ADC Y DAC EN BUS I2C .....</b>	<b>423</b>	30.9 Ide
27.1 PCF8591 .....	423	30.10 Co
27.2 Direccionamiento como esclavo.....	424	30.11 Co
27.3 Registro de control .....	425	30.12 Co
27.4 El PCF8591 como DAC.....	426	30.13 Re
27.5 Resolución del DAC.....	428	30.14 Co
27.6 Ejemplos del PCF8591 como DAC.....	430	30.15 Co
27.7 El PCF8591 como ADC.....	436	<b>Capítulo</b>
27.8 Ejemplo del PCF8591 como ADC .....	438	31.1 Ser
<b>Capítulo 28: BUS DE UNA LÍNEA .....</b>	<b>441</b>	31.2 Fue
28.1 Sensor de temperatura DS1820.....	441	31.3 Ter
28.2 Diagrama en bloques del DS1820 .....	442	31.4 Col
28.3 Lectura de la temperatura .....	443	<b>Capítulo</b>
28.4 Bus de una línea.....	444	32.1 Ser
28.5 Señales del bus de una línea.....	445	32.2 Inv
28.6 Inicialización: Pulsos Reset y Presence.....	445	32.3 LD
28.7 Escritura de un bit sobre el DS1820 .....	446	32.4 Fot
28.8 Lectura de un bit procedente del DS1820 .....	446	32.
28.9 Librería de subrutinas para bus de 1 línea .....	447	32.
28.10 Único DS1820 conectado al bus de 1 línea.....	450	32.
28.11 Termostato digital.....	453	32.5 Sen
<b>Capítulo 29: MOTORES DE CORRIENTE CONTINUA .....</b>	<b>467</b>	32.
29.1 Puente en H.....	468	32.
29.2 Driver L293B.....	469	32.
29.3 Giro en un único sentido .....	471	32.6 Rec
29.4 Giro en los dos sentidos .....	472	32.6
29.5 Conexión de motor c.c. y PIC16F84 .....	473	32.6
29.6 Control de velocidad .....	475	32.6
<b>Capítulo 30: MOTORES PASO A PASO.....</b>	<b>481</b>	32.7 Sen
30.1 Motores paso a paso (PAP).....	481	32.8 Bum
30.2 Principio de funcionamiento .....	482	32.9 Dete
30.3 Motores PAP bipolares .....	484	<b>Capítulo</b>
30.3.1 Motor PAP bipolar en modo Full Step.....	485	33.1 Intr
30.3.2 Motor PAP bipolar en modo Half Step .....	485	33.2 Nive
30.4 Motores PAP unipolares .....	486	33.2

.....	414	30.5 Constitución interna de un motor PAP .....	489
.....	415	30.6 Disposición de las bobinas .....	490
.....	416	30.7 Parámetro de los motores PAP .....	491
.....	417	30.8 Control de los motores paso a paso .....	492
.....	423	30.9 Identificación de un motor PAP .....	492
.....	423	30.10 Conexión motor PAP bipolar y PIC16F84 .....	494
.....	424	30.11 Conexión motor PAP Unipolar y PIC16F84 .....	495
.....	425	30.12 Control de motor PAP en modo Full Step .....	496
.....	426	30.13 Realización de secuencias de movimientos .....	498
.....	428	30.14 Control de motor PAP en modo Half Step .....	500
.....	430	30.15 Control de velocidad .....	502
.....	436	<b>Capítulo 31: SERVOMOTORES DE RADIOCONTROL .....</b>	<b>505</b>
.....	438	31.1 Servomotores para microrobótica .....	505
.....	441	31.2 Funcionamiento del servomotor .....	507
.....	441	31.3 Terminales .....	508
.....	442	31.4 Conexión de un servomotor a un PIC16F84 .....	509
.....	443	<b>Capítulo 32: SENSORES PARA MICROROBÓTICA .....</b>	<b>515</b>
.....	444	32.1 Sensores para microrobótica .....	515
.....	445	32.2 Inversor Trigger Schmitt 40106 .....	515
.....	445	32.3 LDR .....	518
.....	446	32.4 Fotosensores activos .....	521
.....	446	32.4.1 Sensor óptico CNY70 .....	522
.....	447	32.4.2 Sensores ópticos OPB703/4/5 .....	524
.....	450	32.4.3 Ejemplo de aplicación .....	525
.....	453	32.4.4 Sensor óptico de barrera H21A1 .....	526
.....	467	32.5 Sensores infrarrojos GP2DXX .....	527
.....	468	32.5.1 Principio de funcionamiento .....	528
.....	469	32.5.2 GP2D05 .....	529
.....	471	32.5.3 GP2D15 .....	530
.....	472	32.5.4 GP2D12 .....	530
.....	473	32.6 Receptor para control remoto SFH5110 .....	531
.....	475	32.6.1 Descripción .....	531
.....	481	32.6.2 Circuito detector .....	533
.....	481	32.6.3 Circuito emisor .....	533
.....	482	32.7 Sensor de proximidad IS471F .....	533
.....	484	32.8 Bumpers .....	535
.....	485	32.9 Detector por ultrasonido SRF04 .....	537
.....	485	<b>Capítulo 33: CONSTRUCCIÓN DE UN MICROROBOT .....</b>	<b>543</b>
.....	486	33.1 Introducción a la Micrótica .....	543
.....	486	33.2 Nivel físico. Motores .....	545
.....	486	33.2.1 Motores de corriente continua de pequeña potencia .....	545

33.2.2 Motores de corriente continua con reductoras .....	545
33.2.3 Servomotores .....	545
33.2.4 Modificación de un servomotor .....	546
33.2.5 Fijación del motor a la estructura .....	549
33.3 Nivel físico. Estructura .....	550
33.3.1 Estructuras comerciales .....	550
33.3.2 Estructura del microrobot experimental "Trasto" .....	551
33.4 Nivel físico. Ruedas .....	552
33.4.1 Estructuras según la colocación de las ruedas .....	552
33.4.2 Ruedas "locas" .....	553
33.4.3 Ruedas de tracción .....	554
33.5 Nivel físico. Movilidad .....	555
33.6 Nivel de reacción .....	556
33.7 Nivel de control .....	559
33.7.1 Estrategia a seguir para un microbot rastreador .....	559
33.7.2 Programa del rastreador .....	561
33.7.3 Estrategia a seguir para un robot detector de baliza .....	562
33.7.4 Programa de robot detector de baliza .....	563

## APÉNDICES

A. CARACTERÍSTICAS TÉCNICAS DEL PIC16F84A .....	565
B. REPERTORIO DE INSTRUCCIONES .....	569
C. CONSTANTES Y OPERADORES .....	587
D. PRINCIPALES DIRECTIVAS DEL ENSAMBLADOR MPASM .....	589
E. REGISTROS ESPECIALES .....	601
F. GRABADOR TE20-SE .....	611
G. CÓDIGO ASCII .....	613
H. DIRECCIONES DE INTERNET .....	615
I. CONTENIDO DEL CD-ROM .....	617
 ÍNDICE ALFABÉTICO .....	 619

545  
545  
546  
549  
550  
550  
551  
552  
552  
553  
554  
555  
556  
559  
559  
561  
562  
563

## PRÓLOGO

---

Los microcontroladores se utilizan en circuitos electrónicos comerciales desde hace unos años de forma masiva, debido a que permiten reducir el tamaño y el precio de los equipos. Un ejemplo de éstos son los teléfonos móviles, las cámaras de video, la televisión digital, la transmisión por satélite y los hornos microondas. Pero hasta hace poco tiempo, para el aficionado a la electrónica resultaba poco menos que imposible incluirlos en sus montajes por diversas razones: alto precio, complejidad de los montajes y, principalmente, por la escasez y el alto precio de las herramientas software.

En los últimos años se ha facilitado enormemente el trabajo con los microcontroladores al bajar los precios, aumentar las prestaciones y simplificar los montajes, de manera que en muchas ocasiones merece la pena utilizarlos en aplicaciones donde antes se utilizaba lógica discreta.

Diversos fabricantes ofrecen amplias gamas de microcontroladores para todas las necesidades. Pero, sin duda, hoy en día los microcontroladores más aceptados para diseños aficionados (y buena parte de los profesionales) son los microcontroladores PIC fabricados por *Microchip Technology Inc*, que recientemente se anunciaba como el mayor fabricante del mundo de microcontroladores de 8 bits.

En este auge ha influido decisivamente la política de *Microchip* al ofrecer la documentación y todo el software necesario de forma gratuita en su página Web [www.microchip.com](http://www.microchip.com). Esto, junto con otras cuestiones técnicas, han hecho que hoy en día resulte muy fácil incluir los microcontroladores PIC no solo en los diseños de los aficionados a la electrónica, sino también en complejos sistemas digitales.

Entre los microcontroladores PIC destaca el PIC16F84 cuya simplicidad, prestaciones, facilidad de uso y precio lo han convertido en el más popular de los

microcontroladores, siendo un chip ordinario de 18 patillas, cuya pequeña estructura de plástico contiene mucha de la tecnología que se necesita conocer para entender los sistemas de control con microprocesadores.

El PIC16F84 es un dispositivo ideal para aprender técnicas de software y del microprocesador, especialmente para estudiantes de electrónica con un conocimiento previo mínimo. Es relativamente barato y, mejor aún, puede volverse a utilizar porque es fácilmente reprogramable.

El microcontrolador es un dispositivo independiente y programable; y el estudiante, ya sea ingeniero o aficionado, puede utilizarlo sin saber en detalle cómo funciona. Por otra parte, podemos aprender mucho estudiándolo internamente.

Los autores de este libro llevamos muchos años dedicándonos a la enseñanza de microprocesadores y microcontroladores. Cuando comenzamos a impartir nuestras primeras clases sobre microprocesadores, las prácticas de laboratorio eran realizadas por los alumnos sobre unos entrenadores más bien voluminosos, que disponían de un teclado, una pantalla con unos pocos displays y todo el sistema que hacía que aquello funcionara. Los estudiantes tecleaban una lista de códigos hexadecimales, pulsaban la tecla *run* y comprobaban si esa lista de códigos realizaba la función esperada. Pocos alumnos tenían claro lo que estaban haciendo e incluso algunos creían que esa "maleta" era el microprocesador. Como anécdota, cuando uno de ellos tuvo un microprocesador en sus manos preguntó dónde estaba el teclado.

Nuestra experiencia docente nos ha demostrado que para dominar este mundo de la electrónica programable es necesario que el alumno realice "todo", desde el esquema a la soldadura del último terminal, e incluso, la maqueta que dé sentido al sistema microprogramable.

Este método de trabajo plasmado en el libro tiene grandes ventajas. Entre otras, dispone que sea el propio lector quien trabaje en un entorno real, creado por él mismo y no en un sistema preconcebido que en la mayoría de los casos tarda en comprender. No implica depender de costosos equipos, accesibles únicamente en laboratorios, sino que precisa exclusivamente de medios de trabajo presentes en la vida cotidiana. Y por último, potencia la autosuficiencia y la capacidad creativa del lector, al ser el mismo el que constantemente modifica el sistema y, afianza su autoestima al comprobar que lo que ha creado, funciona y realiza la tarea para la que fue ideado.

En este libro tratamos de plasmar la experiencia adquirida en nuestros años de docencia y facilitar la creación de nuevas prácticas tomando como base las aquí descritas, habiéndose comprobado su correcto funcionamiento, salvo error de transcripción. También hemos querido mostrar una gran variedad de periféricos que pueden usarse junto con un microprocesador/microcontrolador, desde el más común de los transistores, a complejos sensores para bus I2C.

La imprescindible sobre circuitos

Espe la finalidad disponer en autodidacta ser capaz de diseños.

Nuev Semiconductores constante es:

Quer prestada, su Redondo, Escobosa, Sergio Gómez Eduardo F.

ña estructura de  
ra entender los

software y del  
n conocimiento  
utilizar porque es

gramable; y el  
en detalle cómo  
ente.

la enseñanza de  
mpartir nuestras  
an realizadas por  
an de un teclado,  
uello funcionara.  
n la tecla run y  
s alumnos tenian  
"maleta" era el  
rocesador en sus

r este mundo de  
sde el esquema a  
ntido al sistema

jas. Entre otras,  
por el mismo y  
comprender. No  
tarios, sino que  
ia. Y por Ultimo,  
l mismo el que  
ar que lo que ha

uestros años de  
is aquí descritas.  
ipción. También  
se junto con un  
es, a complejos

La teoría desarrollada va siempre dirigida a hacer las prácticas, siendo la imprescindible para la realización de los proyectos de dificultad creciente que trabajan sobre circuitos reales.

Esperamos que la lectura de este libro le resulte sencilla y sobre todo que cumpla la finalidad para la que está escrito, que con los pocos medios técnicos de los que se suele disponer en casa, sea capaz de desarrollar proyectos microprogramables de una forma autodidacta. Cualquier aficionado, estudiante o ingeniero, con ayuda de este libro, debe ser capaz de empezar a utilizar el PIC16F84 inmediatamente en sus propios proyectos y diseños.

Nuestro agradecimiento a las fabricantes *Microchip Technology Inc*, *Philips Semiconductors* y *Dallas Semiconductors*, así como a la empresa *Sagitron* por su constante esfuerzo en ayudar a los usuarios en la utilización de sus productos.

Queremos finalmente agradecer a todos los compañeros y alumnos, la ayuda prestada, sugerencias y participación en el desarrollo de esta obra: Carmen Gómez, Julio Redondo, Jesús Sanz, Javier Temprado, Gemma Gil, Juan M. Morales, José M. Escobosa, Ana Zamora, Nuria Torijano, José A. Sanz, Alejandro Pico, Loli Moreno, Sergio González-Nicolás, Javier García-Caro, Diego A. Córdoba, Alfonso Martín, Eduardo F. García Folgar, Ángel Toledo y Fernando Blanco.

#### LOS AUTORES

## CAPITULO 1

# MICROCONTROLADOR PIC16F84

---

### 1 . MICROCONTROLADORES PIC

Un microcontrolador es un circuito integrado programable que contiene todos los componentes necesarios para controlar el funcionamiento de una tarea determinada, como el control de una lavadora, un teclado de ordenador, una impresora, un sistema de alarma, etc. Para ésto, el microcontrolador utiliza muy pocos componentes asociados. Un sistema con microcontrolador debe disponer de una memoria donde se almacena el programa que gobierna el funcionamiento del mismo que, una vez programado y configurado, sólo sirve para realizar la tarea asignada. La utilización de un microcontrolador en un circuito reduce notablemente el tamaño y número de componentes y, en consecuencia, disminuye el número de averías y el volumen y el peso de los equipos, entre otras ventajas.

El microcontrolador es uno de los inventos más notables del siglo XX. En el mercado hay gran cantidad de ellos, con multitud de posibilidades y características. Cada tipo de microcontrolador sirve para una serie de casos y es el diseñador del sistema quien debe decidir cuál es el microcontrolador más idóneo para cada uso.

En los últimos años han tenido un gran auge los microcontroladores PIC fabricados por *Microchip Technology Inc.* Los **PIC** (*Peripheral Interface Controller*) son una familia de microcontroladores que ha tenido gran aceptación y desarrollo en los últimos años gracias a que sus buenas características, bajo precio, reducido consumo, pequeño tamaño, gran calidad, fiabilidad y abundancia de información, lo convierten en muy fácil, cómodo y rápido de utilizar.

Este libro se centra en el estudio de un microcontrolador PIC muy popular, el PIC16F84. Está encapsulado en un económico DIL de 18 pines (figura 1-1). Debido a sus múltiples aplicaciones y facilidad de uso es uno de los microcontroladores más utilizados en la actualidad para la realización de proyectos sencillos.

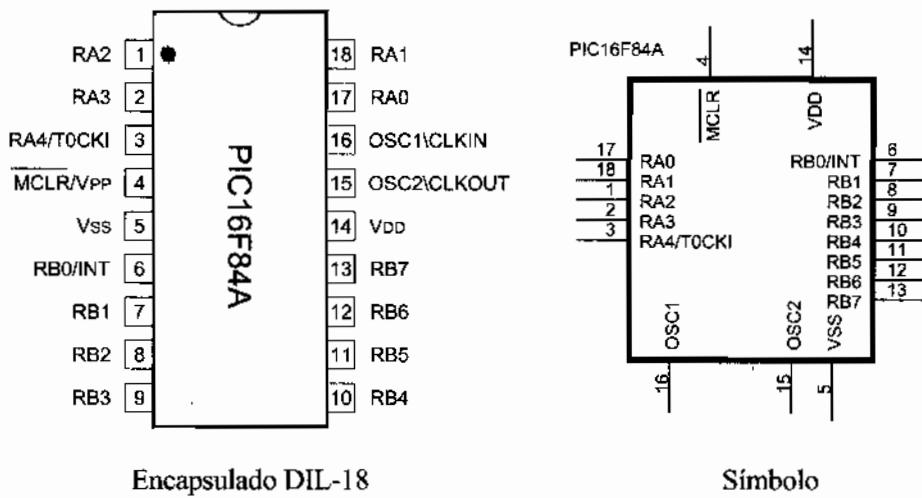


Figura 1-1 Microcontrolador PIC16F84A

El microcontrolador PIC16F84 puede trabajar con una frecuencia máxima de 10 MHz. La versión avanzada PIC16F84A-20 puede llegar hasta los 20 MHz. Todos los circuitos planteados en este libro se han realizado con el PIC16F84A-04 (4 MHz máx).

En la figura 1-2 se describe un ejemplo de aplicación. Se trata del entrenador básico que se va a utilizar en los primeros temas de este libro para el aprendizaje de su manejo y programación. Este circuito se explica a continuación.

## 1.2 ALIMENTACIÓN DE UN PIC16F84

Normalmente el microcontrolador PIC16F84 se alimenta con 5 voltios aplicados entre los pines  $V_{DD}$  y  $V_{SS}$  que son, respectivamente, la alimentación y la masa del chip.

La figura 1-2 describe un circuito de alimentación que obtiene los 5 voltios a partir de una tensión continua de 12 voltios y de al menos 1 amperio. Este circuito se basa en el popular regulador de tensión 7805. Dispone de un diodo a la entrada para protegerlo en el caso que se aplicaran tensiones con la polaridad invertida. El condensador C4 reduce considerablemente el rizado de la tensión de entrada que finalmente el regulador 7805 se encarga de estabilizar a los 5 voltios de alimentación de todo el entrenador. Por último dispone de un diodo LED indicador de encendido.

popular, el  
debido a sus  
s utilizados

El consumo de corriente para el funcionamiento del microcontrolador depende de la tensión de alimentación, de la frecuencia de trabajo y de las cargas que soporten sus salidas, siendo del orden de unos pocos miliamperios.

El circuito de alimentación del microcontrolador debe tratarse como el de cualquier otro dispositivo digital, debiendo conectarse un condensador de desacoplo de unos 100 nF lo más cerca posible de los pines de alimentación.

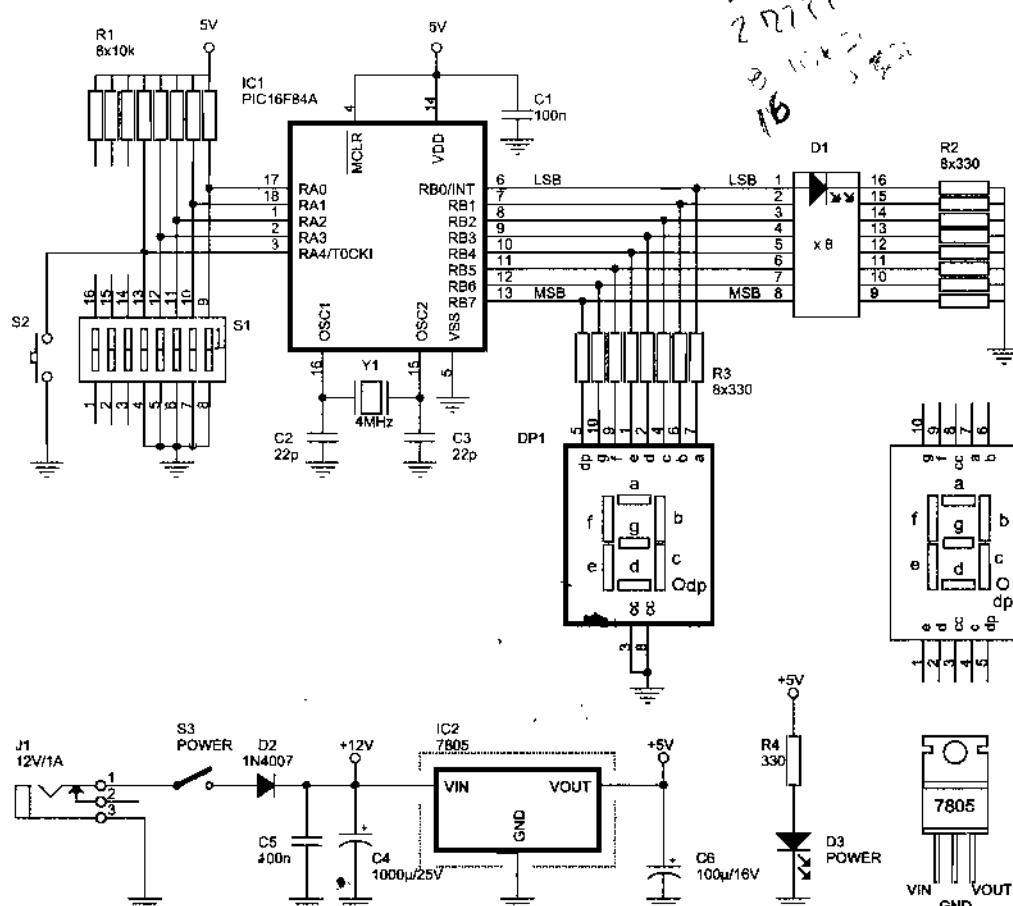


Figura 1-2 Entrenador para aprendizaje del microcontrolador PIC16F84

### 1.3 PUERTOS DE ENTRADA/SALIDA

El microcontrolador se comunica con el mundo exterior a través de los **puertos**. Estos están constituidos por líneas digitales de entrada/salida que trabajan entre 0 y 5 V. Los puertos se pueden configurar como entradas para recibir datos o como salidas para gobernar dispositivos externos.

El PIC16F84 tiene dos puertos, tal como se ilustra en la figura 1-2:

- El **Puerto A** con 5 líneas, pines RA0 a RA4.
- El **Puerto B** con 8 líneas, pines RB0 a RB7.

Cada línea puede ser configurada como entrada o como salida, independientemente unas de otras, según se programe. Así, por ejemplo, en el circuito de la figura 1-2 el Puerto A es configurado como entrada para leer los interruptores y el Puerto B es configurado como salida para activar la barra de diodos LEDs y el display de siete segmentos.

Las líneas son capaces de entregar niveles TTL cuando la tensión de alimentación aplicada en  $V_{DD}$  es de 5V. La máxima capacidad de corriente de cada una de ellas es:

- 25 mA, cuando el pin está a nivel bajo, es decir, cuando consume corriente (modo *sink*). Sin embargo, la suma de las intensidades por las 5 líneas del Puerto A no puede exceder de 80 mA, ni la suma de las 8 líneas del Puerto B puede exceder de 150 mA.
- 20 mA, cuando el pin está a nivel alto, es decir, cuando proporciona corriente (modo *source*). Sin embargo, la suma de las intensidades por las 5 líneas del Puerto A no puede exceder de 50 mA, ni la suma de las 8 líneas del Puerto B puede exceder de 100 mA.

## 1.4 OSCILADOR

Todo microcontrolador requiere de un circuito que le indique la velocidad de trabajo, es el llamado **oscilador o reloj**. Éste genera una onda cuadrada de alta frecuencia que se utiliza como señal para sincronizar todas las operaciones del sistema. Este circuito es muy simple pero de vital importancia para el buen funcionamiento del sistema. Generalmente todos los componentes del reloj se encuentran integrados en el propio microcontrolador y tan solo se requieren unos pocos componentes externos, como un cristal de cuarzo o una red RC, para definir la frecuencia de trabajo.

En el PIC16F84 los pines **OSC1/CLKIN** y **OSC2/CLKOUT** son las líneas utilizadas para este fin. Permite cinco tipos de osciladores para definir la frecuencia de funcionamiento:

- **XT.** Cristal de cuarzo.
- **RC.** Oscilador con resistencia y condensador.
- **HS.** Cristal de alta velocidad.
- **LP.** Cristal para baja frecuencia y bajo consumo de potencia.
- **Externa.** Cuando se aplica una señal de reloj externa.

### 1.4.1 Oscilador XT

Es el más utilizado y está basado en el oscilador a cristal de cuarzo o en un resonador cerámico. Es un oscilador estándar que permite una frecuencia de reloj muy estable comprendida entre 100 kHz y 4 MHz.

La figura 1-2 muestra la conexión típica. En muchos proyectos se utiliza un cristal de 4 MHz. El cristal debe ir acompañado de dos condensadores entre 15 y 33 pF.

Si se comprueba con un osciloscopio la señal en el pin **OSC2/CLKOUT**, se debe visualizar una onda senoidal de igual frecuencia que la del cristal utilizado.

### 1.4.2 Oscilador RC

Es un oscilador de bajo coste formado por una red RC (figura 1-3). Su principal inconveniente es la baja precisión, pero como contrapartida está su bajo precio, que lo hace interesante para muchas aplicaciones en las que no importa la exactitud de tiempos.

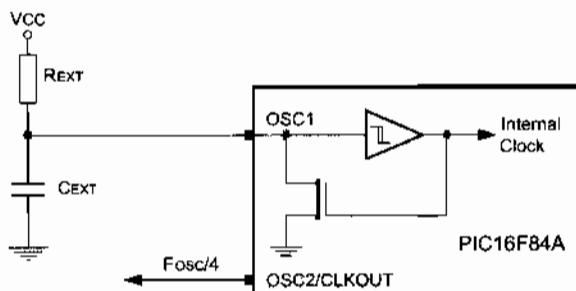


Figura 1-3 Configuración del oscilador RC

Los valores recomendados por el fabricante para este tipo de oscilador son:  $5 \text{ k}\Omega \leq R_{\text{ext}} \leq 100 \text{ k}\Omega$  y  $C_{\text{ext}} > 20 \text{ pF}$ . Los valores de frecuencia correspondiente para cada valor de condensador y resistencia se reflejan en la tabla 1-1.

La frecuencia del oscilador dividida por cuatro, está disponible en el pin **OSC2/CLKOUT** y puede ser usada para sincronizar otros circuitos.

### 1.4.3 Osciladores HS y LP

El oscilador de cristal o resonador de alta velocidad HS (*High Speed Crystal/Resonator*) trabaja a una frecuencia comprendida entre 4 MHz y 20 MHz para el PIC16F84A.

<b>C<sub>EXT</sub></b>	<b>R<sub>EXT</sub></b>	<b>FRECUENCIA</b>
20 pF	5 k	4,61 MHz
	10 k	2,66 MHz
	100 k	311 kHz
100 pF	5 k	1,34 MHz
	10 k	756 kHz
	100 k	82,8 kHz
300 pF	5 k	428 kHz
	10 k	243 kHz
	100 k	26,2 kHz

Tabla 1-1 Frecuencia del oscilador RC para diferentes valores de componentes

El oscilador de cristal de cuarzo o resonador cerámico de baja potencia LP (*Low Power Crystal*) es un oscilador de bajo consumo. Su cristal o resonador está diseñado para trabajar con frecuencias comprendidas entre 32 kHz y 200 kHz.

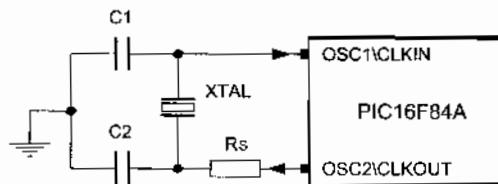


Figura 1-4 Oscilador configuración HS, XT y LP

El circuito para cualquiera de las configuraciones HS, LP y XT es el mismo (figura 1-4). El valor de los condensadores C1 y C2 depende del cristal o resonador según unas tablas que facilita el fabricante. La resistencia R<sub>S</sub> sólo es necesaria para algunas versiones del tipo HS.

#### 1.4.4 Utilizando una señal de reloj externa

Esta posibilidad suele ser utilizada para hacer funcionar varios microcontroladores a partir de una única señal de reloj (figura 1-5). La frecuencia del oscilador dividida por cuatro, está disponible en el pin OSC2/CLKOUT. Se utiliza en pocas ocasiones

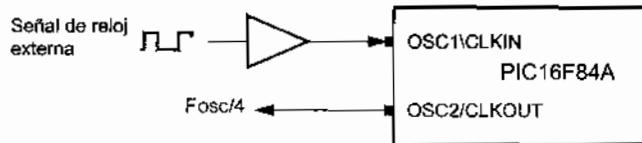
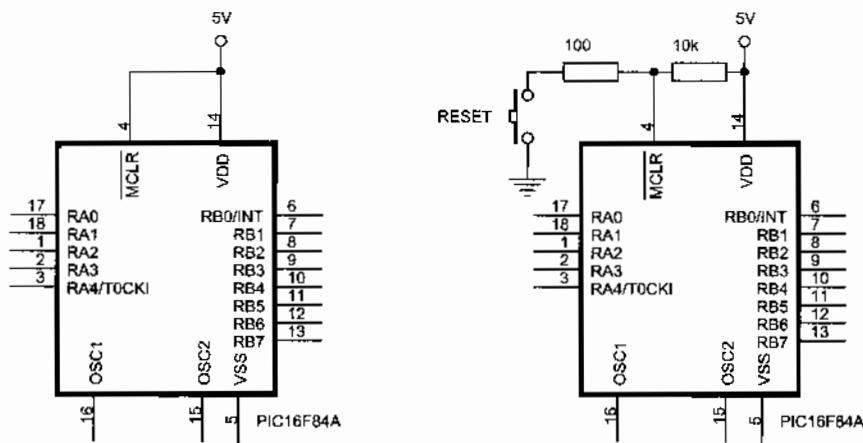


Figura 1-5 Circuito para señal de reloj externa

## 1.5 RESET

El llamado **reset** en un microcontrolador provoca la reinicialización de su funcionamiento, un “comienzo a funcionar desde cero”. En este estado, la mayoría de los dispositivos internos del microcontrolador toman un estado conocido.

En los microcontroladores se requiere un pin de reset para reiniciar el funcionamiento del sistema cuando sea necesario. El pin de reset en los PIC se denomina **MCLR** (*Master Clear*) y produce un reset cuando se le aplica un nivel lógico bajo.



A) TÍPICA CONEXIÓN DEL PIN MCLR.

B) RESET MEDIANTE PULSADOR EN PIN MCLR.

Figura 1-6 Algunas conexiones para el pin MCLR

Para tener un control sobre el reset del sistema, se puede conectar un pulsador tal como se muestra en la figura 1-6(B) y conseguir un reset manual llevando momentáneamente el pin MCLR a masa cada vez que se presiona el pulsador. El reset permanecerá mientras tengamos el pulsador presionado y no comenzará la secuencia de arranque hasta que no lo liberemos, suministrando así un nivel lógico “1” al pin MCLR. El fabricante recomienda conectar en serie con el pulsador una resistencia de 50 a 100 Ω.

El PIC16F84 también permite el llamado **Power-On Reset (POR)**, que proporciona un reset al microcontrolador en el momento de conectar la fuente de alimentación. El PIC dispone de un temporizador denominado **Reset PWRT (Power-up Timer)**, que proporciona un retardo de 72 ms desde el momento de la conexión a la alimentación; un reset se mantiene durante este tiempo, garantizando que  $V_{CC}$  alcance un nivel aceptable de tensión para un arranque correcto del sistema. Para utilizar este tipo de reset, hay que conectar el pin MCLR al positivo de la alimentación (figuras 1-6). Además, hay que programarlo así durante el proceso de grabación. Con ésto se evita utilizar las tradicionales redes RC externas de otros microcontroladores.

El PIC16F84 permite otras causas de reset que serán explicadas en capítulos posteriores.

## 1.6 MONTAJE DEL ENTRENADOR

Una vez analizado el entrenador para el aprendizaje del microcontrolador PIC16F84 descrito en la figura 1-2 se puede pasar a su montaje bien sobre placa de prototipos *Protoboard*, sobre una placa *wrapping*, circuito impreso, o cualquier otro soporte. En el montaje hay que respetar las siguientes normas:

- Comprobar todos los componentes que sea posible antes de su montaje.
- El PIC se situará de manera tal que sea fácilmente extraíble, preferiblemente en el lateral derecho de la placa (izquierdo para zurdos), de modo que no pasen cables por encima.
- Es recomendable alojar el microcontrolador en un zócalo de pines torneados para no doblar las patillas y evitar que se rompan.
- Los indicativos de menor peso de los arrays de diodos LEDs e interruptores se colocarán a la derecha.
- Los cables deben ser lo más corto posible. No se deben utilizar cables contiguos del mismo color.
- Hay que respetar el rojo para el positivo de la alimentación y el negro para el negativo.
- Una vez terminado todo, debe procederse a la comprobación de nuevo de todas las conexiones. No se debe temer el "perder" diez minutos en esta fase, ya que después puede ahorrar gran cantidad de tiempo en averías.

La tensión de alimentación de 12 V de tensión continua se puede obtener con un alimentador que puede adquirirse fácilmente en cualquier tienda de electrónica o en un hipermercado. Si lo desea el lector puede montarlo a partir del esquema de la figura 1-7.

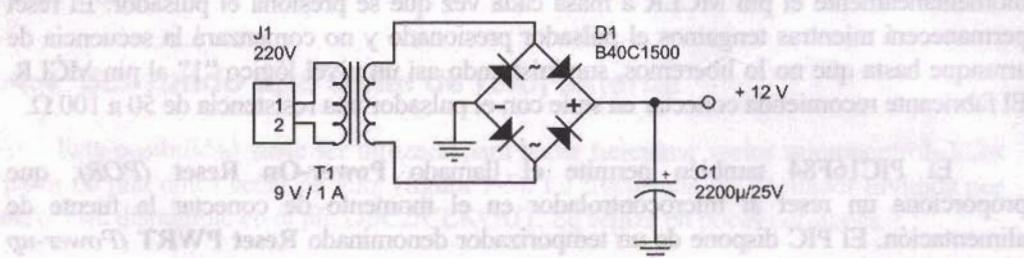


Figura 1-7 Esquema de un alimentador de 12 V de tensión continua

esque  
salida  
para  
capít

cual  
versi  
enco  
www  
com

práct  
este  
los d

2.1

circu  
salida  
1,2 y

s en capítulos

microcontrolador  
sobre placa de  
cualquier otro

ntaje.

riblemente en el  
no pasen cables

s torneados para

interruptores se

ables contiguos

el negro para el

nuevo de todas  
sta fase, ya que

obtener con un  
trónica o en un  
la figura 1-7.

inua

## CAPÍTULO 2

# PERIFÉRICOS BÁSICOS

Las prácticas de los primeros capítulos se realizarán sobre el entrenador del esquema de la figura 1-2 en el que se utiliza como entrada unos interruptores y cuya salida se aplica a unos diodos LEDs. Es posible que el lector desee adaptar estas prácticas para su proyecto utilizando otros tipos de periféricos como los explicados en este capítulo.

En este libro se ha procurado utilizar componentes fácilmente localizables en cualquier tienda de electrónica. Si el lector desea buscar las características técnicas de versiones concretas, o adquirir componentes que en su tienda habitual no le es posible encontrar, puede intentarlo en las siguientes direcciones de Internet: [www.amidata.es](http://www.amidata.es), [www.telkron.es](http://www.telkron.es) o [www.farnell.com](http://www.farnell.com). Las características técnicas de todos los componentes utilizados también se facilitan en el CD-ROM que acompaña a esta obra.

Si el lector está impaciente por comenzar a programar el microcontrolador y las prácticas las va a realizar sobre el entrenador de la figura 1-2, puede por ahora saltarse este capítulo sin que sufra la continuidad del libro y volver a él conforme vaya utilizando los distintos componentes y circuitos que describimos en este tema.

### 2.1 DIODO LED

El diodo LED es un dispositivo que permite comprobar el funcionamiento de los circuitos de forma cómoda mediante la emisión de luz. Es barato y fácil de conectar a la salida de un microcontrolador. Se polariza en directo con una tensión en extremos entre 1,2 y 2,2 V, según modelo, y sólo requiere de 5 a 30 mA para su encendido.

El PIC16F84 es capaz de gobernar directamente diodos LED de dos formas distintas, tal como se indica en la figura 2-1:

- Conectando el cátodo del diodo a la salida del microcontrolador y el ánodo al positivo de la alimentación a través de una resistencia limitadora, como el diodo D1 de la figura 2-1. En este caso, el LED se ilumina con un nivel bajo de salida (0 V).
- Conectando el ánodo del diodo a la salida del microcontrolador a través de una resistencia limitadora y el cátodo a masa, como el diodo D2 de la figura 2-1. En este caso, el LED se ilumina con un nivel alto de salida (5 V).

La resistencia limita el valor de la corriente a un valor adecuado para iluminar el LED. Debe tener un valor comprendido entre 220 y 330 Ω. En la figura 2-1 se ha elegido 330 Ω que limita la corriente a un valor de unos 10 mA que proporciona una luminosidad suficiente para la mayoría de las aplicaciones. Si al lector le gusta que emita más luz, puede bajar su valor a 220 Ω.

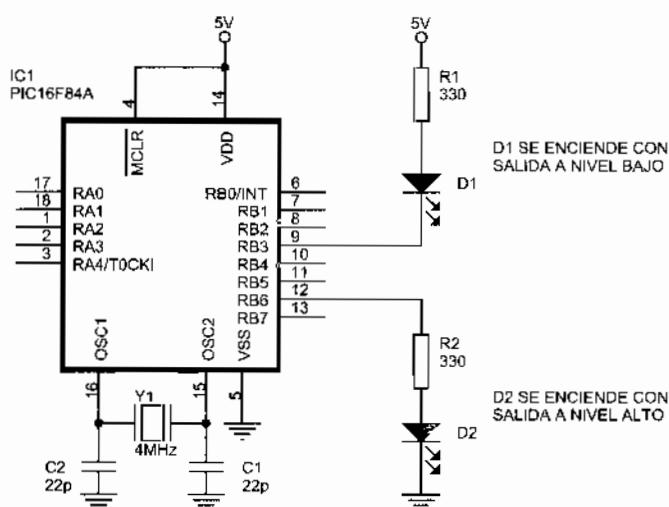


Figura 2-1 Formas de conectar un LED a un microcontrolador

Los diodos LED también se pueden encontrar encapsulados en barras de 8 ó 10 LED (figura 2-2) como los que se conectan a la salida del Puerto B de la figura 1-2.



Figura 2-2 Barra de 10 diodos LED

de dos formas

or y el ánodo al  
adadora, como el  
un nivel bajo de

a través de una  
de la figura 2-1.  
/).

para iluminar el  
2-1 se ha elegido  
una luminosidad  
e emita más luz,

## 2.2 INTERRUPTORES Y PULSADORES

Estos dispositivos permiten introducir un nivel lógico "0" ó "1" según la posición en que se encuentren, "cerrado" o "abierto".

La lectura del estado de interruptores y pulsadores es muy simple, basta con conectar estos dispositivos entre una entrada y masa, tal como se indica en la figura 2-3 y forzar la entrada a un nivel lógico alto (5 V) mediante una resistencia de Pull-Up de unos 10 k.

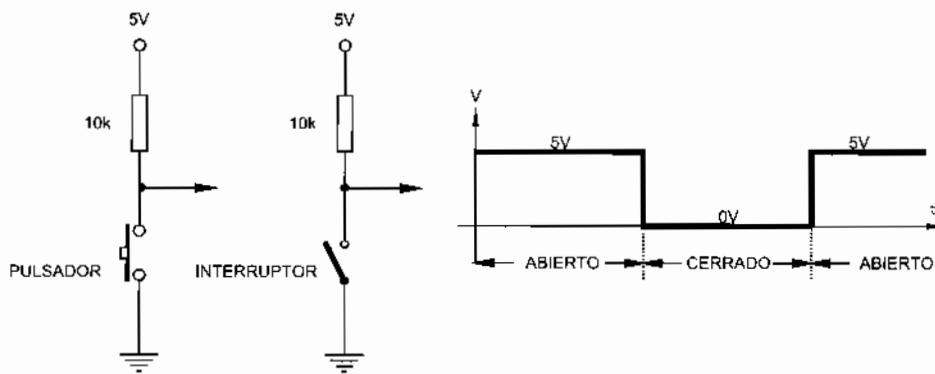


Figura 2-3 Niveles de tensión ideales en bornes de un interruptor, pulsador, etc.

Mientras el dispositivo está abierto, la entrada mantiene una tensión de 5 V que corresponde a un nivel lógico "1". Cuando se cierra, la entrada pasa a valer 0 V correspondiente al nivel lógico "0".

Hay muchos tipos de comutadores, finales de carrera, detectores y sensores digitales con un funcionamiento similar a los pulsadores e interruptores.

## 2.3 ENTRADAS DIGITALES CON OPTOACOPLADORES

En algunos proyectos es necesario utilizar como entrada señales de alta tensión o señales relacionadas con la tensión de la red eléctrica. Estas tensiones no se pueden aplicar directamente al microcontrolador y es necesario aislar eléctricamente el circuito mediante un optoacoplador con un montaje como el de la figura 2-4.

El 4N25 es un popular optoacoplador, en cuya cápsula DIL-6 se encierra un diodo LED y un fototransistor. Es fácil deducir su funcionamiento:

- Cuando se aplica una tensión  $V_{IN}$ , circula una corriente por el LED del optoacoplador emitiendo un haz de luz que incide sobre el transistor y lo satura. En este caso a la entrada del microcontrolador se aplica un nivel bajo, igual que cuando estaba cerrado el interruptor de la figura 2-3.

- Cuando no se aplica tensión alguna el LED está apagado bloqueando el transistor. En la entrada se aplica un nivel alto igual que cuando estaba abierto el interruptor del circuito de la figura 2-3.

La tensión directa en extremos del LED 4N25 en conducción es de 1,2 V y para que se ilumine hay que hacer circular una corriente de unos 5 mA. La resistencia en serie con el LED debe permitir que circule esta intensidad, así, por ejemplo para una tensión  $V_{IN}$  de 24 voltios se conectaría una resistencia con un valor comercial de 4k7 (o mejor 3k9) ya que:

$$R1 = \frac{V_{IN} - 1,2V}{5mA} = \frac{24V - 1,2V}{5mA} = 4,56k$$

El diodo LED soporta una tensión máxima inversa de sólo 3 V. Para aplicaciones en corriente alterna hay que conectar en paralelo con el LED un diodo de protección en inverso o, mejor, utilizar un optoacoplador que ya lo lleve integrado como el HJ1A1.

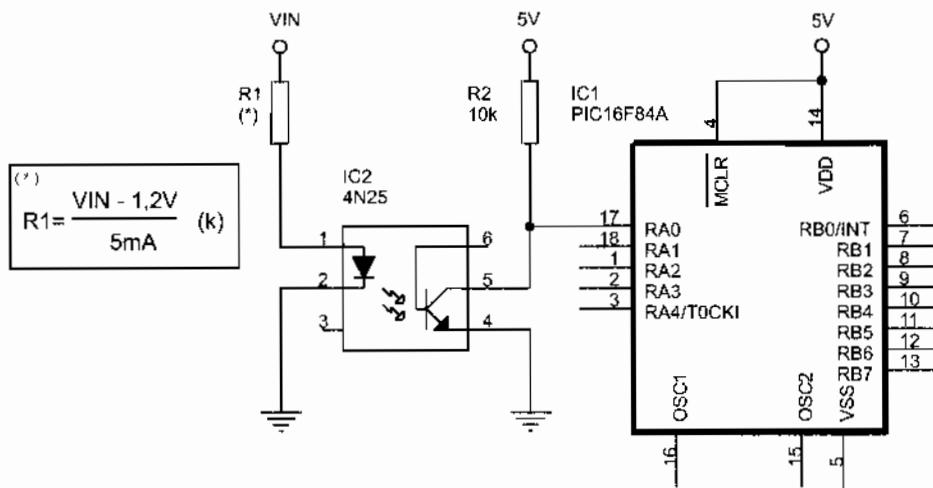


Figura 2-4 Gobierno de una entrada mediante optoacoplador 4N25

Puede comprobarse que los dos circuitos están eléctricamente aislados. La única comunicación entre ambos es la luz que emite el LED.

## 2.4 DISPLAY DE SIETE SEGMENTOS

El display de siete segmentos es un periférico digital de salida que se utiliza para representar valores numéricos (figura 2-5). Cada display consta de 7 segmentos y un punto decimal, todos ellos son diodos LEDs. Estos diodos se pueden encontrar en dos configuraciones posibles, según los pines que tengan unidos: ánodo común o cátodo

común. La conexión depende de la configuración.

La configuración más sencilla es la conexión en común.

El resultado es una salida digital de 7 bits.

IC1  
PIC16F84A

17  
18  
1  
2  
3  
RAD  
RA1  
RA2  
RA3  
RA4  
OSC1

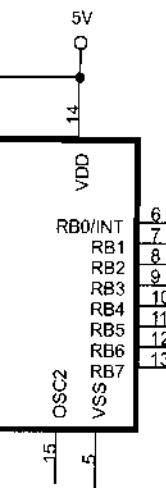
C2  
22p  
OSC1  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8  
7  
6  
VDD  
RB7  
RB6  
RB5  
RB4  
RB3  
RB2  
RB1  
RB0/INT  
RA7  
RA6  
RA5  
RA4/T0CKI  
RA3  
RA2  
RA1  
RA0  
MCLR  
OSC2  
VSS  
5  
16  
15  
14  
13  
12  
11  
10  
9  
8<br

pagado bloqueando el cuando estaba abierto

ción es de 1,2 V y para . La resistencia en serie ejemplo para una tensión mercial de 4k7 (o mejor

56k

3 V. Para aplicaciones diodo de protección en o como el H11A1.



lador 4N25

nte aislados. La única

común. Los segmentos de los displays se controlan directamente mediante el Puerto B y según el tipo de display la conexión al microcontrolador varía según indica la figura 2-5.

La figura 11-2 del capítulo “Manejo de tablas” muestra las posibles combinaciones de salida para la representación de números, letras y algunos signos en estos displays.

El principal problema de los displays de 7 segmentos es que requieren de muchas líneas para su control. El capítulo 25 explica el circuito integrado SAA1064 que permite el gobierno de 4 displays con un escaso número de líneas.

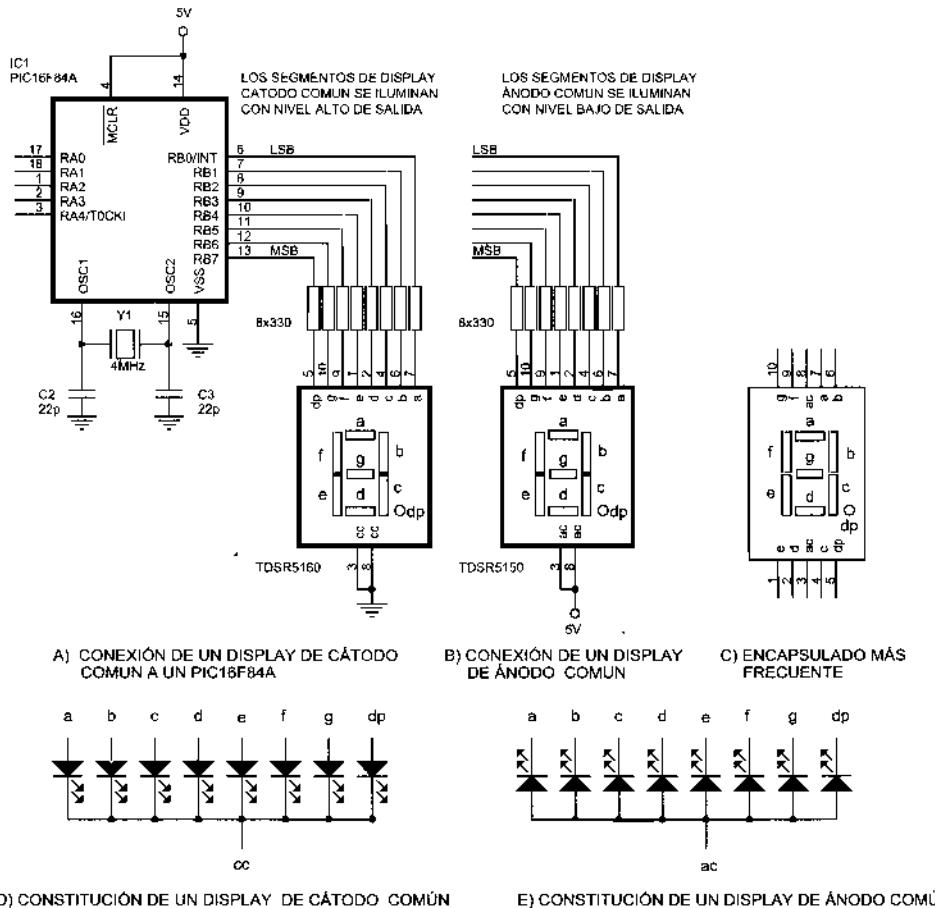


Figura 2-5 El display de siete segmentos

## 2.5 CONTROLANDO CARGAS A 230 V

Las explicaciones de los primeros capítulos se realizarán sobre el entrenador de la figura 1-2. Es probable que el lector desee probar para su proyecto algunas aplicaciones

que se exponen en estos capítulos, sobre circuitos reales con cargas de 230 V (bombillas, calefactores, motores, etc). A continuación se detallan varios circuitos apropiados donde el PIC16F84 controla una carga alimentada con los 230 V de la red eléctrica.

Es importante advertir que estos circuitos trabajan sobre la red eléctrica de 230V. Cualquier error, además de ocasionar daños serios en el circuito, puede provocar lesiones personales, luego hay que ser muy cauto durante el montaje y revisarlo concienzudamente. Tenga especial cuidado en aislar bien todas aquellas conexiones o cables cuyo contacto con la piel humana pueda producir descargas eléctricas, sin olvidarse tampoco de la parte metálica de los componentes de potencia.

### 2.5.1 Control con relé

La utilización de un relé es la forma más sencilla para gobernar dispositivos a partir de una salida del puerto, como muestra la figura 2-6. Un par de transistores Darlington son necesarios para controlar el relé.

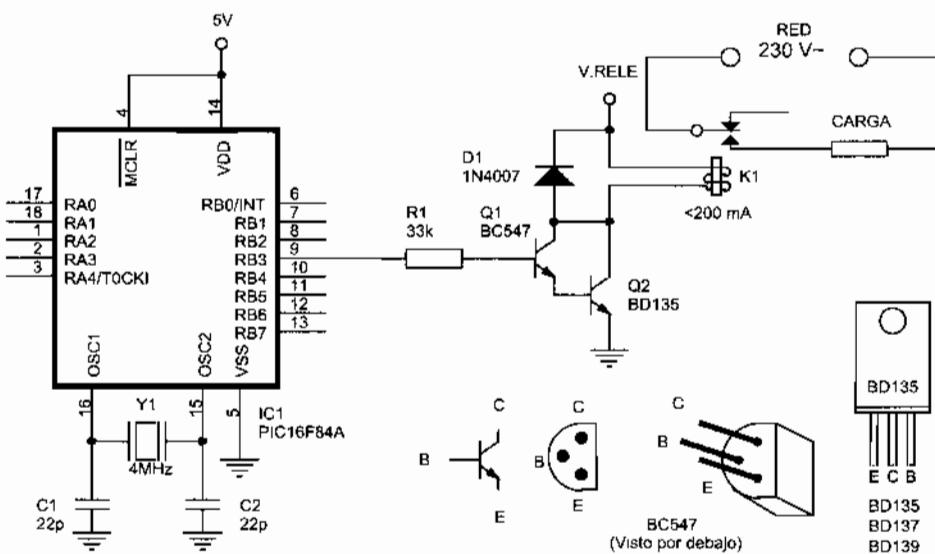


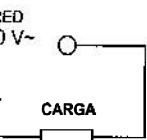
Figura 2-6 Gobierno de una carga de potencia a través de un relé

Cuando la salida del microcontrolador proporciona un nivel alto a la base del Darlington, pasa a conducción y activa el relé que, al cerrar sus contactos, puede controlar una potencia mayor en la carga. Este circuito también aísla eléctricamente la carga del microcontrolador. El valor de la potencia a controlar depende de los contactos del relé y varía mucho según el modelo, aunque casi todos ellos pueden soportar más de 5 Amperios.

30 V (bombillas).  
apropiados donde  
ica.

eléctrica dc de 230V.  
provocar lesiones  
aje y revisarlo  
las conexiones o  
as eléctricas, sin

alar dispositivos a  
ir de transistores



Es indispensable conectar un diodo en paralelo con la bobina del relé, tal como muestra la figura 2-6, como protección frente a los picos de fuerza contraelectromotriz producidos por la carga inductiva de la bobina en el momento de la conmutación.

Para controlar un cierto número de relés a partir del mismo microcontrolador, se puede utilizar un circuito integrado especializado tal como el ULN2003, figura 2-7. Este chip dispone de siete circuitos inversores realizados internamente con circuitos Darlington, que aguantan una tensión máxima de 50 V y pueden alimentar cargas de hasta 500 mA, incorpora también los indispensables diodos de protección.

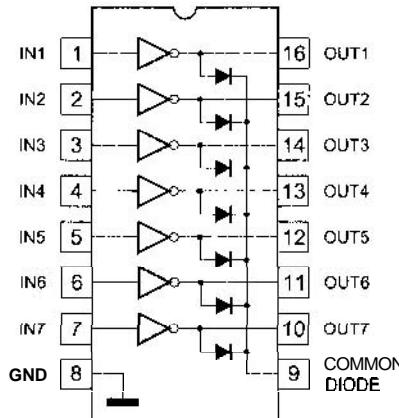


Figura 2-7 Driver ULN2003

La figura 2-8 describe el esquema típico de conexión, donde el ULN2003 alimenta las bobinas de siete relés.

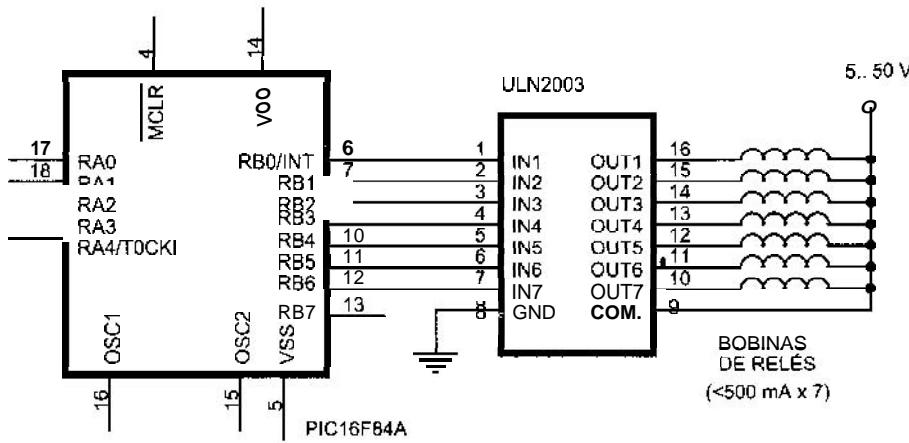


Figura 2-8 Circuito típico de gobierno de varios relés con ULN2003

## 2.5.2 Control con relé miniatura en cápsula DIL

Para cargas de hasta 10 W es mejor utilizar relés de láminas encapsulados en DIL, que necesitan una menor intensidad de activación, aunque sus contactos no permiten activar cargas grandes. La figura 2-9 muestra un ejemplo de aplicación donde sólo es necesario un transistor para gobernar el relé. Normalmente estos relés llevan incorporados dentro de la cápsula el diodo de protección, como se puede apreciar en la figura, para los modelos que no lo llevan es necesario conectarlo en el circuito.

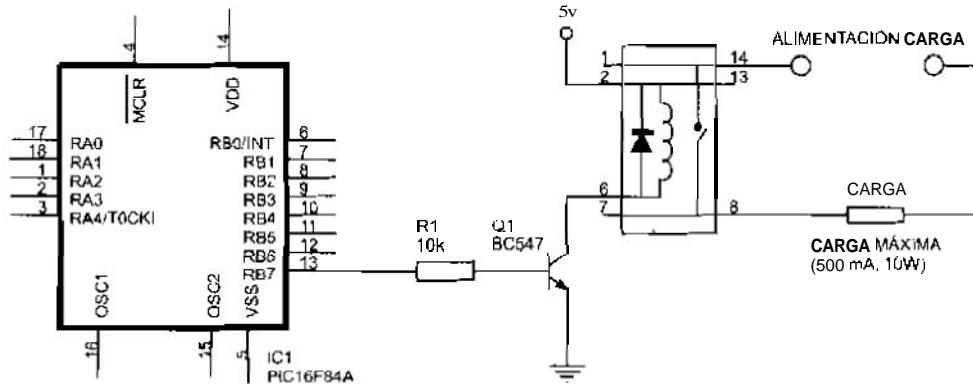


Figura 2-9 Gobierno de pequeñas cargas u través de un relé de láminas en cápsula DIL

## 2.5.3 Control mediante fototriac

En el circuito de la figura 2-10 los contactos del relé son sustituidos por un fototriac, cuyo funcionamiento es similar al de un interruptor controlado por luz.

El necesario aislamiento entre el microcontrolador y la carga de 230V se hace mediante un optoacoplador MOC3041, que es un circuito integrado que incluye un LED que controla al fototriac. Este dispositivo está especialmente diseñado para usarse como interfaz de sistemas lógicos con equipos que tienen que alimentarse con los 230 V de la red eléctrica. Sus características más significativas son:

- Incorpora un pequeño y económico encapsulado DIP 6.
- Su tensión de aislamiento de 7500 V garantiza un perfecto aislamiento entre la red eléctrica y el microcontrolador.
- Es capaz de proporcionar hasta 100 mA, que le permitiría alimentar directamente pequeñas cargas de hasta 20 W.
- Su fototriac interno permite el control de la casi totalidad de los grandes triacs, lo que no sería posible si se utilizara un fototransistor ordinario.
- Cuenta con un detector de paso por cero interno, lo que permite economizar un número no despreciable de componentes externos.

Cuando el instante de la conmutación de un triac no coincide con un cruce por cero de la tensión de la red el cambio repentino en la corriente produce un ruido eléctrico de alta frecuencia que introduce interferencias en la tensión de red que, por ejemplo, puede dar lugar a que señales indeseables aparezcan en la pantalla de un receptor de televisión o que se hagan audibles "chasquidos" en el altavoz de un receptor de radio. Para evitar estos problemas el MOC3041 posee un detector de paso por cero que conmuta al fototriac únicamente cuando la tensión aplicada al mismo pase por cero.

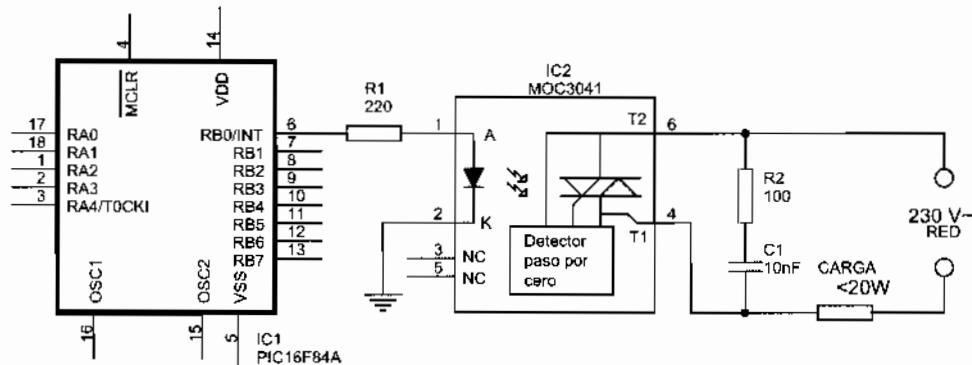


Figura 2-10 El PIC16F84A controlando una carga alimentada a 230 V y menor de 20W

En la figura 2-10, cuando la línea del Puerto B se ponga a nivel alto TTL (5 V) hará circular una corriente de unos 15 mA por el diodo LED del MOC3041, éste emitirá luz, lo que provocará que el fototriac entre en conducción en el siguiente paso por cero de la tensión de red. Una vez que el fototriac entra en conducción, se comporta prácticamente como un interruptor cerrado que enciende la carga. Hay que recordar que el triac se desactiva automáticamente cada vez que la corriente pasa por cero, por lo que es necesario bien redisparar el triac en cada semiperiodo, bien mantenerlo con la señal de control activada durante el tiempo que necesite mantenerse encendida la salida.

Cuando la línea del Puerto B pasa a nivel bajo TTL (0 V) el LED del MOC3041 se apaga. En el siguiente paso por cero de la tensión de red, el triac deja de conducir, comportándose como un interruptor abierto de forma que la carga deja de recibir corriente y se apaga.

La resistencia R1 de 220 Ω conectada al ánodo del LED de entrada al MOC3041 garantiza una circulación de los 15 mA que especifica el fabricante.

La red serie del condensador de 10 nF y la resistencia de 100 Ω en los esquemas de las figuras 2-10 y 2-11 conectada en paralelo con el triac mejora el funcionamiento del circuito para disparos indeseables del triac producidos por los picos bruscos de la tensión de red que se pueden presentar aleatoriamente. Esta red R-C es responsable de que el circuito tenga un cierto consumo, de aproximadamente 0,1 W, aunque esté desactivado. El condensador C1 debe ser capaz de soportar al menos 400 V.

Debido a las características de los triacs la carga debe tener una potencia mínima por debajo de la cual el circuito no funciona. Esta potencia mínima es de aproximadamente 1 W para el circuito de la figura 2-10. El fototriac integrado en el MOC3041 puede conmutar cargas de hasta 100 mA, es decir, 23 W a 230V. Por tanto, esta configuración puede manejar cargas entre 1 y 23 W.

### 2.5.4 Control de potencia con triac

La figura 2-11 describe un circuito típico de control de potencia con triac, donde la carga es conmutada mediante el triac Q1, cuyo funcionamiento es similar al de un interruptor pero controlado por la corriente que circula por su entrada G. A su vez, esta entrada es gobernada por el fototriac del MOC3041. Este circuito puede controlar cargas con potencias entre 10 y 1500 W o mayores dependiendo del triac utilizado.

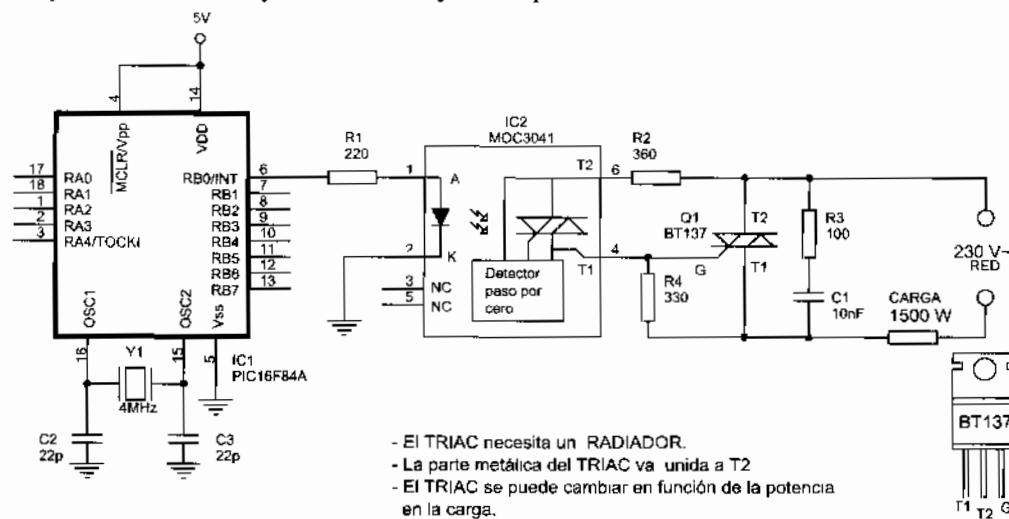


Figura 2-11 El PIC16F84A controlando una carga alimentada a 230 V

Cuando el fototriac del MOC3041 entra en conducción, drena la corriente suficiente a través del terminal de gobierno de Q1 como para conseguir que éste entre en conducción. La resistencia R2 de 360 Ω limita la corriente que pasa por el fototriac para evitar que supere su valor máximo de 100 mA.

El triac utilizado en este ejemplo es un BT137-400 de 400V/8A, con el que se puede controlar hasta cargas de 1500 W. Este triac se puede cambiar en función de la potencia en la carga por alguno de los siguientes modelos: BT136 (4 A), BT137 (8 A), BT138 (12 A), BT139 (16 A) o sus equivalentes. Para cargas fuertemente inductivas es conveniente elegir un triac que aguante hasta 600 V, tal como el BT137-600.

cia mínima  
ma es dc  
grado en el  
. Por tanto.

Para conseguir estas potencias, el triac debe ir montado sobre un buen radiador de calor, de forma que el semiconductor se refrigerue adecuadamente. A la hora de poner el radiador hay que señalar que la parte metálica del componente suele conectarse al terminal T2, por lo que se debe aislar cuidadosamente el triac del radiador mediante una lámina de mica y un separador de plástico para el tornillo.

## 2.6 ZUMBADOR

En muchos proyectos es necesario indicar mediante una señal audible la ocurrencia de un evento. Para ello normalmente se utiliza un zumbador piezoelectrónico miniatura corno cl de la figura 2-12.

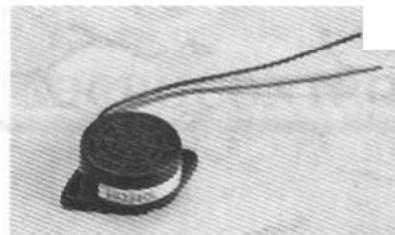


Figura 2-12 Zumbador piezoelectrónico

Un zumbador miniatura funciona con tensiones comprendidas entre 3 y 16 V y su consumo no supera los 10 mA, por lo que puede ser alimentado directamente por la salida de un microcontrolador, tal como se indica en la figura 2-13.

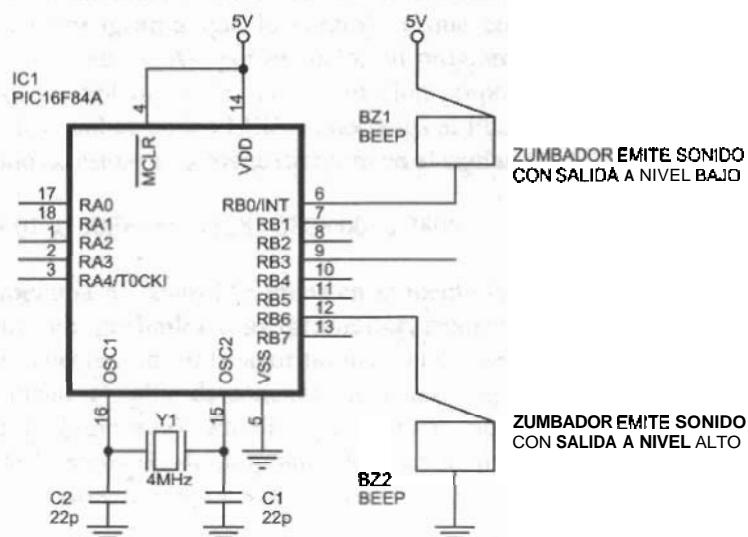


Figura 2-13 Conexión de un zumbador miniatura a un microcontrolador

## GRABACIÓN DE MICROCONTROLADORES PIC

---

### 3.1 GRABACIÓN DE UN MICROCONTROLADOR

Un microcontrolador es un circuito integrado programable que contiene todos los componentes necesarios para controlar el funcionamiento de una tarea determinada, como el control de un teclado de ordenador, una impresora, un sistema de alarma, una lavadora, etc. El microcontrolador dispone de una **memoria de programa** interna donde se almacena el **programa** que lo controla y que consiste realmente en una serie de números hexadecimales. Así por ejemplo, un programa para el entrenador básico de la figura 1-2 que simplemente lea la información proporcionada por los interruptores del Puerto A y la visualice en los LEDs conectados al Puerto B, tendría el siguiente formato (el significado de estos números se discutirá en el capítulo 6):

1683 0186 30FF 0085 1283 0805 0086 2805

El programa de control se graba en la memoria de programa mediante un equipo físico denominado **grabador** o **programador**, siguiendo el esquema de la figura 3-1. El grabador se conecta a un ordenador normalmente a través de un puerto serie COM 1 o COM 2 mediante el cable de conexión adecuado (algunos grabadores utilizan el puerto paralelo de la impresora). En el ordenador se ejecuta un software que controla la grabación de la memoria de programa del microcontrolador. Este proceso se denomina **grabar** o **programar** el microcontrolador.

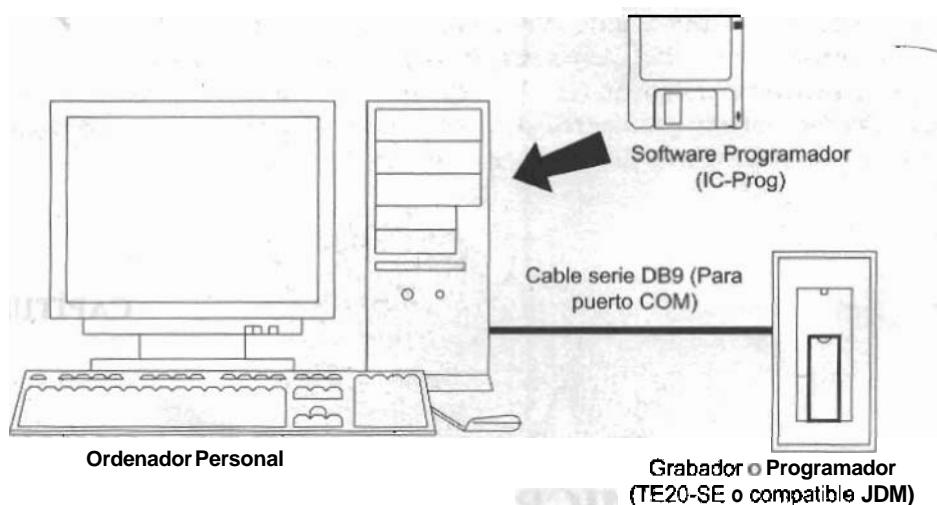


Figura 3-1 Configuración para grabar microcontroladores con medios reducidos

### 3.2 GRABADORES

El **grabador o programador** es el equipo físico donde se procede a grabar la memoria del microcontrolador con las instrucciones del programa de control. Tiene un zócalo libre sobre el que se inserta el circuito integrado a grabar, el cual debe orientarse adecuadamente siguiendo la señal de la cápsula del chip. Hay multitud de grabadores comerciales en el mercado que se pueden adquirir en cualquier tienda de electrónica.

*Microchip* ofrece el grabador PICSTART PLUS, de muy fácil utilización y garantizada fiabilidad respaldada por el fabricante (figura 3-2).

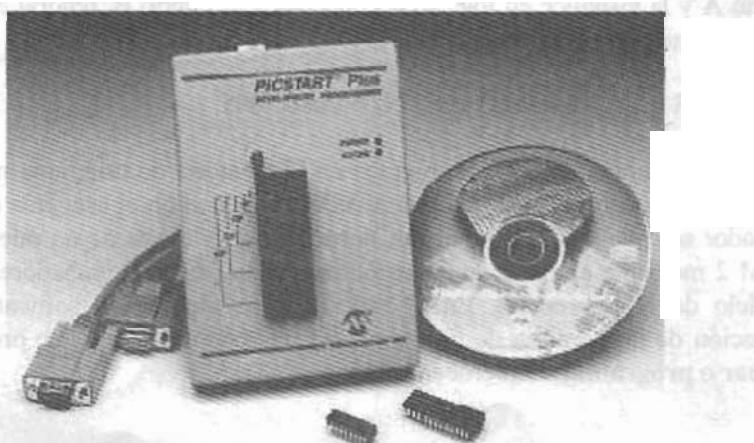


Figura 3-2 Programador PICSTART PLUS de Microchip Technology Inc

En  
microcontr  
múltiples v  
tienda de c  
por sí mism

El p  
diseñador Je



Figura 3-4

En las  
que apenas t  
mucho meno  
muy interesan  
que estos gra  
portátiles sob

En Internet **pueden** localizarse **múltiples grabadores de bajo costo** para microcontroladores PIC. Uno de los más populares **es** el denominado *JDM* y sus múltiples versiones mejoradas, tal como el *TEJO-SE* que se puede adquirir en cualquier tienda *de electrónica* por un precio **muy asequible** (figura 3-3). Si el lector desea montarlo por si mismo, en el apéndice F se proporciona información para ello.

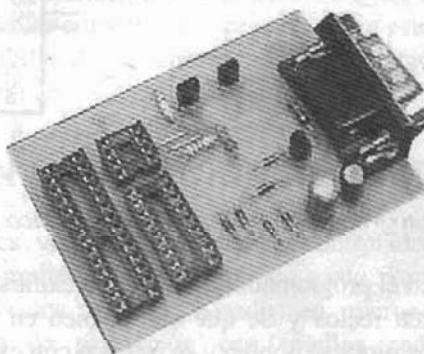


Figura 3-3 Grabador TE20-SE

El programador **JDM** y algunas de sus versiones está descrito en la **Wcb** de su diseñador Jens Dyekjær, [www.jdm.homepage.dk/newpic.htm](http://www.jdm.homepage.dk/newpic.htm).

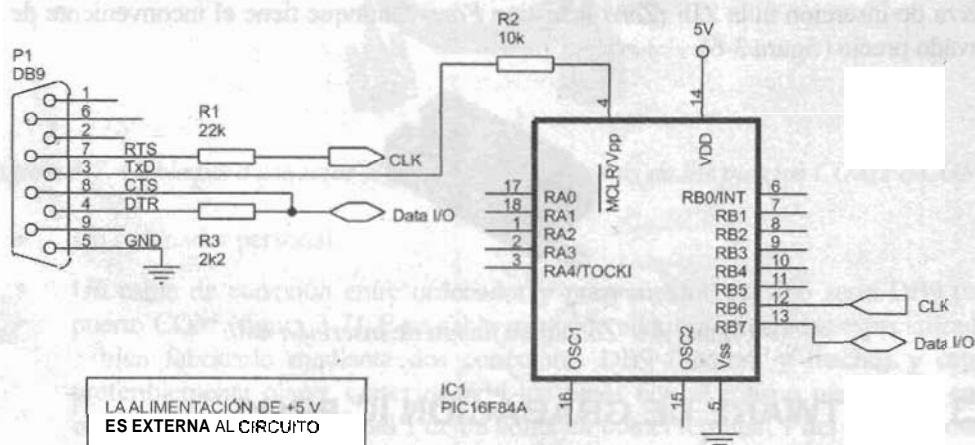
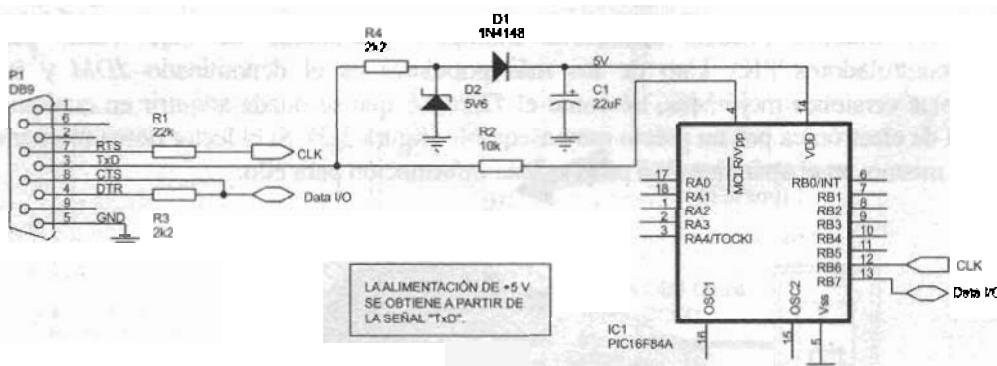


Figura 3-4 Esquema de un grabador compatible JDM básico con alimentación externa

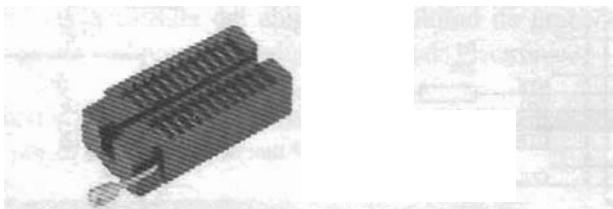
En las figuras 3-4 y 3-5 se describen dos versiones básicas de este programador que apenas requiere componentes. Evidentemente la fiabilidad de este programador es mucho menor que la del PICSTART PLUS, pero su facilidad de construcción lo hace muy interesante para múltiples aplicaciones. Aunque hay que hacer una observación y es que estos grabadores tan básicos no funcionan correctamente en algunos ordenadores, portátiles sobre todo. Además, una conexión incorrecta puede dañar el ordenador.



*Figura 3-5 Esquema de un grabador compatible JDM básico sin alimentación externa*

Al insertar un chip en el programador, hay que cerciorarse de que todos los pines o patillas del dispositivo estén rectos y de que entren bien en el zócalo. Hay que tener mucho cuidado porque estos pines se doblan y se rompen con extrema facilidad.

Cuando se realizan frecuentes reprogramaciones es aconsejable utilizar un zócalo auxiliar entre el microcontrolador y el zócalo del programador, de tal modo que sea los pines del zócalo auxiliar los que sufran las frecuentes inserciones y no los pines del dispositivo. Otra alternativa, es reemplazar el zócalo del programador por un zócalo de fuerza de inserción nula ZIF (*Zero Insertion Force*), aunque tiene el inconveniente de su elevado precio (figura 3-6).



*Figura 3-6 Zócalo de fuerza de inserción nula*

### 3.3 SOFTWARE DE GRABACIÓN IC-PROG

**El IC-Prog** es uno de los software más populares para la grabación de microcontroladores PIC. Permite la programación de muchos dispositivos y está probado con numerosos programadores, entre ellos todos los compatibles con JDM. Es de libre distribución y en la página Web [www.ic-prog.com](http://www.ic-prog.com) se puede descargar y recoger toda la información de uso.

Una vez descargado, la instalación de este software es muy sencilla. basta con descomprimir el fichero *icprog.zip* y seguir el procedimiento usual en Windows. Este archivo consta del fichero *icprog.exe*, que contiene todo el código necesario para su

**funcionamiento**  
utilizar este  
*icprog.sys* de

**En las**  
la última ver:  
ayudar a todo  
profundo de  
programa **IC**

3.4 GR

Una desatollar e sólo ocasionalmente utilizar el material:

## **material:**

Figura 3-71

- Un c  
puer  
o bi  
prefe  
extre  
3 con
  - Un  
elect  
ofrec

funcionamiento, con versiones para cualquier sistema operativo Windows. En caso de utilizar este software con Windows XP, 2000 o NT, es necesario descargar el archivo *icprog.sys* de la misma Web y situarlo en la misma carpeta, junto con el *icprog.exe*.

En las próximas páginas se expondrán las conocimientos básicos para trabajar con la última versión, al cierre de la edición de este libro, *IC-Prog 1.05C*, con la pretensión de ayudar a todos aquéllos que se enfrentan por primera vez a este software. Para un análisis profundo de todas sus posibilidades se remite a la documentación técnica, ya que el propio programa *IC-Prog* tiene un buen sistema de ayuda que permite un rápido aprendizaje.

### 3.4 GRABACIÓN CON MEDIOS REDUCIDOS

Una de las grandes ventajas de los microcontroladores PIC es que permiten desarrollar el proceso de grabación con muy poco gasto. Para aquéllos que desarrollan sólo ocasionalmente proyectos sencillos basados en microcontroladores, es suficiente utilizar el procedimiento de grabación con medios reducidos que se indica a continuación. Según el esquema de la figura 3-1, este procedimiento utiliza el siguiente material:



Figura 3-7 Cable para conectar el programador a través de los puertos COM1 o COM2

- Un ordenador personal.
- Un cable de conexión entre ordenador y programador del tipo serie DB9 para puerto COM (figura 3-7). Este cable se puede adquirir en tiendas especializadas o bien fabricarlo mediante dos conectores DR9 (hembra y macho) y cable, preferiblemente plano, conexionando los pines con el mismo número en cada extremo, es decir el terminal 1 de un conector con el terminal 1 del otro. 2 con 2, 3 con 3, etc.
- Un programador TE20-SF, de adquisición en cualquier tienda de electrónica. En apéndice F y en el CD-ROM que acompaña a este libro se ofrecen planos para su construcción.
- Como software se utiliza el IC-Prog 1.05C, que puede bajarse libremente de la Web [www.ic-prog.com](http://www.ic-prog.com), y que se incluye en el CD-ROM que acompaña a esta obra, gracias a la generosidad de Bonny Gijzen, su autor.

### 3.5 PROCESO DE GRABACIÓN

Antes de nada hay que conectar el programador a uno de los puertos serie COM disponibles en el ordenador formando la estructura indicada en la figura 3-1. A continuación se inserta el microcontrolador PIC en el zócalo del programador respetando la correcta orientación de la cápsula.

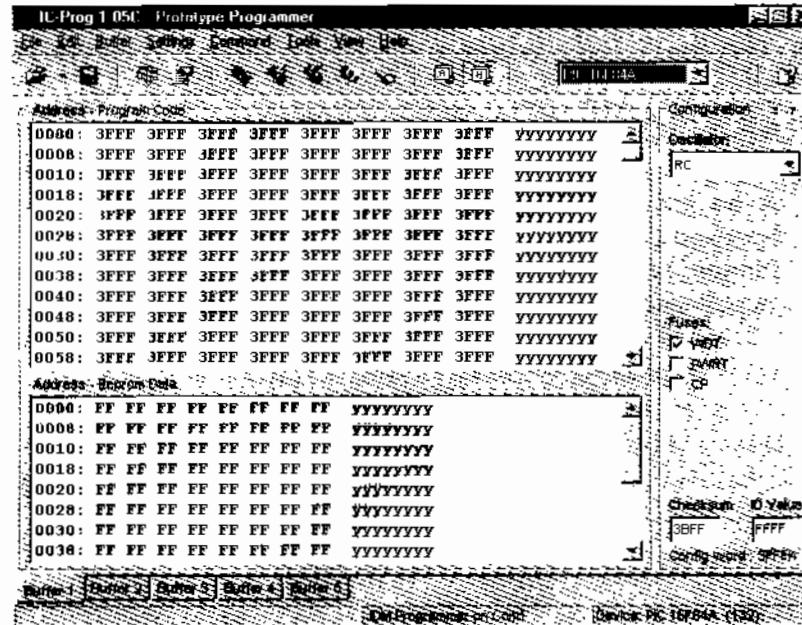


Figura 3-8 Pantalla típica del IC-Prog

Una vez que el programa está correctamente instalado, los pasos a seguir para trabajar con el *IC-Prog 1.05C* son los siguientes. Sugerimos al lector que los vaya probando en su ordenador según se va explicando:

- 1º  Iniciar el programa ejecutando el fichero *icprog.exe* o pulsando sobre el ícono correspondiente. Conviene crear un acceso directo desde el escritorio de Windows para mayor comodidad.
- 2º La primera vez que lo ejecutamos entraremos en una pantalla de presentación de la que se saldrá aceptando todas las opciones por defecto. Si trabaja con Windows 2000 o XP aparecerán unas pantallas de error debido a que todavía no se ha configurado correctamente como se explica en el apartado 3.7. Seguidamente aparece una pantalla en inglés, similar a la figura 3-8, donde se presenta toda la información necesaria para programar el dispositivo. Esta pantalla posee, al menos:

puertos serie COM  
la figura 3-1. A  
ramador respetando

- Un área de código (*Program Code*), donde se almacena la información a grabar. La columna de la izquierda contiene la dirección física de memoria del dispositivo, (*Address*). En el centro del campo se presenta el valor hexadecimal y la columna de la derecha contiene la misma información en código ASCII.
- Un área de configuración (*Configuration*), donde se indica el valor de algunos parámetros necesarios para la correcta grabación.

- 3º Para cambiar el idioma se debe seleccionar en el menú *Setting > Options > Language* y elegir el idioma (figura 3-9).

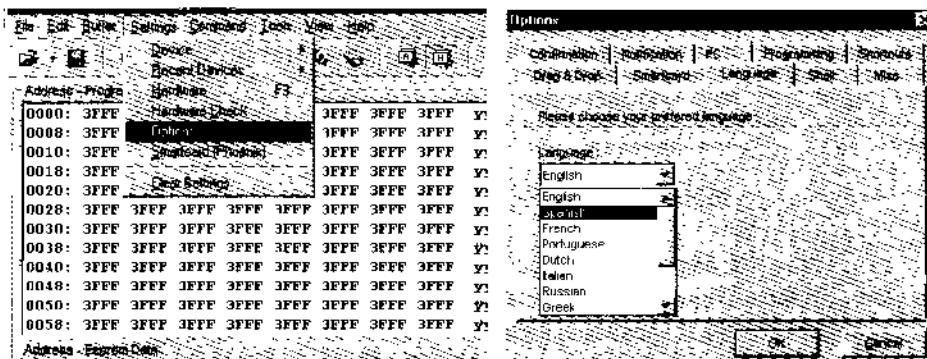


Figura 3-9 Elección del idioma

- 4º Configurar el hardware necesario para programar los microcontroladores PIC, es decir adaptar el IC-Prog al programador utilizado, en el caso que nos ocupa un programador compatible con JDM. Para ello hay que acceder al menú *Ajustes > Tipo hardware*, con lo que aparecerá la pantalla de la figura 3-10, en la que se debe elegir el tipo de programador como *JDM* y seleccionar el puerto serie adecuado (*COM 1 o COM 2*), según lo tenga conectado en el ordenador.

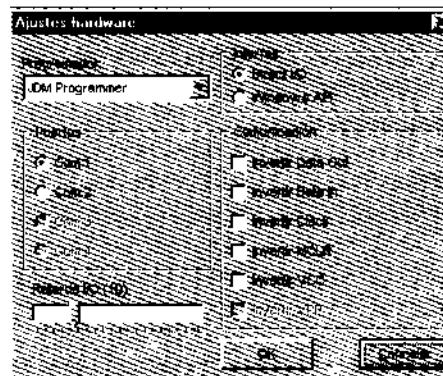


Figura 3-10 Selección del programador

- 5º A continuación se debe seleccionar el dispositivo a grabar, en este caso el microcontrolador PIC16F84A, en el menú *Ajustes > Dispositivo > Microchip PIC > Más > PIC16F84A*, tal como se describe en la figura 3-11.



Figura 3-11 Selección del microcontrolador

El nombre del dispositivo seleccionado aparecerá en una ventana de la barra de herramientas (figura 3-12). Pulsando en la flecha de la ventana se puede elegir cualquiera de los dispositivos soportados por el software *IC-Prog*.

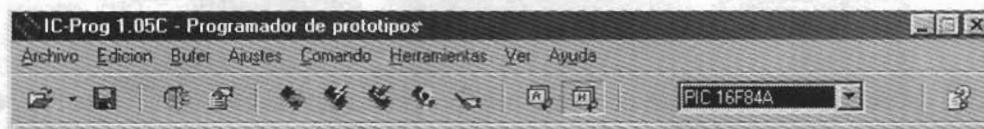


Figura 3-12 El nombre del dispositivo seleccionado aparece en la ventana

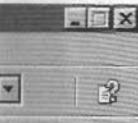
- 6º Elegir el oscilador que va a utilizar el microcontrolador en cuestión (LP, RC, XT, HS). Para ello en la ventana *Oscilador* se elige el tipo *XT* (oscilador a cristal de cuarzo) para los montajes que se realizan en este libro (figura 3-13).
- 7º A continuación es necesario activar los **Bits de configuración**, que permiten seleccionar varias configuraciones del dispositivos (figura 3-13). En la pantalla del IC-Prog se muestran tres:

- **WDT (Watchdog Timer)**. Habilitación del *Watchdog*, cuyo significado se explicará en el capítulo 16. En aplicaciones sencillas se deshabilita.
- **PWRT (Power-up Timer)**. Temporizador al encendido. En aplicaciones sencillas se activa.
- **CP (Code Protect)**. Protección de código de programa. Cuando se programa la protección del código, no es posible leer el contenido de la memoria, de tal manera que el código del programa no se puede copiar, ni

n este caso el  
o > Microchip

Mas  
PIC 16C73A  
PIC 16C73B  
PIC 16C74A  
PIC 16C74B  
PIC 16C76  
PIC 16C77  
PIC 16F73  
PIC 16F74  
PIC 16F76  
PIC 16F77  
PIC 16C84  
PIC 16F83  
PIC 16F84  
PIC 16F84A  
PIC 16F84B

a de la barra de  
se puede elegir



ventana

a (LP, RC, XT,  
dor a cristal de

, que permiten  
En la pantalla

significado se  
habilita.

n aplicaciones

a. Cuando se  
contenido de la  
uede copiar, ni

alterar, aunque sí se puede volver a borrar completamente todo el microcontrolador. En aplicaciones sencillas se suele deshabilitar.



Figura 3-13 Selección del tipo de oscilador y de los bits de configuración

- 8º El IC-Prog ya está en condiciones de proceder a la grabación de datos en el dispositivo insertado en el programador. Para ello, en la pantalla de edición se escriben los datos del programa de control a grabar. Por ejemplo, un programa de control para el circuito de la figura 1-2 que lea la información proporcionada por los interruptores del puerto A y la visualice en los LEDs conectados al Puerto B tendría el formato: "1683 0186 30FF 0085 1283 0805 0086 2805", (el significado de estos números hexadecimales se discutirá en el capítulo 6). Una vez escritos estos códigos, la pantalla de edición tendría el aspecto de la figura 3-14. En el capítulo 7 se explicará como cargar estos datos más eficazmente a partir de un archivo creado anteriormente, sin necesidad de teclearlo.



Figura 3-14 Datos a grabar en el microcontrolador

- 9º Para proceder a la grabación del chip basta con activar el menú *Comando > Programa todo* (figura 3-15) o bien pulsar la tecla de función F5. También puede pulsar sobre el ícono correspondiente de la barra de herramientas (rayo sobre chip). El chip comenzará a ser programado con los datos cargados en el buffer activo.

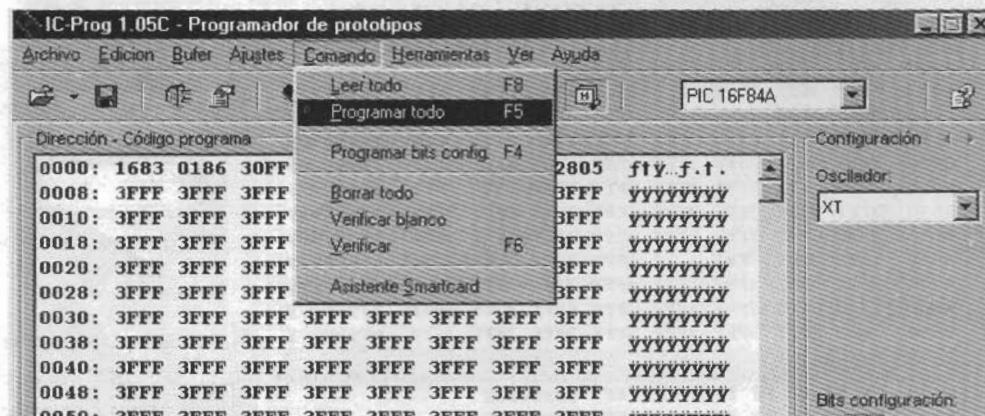


Figura 3-15 Comenzar a programar el PIC16F84A

- 10º El proceso de grabación se irá mostrando, tal como puede apreciarse en la figura 3-16. El tiempo empleado en la grabación del PIC16F84A dependerá de la rapidez del ordenador con el que se esté trabajando.
- 11º Una vez terminada la programación se procederá automáticamente a la verificación de los datos escritos en el chip, informando de este proceso con una pantalla como la que se muestra en la figura 3-16. De este modo, asegura que la programación del dispositivo ha sido efectuada correctamente.

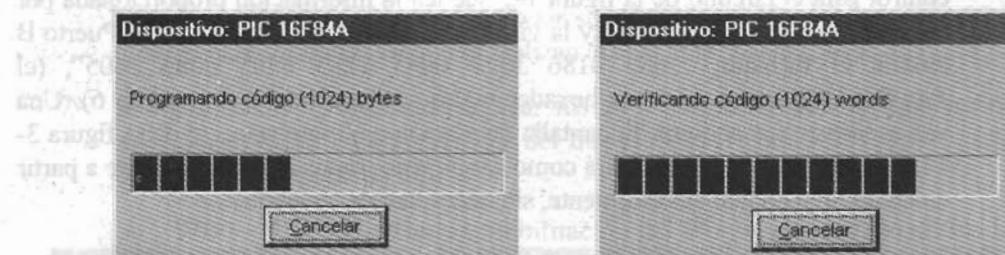


Figura 3-16 Pantallas que aparecen durante el proceso de programación y verificación

En el caso de que la verificación haya sido correcta, se informará de tal hecho mediante la ventana representada en la figura 3-17 y el proceso de grabación habrá finalizado.

### 3.6

puede  
operac  
cada u  
de la p

### 3.7

tener e

ú Comando >  
también puede  
as (rayo sobre  
os en el buffer

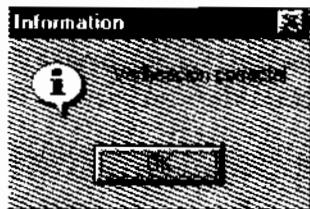


Figura 3-17 Grabación correcta

Si un microcontrolador está protegido contra la lectura de código, es decir tiene habilitada la opción **CP (Code Protect)** del área de configuración (figura 3-21), los datos grabados no pueden ser leídos en la fase de verificación y, por tanto, ésta no puede realizarse visualizando un error de verificación, sin embargo la grabación puede haber sido realizada correctamente. Más adelante se explica como evitar esta pantalla de error.

- 12º Una vez grabado el PIC16F84A se debe extraer del programador y comprobar su correcto funcionamiento dentro del circuito correspondiente. En este caso el programa lo que hace es sacar por el Puerto B el dato leído de las cinco líneas del Puerto A al que está conectado un array de interruptores. Esto se puede comprobar con el circuito de la figura 1-2.
- 13º Los datos grabados en el microcontrolador y la configuración se pueden salvar a un fichero utilizando el procedimiento usual de Windows mediante la selección del menú: *Archivo > Guardar como* y poniendo al archivo un nombre con extensión \*.bin, por ejemplo Entrenador\_01.bin.

### 3.6 BUFFER DE ALMACENAMIENTO DE PROGRAMAS

El IC-Prog dispone de 5 buffers para guardar datos en memoria, cada uno de ellos puede almacenar bien el contenido de un chip o de un archivo. Permite realizar diferentes operaciones con ellos, como programar un chip, comparar su contenido, etc. El acceso a cada uno de estos buffers se hace mediante las pestañas que aparecen en la parte inferior de la pantalla principal del IC-Prog (figura 3-18).

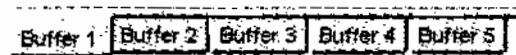


Figura 3-18 Acceso a los buffers de datos

### 3.7 IC-PROG TRABAJANDO BAJO WINDOWS 2000 O XP

En caso de trabajar con los sistemas operativos Windows 2000, XP o NT, se debe tener en la misma carpeta que el archivo ejecutable *icprog.exe*, el fichero *icprog.sys* para

Windows XP. Este fichero se puede descargar de la Web [www.ic-prog.com](http://www.ic-prog.com). Además hay que activar la opción de *Habilitar Driver NT/2000/XP*, localizada en el menú *Ajustes > Opciones > Miscelánea* (figura 3-19).

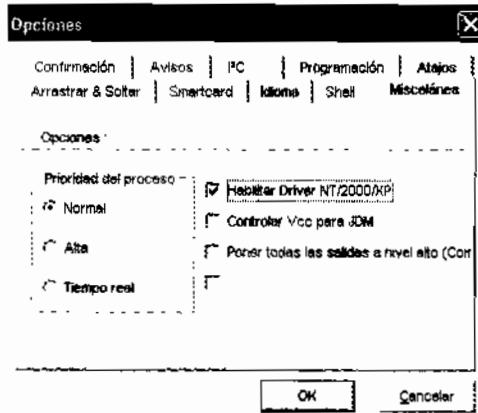


Figura 3-19 Para Windows NT, 2000 y XP hay que “Habilitar Driver NT/2000/XP”

### 3.8 ERRORES FRECUENTES EN LA PROGRAMACIÓN

Cuando por alguna causa la programación no se realiza correctamente IC-Prog informa de ello mediante el aviso oportuno. El error más frecuente en la programación de un PIC es el mostrado en la ventana de la figura 3-20, que informa de un error al verificar el contenido del chip en la dirección 0000h, que es la primera posición de memoria de programa del mismo.

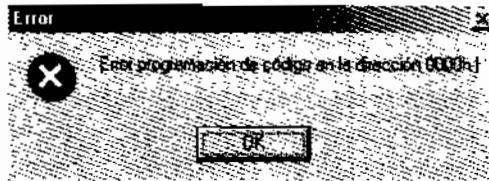


Figura 3-20 Pantalla típica de error en la programación

Este error suele estar motivado por algún fallo en la comunicación entre el ordenador y el programador, como puede ser:

- La mala conexión del cable serie.
- La colocación errónea del PIC en el programador.
- La configuración incorrecta del puerto serie COM 1 o COM 2.
- Un chip defectuoso.
- Una configuración de la protección de código.

og.com . Además  
en el menú *Ajustes*

Para solucionarlo se deben seguir los pasos enumerados a continuación:

- 1º Asegurarse que no está seleccionada la protección de código CP en los bits de configuración (figura 3-21). Este es un error muy típico y en este caso el programa leería los 8 primeros bytes de datos como ceros, avisando el error de verificación en la primera dirección.



Figura 3-21 La protección del código puede ocasionar una pantalla de aparente error

Si desea mantener la protección de código y que no aparezca este error en la fase de verificación, es necesario que la deshabilite. Para ello debe seleccionar el menú *Ajustes > Opciones > Programación* y deshabilitar las dos casillas de verificación tal como se muestra en la figura 3-22.

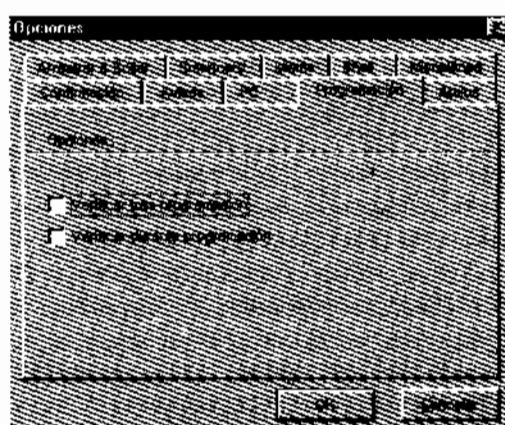


Figura 3-22 Si configura la protección del código, conviene deshabilitar la verificación

- 2º Una vez verificado que no ha elegido la opción de protección de código para programar el PIC puede continuar con las siguientes comprobaciones:
  - Revisar que el PIC se encuentra bien introducido en el zócalo del programador, con la dirección de la cápsula correcta y sin ninguna patilla dobrada o rota, lo que desgraciadamente es frecuente.
  - Verificar que el cable serie está bien conectado en sus dos extremos, tanto en el conector del programador como en el ordenador.

- Comprobar con un polímetro que no hay ninguna conexión del cable rota.
  - Debe asegurarse de que ha elegido en *Ajustes > Tipo > Hardware*, el tipo de programador correcto (*JDM*) y el puerto *COM* adecuado.
  - Comprobar que el dispositivo elegido es el que realmente está programando (*PIC16F84A*).
- 3º En caso de que ninguna de estas comprobaciones diera resultado se aconseja repetir la programación con otro chip que esté en buen estado para poder descartar que el fallo esté en el software o grabador y así asegurar que es el chip el que se encuentra en mal estado.

Cuando se trabaja en uno de los sistemas operativos Windows XP, Windows NT o Windows 2000, también se puede dar la pantalla de error mostrada en la figura 3-23.

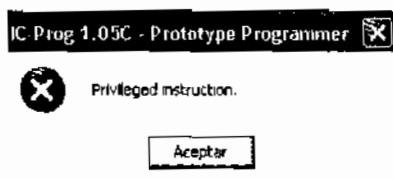


Figura 3-23 Error típico para sistemas operativos Windows 2000, XP y NT

Este error indica que el sistema no puede acceder a los puertos del ordenador y por tanto tampoco al grabador conectado a ellos, como se indicó anteriormente para solucionarlo se deberá instalar el fichero *icprog.sys* en el mismo directorio que el archivo ejecutable *icprog.exe* y, además, activar la opción de *Habilitar Driver NT/2000/XP*, que se encuentra en el menú *Ajustes > Opciones > Miscelánea* (figura 3-19). Una vez habilitado este driver, el programa pedirá volver a iniciarse y, a partir de ese momento, ya estará perfectamente preparado para realizar la programación de los dispositivos necesarios.

### 3.9 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en este tema, realícese la grabación en el microcontrolador de los programas que se indican y compruébese su correcto funcionamiento en el entrenador básico de la figura 1-2.

**Entrenador\_01.bin:** Los diodos LEDs conectados al nibble bajo del Puerto B se apagan y los del nibble alto se encienden. Los datos a grabar son (el significado de estos números hexadecimales se explicará en un próximo capítulo):

1683 0186 1283 30F0 0086 2804

Entrenador del Puerto A, el

- Si RA-MA
- Si RA-MA

Los datos

1683

0086

Entrenador cantidad indicado dato "\_\_\_0010 código "0001 grabar son:

1683

2805

343F



Figura 3-

Entrenador durante 0,4 segundos impares durante

1683

2805

0B8D

Entrenador encendido roza el final se apaga (observad que)

1683

2805

0B8D

xión del cable rota.

> Hardware, el tipo  
uado.

se realmente está

sultado se aconseja  
estado para poder  
gurar que es el chip

XP, Windows NT o  
la figura 3-23.

000, XP y NT

del ordenador y por  
anteriormente para  
torio que el archivo  
r NT/2000/XP, que  
ra 3-19). Una vez  
de ese momento, ya  
de los dispositivos

la grabación en el  
ébese su correcto

ajo del Puerto B se  
significado de estos

**Entrenador\_02.bin:** El Puerto B, que actúa como salida, es controlado por el bit 0 del Puerto A, que actúa como entrada. De manera tal, que:

- Si RA0 = 1, se encienden todos los LEDs de salida.
- Si RA1 = 0, se encienden sólo los LEDs del nibble alto.

Los datos a grabar son:

1683	0186	30FF	0085	1283	30FF	1C05	30F0
0086	2805						

**Entrenador\_03.bin:** Lee por el Puerto A una cantidad siempre menor de 8. Esta cantidad indica el número de LEDs que se ilumina a la salida. Así, por ejemplo, si lee el dato "000101" (cinco en decimal), en los LEDs conectados al Puerto B se iluminará el código "00011111", con los cinco diodos LEDs (D4, D3, D2, D1 y D0). Los datos a grabar son:

1683	0186	30FF	0085	1283	0805	2009	0086
2805	0782	3400	3401	3403	3407	340F	341F
343F	347F	34FF					

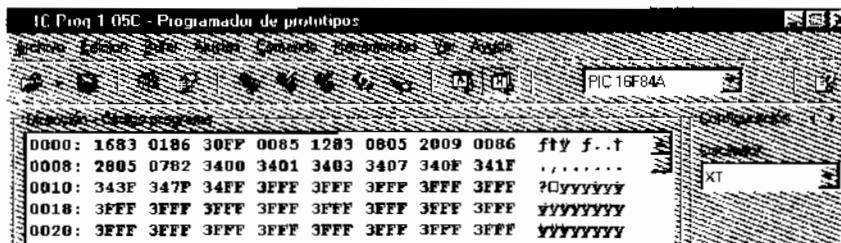


Figura 3-24 Datos a grabar con el Ic-Prog para el programa Entrenador\_03.bin

**Entrenador\_04.bin:** Los diodos pares conectados al Puerto B se encienden durante 0,4 segundos y los impares permanecen apagados. Después se encienden los impares durante el mismo tiempo y se apagan los pares. Los datos a grabar son:

1683	0186	1283	3055	0086	2009	2009	0986
2805	30C8	008D	30F9	008C	0000	0B8C	280D
0B8D	280B	0008					

**Entrenador\_05.bin:** Por la barra de LEDs conectada al Puerto B, un LED encendido rota a la izquierda durante 0,4 segundos en cada posición. Cuando llega al final se apagan todos los LEDs y de nuevo repite la operación. Los datos a grabar son (observad que son muy parecidos al anterior):

1683	0186	1283	1403	0186	2009	2009	0D86
2805	30C8	008D	30F9	008C	0000	0B8C	280D
0B8D	280B	0008					

**Entrenador\_06.bin:** En el display de 7 segmentos conectado al Puerto B se visualiza la cantidad leída por el puerto A. Así por ejemplo si por la entrada se lee "0101", en el display se visualiza "5". Los datos a grabar son:

1683	0186	30FF	0085	1283	0805	390F	200A
0086	2805	0782	343F	3406	345B	344F	3466
346D	347D	3407	347F	3467	3477	347C	3439
345E	3479	3471					

## 4.1 AP

La siguientes

- Me
- Me
- o
- o
- AL
- ope
- el p
- Do
- <R
- Co
- pah
- imp

## 4.2 O

Der

- M
- ins
- pro

erto B se  
se lee "---

## CAPÍTULO 4

# ORGANIZACIÓN DE LA MEMORIA

## 4.1 ARQUITECTURA INTERNA DEL PIC16F84

La figura 4-1 representa el diagrama de bloques del PIC16F84. Destacan los siguientes componentes que serán explicados más adelante:

- Memoria de programa tipo ROM Flash de 1 k x 14 bits.
- Memoria de datos dividida en 2 áreas:
  - Área RAM constituida por 22 registros de propósito específico (SFR) y 68 de propósito general.
  - Área EEPROM de datos formada por 64 registros de 8 bits.
- ALU de 8 bits y registro de trabajo W, del que normalmente recibe un operando que puede ser cualquier registro, memoria, puerto de entrada/salida o el propio código de instrucción.
- Dos puertos para la comunicación con el mundo exterior: PORTA de 5 bits <RA4:RA0> y PORTB de 8 bits <RB7:RB0>.
- Contador de programa de 13 bits, lo que en teoría permitiría direccionar 4 k de palabras de memoria, aunque el PIC16F84 sólo dispone de 1 k de memoria implementada.

## 4.2 ORGANIZACIÓN DE LA MEMORIA

Dentro del PIC16F84 se distinguen tres bloques de memoria:

- **Memoria de programa.** En sus 1024 posiciones contiene el programa con las instrucciones que gobiernan la aplicación. Es del tipo no volátil, es decir, el programa se mantiene aunque desaparezca la alimentación.

- **Memoria de datos RAM.** Se destina a guardar las variables y datos. Es volátil, es decir, los datos almacenados se borran cuando desaparece la alimentación.
- **Memoria EEPROM de datos.** Es una pequeña área de memoria de datos de lectura y escritura no volátil, gracias a la cual, un corte del suministro de la alimentación no ocasiona la pérdida de la información, que estará disponible al reinicializarse el programa. Se analizará con detenimiento en el capítulo 14.

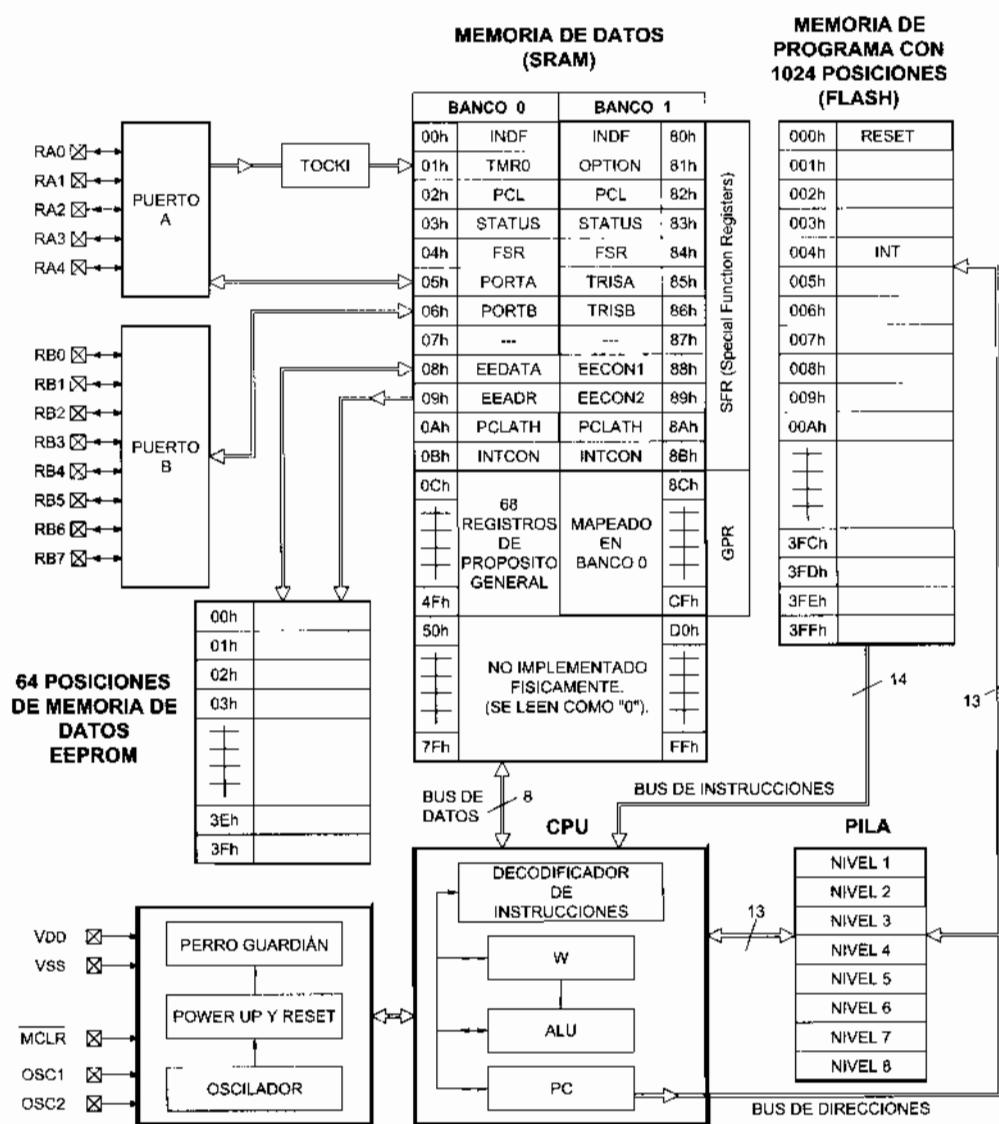


Figura 4-1 Arquitectura interna del PIC16F84

**4.3 ME**

El n  
almacen  
siempre e  
características  
alimentaci  
cada vez q

La  
mediante  
conectar a  
programa  
microcont

El P  
denominad  
que repres  
PIC16F84  
14 bits. As  
de reset) y  
el fabrican  
años.

**4.4 E**

Un  
forma sec  
memoria c

El c  
utiliza pa  
en la mem  
próxima  
secuencia

El i  
direccional  
000h hast

En  
se conecta  
cero forzad  
será la que

os. Es volátil, mentación.

a de datos de ministro de la disponible al capítulo 14.

IA DE  
A CON  
CIONES  
SH)

RESET

INT

14

13

A

NES

## 4.3 MEMORIA DE PROGRAMA

El microcontrolador está diseñado para que en su **memoria de programa** se almacenen todas las instrucciones del programa de control. El programa a ejecutar siempre es el mismo, por tanto, debe estar grabado de forma permanente. Esta característica de “no volatilidad” garantiza que la memoria mantenga su contenido aún sin alimentación, de forma que el programa no necesite volver a ser cargado en el sistema cada vez que se utilice.

La información contenida en estas memorias debe ser grabada previamente mediante un equipo físico denominado programador o grabador. Este equipo se debe conectar a un ordenador que mediante un software controla la grabación de la memoria de programa del microcontrolador. A este proceso se le llama programar o grabar el microcontrolador y se ha estudiado ampliamente en el capítulo anterior.

El PIC16F84 es un microcontrolador con un tipo memoria de programa no volátil denominada **ROM Flash**, que permite una grabación muy sencilla, rápida y cómoda, lo que representa gran facilidad en el desarrollo de diseños. La memoria del programa del PIC16F84, tiene una capacidad de 1 k (1024 posiciones) y está organizada en palabras de 14 bits. Así pues, la memoria de programa comienza en la posición 000h (posición inicial de reset) y llega hasta la 3FFh (figura 4-1). El PIC16F84 admite unas 1.000 grabaciones, y el fabricante garantiza que la información permanece inalterable durante varias decenas de años.

## 4.4 EL CONTADOR DE PROGRAMA (PC)

Un programa está compuesto por instrucciones que generalmente se ejecutan de forma secuencial. En el PIC16F84 cada una de esas instrucciones ocupa una posición de memoria de programa.

El **contador de programa** o PC (*Program Counter*) es un registro interno que se utiliza para direccionar las instrucciones del programa de control que están almacenadas en la memoria de programa (ver figura 4-1). Este registro contiene la dirección de la próxima instrucción a ejecutar y se incrementa automáticamente de manera que la secuencia natural de ejecución del programa es lineal, una instrucción después de otra.

El microcontrolador PIC16F84 dispone de un contador de programa que le permite direccionar los 1k x 14 bits de memoria de programa implementada, desde la posición 000h hasta la 3FFh.

En el esquema del entrenador básico de la figura 1-2, cuando el microcontrolador se conecta a la alimentación o cuando ocurre un reset, el contador de programa se pone a cero forzando así que la dirección de inicio sea la 000h. La primera instrucción ejecutada será la que esté grabada en esta posición.

4.5

Dirección	Nombre	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0					
<b>BANCO 0</b>														
00h	INDF	Registro utilizado en el direccionamiento indirecto (no es un registro físico)												
01h	TMR0	Timer / Contador de 8 bit												
02h	PCL	Registro con los 8 bits más bajos del Contador de Programa												
03h	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C					
04h	FSR	Registro utilizado como puntero en el direccionamiento indirecto												
05h	PORTA	--	--	--	RA4/T0CKI	RA3	RA2	RA1	RA0					
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT					
07h		Posición no implementada, se lee como 0												
08h	EEDATA	Registro de datos EEPROM												
09h	EEADR	Registro de direcciones EEPROM												
0Ah	PCLATH	--	--	--	Buffer escrito con los 5 bit más altos del PC									
0Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF					
<b>BANCO 1</b>														
80h	INDF	Registro utilizado en el direccionamiento indirecto (no es un registro físico)												
81h	OPTION	/RBPU	INTEDG	TOSC	TOSE	PSA	PS2	PS1	PS0					
82h	PCL	Registro con los 8 bit más bajos del Contador de Programa												
83h	STATUS	IRP	RP1	RP0	/TO	/PD	Z	DC	C					
84h	FSR	Registro utilizado como puntero en el direccionamiento indirecto												
85h	TRISA	--	--	--	Reg. de configuración de las líneas del Puerto A									
86h	TRISB	Registro de configuración de las líneas del Puerto B												
87h		Posición no implementada, se lee como 0												
88h	EECON1	--	--	--	EEIF	WRERR	WREN	WR	RD					
89h	EECON2	Reg. de control para grabar en la EEPROM de datos (no es un registro físico)												
8Ah	PCLATH	--	--	--	Buffer escrito con los 5 bit más altos del PC									
8Bh	INTCON	GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF					

Tabla 4-1 Registros del SFR (Special Function Registers)

4.6

bit 1	bit 0
<hr/>	
registro físico)	
<hr/>	
DC	C
ecto	
RA1	RA0
RB1	RB0/INT
<hr/>	
itos del PC	
INTF	RBIF
<hr/>	
registro físico)	
PS1	PS0
<hr/>	
DC	C
ecto	
cas del Puerto A	
<hr/>	
WR	RD
registro físico)	
<hr/>	
os del PC	
INTF	RBIF

## 4.5 MEMORIA DE DATOS

En esta memoria se almacenan los datos que se manejan en un programa. Estos datos varían continuamente, por lo que esta memoria debe ser de lectura y escritura. Se utiliza memoria denominada RAM que es de tipo volátil, con lo cual los datos se borran en caso de que desaparezca la alimentación.

La figura 4-1 muestra la estructura de la memoria de datos RAM del PIC16F84 donde se aprecia que está dividida en dos partes:

- **Registros de Funciones Especiales SFR** (*Special Function Registers*). Son los primeros registros, cada uno de ellos cumple un propósito especial en el control del microcontrolador.
- **Registros de Propósito General GPR** (*General Purpose Registers*). Son registros de uso general que se pueden usar para guardar los datos temporales del programa que se esté ejecutando. Tiene 68 posiciones para el PIC16F84.

La memoria de datos cuenta con dos **bancos** de memoria, Banco 0 y Banco 1:

- Los registros del SFR están agrupados entre las direcciones 00h a 0Bh para el Banco 0 y entre las direcciones 80h hasta 8Bh para el Banco 1. Algunos de los registros del SFR se encuentran duplicados en la misma dirección en los dos bancos, con el objeto de simplificar su acceso. Así por ejemplo, el registro STATUS se localiza en las direcciones 03h (Banco 0) y 83h (Banco 1).
- El banco de registros de propósito general está formado por 68 posiciones de memoria, ya que sólo son operativas las del Banco 0 (direcciones desde la 0Ch hasta la 4Fh), porque las del Banco 1 se mapean sobre el Banco 0. Es decir, cuando se apunta a un registro de propósito general del Banco 1 (direcciones de 8Ch hasta 0CFh), realmente se accede al mismo registro del Banco 0.

Para seleccionar el banco a acceder hay que configurar el bit 5 (RP0) del registro STATUS. Con RP0 = 0 se accede al Banco 0 y con RP0 = 1 se accede al Banco 1. El Banco 0 es seleccionado automáticamente después de un reset.

Las zonas de memoria 50h-7Fh y D0h-FFh no son empleadas y devuelven 0 en caso de lectura.

## 4.6 DIFERENCIAS ENTRE EL PIC16F84A Y EL PIC16C84

El microcontrolador PIC16C84 es un microcontrolador anterior al PIC16F84A y totalmente compatible con él, la diferencia principal es que su memoria de datos tiene menor tamaño. El PIC16C84 tiene 32 registros de propósito general (el mapa de memoria de datos llega hasta 2Fh) frente a los 68 registros disponibles en el PIC16F84. El

PIC16C84 fue reemplazado por el PIC16F84A de modo que los diseños que lo utilicen como elemento de control deben ser actualizados. Este proceso es transparente, se puede sustituir uno por otro sin realizar ningún tipo de modificación en la mayoría de las aplicaciones.

## 4.7 REGISTROS DEL SFR

La tabla 4-1 detalla los **registros de funciones especiales SFR** (*Special Function Registers*). Estos registros se describen en su totalidad en el apéndice E y se irán explicando a lo largo del libro. Por ahora se destacan los siguientes:

## 4.8 REGISTROS RELACIONADOS CON LOS PUERTOS

Los registros relacionados directamente con los puertos son:

- **PORTA**, en posición 05h del Banco 0. Puerto de entrada/salida de 5 bits (pines RA4:RA0). El Puerto A puede leerse o escribirse como si se tratara de un registro cualquiera. El registro que controla el sentido (entrada o salida) de sus pines se llama TRISA y está localizado en la dirección 85h del Banco 1.
- **PORTB**, en posición 06h del Banco 0. Puerto de entrada/salida de 8 bits (pines RB7:RB0). El Puerto B puede leerse o escribirse como si se tratara de un registro cualquiera. El registro que controla el sentido (entrada o salida) de sus pines se llama TRISB y está localizado en la dirección 86h del Banco 1.
- **TRISA**, posición 85h del Banco 1. Registro de configuración de las líneas del Puerto A. Es el registro de control para el Puerto A. Un “0” en el bit correspondiente al pin lo configura como salida, mientras que un “1” lo hace como entrada.
- **TRISB**, posición 86h del Banco 1. Registro de configuración de las líneas del Puerto B. Es el registro de control para el Puerto B. Un “0” en el bit correspondiente al pin lo configura como salida, mientras que un “1” lo hace como entrada.

## 4.9 REGISTRO PCL Y CONTADOR DE PROGRAMA

El PIC16F84 dispone de un contador de programa de 13 bits constituido por dos registros (figura 4-2):

- **PCL** (*Program Counter Low byte*), implementado en la posición de memoria RAM 02h (y duplicado en la posición 82h del Banco 1). Su contenido corresponde con los 8 bits más bajo del contador de programa. Este registro puede ser leído o escrito directamente.

iseños que lo utilicen transparente, se puede en la mayoría de las

TR (*Special Function* apéndice E y se irán

## S PUERTOS

alida de 5 bits (pines tratará de un registro alida) de sus pines se 1.

alida de 8 bits (pines tratará de un registro alida) de sus pines se 1.

ión de las líneas del Un "0" en el bit que un "1" lo hace

ión de las líneas del Un "0" en el bit que un "1" lo hace

## GRAMA

s constituido por dos

osición de memoria 1). Su contenido grama. Este registro

- **PCH** (*Program Counter High byte*). Los cinco bits de mayor peso del PC corresponden con este registro. No puede ser leído ni escrito directamente.

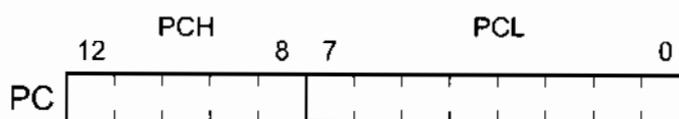


Figura 4-2 Composición del Contador de Programa (PC, Program Counter)

Durante la ejecución normal del programa el PCL se incrementa con cada instrucción, a menos que se trate de alguna instrucción de salto. Al conectar la alimentación se inicializa a  $(PCL) = b'00000000'$  y  $(PCH) = b'00000'$ .

Los 13 bits del contador de programa le permiten direccionar hasta  $8 \text{ k} \times 14$  bits. Sin embargo, el PIC16F84 dispone tan solo de  $1\text{k} \times 14$  bits de memoria implementada, desde la posición 000h hasta la 3FFh. Los 3 bits de mayor peso del PC no los tiene en cuenta, así pues, las direcciones 30h, 430h, 830h, C30h, 1430h, 1830h y 1C30h se consideran la misma.

## 4.10 REGISTRO DE TRABAJO W

El **registro de trabajo W** (*Work*) es el registro principal y participa en la mayoría de las instrucciones. Se localiza dentro de la CPU del PIC16F84 como se aprecia en la figura 4-1.

La misma figura muestra como el microcontrolador posee una ALU (*Arithmetic Logic Unit*) de 8 bits. Ésta se encarga de realizar las operaciones lógicas o aritméticas que requiere la ejecución del programa con dos operandos, uno que proviene del registro W y el otro que se encuentra en cualquier otro registro o en el propio código de instrucción.

## 4.11 REGISTRO DE ESTADO O STATUS

El **registro de estado o STATUS** ocupa la posición 03h del Banco 0 ó la 83h del Banco 1 y es uno de los registros más importantes y utilizados. Los bits de este registro indican el estado de la última operación aritmética o lógica realizada, la causa de reset y los bits de selección de banco para la memoria de datos. A los bits del registro de estado se les suele denominar *flags* o banderas.

Su constitución interna se muestra en la tabla 4-2. A continuación se explican sus bits principales, en capítulos posteriores se explicarán el resto y en el apéndice D se realiza una descripción detallada de todo el conjunto.

IRP	RP1	RP0	/TO	/PD	Z	DC	C
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 4-2 Registro de estado o STATUS

- **C (Carry bit).** Flag de acarreo en el octavo bit. En instrucciones de suma aritméticas se activa cuando se presenta acarreo desde el bit más significativo del resultado, lo que indica que el resultado ha desbordado la capacidad del registro sobre el que trabaja, es decir, el resultado de la operación ha superado el valor  $1111111_2$ , ( $255_{10}$ ), que es el máximo valor que se puede representar con 8 bits. En el capítulo 8 se explica en detalle su funcionamiento. El significado de los dos valores posibles es:
  - C = 0. En la suma significa que no ha habido acarreo y en la resta que el resultado ha sido negativo.
  - C = 1. En la suma significa que ha habido acarreo y en la resta que el resultado ha sido positivo.
- **DC (Digit Carry).** Flag de acarreo en el 4º bit de menos peso. En operaciones aritméticas se activa cuando hay un acarreo entre el bit 3 y 4, es decir, cuando hay acarreo entre los nibbles de menor y de mayor peso. Tiene un comportamiento análogo al del bit C.
- **Z (Zero).** Flag de cero. Se activa a "1" cuando el resultado de una operación aritmética o lógica es cero.
  - Z = 0. El resultado de la última operación ha sido distinto de cero.
  - Z = 1. El resultado de la última operación ha sido cero.
- **RP0 (Register Bank Select bit).** Selección del banco para el direccionamiento directo. Señala el banco de memoria de datos seleccionado.
  - RP0 = 0. Selecciona el Banco 0.
  - RP1 = 1. Selecciona el Banco 1.

## 4.12 ESTADO DE LOS REGISTROS TRAS UN RESET

Después de un reset, los registros se encontrarán en el estado que se indica en la tabla 4-3 en la que las dos primeras columnas muestran los datos más importantes, que son:

- Conexión a la alimentación. Estado de los registros inmediatamente después de conectar la alimentación.
- MCLR modo normal. Estado de los registros después de llevar el pin MCLR a masa en funcionamiento normal.

4.13

14 bits  
puede  
reserva  
de con

Bi

DC	C
Bit 1	Bit 0

strucciones de suma más significativo del apacido del registro ha superado el valor epresentar con 8 bits. significado de los dos

y en la resta que el y en la resta que el

peso. En operaciones , es decir, cuando hay c un comportamiento do de una operación nunto de cero. el direccionamiento

## N RESET

o que se indica en la más importantes, que

atamente después de evar el pin MCLR a

REGISTRO	CONEXIÓN A LA ALIMENTACIÓN	/MCLR MODO NORMAL	/MCLR EN "SLEEP"	INTERRUPCIÓN EN "SLEEP"
W	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	(Incrementa)
STATUS	0001 1xxx	000u uuuu	0001 0uuu	uuu1 0uuu
FSR	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu quuu
PORTA	--x xxxx	--u uuuu	--u uuuu	--u uuuu
PORTB	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TRISA	--1 1111	--1 1111	--1 1111	--u uuuu
TRISB	1111 1111	1111 1111	1111 1111	uuuu uuuu
EEDATA	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
EEADR	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCLATH	--0 0000	--0 0000	--0 0000	--u uuuu
JNTCON	0000 000x	0000 000u	0000 000u	uuuu uuuu
OPTION	1111 1111	1111 1111	1111 1111	uuuu uuuu
EECON1	--0 x000	--0 x000	--0 x000	--0 uuuu

u = No cambia, (*unchanged*). x = Desconocido. - = No implementado.

Tabla 4-3 Estado de algunos registros después de los distintos tipos de reset

## 4.13 REGISTRO DE CONFIGURACIÓN

El PIC16F84A dispone de una palabra de configuración (*Configuration Word*) de 14 bits que se escribe durante el proceso de grabación del microcontrolador y que no se puede modificar durante la ejecución de un programa. Dichos bits ocupan la posición reservada de memoria de programa 2007h. La tabla 4-4 muestra la estructura de la palabra de configuración.

			CP	PWRTE	WDTE	FOSC1	FOSC0
Bit 13	...	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 4-4 Registro de configuración

- **FOSC<1:0>** (*Flag Oscillator Selection*). Selección del tipo de oscilador:
  - FOSC = 00. Oscilador de bajo consumo LP (32 kHz - 200 kHz)
  - FOSC = 01. Oscilador estándar XT ( 100 kHz- 4MHz)
  - FOSC = 10. Oscilador de alta velocidad HS ( 4 MHz - 20 MHz)
  - FOSC = 11. Oscilador de bajo coste RC.
- **WDTE** (*Watchdog Enable*). Bit de habilitación del *Watchdog*.
  - WDTE = 0. Watchdog deshabilitado.
  - WDTE = 1. Watchdog habilitado.
- **PWRTE** (*Power-up Timer Enable*). Activación del temporizador *Power-Up*.
  - PWRTE = 0. Temporizador *Power-Up* deshabilitado.
  - PWRTE = 1. Temporizador *Power-Up* habilitado.

- **CP** (*Code Protection bit*) Bit de protección de código.
  - CP = 0. Toda la memoria de programa está protegida contra lecturas indeseables.
  - CP = 1. La memoria de programa se puede leer. No está protegida.

Durante la grabación de la palabra de configuración también se graban los identificadores ID que permiten conocer el programa, versión o cliente, correspondiente a esa grabación. Hay cuatro registros identificadores de 14 bits cada uno, pero sólo se utilizan los cuatro bits de menor peso de cada uno de ellos. Su distribución es “11 1111 1000 xxxx”, siendo “x” los bits en los cuales se asigna la identificación particular.

Tanto el registro *Configuration Word* como los identificadores se pueden programar con el IC-Prog dentro del área de configuración, en la zona derecha de la figura 4-3.

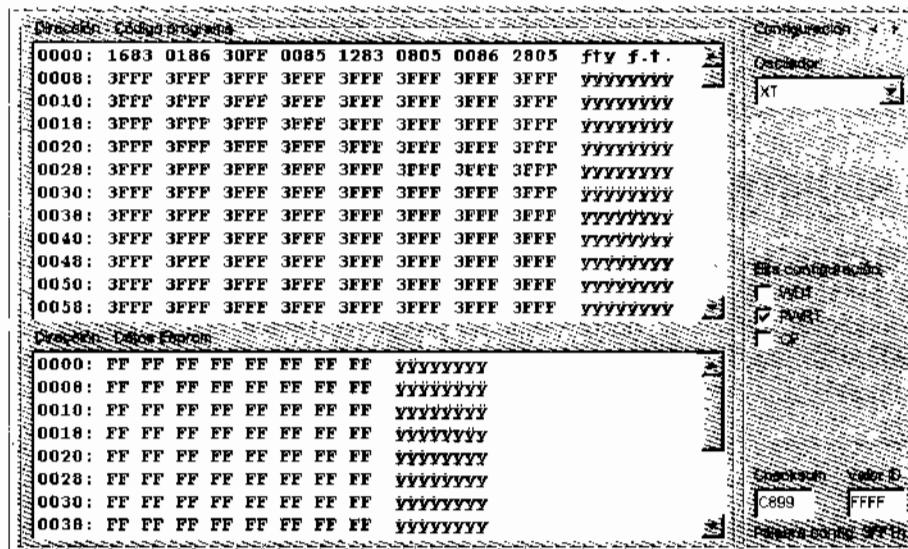


Figura 4-3 Opciones de configuración en el IC-Prog

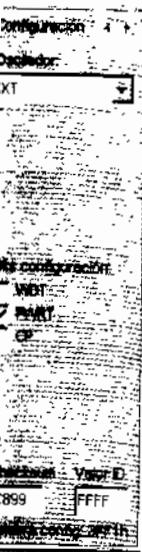
da contra lecturas  
rotegida.

ién se graban los  
; correspondiente a  
uno, pero sólo se  
ucción es "11 1111  
particular.

adores se pueden  
zona derecha de la

## CAPÍTULO 5

# ARQUITECTURA INTERNA



Al igual que los demás miembros de su familia, el PIC16F84 se caracteriza por:

- Tener una arquitectura *Harvard*.
- Su procesador es segmentado ó *Pipeline*.
- Su procesador es tipo *RISC*.
- El formato de las instrucciones es *ortogonal*.
- La arquitectura está basada en banco de registros.

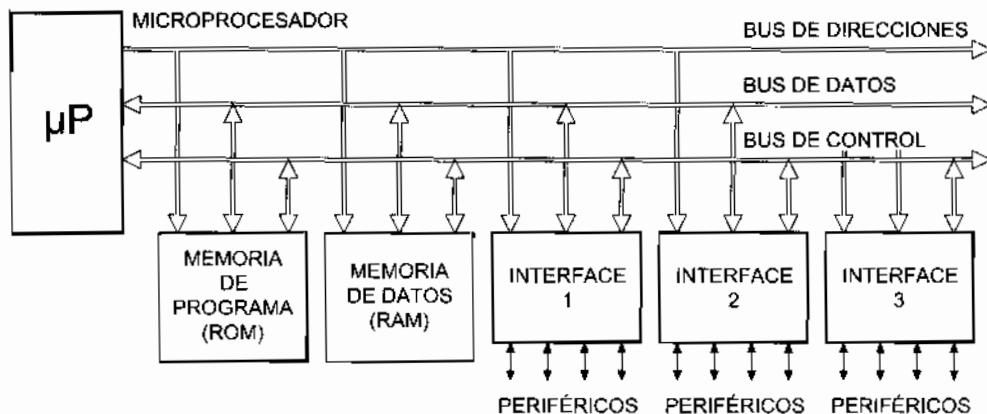
A continuación se explican algunos de estos conceptos, si bien es cierto que su conocimiento no es imprescindible para la comprensión del resto del libro es recomendable tener unas ciertas nociones sobre los mismos. Algunos de los conceptos mencionados (como por ejemplo las interrupciones) no serán analizados hasta capítulos posteriores. Si así lo desea, el lector puede saltarse este capítulo sin que sufra la continuidad del libro y volver a él conforme lo vaya necesitando.

## 5.1 MICROPROCESADOR Y MICROCONTROLADOR

Las figuras 5-1 y 5-2 demuestran las diferencias entre los sistemas digitales basados en microprocesador respecto de los basados en microcontrolador.

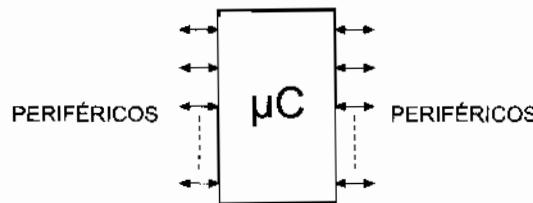
Un **microprocesador** es básicamente un chip que contiene la CPU (*Central Processing Unit*) que se encarga de controlar todo el sistema. Un sistema digital basado en un microprocesador es un sistema abierto ya que su configuración difiere según la aplicación a la que se destine. Se pueden acoplar los módulos necesarios para configurarlo con las características que se desee (figura 5-1). Para ello saca al exterior las líneas de sus buses de datos, direcciones y control de modo que permita su conexión con la memoria y

los módulos de entrada/salida. Finalmente resulta un sistema implementado por varios circuitos integrados dentro de una misma placa de circuito impreso.



*Figura 5-1 Estructura de un sistema digital basado en microprocesador*

Un **microcontrolador** es un sistema cerrado, lo que quiere decir que en un solo circuito integrado se encierra un sistema digital programable completo (figura 5-2). Este dispositivo se destina a gobernar una sola tarea que no se puede modificar. Los microcontroladores disponen de los bloques esenciales: CPU, memorias de datos y de programa, reloj, periféricos de entradas/salidas, etc.



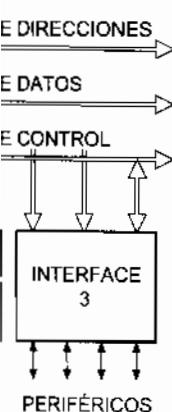
*Figura 5-2 Estructura de un sistema digital basado en microcontrolador*

La diferencia fundamental entre ambos es que un sistema digital basado en un microcontrolador está formado por un solo circuito integrado lo que reduce notablemente el tamaño y el coste, mientras que un sistema basado en un microprocesador, al estar compuesto por varios circuitos integrados para soportar las memorias y los módulos de entrada/salida, tiene mayor tamaño, más coste y menor fiabilidad.

## 5.2 ARQUITECTURA DE VON NEUMANN

La arquitectura tradicional de sistemas digitales programables se basa en el esquema propuesto por John **Von Neumann**. En este modelo la unidad central de proceso

entado por varios



rocesador

cir que en un solo  
(figura 5-2). Este  
de modificar. Los  
ias de datos y de

o CPU está conectada a una memoria única que contiene las instrucciones del programa y los datos, tal como describe la figura 5-3.

El tamaño de la unidad de datos o instrucciones está fijado por el ancho del bus de datos de la memoria exterior utilizada, que es de 8 bits. Un microprocesador con un bus de 8 bits que lo conecta con la memoria deberá manejar datos e instrucciones de una o más unidades de 8 bits de longitud. Cuando deba acceder a una instrucción o dato de más de un byte de longitud, deberá realizar más de un acceso a la memoria. Por otro lado este bus único limita la velocidad de operación del microprocesador, ya que no se puede buscar en la memoria una nueva instrucción antes de que finalicen las transferencias de datos que pudieran resultar de la instrucción anterior.

Resumiendo, las dos principales limitaciones de la arquitectura tradicional o de Von Neumann son:

- La longitud de las instrucciones está limitada por la unidad de longitud de los datos, por lo tanto el microprocesador debe hacer varios accesos a memoria para buscar instrucciones complejas.
- La velocidad de operación está limitada por el efecto de cuello de botella que significa un único bus para datos e instrucciones, que impide superponer ambos tiempos de acceso.

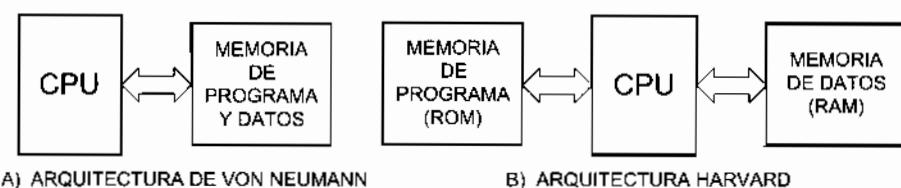


Figura 5-3 Diferencia entre la arquitectura Von Neumann y arquitectura Harvard

### 5.3 ARQUITECTURA HARVARD

Tradicionalmente los sistemas digitales programables se basaban en la arquitectura de Von Neumann, caracterizada por disponer de una única memoria en la que se almacenan tanto los datos como las instrucciones. A esta memoria se accede a través de un sistema de buses único. La única ventaja que posee es que simplifica la lógica del microcontrolador.

Los microcontroladores PIC utilizan una arquitectura **Harvard** que dispone de dos memorias independientes a las que se conecta mediante dos grupos de buses separados (figura 5-3):

- Memoria de datos.
- Memoria de programa.

Ambos buses son totalmente independientes y pueden ser de distintos anchos, ésto permite que la CPU pueda acceder de forma independiente y simultánea a la memoria de datos y a la de instrucciones, consiguiendo que las instrucciones se ejecuten en menos ciclos de reloj.

Esta dualidad de la memoria de datos por un lado y por otro la memoria de programa, permite la adecuación del tamaño de las palabras y los buses a los requerimientos específicos de las instrucciones y los datos.

Se puede concluir que las principales ventajas de la arquitectura Harvard son:

- El tamaño de las instrucciones no está relacionado con el de los datos y, por lo tanto, puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa. Así se logra una mayor velocidad y una menor longitud de programa.
- El tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

5.5

fundam

## 5.4 PROCESADOR SEGMENTADO

Un **procesador segmentado** o *Pipeline* realiza simultáneamente la ejecución de una instrucción y la búsqueda de código de la siguiente, de esta manera, se puede ejecutar una instrucción en un ciclo. La tabla 5-1 describe el funcionamiento para un ejemplo concreto, donde cada instrucción se ejecuta en el denominado **ciclo máquina** que está constituido por cuatro ciclos de reloj.

PROGRAMA:	1º Ciclo	2º Ciclo	3º Ciclo	4º Ciclo	5º Ciclo
	Búsqueda 1	Ejecuta 1			
1. bsf STATUS,RP0		Búsqueda 2	Ejecuta 2		
2. clrf TRISB					
3. movlw 0xFF			Búsqueda 3	Ejecuta 3	
4. movwf TRISA				Búsqueda 4	Ejecuta 4

Tabla 5-1 Ejemplo de funcionamiento de un procesador segmentado o Pipeline

Este sistema acompañado de una estructura Harvard, permite que las instrucciones se ejecuten en un sólo ciclo máquina (4 ciclos de reloj), salvo en el caso de saltos de programa. Lo que se hace internamente es que mientras se ejecuta la instrucción actual se carga la siguiente instrucción, alcanzando una alta velocidad de ejecución, (figura 5-4).

arquitecto  
de los  
esporádico  
con progra  
de instrucejecuta  
como e

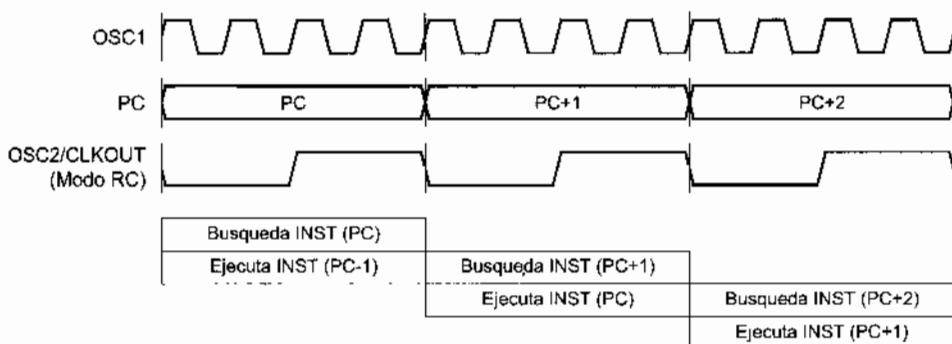


Figura 5-4 Cronograma de tiempo de un procesador segmentado

## 5.5 PROCESADOR RISC

Las CPUs atendiendo al tipo de instrucciones que utilizan pueden clasificarse fundamentalmente en:

- **CISC (Complex Instruction Set Computer).** Son procesadores con un juego de instrucciones complejo. Su repertorio de instrucciones es elevado y algunas de ellas son muy sofisticadas y potentes. Su problema es que requieren de muchos ciclos de reloj para ejecutar las instrucciones complejas.
- **RISC (Reduced Instruction Set Computer).** Son microprocesadores con un repertorio de instrucciones reducido. Las instrucciones son muy simples y suelen ejecutarse en un ciclo máquina. Los procesadores RISC suelen tener una estructura *Pipeline* y ejecutar casi todas las instrucciones en el mismo tiempo. El PIC16F84 es un microcontrolador RISC con sólo 35 instrucciones.
- **SISC (Specific Instruction Set Computer).** Estos procesadores poseen un juego de instrucciones específico para cada aplicación. Están destinados a aplicaciones muy concretas.

Los procesadores RISC representan un importante avance en el desarrollo de la arquitectura de los microcontroladores. Está demostrado que las instrucciones complejas de los microcontroladores CISC son empleadas sólo por algunos usuarios y muy esporádicamente. Por tal motivo *Microchip* decidió diseñar sus microcontroladores PIC con procesador RISC optimizado para ejecutar a muy alta velocidad un reducido número de instrucciones, las más frecuentemente utilizadas.

En los microcontroladores RISC las instrucciones complejas se obtienen ejecutando un conjunto de instrucciones disponibles, en lugar de ser una única instrucción como en los CISC.

## 5.6 ARQUITECTURA ORTOGONAL

En un microprocesador con **arquitectura ortogonal** una instrucción puede utilizar cualquier elemento de la arquitectura como fuente o destino. Ésta es una diferencia muy significativa respecto de otros microcontroladores.

La figura 5-5 representa un diagrama simplificado de la arquitectura interna del camino de los datos en la CPU de los microcontroladores PIC frente a los tradicionales. Este diagrama no representa con exactitud el circuito interno de estos microcontroladores, pero es exacto y claro desde la óptica del diseñador de sistemas digitales. La principal diferencia entre ambos radica en la ubicación del registro de trabajo, que para los PIC se denomina W (*Work Register*) y para los tradicionales es el "Acumulador".

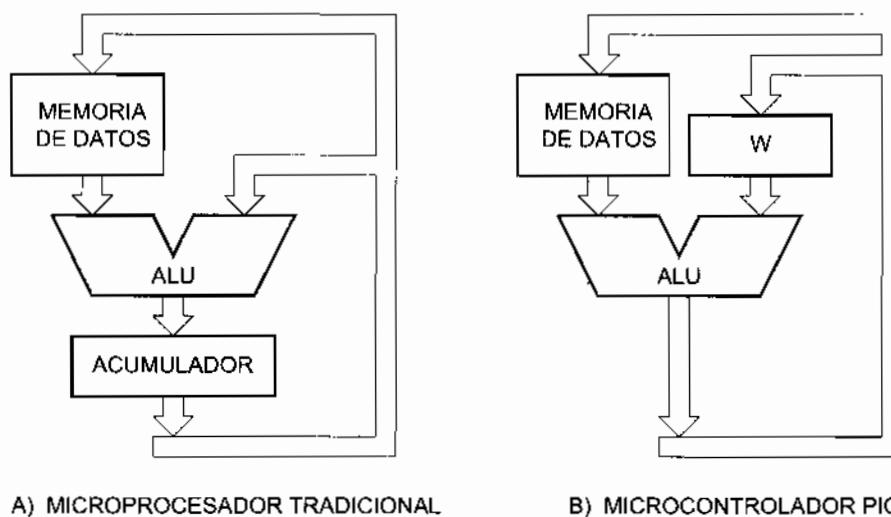


Figura 5-5 Diferencia de los microcontroladores PIC respecto de los tradicionales

En los microcontroladores tradicionales todas las operaciones se realizan sobre el acumulador. La salida del mismo está conectada a una de las entradas de la Unidad Aritmética y Lógica (ALU) y, por lo tanto, éste es siempre uno de los dos operandos de cualquier instrucción. La salida de la ALU va solamente a la entrada del acumulador, el resultado de cualquier operación siempre quedará en este registro.

En los microcontroladores PIC la salida de la ALU va al registro W y también a la memoria de datos, así el resultado puede guardarse en cualquiera de los dos destinos. En las instrucciones de doble operando, uno de los dos datos siempre debe estar en el registro W. La gran ventaja de esta arquitectura es que permite un gran ahorro de instrucciones ya que el resultado de cualquier instrucción que opere con la memoria, puede dejarse en la misma posición de memoria o en el registro W.

encu  
perif  
(figu  
regis  
regis

5.7

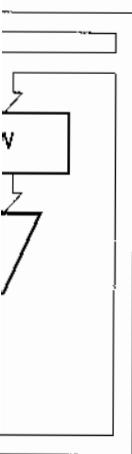
pued  
misn  
estos  
detal

5.8

cont  
corre  
bits  
confi  
línea

ción puede utilizar  
na diferencia muy

tección interna del  
a los tradicionales.  
microcontroladores,  
tales. La principal  
ue para los PIC se  
r".



OLADOR PIC

os tradicionales

e realizan sobre el  
das de la Unidad  
dos operandos de  
el acumulador, el

W y también a la  
s dos destinos. En  
estar en el registro  
de instrucciones ya  
uede dejarse en la

Como se estudió en el capítulo anterior, en la memoria de datos de los PICs se encuentran ubicados casi todos los registros de control del microcontrolador y sus periféricos de entrada/salida, así como las posiciones de memoria de usos generales (figura 4-1). El PIC16F84 es un microcontrolador con arquitectura basada en bancos de registros ya que todos los elementos del sistema están implementados físicamente como registros.

## 5.7 PUERTOS

El PIC16F84 dispone de dos puertos paralelos A y B. Las líneas de estos puertos se pueden programar individualmente como entradas o como salidas y se utilizan casi de la misma forma. Debido al escaso encapsulado, con sólo 18 pines, determinadas líneas de estos puertos se comparten con otros recursos internos. A continuación se analiza en detalle la constitución interna de éstos.

## 5.8 PUERTO A

El Puerto A está constituido por 5 líneas RA4:RA0 cuyo sentido de trabajo es controlado mediante el registro TRISA, en el que un bit a "0" configura la línea correspondiente como salida, y un bit a "1" como entrada. Después de un reset, todos los bits del registro TRISA quedan a uno, por lo que todas las líneas del Puerto A quedan configuradas como entradas. La figura 5-6 representa el diagrama interno de una de las líneas RA0 a RA3 del Puerto A.

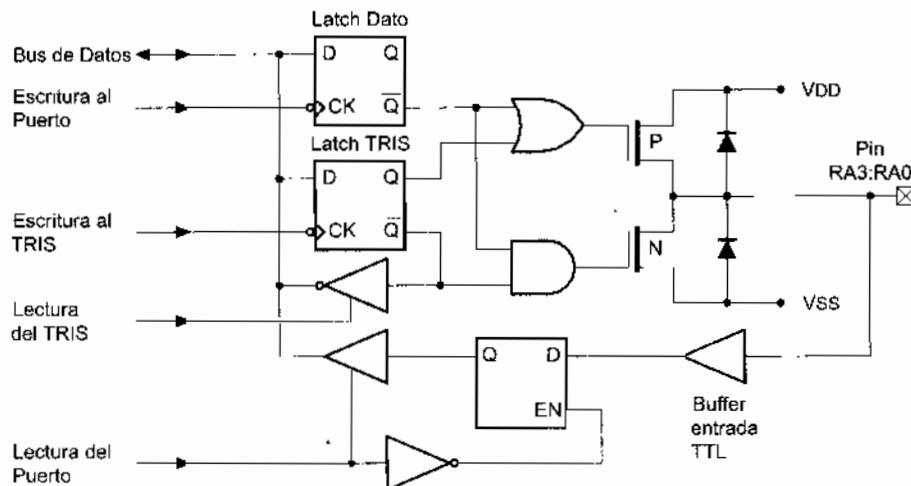


Figura 5-6 Constitución interna de los pines RA3:RA0

A continuación se analiza el funcionamiento de una línea como salida suponiendo que saca un dato a nivel lógico “1”:

- 1º Previamente el *Latch TRIS* de configuración debe contener un “0” para que actúe como salida.
- 2º El dato aparece en la línea correspondiente del bus de datos interno.
- 3º Se activa la señal de escritura al puerto, produciendo el almacenamiento de dicho nivel en el *Latch Data*. Si el bus de datos tenía un “1”, la salida /Q pasaría a tener un “0”.
- 4º Con estos valores la puerta OR tendrá un “0” en su salida al igual que la AND y producen la conducción del transistor PMOS superior y el bloqueo del NMOS inferior.
- 5º La línea exterior queda conectada a la alimentación  $V_{DD}$  obteniendo un nivel alto.
- 6º El *Latch Data* provoca que la línea de salida esté “latcheada” conservando su valor hasta que no se reescriba el mismo.
- 7º Al estar “latcheada”, cuando se configura como salida saca al exterior el nivel lógico que se haya cargado por última vez en el registro de salida.

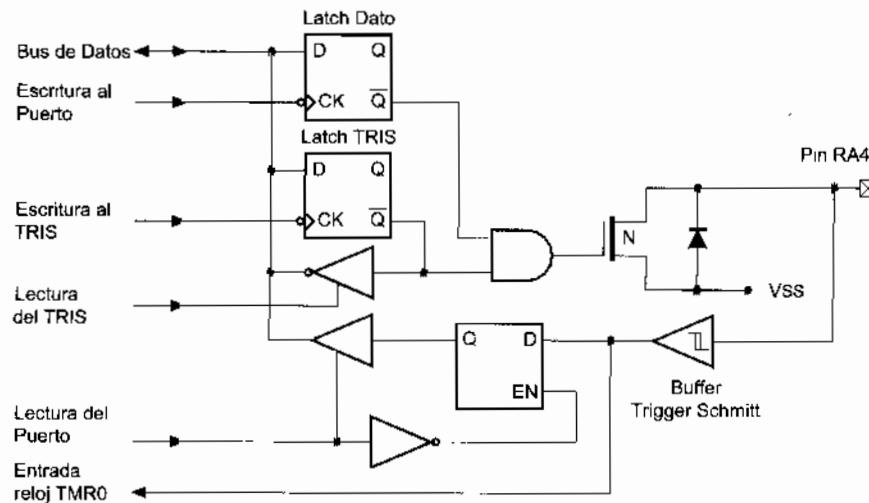


Figura 5-7 Constitución interna del pin RA4

Si una de estas líneas actúa como entrada hay que tener presente que:

- 1º Al programarse como entrada, la salida del *Latch TRIS* provocará un nivel alto a la salida de la puerta OR y un nivel bajo a la salida de la puerta AND, por lo que los dos transistores MOS de salida quedan bloquedados y en alta impedancia.
- 2º El nivel de tensión aplicado a ella desde el exterior pasa al bus interno a través del buffer triestado cuando se activa la señal de lectura del puerto.
- 3º Al leer una línea de entrada se obtiene el estado actual del pin correspondiente, no el dato almacenado en el latch.

- 4º La información presente en una línea de entrada debe mantenerse estable durante todo el ciclo de instrucción.

La línea RA4 adopta una estructura diferente (figura 5-7):

- Cuando se configura como salida tiene una configuración de tipo drenador abierto, por tanto, necesita una resistencia de Pull-Up externa. Este es un error muy común en los diseñadores novedosos cuando utilizan el pin RA4 como salida por primera vez.
- Cuando se configura como entrada está provista de un Trigger Schmitt que proporciona una buena inmunidad al ruido, por ello esta línea se debe utilizar preferentemente como entrada frente a cualquier otra. Es común con la entrada externa del Timer 0.

Estas líneas son capaces de entregar niveles TTL cuando la tensión de alimentación aplicada en  $V_{DD}$  es de 5V. Cada línea de salida puede suministrar una corriente máxima de 20 mA cuando está a nivel alto o absorber una corriente máxima de 25 mA cuando está a nivel bajo. Como hay una limitación de dissipación máxima de potencia del chip, está limitada la suma de corriente por las cinco líneas del Puerto A, que no puede exceder de 50 mA cuando están a nivel alto y 80 mA cuando están a nivel bajo.

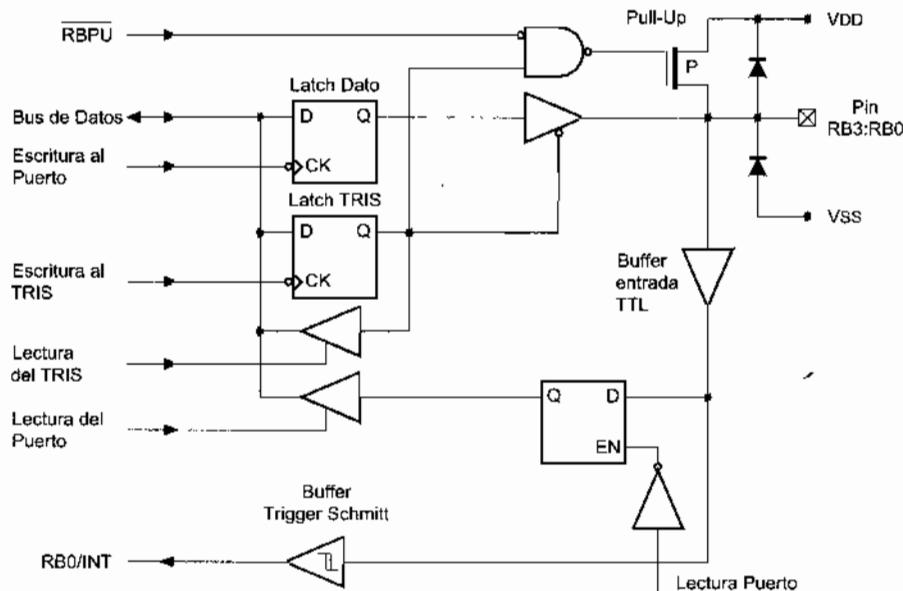


Figura 5-8 Constitución interna de los pines RB3:RB0

## 5.9 PUERTO B

El Puerto B es un puerto bidireccional de 8 bits completo, en el que sólo la línea RB0/INT tiene dos funciones multiplexadas, la propia de entrada/salida del puerto y la petición de interrupción externa. Las líneas RB0 a RB3 adoptan una estructura distinta a las de las líneas RB4 a RB7 según se aprecia en la figura 5-8. La razón de ser de esta diferencia radica en el hecho de que es posible programar la generación de una interrupción durante un cambio de estado de una cualquiera de las líneas RB7:RB4. Todas las líneas del Puerto B disponen de una resistencia de Pull-Up de alto valor, conectada a la alimentación.

Las líneas RB7:RB4, cuando actúan como entradas, pueden ser programadas para generar una interrupción si alguna de ellas cambia su estado lógico. La figura 5-9 muestra el diagrama interno de una de estas líneas. El circuito permite detectar la variación de una de estas señales cuando está en modo entrada, para ello, compara la última señal memorizada durante la última lectura del Puerto B. El cambio de una de las señales de entrada produce una interrupción que se refleja en el flag RBIF del registro INTCON.

Estas líneas son capaces de entregar niveles TTL cuando la tensión de alimentación aplicada en  $V_{DD}$  es de 5V. Cada línea de salida puede suministrar una corriente máxima de 20 mA cuando está a nivel alto o absorber una corriente máxima de 25 mA cuando está a nivel bajo. Como hay una limitación de disipación máxima de potencia del chip, está limitada la suma de corriente por las ocho líneas del Puerto B, que no puede exceder de 100 mA cuando están a nivel alto y 150 mA cuando están a nivel bajo.

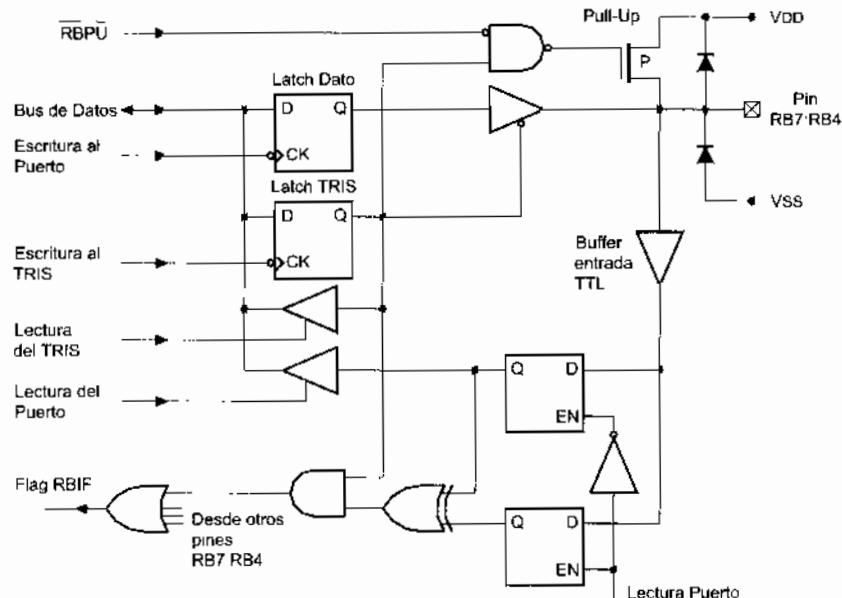


Figura 5-9 Constitución interna de los pines RB7:RB4

que sólo la línea  
a del puerto y la  
ructura distinta a  
n de ser de esta  
eración de una  
RB7:RB4. Todas  
ir, conectada a la

rogramadas para gura 5-9 muestra variación dc una la ultima señal le las señales de o INTCON.

de alimentación  
corriente máxima  
mA cuando está  
ciudad del chip, está  
puede exceder de

• VDD

vss

## CAPÍTULO 6

## **ENSAMBLADOR**

## 6.1 LENGUAJE MÁQUINA

El Único lenguaje que entienden los microcontroladores es el formado por los ceros y unos del sistema binario. Cualquier instrucción que deba ser ejecutada por el microcontrolador debe estar expresada en binario. A este lenguaje se le denomina lenguaje máquina, por ser el que comprende el microcontrolador. Los códigos de este lenguaje que forman las instrucciones se llaman códigos máquina. Así por ejemplo, cuando el microcontrolador PIC16F84 lee el código máquina "11111000111010", está recibiendo la instrucción: "suma 58 al registro de trabajo W y guarda el resultado en este mismo registro W".

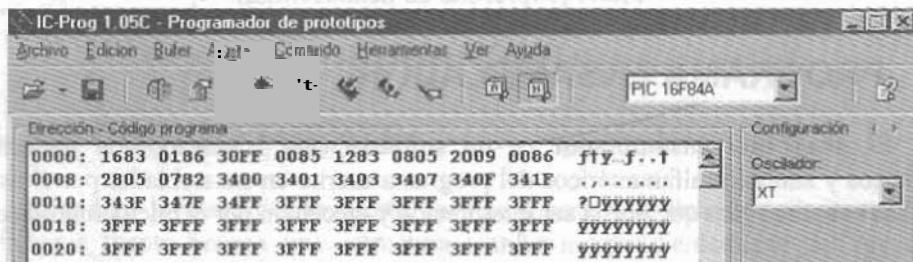


Figura 6-1 El IC-Prog utiliza el lenguaje máquina codificado en hexadecimal

Cualquier otro lenguaje que se utilice debe ser traducido a unos y ceros para que el microcontrolador pueda procesarlo. Dicha codificación binaria resulta incómoda para trabajar, por lo que muchas veces se utiliza la codificación hexadecimal para facilitar la interpretación de los códigos máquina y no saturar las pantallas (ni los cerebros) de unos y ceros. Así por ejemplo, en el capítulo 3 se procedió a grabar los microcontroladores

utilizando el programa IC-Prog, que trabaja en lenguaje **máquina**, pero utilizando la codificación hexadecimal, tal como se aprecia en el ejemplo de la figura 6-1.

## 6.2 LENGUAJE ENSAMBLADOR

El lenguaje máquina es difícil de utilizar por el hombre ya que *se aleja* de su forma natural de expresarse, **por** esto se utiliza el **lenguaje ensamblador**, que es la forma de expresar las instrucciones de una forma más natural al hombre y que, sin embargo, es muy cercana al microcontrolador porque cada **una** de sus instrucciones se corresponde con otra en código máquina que el microcontrolador es capaz de interpretar.

El lenguaje ensamblador utiliza **nemáticos** que son grupos de caracteres alfanimáticos que simbolizan las órdenes o tareas a realizar con cada instrucción. Los nemáticos se corresponden con las iniciales del nombre de la instrucción en inglés, de forma que "recuerdan" la operación que realiza la instrucción, lo que facilita su memorización.

Así, por ejemplo: para ordenar al microcontrolador PIC16F84: "suma 58 al registro de trabajo W y guarda el resultado en este mismo registro W", en lenguaje ensamblador sería "*addlw d'58'*" que es mucho más amable que el "11111000111010" del lenguaje máquina.

Resumiendo con un ejemplo:

- Instrucción: "Suma 58 al registro de trabajo W y guarda el resultado en este mismo registro".
- Ensamblador: *addlw d'58'*.
- Máquina: 11 1110 0011 1010 (expresado en binario).  
3E3A (expresado en hexadecimal).

## 6.3 PROGRAMA ENSAMBLADOR

El **programa ensamblador** es un **software** que se encarga de traducir los nemáticos y símbolos alfanuméricos del programa escrito en ensamblador por el usuario a código máquina, para que pueda ser interpretado y ejecutado por el microcontrolador.

El programa escrito en lenguaje ensamblador recibe la denominación de **código fuente**, **archivo fuente** o **fichero fuente**. Suele tener la extensión \*.asm. El archivo fuente debe ser traducido a código máquina, de lo cual se encarga el **programa ensamblador**. La mayoría de los ensambladores proporcionan a su salida un fichero que suele tener la extensión \*.hex. Este fichero puede ser grabado en la memoria de programa mediante la utilización de un grabador de microcontroladores.

ero utilizando la  
5-1.

aleja de su forma  
e es la forma de  
sin embargo, es  
s se corresponde  
ar.

s de caracteres  
instrucción. Los  
ón en inglés, de  
que facilita su

4: "suma 58 al  
W", en lenguaje  
111000111010"

nda el resultado

de traducir los  
or por el usuario  
ocontrolador.

ación de código  
asm. El archivo  
ga el programa  
n un fichero que  
ria de programa

El ensamblador más utilizado para los PIC es el **MPASM**, que trabaja dentro de un entorno software denominado **MPLAB** que se describe en el próximo capítulo. Este entorno de trabajo es puesto a libre disposición de los usuarios por *Microchip Technology* en su página Web [www.microchip.com](http://www.microchip.com).

## 6.4 FICHEROS RESULTANTES DEL ENSAMBLADO

Tras el ensamblado del fichero fuente \*.asm se producen varios ficheros (figura 6-2). Los más importantes son:

- **Fichero ejecutable o hexadecimal.** Es un fichero con datos numéricos codificados en hexadecimal. Tiene la extensión \*.hex. Contiene los códigos máquina del programa que servirán para grabar la memoria de programa del microcontrolador y ejecutarlo.
- **Fichero de errores.** Es un fichero con la extensión \*.err. Contiene los errores producidos durante el proceso de ensamblado.
- **Fichero listable.** Es un fichero de texto con la extensión \*.lst que contiene toda la información del programa: código fuente, códigos máquina, direcciones de cada instrucción, errores producidos, etc.

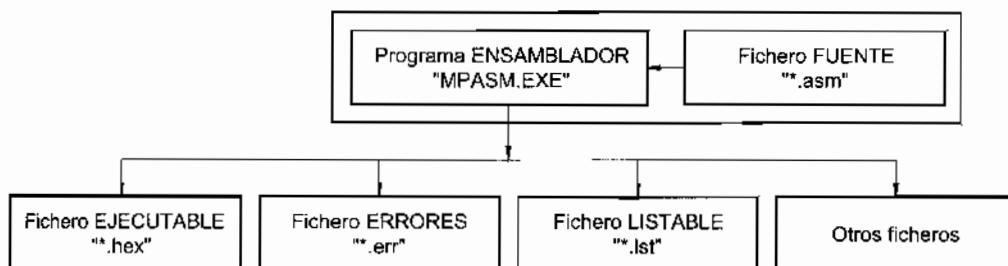


Figura 6-2 Ficheros resultantes del proceso de ensamblado

## 6.5 EL CÓDIGO FUENTE

El código fuente está compuesto por una sucesión de líneas de programa. Todos los ficheros fuente poseen una estructura similar independientemente del procesador utilizado. Cada línea de programa suele estar compuesta por 4 campos o columnas separados por uno o más espacios o tabulaciones. Estos campos son:

- 1º Campo de etiquetas.
- 2º Campo del código de operación.
- 3º Campo de operandos y datos.
- 4º Campo de comentarios.

A continuación se muestra un ejemplo:

1<sup>a</sup>2<sup>a</sup>3<sup>a</sup>4<sup>a</sup> columna

```
***** Ensam_01.ASM *****
;
; Por los LEDs conectados al puerto B se visualiza el valor de una constante, por ejemplo
; el número binario 01010101.
;
; ZONA DE DATOS *****
        _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC ; Configuración para el
; grabador.
        LIST      P=16F84A          ; Procesador utilizado.
        INCLUDE <P16F84A.INC>    ; En este fichero se definen las etiquetas del PIC.

        Constante EQU      b'01010101' ; Por ejemplo.

; ZONA DE CÓDIGOS *****
        ORG      0                  ; El programa comienza en la dirección 0 de la memoria de
; programa.
        Inicio   bsf      STATUS,RP0    ; Pone a 1 el bit 5 del STATUS. Acceso al Banco 1.
        clrf     TRISB               ; Las líneas del puerto B se configuran como salida.
        bcf      STATUS,RP0    ; Pone a 0 el bit 5 de STATUS. Acceso al Banco 0.
        movlw   Constante         ; Carga el registro de trabajo W con la constante.
        Principal
        movwf   PORTB               ; El contenido de W se deposita en el puerto de salida.
        goto    Principal           ; Se crea un bucle cerrado e infinito
        END                      ; Fin del programa
```

1º.

2º.

3º.

## 6.5.2

## 6.5.3

### 6.5.1 Etiquetas

La primera columna es el campo de **etiquetas**. Éstas son expresiones alfanuméricas escogidas por el usuario, su principal función es identificar a una determinada instrucción del programa, asignando a esa etiqueta el valor de la dirección de memoria correspondiente a dicha instrucción. Así, la dirección de memoria se designa en el programa por el nombre de la etiqueta facilitando la identificación de las direcciones, sobre todo en las instrucciones de salto. Por ejemplo:

Principal ....

```
        goto    Principal           ; Salta a la posición de memoria identificada mediante
; la etiqueta "Principal".
```

Asignando etiquetas a direcciones del programa se posibilita que las instrucciones puedan saltar o hacer referencia a esas instrucciones sin necesidad de recordar las direcciones físicas donde están ubicadas. Las etiquetas también tienen otras funciones que se comentarán más adelante.

## 6.5.4

Es obligatorio que las etiquetas cumplan las siguientes normas:

- 1º. Deben empezar por letras y luego admiten letras, números y el carácter subrayado (guión bajo) “\_”. Ejemplo: *Bucle\_I*.
- 2º. No se pueden insertar espacios o tabulaciones antes de la etiqueta, si no será considerada como una instrucción.
- 3º. No pueden utilizarse expresiones reservadas para la utilización del ensamblador tales como:
  - Instrucciones. Por ejemplo: “*goto*”, “*sleep*”, etc.
  - Nombres de registros especiales (SFR). Por ejemplo: “*STATUS*”, “*PCL*”, etc.
  - Nombre de cada uno de los bits de los registros especiales. Por ejemplo: “*C*”, “*Z*”, “*DC*”, etc.
  - Directivas del propio ensamblador. Por ejemplo: “*END*”, “*ORG*”, “*LIST*”, etc.

### 6.5.2 Código de operación

La segunda columna suele ser el campo del **código de operación**, que especifica la tarea a realizar por el microcontrolador. Suele ser una instrucción del microcontrolador que es directamente traducida a código máquina por el ensamblador, por ejemplo, “*sleep*” es traducida a código máquina “00 0000 0110 0011”.

### 6.5.3 Operando

La tercera columna es el campo de **operando o datos**. Contiene los operandos para el campo de instrucciones. Por ejemplo “*movwf PORTB*”, donde “*PORTB*” es el operando. Este campo puede contener uno o más operandos separados por comas. Dependiendo de la instrucción pueden ser números o etiquetas que representen constantes o direcciones, tal como en el programa ejemplo detallado en el apartado anterior.

Conviene que los operandos que designan a los registros de SFR, respeten la ortografía que se establece en el fichero P16F84A.INC que proporciona el fabricante *Microchip*. Este fichero establece por ejemplo que al Puerto B hay que llamarle “*PORTB*”, al registro de estado “*STATUS*”, etc. Debido a su importancia, el fichero P16F84A.INC será expuesto en su integridad en un próximo apartado.

### 6.5.4 Comentarios

La última columna es el campo de **comentarios** de los programas. Son elementos indispensables que ayudan al programador a documentar su programa, aclarar el sentido de las instrucciones y, con el tiempo, facilitar la posibilidad de correcciones y

modificaciones que lo mejoren. Los comentarios no son tenidos en cuenta por el ensamblador y por tanto no se codifican a lenguaje máquina.

Los comentarios pueden extenderse durante varias líneas y todas ellas deben comenzar siempre por punto y coma (;). No necesita tener espacios o tabulaciones que lo separen del campo anterior, e incluso puede empezar en la primera posición de la línea. El ensamblador ignora todo el texto que contenga la línea después de un carácter punto y coma, de esta manera pueden incluirse líneas que contengan sólo comentarios.

Cuando el ensamblador encuentra un punto y coma (;) lo interpreta como comentario y no genera código máquina. Es frecuente, utilizar esta técnica en la depuración de programas para anular temporalmente instrucciones que no se desean ejecutar pero que tampoco se desean borrar definitivamente.

El objetivo final de la programación será, además de conseguir programas que funcionen correctamente y sean eficientes, proporcionar sobre ellos toda la documentación necesaria. Debe tenerse en cuenta que un programa indebidamente documentado sería muy difícil de modificar en un futuro, o al menos podría resultar tan costoso que haría renunciar a tal propósito. Es buena práctica hacer uso y abuso de esta posibilidad para que los programas resulten suficientemente documentados. Con ello se consigue que los programas fuentes ganen en claridad, aunque a costa de aumentar su extensión. En este libro se ha “abusado” de los comentarios por motivos pedagógicos.

### 6.5.5 Normas de estilo para escribir un archivo fuente

Para simplificar la codificación del programa fuente, los ensambladores permiten que cada línea pueda estar escrita en formato libre, esto es, puede haber cualquier número de espacios entre dos campos de una línea y se puede utilizar mayúsculas o minúsculas a criterio del diseñador de programas. Sin embargo, cuando se escribe un archivo fuente hay una serie de normas de estilo comunes entre los programadores, que si bien no son obligatorias, facilitan su lectura como puede verse en el ejemplo del código fuente detallado en el apartado anterior. Éstas son:

- Conviene respetar la designación de los registros, tal como establece el fichero P16F84A.INC descrito más adelante.
- Se deben respetar las columnas. Para una mejor legibilidad del programa, se recomienda utilizar los tabuladores para definir las columnas de cada campo.
- Conviene numerar todas las filas, para un seguimiento más sencillo y una mejor corrección del archivo fuente. Normalmente es numerado de forma automática por el programa editor.
- Los espacios en blanco no son significativos en ningún campo, como tampoco lo son ni las líneas en blanco ni los tabuladores, sólo sirven para darle forma al texto.

en cuenta por el  
odas ellas deben  
bulaciones que lo  
sición de la línea.  
carácter punto y  
arios.

interpreta como  
ta técnica en la  
ue no se desean

r programas que  
ellos toda la  
a indebidamente  
odría resultar tan  
o y abuso de esta  
dos. Con ello se  
i de aumentar su  
pedagógicos.

ente

ladores permiten  
cualquier número  
s o minúsculas a  
n archivo fuente  
e si bien no son  
el código fuente

ablece el fichero

el programa, se  
cada campo.

sencillo y una  
rado de forma

como tampoco  
a darle forma al

- Los nemónicos de las instrucciones se escriben en minúsculas. Ejemplo: `movlw d'15'`.
- El nombre de la etiqueta debe aclarar en lo posible el funcionamiento del programa. Deben ser reales y referidos al programa. La tendencia actual en programación es que las etiquetas sean lo suficientemente descriptivas por sí mismas, aunque resulten un poco extensas. Para ello se juntan las palabras de una frase explicativa, comenzando la primera letra de cada palabra por mayúscula y el resto serán minúsculas. Por ejemplo, "MotorRotaDerecha".

## 6.6 CONSTANTES NUMÉRICAS Y ALFANUMÉRICAS

El ensamblador MPASM soporta los sistemas de numeración decimal, hexadecimal, octal, binario y el código alfanumérico ASCII. La tabla 6-1 representa la forma de especificar el sistema de numeración o códigos alfanuméricos, con un ejemplo por caso.

TIPO	SINTAXIS	EJEMPLO
Decimal	D'<cantidad>' d'<cantidad>' .<cantidad>	Movlw D'109' movlw d'109' movlw .109
Hexadecimal	H'<cantidad>' h'<cantidad>' 0x<cantidad> <cantidad>H <cantidad>h	Movlw H'6D' movlw h'6D' movlw 0x6D movlw 6DH movlw 6Dh
Octal	O'<cantidad>' o'<cantidad>'	Movlw O'155' movlw o'155'
Binario	B'<cantidad>' b'<cantidad>'	Movlw B'01101101' movlw b'01101101'
ASCII	A'<carácter>' a'<carácter>' '<carácter>'	Movlw A'M' movlw a'M' movlw 'M'
"String" o Cadena de Caracteres.	"<string>"	DT "Estudia DPE"

Tabla 6-1 Formato de las constantes

Las constantes hexadecimales que comienzan por una letra (A-F) deben ir precedidas de un cero para que no sean confundidas con una etiqueta, ejemplo: `movlw`

*0FAh*. Las constantes pueden ser opcionalmente precedidas por un signo "+" (valores positivos) o "-" (valores negativos), si no aparece signo alguno se asume que el valor es positivo. Cuando los operandos son caracteres ASCII, deben estar encerrados entre comillas simples, ejemplo: *movlw 'G'*.

## 6.7 OPERADORES ARITMÉTICOS

En los operandos de las instrucciones pueden aparecer expresiones matemáticas cuyo valor numérico será calculado por el programa ensamblador en el momento de ensamblar el programa fuente. Por ejemplo:

```

    movlw  MensajeFin-MensajeInicio ; El ensamblador calcula automáticamente la longitud
                                     ; del mensaje realizando una resta.
                                     ; Resto del programa.

MensajeInicio ..... ....
                                     ; (Aquí iría un mensaje de longitud desconocida).

MensajeFin ..... ....

```

En el apéndice C se detalla una lista con los principales operadores lógicos y matemáticos.

## 6.8 EL REPERTORIO DE INSTRUCCIONES

El repertorio del PIC16F84 está compuesto por 35 instrucciones que pueden ser agrupadas para su estudio en los siguientes grupos:

- Instrucciones de carga.
- Instrucciones aritméticas.
- Instrucciones lógicas.
- Instrucciones de bit.
- Instrucciones de salto.
- Instrucciones para manejo de subrutinas.
- Instrucciones especiales.

Las principales características del repertorio de instrucciones del PIC16F84 son:

- Es un juego reducido de 35 instrucciones simples y rápidas.
- La mayoría de las instrucciones se ejecutan en 4 ciclos de reloj, menos las de salto que requieren 8 ciclos.
- Las instrucciones son ortogonales. Casi todas las instrucciones pueden usar cualquier operando.
- Todas las instrucciones tienen la misma longitud, 14 bits y todos los datos son de 8 bits.

NEMC	
clrf	
clrw	
movf	
movlw	
movvw	
bcf	
bsf	
addlw	
addwf	
decf	
incf	
sublw	
subwf	
andlw	
andwf	
comf	
iorlw	
iorwf	
rlf	
rrf	
swapf	
xorlw	
xorwf	
btfsc	
btfss	
decfsz	
incfsz	
goto	

NEMÓNICO	DESCRIPCIÓN	CÓDIGO DE OPERACIÓN	FLAGS AFECTADOS
Instrucciones de CARGA			
clrf f	00 → (f)	00 0001 1fff ffff	Z
clrwf	00 → (W)	00 0001 0xxx xxxx	Z
movf f,d	(f) → (destino)	00 1000 dfff ffff	Z
movlw k	k → (W)	11 00xx kkkk kkkk	Ninguno
movwf f	(W) → (f)	00 0000 1fff ffff	Ninguno
Instrucciones de BIT			
bcf f,b	Pone a 0 el bit 'b' del reg. 'f'.	01 00bb bfff ffff	Ninguno
bsf f,b	Pone a 1 el bit 'b' del reg. 'f'.	01 01bb bfff ffff	Ninguno
Instrucciones ARITMÉTICAS			
addlw k	(W) + k → (W)	11 111x kkkk kkkk	C, DC, Z
addwf f,d	(W) + (f) → (destino)	00 0111 dfff ffff	C, DC, Z
decf f,d	(f) - 1 → (destino)	00 0011 dfff ffff	Z
incf f,d	(f) + 1 → (destino)	00 1010 dfff ffff	Z
sublw k	k - (W) → W	11 110x kkkk kkkk	C, DC, Z
subwf f,d	(f) - (W) → (destino)	00 0010 dfff ffff	C, DC, Z
Instrucciones LÓGICAS			
andlw k	(W) AND k → (W)	11 1001 kkkk kkkk	Z
andwf f,d	(W) AND (f) → (destino)	00 0101 dfff ffff	Z
comf f,d	(/f) → (destino)	00 1001 dfff ffff	Z
iorlw k	(W) OR k → (W)	11 1000 kkkk kkkk	Z
iorwf f,d	(W) OR (f) → (destino)	00 0100 dfff ffff	Z
rlf f,d	Rota (f) a izquierda a través del Carry → (destino)	00 1101 dfff ffff	C
rff f,d	Rota (f) a derecha a través del Carry → (destino)	00 1100 dfif ffff	C
swapf f,d	Intercambia los nibbles de f → (destino)	00 1110 dfff ffff	Ninguno
xorlw k	(W) XOR k → (W)	11 1010 kkkk kkkk	Z
xorwf f,d	(W) XOR (f) → (destino)	00 0110 dfff ffff	Z
Instrucciones de SALTO			
btfsc f,b	Salta si el bit 'b' del 'f' es 0	01 10bb bfff ffff	Ninguno
btfss f,b	Salta si el bit 'b' del 'f' es 1	01 11bb bfff ffff	Ninguno
decfsz f,d	(f)-1 → destino y salta si es 0	00 1011 dfff ffff	Ninguno
incfsz f,d	(f)+1 → destino y salta si es 0	00 1111 dfff ffff	Ninguno
goto k	Salta a la dirección 'k'	10 1kkk kkkk kkkk	Ninguno

Instrucciones de manejo de SUBRUTINAS				
call k	Llamada a subrutina	10 0kkk kkkk kkkk	Ninguno	
retfie	Retorno de una interrupción	00 0000 0000 1001	Ninguno	
retlw k	Retorno con un literal en W	11 01xx kkkk kkkk	Ninguno	
return	Retorno de una subrutina	00 0000 0000 1000	Ninguno	
Instrucciones ESPECIALES				
clrwdt	Borra Timer del Watchdog	00 0000 0110 0100	/TO, /PD	
nop	No operación	00 0000 0xx0 0000	Ninguno	
sleep	Entra en modo bajo consumo	00 0000 0110 0011	/TO, /PD	

Tabla 6-2 Repertorio de instrucciones del PIC16F84

Las instrucciones se recogen en la tabla 6-2 y detallan en su totalidad en el apéndice B. En este capítulo se estudiarán las instrucciones más sencillas.

## 6.9 INSTRUCCIONES DE CARGA

Las instrucciones de transferencias de datos son típicas de todos los procesadores y su misión es transferir el contenido un registro fuente (f) a un registro destino (d) o bien cargar el destino con una constante. En los microcontroladores PIC todos los datos residen en posiciones de la memoria de datos y en el registro de trabajo W.

En la explicación de estas instrucciones se emplea muchas veces una nomenclatura especial muy simple basada en paréntesis y flechas. Con los paréntesis se destaca que se trata del "contenido" de las posiciones de memoria y la flecha la dirección de la transferencia de los datos. Algunos ejemplos:

- $(W) \rightarrow (PORTB)$ : Significa "el contenido del registro W se transfiere al puerto B".
- $(2Bh) \rightarrow (W)$ : Significa "el contenido de la posición 2Bh de RAM de datos se transfiere al registro de trabajo W".
- $2Bh \rightarrow (W)$ : Significa "el registro de trabajo se carga con el dato 2Bh". (Notar la ausencia de paréntesis).

Hay cinco instrucciones de carga propiamente dichas:

### 6.9.1 clrw

(Clear W). El contenido del registro W se borra (se carga con b'00000000') y el flag Z se activa a "1". Esta instrucción también se podría considerar como aritmética.

Ninguno  
Ninguno  
Ninguno  
Ninguno

/TO, /PB  
Ninguno  
*m*, /PD

a totalidad en el

os procesadores y  
destino (d) o bien  
' todos los datos  
v.

una nomenclatura  
se destaca que se  
dirección de la

se transfiere al  
IBA de RAM de  
W'.  
rga con el dato

'00000000') y el  
o aritmética.

Ejemplo: ***clrw*** ; 0 → (W)  
Antes instrucción: (W) = ? y Z = ?  
Después instrucción: (W) = 0x00 y Z = 1.

### 6.9.2 clrf f

(Clear f). El contenido del registro 'f' se borra (se carga con b'00000000') y el flag Z se activa a uno. Esta instrucción también se podría considerar como aritmética.

Ejemplo: ***clrf FlagReg*** ; 0 → (FlagReg)  
Antes instrucción: (FlagReg) = ? y Z = ?  
Después instrucción: (FlagReg) = 0x00 y Z = 1.

### 6.9.3 movlw k

(Move Literal to W). El registro W se carga con el valor de los 8 bits de la constante 'k'. Ningún flag del registro de estado es afectado.

Ejemplo: ***movlw 0x5A*** ; 5Ah → (W)  
Antes instrucción: (W) = ?  
Después instrucción: (W) = 0x5A

### 6.9.4 movf f,d

(Move f). El contenido del registro 'f' se carga en el registro destino dependiendo del valor de 'd'. Si 'd' = 0 el destino es el registro W, si 'd' = 1 el destino es el propio registro 'f'. El flag Z del registro STATUS queda afectado: Z se activa a "1" si el resultado de la operación es cero.

Ejemplo 1: ***movf PORTA, 0*** ; (PORTA) → (W)  
Antes instrucción: (PORTA) = 0x1A, (W) = ? y Z = ?  
Después instrucción: (PORTA) = 0x1A, (W) = 0x1A y Z = 0

Ejemplo 2: ***movf FSR, 1*** ; (FSR) → (FSR)  
Antes instrucción: (FSR) = 0x00, y Z = ?  
Después instrucción: (FSR) = 0x00, y Z = 1 (el resultado de la operación es "0").

### 6.9.5 movwf f

(Move W to f). Carga el contenido del registro W al registro 'f'. Ningún flag del registro de estado es afectado.

Ejemplo: ***movwf PORTB*** ; (W) → (PORTB)  
Antes instrucción: (PORTB) = ?, y (W) = 0x4F.  
Después instrucción: (PORTB) = 0x4F y (W) = 0x4F

## 6.10 INSTRUCCIONES DE BIT

Estas instrucciones ponen a nivel lógico "0" ó "1" un determinado bit de un registro de la memoria de datos.

### 6.10.1 bcf f,b

(*Bit Clear f*). Pone a cero el bit 'b' del registro 'f'.

Ejemplo:	<i>bcf FlagReg,7</i>	; 0 → (FlagReg,7)
Antes instrucción:	(FlagReg) = b'11000111'.	
Después instrucción:	(FlagReg) = b'01000111'.	

### 6.10.2 bsf f,b

(*Bit Set f*). Pone a uno el bit 'b' del registro 'f'.

Ejemplo:	<i>bsf FlagReg,7</i>	; 1 → (FlagReg,7)
Antes instrucción:	(FlagReg) = b'01000111'.	
Después instrucción:	(FlagReg) = b'11000111'.	

## 6.11 INSTRUCCIÓN "GOTO K"

En casi todos los programas se usa la instrucción de salto incondicional *goto k* (*Unconditional Branch*) que produce un salto a la dirección del programa indicada por "k". La constante literal "k" es la dirección de destino del salto, es decir, la nueva dirección de memoria de programa a partir de la cual comenzarán a leerse las instrucciones después de ejecutar la instrucción *goto*. Así pues, esta instrucción simplemente carga la constante k en el contador de programa (PC).

Ejemplo:	<i>goto Bucle</i>	; Bucle → (PC)
Antes instrucción:	(PC) = ?	
Después instrucción:	(PC) = Dirección apuntada por la etiqueta "Bucle".	

A veces se utiliza el carácter de control dólar (\$) para señalar que el salto se produce a la misma posición de programa donde se encuentra situado en ese momento. Así por ejemplo, las dos próximas instrucciones son equivalentes:

AquiMismo	<i>goto</i>	AquiMismo
	<i>goto</i>	\$

## 6.12 CO

Al c  
automáti  
la memoria

Para  
TRISB qu  
previamente  
Una vez co  
STATUS p  
entradas o p  
valores se h  
y Puerto B

A co  
entrada y e  
al circuito

\*\*\*\*\*  
; Por el Puerto  
; un array de i  
; el Puerto B a

; ZONA DE D

LIS  
INC

; ZONA DE C

ORC

Inicio

bsf

clrf

mov

mov

bcf

Principal

mov

mov

goto

END

## 6.12 CONFIGURAR LAS LÍNEAS DE LOS PUERTOS

Al conectar por primera vez el PIC16F84 el bit RP0 del registro STATUS se carga automáticamente con “0”, con lo que se permite el acceso a las posiciones del Banco 0 de la memoria de datos.

Para configurar las líneas de los puertos hay que acceder a los registros TRISA y TRISB que se hallan en el Banco 1 y cargar en ellos los valores adecuados. Por ello previamente hay que acceder al Banco 1 poniendo a “1” el bit RP0 del registro STATUS. Una vez configurados los puertos, habrá que volver a poner a “0” el bit RP0 del registro STATUS 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, que son las direcciones del Puerto A y Puerto B respectivamente (figura 4-1)

A continuación se muestra un ejemplo, donde el Puerto A se configura como entrada y el Puerto B se configura como salida. Este programa ejemplo se puede aplicar al circuito de la figura 6-3.

```
***** Ensam_03.asm *****
;
; Por el Puerto B se saca el dato de las cinco líneas del Puerto A al que están conectado
; un array de interruptores. Por ejemplo, si por el Puerto A se introduce "—11001", por
; el Puerto B aparecerá "xxx11001" (el valor de las tres líneas superiores no importa).
;
; ZONA DE DATOS *****
;
; _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC ; Configuración para el
; grabador.
;
LIST      P=16F84A          ; Procesador utilizado.
INCLUDE <P16F84A.INC>    ; Definición de algunos operandos utilizados.
;
; ZONA DE CÓDIGOS *****
;
ORG      0                  ; El programa comienza en la dirección 0 de memoria de
                            ; programa.
;
Inicio   bsf    STATUS,RP0    ; Acceso al Banco 1.
        clrf   TRISB           ; Las líneas del Puerto B se configuran como salidas.
        movlw  b'00011111'       ; Las 5 líneas del Puerto A se configuran como entradas.
        movwf  TRISA           ; Acceso al Banco 0.
;
Principal
        bcf    STATUS,RP0
        movf   PORTA,W          ; Carga el registro de datos del Puerto A en W.
        movwf  PORTB           ; El contenido de W se deposita en el Puerto B.
        goto   Principal        ; Se crea un bucle cerrado e infinito.
;
        END                ; Fin del programa.
```

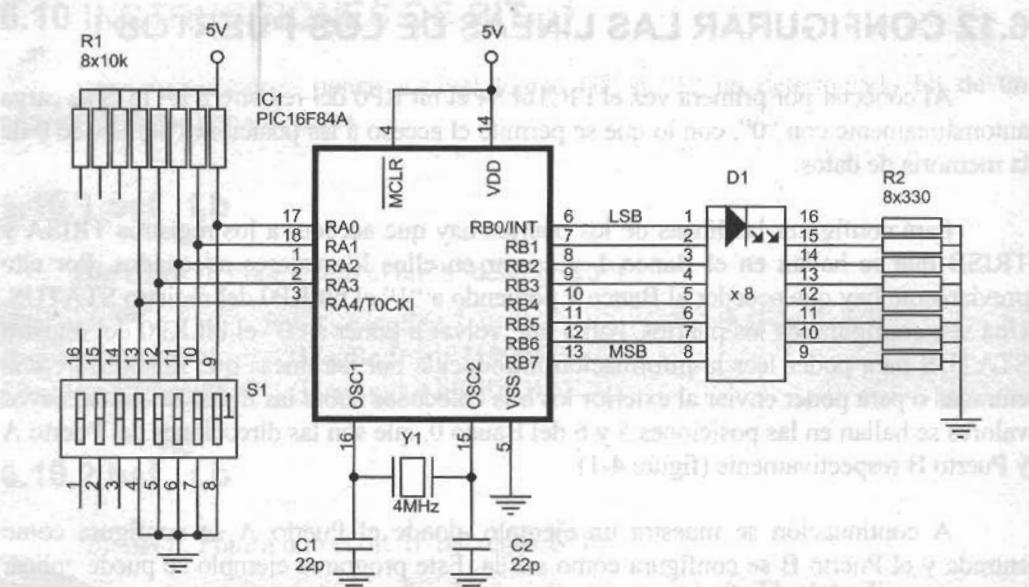


Figura 6-3 Circuito para comprobar el programa *Ensam\_03.asm*

Los registros TRISA y TRISB ocupan las posiciones 5 y 6 del Banco 1. Los puertos PORTA y PORTB ocupan las posiciones 5 y 6 del Banco 0 (figura 4-1), por tanto, el programa anterior se puede modificar como sigue:

```
***** Ensam_03.asm *****
;
; (Principio como el anterior)
;
Inicio    bsf      STATUS,RP0      ; Acceso al Banco 1.
          clrf     PORTB        ; Las líneas del Puerto B se configuran como salidas.
          movlw    b'00011111'
          movwf    PORTA        ; Las 5 líneas del Puerto A se configuran como entradas.
          bcf      STATUS,RP0      ; Acceso al Banco 0.
Principal
;
; (Continua igual que el anterior)
```

Según la tabla 6-2 el código máquina para la instrucción *clrf f* es 0000011fffffff, es decir, sólo se toman los siete bits bajos de la dirección del registro *f*. Como la dirección del TRISB es 86h (*b'10000110'*) y la del PORTB es 06h (*b'00000110'*) sus siete bits más bajos son idénticos y, por tanto, las instrucciones *clrf TRISB* y *clrf PORTB* tienen exactamente el mismo código máquina que es el 00000110000110 (186h). Lo mismo se puede decir para las instrucciones *movwf TRISA* y *movwf PORTA* que tienen el código máquina 00000010000101 (085h).

En el  
internos pas  
debe añadir

```
*****
;
; Inicio    clrf
;           bsf
;           clrf
;           mov
;           mov
;           bcf
Principal
```

## 6.13 DIR

Para  
información  
controla el  
microproces  
microcontrol

MPAS  
50 directivas  
GUIDE" qu  
www.microc  
libro. Algun  
programas, p  
realización d  
columna y en

### 6.13.1 EN

(End F  
Al ensambl  
el proceso, p  
programa y  
posteriores a  
ensamblan. E

```
Inicio    bsf
END
```

En el momento de la definición del puerto como salida, el contenido de sus latches internos pasan al exterior, en caso de que sea importante que este sea cero el programa debe añadir una instrucción *clr PORTB* al principio:

***** Ensam_03.asm *****			
;			
;			
...			
Inicio	clrf	PORTB	; Inicializa el puerto para que la línea vaya a nivel bajo ; cuando ésta se configure como salida.
	bsf	STATUS,RP0	; Acceso al Banco 1.
	clrf	TRISB	; Las líneas del Puerto B se configuran como salidas.
	movlw	b'0001111'	
	movwf	TRISA	; Las 5 líneas del Puerto A se configuran como entradas.
	bcf	STATUS,RP0	; Acceso al Banco 0.
Principal			

## **6.13 DIRECTIVAS**

Para poder ensamblar un programa automáticamente el ensamblador necesita información en forma de **directivas**, que son comandos insertados en el programa que controla el proceso de ensamblado. No son parte del repertorio de instrucciones del microprocesador y, por lo tanto, no tienen traducción al código máquina del microcontrolador, por lo que también se les llama pseudoinstrucciones.

MPASM es el programa ensamblador del PIC16F84 más utilizado, tiene más de 50 directivas que se explican en detalle en su menú *help* y en la guía “*MPASM. USER'S GUIDE*” que se puede obtener gratuitamente en la página Web del fabricante [www.microchip.com](http://www.microchip.com). Las principales directivas se explican en el apéndice D de este libro. Algunas de estas directivas deben ser utilizadas obligatoriamente en todos los programas, pero la mayoría son opcionales y sirven para facilitar el desarrollo y la realización del programa. Las directivas del ensamblador suelen escribirse en la segunda columna y en mayúsculas, aunque no es obligatorio. Las más utilizadas son:

### **6.13.1 END**

*(End Program Block).* Indica el fin del programa. Es la única directiva obligatoria. Al ensamblar un programa, el programa ensamblador debe saber dónde tiene que detener el proceso, para eso está la directiva *END*, la cual debe estar en la última línea del programa y explícitamente le indica al ensamblador el fin de aquél. Todas las líneas posteriores a la línea en la que se encuentra esta directiva son ignoradas y no se ensamblan. Ejemplo:

Inicio      bsf      STATUS,RP0      ; Comienza el programa ejecutable  
               ...      ...  
               END      ; Fin del programa

### 6.13.2 EQU

(*Define an Assembler Constant*). Su sintaxis es:

*<label> EQU <expr>*

Es una directiva de asignación. El valor de *<expr>* es asignado a la etiqueta *<label>*. Usualmente las asignaciones con *EQU* van al principio del programa (antes de las instrucciones). Siempre que el nombre aparece en el programa es sustituido por el valor numérico de la expresión que se le haya asignado. Ejemplo:

```
ValorCarga EQU d'147' ; Asigna el valor numérico de 147 a la etiqueta "ValorCarga".
```

Generalmente la etiqueta es un nombre que describe el valor de manera más significativa para el programador. Suele utilizarse para definir constantes y direcciones de memoria. Así, es más fácil recordar "ValorCarga" que su valor 147, o en el caso de una dirección de memoria PORTA que 0x05.

### 6.13.3 ORG

(*Set Program Origin*). Su sintaxis es:

*[<label>] ORG <expr>*

Esta directiva indica al programa ensamblador la dirección en memoria de programa a partir de la cual deben ensamblarse las instrucciones del código fuente. Es decir, la dirección de la memoria de programa donde se van a almacenar esas instrucciones es la fijada por la *<expr>* de la directiva. Ejemplo:

```
ORG 0x04
```

; Los códigos máquinas correspondientes a las instrucciones que aparecen en las líneas siguientes, deben ser ensamblados en la memoria de programa a partir de la dirección 4.

Si las instrucciones de un programa comienzan a escribirse sin indicar *ORG*, el ensamblador toma por defecto *ORG 0x00*.

Cuando las distintas secciones de un programa deben colocarse en diferentes áreas de memoria se usa una directiva *ORG* antes de cada parte del programa para especificar la dirección inicial u origen de esa porción del programa. Es decir, en un programa pueden aparecer varios *ORG* dependiendo de las necesidades.

En el ejemplo del fichero *Ensam\_03.ASM* se aprecia la utilización de otras tres directivas:

- N
- N
- S
- S

### 6.13.4 \_\_ CONFIG

En el programa Ensam\_03.asm aparece como:

```
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
```

Esta directiva indica la configuración elegida para el proceso de grabación del microcontrolador. En este caso:

- No hay protección de código (\_CP\_OFF).
- No se habilita el Watchdog (\_WDT\_OFF).
- Se habilita el reset mediante Power-Up Timer (\_PWRTE\_ON).
- Se utiliza el oscilador por cristal de cuarzo. (\_XT\_OSC).

Es importante resaltar que “\_\_CONFIG” se inicia con dos subrayados (guión bajo), no con uno. Este error es muy frecuente en los primeros programas.

### 6.13.5 LIST P=16F84A

Indica el tipo de procesador utilizado.

### 6.13.6 INCLUDE <P16F84A.INC>

Indica el fichero donde se localizan las etiquetas que nombran a los diferentes registros y el valor que le corresponde a cada uno, es decir, en el fichero P16F84A.INC se muestra como hay que nombrar a todos los registros propios del microcontrolador. Este fichero se localiza en el directorio principal del programa ensamblador y su contenido es el siguiente:

```
LIST
; P16F84A.INC Standard Header File, Version 2.00 Microchip Technology, Inc.
; NOLIST

; This header file defines configurations, registers, and other useful bits of
; information for the PIC16F84 microcontroller. These names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included. The processor may be selected the following ways:

; 1. Command line switch:
; C:\MPASM\myfile.asm /PIC16F84A
; 2. LIST directive in the source file
; LIST P=PIC16F84A
; 3. Processor Type entry in the MPASM full-screen interface
```

### 6.13.4 \_\_ CONFIG

En el programa Ensam\_03.asm aparece como:

```
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
```

Esta directiva indica la configuración elegida para el proceso de grabación del microcontrolador. En este caso:

- No hay protección de código (\_CP\_OFF).
- No se habilita el Watchdog (\_WDT\_OFF).
- Se habilita el reset mediante Power-Up Timer (\_PWRTE\_ON).
- Se utiliza el oscilador por cristal de cuarzo. (\_XT\_OSC).

Es importante resaltar que “**\_\_CONFIG**” se inicia con dos subrayados (guión bajo), no con uno. Este error es muy frecuente en los primeros programas.

### 6.13.5 LIST P=16F84A

Indica el tipo de procesador utilizado.

### 6.13.6 INCLUDE <P16F84A.INC>

Indica el fichero donde se localizan las etiquetas que nombran a los diferentes registros y el valor que le corresponde a cada uno, es decir, en el fichero P16F84A.INC se muestra como hay que nombrar a todos los registros propios del microcontrolador. Este fichero se localiza en el directorio principal del programa ensamblador y su contenido es el siguiente:

```
LIST
; P16F84A.INC Standard Header File, Version 2.00 Microchip Technology, Inc.
; NOLIST

; This header file defines configurations, registers, and other useful bits of
; information for the PIC16F84 microcontroller. These names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file is
; included. The processor may be selected the following ways:

; 1. Command line switch:
; C:\MPASM\myfile.asm /PIC16F84A
; 2. LIST directive in the source file
; LIST P=PIC16F84A
; 3. Processor Type entry in the MPASM full-screen interface
```

## Revision History

Rev: Date: Reason:

1.00 2/15/99 Initial Release

## Verify Processor

IFNDEF \_16F84A

MESSG "Processor-header file mismatch. Verify selected processor."  
ENDIF

## Register Definitions

W	EQU	H'0000'
F	EQU	H'0001'

## ---- Register Files ----

INDF	EQU	H'0000'
TMR0	EQU	H'0001'
PCL	EQU	H'0002'
STATUS	EQU	H'0003'
FSR	EQU	H'0004'
PORTA	EQU	H'0005'
PORTB	EQU	H'0006'
EEDATA	EQU	H'0008'
EEADR	EQU	H'0009'
PCLATH	EQU	H'000A'
INTCON	EQU	H'000B'
OPTION_REG	EQU	H'0081'
TRISA	EQU	H'0085'
TRISB	EQU	H'0086'
EECON1	EQU	H'0088'
EECON2	EQU	H'0089'

## ---- STATUS Bits ----

IRP	EQU	H'0007'
RP1	EQU	H'0006'
RP0	EQU	H'0005'
NOT_TO	EQU	H'0004'

NOT\_PD  
Z  
DC  
C

;---- INTCO

GIE

EEIE

TOIE

INTE

RBIE

TOIF

INTF

RBIF

;---- OPTI

NOT\_RBPU

INTEDG

T0CS

T0SE

PSA

PS2

PS1

PS0

;---- EECO

EEIF

WRERR

WREN

WR

RD

;---- RA

;---- Co

CP\_ON  
CP\_OFF  
PWRTE\_ON  
PWRTE\_OFF

NOT_PD	EQU	H'0003'
Z	EQU	H'0002'
DC	EQU	H'0001'
C	EQU	H'0000'

;<--- INTCON Bits -----

GIE	EQU	H'0007'
EEIE	EQU	H'0006'
TOIE	EQU	H'0005'
INTE	EQU	H'0004'
RBIE	EQU	H'0003'
TOIF	EQU	H'0002'
INTF	EQU	H'0001'
RBIF	EQU	H'0000'

;<--- OPTION\_REG Bits -----

NOT_RBPU	EQU	H'0007'
INTEDG	EQU	H'0006'
T0CS	EQU	H'0005'
T0SE	EQU	H'0004'
PSA	EQU	H'0003'
PS2	EQU	H'0002'
PS1	EQU	H'0001'
PS0	EQU	H'0000'

;<--- EECON1 Bits -----

EEIF	EQU	H'0004'
WRERR	EQU	H'0003'
WREN	EQU	H'0002'
WR	EQU	H'0001'
RD	EQU	H'0000'

;  
;  
; RAM Definition  
;

MAXRAM	EQU	H'CF'
BADRAM	EQU	H'07', H'50'-H'7F', H'87'

;  
;  
; Configuration Bits  
;

CP_ON	EQU	H'000F'
CP_OFF	EQU	H'3FFF'
PWRTE_ON	EQU	H'3FF7'
PWRTE_OFF	EQU	H'3FFF'

_WDT_ON	EQU	H'3FFF'
_WDT_OFF	EQU	H'3FFB'
_LP_OSC	EQU	H'3FFC'
_XT_OSC	EQU	H'3FFD'
_HS_OSC	EQU	H'3FFE'
_RC_OSC	EQU	H'3FFF'

LIST

Es interesante hacer notar los valores que toman los caracteres "W" y "F", que son "0" y "1" respectivamente. Esto permite una mayor facilidad para indicar el destino del resultado de las instrucciones. Así por ejemplo:

- La instrucción *movf Registro,0* es igual que *movf Registro,W*.
- La instrucción *movf Registro,1* es igual que *movf Registro,F*.

## 7.1

Desarrollar

con mi  
ensambl  
de dato  
ejecuci

•  
•  
•  
•

E  
[www.m](http://www.m)  
Su insta  
operativ

"F", que son  
el destino del

## CAPÍTULO 7

# MPLAB

### 7.1 ENTORNO MPLAB

El **MPLAB IDE** es un software de “Entorno de Desarrollo Integrado” (*Integrated Development Environment, IDE*) que se ejecuta bajo Windows. Con este entorno se puede desarrollar aplicaciones para los microcontroladores PIC.

El MPLAB incluye todas las utilidades necesarias para la realización de proyectos con microcontroladores PIC, permite editar el archivo fuente del proyecto, además de ensamblarlo y simularlo en pantalla para comprobar como evolucionan tanto la memoria de datos RAM, como la de programa ROM, los registros del SFR, etc. según progresá la ejecución del programa.

El MPLAB incluye:

- Un editor de texto.
- Un ensamblador llamado **MPASM**.
- Un simulador llamado **MPLAB SIM**.
- Un organizador de proyectos.

Este programa es gratuito. Se puede bajar en la dirección Internet del fabricante [www.microchip.com](http://www.microchip.com). También se ha incluido en el CD-ROM que acompaña a este libro. Su instalación es muy sencilla y similar a cualquier otro programa para el sistema operativo Windows.

## 7.2 PRIMEROS PASOS CON MPLAB IDE

Una vez que el programa está correctamente instalado, los primeros pasos a seguir para trabajar con el *MPLAB IDE v. 6.xx* son los siguientes. Sugerimos al lector que los vaya probando en su ordenador según se va explicando:

- 1º Con el Explorador de Windows acceder a la carpeta "Mis Documentos" y dentro de ella crear una nueva que se llamará "PIC16F84A" (u otro nombre que considere el lector), donde se irán guardando todos los programas que se vayan diseñando. La trayectoria absoluta o *path* del fichero no puede superar la longitud máxima de 62 caracteres, esto es importante tenerlo en cuenta si se trabaja en Windows 2000 o XP donde los *path* absolutos suelen ser bastante largos. Así pues, el subdirectorio donde se guardarán los ejercicios será del tipo "C:/Mis documentos/PIC16F84A" o similar.
  
- 2º Iniciar el programa actuando sobre el ícono correspondiente a MPLAB situado en el escritorio. Se entrará en una pantalla similar a la figura 7-1.

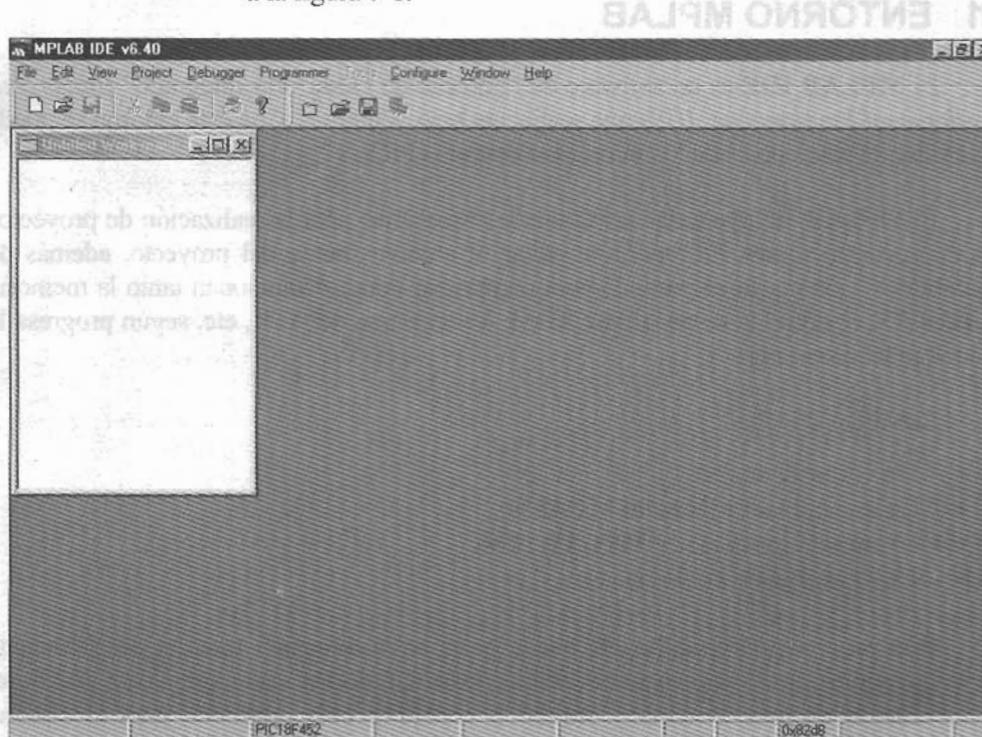


Figura 7-1 Pantalla de inicio del MPLAB IDE

neros pasos a seguir nos al lector que los

“documentos” y dentro u otro nombre que gramas que se vayan e superar la longitud enta si se trabaja en bastante largos. Así erá del tipo “C:/Mis

o correspondiente a una pantalla similar

A continuación se entra en la pantalla de edición, que al maximizar la hoja de trabajo queda como la figura 7-2.

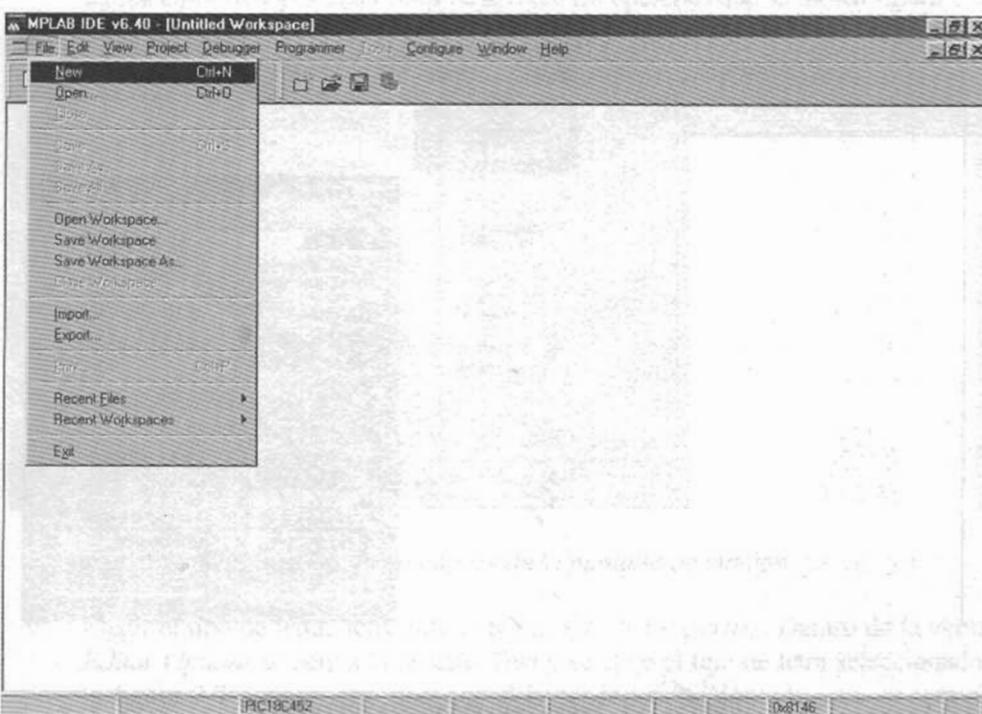


Figura 7-2 Pantalla de edición de programa en el MPLAB IDE

- 3º Elegir el tipo de microcontrolador. Para ello acceder al menú *Configure > Select Device* y seleccionar *PIC16F84A*, tal como se muestra en la figura 7-3.
- 4º A continuación es conveniente seleccionar el simulador, para ello activar el menú *Debugger > Select Tool > MPLAB SIM* (figura 7-4).
- 5º La frecuencia de trabajo del MPLAB SIM debe coincidir con la del circuito a simular. Para seleccionarlo acceder a *Debugger > Settings > Clock* y después comprobar que está a 4 MHz (figura 7-5).
- 6º Crear un nuevo archivo fuente, para ello ir al menú *File > New* (ver figura 7-2). Se entrará en la pantalla de edición en blanco donde se puede escribir el primer programa.
- 7º A continuación se da nombre al fichero fuente accediendo al menú *File > Save As...*. Aparece un cuadro de diálogo que solicita el nombre del archivo. Se puede nombrar por ejemplo “Ensam\_03.asm” y se guarda en la carpeta “C:/Mis

documentos/PIC16F84A" creada anteriormente o en aquella otra que el lector haya determinado.

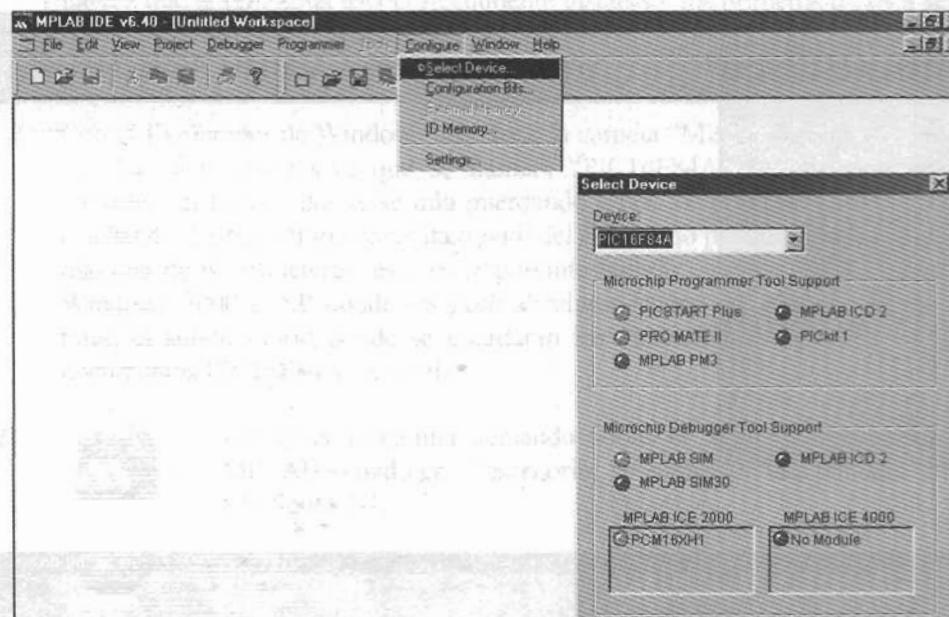


Figura 7-3 Selección del microcontrolador

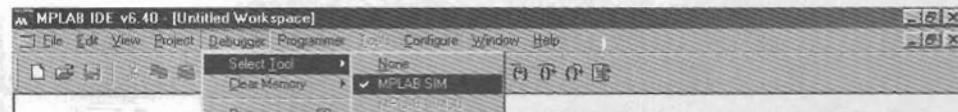


Figura 7-4 Selección del simulador

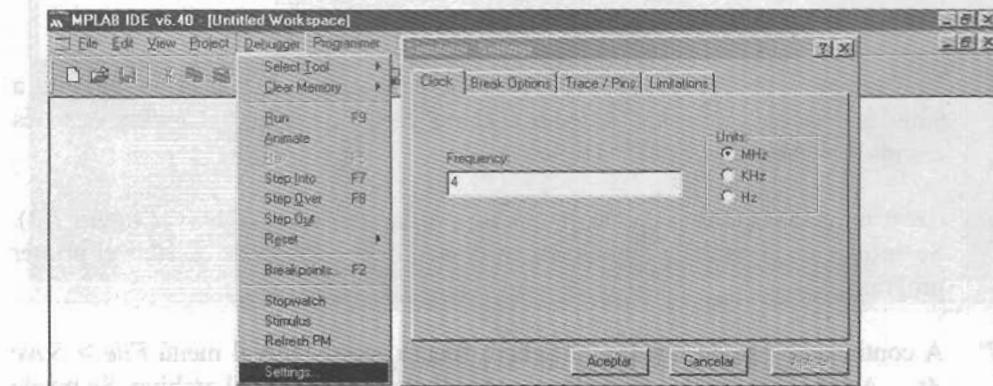


Figura 7-5 Selección de la frecuencia de simulación para el MPLAB SIM

otra que el lector

- 8º Para trabajar con más comodidad es conveniente visualizar el número de cada línea. Para ello seleccionar el menú *Edit > Properties*. Dentro de la ventana *Editor Options* y pestaña *Editor* se activan las opciones que se indica figura 7-6

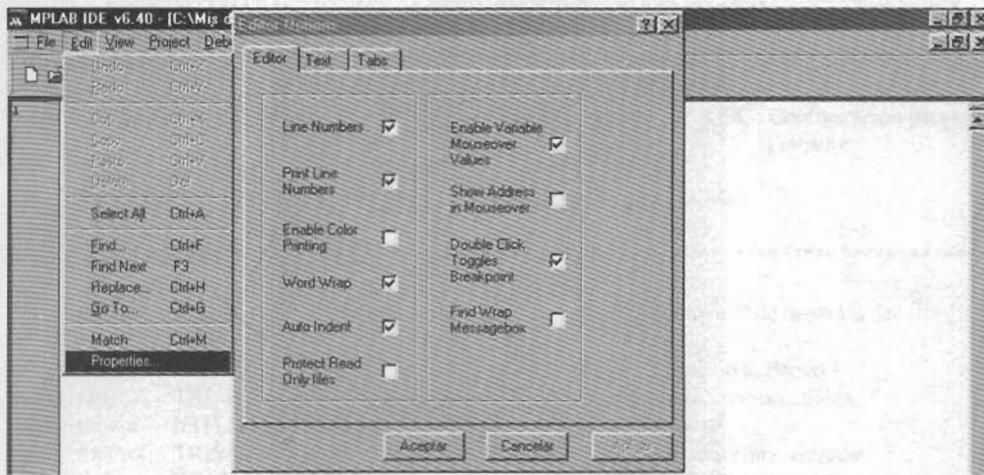


Figura 7-6 Propiedades de la pantalla de edición

- 9º Elegir el tipo de letra, activando el menú *Edit > Properties*. Dentro de la ventana *Editor Options* se activa la pestaña *Text* y se elige el tipo de letra seleccionado en la figura 7-7 o aquél otro tipo que al lector le resulte cómodo para su forma de trabajar.

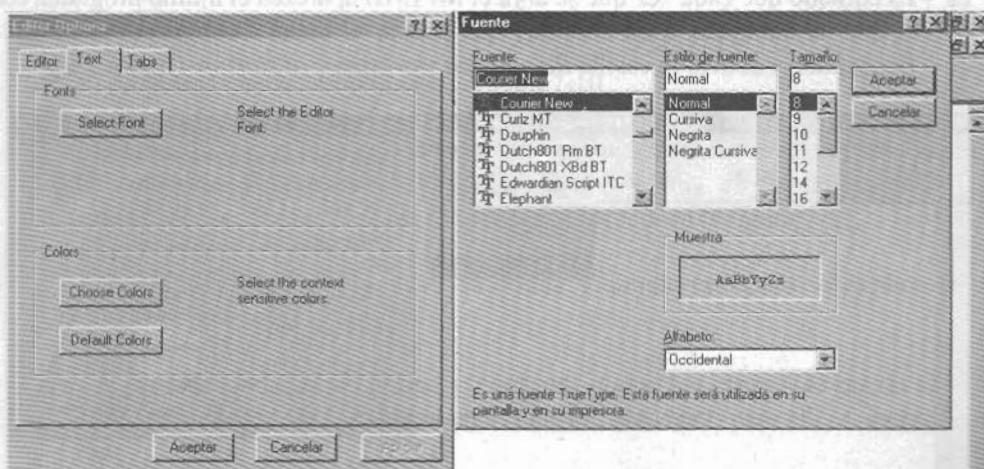


Figura 7-7 Elección del tipo de letra

- 10º A continuación hay que seleccionar la tabulación deseada, activando el menú *Edit > Properties*. Dentro de la ventana *Editor Options* se activa la pestaña *Tabs* y se elige el valor indicado en la figura 7-8 o aquél otro que el lector considere.

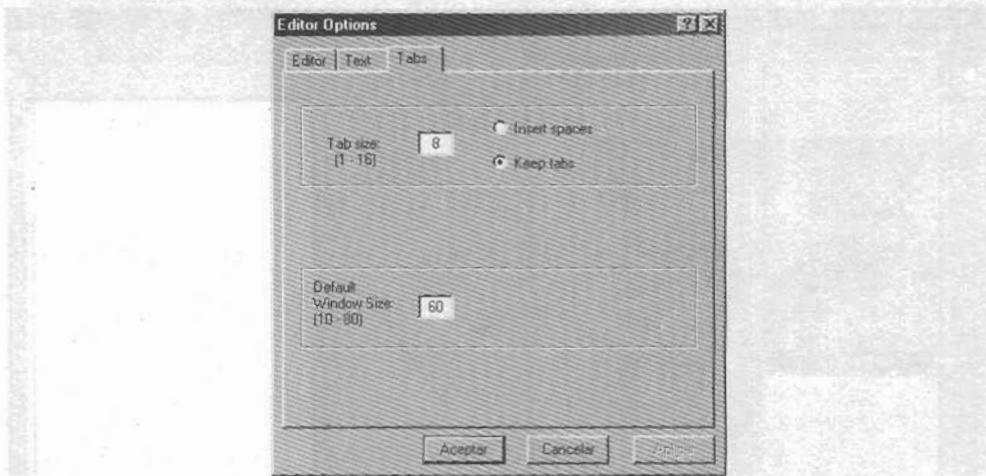


Figura 7-8 Elección de la tabulación

- 11º Es conveniente dejar los colores de los caracteres configurados por defecto. Para ello, activar el menú *Edit > Properties*. Dentro de la ventana *Editor Options* se activa la pestaña *Text* y el botón *Default Colors* (figura 7-7)

- 12º Es cómodo que cada vez que se abra el MPLAB aparezca el último programa con el que ha trabajado. Para ello, hay que activar el menú *Configure > Settings > Workspace* y activar la casilla *Reload last workspace at startup* (figura 7-9).

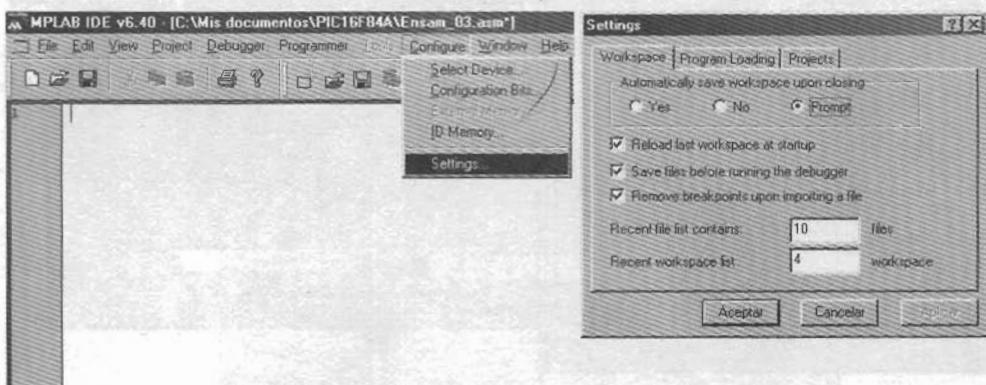


Figura 7-9 Configurar la recarga del último trabajo realizado al abrir el MPLAB

- 13º En la pantalla de edición se puede escribir el siguiente programa ejemplo:

```
***** Ensam_03.asm *****
;
; Por el Puerto B se obtiene el dato de las cinco líneas del Puerto A al que está conectado
; un array de interruptores. Por ejemplo, si por el Puerto A se introduce "___11001", por
; el Puerto B aparecerá "xxx11001" (el valor de las tres líneas superiores no importa).
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC ; Configuración para el
; grabador.
LIST P=16F84A ; Procesador.
INCLUDE <P16F84A.INC> ; Definición de los operandos utilizados.

;
; ZONA DE CÓDIGOS *****
ORG 0 ; El programa comienza en la dirección 0 de memoria de
; programa.
Inicio bsf STATUS,RP0 ; Pone a 1 el bit 5 del STATUS. Acceso al Banco 1.
clrf TRISB ; Las líneas del Puerto B se configuran como salidas.
movlw b'11111111'
movwf TRISA ; Las líneas del Puerto A se configuran como entradas.
bcf STATUS,RP0 ; Pone a 0 el bit 5 de STATUS. Acceso al Banco 0.
Principal
movf PORTA,W ; Lee el Puerto A.
movwf PORTB ; El contenido de W se visualiza por el Puerto B.
goto Principal ; Crea un bucle cerrado.
END ; Fin del programa.
```

Es importante recordar que “CONFIG” se inicia con dos subrayados (guiones bajos), no con uno, (este error es muy frecuente en los primeros programas).

- 14º El programa queda como ilustra la figura 7-10.
- 15º Mientras el programa es editado sin salvar el nombre del mismo aparece terminado en “\*\*” (asterisco). Una vez que el programa es grabado en disco duro este asterisco desaparece. Esta diferencia se aprecia en la figura 7-11.
- 16º A continuación el programa se ensambla y simula tal como se explica en los próximos apartados.
- 17º Una vez simulado el programa y corregido todos los errores se puede salir del MPLAB por el método habitual en Windows, activando para ello el menú File > Exit.



el MPLAB

mplo:

```

1 ;***** Ensam_03.asm *****
2 ;
3 ; Por el Puerto B se obtiene el dato de las cinco líneas del Puerto A al que está conectado
4 ; un array de interruptores. Por ejemplo, si por el Puerto A se introduce "---11001", por
5 ; el Puerto B aparecerá "xx11001" (el valor de las tres líneas superiores no importa).
6 ;
7 ; ZONA DE DATOS *****
8
9     _CONFIG _CP_OFF & _WDT_OFF & _PWRT_ON & _XT_OSC ; Configuración para el
10          ; grabador.
11     LIST P=16F84A ; Procesador.
12     INCLUDE <16F84A.INC> ; Definición de los operandos utilizados.
13
14 ; ZONA DE CÓDIGOS *****
15
16     ORG 0           ; El programa comienza en la dirección 0 de memoria de
17          ; programa.
18     Inicio bcf STATUS,RFO ; Pone a 1 el bit 5 del STATUS. Acceso al Banco 1.
19         clrf TRISB ; Las líneas del Puerto B se configuran como salidas.
20         movlw b'11111111'
21         movwf TRISA ; Las líneas del Puerto A se configuran como entradas.
22         bcf STATUS,RFO ; Pone a 0 el bit 5 de STATUS. Acceso al Banco 0.
23     Principal
24         movf PORTA,W ; Lee el Puerto A.
25         movwf PORTB ; El contenido de W se visualiza por el Puerto B.
26         goto Principal ; Crea un bucle cerrado.
27
28     END             ; Fin del programa.

```

MPLAB SIM | PIC16F84A | Ipc0 | IWD | Inovz do c | 0x200 | Ln 28, Col 52 | INS

Figura 7-10 Programa ejemplo completamente editado

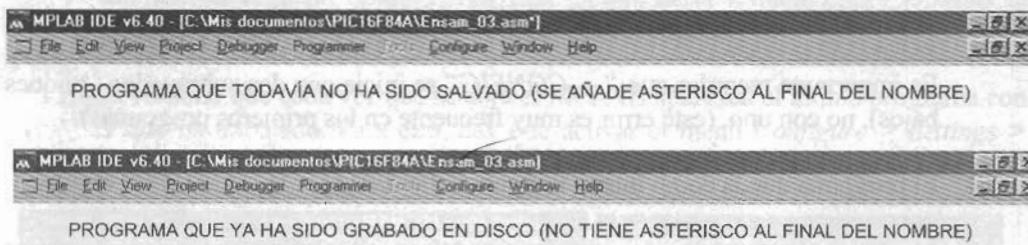


Figura 7-11 Identificación de un programa que todavía no ha sido salvado a disco

### 7.3 ENSAMBLADO DEL PROGRAMA

Una vez terminado de editar el programa hay que proceder a ensamblar el archivo fuente Ensam\_03.asm. Para ello, hay que seleccionar el menú *Project > Quickbuild Ensam\_03.asm*, o mejor abreviar con la combinación de teclas Alt+F10 (figura 7-12). En esta etapa se realiza en forma automática el ensamblado del archivo fuente y el traspaso de éste a la memoria de simulación.

*Figura 7-10 Configurar los interruptores de tristate mediante el puerto C*

*Figura 7-11 La pantalla de edición de código fuente muestra el nombre del archivo*

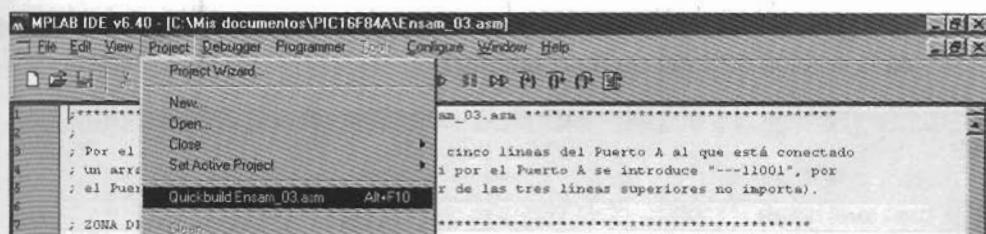


Figura 7-12 Ensamblar archivo fuente

Momentáneamente aparecerá una ventana indicando el proceso de ensamblado (figura 7-13).

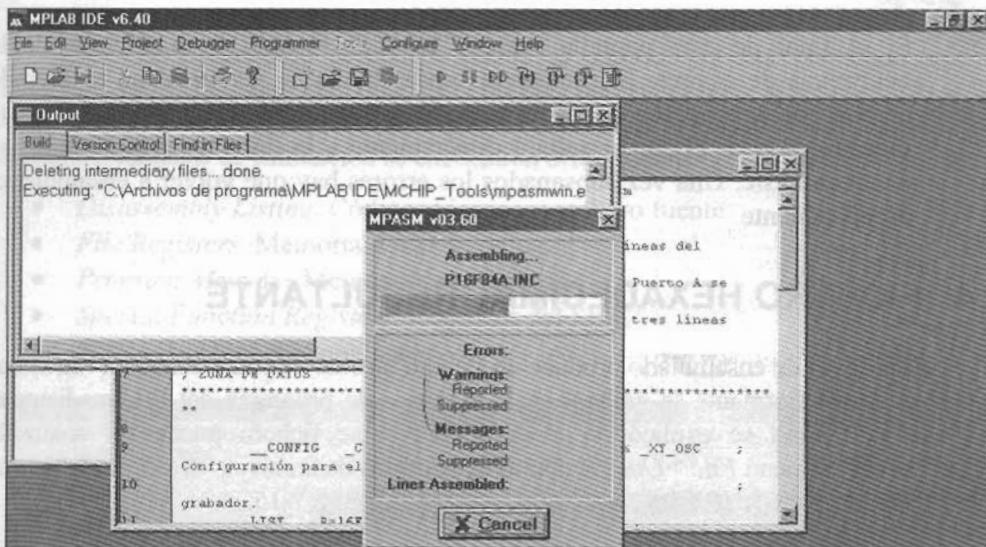


Figura 7-13 Proceso de ensamblado

Al finalizar el ensamblado, aparecerá una pantalla *MPLAB – [Output]* tal como muestra la figura 7-14, en la que indica la ocurrencia de uno de estos dos casos:

- Si al final de esta pantalla indica “*BUILD SUCCEEDED*” se confirma que el ensamblado se ha producido con éxito. Por tanto, ya se está en condiciones de pasar a la simulación. En esta pantalla pueden aparecer algunos mensajes de aviso “*Message*”, que llaman la atención sobre situaciones a tener en cuenta y que podrían ocasionar un error en el programa pero que no impiden el correcto ensamblado. Por ejemplo, en la figura 7-14, los mensajes llaman la atención sobre las líneas 19 y 21 que utilizan los registros TRISA y TRISB que no trabajan en el Banco 0 y hay que asegurarse que se ha programado correctamente, (para evitar este mensaje en particular se puede utilizar

"PORTA" y "PORT B" en lugar de "TRISA" y "TRISB" tal como se explicó en la sección 6.12 del capítulo anterior).

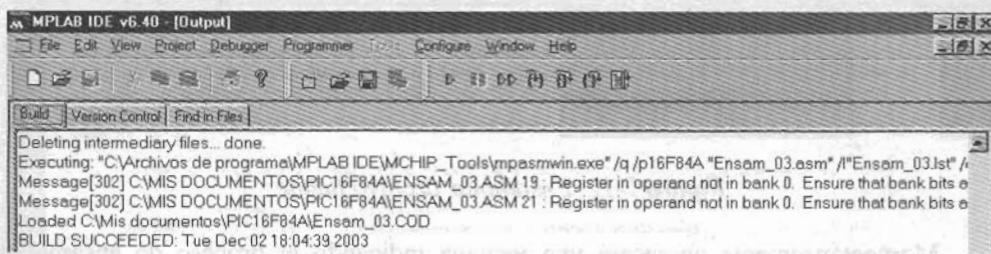


Figura 7-14 Pantalla final de proceso de ensamblado con éxito

- Si al final de esta pantalla indica "BUILD FAILED" se advierte de la ocurrencia de errores. El proceso de ensamblado ha generado un archivo de errores con descripción de los mismos. Si se hace doble clic sobre la línea que muestra el error el cursor saltará directamente a la línea de código donde se encuentra éste. Una vez subsanados los errores hay que volver a ensamblar el archivo fuente.

## 7.4 FICHERO HEXADECIMAL RESULTANTE

El proceso de ensamblado produce un fichero ejecutable con extensión (\*.hex) que será el que posteriormente se grabará en la memoria de programa del PIC mediante el grabador, tal como se explicó en el capítulo 3. Ese fichero puede ser analizado seleccionando el menú *File > Open* y dentro de los tipos de archivos *All Files (\*.\*)* se ha de elegir el *Ensam\_03.HEX*, tal como se muestra en la figura 7-15.

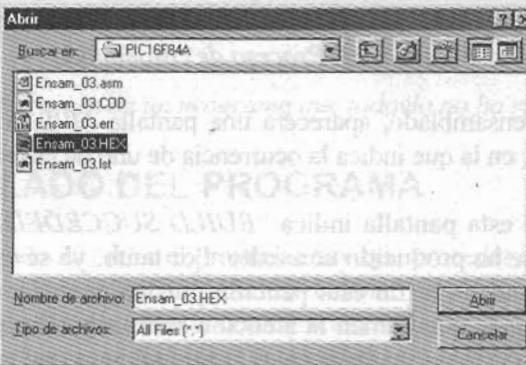


Figura 7-15 Abrir el archivo resultado del ensamblado en código máquina \*.HEX

Se observa que este archivo Ensam\_03.hex únicamente contiene números hexadecimales, que es la forma de representar los ceros y unos binarios de la información

que se gra  
ilustra la fi

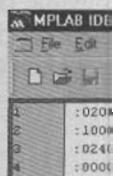
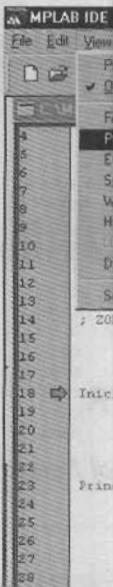


Figura 7-

## 7.5 VE

Una  
simulación  
ventanas q  
principales.

- D
- F
- P
- S
- Sp
- Wa



como se explicó

que se grabará posteriormente en la memoria de programa del microcontrolador, tal como ilustra la figura 7-16.

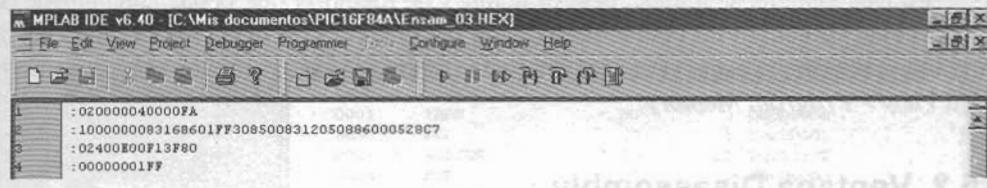


Figura 7-16 Contenido del archivo resultado del ensamblado en código máquina \*.Hex

## 7.5 VENTANAS DE VISUALIZACIÓN

Una vez ensamblado el código fuente el entorno ya está preparado para la simulación del programa. Para que este trabajo sea más eficaz conviene activar las ventanas que indican el estado de todas las memorias y registros del sistema. Las principales ventanas de simulación se encuentran dentro del menú *View* y son:

- *Disassembly Listing*. Código máquina y archivo fuente.
- *File Registers*. Memoria RAM de datos.
- *Program Memory*. Memoria de programa.
- *Special Function Registers*. Registros del SFR.
- *Watch*. Ventana personalizada.

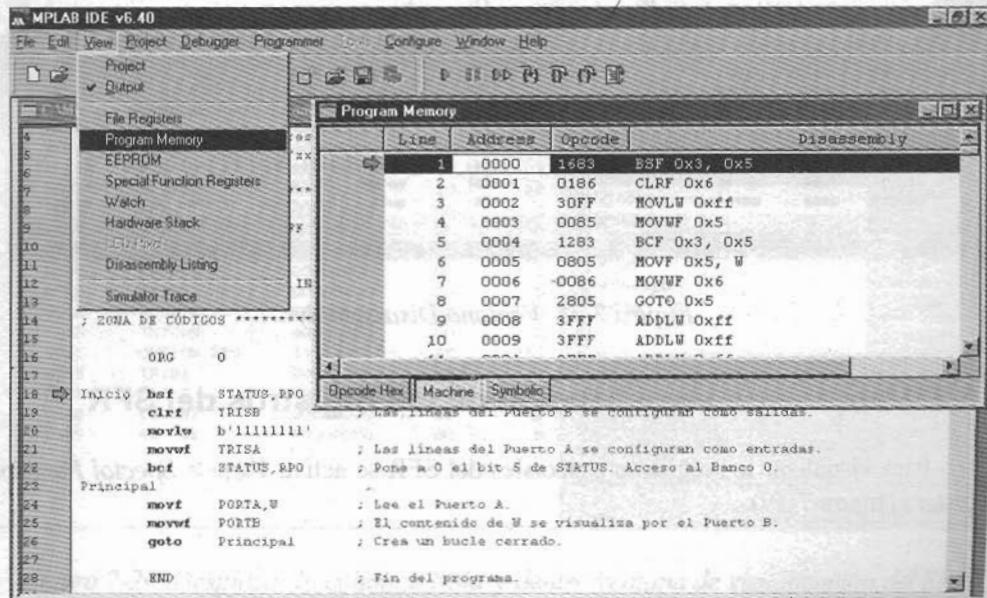


Figura 7-17 Ventana de visualización de la memoria de programa

### 7.5.1 Ventana de visualización de la memoria de programa

En esta ventana se aprecian las posiciones de memoria que ocupa cada una de las instrucciones, el código de operación de cada instrucción y la dirección de memoria de programa que se le ha asignado a cada etiqueta (figura 7-17). Se entra en ella activando el menú *View > Program Memory*.

### 7.5.2 Ventana Disassembly

Esta ventana es similar a la anterior, en la que además se ha incluido el archivo fuente (figura 7-18). Se entra en ella activando el menú *View > Disassembly Listing*.

```

MPLAB IDE v6.40 - [Disassembly Listing]
File Edit View Project Debugger Programmer Configure Window Help
Project Output
File Registers Program Memory EEPROM Special Function Registers Watch Hardware Stack Inspector Disassembly Listing Simulator Trace
784A\ENSAM_03.ASM
1: ;***** Ensam_03.ASM *****
2: ;
3: ; Por el Puerto B se obtiene el dato de las cinco
4: ; un array de interruptores. Por ejemplo, si por
5: ; el Puerto B aparecerá "xxxx1001" (el valor de
6: ;
7: ; ZONA DE DATOS *****
8: ;
9: _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON &
10: ; grabador.
11: LIST P=16F84A ; Procesador
12: INCLUDE <16F84A.INC> ; Definición de los
13: ;
14: ; ZONA DE CÓDIGOS *****
15: ;
16: ORG 0 ; El programa comienza en la
17: ; programa.
18: Inicio bcf STATUS,RPO ; Pone a 1 el bit 5 del
19: clrf TRISB ; Las líneas del Puerto
20: movwf b'11111111'
21: movwf TRISA ; Las líneas del Puerto
22: bcf STATUS,RPO ; Pone a 0 el bit 5 de STATUS
23: Principal
24: movf PORTA,W ; Lee el Puerto A
25: movwf PORTB ; El contenido de W se vuelve
26: goto Principal ; Crea un bucle cerrado.
000000 1683 BSE 0x3, 0x5
000001 186 CLRF 0x6
000002 30FF MOVLW 0xffff
000003 085 MOVWF 0x5
000004 1283 BCF 0x3, 0x5
000005 805 MOVE 0x5, W
000006 096 MOVWF 0x6
000007 2805 GOTO 0x5

```

Figura 7-18 Ventana Disassembly

### 7.5.3 Ventana de visualización de los registros del SFR

Para visualizar los registros especiales del SFR se activa *View > Special Function Registers* (figura 7-19).

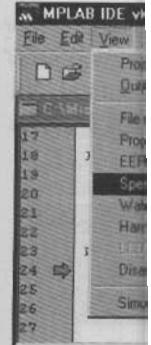


Figura 7-19

## • programa

upa cada una de las  
cción de memoria de  
en ella activando el

a incluido el archivo  
embly Listing.

```

----- RAM -----
ene el dato de las cin
es. Por ejemplo, si po
xx11001" (el valor de
-----  

_WDT_OFF & _PWRTE_ON &
; grabador.
Procesador.
; Definición de los
-----  

rograma comienza en la
Pone a 1 el bit 5 del
Las líneas del Puerto
Las líneas del Puerto
a 0 el bit 5 de STATUS
Lee el Puerto A.
El contenido de W se v.
Crea un bucle cerrado.
-----  

0x899

```

## del SFR

> Special Function

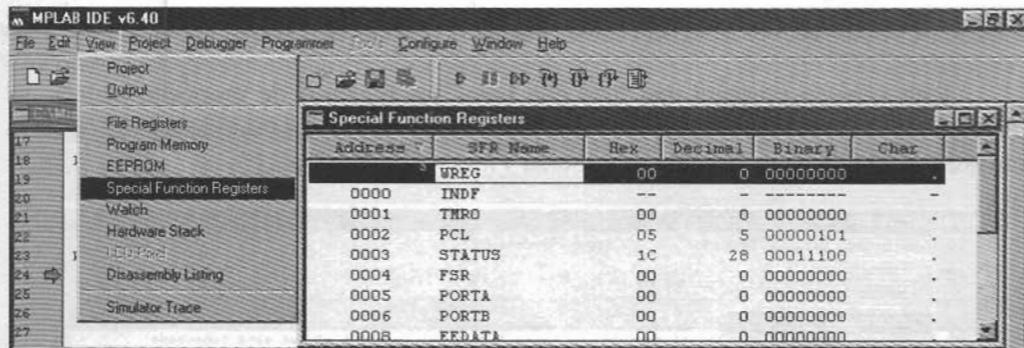


Figura 7-19 Ventana de visualización de los registros especiales

Para modificar manualmente uno de estos registros hay que hacer doble clic en la fila del registro correspondiente sobre alguna de las columnas *Hex*, *Decimal*, *Binary*, o *Char* y modificarlo. Esto no es válido para los puertos que actúen como entrada, en cuyo caso hay que utilizar, dentro del menú, *Debugger*, la opción *Stimulus* tal como se explicará más adelante.

En esta ventana suele ser interesante situar la columna *Binary* a continuación de la columna *SFR Name*. Para ello, pulsar el botón derecho del ratón y elegir la opción *Properties*, saldrá una ventana como la que se muestra en la 7-20. Señalando la casilla *Binary* y pulsando sobre el botón *Move Up*, la columna *Binary* se desplazará hasta situarse en la posición deseada tal como se explica en la figura.

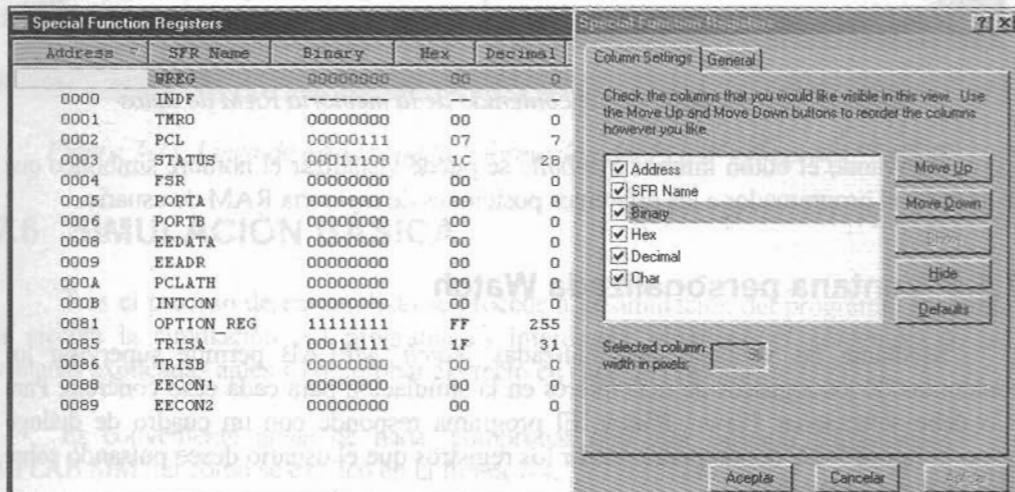


Figura 7-20 Desplazar la columna Binary dentro ventana de visualización del SFR

### 7.5.4 Ventana de contenido de la memoria RAM

Esta ventana presenta una lista con todos los registros de propósito general del microcontrolador simulado (figura 7-21). Para visualizar la ventana de contenido de la memoria RAM de datos hay que seleccionar *View > File Registers*.

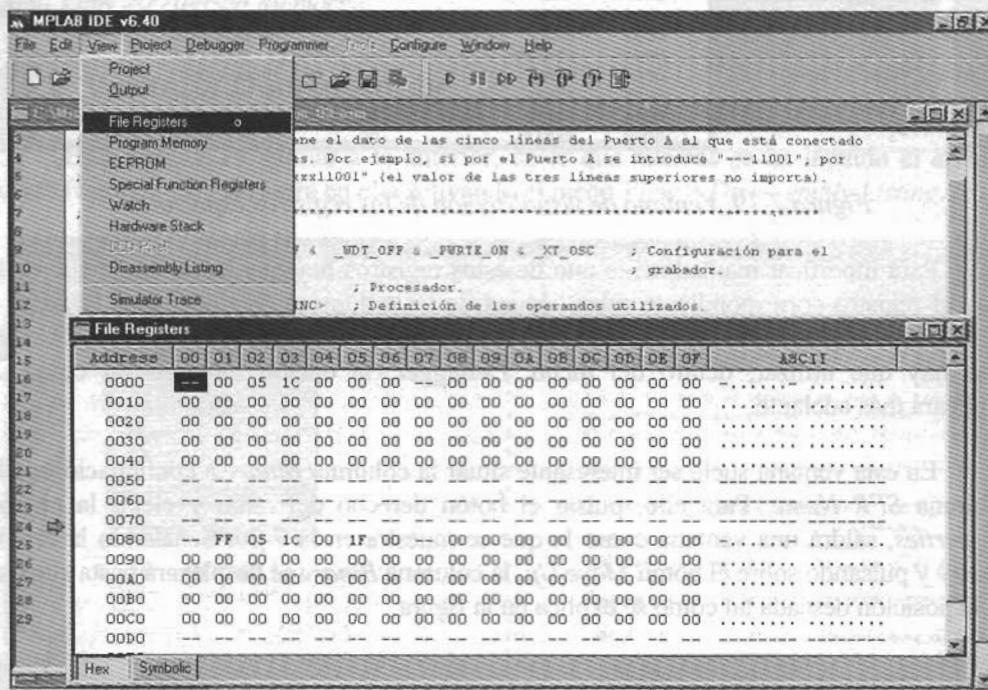


Figura 7-21 Ventana con el contenido de la memoria RAM de datos

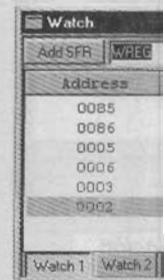
Activando el botón inferior *Symbolic* se puede visualizar el nombre simbólico que le ha dado el programador a las diferentes posiciones de memoria RAM de usuario.

### 7.5.5 Ventana personalizada Watch

Mediante las ventanas personalizadas *Watch*, MPLAB permite supervisar los contenidos de los registros de más interés en la simulación para cada caso concreto. Para ello debe seleccionar *View > Watch*. El programa responde con un cuadro de diálogo como la figura 7-22, donde podrá añadir los registros que el usuario desee pulsando sobre *Add SFR* o *Add Symbol*.

La configuración de esta ventana se puede salvar para utilizarla en posteriores ocasiones. Para ello, estando situado sobre esta ventana pulsar el botón derecho del ratón

y seleccionar *Windows*.



### 7.5.6 Líneas

Aunque proporciona información en la parte inferior de la simulación.

Es específica del registro de tipo en mayúsculas indicada en la figura 7-23.

Figura 7-22

### 7.6 SIMU

Tras el programa se ejecuta la simulación. Las ventanas explicadas

Es conveniente usar MPLAB SIM, ya que

Los cinco menús *Debugger*

y seleccionar *Output to File* y salvar el fichero de la forma ya conocida en el entorno Windows.

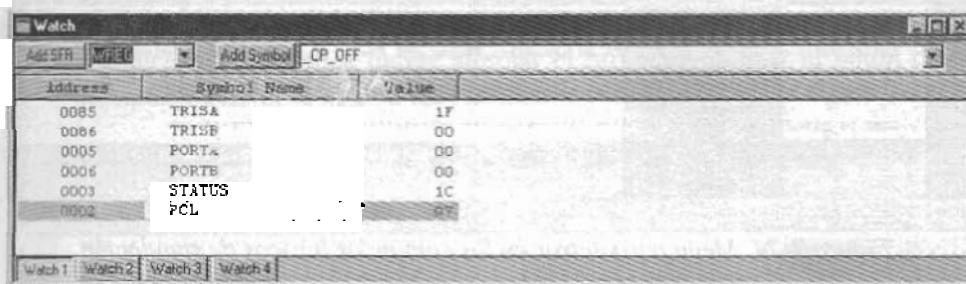


Figura 7-22 Ventana personalizada Watch

### 7.5.6 Línea de estado

Aunque no es una ventana de visualización propiamente dicha, la **Línea de estado** proporciona información muy útil sobre la situación actual del microcontrolador. Se ubica en la parte inferior de la pantalla y ofrece información en todo momento del estado de la simulación.

Es especialmente útil la información sobre el contenido del contador de programa, del registro de trabajo W. También indica el valor de las flags de STATUS. Si la letra está en mayúsculas indica que ese flag vale "1" y, si es minúsculas vale "0". Así en el ejemplo de la figura 7-13, Z=0, DC=1 y C=1.



Figura 7-23 Línea de estado con la información del PC, W y flags del STATUS

## 7.6 SIMULACIÓN BÁSICA

Tras el proceso de ensamblado se procede a la simulación del programa. Mientras se ejecuta la simulación del programa es interesante visualizar el contenido de las ventanas explicadas antes y comprobar el efecto en cada una de ellas.

Es conveniente antes de nada, comprobar que está cargado correctamente el MPLAB SIM, tal como se explicó en la figura 7-4.

Los cinco comandos más importantes para la simulación se localizan dentro del menú *Debugger* y se muestran en la figura 7-24.

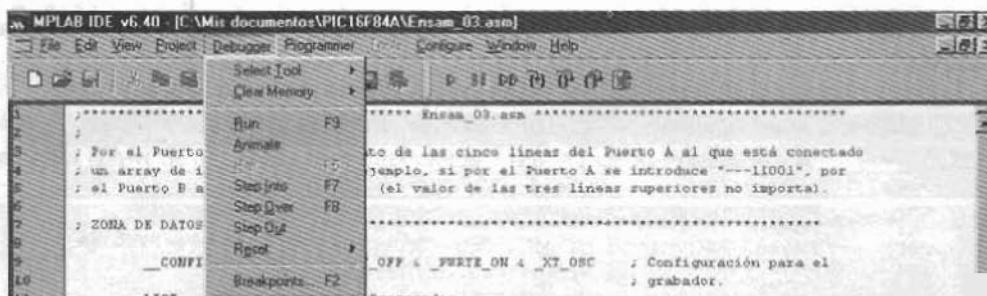


Figura 7-24 Menú para entrar en los comandos básicos de simulación

- **Run.** Modo de ejecución continua. Ejecuta el programa constantemente. Las ventanas abiertas en el paso anterior no se actualizan hasta que no se produce una parada. Es la forma más rápida de simular el programa, pero no se "ve" ni como evoluciona la memoria ni los distintos registros. En este modo se entra seleccionando *Debugger > Run* o pulsando la tecla F9, también al pulsar sobre el ícono correspondiente de la barra de herramientas (flecha azul).
- **Animate** (o teclas **ctrl+F9**). Modo de ejecución animada. Ejecuta el programa de forma continua pero actualizando todas las ventanas cada vez que se ejecuta una instrucción. Es más lento que el modo "Run" pero permite ver como van cambiando los registros. Tal vez sea el modo de ejecución más útil y recomendable. Se entra en este modo seleccionando *Debugger > Animate*, o también al pulsar sobre el ícono correspondiente de la barra de herramientas (doble flecha azul).
- **Halt.** Paro. Para la ejecución del programa y actualiza todas las ventanas. Se consigue seleccionando *Debugger > Run* o pulsando la tecla **E5**. También se entra en este modo al activar el ícono correspondiente de la barra de herramientas (dos barras verticales azules).
- **Step Into.** Ejecución paso a paso. Ejecuta una sola instrucción del programa cada vez actualizando los valores de las ventanas. Es la forma más lenta de simulación pero se comprueba fácilmente como van evolucionando todos los registros y memorias, siendo muy fácil detectar los posibles errores. En este modo se entra seleccionando *Debugger > Step Into* o pulsando la tecla F7. También pulsando sobre el ícono correspondiente de la barra de herramientas.
- **Reset.** Equivale a un reset por activación del pin **MCLR**. En este modo se entra seleccionando *Debugger > Reset* o pulsando la tecla F6. También si se pulsa sobre el ícono correspondiente de la barra de herramientas.

## 7.7 SIMULACIÓN

Un programa de simulación es una herramienta que nos permite analizar el comportamiento de un sistema sin tener que construirlo físicamente. Una simulación es la representación matemática de un sistema real o hipotético. La simulación es una técnica de modelado y análisis que permite prever el comportamiento futuro de un sistema basándose en su modelo matemático. Los resultados de la simulación se presentan en forma de gráficos, tablas y otros medios que permiten visualizar y comprender el funcionamiento del sistema.



Para comenzar la simulación, se debe abrir el archivo de proyecto y seleccionar la pestaña "Simulación". Una vez allí, se deben configurar los parámetros de simulación, como el tiempo de simulación, las condiciones iniciales y las entradas. Una vez configurados, se puede iniciar la simulación y observar el comportamiento de los sistemas en el tiempo.

Otro modo de simulación muy interesante es:

- **Run to Cursor.** Ejecución hasta la posición actual del cursor. Para entrar en este modo de simulación, el cursor debe situarse en la línea donde está la instrucción hasta donde quiere simular el programa, pulsar el botón derecho del ratón y activar la opción *Run to Cursor* tal como indica la figura 7-25.

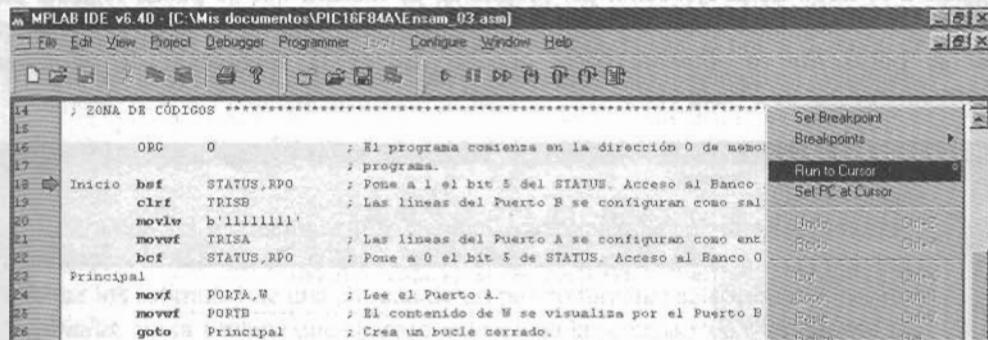


Figura 7-25 Modo de simulación "Run to Cursor"

## 7.7 SIMULACIÓN MEDIANTE BREAKPOINTS Y TRAZA

Un **punto de ruptura** o *BreakPoint* es un punto o instrucción donde la ejecución del programa se detiene, por ello también se le suele llamar **punto de paro**, permitiendo el análisis del estado del microcontrolador. Para continuar la ejecución del programa hay que volver a pulsar sobre *Run* o *Animate*.

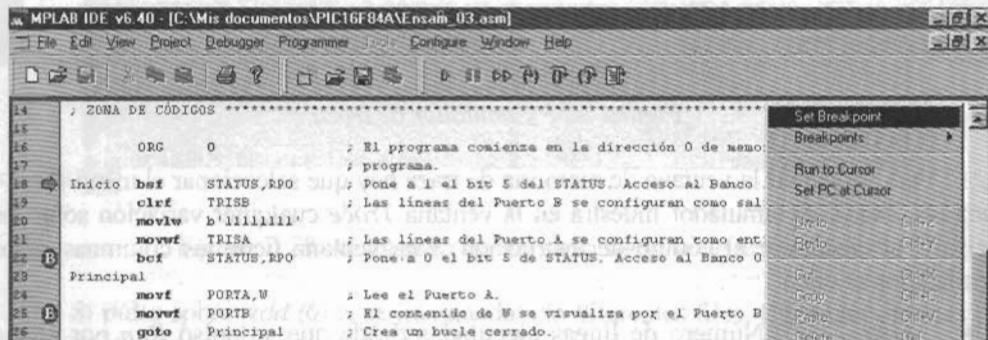


Figura 7-26 Situar un Breakpoint

Para situar un *Breakpoint* sobre una línea señalada por el cursor se pulsa el botón derecho del ratón, de manera que aparece el menú desplegable, como en la figura 7-26. Selecciona *Set Breakpoint* y aparecerá sobre el programa una "B" en rojo en la posición donde ha situado el punto de paro. Otra forma de situar o eliminar un *Breakpoint* es

realizando una doble pulsación con el ratón sobre el número de línea donde se quiere situar el punto de parada.

La ventana de **memoria de traza** es una herramienta que ayuda a simular los programas (figura 7-27). El *Simulate Trace* toma “una instantánea” de la ejecución del programa. En el simulador el buffer de traza o memoria de traza es útil para visualizar un registro a lo largo de la ejecución del programa, de manera que se puede registrar por dónde pasa el programa y después analizarlo. El simulador toma datos desde la última vez que se pulsó *Run* o *Animate* hasta que se detiene la simulación del programa (normalmente por un *Breakpoint*).

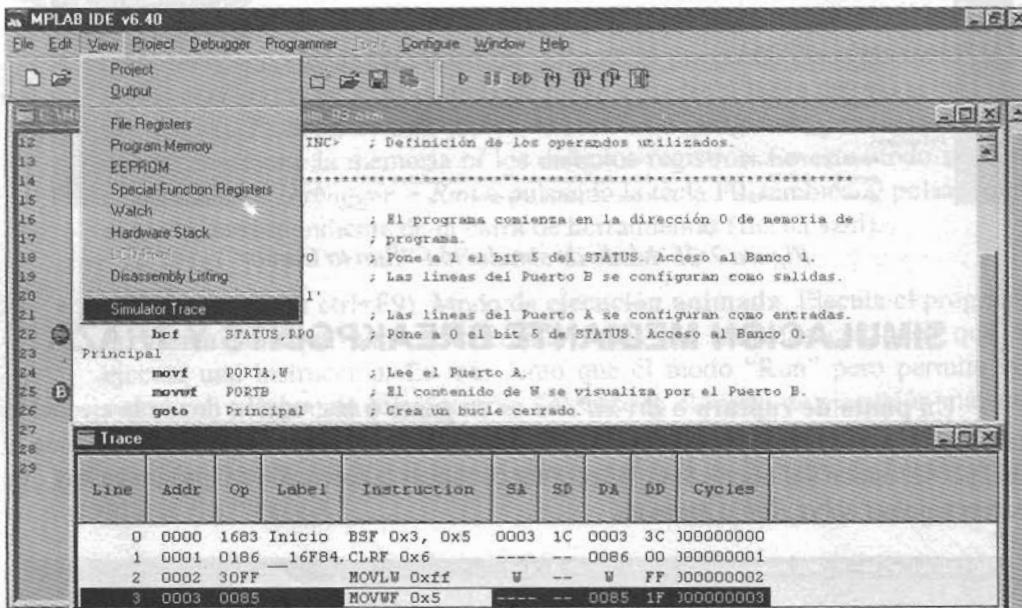


Figura 7-27 Simulador de traza

Para visualizar la ventana de memoria de traza hay que seleccionar el menú *View > Simulate Trace*. El simulador muestra en la ventana *Trace* cualquier variación sobre los registros al ejecutarse el código de instrucción. Esta ventana tiene las columnas cuyos significados se citan:

- *Line* Número de líneas ejecutadas desde que se pulsó *Run* por última vez.
- *Addr* Dirección de la memoria de programa donde se encuentra la instrucción.
- *Op* Código de operación numérico de la instrucción.
- *Label* Etiqueta de la instrucción si la tuviese.
- *Instruction* Instrucción ejecutada.

- *SAT*
- *SDA*
- *DA*
- *DD*
- *Cy*

El co análisis. Pa seleccionar:

## 7.8 SIM

Una los valores cambiar los > *Stimulus*.



Si pu corresponde sobre la casi de cambio q

- *Hig*
- *Low*
- *Tog*
- *Puls*

yuda a simular los de la ejecución del l para visualizar un puede registrar por desde la última vez ión del programa

- *SA* Dirección numérica del registro fuente.
  - *SD* Dato del registro fuente.
  - *DA* Dirección numérica del registro destino.
  - *DD* Dato del registro destino.
  - *Cycles* Ciclos máquina transcurridos.

El contenido de la memoria de traza se puede salvar a un fichero para un posterior análisis. Para ello, estando situado sobre esta ventana pulsar el botón derecho del ratón y seleccionar *Output to File* salvando por el procedimiento conocido en Windows.

## 7.8 SIMULACIÓN DE ENTRADAS

Una de las operaciones más habituales de cualquier simulación consiste en variar los valores de las líneas de entrada. A ésto se denomina “estimular” la entrada. Para cambiar los estímulos de una entrada de un puerto hay que seleccionar el menú *Debugger* > *Stimulus*. En la ventana que aparece, selecciona la pestaña *Pin Stimulus* (figura 7-28).

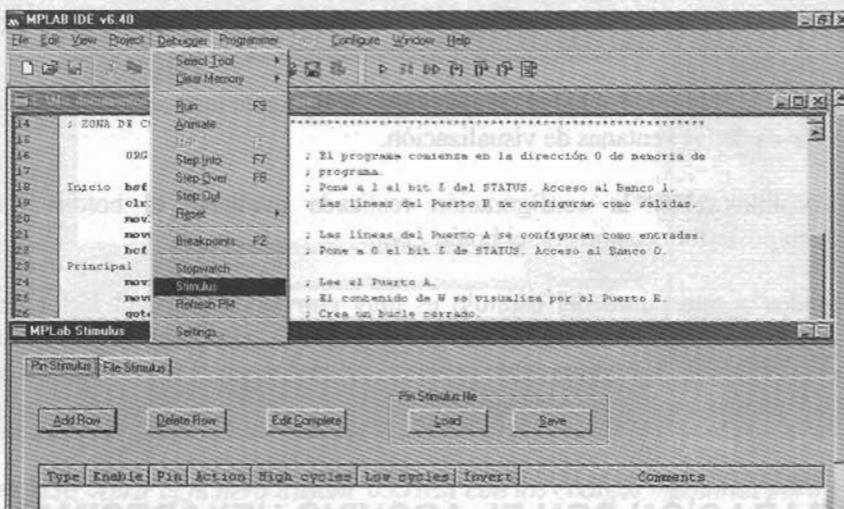


Figura 7-28 Menú para entrar en la ventana de estímulos

Si pulsa sobre *Add Row*, se irán añadiendo diferentes filas. Cada una de estas filas corresponde a un estímulo sobre una línea de entrada. La forma de editarlos es pulsar sobre la casilla correspondiente y seleccionar la patilla a la que se quiere vincular y el tipo de cambio que se desea realizar con ese pin para cada pulsación:

- *High*, poner la entrada a “1”.
  - *Low*, poner la entrada a “0”.
  - *Toggle*, cambiar de valor cada vez que se pulse. Ésta es la más habitual.
  - *Pulse*, cambia el estado del pin y retorna de nuevo a su valor actual.

La figura 7-29 muestra como se ha configurado para las cinco líneas del puerto A, como entrada y en modo *Toggle*.

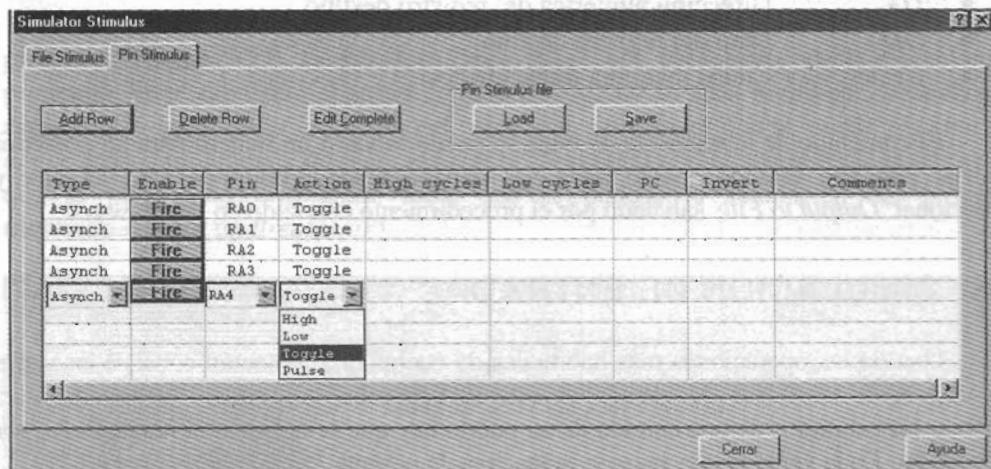


Figura 7-29 Configurar los estímulos para el Puerto A como entrada y modo Toggle

Tras pulsar el botón *fire* habrá de ejecutarse la siguiente instrucción antes de ver los cambios a través de las ventanas de visualización.

Es posible salvar la configuración realizada mediante el botón *Save* para recuperarla en posterior ocasión con el botón *Load*.

Llegados a este punto del capítulo es muy conveniente que el lector simule el funcionamiento del programa ensamblado *Ensam\_03.asm* con diferentes combinaciones de las líneas del Puerto A y compruebe los valores de salida en el Puerto B y los cambios en todas las ventanas explicadas anteriormente.

## 7.9 GRABACIÓN CON EL ARCHIVO HEXADECIMAL

Una vez simulado el programa y comprobado que funciona correctamente, es tiempo de grabar físicamente dicho programa en el PIC16F84A. Con el ensamblado del archivo fuente se ha generado un archivo ejecutable con los códigos máquina a grabar en el PIC16F84A. Este archivo tiene extensión \*.hex y se graba en el PIC16F84A con ayuda de un grabador y el software asociado, tal como se ha estudiado en el capítulo 3. En este caso se utilizará un programador compatible JDM y el software IC-Prog. Para ello se deben seguir los siguientes pasos:

Figura

4º C

T

en

A

•

•

•

•

•

•

•

•

•

•

•

•

•

•

•

•

•

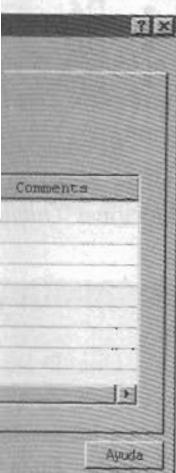
•

•

•

•

neas del puerto A,



y modo Toggle

ón antes de ver los

boton Save para

el lector simule el  
es combinaciones  
o B y los cambios

CIMAL

correctamente, es  
el ensamblado del  
quina a grabar en  
6F84A con ayuda  
apítulo 3. En este  
Prog. Para ello se

- 1º Conectar el grabador al ordenador. Insertar el PIC16F84A en el zócalo correspondiente, teniendo en cuenta la orientación correcta del chip ayudándonos por la muesca de la cápsula.
- 2º Abrir el programa IC-Prog y comprobar que está correctamente configurado.
- 3º Abrir el archivo con extensión \*.hex que contiene los datos a programar en el PIC16F84A. Para ello, seleccionar el menú Archivo > Abrir archivo y, una vez dentro de la carpeta apropiada, elegir el fichero a grabar, en este ejemplo el *Ensam\_03.HEX* (figura 7-30).

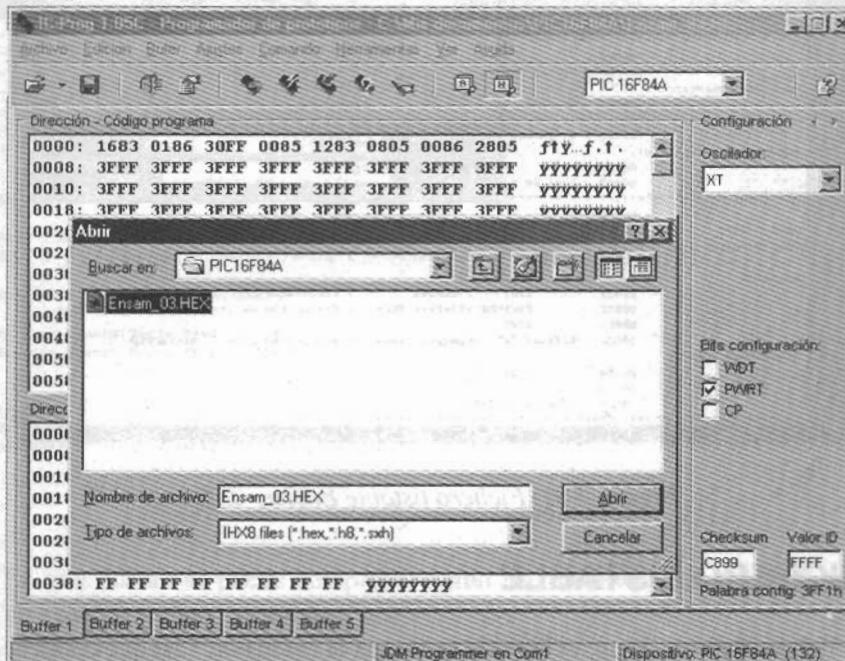


Figura 7-30 Abrir el fichero *Ensam\_03.HEX* con los códigos máquinas para grabar

- 4º Comprobar que los datos se han cargado en el área de *Código de programa*. También debe comprobarse que los *Bits configuración* se ajustan a lo pretendido en el programa con la línea: “*\_CONFIG = CP\_OFF & WDT\_OFF & PWRTE\_ON & XT\_OSC*”. En este caso:
  - *CP* en OFF: Protección de código inactiva.
  - *WDT* en OFF: Watchdog inactivo.
  - *PWRT* en On: Power-up Timer activado.
  - *Oscilador*: XT (a cristal de cuarzo).
- 5º Proceder a la grabación física del chip, tal como se explicó en el capítulo 3.

Una vez grabado el PIC16F84A se debe extraer del grabador y comprobar su correcto funcionamiento dentro del circuito correspondiente. En este caso el programa Ensam\_03.asm lo que hace es visualizar por los LEDs conectados al Puerto B el dato leído de las cinco líneas del Puerto A al que está conectado un array de interruptores. Esto se puede comprobar con el circuito de la figura 1-2.

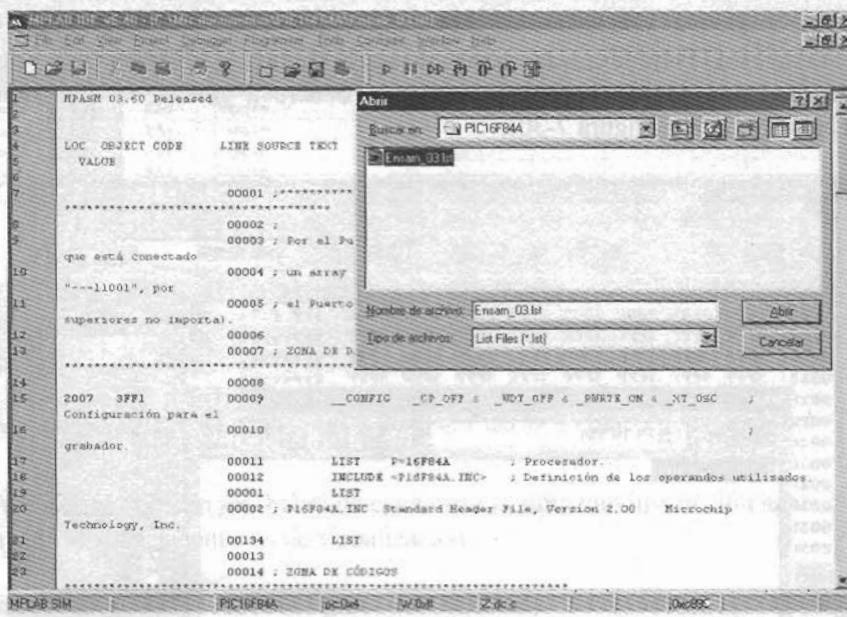


Figura 7-31 Fichero listable Ensam\_03.lst

## 7-10 FICHERO LISTABLE

El fichero **listable** es un archivo de texto con la extensión \*.lst que contiene toda la información del programa: código fuente, códigos máquina, direcciones de cada instrucción, errores producidos, etc. Este fichero puede ser analizado seleccionando el menú *File >Open* y dentro de los tipos de archivos *List Files (\*.lst)* se ha de elegir el *Ensam\_03.lst* (figura 7-31).

Este archivo listable contiene una copia del fichero del código fuente pero con tres columnas añadidas a la izquierda:

- Columna *LOC OBJECT VALUE*, que representa las posiciones de la memoria de programa donde se ubican las instrucciones.
  - Columna *CODE*, que presenta el código hexadecimal de cada una de las instrucciones.
  - La columna *LINE SOURCE TEXT*, que indica el número de línea del código fuente.

y comprobar su uso el programa Puerto B el dato interruptores. Esto

Finalmente, en las siguientes columnas aparece el código fuente tal y como se editó.

Al final del listado aparece una información adicional sobre las etiquetas utilizadas y el valor asociado a ellas. También aparece un mapa representativo del espacio de memoria utilizado. Por último informa sobre el número de errores, avisos y mensajes ocurridos (figura 7-32).

```

MPLAB IDE v6.40 - [C:\Mis documentos\PIC16F84A\Ensam_03.lst]
File Edit View Project Debugger Programmer Configure Window Help
105  _PU2TE_OFF           00003FFF
106  _PU2TE_ON            00003FFF
107  _DC_OSC               00003FFF
108  _WDT_OFF              00003FFB
109  _WDT_ON               00003FFF
110  _XT_OSC               00003FFD
111  _16F84A                00000001
112
113
114 MEMORY USAGE MAP ('X' = Used, '-' = Unused)
115
116 0000 : XXXXXXXX-
117 2000 : -----X-
118
119 All other memory blocks unused.
120
121 Program Memory Words Used: 8
122 Program Memory Words Free: 1016
123
124
125 Errors : 0
126 Warnings : 0 reported, 0 suppressed
127 Messages : 2 reported, 0 suppressed

```

Figura 7-32 Final del fichero listable Ensam\_03.lst

En este fichero se puede comprobar como las directivas y comentarios del código fuente no generan código máquina.

## 7.11 PRÁCTICAS DE LABORATORIO

El circuito para comprobar el correcto funcionamiento de los programas será el entrenador básico para aprendizaje del PIC16F84A, representado en el esquema de la figura 1-2, que ya fue montado en el primer capítulo.

Respetando el procedimiento descrito en el capítulo diseñar, ensamblar, simular y grabar el microcontrolador y comprobar los siguientes programas. Visualizar los ficheros resultantes \*.hex y \*.lst de cada uno de los programas. Las soluciones se ofrecen en el CD-ROM que acompaña a este libro.

**Ensam\_01.asm:** Por los LEDs conectados al puerto B se visualiza el valor de una constante, por ejemplo el número binario b'01010101'.

**Ensam\_02.asm:** Los LEDs conectados al nibble bajo del Puerto B se apagarán y los del nibble alto se encenderán.

**Ensam\_03.asm:** Por el Puerto B se obtiene el dato de las cinco líneas del Puerto A al que está conectado un array de interruptores. Por ejemplo, si por el Puerto A se introduce “---11001”, por el Puerto B aparecerá “xxx11001” (no importa el valor de los tres bits más altos del Puerto B, ver esquema de la figura 1-2).

En la entrada:

- Un interruptor cerrado representa un “0” lógico.
- Un interruptor abierto representa un “1” lógico.

En la salida:

- Un LED apagado representa un “0” lógico.
- Un LED encendido representa un “1” lógico.

el valor de una

se apagarán y

as del Puerto A  
el Puerto A se  
el valor de los

## 8.2 INSTRUCCIONES DE RESTA

### CAPÍTULO 8

# PROGRAMACIÓN ELEMENTAL

En este capítulo se comienza a realizar los primeros programas elementales con las instrucciones aritméticas y lógicas, iniciando al lector en la programación del PIC16F84.

## 8.1 INSTRUCCIONES DE SUMA

El PIC16F84 puede realizar las siguientes operaciones aritméticas: sumar, restar, incrementar y decrementar un registro.

La suma se realiza en aritmética binaria pura sin signo y afecta a los flags del STATUS de la siguiente forma:

- Al flag **Z** (*Zero*). El bit Z se pone en “1” si el resultado de la operación es cero (b’00000000’) y se pone Z en “0” si el resultado tiene cualquier otro valor.
- Al flag **C** (*Carry*). Si hay un acarreo del bit 7, es decir, si el resultado es mayor que b’11111111’ (255 en decimal) el bit C (*Carry*) se activa a “1”, en caso contrario resulta C = 0.
- Al flag **DC** (*Digit Carry*). Si hay un acarreo del bit 3 al 4, es decir que la suma de las dos mitades (nibbles) menos significativa (bits 0 a 3) resulta mayor que 15, el bit DC se pone a “1”, en caso contrario se pone a “0”.

Ejemplo 1:

163 (decimal)	1010 0011 (binario)	A3 (hex.)			
+ 79 (decimal)	+ 0100 1111 (binario)	4F (hex.)	C	DC	Z
242 (decimal)	1111 0010 (binario)	F2 (hex.)	0	1	0

**Ejemplo 2:**

209 (decimal)	11010001 (binario)	D1 (hex.)	C	DC	Z
+ 56 (decimal)	+ 00111000 (binario)	38 (hex.)			
265 (decimal)	00001001 (binario)	09 (hex.)	1	0	0

En este último ejemplo se ha superado el valor máximo de 255 (decimal) y por tanto el Carry se activa a "1". En este caso el resultado obtenido b'000001001' (9 en decimal) no corresponde con el valor decimal correcto que es 265. Si se añade el flag Carry al resultado obtenido, se obtiene b'100001001', que sí coincide con el valor correcto de  $265_{10}$ . Como conclusión, se puede afirmar que el flag Carry es el noveno bit del registro de trabajo y por lo tanto del resultado.

El PIC16F84 tiene dos instrucciones de suma:

**8.1.1 addlw k**

(Add Literal to W). Suma el contenido del registro W con el literal o constante 'k'. Almacena el resultado en W. Si se produce acarreo el flag C se pone a "1".

Ejemplo 1: *addlw 0x4F ; (W) + 4F → (W)*

Antes instrucción:  $(W) = 0xA3$ , y  $C = ?$

Después instrucción:  $(W) = 0xF2$ , (ya que  $0xA3 + 0x4F = 0xF2$ ) y  $C = 0$ .

Ejemplo 2: *addlw 0x38 ; (W) + 4F → (W)*

Antes instrucción:  $(W) = 0xD1$ , y  $C = ?$

Después instrucción:  $(W) = 0x09$ , (ya que  $0x38 + 0xD1 = 0x09$  más acarreo) y  $C = 1$ .

**8.1.2 addwf f,d**

(Add W and f). Suma el contenido del registro W al contenido del registro 'f'. Almacena el resultado en W si 'd' = 0 y en el registro 'f' si 'd' = 1. Si se produce acarreo el flag C se pone a "1".

Ejemplo 1: *addwf PORTA,W ; (PORTA) + (W) → (W)*

Antes instrucción:  $(W) = 0x17$ ,  $(PORTA) = 0xC2$ , y  $C = ?$

Después instrucción:  $(W) = 0xD9$ ,  $(PORTA) = 0xC2$ , y  $C = 0$ .

Ejemplo 2: *addwf Contador,F ; (Contador) + (W) → (Contador)*

Antes instrucción:  $(W) = 0xD1$ ,  $(Contador) = 0x38$ , y  $C = ?$

Después instrucción:  $(W) = 0xD1$ ,  $(Contador) = 0x09$ , y  $C = 1$ .

**8.2 INSTRUCCIONES DE RESTA**

La resta se realiza entre el contenido del registro W y el literal o constante 'k'. La resta se efectúa con signo y el resultado es positivo. El resultado es positivo en tres casos:

- Si el resultado es menor que el minimo.
- Si el resultado es igual al minimo.
- Si el resultado es mayor que el maximo.

El PIC16F84 tiene tres instrucciones de resta:

**8.2.1 sublw**

(Subtract Word). Resta el literal o constante 'k' menos el contenido del registro W.

Ejemplo 1:

Antes instrucción:

Después instrucción:

Ejemplo 2:

Antes instrucción:

Después instrucción:

Ejemplo 3:

Antes instrucción:

Después instrucción:

**8.2.2 subwf**

(Subtract Word). Resta el contenido del registro W menos el contenido del registro 'f'. El resultado es uno.

Ejemplo 1:

Antes instrucción:

Después instrucción:

Ejemplo 2:

Antes instrucción:

Después instrucción:

DC	Z
0	0

(decimal) y por  
00001001' (9 en  
se añade el flag  
de con el valor  
es el noveno bit

l o constante 'k'.

C = 0.

acarreo) y C = 1.

o del registro 'f'.  
e produce acarreo

(W)

(Contador)

## 8.2 INSTRUCCIONES DE RESTA

La resta se realiza sumando, en binario puro sin signo, el registro "f" (o el literal "k") más el complemento a dos del contenido del registro W. Al realizar la resta en 8 bits con signo el resultado no puede exceder de +127 ni de (-128). El flag de Carry indica si el resultado es positivo (C se pone a "1") o es negativo (C se pone a "0"). Pueden ocurrir tres casos:

- Si el resultado es positivo distinto de cero, C = 1 y Z = 0.
- Si el resultado es cero, C = 1 y Z = 1.
- Si el resultado es negativo, C = 0 y Z = 0.

El PIC16F84 dispone de dos instrucciones de resta:

### 8.2.1 sublw k

*(Subtract W from Literal).* Resta (en complemento a 2) el contenido de la constante 'k' menos el contenido del registro W y almacena el resultado en W.

Ejemplo 1: `sublw 0x03 ; 03h - (W) → (W)`

Antes instrucción:  $(W) = 0x01, C = ? \text{ y } Z = ?$

Después instrucción:  $(W) = 0x02, C = 1 \text{ (el resultado es positivo) y } Z = 0.$

Ejemplo 2: `sublw 0x02 ; 02h - (W) → (W)`

Antes instrucción:  $(W) = 0x02, C = ? \text{ y } Z = ?$

Después instrucción:  $(W) = 0x00, C = 1 \text{ y } Z = 1 \text{ (el resultado es cero).}$

Ejemplo 3: `sublw 0x02 ; 02h - (W) → (W)`

Antes instrucción:  $(W) = 0x03, (+3 \text{ en decimal}), C = ? \text{ y } Z = ?$

Después instrucción:  $(W) = 0xFF, (-1 \text{ en decimal}), C = 0 \text{ (resultado negativo) y } Z = 0.$

### 8.2.2 subwf f,d

*(Subtract W from f).* Resta (en complemento a 2) el contenido del registro 'f' menos el contenido del registro W. Almacena el resultado en W si 'd' es cero y en 'f' si 'd' es uno.

Ejemplo 1: `subwf Reg1,F ; (Reg1) - (W) → (Reg1)`

Antes instrucción:  $(Reg1) = 0x03, (W) = 0x02, C = ? \text{ y } Z = ?$

Después instrucción:  $(Reg1) = 0x01, (W) = 0x02, C = 1 \text{ (positivo) y } Z = 0.$

Ejemplo 2: `subwf Reg1,F ; (Reg1) - (W) → (Reg1)`

Antes instrucción:  $(Reg1) = 0x02, (W) = 0x02, C = ? \text{ y } Z = ?$

Después instrucción:  $(Reg1) = 0x00, (W) = 0x02, C = 1 \text{ y } Z = 1 \text{ (resultado cero).}$

Ejemplo 3:	<code>subwf Reg1,F ; (Reg1) - (W) → (Reg1)</code>
Antes instrucción:	(Reg1) = 0x01, (+1 decimal), (W) = 0x02, C = ? y Z = ?
Después instrucción:	(Reg1) = 0xFF, (-1 decimal), (W) = 0x022, C = 0 (neg.) y Z = 0.

## 8.3 INCREMENTAR Y DECREMENTAR

El PIC16F84 posee otras instrucciones aritméticas. Son las siguientes:

### 8.3.1 decf f,d

(Decrement f). El contenido del registro 'f' se decrementa en una unidad. Almacena el resultado en W si 'd' = 0 (en cuyo caso 'f' no varía) y en el registro 'f' si 'd' = 1. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo 1:	<code>decf Contador,F ; (Contador) - 1 → (Contador)</code>
Antes instrucción:	(Contador) = 0x01 y Z = ?
Después instrucción:	(Contador) = 0x00 y Z = 1.

Ejemplo 2:	<code>decf Contador,W ; (Contador) - 1 → (W)</code>
Antes instrucción:	(Contador) = 0x04, (W) = ? y Z = ?
Después instrucción:	(Contador) = 0x04, (W) = 0x03 y Z = 0.

### 8.3.2 incf f,d

(Increment f). El contenido del registro 'f' se incrementa en una unidad. Almacena el resultado en W si 'd' = 0 (en cuyo caso 'f' no varía) y en el registro 'f' si 'd' = 1. El flag Z se activa a "1" si el resultado de la operación es cero, es decir, si hay desbordamiento al pasar de b'11111111' a b'00000000'.

Ejemplo 1:	<code>incf Contador,F ; (Contador) + 1 → (Contador)</code>
Antes instrucción:	(Contador) = 0xFF y Z = ?
Después instrucción:	(Contador) = 0x00 y Z = 1.

Ejemplo 2:	<code>incf Contador,W ; (Contador) + 1 → (W)</code>
Antes instrucción:	(Contador) = 0x01, (W) = ? y Z = ?
Después instrucción:	(Contador) = 0x01, (W) = 0x02 y Z = 0.

## 8.4 INSTRUCCIONES LÓGICAS

Las operaciones lógicas que se pueden realizar con el PIC16F84 son la AND, OR, OR exclusiva, inversión (o complemento), la rotación y el intercambio de nibbles:

### 8.4.1 and

(AND) registro W y el resultado de la

Ejemplo:  
Antes instrucción:  
Después instrucción:

### 8.4.2 and

(AND W y el contenido de

Ejemplo 1:  
Antes instrucción:  
Después instrucción:

Ejemplo 2:  
Antes instrucción:  
Después instrucción:

### 8.4.3 comt

(Completo) invirtiendo su resultado en W y Z se activa a "1".

Ejemplo 1:  
Antes instrucción:  
Después instrucción:

Ejemplo 2:  
Antes instrucción:  
Después instrucción:

### 8.4.4 iorlw

(Inclusivo o) del registro W y el resultado de la

g1)  
;? y Z = ?  
0 (neg.) y Z = 0.

#### 8.4.1 andlw k

(*AND Literal with W*). Efectúa la operación AND lógica entre el contenido del registro W y la constante 'k'. Almacena el resultado en W. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo: `andlw b'01011111'; (W) AND b'01011111' → (W)`

Antes instrucción:  $(W) = b'10100011'$  y  $Z = ?$

Después instrucción:  $(W) = b'00000011'$  y  $Z = 0$ .

#### 8.4.2 andwf f,d

(*AND W with f*). Efectúa la operación AND lógica entre el contenido del registro W y el contenido del registro 'f'. Almacena el resultado en W si 'd' = 0 y en el registro 'f' si 'd' = 1. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo 1: `andwf FSR,F ; (W) AND (FSR) → (FSR)`

Antes instrucción:  $(W) = b'00101111'$ ,  $(FSR) = b'11000010'$  y  $Z = ?$

Después instrucción:  $(W) = b'00101111'$ ,  $(FSR) = b'00000010'$  y  $Z = 0$ .

Ejemplo 2: `andwf FSR,W ; (W) AND (FSR) → (W)`

Antes instrucción:  $(W) = b'00101111'$ ,  $(FSR) = b'11000010'$  y  $Z = ?$

Después instrucción:  $(W) = b'00000010'$ ,  $(FSR) = b'11000010'$  y  $Z = 0$ .

#### 8.4.3 comf f,d

(*Complement f*). Realiza un complemento del contenido del registro 'f' bit a bit, invirtiendo su valor, es decir, cambia los unos por ceros y viceversa. Almacena el resultado en W si 'd' = 0 (en cuyo caso 'f' no varía) y en el registro 'f' si 'd' = 1. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo 1: `comf Reg1,F ; (/Reg1) → (Reg1)`

Antes instrucción:  $(Reg1) = b'00010011'$ , y  $Z = ?$

Después instrucción:  $(Reg1) = b'11101100'$  (invertidos unos y ceros) y  $Z = 0$ .

Ejemplo 2: `comf Reg1,W ; (/Reg1) → (W)`

Antes instrucción:  $(Reg1) = b'00010011'$ ,  $(W) = ?$  y  $Z = ?$

Después instrucción:  $(Reg1) = b'00010011'$ ,  $(W) = b'11101100'$  (invertidos unos y ceros) y  $Z = 0$ .

#### 8.4.4 iorlw k

(*Inclusive OR Literal with W*). Efectúa la operación OR lógica entre el contenido del registro W y la constante 'k'. Almacena el resultado en W. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo:  $iorlw \quad b'00110101' ; (W) OR b'00110101' \rightarrow (W)$   
 Antes instrucción:  $(W) = b'10011010'$  y  $Z = ?$   
 Despues instrucción:  $(W) = b'10111111'$  y  $Z = 0.$

#### 8.4.5 iorwf f,d

(Inclusive OR W with f). Efectúa la operación OR lógica entre el contenido del registro W y el contenido del registro 'f'. Almacena el resultado en W si 'd' = 0 y en el registro 'f' si 'd' = 1. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo:  $iorwf \quad Resultado, W ; (W) OR (Resultado) \rightarrow (W)$   
 Antes instrucción:  $(Resultado) = b'00010011'$ ,  $(W) = b'10010001'$  y  $Z = ?$   
 Despues instrucción:  $(Resultado) = b'00010011'$ ,  $(W) = b'10010011'$  y  $Z = 0.$

#### 8.4.6 rlf f,d

(Rotate Left f through Carry). Rotación de un bit a la izquierda del registro 'f', pasando por el bit de acarreo C. El desplazamiento es cerrado, formando un anillo con el bit C (Carry) del registro de estado o STATUS. Si 'd' = 1 el resultado se almacena en 'f', si 'd' = 0 el resultado se almacena en W. El contenido del Carry pasa a la posición del bit de menor peso y el bit de mayor peso pasa al Carry (figura 8-1).

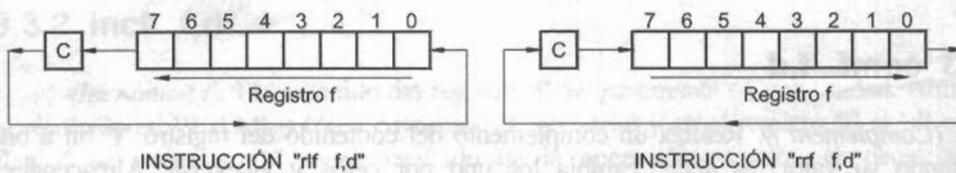


Figura 8-1 Instrucciones de rotación

Ejemplo 1:  $rlf \quad Reg1, W$   
 Antes instrucción:  $(Reg1) = b'1110\ 0110'$ ,  $W = ?$  y  $C = 0.$   
 Despues instrucción:  $(Reg1) = b'1110\ 0110'$ ,  $(W) = b'1100\ 1100'$  y  $C = 1.$

Ejemplo 2:  $rlf \quad Reg1, F$   
 Antes instrucción:  $(Reg1) = b'1110\ 0110'$  y  $C = 0.$   
 Despues instrucción:  $(Reg1) = b'1100\ 1100'$  y  $C = 1.$

#### 8.4.7 rrf f,d

(Rotate Right f through Carry). Rotación de un bit a la derecha del registro 'f', pasando por el bit de acarreo C. Similar que el anterior pero el desplazamiento es a la derecha tal como se ilustra en la figura 8-1.

Ejemplo 1  
 Antes instrucción  
 Despues instrucción

Ejemplo 2  
 Antes instrucción  
 Despues instrucción

#### 8.4.8 s

(Sw) los 4 bits de si 'd' = 1 e

Ejemplo 1:  
 Antes instrucción  
 Despues instrucción

Ejemplo 2:  
 Antes instrucción  
 Despues instrucción

#### 8.4.9 xc

(Exc) del registro activa a "1"

Ejemplo:  
 Antes instrucción  
 Despues instrucción

#### 8.4.10 xc

(Excl) registro W y es 0. El flag

Ejemplo 1:  
 Antes instrucción  
 Despues instrucción

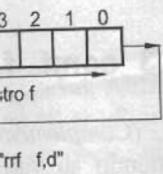
Ejemplo 2:  
 Antes instrucción  
 Despues instrucción

1' → (W)

el contenido del  
si 'd' = 0 y en el  
ción es cero.

0 → (W)  
001' y Z = ?.  
011' y Z = 0.

da del registro 'f',  
ndo un anillo con el  
se almacena en 'f',  
a la posición del bit



' y C = 1.

echo del registro 'f',  
splazamiento es a la

Ejemplo1:	<i>rrf RegI,W</i>
Antes instrucción:	(RegI) = b'1110 0110', (W) = ? y C = 0.
Después instrucción:	(RegI) = b'1110 0110', (W) = b'0111 0011' y C = 0.
Ejemplo2:	<i>rrf RegI,F</i>
Antes instrucción:	(RegI) = b'1110 0110', y C = 0.
Después instrucción:	(RegI) = b'0111 0011', y C = 0.

#### 8.4.8 swapf f,d

*(Swap Nibbles in f).* Los cuatro bits de más peso del registro 'f' se intercambian con los 4 bits de menos peso del mismo registro 'f'. Si 'd' = 0 el resultado se almacena en W, si 'd' = 1 el resultado se almacena en 'f'.

Ejemplo 1:	<i>swapf RegI,W</i>
Antes instrucción:	(RegI) = 0xA5 y (W) = ?
Después instrucción:	(RegI) = 0xA5 y (W) = 0x5A.
Ejemplo 2:	<i>swapf RegI,F</i>
Antes instrucción:	(RegI) = 0xA5.
Después instrucción:	(RegI) = 0x5A.

#### 8.4.9 xorlw k

*(Exclusive OR Literal with W).* Realiza la función OR-Exclusiva entre el contenido del registro W y la constante 'k' de 8 bits. El resultado se almacena en W. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo:	<i>xorlw b'10101111'; (W) XOR b'10101111' → (W)</i>
Antes instrucción:	(W) = b'10110101' y Z = ?
Después instrucción:	(W) = b'00011010' y Z = 0.

#### 8.4.10 xorwf f,d

*(Exclusive OR W with f).* Realiza la función OR-Exclusiva entre el contenido del registro W y el contenido del registro f. Almacena el resultado en f si 'd' = 1 y en W si 'd' es 0. El flag Z se activa a "1" si el resultado de la operación es cero.

Ejemplo1:	<i>xorwf Reg,F ; (W) XOR (Reg) → (Reg)</i>
Antes instrucción:	(Reg) = b'10101111', (W) = b'10110101' y Z = ?
Después instrucción:	(Reg) = b'00011010', (W) = b'10110101' y Z = 0.
Ejemplo2:	<i>xorwf Reg,W ; (W) XOR (Reg) → (W)</i>
Antes instrucción:	(Reg) = b'10101111', (W) = b'10110101' y Z = ?
Después instrucción:	(Reg) = b'10101111', (W) = b'00011010' y Z = 0.

## 8.5 INSTRUCCIÓN “SLEEP”

Son abundantes las situaciones reales de trabajo en que el microcontrolador debe esperar, sin hacer nada, a que se produzca algún acontecimiento externo que le ponga de nuevo en funcionamiento. Para ahorrar energía, los microcontroladores PIC disponen de la instrucción especial *sleep* que les pasa al estado de reposo o de bajo consumo, en el cual los requerimientos de potencia son mínimos. En dicho estado se detiene el reloj principal y se “congelan” sus circuitos asociados, quedando el microcontrolador en un “sueño” (“sleep” en inglés). Al activarse una “interrupción” ocasionada por el acontecimiento esperado, el microcontrolador se “despierta” y reanuda su trabajo.

El concepto de interrupción será estudiado con detenimiento en capítulos posteriores. Por ahora, sólo hay que conocer que el modo de bajo consumo es un modo especial de funcionamiento que ocasiona un consumo muy bajo y se entra en él con la ejecución de la instrucción *sleep*.

En el siguiente ejemplo se repite el programa de lectura del Puerto A analizado en capítulos anteriores, con la particularidad que al finalizar entra en modo bajo consumo mediante la instrucción *sleep*.

```
***** Elemental_10.asm *****
;
; Por el Puerto B se obtiene el dato de las cinco líneas del Puerto A al que están conectado
; un array de interruptores. Por ejemplo, si por el Puerto A se introduce "--11001", por
; el Puerto B aparecerá "xxx11001" (el valor de las tres líneas superiores no importa).
;
; Esta operación la realizará una única vez. Después el programa entrará en modo
; "Standby" o de bajo consumo del cual no podrá salir después.
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC ; Configuración para el
; grabador.
LIST      P=16F84A      ; Procesador utilizado.
INCLUDE <P16F84A.INC> ; Definición de algunos operandos utilizados.
;
; ZONA DE CÓDIGOS *****
ORG      0          ; El programa comienza en la dirección 0.
Inicio
    bsf    STATUS,RP0   ; Acceso al Banco 1.
    clrf   TRISB       ; Las líneas del Puerto B se configuran como salida.
    movlw  b'00011111'  ; Las 5 líneas del Puerto A se configuran como entrada.
    movwf  TRISA
    bcf    STATUS,RP0   ; Acceso al Banco 0.
Principal
    movf   PORTA,W     ; Carga el registro de datos del Puerto A en W.
    movwf  PORTB       ; El contenido de W se deposita en el Puerto B.
    sleep             ; El programa entra en modo "Bajo Consumo" del cual no
                      ; podrá salir.
    END               ; Fin del programa.
```

## 8.6 AL

A ce  
realización e

- Incre  
Conta  
incf
- Incre  
unida  
movlw  
addwf
- Decre  
Centr  
decf C
- Decre  
unidad  
movlw  
subwf
- Incre  
de dato  
decrem  
tanto, p  
forma:  
addlw 0x
- Decrem  
hacerlo  
sublw
- Comple  
de la op  
con un "  
xorlw b'1
- Negar V  
sublw

## 8.6 ALGUNAS INSTRUCCIONES ÚTILES

A continuación se comentan algunas instrucciones de gran utilidad en la realización de programas de aplicación.

- **Incrementar un registro en una unidad.** Por ejemplo el registro llamado Contador.

```
incf    Contador,F
```

- **Incrementar un registro en un valor “n”.** Por ejemplo, para incrementar en 7 unidades el registro Operando.

```
movlw   d'7'  
addwf   Operando,F
```

- **Decrementar un registro en una unidad.** Por ejemplo el registro llamado Centenas.

```
decf    Centenas,F
```

- **Decrementar un registro en un valor “n”.** Por ejemplo, decrementar en 15 unidades el registro MensajeLongitud.

```
movlw   15  
subwf   MensajeLongitud,F
```

- **Incrementar W.** El registro de trabajo W no ocupa posición en la memoria RAM de datos, por tanto no se puede operar directamente con él para incrementarle o decrementarle. Así por ejemplo la instrucción “*incf W,1*” no es admitida. Por tanto, para incrementar el contenido del registro W hay que hacerlo de la siguiente forma:

```
addlw 0x01
```

- **Decrementar W.** Por el mismo motivo anterior, para decrementar W hay que hacerlo de la siguiente forma:

```
sublw 0x01
```

- **Complementar W, invirtiendo sus bits.** Se hace aprovechando la particularidad de la operación lógica XOR que complementa un bit cuando se realiza una XOR con un “1”: (bit) XOR 1 = (/bit).

```
xorlw b'11111111
```

- **Negar W.** Hallando su complemento a 2, es decir, restándole de cero.

```
sublw 0 ; ya que 0 - (W) = -(W)
```

- Poner a cero varios bits del registro W sin alterar el resto.** Se usa la operación lógica AND, poniendo en el operando inmediato un 0 en aquellos bits que se desea poner a "0" mientras que los restantes se ponen a "1". A la constante con la que se hace la operación AND se la llama **máscara**. Así por ejemplo, si desea poner a "0" los bits pares del registro W sin modificar los restantes puede usar la instrucción:

```
andlw b'10101010'
```

- Poner a uno varios bits del registro 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". A la constante con la que se hace la operación OR se la llama **máscara**. Así por ejemplo, si se desea poner a "1" los 5 bits de menos peso de W sin modificar los otros tres, puede usarse la instrucción:

```
iorlw b'00011111'
```

- Invertir varios bits de w sin alterar el resto.** Se usa la operación lógica XOR, poniendo en el operando inmediato un "1" en aquellos bits que se desea invertir mientras que los restantes se ponen a "0". A la constante con la que se hace la operación XOR se la llama **máscara**. Así por ejemplo, si desea invertir los 3 bits de mayor peso de W sin modificar los otros cinco, puede usar la instrucción:

```
xorlw b'11100000'
```

- Multiplicar un registro por un numero potencia de 2.** Se puede hacer fácilmente desplazando un bit varias posiciones hacia la izquierda e introduciendo ceros por la derecha. Se debe tener precaución de no perder dígitos significativos por la izquierda, es decir, el resultado no puede ser mayor de 255 que es el máximo número representable con 8 bits. Por ejemplo para multiplicar el contenido del registro Operando por 2, 4 y 8 se haría:

bcf STATUS,C	; El cero a introducir por la derecha.
rlf Operando,F	; Se multiplica por dos.
bcf STATUS,C	; El cero a introducir por la derecha.
rlf Operando,F	; Se multiplica por cuatro.
bcf STATUS,C	; El cero a introducir por la derecha.
rlf Operando,F	; Se multiplica por ocho.

; Así por ejemplo, si en principio (Operando) = b'00011010' = 26 (decimal),  
; por cada desplazamiento a la derecha quedaría:  
; (Operando) = b'00011010' = 26 (decimal), al principio.  
; (Operando) = b'00110100' = 52 (decimal), ha multiplicado por 2, (26 x 2 = 52).  
; (Operando) = b'01101000' = 104 (decimal), ha multiplicado por 4, (26 x 4 = 104).  
; (Operando) = b'11010000' = 208 (decimal), ha multiplicado por 8, (26 x 8 = 208).

- Dividir** un número en el registro Operando por 2, 4 y 8 desplazando bits hacia la izquierda.

bcf STATUS,C

rlf Operando,F

bcf STATUS,C

rlf Operando,F

bcf STATUS,C

rlf Operando,F

; Así por ejemplo,

; por cada desplazamiento a la derecha

; (Operando) = b'00011010' = 26 (decimal), al principio.

; (Operando) = b'00110100' = 52 (decimal), ha multiplicado por 2, (26 x 2 = 52).

; (Operando) = b'01101000' = 104 (decimal), ha multiplicado por 4, (26 x 4 = 104).

; (Operando) = b'11010000' = 208 (decimal), ha multiplicado por 8, (26 x 8 = 208).

## 8.7 HERRAMIENTAS

Cuando se trabaja con el PIC16F84 es necesario tener las siguientes herramientas para la puesta a punto del sistema:

- Un ordenador con Windows.
- Un lector de memoria.
- Un programador.
- Un ensamblador.
- Un simulador.

### 8.7.1 Ensamblador

El programa que se incluye en el CD-ROM contiene el ensamblador más avanzado y completo que existe para el PIC16F84. Los mnemónicos del lenguaje de ensamblador están basados en el lenguaje C. Al finalizar el proceso de ensamblado, el resultado es una secuencia de bytes que se grabará en la memoria del PIC16F84.

El programa que se incluye en el CD-ROM es de alto nivel (por lo tanto es más fácil de usar que el tradicional lenguaje de ensamblador). El programa es un ensamblador para el PIC16F84 desarrollado por MicroEngineering Systems (www.ccsinfo.com).

### 8.7.2 Simulador

Una vez que se tiene el ensamblador, es necesario tener un simulador para la posesión de un fichero de ejecución.

a la operación  
s que se desea  
con la que se  
ea poner a "0"  
nstrucción:

a la operación  
s que se desea  
e con la que se  
desea poner a  
uede usarse la

n lógica XOR,  
desea invertir  
que se hace la  
invertir los 3 bits  
ucción:

e puede hacer  
introduciendo  
s significativos  
e es el máximo  
contenido del

- **Dividir un registro entre un número potencias de 2.** Se puede hacer fácilmente desplazando un bit varias posiciones hacia la derecha e introduciendo ceros por la izquierda. Por ejemplo para dividir el registro Operando por 2, 4 y 8 se haría:

bcf	STATUS,C	; El cero a introducir por la izquierda.
rff	Operando,F	; Se divide entre dos.
bcf	STATUS,C	; El cero a introducir por la izquierda.
rff	Operando,F	; Se divide entre cuatro.
bcf	STATUS,C	; El cero a introducir por la izquierda.
rff	Operando,F	; Se divide entre ocho.

; Así por ejemplo, si en principio (Operando) = b'00011010' = 26 (decimal),  
; por cada desplazamiento a la izquierda quedaría:  
; (Operando) = b'00011010' = 26 (decimal), al principio.  
; (Operando) = b'00001101' = 13 (decimal), ha dividido entre 2, (26/2 = 13).  
; (Operando) = b'00000110' = 6 (decimal), ha dividido entre 4, (26/4 = 6).  
; (Operando) = b'00000011' = 3 (decimal), ha dividido entre 8, (26/8 = 3).

## 8.7 HERRAMIENTAS

Cuando se diseñan sistemas con circuitos programables se precisan herramientas para la puesta a punto del hardware y del software. Las más importantes son: editores de texto, ensambladores y compiladores, simuladores, grabadores, emuladores y sistemas de desarrollo.

### 8.7.1 Ensambladores y compiladores

El programa **ensamblador** traduce las instrucciones que se han escrito, usando los nemáticos del lenguaje máquina, a código binario ejecutable por el microcontrolador. El proceso de ensamblado produce un fichero \*.hex que será el que posteriormente se grabará en la memoria de programa del PIC mediante el grabador o programador. La secuencia de nemáticos se llama código fuente del programa. El ensamblador más utilizado para los PIC es el MPASM que trabaja dentro del entorno software MPLAB.

El programa compilador traduce las instrucciones que se han escrito en lenguaje de alto nivel (por ejemplo en lenguaje C), a código binario ejecutable por el microcontrolador. Los compiladores para lenguaje C más populares son el **PICC** desarrollado por *Hi-Tech Software* ([www.htsoft.com](http://www.htsoft.com)) y el **PCW** de la empresa *CCS* ([www.ccsinfo.com](http://www.ccsinfo.com)). Un compilador para Basic es el **PICBasic Pro** propiedad de *MicroEngineering Labs* ([www.melabs.com](http://www.melabs.com)).

### 8.7.2 Simuladores software

Una vez que el programa se ha escrito y ensamblado o compilado se está en posesión de un fichero binario \*.hex que es el que se graba en el microcontrolador. Es

casi indispensable probar este programa, haciéndolo funcionar en condiciones tan próximas como sea posible a las de utilización real. Para hacer esto hay varias posibles soluciones, las más utilizadas son dos: utilizar un económico simulador software o un potente emulador.

Como su propio nombre indica, un **simulador** por software "simula" la ejecución de las instrucciones de un programa desarrollado para un modelo de microcontrolador específico. Representa el comportamiento interno del microcontrolador y el estado de sus líneas de entrada/salida. Al ejecutar el simulador se pueden visualizar fácilmente las instrucciones donde se producen funcionamientos no deseados. Como el microcontrolador se simula por software el comportamiento no es idéntico a la realidad, sin embargo, proporciona una aproximación aceptable, especialmente cuando no es esencial el trabajo en tiempo real. Su gran ventaja es el bajo precio.

El simulador realiza la ejecución del programa mucho más lento que lo haría el mismo programa directamente sobre el microcontrolador, por eso determinadas operaciones en las que son necesarios tiempos muy precisos o críticos no se pueden probar mediante la simulación. No obstante, bien utilizado permite desarrollar aplicaciones interesantes con una mínima inversión.

El simulador gratuito más utilizado para los PIC, es el **MPLAB SIM** que trabaja dentro del entorno MPLAB ya analizado en el capítulo anterior.

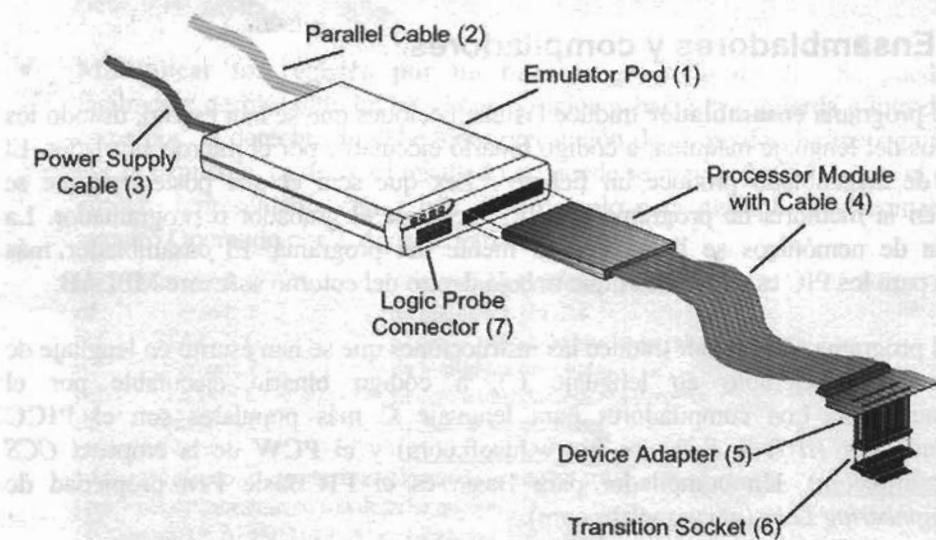


Figura 8-2 Emulador MPLAB-ICE 2000, (cortesía de Microchip Technology Inc)

### 8.7.3 E

El controlado  
microcont  
página We

El c  
microcont  
microcont  
sistema co  
visualiza e  
funcionam  
resultados  
simuladore

El e  
pero su a  
doméstico.

### 8.7.4 G

Una  
proceder a  
grabador o  
PLUS ofre

En I  
programado  
que se utili  
en el capítu

### 8.7.5 S

Un s  
software qu  
desarrollo d  
Technology

Hay  
desarrollar  
eficaces y c

condiciones tan  
varias posibles  
r software o un

"emula" la ejecución  
microcontrolador  
el estado de sus  
fácilmente las  
os. Como el  
ico a la realidad,  
e cuando no es

o que lo haría el  
so determinadas  
os no se pueden  
mite desarrollar

SIM que trabaja

odule  
(4)



technology Inc)

### 8.7.3 Emuladores

El **emulador** es una potente herramienta que consiste en un complejo equipo físico controlado por un software desde un ordenador y que se comporta exactamente como el microcontrolador al que reemplaza. El fabricante *Microchip* ofrece algunos de ellos en su página Web, tal como el MPLAB-ICE 2000 (figura 8-2).

El emulador ICE 2000 dispone de una "cabeza" con idéntico patillaje que el del microcontrolador que emula. Esta cabeza se inserta en el zócalo donde irá el microcontrolador con el programa que trata de comprobar. El emulador hace funcionar el sistema como si hubiese un microcontrolador real y además con la ventaja de que visualiza en el monitor del ordenador toda la información necesaria para comprobar el funcionamiento de los programas, permitiendo realizar todo tipo de pruebas. Los resultados obtenidos son idénticos a los del producto final, puesto que a diferencia de los simuladores software la ejecución se realiza en tiempo real.

El emulador es el método de depuración más sofisticado que se puede emplear, pero su alto precio no lo hace especialmente accesible y mucho menos para uso doméstico. Para muchas aplicaciones es suficiente con el simulador software.

### 8.7.4 Grabadores o programadores

Una vez realizado el programa y comprobado en el simulador o emulador, se debe proceder a grabarlo en la memoria de programa del microcontrolador mediante un grabador o programador, tal como se estudió en el capítulo 3. El grabador PICSTART PLUS ofrecido por *Microchip* es uno de los más utilizados.

En Internet se ofrecen numerosos esquemas para la construcción de económicos programadores de PIC. Uno de los más populares es el JDM en sus diferentes versiones, que se utiliza junto con el programa IC-Prog. Estos ya fueron ampliamente comentados en el capítulo 3.

### 8.7.5 Sistemas de desarrollo

Un sistema de desarrollo está formado por un conjunto de herramientas hardware y software que permite el desarrollo de proyectos con microcontroladores. Todo sistema de desarrollo debe contar con una placa donde probar las diferentes aplicaciones. *Microchip Technology Inc* ofrece la placa de demostración PICDEM 4 (figura 8-3).

Hay muchas empresas que ofrecen completos sistemas de desarrollo para desarrollar proyectos complejos con microcontroladores PIC, que pueden incluir hasta eficaces y costosos emuladores.

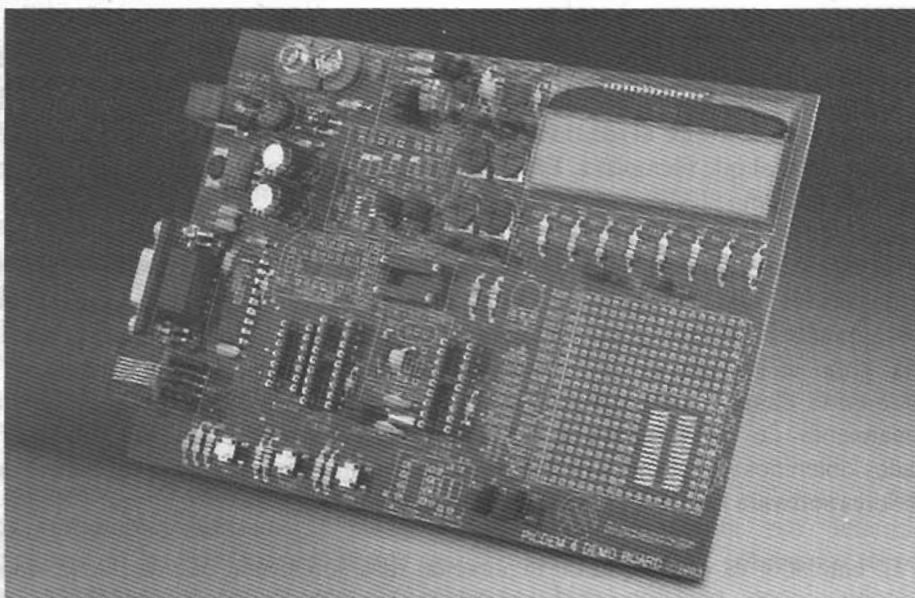


Figura 8-3 PICDEM 4 Demo Board, (cortesía de Microchip Technology Inc)

## 8.8 PROYECTOS CON MEDIOS REDUCIDOS

*Microchip Technology Inc* se ha dado cuenta de que para vender mejor sus microcontroladores tiene que facilitar y popularizar su empleo, de modo que ha puesto a disposición de los usuarios particulares herramientas eficaces y económicas que les permitan desarrollar proyectos.

Uno de los motivos del gran éxito de los microcontroladores PIC es la posibilidad de que el usuario realice su propio sistema de desarrollo con muy pocos medios. Este libro pretende ayudar en esta tarea: que el lector pueda construir su propio sistema de desarrollo de proyectos con medios reducidos. El sistema de desarrollo utilizado en este libro para el aprendizaje del microcontrolador PIC16F84 está compuesto por:

- A nivel **software**: El entorno MPLAB, que proporciona editor de programas, ensamblador y simulador, explicado en capítulo 7.
- **Grabador**: Se utilizará un grabador TE20-SE junto con el software IC-Prog explicados en capítulo 3.
- **Placa de montajes**. El entrenador descrito en la figura 1-2 servirá para el aprendizaje de las características básicas del PIC16F84. Este circuito es muy sencillo de montar y económico. A lo largo del libro se irán describiendo otros.

## 8.9 DESAM

Una de la posibilidad de utr proyectos, inclus

Cada nuev implementa. La n descomponer en desarrollo de cada propias. La figura medios reducidos:

1º Definir cl prestacione determinan programa.

2º Diseñar el circuito fisi

3º Construcci componentes la fase anter y verificar e

4º Plantear el general del este paso correctamen y en errores.

5º Editar el pr el editor del

6º Ensamblar programa fu ensamblador

7º Simular el p memoria del programa. El MPLAB.

## 8.9 DESARROLLO DE PROYECTOS SENCILLOS

Una de las grandes ventajas que aportan los microcontroladores PIC es la posibilidad de utilizar medios muy económicos para poder desarrollar gran cantidad de proyectos, incluso de cierta complejidad.

Cada nuevo proyecto nace con una idea y termina con el prototipo que la implementa. La realización de cualquier proyecto para un microcontrolador se puede descomponer en varias fases, aunque no hay una separación nítida entre ellas. En el desarrollo de cada fase hay que utilizar un conjunto de herramientas hardware y software propias. La figura 8-4 describe las fases que constituyen el desarrollo de proyectos con medios reducidos:

- 1º **Definir** claramente el proyecto que se desea realizar con sus características y prestaciones hasta la absoluta comprensión de lo que se desea realizar. Se determinarán las necesidades de hardware, así como las partes que integrarán el programa.
- 2º **Diseñar el circuito físico** que implementará el proyecto. Es decir, dibujar el circuito físico que va a controlar el programa.
- 3º **Construcción del prototipo.** En esta fase hay que adquirir todos los componentes que configuran el proyecto. Realizar el circuito físico diseñado en la fase anterior, montándolo en una placa de circuito impreso, *wrapping o Board*, y verificar el comportamiento en lo que sea posible.
- 4º **Plantear el programa** necesario. Es buena práctica dibujar el diagrama de flujo general del programa antes de comenzar la edición del código fuente. Depurar este paso hasta la saciedad. Todo el tiempo que se utilice en diseñar correctamente el programa se ahorra posteriormente en tiempo de programación y en errores. Ya se tiene el diseño en papel de lo que se desea hacer.
- 5º **Editar el programa** de control del sistema utilizando lenguaje ensamblador en el editor del MPLAB.
- 6º **Ensamblar** el programa convirtiéndolo a código máquina grabable en ROM, el programa fuente editado en la fase anterior. Para ello se utilizará el programa ensamblador MPASM que se facilita en el MPLAB.
- 7º **Simular** el programa comprobando su comportamiento antes de grabarlo en la memoria del microcontrolador. Si hubiese algún error hay que volver a editar el programa. El simulador más accesible es el MPLAB SIM integrado dentro del MPLAB.

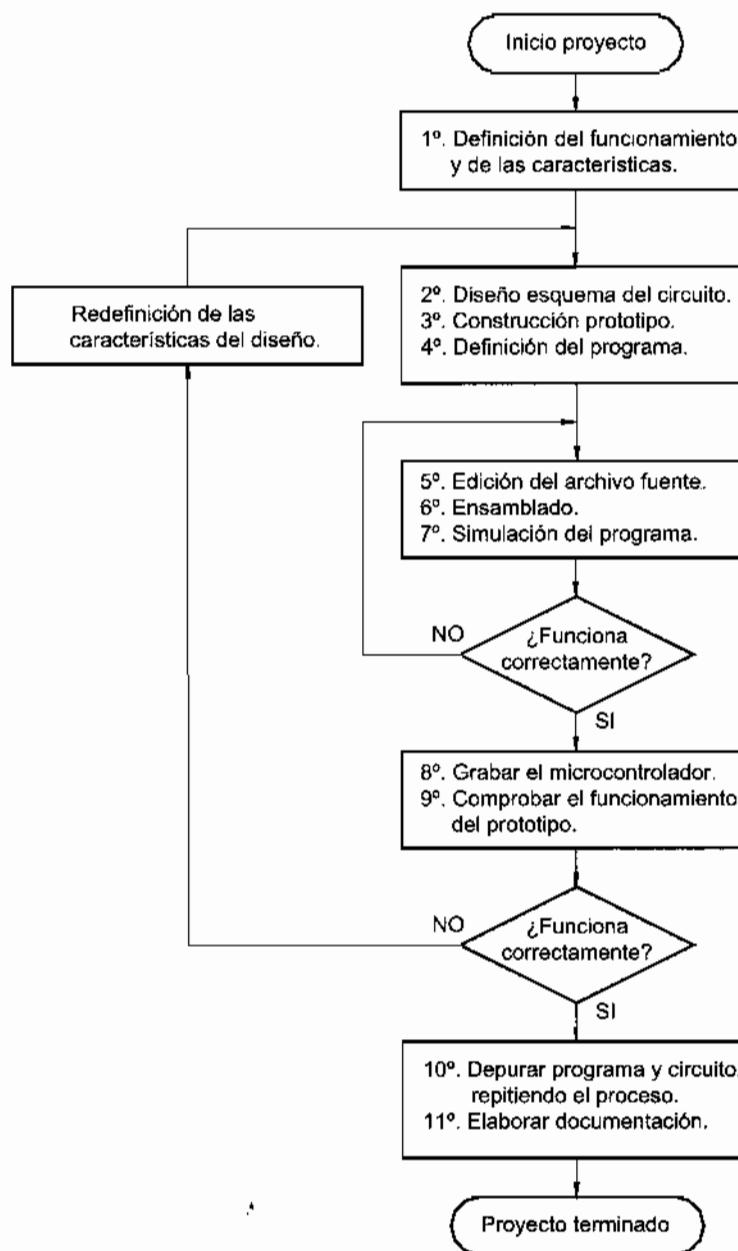


Figura 8-4 Proceso de realización de un proyecto con microcontrolador

- 8º **Grabar el microcontrolador** una vez comprobado el correcto funcionamiento del programa. Para ello se puede utilizar el programador TE20-SE y el programa IC-Prog. El fichero a grabar en la memoria del microcontrolador es generalmente el

ejecutable con extensión \*.hex y se obtiene después de realizar el ensamblado del código fuente.

- 9º **Comprobar** el funcionamiento del prototipo con el microcontrolador grabado. Si no cumple el funcionamiento previsto hay que volver a repetir todos los pasos de edición del programa, ensamblado, simulación y grabación con las oportunas correcciones.
- 10º **Depurar** el programa y el circuito repitiendo el proceso hasta alcanzar los objetivos perseguidos.
- 11º Elaborar la **documentación** de todo el proyecto.

Un principio general de la programación es que el software es equivalente en importancia al hardware. Esto significa que un dominio adecuado de las herramientas del ensamblador permite alcanzar un nivel elevado de aprovechamiento de las prestaciones hardware del microcontrolador, sin embargo, un deficiente aprendizaje de la programación obliga generalmente a adoptar soluciones hardware improvisadas y particulares a problemas que podrían resolverse a nivel software con mucho menor coste y mayor flexibilidad.

## 8.10 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador y comprobar los siguientes programas para el esquema del entrenador de la figura 1-2. El lector puede introducir todas las mejoras que considere conveniente. Las soluciones se ofrecen en el CD-ROM que acompaña a este libro.

**Elemental\_01.asm:** Por el Puerto B se obtiene el dato de las cinco líneas del Puerto A, al que está conectado un array de interruptores, sumándole el valor de una constante, por ejemplo 74 decimal. Es decir:  $(PORTB) = (PORTA) + \text{Constante}$ .

**Elemental\_02.asm:** Por el Puerto B se obtiene el dato del Puerto A multiplicado por 2. Es decir:  $(PORTB) = 2 \cdot (PORTA) = (PORTA) + (PORTA)$

**Elemental\_03.asm:** Por el Puerto B se obtiene el dato introducido por el Puerto A, pero los bits pares de la salida se fijan siempre a “1”. El orden de los bits será “b7 b6 b5 b4 b3 b2 b1 b0”, siendo los pares el b6, b4, b2 y b0. Por ejemplo, si por el Puerto A se introduce el dato ‘---01100’, por el Puerto B se visualiza ‘---11101’. Observa que:

- Los bits pares están a “1”, efectivamente  $(Puerto B) = \text{---}1x1x1$
- Los impares permanecen con el dato del puerto de entrada, efectivamente:  $(Puerto A) = \text{---}x1x0x$  y  $(Puerto B) = \text{---}x1x0x$

**Elemental\_04.asm:** Por el Puerto B se obtiene el contenido del Puerto A, pero los bits impares de la salida se fijan siempre a "0". El orden de los bits será "b7 b6 b5 b4 b3 b2 b1 b0", siendo los impares el b7, b5, b3 y b1. Por ejemplo, si por el Puerto A se introduce el dato b'---01100', por el Puerto B se visualiza '00000100'. Observar que:

- Los bits impares están a "0", efectivamente: (Puerto B) = '0x0x0x0x'.
- Los pares permanecen con el dato del puerto de entrada, efectivamente: (Puerto A) = '---0x1x0' y (Puerto B) = '---0x1x0'.

**Elemental\_05.asm:** Por el Puerto B se obtiene el dato del Puerto A invertidos los unos y los ceros. Por ejemplo, si por el Puerto A se introduce "\_\_\_11001", por el Puerto B aparecerá "xxx00110". (No importa el estado de los tres bits superiores del Puerto B).

**Elemental\_06.asm:** Por el Puerto B se obtiene el dato del Puerto A intercambiando los nibbles alto y bajo. Por ejemplo, si por el Puerto A se introduce "\_\_\_1001", por el Puerto B aparecerá "1001xxx1".

**Elemental\_07.asm:** Por el Puerto B se obtiene el dato del Puerto A desplazando un bit hacia la izquierda, por la derecha entrará un "1". Por ejemplo, si por el Puerto A se introduce "\_\_\_11001", por el Puerto B aparecerá "xx110011".

**Elemental\_08.asm:** Por el Puerto B se saca el dato del Puerto A desplazando un bit hacia la derecha, por la izquierda entrará un "0". Por ejemplo, si por el Puerto A se introduce "\_\_\_11001", por el Puerto B aparecerá "0xxx1100".

**Elemental\_09.asm:** Por el Puerto B se saca el dato del Puerto A invirtiendo los bits pares. Los impares se dejan como en la entrada.

**Elemental\_10.asm:** Por el Puerto B se obtiene el dato de las cinco líneas del Puerto A al que están conectado un array de interruptores. Esta operación la realizará una única vez. Después el programa entrará en modo *Standby* o de bajo consumo del cual no podrá salir.

La  
instrucció  
por una t  
PIC16F84

## 9.1 SA

En  
incondici  
condiciona  
una condici  
condiciona

El re  
condiciona

- Aq  
bj
- Aq  
dist

Puerto A, pero los bits "b7 b6 b5 b4 b3" por el Puerto A se observar que:

'0x0x0x'.  
ativamente:

o A invertidos los bits "1", por el Puerto B (del Puerto B).

o del Puerto A  
A se introduce "---

erto A desplazando un bit por el Puerto A se

o A desplazando un bit por el Puerto A se

o A invirtiendo los

as cinco líneas de dirección la realizará una bifurcación del consumo del cual no

## CAPÍTULO 9

# SALTOS

La ejecución de los programas no suele ser lineal ejecutándose una lista de instrucciones una tras otras. En puntos determinados, esta secuencia tiene que romperse por una toma de decisión o por cualquier otro motivo. Para ello, el microcontrolador PIC16F84 dispone de varias instrucciones de salto que pasan a describirse a continuación.

### 9.1 SALTOS CONDICIONALES

En el capítulo 6 se describió el funcionamiento de la instrucción de salto incondicional *goto*. El repertorio del PIC16F84 también dispone de instrucciones de **salto condicional**, que son aquéllas que producen un salto en función de que se cumpla o no una condición. Estas instrucciones son el único medio para realizar bifurcaciones condicionales en un programa.

El repertorio de instrucciones del PIC16F84 incluye cuatro instrucciones de salto condicional clasificadas en dos grupos:

- Aquéllas que pueden producir el salto en función del estado de un bit. Son *btfsc* y *btfss*
- Aquéllas que pueden producir el salto en función del contenido de un registro distinto de cero. Son *decfsz* e *incfsz*.

## 9.2 SALTOS EN FUNCIÓN DE UN BIT

Son muy poderosas ya que permiten al programa tomar decisiones en función del estado de un bit de cualquier registro o puerto de entrada/salida. Hay dos instrucciones de este tipo:

### 9.2.1 Instrucción “btfsC f,b”

(*Bit Test f, Skip if Clear*). Esta instrucción puede actuar de dos formas:

- Si el bit número ‘b’ del registro ‘f’ es “1” la instrucción que sigue a ésta se ejecuta normalmente.
- Si el bit número ‘b’ del registro ‘f’ es “0” la instrucción que sigue a ésta se ignora y se salta.

Ejemplo:

Aquí	btfsC	Flag,1	; Si el bit 1 del registro Flag es “0” salta.
Falso	goto	ProcesoX	; Ha sido “1”.
Verdad	...		; Ha sido “0”.

...

Antes instrucción: (PC) = Dirección de “Aquí”.

Después instrucción: Si el bit 1 del registro Flag = 0, (PC) = Dirección de “Verdad”.  
Si el bit 1 del registro Flag = 1, (PC) = Dirección de “Falso”.

### 9.2.2 Instrucción “btfsS f,b”

(*Bit Test f, skip if Set*). Esta instrucción actúa de forma contraria a la instrucción anterior:

- Si el bit número ‘b’ del registro ‘f’ es “0” la instrucción que sigue a ésta se ejecuta normalmente.
- Si el bit número ‘b’ del registro ‘f’ es “1” la instrucción que sigue a ésta se ignora y se salta.

Ejemplo:

Aquí	btfsS	Flag,4	; Si el bit 4 del registro Flag es “1” salta.
Falso	goto	ProcesoX	; Ha sido “0”.
Verdad	...		; Ha sido “1”.

...

Antes instrucción: (PC) = Dirección de “Aquí”.

Después instrucción: Si el bit 4 del registro Flag = 1, (PC) = Dirección de “Verdad”.  
Si el bit 4 del registro Flag = 0, (PC) = Dirección de “Falso”.

## 9.3 SALTOS EN FUNCIÓN DE UN REGISTRO

Las instrucciones de salto condicional “*decfsz f,d*” e “*incfsz f,d*” pueden producir el salto en función del contenido de un registro distinto de cero y son casos especiales de las de incremento y decremento de un registro analizadas en el capítulo 8. Estas instrucciones podrían categorizarse dentro del grupo de instrucciones aritméticas ya que efectivamente operan de forma aritmética (decrementando o incrementando) sobre los registros. Pero, a diferencia de las otras, además pueden alterar el flujo lineal del programa y por eso se las incluye en este grupo. Su forma de actuar se describe a continuación:

### 9.3.1 Instrucción “*decfsz f,d*”

(*Decrement f, Skip if 0*). Esta instrucción decremente en una unidad el contenido del registro ‘f’. Almacena el resultado en W si ‘d’ = 0 (en cuyo caso ‘f’ no varía) y en el registro ‘f’ si ‘d’ = 1. Después de decrementar, pueden ocurrir dos casos:

- Si el resultado es distinto de cero la instrucción que sigue a esta se ejecuta normalmente.
- Si el resultado es cero la instrucción que sigue a esta se ignora y se salta.

Ejemplo:

Aquí	<i>decfsz</i>	Contador,F
	<i>goto</i>	NoEsCero
EsCero	....	
	....	
	....	

Antes instrucción:	(PC) = Dirección de “Aquí”.
Después instrucción:	(Contador) = (Contador) – 1 y además:
	- Si (Contador) = 0, (PC) = Dirección de “EsCero”.
	- Si (Contador) ≠ 0, (PC) = Dirección de “Aquí”+1 .

### 9.3.2 Instrucción “*incfsz f,d*”

(*Increment f, Skip if 0*). Esta instrucción incremente en una unidad el contenido del registro ‘f’. Almacena el resultado en W si ‘d’ = 0 (en cuyo caso ‘f’ no varía) y en el registro ‘f’ si ‘d’ = 1. Después de incrementar, pueden ocurrir dos casos:

- Si el resultado es distinto de cero la instrucción que sigue a esta se ejecuta normalmente.

- Si el resultado es cero (porque al incrementarse se ha desbordado y ha pasado del número b'11111111' al b'00000000') la instrucción que sigue a esta se ignora y se salta.

Ejemplo:

Aqui	incfsz	Contador,F
	goto	Bucle
Continua	...	
	...	

Antes instrucción: (PC) = Dirección de "Aqui".

Después instrucción: (Contador) = (Contador) + 1 y además:

- Si Contador = 0 (256), (PC) = Dirección de "Continua".
- Si Contador  $\neq$  0, (PC) = Dirección de "Aqui"+1.

## 9.4 COMPARACIÓN DE REGISTROS

Una de las más significativa aplicaciones de los saltos condicionales es la comparación entre registros:

### 9.4.1 Comprobar que un registro vale 0

Para saber si un registro vale 0 hay que cargar el registro sobre sí mismo mediante la instrucción "movf" (que es la única que posiciona flags), con lo cual se logra posicionar el flag Z sin variar su contenido. Ejemplo:

EsCero	movf	Registro,F	; (Registro) $\rightarrow$ (Registro) y posiciona flag Z.
	btfss	STATUS,Z	; ¿Es cero?, ¿Z=1?
	goto	NoEsCero	; No.
	...		; Sí es cero y ejecuta la parte del programa correspondiente.
NoEsCero	...		
	...		; No es cero y ejecuta esta parte del programa.

### 9.4.2 Comprobar igualdad entre dos registros

Por ejemplo para comprobar si el contenido de los registros Registro1 y Registro2 son iguales el programa a realizar sería:

Nolguales	movf	Registro1,W	; W se carga con el valor de Registro1.
	subwf	Registro2,W	; (Registro2) - (Registro1) $\rightarrow$ (W) y además posiciona flag Z.
	btfsc	STATUS,Z	; Si Z = 0, salta ya que no son iguales.
	goto	Iguales	; Sí, son iguales y salta a ejecutar lo correspondiente.

Iguales

El  
instrucci

9.4.3

En  
cualesqu

El

"MayorIg  
(RegistroA

mo  
su  
bt  
go  
Mayorigual  
...

Menor

9.4.4 Pr

En e  
será el esqu

lo y ha pasado del  
a esta se ignora y

Iguales

En lugar de la instrucción `subwf Registro2,W` se podría haber utilizado la instrucción `xorwf Registro2,W` con idéntico resultado.

#### 9.4.3 Comprobar que un registro es mayor o menor que otro

En este caso hay que realizar la resta de ambos. Así siendo A y B dos registros cualesquiera y haciendo la operación  $(A-B)$  se tienen las siguientes posibilidades:

Operación (A - B)	Resultado	Bits de Carry y Zero
A > B	Positivo	C = 1 y Z = 0
A < B	Negativo	C = 0 y Z = 0
A = B	Cero	C = 1 y Z = 1

Tabla 9-1 Resultado de la comparación de dos registros

El siguiente fragmento de programa proporciona un salto a la etiqueta "MayorIgual" cuando el (RegistroA)  $\geq$  (RegistroB) y salta a "Menor" cuando el (RegistroA)  $<$  (RegistroB):

movf	RegistroB,W	; (RegistroB) → (W).
subwf	RegistroA,W	; (RegistroA) - (RegistroB) → (W).
btfs	STATUS,C	; $C=1?$ , $\{\text{resultado positivo}\}$ , $\{(RegistroA) \geq (RegistroB)\}$
goto	Menor	
MayorIgual		; C=1. Entonces ha sido $(RegistroA) \geq (RegistroB)$ .
	.....	
	.....	
Menor		; C=0. Entonces ha sido $(RegistroA) < (RegistroB)$ .

#### 9.4.4 Programa ejemplo

En el siguiente programa ejemplo se aplican estos conceptos. El hardware utilizado será el esquema de la figura 9-1 y su explicación gráfica en la figura 9-4.

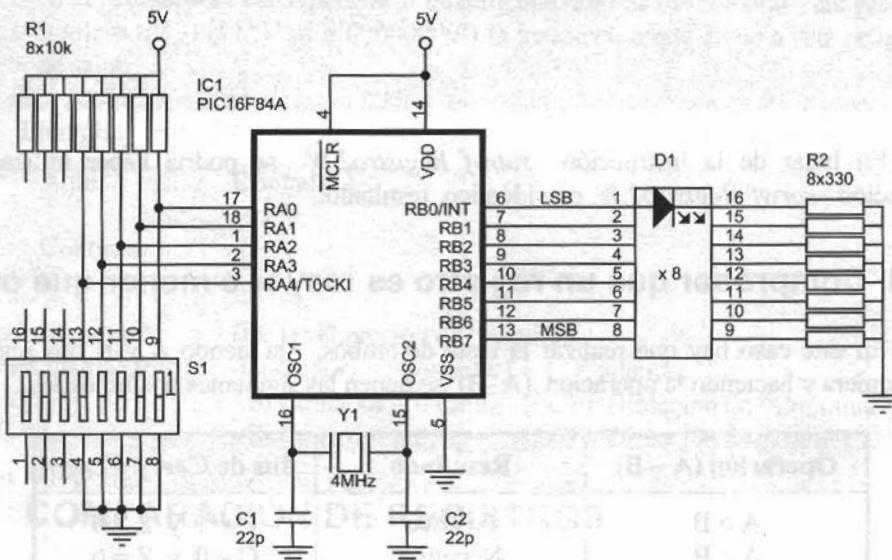


Figura 9-1 Circuito para comprobar el programa Salto\_05.asm

```
***** Salto_05.asm *****
; Compara el dato del puerto de entrada PORTA con un "Número". Tres posibilidades:
; - Si (PORTA) = Número se encienden todos los LEDs de salida.
; - Si (PORTA) > Número se activan los LEDs pares de salida.
; - Si (PORTA) < Número se encienden los LEDs del nibble alto y se apagan los del bajo.

; Hay que destacar que al no haber instrucciones de comparación, estas se realizan
; mediante restas.

; ZONA DE DATOS *****
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A           ; Procesador utilizado.
INCLUDE <PI16F84A.INC>     ; Fichero donde se definen las etiquetas del PIC.

Número EQU    d'13'          ; Por ejemplo, este número a comparar.

; ZONA DE CÓDIGOS *****
ORG      0                  ; El programa comienza en la dirección 0.

Inicio
    bsf    STATUS,RP0        ; Acceso al Banco 1.
    clrf   TRISB            ; Las líneas del Puerto B se configuran como salida.
    movlw  b'00011111'       ; Las 5 líneas del Puerto A se configuran como entrada.
    movwf  TRISA
    bcf    STATUS,RP0        ; Acceso al Banco 0.

Principal
    movlw  Número           ; Carga el número a comparar.
```

## 9.5 LAZOS

Otra  
Estos son los principales

A) LAZO

## 9.5.1 Lazo

Es un  
lazo de repetición  
instrucción

Principal

...

goteo

R2  
8x330

```

subwf PORTA,W           ; (PORTA) - Numero --> (W)
movlw b'11110000'         ; Supone (PORTA) es menor.
btfsf STATUS,C           ; ¿C=1?, ¿(W) positivo?, ¿(PORTA) >= Numero?
goto ActivaSalida        ; No. C=0, por tanto (PORTA) < Numero.
movlw b'11111111'         ; Supone que son iguales.
btfsf STATUS,Z           ; ¿Z=0?, ¿son distintos?
goto ActivaSalida        ; No. Son iguales ya que Z = 1.
movlw b'01010101'         ; Sí, por tanto (PORTA) > Numero.

ActivaSalida
movwf PORTB
goto Principal

END                      ; Resultado se visualiza por el puerto de salida.
                           ; Crea un bucle cerrado e infinito.

                           ; Fin del programa.

```

## 9.5 LAZOS O BUCLES

Otra aplicación muy importante de los saltos condicionales son los **lazos o bucles**. Estos son fragmentos de programa que se repiten un número finito o infinito de veces. Los principales tipos están descritos en la figura 9-2.

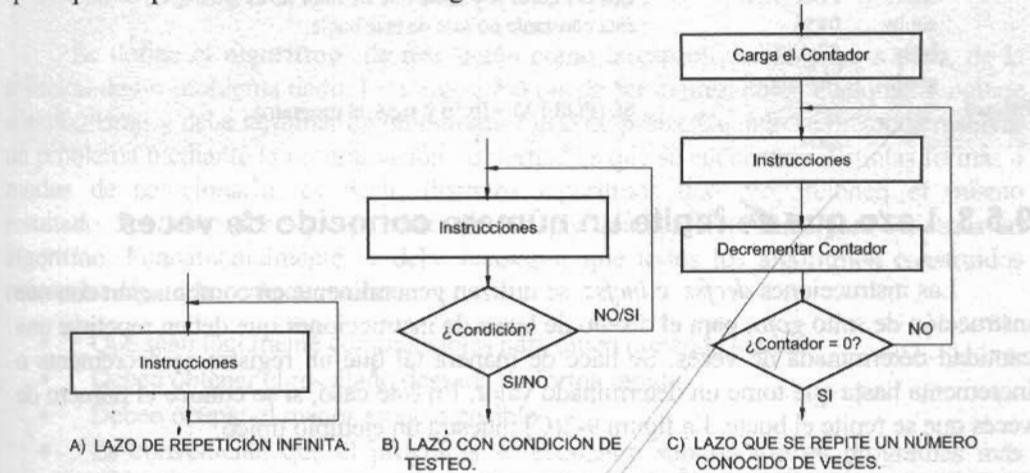


Figura 9-2 Tipos principales de lazos o bucles

### 9.5.1 Lazo de repetición infinita

Es un salto incondicional a una posición anterior del programa conformando un lazo de repetición infinita, sin posibilidad de tomar otro camino, figura 9-2(A). Utiliza la instrucción *goto*. Ejemplo:

```

Principal
...
...
goto Principal

```

## 9.5.2 Lazo con condición de testeo

Se utiliza una instrucción de testeo para controlar la ejecución del bucle. Para este caso la repetición del lazo es finita, pero no se puede precisar el número de veces que se repite, figura 9-2(B).

Ejemplo 1:

EsperaUno	....	
btfss	PORTE,A	; Lee el pin 4 del Puerto A y hasta que no se pone a 1 no sale
goto	EsperaUno	; de este bucle.
	....	

Ejemplo 2:

NoEsIgual	....	
....		
movf	PORTE,W	; Lee el Puerto A y hasta que su valor no es igual al de
sublw	0x56	; esta constante no sale de este bucle.
btfss	STATUS,Z	
goto	NoEsIgual	
EsIgual	....	; Si, (PORTE) = 0x56 y sigue el programa.
	....	

## 9.5.3 Lazo que se repite un número conocido de veces

Las instrucciones *decfsz* e *incfsz* se utilizan generalmente en combinación con una instrucción de salto *goto*, para el diseño de lazos de instrucciones que deben repetirse una cantidad determinada de veces. Se hace de manera tal que un registro se decremente o incremente hasta que tome un determinado valor. En este caso, si se conoce el número de veces que se repite el bucle. La figura 9-2(C) muestra un ejemplo típico.

Lazo	....	; (Instrucciones precedentes del programa).	
	movlw	NumeroVeces	; Este será el número de veces que se repetirá el lazo.
	movwf	Contador	; Carga una posición de la memoria de datos interna
	....	; Lo que sigue se ejecutará "NumeroVeces" veces	
	....	; Aquí pueden ir otras instrucciones que se ejecutarán tantas	
	decfsz	Contador,F	; veces como se haya cargado el Contador.
	goto	Lazo	; Se decrementa el Contador hasta que llegue a 0.
	....	; Si no llega a cero repite el lazo.	
	....	; Cuando llegue a cero no ejecuta el "goto" anterior y sigue la	
	....	; ejecución del programa.	

## 9.6 PROGRAMACIÓN

Después de ver los tipos de bucles y las condiciones de control, es importante comprender que los microprocesadores tienen una amplia gama de instrucciones que permiten construir bucles complejos y eficientes. Los siguientes apartados describen las principales técnicas y estrategias para programar bucles en PIC.

El proceso de programación es similar a la construcción de un edificio. Se deben tener en cuenta las reglas básicas y las estrategias para construir un bucle eficiente. Durante el proceso de programación, es importante seguir las mejores prácticas y seguir las mejores prácticas.

El objetivo es escribir código que sea correcto y eficiente. Es importante tener en cuenta que el código debe ser fácil de modificar y mantener. Siempre es mejor renunciar a una solución simple y elegante por una más compleja y difícil de modificar.

Se define una solución de un problema mediante simbolismo y diagramas. Un problema tiene varios modos de solución, cada uno con su resultado. Por tanto, se debe elegir el mejor algoritmo. Fundamentalmente, se reúnan las siguientes reglas:

- Que sean sencillas y fáciles de entender.
- Deben obtener resultados correctos.
- Deben ocupar poco espacio en memoria.
- Es conveniente utilizar operaciones elementales.
- Además de la funcionalidad, se debe tener en cuenta la documentación y la documentación técnica.

Hay diversos métodos de construcción de bucles más sencillos y útiles para resolver problemas gráficamente.

## 9.7 DIAGRAMAS DE FLUJO

Un diagrama de flujo es una herramienta útil para la resolución de un problema.

## 9.6 PROGRAMACIÓN Y ALGORITMO

Después de haber tratado y examinado el repertorio de instrucciones básicas del microprocesador PIC16F84 es conveniente introducir algunas cuestiones primordiales sobre la metodología de la programación que faciliten el desarrollo de programas aplicables a sistemas físicos.

El proceso de programación debe desarrollarse partiendo de unas ideas que deben ser perfectamente asimiladas, comprendidas y organizadas por la persona o personas que vayan a construir el programa en cuestión. Esto lleva implícita la disminución de errores durante el proceso programador, por lo que se facilita enormemente la tarea.

El objetivo final de la programación es conseguir programas que funcionen correctamente y sean eficientes y además proporcionar la documentación necesaria sobre ellos. Debe tenerse en cuenta que un programa indebidamente documentado sería muy difícil de modificar en un futuro, o al menos podría resultar tan costoso que haría renunciar a tal propósito.

Se define el **algoritmo** de resolución como la especificación paso a paso, de la solución de un problema dado. Este algoritmo puede ser expresado en cualquier lenguaje o simbolismo y debe terminar en un número finito de pasos. Cuando se pretende resolver un problema mediante la programación, lo normal es que se encuentren distintas formas o modos de solucionarlo, es decir, distintos algoritmos que proporcionen el mismo resultado. Por tanto debe quedar claro que no hay una solución única para construir un algoritmo. Fundamentalmente se debe conseguir que todos los algoritmos construidos reúnan las siguientes características:

- Que sean fácilmente comprensibles para quien pretenda leerlos.
- Deben obtener el resultado deseado de forma rápida.
- Deben ocupar el menor espacio posible.
- Es conveniente que el programa se encuentre subdividido en programas más elementales, es decir, el programa debe ser modular y estar “estructurado”.
- Además debe ser razonablemente fácil de modificar. A ésto ayuda la modularidad y la documentación que acompañe al programa.

Hay diversas formas de abordar la construcción de un algoritmo. Cada uno de estos métodos de construcción de algoritmos tiene sus ventajas e inconvenientes. Uno de los más sencillos y utilizados es la realización de los diagramas de flujo que resuelven el problema gráficamente.

## 9.7 DIAGRAMAS DE FLUJO

Un **diagrama de flujo** o *flowchart* es la representación gráfica de un algoritmo para la resolución de un programa. Indica el orden en que las operaciones son llevadas a cabo y

las decisiones que determinan esta secuencia. También recibe el nombre de organigrama, ordinograma, fluojograma y otros.

Los diagramas de flujo están constituidos por una serie de símbolos que contienen los pasos del algoritmo. Estos símbolos van unidos entre sí por flechas que indican el sentido de la evolución del programa. Sobre ellos se puede hacer las anotaciones que se consideren precisas. Los símbolos más importantes empleados en la representación de los diagramas de flujo se muestran en la figura 9-3.

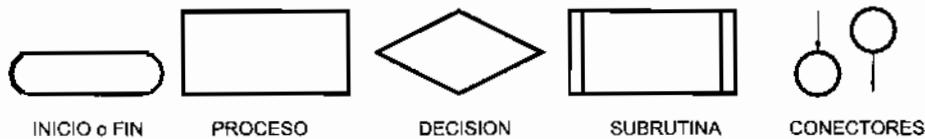


Figura 9-3 Símbolos básicos en los diagramas de flujo

- **Inicio o Fin.** Un óvalo o elipse representa el inicio o el final de la secuencia de operaciones.
- **Proceso.** Un rectángulo representa una operación de proceso, que son la mayor parte de las operaciones realizadas en un programa.
- **Decisión.** Un rombo representa una decisión que da lugar a una transferencia condicional de control. Produce una bifurcación entre dos caminos posibles, dependiendo de que la respuesta a la pregunta realizada sea SI o NO.
- **Subrutinas.** Este símbolo representa un conjunto de operaciones cuyo empleo se repite varias veces durante el programa. Se explica en profundidad en el próximo capítulo.
- **Conectores.** Indican el camino que sigue un programa muy largo que no cabe en una hoja. Como es muy importante mantener la claridad y el orden en los diagramas de flujo, estos conectores indican los puntos de ruptura y reanudación de la trayectoria de flujo, con el fin de evitar el cruce de líneas o permitir la continuación del diagrama en una hoja distinta. Los conectores se suelen representar con círculos identificados mediante un número o carácter, que sirve para identificar los conectores que enlazan una misma trayectoria de flujo.
- **Líneas y flechas.** Los símbolos están conectados por líneas sólidas con puntas de flecha que indican el camino seguido por un programa. Nunca se deben cortar entre sí.

Mientras sea posible, el flujo debe ser de arriba hacia abajo y de izquierda a derecha. Los enunciados que indican las operaciones o decisiones asociadas con cada símbolo aparecen dentro de él. Estas frases no precisan ninguna formalidad, aunque se recomienda que sean suficientemente expresivas.

e organigrama,  
que contienen el  
que indican el  
acciones que se  
entación de los



CONECTORES

la secuencia de

que son la mayor

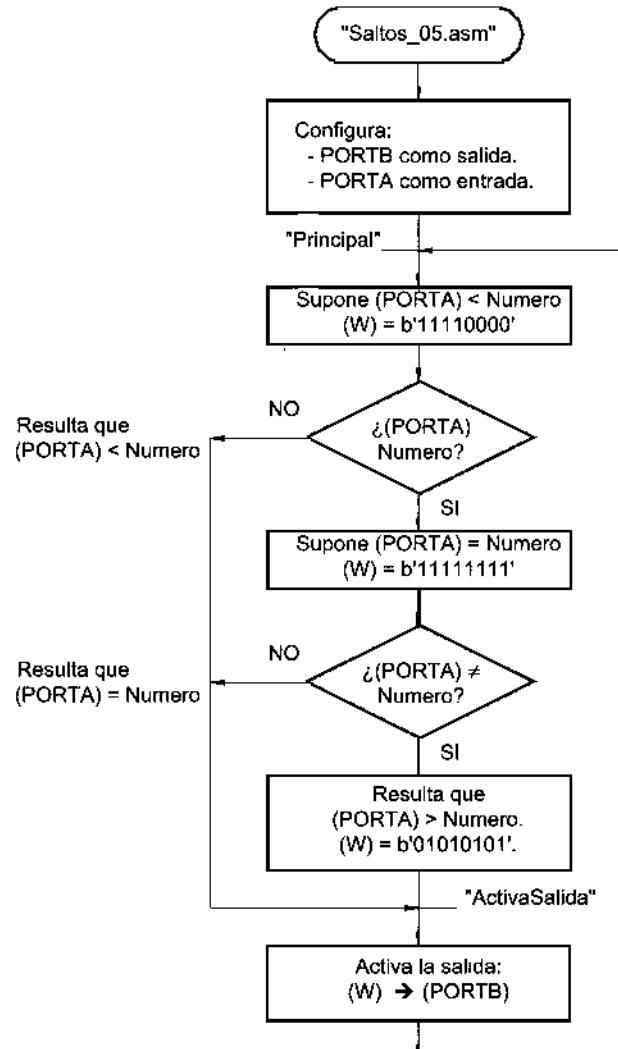
una transferencia  
caminos posibles,  
NO.cuyo empleo se  
ad en el próximoo que no cabe en  
el orden en los  
ra y reanudación  
reas o permitir la  
ctores se suelen  
racter, que sirve  
a de flujo.das con puntas de  
a se deben cortary de izquierda a  
ociadas con cada  
alidad, aunque se

Figura 9-4 Diagrama de flujo del programa Saltos\_05.asm

El diagrama de flujo es el equivalente en software de lo que es el diagrama de bloque en hardware. Sus finalidades son:

- Simplificar la codificación del algoritmo en el lenguaje particular del microcontrolador.
- Facilitar la comprensión del algoritmo por otras personas.

Los diagramas de flujo son herramientas que permiten “visualizar” el desarrollo de un programa. Emplear los diagramas de flujo para la construcción de algoritmos puede

resultar muy ilustrativo, tal como se muestra en el ejemplo descrito en la figura 9-4 que presenta la resolución del programa Saltos\_05.asm descrito anteriormente.

## 9.8 MÁS DIRECTIVAS IMPORTANTES

En el capítulo 6 se explicaron las directivas más importantes. A continuación se exponen otras directivas muy utilizadas: *CBLOCK*, *ENDC* y *#DEFINE*.

### 9.8.1 CBLOCK y ENDC

En la mayoría de las aplicaciones el propósito de las directivas *CBLOCK* (*Define a Block of Constants*) y *ENDC* (*End Constant Block*) es asignar direcciones (generalmente de memoria RAM de datos) a muchas etiquetas. La lista queda enmarcada entre las directivas *CBLOCK* y *ENDC*.

Un ejemplo típico de utilización:

CBLOCK	0x0C ; Las variables se posicionan a partir de esta posición de RAM.
Centenas	, La variable "Centenas" ocupa la posición 0x0C de RAM.
Decenas	, La variable "Decenas" ocupa la posición 0x0D de RAM.
Unidades	, La variable "Unidades" ocupa la posición 0x0E de RAM.
ENDC	

El valor que acompaña a la directiva *CBLOCK* (0x0C en el anterior ejemplo), indica el valor de arranque para el primer nombre del bloque de etiquetas. Si este valor no es encontrado, la primera constante recibirá el valor inmediatamente superior al de la última constante del *CBLOCK* anteriormente definido. Por ejemplo, si a lo largo del programa del ejemplo anterior se encontrase la siguiente definición de variables, ocuparían las posiciones que se indican.

CBLOCK	; Las variables se posicionan a partir de la posición de RAM definida por el último bloque CBLOCK.
Operando	, La variable "Operando" ocupa la posición 0x0F de RAM.
Resultado	, La variable "Resultado" ocupa la posición 0x10 de RAM.
Auxiliar	, La variable "Auxiliar" ocupa la posición 0x11 de RAM.
ENDC	

Si el primer *CBLOCK* en el archivo fuente no define ningún comienzo los valores asignados empiezan con el cero, que corresponde a la zona SFR de la RAM de datos, normalmente no es correcto ya que sobrescribiría los datos de esta zona y el programa no funcionaría correctamente.

### 9.8.2 #DEFINE

(*Define a Text Substitution Label*). Su sintaxis es:

Esta directiva se encuentre, en

Ejemplo 1

```
#DEFINE LED
```

```
bsf
```

Ejemplo 2

```
#DEFINE Banco
```

```
#DEFINE Banco
```

```
#DEFINE LED
```

Inicio

```
Banco1
```

```
bcf
```

```
Banco0
```

```
bsf
```

```
LED
```

Esta directiva este método no es

## 9.9 CONVERSIÓN

La conversión de las operaciones analizada en detalle 01111100, para quedando: 0001 00 conversión. Un pro que se puede comp

```
*****  
;  
; Un número binario de  
; de memorias llamadas:  
; nibble bajo del registro  
; puerto de salida se visu  
;  
; El máximo número a c  
; número binario de entr  
;  
; El procedimiento utiliz  
; ejemplo que trata de la
```

en la figura 9-4 que te.

A continuación se

BLOCK (*Define* a  
nes (generalmente  
marcada entre las

RAM.  
L.  
I.  
M.

anterior ejemplo).  
is. Si este valor no  
superior al: de la  
si a lo largo del  
ión de variables,

definida

4.  
4.

nienzo los valores  
a RAM de datos.  
y el programa no

#DEFINE <name> [<string>]

Esta directiva define una cadena de sustitución de texto. Donde quiera que <name> se encuentre, en el ensamblador se sustituirá por <string>.

Ejemplo 1:

```
#DEFINE LED PORTB,4 ; El LED se conecta en esta línea.  
...  
bsf LED ; Enciende el LED.
```

Ejemplo 2:

```
#DEFINE Banco0 bcf STATUS,RP0 ; Acceso al Banco 0.  
#DEFINE Banco1 bsf STATUS,RP0 ; Acceso al Banco 1.  
#DEFINE LED PORTB,4 ; El LED se conecta en esta línea.
```

Inicio

```
...  
Banco1  
bcf LED ; Acceso al Banco 1.  
Banco0  
bsf LED ; Configura esta línea como salida.  
...  
; Acceso al Banco 0.  
; Enciende el diodo LED
```

Esta directiva emula #DEFINE del ANSI C standard. Los símbolos definidos con este método no están disponibles para ser usados por el MPLAB.

## 9.9 CONVERSIÓN DE BINARIO NATURAL A BCD

La conversión de un numero expresado en binario natural a formato BCD es una de las operaciones más utilizadas en los programas con microcontrolador y que merece ser analizada en detalle. Por ejemplo el valor 124 expresado en binario natural sería 01111100, para expresarlo en BCD hay que separar las centenas, decenas y unidades quedando: 0001 0010 0100. La figura 9-5 explica el diagrama de flujo para resolver esta conversión. Un programa ejemplo que lo implementa, sería el descrito a continuación y que se puede comprobar sobre el circuito de la figura 9-6.

\*\*\*\*\* BCD\_01.asm \*\*\*\*\*

. Un número binario de 8 bits es convertido a BCD. El resultado se guarda en tres posiciones de memoria llamadas Centenas, Decenas y Unidades. Además al final las unidades estarán en el nibble bajo del registro W y las decenas en el nibble alto. En los diodos LEDs conectados al puerto de salida se visualizarán las decenas y las unidades.

: El máximo número a convertir será el 255 que es el máximo valor que puede adquirir el número binario de entrada de 8 bits.

: El procedimiento utilizado es mediante restas de 10 tal como se explica en el siguiente ejemplo que trata de la conversión del número 124 a BCD:

(Centenas)	(Decenas)	(Unidades)	$\{(\text{Unidades}) < 10\}$	$\{(\text{Decenas}) = 10\}$
0	0	124	NO, resta 10	Incrementa (Decenas).
0	1	114	NO, resta 10	NO. Incrementa (Decenas).
0	2	104	NO, resta 10	NO. Incrementa (Decenas).
0	3	94	NO, resta 10	NO. Incrementa (Decenas).
0	4	84	NO, resta 10	NO. Incrementa (Decenas).
0	5	74	NO, resta 10	NO. Incrementa (Decenas).
0	6	64	NO, resta 10	NO. Incrementa (Decenas).
0	7	54	NO, resta 10	NO. Incrementa (Decenas).
0	8	44	NO, resta 10	NO. Incrementa (Decenas).
0	9	34	NO, resta 10	NO. Incrementa (Decenas).
1	0	24	NO, resta 10	Si. $\{(\text{Decenas}) = 0\}$ , y además incrementa (Centenas)
1	1	14	NO, resta 10	NO. Incrementa (Decenas)
1	2	4	SÍ, se acabó.	

; El número a convertir será la constante "Número".

; ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

CBLOCK 0x0C

Centenas

Decenas

Unidades

ENDC

Número EQU .124

; Por ejemplo.

; ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG 0
Inicio
    bcf STATUS,RP0
    clrf TRISB
    bcf STATUS,RP0

Principal
    clrf Centenas
    clrf Decenas
    movlw Número
    movwf Unidades

BCD_Resta0
    movlw .10
    subwf Unidades,W
    btfss STATUS,C
    goto BIN_BCD_Fin

BCD_IncrementaDecenas
```

; El programa comienza en la dirección 0.

; Acceso al Banco 1. Las líneas del Puerto B se configuran como salida.

; Acceso al Banco 0.

Carga los registros con el resultado inicial.

; En principio  $\{(\text{Centenas}) = 0\}$  y  $\{(\text{Decenas}) = 0\}$ .

Se carga el número binario a convertir.

A las unidades se les va restando 10 en cada pasada  $\{(\text{W}) = (\text{Unidades}) - 10\}$ .

$\{\text{C}\} = ?$ ,  $\{\text{W}\}$  positivo?,  $\{(\text{Unidades}) \geq 10\}$ ?

No, es menor de 10. Se acabó.

movwf  
inef  
movlw  
subwf  
btfs  
goto  
BCD\_IncrementaC  
clrf  
incf  
goto  
BIN\_BCD\_Fin  
swapf  
addwf  
movwf  
sleep  
END

Figura 9-5 Diagrama de flujo de la conversión de binario a BCD.

```

movwf  Unidades      ; Recupera lo que queda por restar.
incf   Decenas,F     ; Incrementa las decenas y comprueba si llega a
movlw   .10           ; 10. Lo hace mediante una resta.
subwf   Decenas,W     ; (W)= (Decenas)-10.
btfs   STATUS,C       ; ¿(C)=1?; ¿(W) positivo?, ¿(Decenas)>=10?
goto   BCD_Resta10    ; No. Vuelve a dar otra pasada, restándole 10.

BCD_IncrementaCentenas
    clrf   Decenas      ; Pone a cero las decenas
    incf   Centenas,F   ; e incrementa las centenas.
    goto   BCD_Resta10    ; Otra pasada, resta 10 al número a convertir.

BIN_BCD_Fin
    swapf  Decenas,W     ; En el nibble alto de W también las decenas.
    addwf  Unidades,W    ; En el nibble bajo de W las unidades.
    movwf  PORTB          ; Se visualiza por el puerto de salida.
    sleep                         ; Se queda permanentemente en reposo.

END                                ; Fin del programa.

```

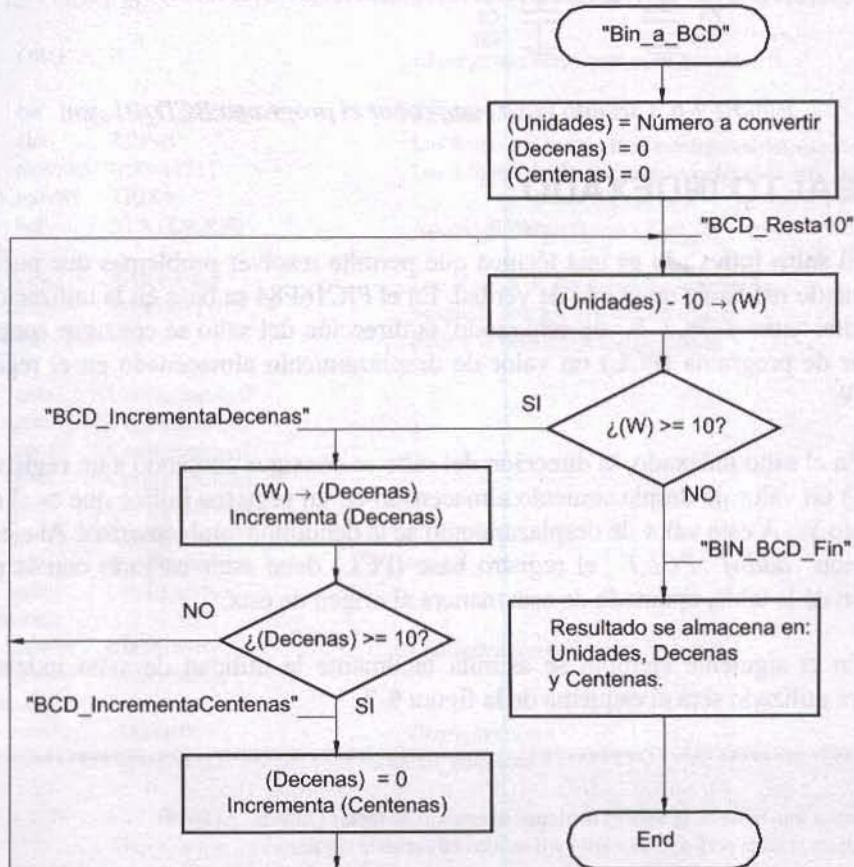


Figura 9-5 Diagrama de flujo para la conversión de un número binario natural a BCD

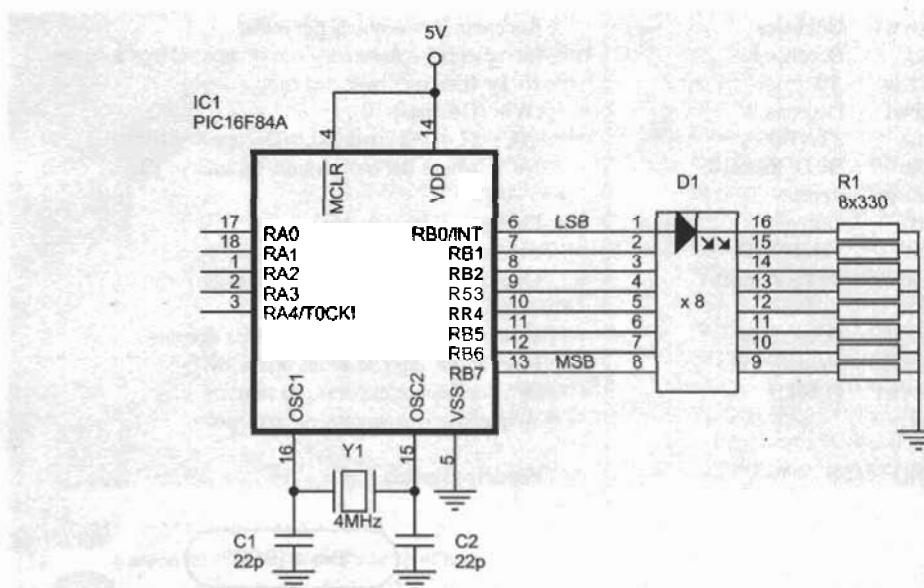


Figura 9-6 Circuito para comprobar el programa BCD\_01.asm

## 9.10 SALTO INDEXADO

El salto indexado es una técnica que permite resolver problemas que puedan ser representado mediante una tabla de verdad. En el PIC16F84 se basa en la utilización de la instrucción `addwf PCL,F`, de este modo, la dirección del salto se consigue sumando al contador de programa (PCL) un valor de desplazamiento almacenado en el registro de trabajo W.

En el salto indexado, la dirección del salto se consigue sumando a un **registro base** (el PCL) un valor de desplazamiento almacenado en un **registro índice** que es el registro de trabajo W. A este valor de desplazamiento se le denomina también *offset*. Al ejecutar la instrucción `addwf PCL,F` el **registro base** (PCL) debe estar cargado con la primera dirección de la tabla, apuntado de esta manera al origen de ésta.

En el siguiente ejemplo se asimila fácilmente la utilidad de salto indexado. El hardware utilizado será el esquema de la figura 9-7.

```
;***** Indexado_01.asm *****
```

; Implementar una tabla de la verdad mediante el manejo de tablas grabadas en ROM.  
; Por ejemplo, la tabla será de 3 entradas y 6 salidas tal como la siguiente:

C	B	A	S5	S4	S3	S2	S1	S0
0	0	0	0	0	1	0	1	0

0 0 0 | 0 0 1 0 1 0; (Configuración 0).

; Las entradas C, B, A  
; Las salidas se obtienen  
; RB5 (S5), RB4 (S4)

; ZONA DE DATOS

CONF1  
LIST  
INCLUDE

; ZONA DE CÓDIGO

ORG	CODIGO
Inicio	bsf S0 clrf T0 movlw b0 movwf 30 bef S0
Principal	movf P0 andlw b0 addwf P0
Tabla	goto C0 goto C1 goto C2 goto C3 goto C4 goto C5 goto C6 goto C7

Configuracion0	movlw b'00000000' goto A0
Configuracion1	movlw b'00000001' goto A1
Configuracion2	movlw b'00000010' goto A2
Configuracion3	movlw b'00000011' goto A3
Configuracion4	movlw b'00000100' goto A4

```

; 0 0 1 | a 0 1 0 0 1 ;(Configuración 1).
; 0 1 0 | 1 0 0 0 1 1 ;(Configuración 2).
; 0 1 1 | 0 0 1 1 1 1 ;(Configuración 3).
; 1 0 0 | 1 0 0 0 0 0 ;(Configuración 4).
; 1 0 1 | 0 0 0 1 1 1 ;(Configuración 5).
; 1 1 0 | 0 1 0 1 1 1 ;(Configuración 6).
; 1 1 1 | 1 1 1 1 1 1 ;(Configuración 7).

```

; Las entradas C, B, A se conectarán a las líneas del puerto A: RA2(C), RA1(B) y RA0(A).

; Las salidas se obtienen en el puerto B:

; RB5(S5), RB4(S4), RB3(S3), RB2(S2), RB1(S1) y RB0(S0).

; ZONA DE DATOS \*\*\*\*\*

```

CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

```

; ZONA DE CÓDIGOS \*\*\*\*\*

Inicio	ORG 0	; El programa comienza en la dirección 0.
	bsf STATUS,RP0	; Acceso al Banco 1.
	clrf TRISB	; Las líneas del Puerto B se configuran como salida.
	movlw b'00011111'	; Las 5 líneas del Puerto A se configuran como entrada.
	movwf TRISA	
	bcf STATUS,RP0	; Acceso al Banco 0.
Principal	movf PORTA,W	; Leo el valor de las variables de entrada.
	andlw b'00000111'	; Se queda con los tres bits de entrada.
	addwf PCL,F	; Salta a la configuración adecuada.
Tabla	goto Configuracion0	
	goto Configuracion1	
	goto Configuracion2	
	goto Configuracion3	
	goto Configuracion4	
	goto Configuracion5	
	goto Configuracion6	
	goto Configuracion7	
Configuracion0	movlw b'00001010'	; Configuración 0.
	goto ActivaSalida	
Configuracion1	movlw b'00001001'	; Configuración 1.
	goto ActivaSalida	
Configuracion2	movlw b'00100011'	; Configuración 2,
	goto ActivaSalida	
Configuracion3	movlw b'00001111'	; Configuración 3.
	goto ActivaSalida	
Configuracion4		

```

        movlw b'00100000'          ; Configuración 4.
        goto ActivaSalida
Configuracion5
        movlw b'00000111'          ; Configuración 5.
        goto ActivaSalida
Configuracion6
        movlw b'00010111'          ; Configuración 6.
        goto ActivaSalida
Configuracion7
        movlw b'00111111'          ; Configuración 7.
ActivaSalida
        movwf PORTB               ; Visualiza por el puerto de salida
        goto Principal
        END
    
```

En las soluciones de los ejercicios facilitadas en el CD-ROM que acompaña a este libro, se describe otra forma de implementación más eficaz mediante el programa Indexado\_01B.asm.

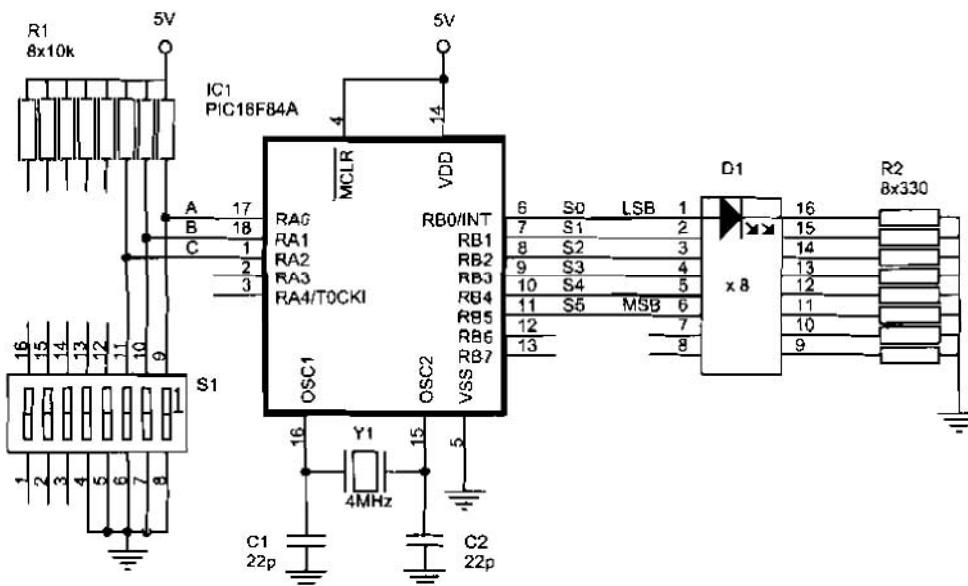


Figura 9-7 Circuito para comprobar los programas de tabla de verdad

## 9.11 SALTO INDEXADO DESCONTROLADO

Como ya se explicó en el tema 4, los 13 bits contenidos en el contador de programa y que direccionan la memoria de código están guardados en dos registros específicos (figura 9-8):

- El reg un reg
- Los bi puede PCLA

Figura 5

El PCL  
bits son transfe  
tiznen por dsci

Cuando  
instrucción ad  
es así, el salto e  
que el registro  
prueba de un sa

;\*\*\*\*\*  
; Programa para cor  
; Se debe comproba  
; ZONA DE DATO

INCLUD  
LIST

:ZONA DE CÓDIGO

ORG	
Inicio	
goto	
clrw	
goto	
ORG	
Principal	
movlw	
addwf	

- El registro **PCL** guarda los 8 bits de menor peso, bits <7:0> del PC. Al tratarse de un registro localizado dentro del SFR (figura 4-1) se puede escribir y leer.
- Los bits <12:8> del PC se alojan en el registro **PCH**, que es un registro que no se puede leer ni escribir directamente. Para acceder a estos bits se utiliza el registro **PCLATH**.

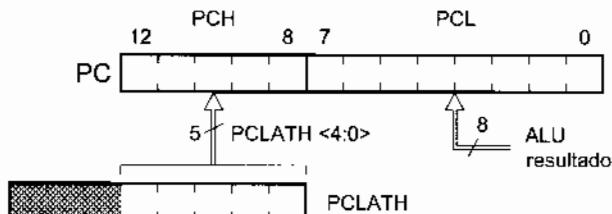


Figura 9-8 Composición del PC en instrucciones con PCL como destino

El **PCLATH** (*PC Latch High*) es un registro del SFR (figura 4-1) cuyos primeros 5 bits son transferidos al registro PCH del contador de programas, en las instrucciones que tienen por destino al PCL, como por ejemplo en la instrucción *addwf PCL,F* (figura 9-8).

Cuando se utiliza el salto indexado hay que asegurarse que al ejecutar el salto de la instrucción *addwf PCL,F*, el registro PCLATH tiene el valor correcto, porque si esto no es así, el salto es impredecible. Esto suele ocurrir cuando el valor de PCL se desborda sin que el registro PCLATH se incremente convenientemente. El siguiente programa es una prueba de un salto indexado descontrolado.

```
***** Indexado_03.asm *****
;
; Programa para comprobar el efecto de un uso incorrecto de la instrucción "addwf PCL,F".
; Se debe comprobar con el simulador del MPLAB.
;
; ZONA DE DATOS *****

INCLUDE <P16F84.INC>
LIST      P=16F84

; ZONA DE CÓDIGOS *****

    ORG      0
    Inicio
        goto    Principal ; Posición 000h de memoria de programa.
        clrw
        goto    Inicio   ; Posición 001h de memoria de programa.
                        ; Posición 002h de memoria de programa.

    Principal
        ORG      0xFE      ; Fija la posición de la memoria de programa en 0x0FE.
        movlw   .1
        addwf   PCL,F     ; Posición 0FEh de memoria de programa.
                        ; Posición 0FFh de memoria de programa.
```

```

goto    Configuracion0 ; Posición 100h de memoria de programa.
goto    Configuracion1 ; Posición 101h de memoria de programa.

```

; La intención de la instrucción "addwf PCL,F" es saltar a la posición de la instrucción  
; "goto Configuracion1", que está en la posición 101h de memoria de programa pero, como el  
; contenido del registro PCLATH no ha cambiado de valor, realmente salta a la posición que  
; indica el contador de programa que en este caso es (PC)=(PCLATH)(PCL) = 0001h, es decir, ha  
; saltado a la posición donde se encuentra la instrucción "clrw". El salto se ha descontrolado.

```

Configuracion0
Configuracion1

END           ; Fin del programa.

```

Una manera muy sencilla de resolver este problema, consiste en no utilizar la instrucción *addwf PCL,F* cuando el salto llega más allá de la dirección OFFh de la memoria de programa o prevea que el PCL se va a desbordar. De todas formas, en la nota de aplicación AN556 del fabricante describe el manejo correcto del registro PCLATH.

## 9.12 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular, grabar el microcontrolador y comprobar los siguientes programas para el esquema de la figura 1-2. El lector puede introducir todas las mejoras que considere conveniente.

**Saltos\_01.asm:** El Puerto B, que actúa como salida es controlado por el bit 0 del Puerto A, que actúa como entrada. De manera tal, que:

- Si el bit 0 del PORTA = 1, se encienden todos los LEDs de salida.
- Si el bit 0 del PORTA = 0, sólo se encienden los LEDs del nibble alto.

**Saltos\_02.asm:** Compara el dato del puerto de entrada PORTA con un número (por ejemplo, el 13). Pueden darse dos posibilidades:

- Si (PORTA) = Numero, se encienden todos los LEDs de salida.
- Si (PORTA) ≠ Numero, se activan los LEDs pares y se apagan los impares.

**Saltos\_03.asm:** Compara el dato introducido por el Puerto A que actúa como entrada con un número. Pueden darse dos posibilidades:

- Si (PORTA) es mayor o igual que "Numero" se encienden todos los LEDs.
- Si (PORTA) es menor que "Numero" se activan los LEDs pares de la salida.

**Saltos\_04.asm:** Compara el dato del puerto de entrada PORTA con un número. Pueden darse dos posibilidades:

- Si (PORTA) es mayor que "Numero" se encienden todos los LEDs de salida.

- Si (PORTA) es menor o igual que "Numero" se activan los LEDs pares.

**Saltos\_05.asm:** Compara el dato del puerto de entrada PORTA con un número. Pueden darse tres posibilidades:

- Si (PORTA) = Numero se encienden todos los LEDs de salida.
- Si (PORTA) > Numero se activan los LEDs pares de salida.
- Si (PORTA) < Numero sólo se encienden los LEDs del nibble alto.

**Saltos\_06.asm:** Lee las tres líneas más bajas del puerto A, que fijan el número de LEDs que se iluminarán a la salida. Así por ejemplo, si lee el dato "--00101" (cinco) en los LEDs conectados al Puerto B se iluminará el código "00011111", encendiéndose cinco diodos LEDs (D4, D3, D2, D1 y D0). Se utilizará la instrucción de rotación *rfl*.

**BCD\_01.asm:** Un número binario de 8 bits es convertido a BCD. El resultado se guarda en tres posiciones de memorias llamadas Centenas, Decenas y Unidades. Además al final las decenas y unidades estarán en el nibble alto y bajo respectivamente del registro W. En los LEDs conectados al puerto de salida se visualizarán las decenas y las unidades. El número a convertir será la constante "Numero" que como máximo tendrá un valor de b'11111111' (255 decimal).

**Indexado\_01.asm:** Implementar una tabla de la verdad mediante el manejo de saltos indexados. La tabla será ideada por el lector con el número de entradas y salidas que desee y que sea físicamente posible implementar por el microcontrolador.

**Indexado\_02.asm:** Diseñar un programa para controlar el nivel del depósito de líquido de la figura 9-9. Utiliza (entre paréntesis las líneas del microcontrolador conectadas):

- Tres sondas detectoras: SV, Sonda de Vacío (RA0); SLL, Sonda de Llenado (RA1); SR, Sonda de Rebose (RA2).
- Dos bombas de agua: B1 (RB5), B2 (RB6).
- Cinco indicadores: Vacío (RB0); Llenandose (RB1); Lleno (RB2); Rebose (RB3); Alarma (RB4).

Su funcionamiento:

- Cuando ninguna de las sondas está mojada se entiende que el depósito está vacío y se accionarán las dos bombas. El indicador "Vacio" se iluminará.
- Cuando el nivel del líquido toque la sonda de vacío "SV" seguirá llenándose el depósito con las dos bombas. El indicador "Llenandose" se ilumina.
- Cuando el nivel del líquido toque la sonda de llenado "SLL" se para la bomba B2, quedando la bomba B1 activada en modo mantenimiento. El indicador "Lleno" se ilumina.

- Si el nivel del líquido moja la sonda de rebose "SR" se apaga también la bomba B1, quedando las dos bombas fuera de servicio. El indicador "Rebose" se enciende.
- Cuando se produce un fallo o mal funcionamiento en las sondas de entrada (por ejemplo que se active la sonda de rebose y no la de vacío) se paran las dos bombas. El indicador "Alarma" se ilumina.

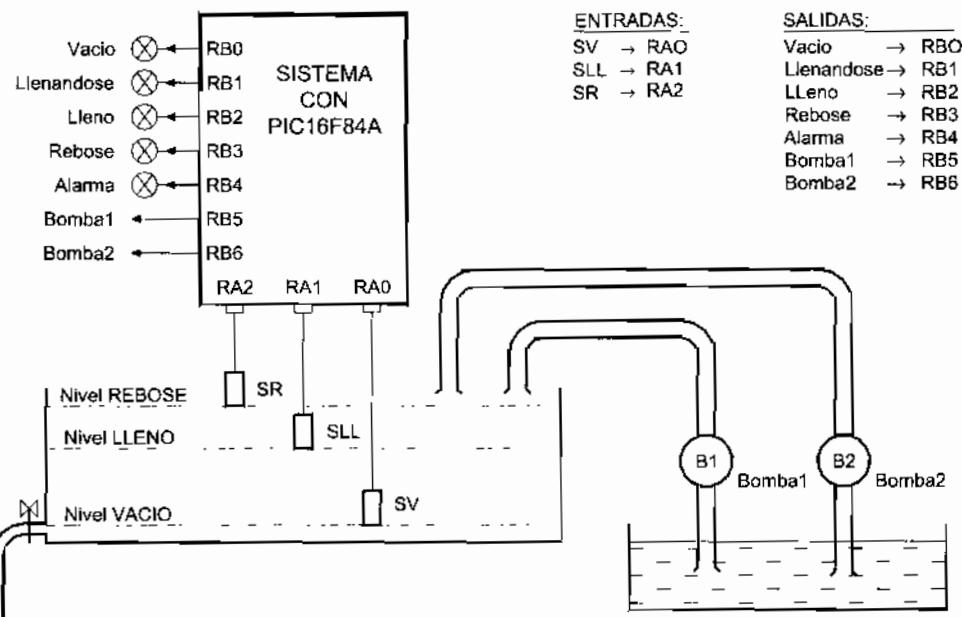


Figura 9-9 Control de un depósito de líquidos mediante microcontrolador

**Indexado\_03.asm:** Realizar un programa ejemplo en el que se produzca un salto indexado descontrolado debido a un uso incorrecto de la instrucción `addwf PCL,F`.

**Fibonacci.asm:** Obtener el último término de la secuencia de Fibonacci menor de 256 y sacar ese valor por el puerto de salida. Recordar que los términos de la secuencia de Fibonacci son: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... En esta secuencia los dos primeros números (0 y 1) se denominan "semillas" de la secuencia, ya que son conocidos de antemano y a partir de ellos se construyen los demás términos, de acuerdo con la regla de que cada número es la suma de los dos términos que le preceden. Por ejemplo, el término que sigue al 55 será  $34 + 55 = 89$ .

Previamente es conveniente realizar el diagrama de flujo. (La solución se ofrece en el documento Fibonacci.doc que se encuentra en el CD-ROM que acompaña a este libro).

En la tarea específica estas técnicas

## 10.1 S

Algunos programas tramo de veces como atacar el programa de como se mu veces.

La so de instrucciones para ser ejec figura 10-1.

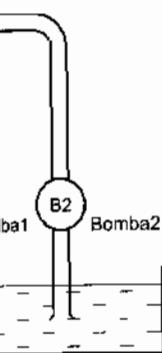
Una s cualquier pu ejecuta cada

apaga también la  
dicador "Rebose"

sondas de entrada  
(acío) se paran las

**SALIDAS:**

Vacio	→ RBO
Llenandose	→ RB1
LLeno	→ RB2
Rebose	→ RB3
Alarma	→ RB4
Bomba1	→ RB5
Bomba2	→ RB6



controlador

produzca un salto  
*dwf PCL.F.*

ibonacci menor de  
s de la secuencia de  
a los dos primeros  
son conocidos de  
erdo con la regla de  
ejemplo, el término

olución se ofrece en  
mpaña a este libro).

## CAPÍTULO 10

# SUBRUTINAS

En la actualidad los programas tienden a diseñarse de una forma modular: cada tarea específica es realizada por un “módulo” determinado. En este capítulo trataremos estas técnicas para realizar códigos de programa más cortos y eficaces.

### 10.1 SUBRUTINAS

Algunas veces el mismo grupo de instrucciones es ejecutado en diferentes partes de un programa. Según el procedimiento planteado hasta el presente, cada vez que dicho tramo de programa es requerido deberá insertarse dentro del programa principal tantas veces como éste sea necesario. Sin embargo, aunque esta parezca la forma más directa de atacar el problema, la mayoría de las veces es la más ineficiente, ya que requiere mayor extensión de los programas y, en consecuencia, mayor utilización de la memoria ROM de programa del microcontrolador. El flujo del programa resulta meramente secuencial, tal como se muestra en el ejemplo de la figura 10-1, donde un mismo proceso se repite 3 veces.

La solución más efectiva en términos de ahorro de memoria, se obtiene si el grupo de instrucciones que se repite aparece una sola vez en el programa, pero con capacidad para ser ejecutado desde todos los puntos en que aquél se pide, tal como se muestra en la figura 10-1. La estructura de programación que implementa esta solución es la **subrutina**.

Una subrutina es un conjunto de instrucciones al que se tiene acceso desde cualquier punto del programa principal. Es decir, una subrutina es un subprograma que se ejecuta cada vez que el programa principal lo necesita.

Como una subrutina conceptualmente queda fuera del flujo secuencial del programa principal, son necesarios ciertos mecanismos para poder llegar a ella y, una vez que se han ejecutado las instrucciones que la componen, debe ser posible regresar al punto donde se quedó la ejecución del programa.

La acción de pasar del programa principal a la subrutina se denomina "llamada a la subrutina" y se realiza con la instrucción *call* que se debe intercalar en el programa principal, tal como se describe en la figura 10-1.

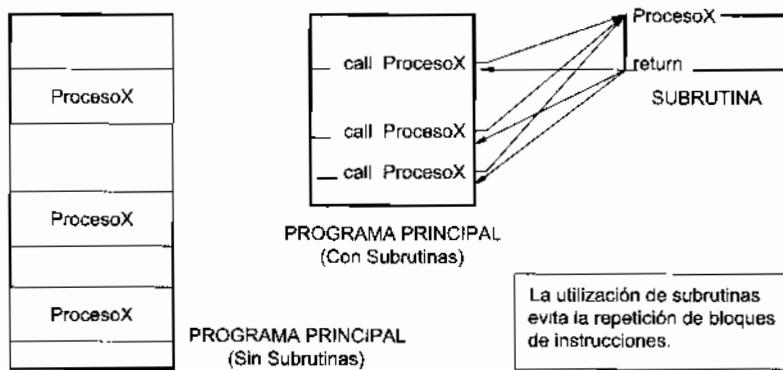


Figura 10-1 Utilización de las subrutinas

La acción de volver al programa principal después de llevar a cabo las tareas determinadas por la subrutina se llama "retorno de la subrutina" y se realiza con la instrucción *return*, con la que debe finalizar siempre las subrutinas (figura 10-1). Efectivamente, *return* es un "salto inteligente" que sabe que tiene que volver a la instrucción inmediatamente después del *call* que llamó a la subrutina.

La figura 10-1 ilustra el procedimiento de ejecución del programa con subrutinas. Las instrucciones del programa principal son ejecutadas sucesivamente hasta que se encuentra la primera instrucción *call ProcesoX*, después de lo cual, la subrutina ProcesoX se ejecuta como cualquier otra sección del programa. La última instrucción de la subrutina es *return* que causa el regreso de la secuencia de ejecución al programa principal. La instrucción ejecutada después del *return* es la que sigue al primer *call ProcesoX* en el programa principal. La ejecución del programa continúa normalmente hasta que aparece una segunda llamada al ProcesoX, *call ProcesoX*, de nuevo se transfiere el control a la subrutina y la subrutina ProcesoX es ejecutada. Al llegar a la instrucción *return* ocurre un nuevo retorno al programa principal, esta vez a la instrucción siguiente al segundo *call ProcesoX*. Lo mismo ocurre con el tercer *call ProcesoX*.

El programa con subrutinas de la figura 10-1 podría tener el siguiente formato:

secuencial del  
a ella y, una vez  
regresar al punto

ina "llamada a la  
· en el programa

TINA

tinas  
logues

Al cabo las tareas se realizan con las (figura 10-1), que volver a la

a con subrutinas. nte hasta que se ual, la subrutina na instrucción de ción al programa e al primer *call* núa normalmente X, de nuevo se da. Al llegar a la il, esta vez a la on el tercer *call*

nte formato:

Principal	....	; Comienzo del programa principal.
call	ProcesoX	; El control del programa pasa a la subrutina "ProcesoX".
....		; Después de ejecutar la subrutina la ejecución del programa
....		; vuelve a la instrucción inmediatamente después de este "call".
....		
call	ProcesoX	; El control del programa pasa a la subrutina "ProcesoX".
....		; Después de ejecutar la subrutina la ejecución del programa
....		; vuelve a la instrucción inmediatamente después de este "call".
....		
call	ProcesoX	; El control del programa pasa a la subrutina "ProcesoX".
....		; Después de ejecutar la subrutina la ejecución del programa
....		; vuelve a la instrucción inmediatamente después de este "call".
....		
....		; Aquí la instrucción final del programa principal.
;	Subrutina "ProcesoX"	-----
ProcesoX	....	
....		; Aquí las instrucciones de la subrutina.
....		
return		; La subrutina devuelve con "return" el control al programa principal
....		
....		
END		; La directiva "END" se pondrá al final del programa.

La principal ventaja de las subrutinas es que la extensión de los programas se hace mucho más corta, tal como se aprecia en la figura. No obstante, las subrutinas presentan una desventaja que se puede detectar comparándolo con el flujo de ejecución del programa sin subrutinas de la misma figura. Se observa que el uso de subrutinas provoca una ejecución más lenta debido que se tienen que ejecutar dos instrucciones extras *call* y *return* cada vez que se realiza una llamada y el obligatorio retorno de subrutina.

## 10.2 SUBRUTINAS ANIDADAS

Cuando una subrutina llama a otra subrutina se produce la situación conocida como **anidamiento de subrutinas**, es decir, hay subrutinas anidadas dentro de otras. Cada *call* sucesivo sin que intervenga un *return* crea un nivel de anidamiento adicional. Esto se ilustra en la figura 10-2, donde este programa podría tener el siguiente formato:

Principal ; Comienzo del programa principal.  
call Proceso1 ; El control del programa pasa a la subrutina "Proceso1".  
..... ; Después de ejecutar la subrutina "Proceso1" la ejecución del programa  
..... ; vuelve a la instrucción inmediatamente después del "call".

```

    ... ; Aquí la instrucción final del programa principal.

; Subrutina "Proceso1"
; ...
Proceso1
    ... ; Aquí las instrucciones de la subrutina "Proceso1".
    ...
    call Proceso2 ; El control del programa pasa a la subrutina "Proceso2".
    ... ; Después de ejecutar la subrutina "Proceso2" la ejecución del programa
    ... ; vuelve a la instrucción inmediatamente después del "call".
    return ; Esta subrutina devuelve el control al programa principal.

; Subrutina "Proceso2"
; ...
Proceso2
    ... ; Aquí las instrucciones de la subrutina "Proceso2".
    ...
    call Proceso3 ; El control del programa pasa a la subrutina "Proceso3".
    ... ; Después de ejecutar la subrutina "Proceso3" la ejecución del programa
    ... ; vuelve a la instrucción inmediatamente después del "call".
    return

; Subrutina "Proceso3"
; ...
Proceso3
    ... ; Aquí las instrucciones de la subrutina "Proceso3".
    ...
    return
    ...
END ; La directiva "END" se pondrá al final del programa.

```

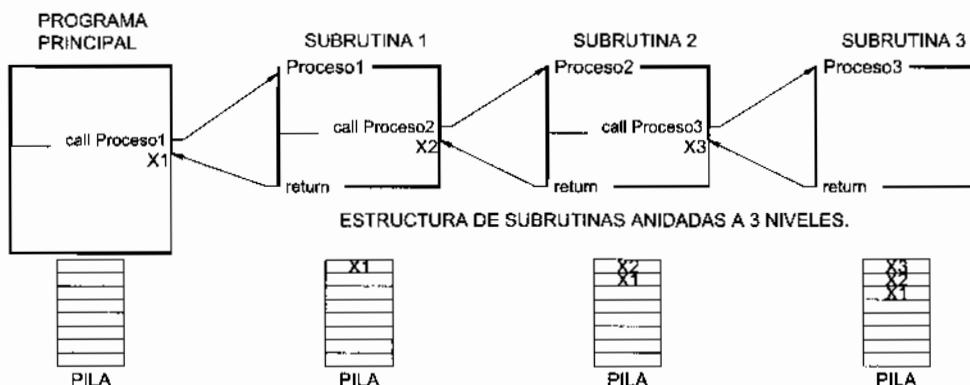


Figura 10-2 Subrutinas anidadas

*Figura 10-*

El PIC16F84 tiene una pila de 13 bits cada uno d

La manera de implementar el *call*, que almacena

### 10.3 LA PILA

La pila (*stack*) es una estructura de memoria de crecimiento descendente. Su estructura es como una pila de platos, es decir, el primero que entra es el primero que sale.

El nivel de la pila es el número de bytes que quedan en memoria más de ocho subrutinas.

El nivel de anidamiento está limitado para cada microcontrolador y en el microcontrolador PIC16F84 es de 8 niveles. Es decir, para un PIC16F84 no puede haber más de ocho subrutinas anidadas como las esquematizadas en la figura 10-2.

### 10.3 LA PILA

La pila (*stack* en inglés) es una zona de memoria que se encuentra separada tanto de la memoria de programa como de la de datos dentro del microcontrolador (figura 4-1). Su estructura es del tipo LIFO (*Last In First Out*) por lo que el último dato que se guarda es el primero que sale.

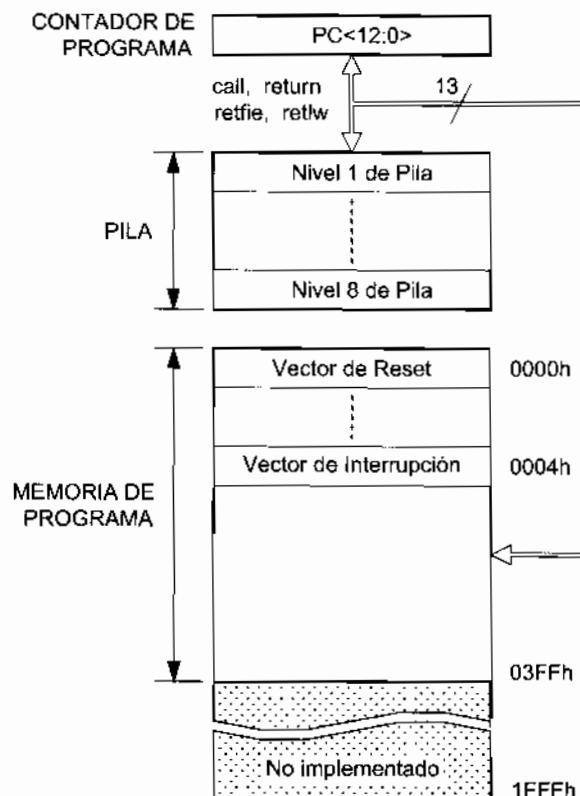


Figura 10-3 Estructura de la pila y memoria de programa del PIC16F84

El PIC16F84 dispone de una pila con ocho niveles o registros de una longitud de 13 bits cada uno de ellos (figura 10-3).

La manera de cargar la pila es a través de la llamada a subrutina con la instrucción *call*, que almacena el contenido del contador de programa (PC) en la posición superior de

la pila. Para recuperar el contenido de la pila en el PC, hay que ejecutar la instrucción de retorno de subrutina *return*.

## 10.4 INSTRUCCIONES "CALL" Y "RETURN"

La localización de una subrutina se identifica por la dirección de su primera instrucción. El efecto de la instrucción *call* es provocar que la ejecución se transfiera a la subrutina. De esto se desprende que la mencionada instrucción contenga la dirección de la primera posición de memoria ocupada por la subrutina (figura 10-4).

Por otro lado, la instrucción *return* que provoca el retorno al programa principal, debe "recordar" la localización de la instrucción que sigue al *call*. Esto es posible sólo si la dirección de esa instrucción ha sido preservada en una zona de memoria, que no es otra que la pila.

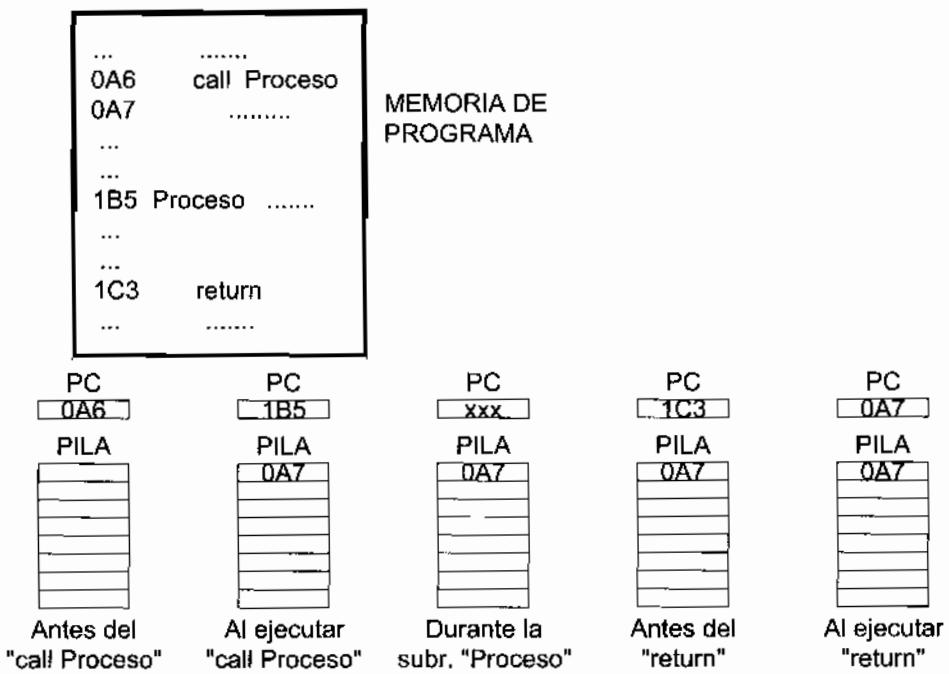


Figura 10-4 Mecanismo de funcionamiento de una subrutina

Antes de transferir el control a una subrutina, la instrucción *call* se encarga de almacenar en la pila la dirección de la instrucción que le sigue. Esto es así porque el contador de programa se incrementa automáticamente cada vez que se usa para obtener una instrucción de la memoria de programa, en el momento en que el microprocesador ejecuta una instrucción el PC ya se encuentra apuntando a la dirección de la siguiente instrucción. Por lo tanto, cuando la instrucción *call* deposita en la pila el contenido del

PC, lo que efectivamente hace es preservar la dirección de la instrucción que sigue al *call*.

La transición entre la ejecución de la subrutina y la ejecución del programa principal se realiza con la dirección de la instrucción que sigue al *call*, que se obtiene de la pila.

La ejecución de la instrucción *return* provoca que la ejecución se transfiera al PC, precisamente en la dirección que sigue al *call*, regresando el control al programa principal.

En las siguientes páginas se detallará más en profundidad el mecanismo de la ejecución de las subrutinas, así como la forma en que se manejan las pilas de cada uno de los procesos que ejecuta el sistema.

## 10.5 EJEMPLOS

A continuación se presentan algunos ejemplos para el circuito de desarrollo.

```
;*****
; Un número binario de 8 posiciones de memoria
; Finalmente también se conectados al Puerto P1
; Realizar este programa
; ZONA DE DATOS
```

```
CONF
LIST
INCLUDE
```

```
CBLOCK
ENDC
```

```
Numero EQU
```

```
ZONA DE CÓDIGO
```

a instrucción de  
PC, lo que efectivamente está preservando es la dirección del punto de retorno. La figura 10-4 ilustra este mecanismo.

La transferencia de la secuencia de ejecución a la subrutina se logra cargando el PC con la dirección contenida en la instrucción *call*. De esta forma, la siguiente instrucción que se obtiene de la memoria proviene de la subrutina.

La ejecución de la instrucción *return* extrae la dirección almacenada en la pila y la transfiere al PC. Así, la próxima instrucción que se ejecuta después del *return* es precisamente la siguiente al *call* (que es la que se había guardado previamente en la pila), regresando el control al programa principal.

En las subrutinas anidadas, el efecto de cada instrucción *call* es guardar en la pila la dirección apropiada para el retorno. La instrucción *return* y sus efectos en la pila aseguran que el control del programa eventualmente regrese a la instrucción siguiente al primer *call*, figura 10-2. El número de subrutinas anidadas está limitado por el tamaño de la pila de cada microcontrolador. En el PIC16F84 la pila tiene un tamaño de 8 posiciones, por tanto el anidamiento máximo es de 8 niveles. No hay ningún mecanismo hardware que indique desbordamiento de la pila, debe ser el diseñador del programa quien debe cuidar que esto no se produzca.

## 10.5 EJEMPLO DE UTILIZACIÓN DE LAS SUBRUTINAS

A continuación se muestra un útil programa ejemplo de utilización de una subrutina para el circuito de la figura 10-5.

```

PC ***** Subrutinas_01.asm *****
0A7
PILA
0A7
*****
Al ejecutar
"return"
; Un número binario de 8 bits es convertido a BCD. El resultado se guarda en tres
; posiciones de memorias RAM de datos llamadas Centenas, Decenas y Unidades.
; Finalmente también las unidades y las decenas se visualizarán en los diodos LEDs
; conectados al Puerto B. El número a convertir será la constante "Numero".
;
; Realizar este programa utilizando una subrutina que se llame BIN_a_BCD.
;
; ZONA DE DATOS *****
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
CBLOCK 0x0C ; En esta posición empieza la RAM de usuario.
ENDC
Numero EQU .124 ; Por ejemplo.
; ZONA DE CÓDIGOS *****

```

```

    ORG 0 ; El programa comienza en la dirección 0.

Inicio   bsf STATUS,RP0 ; Acceso al Banco 1.
          clrf TRISB ; Las líneas del Puerto B se configuran como salida.
          bcf STATUS,RP0 ; Acceso al Banco 0.

Principal movlw Numero
           call BIN_a_BCD
           movwf PORTB ; El resultado se visualiza por la salida.
           goto $ ; Se queda permanentemente en este bucle.

; Subrutina "BIN_a_BCD"
; Un número binario de 8 bits es convertido en BCD. El procedimiento utilizado es mediante
; restas de 10, tal como se explicó en el capítulo 9.

; Entrada: En el registro W el número binario a convertir.
; Salidas: En (Centenas), (Decenas) y (Unidades).
; También las decenas (nibble alto) y unidades (nibble bajo) en el registro (W).

CBLOCK
Centenas
Decenas
Unidades
ENDC

; BIN_a_BCD
        clrf Centenas
        clrf Decenas
        movwf Unidades ; En las subrutinas no se debe fijar la dirección
                      ; de la RAM de usuario. Definida a continuación de
                      ; la última asignada.

Resta10
        movlw .10 ; Carga los registros con el resultado inicial.
        subwf Unidades,W ; En principio (Centenas)=0 y (Decenas)=0.
        btfss STATUS,C ; Se carga el número binario a convertir.

        goto Fin_BIN_BCD ; A las unidades se le va restando 10 en cada
                           ; pasada. (W)=(Unidades)-10.
                           ; ¿(Unidades)>=10? , ¿(W) positivo?, ¿C=1?
                           ; No, es menor de 10. Se acabó.

IncrementaDecenas
        movwf Unidades ; Recupera lo que queda por restar.
        incf Decenas,F ; Incrementa las decenas y comprueba si ha
                         ; llegado a 10. Lo hace mediante una resta.
        movlw .10 ; (W)=(Decenas)-10.
        subwf Decenas,W ; ¿C=1?, ¿(W) positivo?, ¿C=1?
        btfss STATUS,C ; No. Vuelve a dar otra pasada, restándole 10.

IncrementaCentenas
        clrf Decenas ; Pone a cero las decenas
        incf Centenas,F ; e incrementa las centenas.
        goto Resta10 ; Otra pasada: Resta 10 al número a convertir.

Fin_BIN_BCD
        swapf Decenas,W ; En el nibble alto de (W) también las decenas.
        addwf Unidades,W ; En el nibble bajo de (W) las unidades.
        return ; Vuelve al programa principal.

        END ; Fin del programa..

```

**10.6 V**

El en

- Se p
  - muc
    - bina
- Dan
  - mód
    - subr
- Redi
  - El c
    - subr

**10.7 LI**

Es fre
que algunas
disponer de b

En ca
MPASM dis
“pegando” el
durante el pro

**10.8 DIF**

El form

El fich
efecto es el m
del fichero o
tener en cuen
cual debe ir a
muchos quebr

Esta di
al principio en

INCLU

Con la
registros de S

## 10.6 VENTAJAS DE LAS SUBRUTINAS

El empleo de subrutinas aporta muchas ventajas, entre las que destacan:

- Se pueden escribir como subrutinas secciones de código y ser empleadas en muchos programas (por ejemplo, la subrutina de conversión de un número binario natural a BCD).
- Dan a los programas un carácter modular. Se pueden codificar diferentes módulos para usarlos en cualquier programa obteniendo así una librería de subrutinas.
- Reduce notablemente el tiempo de programación y la detección de errores.
- El código es más fácil de interpretar, dado que las instrucciones de las subrutinas no aparecen en el programa principal. Sólo figuran las llamadas *call*.

## 10.7 LIBRERÍA DE SUBRUTINAS

Es frecuente necesitar más de una subrutina en los programas. También es habitual que algunas subrutinas se utilicen en varios programas. En estos casos es conveniente disponer de bibliotecas de subrutinas denominadas **librerías**.

En cada programa se cargan las subrutinas que se precisen. El ensamblador MPASM dispone de una directiva denominada **INCLUDE** que realiza esta función “pegando” el fichero de referencia en el programa. Dicho fichero se inserta en el código durante el proceso de ensamblado.

## 10.8 DIRECTIVA “INCLUDE”

El formato de la directiva **INCLUDE** es:

*INCLUDE <include\_file>*

El fichero especificado por *<include\_file>* es leído como un fichero fuente. El efecto es el mismo que si el texto entero del *<include\_file>* hubiera sido insertado dentro del fichero origen en la localización donde esta directiva se encuentre. Es muy importante tener en cuenta que este fichero *<include\_file>* no debe finalizar con la directiva **END**, la cual debe ir situada en el programa principal. Este error es muy frecuente y ocasiona muchos quebraderos de cabeza a los programadores noveles.

Esta directiva se ha usado hasta ahora en todos los ejercicios cuando se ha incluido al principio en los programas:

INCLUDE <P16F84A.INC>

Con la directiva anterior lo que se hace es añadir al programa la definición de los registros de SFR y de sus bits.

El <include\_file> puede ir encerrado entre comillas o entre signos <>, tal como se indica en el ejemplo:

```
INCLUDE "c:\sys\sysdefs.inc"      ; Definiciones del sistema.
INCLUDE <P16F84A.INC>          ; Definiciones de algunos registros.
```

Si es especificada la trayectoria completa del fichero <include\_file>, sólo en ese trayecto será buscado. En caso contrario el orden de búsqueda será: directorio actual de trabajo, directorio del fichero fuente y, por último, directorio del fichero ejecutable MPASM.EXE.

La extensión "\*.INC" no es obligatoria, aunque se recomienda su utilización para diferenciarlos de los programas propiamente dichos con extensión "\*.asm".

Los ficheros *include* a su vez pueden tener otras directivas *INCLUDE*, pero no es recomendable ya que puede provocar confusión. De todas formas este anidamiento tampoco puede ser ilimitado, puesto que el MPASM sólo permite hasta seis niveles.

El siguiente programa ejemplo llamado Subrutinas\_02.asm, muestra la utilización de una subrutina incluida en una librería de programas BIN\_BCD.INC. Este programa se puede comprobar en el circuito de la figura 9-6.

```
***** Subrutinas_02.asm *****
;
; Repetir el programa anterior de conversión de un número binario a decimal pero
; utilizando la librería "BIN_BCD.INC".
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C           ; En esta posición empieza la RAM de usuario.
ENDC

Numero EQU .124       ; Por ejemplo.

; ZONA DE CÓDIGOS *****
ORG 0                 ; El programa comienza en la dirección 0.

Inicio
    bsf STATUS,RP0   ; Acceso al Banco 1.
    clrf TRISB        ; Líneas del Puerto B configuradas como salida.
    bcf STATUS,RP0   ; Acceso al Banco 0.

Principal
    movlw Numero
    call BIN_a_BCD
    movwf PORTB        ; El resultado se visualiza por la salida.
    goto $              ; Se queda permanentemente en este bucle.
```

INC

EN

La s

del progra

\*\*\*\*\*

;

; Un númer

; posiciones de

;

; El procedimi

;

; Entrada: En e

; Salidas: En e

; En e

; Subrutina "B

CBI

BCD

BCD

BCD

END

;

BIN\_a\_BCD

clrf

clrf

mov

BCD\_Resta10

mov

; subw

btfs

goto

BCD\_Incremen

mov

incf

movl

subw

btfs

goto

BCD\_Incremen

clrf

incf

goto

BIN\_BCD\_Fin

swap

addw

return

;

La directiva "E

gnos <>, tal como

file>, sólo en ese directorio actual de fichero ejecutable

su utilización para sm".

LUDE, pero no es este anidamiento seis niveles.

uestra la utilización C. Este programa se

\*\*\*\*\*

\*\*\*\*\*

de usuario.

\*\*\*\*\*

ción 0.

como salida.

ida.  
e bucle.

```
INCLUDE <BIN_BCD.INC> ; La subrutina se añadirá al final del programa
END ; principal
; Fin del programa.
```

La subrutina BIN\_a\_BCD se localiza en la librería BIN\_BCD.INC incluida al final del programa mediante la directiva INCLUDE. Esta librería podría ser:

```
***** Librería "BIN_BCD.INC" ****
;
; Un número binario natural de 8 bits es convertido a BCD. El resultado se guarda en tres
; posiciones de memorias llamadas: BCD_Centenas, BCD_Decenas y BCD_Unidades.
;
; El procedimiento utilizado es mediante restas de 10, tal como se explicó en el capítulo 9.
;
; Entrada: En el registro W el número binario natural a convertir.
; Salidas: En (BCD_Centenas), (BCD_Decenas) y (BCD_Unidades).
; En el registro W también las decenas (nibble alto) y unidades (nibble bajo).
;
; Subrutina "BIN_a_BCD" -----
```

CBLOCK BCD_Centenas BCD_Decenas BCD_Unidades ENDC ; BIN_a_BCD clrf    BCD_Centenas clrf    BCD_Decenas movwf  BCD_Unidades BCD_Resta10 movlw  .10 subwf  BCD_Unidades,W btfs  STATUS,C goto  BIN_BCD_Fin BCD_IncrementaDecenas movwf  BCD_Unidades inef  BCD_Decenas,F movlw  .10 subwf  BCD_Decenas,W btfs  STATUS,C goto  BCD_Resta10 BCD_IncrementaCentenas clrf    BCD_Decenas inef  BCD_Centenas,F goto  BCD_Resta10 BIN_BCD_Fin swapf  BCD_Decenas,W addwf  BCD_Unidades,W return	; En las subrutinas no se debe fijar la dirección ; de la RAM de usuario. Se toma a continuación de ; la última asignada. ; ; Carga los registros con el resultado inicial. ; En principio las centenas y decenas a cero. ; Se carga el número binario a convertir. ; ; A las unidades se les va restando 10 en cada ; pasada. (W)=(BCD_Unidades)-10. ; ¿C = 1?, ¿(W) positivo?, ¿(BCD_Unidades)>=10? ; No, es menor de 10. Se acabó. ; ; Recupera lo que queda por restar. ; Incrementa las decenas y comprueba si ha llegado ; a 10. Lo hace mediante una resta. ; (W)=(BCD_Decenas)-10. ; ¿C = 1?, ¿(W) positivo?, ¿(BCD_Decenas)>=10? ; No. Vuelve a dar otra pasada, restándole 10 a ; las unidades. ; Pone a cero las decenas ; e incrementa las centenas. ; Otra pasada: Resta 10 al número a convertir. ; ; En el nibble alto de (W) también las decenas. ; En el nibble bajo de (W) las unidades. ; Vuelve al programa principal.
--	--

; La directiva "END" se debe poner en el programa principal no aquí.

Se observa como en la librería de subrutinas la directiva *CBLOCK* no está direccionada y como esta librería NO acaba con *END*.

El fichero Subrutinas\_02.lst contiene el ensamblado completo del programa donde se aprecia como la directiva *INCLUDE* "pega" el fichero BIN\_BCD.INC dentro del programa principal. A continuación se exponen los fragmentos más importantes de este fichero. Se recomienda su análisis completo tal como se sugiere en un ejercicio de la práctica de laboratorio del final del capítulo:

MPASM 03.30.01 Released SUBRUTINAS\_02.ASM 7-2-2003 17:28:00 PAGE 1

LOC OBJECT CODE LINE SOURCE TEXT  
VALUE

```

00001 ;***** Subrutas_02.asm *****
00002 ;
00003 ; Repetir el programa anterior de conversión de un número binario a decimal pero
00004 ; utilizando la librería "BIN_BCD.INC".
00005 ;
00006 ; ZONA DE DATOS *****
00007
2007 3FF1 00008 _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
00009 LIST P=16F84A
00010 INCLUDE <P16F84A.INC>
00001 LIST
00002 ; P16F84A.INC Standard Header File, Version 2.00 Microchip Technology, Inc.
00134 LIST
00011
00012 CBLOCK 0x0C ; En esta posición empieza la RAM de usuario.
00013 ENDC
00014
0000007C 00015 Numero EQU .124 ; Por ejemplo.
00016
00017 ; ZONA DE CÓDIGOS *****
00018
0000 00019 ORG 0 ; El programa comienza en esta dirección
0000 00020 Inicio
0000 1683 00021 bsf STATUS,RP0 ; Acceso al banco 1.
0001 0186 00022 clrf TRISB ; Líneas del Puerto B configuradas como salidas.
0002 1283 00023 bcf STATUS,RP0 ; Acceso al banco 0.
0003 00024 Principal
0003 307C 00025 movlw Numero
0004 2007 00026 call BIN_a_BCD
0005 0086 00027 movwf PORTB ; El resultado se visualiza por la salida.
0006 2806 00028 goto $ ; Se queda permanentemente en este bucle.
00029
00030 INCLUDE <BIN_BCD.INC> ; La subrutina se añadirá al final del programa
00001 ;***** Librería "BIN_BCD.INC" *****
00002 ;
00003 ; Un número binario de 8 bits es convertido en BCD. El resultado se guarda en tres
00004 ; posiciones de memorias llamadas BCD_Centenas, BCD_Decenas y BCD_Unidades.
00005 ;

```

00000000C 0001  
00000000D 0001  
00000000E 0001  
0001  
0001  
0007 0002  
0007 018C 0002  
0008 018D 0002  
0009 008E 0002  
000A 0002  
000A 300A 0002  
000B 020E 0002  
000C 1C03 0002  
000D 2817 0002  
000E 00029  
000E 008E 00030  
000F 0A8D 00031  
0010 300A 00032  
0011 020D 00033  
0012 1C03 00034  
0013 280A 00035  
0014 00036  
0014 018D 00037  
0015 0A8C 00038  
0016 280A 00039  
0017 00040  
0017 0E0D 00041  
0018 070E 00042  
0019 0008 00043  
00044  
00045  
00031  
00032

#### SYMBOL TABLE

#### LABEL

BCD\_Centenas  
BCD\_Decenas  
BCD\_IncremataCent  
BCD\_IncremataDec  
BCD\_Resta0  
BCD\_Unidades  
BIN\_BCD\_Fin

ra CBLOCK no está  
o del programa donde  
BCD.INC dentro del  
as importantes de este  
en un ejercicio de la

8:00 PAGE 1

\*\*\*\*\*

rio a decimal pero

\*\*\*\*\*

&amp; \_XT\_OSC

rochip Technology, Inc.

RAM de usuario.

\*\*\*\*\*

dirección

das como salidas.

salida.

este bucle.

al final del programa  
NC" \*\*\*\*\*lo se guarda en tres  
mas y BCD\_Unidades.

```

00006 ; El procedimiento utilizado es mediante restas de 10 tal, como se explicó en el capítulo 9.
00007 ;
00008 ; Entrada: En el registro W el número binario a convertir.
00009 ; Salidas: En (BCD_Centenas), (BCD_Decenas) y (BCD_Unidades).
00010 ; En el registro W también las decenas (nibble alto) y unidades (bajo).
00011 ;
00012 ; Subrutina "BIN_a_BCD"
00013
00014    CBLOCK
000000C 00015    BCD_Centenas
000000D 00016    BCD_Decenas
000000E 00017    BCD_Unidades
00018    ENDC
00019 ;
0007    00020 BIN_a_BCD
0007 018C 00021    clrf    BCD_Centenas
0008 018D 00022    clrf    BCD_Decenas
0009 008E 00023    movwf   BCD_Unidades
000A    00024 BCD_Resta10
000A 300A 00025    moviw   .10
000B 020E 00026    subwf   BCD_Unidades,W
000C 1C03 00027    btfss   STATUS,C
000D 2817 00028    goto    BIN_BCD_Fin
000E    00029 BCD_IncrementaDecenas
000E 008E 00030    movwf   BCD_Unidades
000F 0A8D 00031    incf    BCD_Decenas,F
0010 300A 00032    movlw   .10
0011 020D 00033    subwf   BCD_Decenas,W
0012 1C03 00034    btfss   STATUS,C
0013 280A 00035    goto    BCD_Resta10
0014    00036 BCD_IncrementaCentenas
0014 018D 00037    clrf    BCD_Decenas
0015 0A8C 00038    incf    BCD_Centenas,F
0016 280A 00039    goto    BCD_Resta10
0017    00040 BIN_BCD_Fin
0017 0E0D 00041    swapf   BCD_Decenas,W
0018 070E 00042    addwf   BCD_Unidades,W
0019 0008 00043    return
00044
00045 ; La directiva "END" se debe poner en el programa principal no aquí.
00031           ; principal
00032    END           ; Fin del programa.

```

## SYMBOL TABLE

LABEL	VALUE
BCD_Centenas	0000000C
BCD_Decenas	0000000D
BCD_IncrementaCentenas	00000014
BCD_IncrementaDecenas	0000000E
BCD_Resta10	0000000A
BCD_Unidades	0000000E
BIN_BCD_Fin	00000017

BIN_a_BCD	00000007
C	00000000
DC	00000001

## 10.9 SIMULACIÓN DE SUBRUTINAS EN MPLAB

En el simulador del MPLAB, las subrutinas se pueden ejecutar de la formas citadas en el capítulo 7. Así, con el comando *Debugger > Step Into* se ejecutarían paso a paso todas y cada una de las instrucciones que conforman la subrutina.

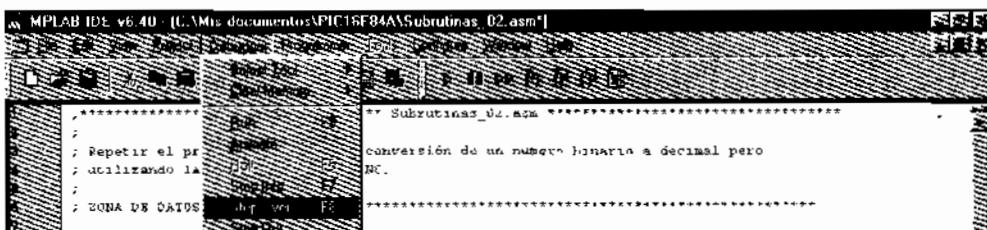


Figura 10-5 Las subrutinas se pueden ejecutar con el comando *Step Over*

Sin embargo, a veces en la simulación interesa ejecutar la subrutina sin tener que ejecutar una por una todas sus instrucciones. Para esto se utiliza el comando *Debugger > Step Over*, figura 10-5. Esta opción ejecuta paso a paso las instrucciones de igual forma que lo hace la opción *Debugger > Step Into*, pero cuando llega a la ejecución de una subrutina (instrucción *call*) ésta se ejecuta de igual forma que si fuera una sola instrucción. Para activar esta opción también se puede pulsar la tecla F8 o bien señalando con el ratón el ícono correspondiente de la barra de herramientas.

El contenido de la pila puede visualizarse seleccionando la opción *View > Hardware Stack*, figura 10-6.

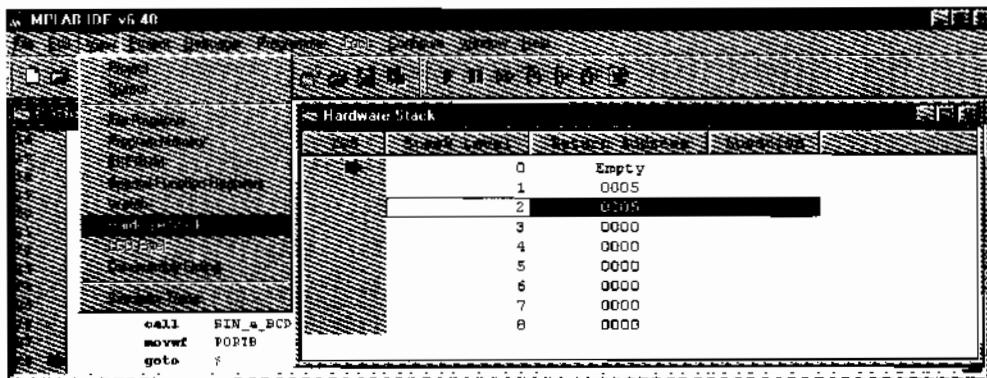


Figura 10-6 Visualización de la pila en el MPLAB

## 10.10 PR

La pro  
programas. S  
procedimient  
uno detrás de  
entre ellos.

Un pro  
tarea del algor  
a la de los pro  
técnica estructur

- Deben consig
- Hay que d
- Lo ante transmi los regi este fin
- El pro únicamente entra
- Las ins principa
- Las sub program

Las vent

- Simplifican el c
- Producen m

En progra  
estructurado. Si  
decisivos.

En todos l  
en una técnica

## 10.10 PROGRAMACIÓN ESTRUCTURADA

La **programación estructurada** es un importante concepto en el diseño de programas. Se entiende como la división del programa principal en módulos o **procedimientos** que realizan una determinada tarea dentro del programa y que se ejecutan uno detrás de otro, de forma preferentemente lineal y con una cantidad mínima de saltos entre ellos.

Un procedimiento consta de un conjunto de pasos necesarios para llevar a cabo una tarea del algoritmo. En el lenguaje ensamblador, las subrutinas hacen una función similar a la de los procedimientos. Para que la programación en ensamblador se aproxime a la técnica estructurada las subrutinas deben tener las siguientes características:

- Deben ser independientes. La modificación de una subrutina no debe llevar consigo la redefinición de otras.
- Hay que plantear cada subrutina de forma que el programa le proporcione unos datos de entrada. La subrutina procesa estos datos y devuelva unos resultados.
- Lo anterior obliga a definir con mucha precisión los parámetros que utilizan para transmitir datos unas subrutinas a otras. Es decir, hay que definir muy claramente los registros que se utilizan como entradas y los que se utilizan como salida. Para este fin, siempre que sea posible hay que utilizar el registro de trabajo W.
- El programa principal debe estructurarse con un único punto de entrada y un único punto de salida. Es mucho más racional entender un programa con una sola entrada y una sola salida que cuando hay saltos a otras partes del programa.
- Las instrucciones de saltos deben utilizarse lo imprescindible en el programa principal.
- Las subrutinas se deben diseñar para que puedan ser utilizadas en diferentes programas sin dificultad.

Las ventajas de la programación estructurada son:

- Simplifica el desarrollo de cada parte del algoritmo por separado, permitiendo concentrar la atención en pequeños detalles.
- Produce programas que son más fiables, fáciles de entender, documentar y modificar.

En programas pequeños quizás no sean evidentes las grandes ventajas del lenguaje estructurado. Sin embargo, cuando se trata de proyectos, éstas se convierten en factores decisivos.

En todos los programas de este libro se han utilizado con reiteración las subrutinas, en una técnica de programación que se aproxima a la estructurada. En el programa

principal se ha potenciado la utilización de llamada a subrutinas con *call*, en lugar de saltos con la instrucción *goto*. Con ello se aprovecha de las ventajas de la programación estructurada, pero se corre el riesgo de desbordar la pila, hay que tener especial precaución de no superar los 8 niveles de anidamiento de subrutinas permitido.

## 10.11 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador con los siguientes programas. Comprobar su correcto funcionamiento con el esquema de la figura 1-2. El lector puede introducir todas las mejoras que considere conveniente.

**Subrutinas\_01.asm:** Un número binario de 8 bits es convertido a BCD. El resultado se guarda en tres posiciones de memorias llamadas Centenas, Decenas y Unidades. Finalmente también las unidades y las decenas se visualizarán en los diodos LEDs conectados al puerto de salida. El número a convertir será la constante *Numero* (por ejemplo 124). Realizar este programa utilizando una subrutina que se llame *BIN\_a\_BCD*.

**Subrutinas\_02.asm:** Repetir el programa anterior utilizando una librería de subrutinas. Una vez comprobado el correcto funcionamiento del programa anterior visualizar el fichero *Subrutinas\_02.lst* generado por el ensamblador hasta su total comprensión.

**Subrutinas\_03.asm:** El valor del puerto de entrada PORTA es convertido a BCD y el resultado se visualiza por el puerto de salida PORTB. Así por ejemplo, si en el PORTA se lee "—10111", (23 en decimal) por el PORTB se visualizará "0010 0011". Una vez comprobado el correcto funcionamiento del programa anterior, visualizar el fichero *Subrutinas\_03.lst* generado por el ensamblador hasta su total comprensión.

En algunas tablas de datos para aplicaciones. A través de un microcontrolador

## 11.1 TABLAS DE DATOS

Una tabla de datos para el programa puede ser:

### 11.1.1 Instancia de una tabla

La instrucción *return*, produce una característica esencial de la memoria de programación.

El formato de la tabla es:

Donde "*k*"

Una de las tablas de verdad más sencillas es:

//, en lugar de  
a programación  
cial precaución

ñar, ensamblar,  
Comprobar su  
introducir todas

do a BCD. El  
mas, Decenas y  
n en los diodos  
te Número (por  
e BIN\_a\_BCD.

una librería de  
grama anterior  
hasta su total

nvertido a BCD  
emplo, si en el  
á "0010 0011".  
or, visualizar el  
tensión.

## CAPÍTULO 11

# MANEJO DE TABLAS

En algún punto de la mayoría de los proyectos es necesario utilizar una o más tablas de datos. Este capítulo trata de su manejo para el PIC16F84 con interesantes aplicaciones. Además, los conceptos expuestos son fácilmente aplicables a otros microcontroladores.

## 11.1 TABLAS DE DATOS EN MEMORIA DE PROGRAMA

Una tabla de datos en la memoria ROM de programa es una lista de constantes que el programa puede recoger mediante la instrucción *retlw*.

### 11.1.1 Instrucción “*retlw*”

La instrucción *retlw* (*Return with Literal in W*), funciona de forma similar que *return*, produce el retorno de una subrutina pero con un valor en el registro W. Dicha característica es de suma importancia cuando se desea acceder a tablas de datos en la memoria de programa.

El formato de esta instrucción es:

*retlw k*

Donde “k” es el valor de la constante que se carga en el registro de trabajo W.

Una de las mayores aplicaciones del manejo de la tabla de datos, es la resolución de tablas de verdad grabadas en ROM. A continuación se muestra un programa ejemplo suficientemente comentado para el circuito de la figura 9-7.



En el programa se aprecia como la tabla está formada por una serie de datos ordenados secuencialmente, de tal forma que para leer uno de ellos se le suma el valor del registro W al contador de programa mediante la instrucción del salto indexado *addwf PCL,F*, posicionándolo en el valor requerido y extrayendo el dato de la tabla con la instrucción *retlw*. Todo lo explicado en el capítulo 9 sobre direccionamiento indexado también es válido aquí.

Los valores de las constantes están grabados y no se puede alterar. La única manera de alterar una tabla ROM es volver a grabar el microcontrolador.

No se debe confundir las tablas en ROM con las tablas de datos en la memoria RAM, que contienen variables almacenadas en los registros de la memoria de datos y que pueden alterarse.

### 11.1.2 Directiva "DT"

Para simplificar el uso de las instrucciones *retlw* el ensamblador MPASM facilita la directiva *DT* (*Define Tabla*) que sustituye el empleo repetitivo de muchas instrucciones *retlw*. Su sintaxis es:

*DT <expr> [, <expr>, ..., <expr>]*

Esta directiva genera durante la fase de ensamblado instrucciones *retlw*, una instrucción por cada *<expr>*. Cada carácter de una cadena es almacenado en su propia instrucción *retlw*.

#### Ejemplo:

*DT "Ra-Ma", 0x10, .15*

Esta directiva genera las instrucciones:

```
retlw 0x52 ; ('R' en ASCII)
retlw 0x61 ; ('a' en ASCII)
retlw 0x2D ; ('-' en ASCII)
retlw 0x4D ; ('M' en ASCII)
retlw 0x61 ; ('a' en ASCII)
retlw 0x10 ; (16 en decimal)
retlw 0x0F ; (15 en decimal)
```

Como ejemplo de aplicación se repite el programa de la sección anterior utilizando esta directiva en lugar de instrucciones *retlw*.

```
***** Tablas_04.asm *****
; Repetir ejercicio Tablas_03.asm utilizando la directiva DT.

; ZONA DE DATOS *****

    _CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    LIST      P=16F84A
    INCLUDE <P16F84A.INC>

; ZONA DE CÓDIGOS *****

    ORG    0
Inicio
    bsf    STATUS,RP0
    clrf   PORTB
    movlw  b'00011111'
    movwf  PORTA
    bcf    STATUS,RP0
Principal
    movf   PORTA,W           ; Lee el valor de las variables de entrada.
    andlw  b'00000111'        ; Se queda con los tres bits de entrada.
    call   TablaVerdad       ; Obtiene la configuración de salida.
    movwf  PORTB             ; Se visualiza por el puerto de salida.
    goto   Principal

; Subrutina "TablaVerdad"
;
TablaVerdad
    addwf  PCL,F
    DT     0xA, 0x9, 0x23, 0xF, 0x20, 0x7, 0x17, 0x3F ; Configuraciones
                                                       ; de salida.
    END


```

Es fácil concluir que la tabla en este programa con la directiva *DT* se ha grabado de forma mucho más sencilla y corta que en el anterior programa con instrucciones *rethw*.

## 11.2 MÁS DIRECTIVAS

En algunos programas de este capítulo aparecen nuevas directivas tales como *MESSG*, *ERROR*, *IF* y *ENDIF* que pasan a explicarse a continuación:

### 11.2.1 MESSG

Esta directiva permite crear mensajes definidos por el usuario que aparecen al finalizar el proceso de ensamblado y en el fichero listable \*.lst. Su sintaxis es:

*MESSG " <message\_text> "*

Donde <message\_text> es el texto del mensaje.

### 11.2.2 ERR

Esta directiva es similar a la anterior, pero el resultado es enviado al ensamblador MPASM que genera un error de compilación si la pantalla es clásica y una advertencia si la pantalla es moderna.

Por ejemplo, si el código anterior se escribe así:

ERROR "Error"

### 11.2.3 IF y ENDIF

Estas directivas permiten la ejecución condicional en el ensamblado. Su sintaxis es:

Si <expr> es verdadero, se ejecutan las instrucciones entre *IF* y *ENDIF*. La evaluación de la lógica, falsa, no se considera como verdadera.

En el siguiente ejemplo se muestra la memoria de programación:

```
FinTabla
;
IF (FinTabla)
  B
  M
  M
ENDIF
```

Hay que señalar que solo se ejecuta el código entre *IF* y *ENDIF* durante la ejecución.

Donde *<message\_text>* es el texto del mensaje que puede tener hasta 80 caracteres. Más adelante se exponen algunos ejemplos.

### 11.2.2 ERROR

Esta directiva genera un mensaje de error idéntico a cualquier error del ensamblador MPASM. Si el proceso de ensamblado ejecuta esta directiva, aparece la clásica pantalla de error. El texto del mensaje debe ir entrecomillado y puede tener hasta 80 caracteres. Su sintaxis es:

*ERROR "<text string>"*

Por ejemplo, para que aparezca un mensaje durante el proceso de ensamblado, habría que poner la siguiente directiva:

*ERROR "CUIDADO!: Ha habido un error durante el ensamblado"*

### 11.2.3 IF y ENDIF

Estas directivas limitan el principio y el final de un bloque condicional de ensamblado. Su sintaxis es:

*IF <expr>  
.....  
ENDIF*

Si *<expr>* es verdadera, el código inmediato al *IF* se ensamblará. En caso contrario las instrucciones siguientes se saltan hasta encontrar una directiva *ELSE* o una directiva *ENDIF*. La evaluación de una expresión que sea cero se considera, desde el punto de vista de la lógica, falsa. La evaluación de una expresión que sea cualquier valor distinto de cero se considera como verdadera.

En el siguiente ejemplo si la etiqueta "FinTabla" se localiza en una dirección de memoria de programa mayor de 0xFF el ensamblador emitirá un mensaje de error.

```
FinTabla  
:  
    IF (FinTabla > 0xFF)  
        ERROR "CUIDADO!: La tabla ha superado el tamaño de los"  
        MESSG "primeros 256 bytes de memoria ROM. NO funcionará correctamente."  
        MESSG "Para solucionarlo, procurad situar esta subrutina de manera que"  
        MESSG "no supere esta posición o leer nota de aplicación AN556"  
    ENDIF
```

Hay que señalar que *IF* y *ENDIF* son directivas, no instrucciones, por tanto actúan solamente durante el tiempo de ensamblado. Las ejecuta el ensamblador MPASM, no el

microcontrolador PIC16F84 y no pueden utilizarse como instrucciones condicionales que deba ejecutar el microcontrolador. Este error es muy frecuente en diseñadores de programas para microcontroladores novedosos pero que hayan trabajado anteriormente con algún lenguaje de alto nivel como C o Pascal.

### 11.3 GOBIERNO DE UN DISPLAY DE 7 SEGMENTOS

Mediante el uso de tablas es posible gobernar la información que aparece en un display de 7 segmentos como en la figura 11-1.

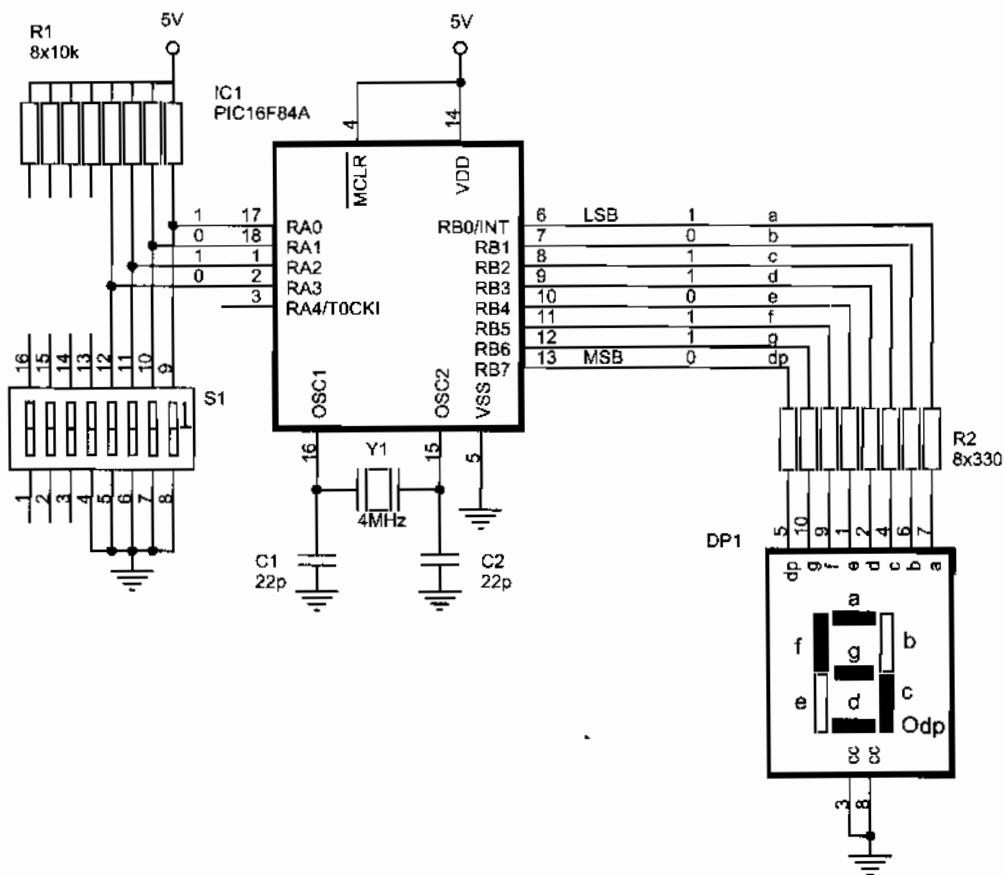


Figura 11-1 Microcontrolador gobernando un display

Así por ejemplo, la figura enseña como para representar el número 5 en el display, por el Puerto B tendría que obtenerse el código (PORTB) = b'01101101' = 6Dh. La figura 11-2 indica los segmentos que deben activarse para la representación de cada uno de los caracteres posibles.

DISPLAY	
Código ASCII (Dec.)	43
Código ASCII (Hex.)	2B
Carácter	+
Siete Segmentos	46
DISPLAY	
Código ASCII (Dec.)	54
Código ASCII (Hex.)	36
Carácter	6
Siete Segmentos	7D
DISPLAY	
Código ASCII (Dec.)	65
Código ASCII (Hex.)	41
Carácter	A
Siete Segmentos	77
DISPLAY	
Código ASCII (Dec.)	76
Código ASCII (Hex.)	4C
Carácter	L
Siete Segmentos	38
DISPLAY	
Código ASCII (Dec.)	87
Código ASCII (Hex.)	57
Carácter	W
Siete Segmentos	1D
DISPLAY	

Figura 11-2

dicionales que  
señadores de  
riamente con

TOS

parece en un

R2  
8x330

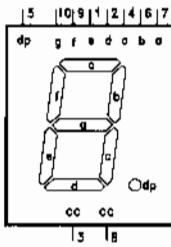
b

c

Odp

8

en el display,  
' = 6Dh. La  
a de cada uno



RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0
dp	g	f	e	d	c	b	a

Conexión del display al puerto de salida.

Código ASCII (Dec.)	43	44	45	46	47	48	49	50	51	52	53
Código ASCII (Hex.)	2B	2C	2D	2E	2F	30	31	32	33	34	35
Carácter	+	,	-	.	(punto)	/	0	1	2	3	4
Síntesis Segmentos	46	04	40	80	00	3F	06	5B	4F	66	60
DISPLAY											
Código ASCII (Dec.)	54	55	56	57	58	59	60	61	62	63	64
Código ASCII (Hex.)	36	37	38	39	3A	3B	3C	3D	3E	3F	40
Carácter	6	7	8	9	:	:	<	=	>	?	®
Síntesis Segmentos	7D	07	7F	67	41	88	00	48	00	00	00
DISPLAY											
Código ASCII (Dec.)	65	66	67	68	69	70	71	72	73	74	75
Código ASCII (Hex.)	41	42	43	44	45	46	47	48	49	4A	4B
Carácter	A	B	C	D	E	F	G	H	I	J	K
Síntesis Segmentos	77	7C	39	5E	79	71	6F	76	19	1E	7A
DISPLAY											
Código ASCII (Dec.)	76	77	78	79	80	B1	82	83	84	85	86
Código ASCII (Hex.)	4C	4D	4E	4F	50	51	52	53	54	55	56
Carácter	L	M	N	0	P	Q	R	S	T	U	V
Síntesis Segmentos	38	37	54	3F	73	67	50	6D	78	1C	3E
DISPLAY											
Código ASCII (Dec.)	87	88	89	90	Del 91 hasta el 164				165	166	167
Código ASCII (Hex.)	57	58	59	5A	Del 5B hasta el A4				A5	A6	A7
Carácter	W	X	Y	Z	Desde [ hasta ñ				ñ	*	*
Síntesis Segmentos	1D	70	6E	49	00				55	FF	63
DISPLAY											

Figura 11-2 Ejemplo de tabla de conversión para display de siete segmentos

El siguiente programa ejemplo muestra una sencilla aplicación.

```
***** Display_01.asm *****
; En un display de 7 segmentos conectado al Puerto B se visualiza la cantidad leída por
; el Puerto A. Así por ejemplo si por la entrada lee "0101" en el display visualiza "5".
; ZONA DE DATOS *****

LIST P=16F84A
INCLUDE <P16F84A.INC>

; ZONA DE CÓDIGOS *****
ORG 0 ; El programa comienza en la dirección 0.
Inicio:
    bsf STATUS,RP0 ; Acceso al Banco 1.
    clrf PORTB ; Las líneas del Puerto B se configuran como salida.
    movlw b'00011111' ; Las 5 líneas del Puerto A se configuran como entrada.
    movwf PORTA
    bcf STATUS,RP0 ; Acceso al Banco 0.

Principal:
    movf PORTA,W ; Lee la entrada
    andlw b'00001111' ; Máscara para quedarse con el valor de las
                      ; entradas correspondientes al nibble bajo.
    call Binario_a_7Segmentos ; Convierte código binario a 7 segmentos del display.
    movwf PORTB ; Resultado se visualiza por el puerto de salida.
    goto Principal

; Subrutina "Binario_7Segmentos"
Binario_a_7Segmentos:
    addwf PCL,F ; Tabla para display de 7 segmentos.
Tabla:
    retiw 3Fh ; El código 7 segmentos para el "0".
    retiw 06h ; El código 7 segmentos para el "1".
    retiw 5Bh ; El código 7 segmentos para el "2".
    retiw 4Fh ; El código 7 segmentos para el "3".
    retiw 66h ; El código 7 segmentos para el "4".
    retiw 6Dh ; El código 7 segmentos para el "5".
    retiw 7Dh ; El código 7 segmentos para el "6".
    retiw 07h ; El código 7 segmentos para el "7".
    retiw 7Fh ; El código 7 segmentos para el "8".
    retiw 67h ; El código 7 segmentos para el "9".
    retiw 77h ; El código 7 segmentos para el "A".
    retiw 7Ch ; El código 7 segmentos para el "B".
    retiw 39h ; El código 7 segmentos para el "C".
    retiw 5Eh ; El código 7 segmentos para el "D".
    retiw 79h ; El código 7 segmentos para el "E".
    retiw 71h ; El código 7 segmentos para el "F".

END ; Fin del programa.
```

Es útil constatar como la siguiente

```
***** *****
; Subrutinas para convertir
; poder activar displays.
; Si el código ASCII es
; el código erróneo b'10000000
; Si se trata de un número
; un numero (de 0 a 9) el
; Entrada: En registro W
; Salida : En registro W,
```

```
CBLOCK
Display7s_Da
ENDC
```

```
Numero_a_7Segmentos
andlw b'00001111'
addlw '0'
```

```
ASCII_a_7Segmentos
movwf Disp
sublw ''
btfs STAB
retlw 00h
movf Disp
sublw 'N'
btfs STAB
retlw 55h
movf Disp
sublw ''
btfs STAB
retlw 63h
```

```
movf Disp
sublw 'Z'
btfs STAB
retlw b'10000000
movlw '4'
subwf Disp
btfs STAB
retlw b'10000000
```

```
InicioTablaASCII
    addwf PCL
    DT 46h,
    DT 7Dh
    DT 77h,
```

Es útil confeccionar un fichero *include* con las subrutinas de gobierno del display, tal como la siguiente librería DISPLAY\_7S.INC:

```
***** Librería "DISPLAY_7S.INC *****

; Subrutinas para convertir un código ASCII en su valor equivalente en 7 segmentos y así
; poder activar displays.

; Si el código ASCII es menor que el signo más '+' o mayor que la letra zeta 'Z' obtiene
; el código erróneo b'10000000' para encender sólo el punto decimal del display.

; Si se trata de un número hay que ejecutar la subrutina "Número_a_7Segmentos" que convierte
; un numero (de 0 a 9) en su equivalente en código de 7 segmentos.

; Entrada: En registro W, el dato en código ASCII a convertir.
; Salida : En registro W, el código 7 segmentos.

CBLOCK
Display7s_Dato
ENDC ; Aquí se reservará el valor del dato de entrada.

Número_a_7Segmentos
andlw b'00001111'
addlw '0' ; Se queda con el nibble bajo.
           ; Se pasa a ASCII sumándole el valor ASCII
           ; del "0" y ejecuta "ASCII_7_Segmentos".

ASCII_a_7Segmentos
movwf Display7s_Dato ; Guarda el valor del carácter.
sublw '' ; Comprueba si es " " (espacio).
btfsr STATUS,Z ; ¿Es distinta de " " (espacio)?, ¿Z=0?
retlw 00h ; Es " ". Devuelve el código 7-Segmentos del " ".
movf Display7s_Dato,W ; Recupera el valor del dato de entrada.
sublw 'N' ; Comprueba si es "N".
btfsr STATUS,Z ; ¿Es distinta de "N"? , ¿Z=0?
retlw 55h ; Es "N". Devuelve el código 7-Segmentos de "N".
movf Display7s_Dato,W ; Recupera el valor del dato de entrada.
sublw 'ñ' ; Comprueba si es "ñ".
btfsr STATUS,Z ; ¿Es distinta de "ñ"? , ¿Z=0?
retlw 63h ; Es "ñ". Devuelve el código 7-Segmentos de "ñ".

; movf Display7s_Dato,W ; Comprueba si el código ASCII es mayor que la "Z".
sublw 'Z' ; (W)=Z-(Display7s_Dato)
btfsr STATUS,C ; ¿C=1? , ¿(W) positivo?, ¿'Z'>=(Display7s_Dato)?
retlw b'10000000' ; Si el código ASCII es mayor que 'Z' es un error.
movlw '+' ; Averigua en qué orden está el carácter leído
subwf Display7s_Dato,W ; dentro de la tabla de conversión, respecto del
btfsr STATUS,C ; primero que es '+'.
retlw b'10000000' ; Si el código ASCII es menor que '+' es un error.

; addwf PCL,F ; Obtiene el código 7 segmentos.

InicioTablaASCII
DT 46h, 04h, 40h, 80h, 00h, 3Fh, 06h, 5Bh, 4Fh, 66h, 6Dh ; Signos y
DT 7Dh, 07h, 7Fh, 67h, 41h, 88h, 00h, 48h, 00h, 00h, 00h ; números.
DT 77h, 7Ch, 39h, 5Eh, 79h, 71h, 6Fh, 76h, 19h, 1Eh, 7Ah, 38h, 37h ; Letras.
```

DT 54h, 3Fh, 73h, 67h, 50h, 6Dh, 78h, 1Ch, 3Eh, 1Dh, 70h, 6Eh, 49h  
FinTablaASCII

; Esta es la tabla para la visualización en display de siete segmentos (Ver figura 11-3).

```
IF (FinTablaASCII > 0xFF)
    ERROR "¡CUIDADO!: La tabla ha superado el tamaño de los"
    MESSG "primeros 256 bytes de memoria ROM. NO funcionará correctamente."
ENDIF
```

Un ejemplo de aplicación podría ser el programa Display\_05.asm suficientemente documentado:

```
***** Display_05.asm *****
; Visualizar un carácter ASCII en el display de 7 segmentos. Utilizar la subrutina
; "ASCII_a_7Segmentos" contenida en la librería "DISPLAY_7S.INC".
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C ; En esta posición empieza la RAM de usuario.
ENDC

Caracter EQU P

; ZONA DE CÓDIGOS *****
ORG 0 ; El programa comienza en la dirección 0.

Inicio
    bsf STATUS,RP0 ; Acceso al Banco 1.
    clrf PORTB ; Las líneas del Puerto B se configuran como salida.
    bcf STATUS,RP0 ; Acceso al Banco 0.

Principal
    movlw Caracter ; Lee el carácter de entrada
    call ASCII_a_7Segmentos ; Convierte a 7 Segmentos.
    movwf PORTB ; Resultado se visualiza por el puerto de salida.
    goto Principal ; Vuelve al principio.

INCLUDE <DISPLAY_7S.INC> ; Subrutina "ASCII_a_7Segmentos".
END ; Fin del programa.
```

## 11.4 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador con los siguientes programas. Comprobar su correcto funcionamiento con el esquema de la figura 1-2. El lector puede introducir todas las mejoras que considere conveniente.

**Tablas\_01.asm**  
LEDs que se iluminan cinco diodos LEDs instrucción *relw*. Saltos\_06.asm). Ser

**Tablas\_02.asm**  
pareces se encienden.

**Tablas\_03.asm**  
grabadas en ROM. que desee y que sea

**Tablas\_04.asm**

**Tablas\_05.asm**  
líquidos tal como se ocasión resolverlo m

**Tablas\_06.asm**  
directiva *DT* y visualizara

**Display\_01.asm**  
la cantidad leída por el display se visualizará

**Display\_02.asm**

**Display\_03.asm**  
una de las 26 letras que determina el orden le

- Si por el Puerto A que está en el
- Si por el Puerto B que es la que

**Display\_04.asm**  
una de las 26 letras que determina el valor de mayúsculas. Así por el display se visualiza la

**Display\_05.asm**  
Utilizar la subrutina A

**Tablas\_01.asm:** Lee las tres líneas más bajas del puerto A, que fijan el número de LEDs que se iluminarán. Así por ejemplo, si se lee el dato "---00101" (cinco) se encienden cinco diodos LEDs (D4, D3, D2, D1 y D0). Se resolverá utilizando tablas mediante la instrucción *retlw*. Este ejercicio se resolvió mediante bucle en el capítulo 9 (programa Saltos\_06.asm). Sería conveniente comparar las dos formas de resolverlo.

**Tablas\_02.asm:** Igual que el anterior, y además si supera el número 8 los LEDs pares se encienden.

**Tablas\_03.asm:** Resolver una tabla de verdad mediante el manejo de tablas grabadas en ROM. La tabla será ideada por el lector con el número de entradas y salidas que desee y que sea físicamente posible implementar por el microcontrolador.

**Tablas\_04.asm:** Repetir el ejercicio anterior utilizando la directiva *DT*.

**Tablas\_05.asm:** Diseñar un programa para controlar el nivel de un depósito de líquidos tal como se especificó en el ejercicio Indexado\_2.asm del capítulo 9. Pero en esta ocasión resolverlo mediante la utilización de la directiva *DT*.

**Tablas\_06.asm:** Halla la longitud de un mensaje grabado en la ROM mediante la directiva *DT* y visualiza el resultado en binario por los LEDs de la salida.

**Display\_01.asm:** En un display de 7 segmentos conectado al Puerto B se visualiza la cantidad leída por el puerto A. Así por ejemplo si por la entrada se lee "---0101" en el display se visualizará "5".

**Display\_02.asm:** Repetir el programa anterior pero utilizando la directiva *DT*.

**Display\_03.asm:** En un display de 7 segmentos conectado al puerto B se visualiza una de las 26 letras del alfabeto internacional: de la "A" a la "Z". La letra a visualizar lo determina el orden leído por el Puerto A. Así por ejemplo:

- Si por el Puerto A se lee "---0000" (cero) la letra visualizada será la "A" que es la que está en el orden cero.
- Si por el Puerto A se lee "---1101" (veinticinco), la letra visualizada será la "Z", que es la que está en el orden veinticinco.

**Display\_04.asm:** En un display de 7 segmentos conectado al puerto B se visualiza una de las 26 letras del alfabeto internacional: de la "A" a la "Z". La letra a visualizar lo determina el valor de la constante "Caracter". El carácter de entrada debe estar en mayúsculas. Así por ejemplo, si la constante se expresa como "Caracter EQU 'P'" en el display se visualiza la letra "P".

**Display\_05.asm:** Visualizar un carácter ASCII en el display de 7 segmentos. Utilizar la subrutina ASCII\_a\_7Segmentos contenida en la librería DISPLAY\_7S.INC.

**Display\_06.asm:** Visualiza por el display conectado a la salida un carácter determinado dentro de un mensaje grabado en la memoria ROM de programa mediante la directiva *DT*. El número del carácter a visualizar será la cantidad leída por la entrada. Así, por ejemplo, si el texto grabado en la ROM es: "ESTUDIA ELECTRONICA" y la cantidad leída por la entrada es "--01001" (9 en decimal) por el display aparecerá "L" que es el carácter situado en el lugar 9 del mensaje (la primera letra "E" está en el lugar 0).

En la ma ejecutarse algun

## 12.1 CICLO

El tiempo oscilador conectado máquina PIC16F84 el ciclo en producirse un 1).



Las instrucciones ejecutarse, excepto los ciclos máquina.

ida un carácter  
rama mediante la  
r la entrada. Así,  
TRONICA" y la  
parecerá "L" que  
n el lugar 0).

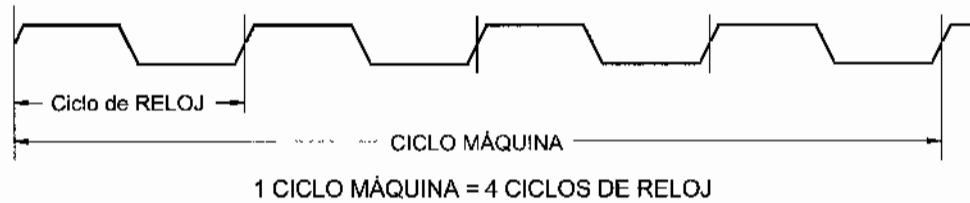
## CAPÍTULO 12

# SUBRUTINAS DE RETARDO

En la mayoría de los proyectos es necesario controlar el tiempo que tardan en ejecutarse algunas acciones. Este capítulo explica una técnica para conseguirlo.

### 12.1 CICLO MÁQUINA

El tiempo que tarda en ejecutarse un programa depende de la frecuencia del oscilador conectado al microcontrolador y del número de ciclos máquina ejecutados. Un **ciclo máquina** es la unidad básica de tiempo que utiliza el microcontrolador. Para el PIC16F84 el ciclo máquina equivale a 4 ciclos de reloj, por lo tanto, el tiempo que tarda en producirse un ciclo máquina es igual a cuatro veces el período del oscilador (figura 12-1).



*Figura 12-1 Ciclo máquina para el PIC16F84*

Las instrucciones en el microcontrolador PIC16F84 necesitan 1 ciclo máquina para ejecutarse, excepto las de salto (*goto*, *call*, *btfss*, *btfsc*, *return*, etc) que necesitan de dos ciclos máquina.

El tiempo que tarda el microcontrolador en ejecutar una tarea viene fijado por la fórmula siguiente:

$$Tiempo = 4 \frac{1}{f} cm$$

Siendo:

- $f$  la frecuencia del oscilador.
- $cm$ , el número de ciclos máquina que tarda en ejecutar la tarea.

**EJEMPLO 1:** Calcular la duración de 1 ciclo máquina para un PIC16F84 que utiliza un cristal de cuarzo de 4 MHz.

**Solución:** Aplicando la ecuación:

$$Tiempo = 4 \frac{1}{f} cm = 4 \frac{1}{4MHz} 1 = 1\mu s$$

Así, para un sistema con un oscilador a cristal de cuarzo de 4 MHz, el ciclo máquina tiene una duración de 1  $\mu s$ .

**EJEMPLO 2:** Calcular el tiempo que tarda en ejecutarse la instrucción *call* si el sistema funcionase con un cristal de cuarzo de 4 MHz.

**Solución:** Al tratarse de un salto esta instrucción dura dos ciclos máquina, por tanto, el tiempo que tarda en ejecutarse será:

$$Tiempo = 4 \frac{1}{f} cm = 4 \frac{1}{4MHz} 2 = 2\mu s$$

**EJEMPLO 3:** En un sistema con microcontrolador PIC16F84 y cristal de cuarzo de 4 MHz se desea generar un retardo de 1,5 ms. Calcular el número de ciclos máquina necesarios.

**Solución:** De la ecuación principal se deduce:

$$Tiempo = 4 \frac{1}{f} cm \quad cm = Tiempo \frac{f}{4} = 1500\mu s \frac{4MHz}{4} = 1500$$

Con el cristal de 4 MHz, el período del oscilador será de 0,25  $\mu s$ , y el ciclo máquina tendrá una duración cuatro veces mayor y de un valor igual a 1  $\mu s$ . Por tanto, para conseguir 1,5 ms serán necesarios 1500 ciclos máquina.

## 12.2 MEDIR

Para calcular contar el número de de la señal de reloj algunas ocasiones re

El MPLAB permite medir el tiem

El cronómetro microcontrolador PIC del oscilador emplea tal como se muestra e se fija la frecuencia básico de la figura 1-

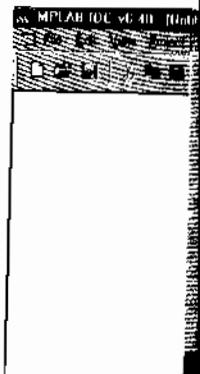


Figura 12-2

Después se activa la ventana que muestra la ejecución de cada instrucción.

Es importante recordar que Hay que tener en cuenta la realidad. Ahora bien, es las instrucciones.

## 12.3 INSTRUCCIONES

La instrucción *call* consume un ciclo máquina.

se fijado por la

PIC16F84 que

MHz, el ciclo

ción *call* si el

s máquina, por

istal de cuarzo  
ciclos máquina

= 1500

l ciclo máquina  
por tanto, para

## 12.2 MEDIR TIEMPOS CON MPLAB

Para calcular el tiempo de ejecución de un programa o de una subrutina, se puede contar el número de instrucciones que se realizan y multiplicarlo por 4 veces la frecuencia de la señal de reloj o por 8 en el caso de que las instrucciones sean de salto. Esto en algunas ocasiones resulta engorroso.

El MPLAB dispone de una opción de cronómetro denominada *Stopwatch* que permite medir el tiempo de ejecución de las instrucciones de los programas.

El cronómetro *Stopwatch* calcula el tiempo basándose en la frecuencia del reloj del microcontrolador PIC que se está simulando. Es necesario fijar previamente la frecuencia del oscilador empleado, para eso, se activa desde el menú *Debugger > Settings > Clock* tal como se muestra en la figura 12-2. Inmediatamente se abre un cuadro de diálogo donde se fija la frecuencia del reloj a 4 MHz para los programas utilizados en el entrenador básico de la figura 1-2.

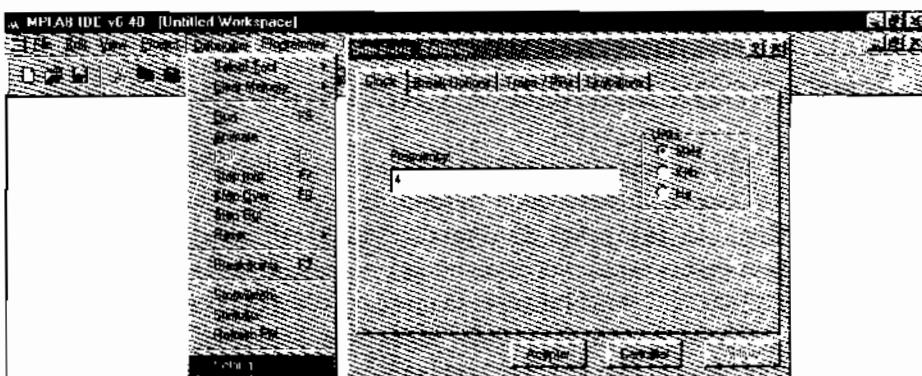


Figura 12-2 Selección de la frecuencia de simulación en el MPLAB SIM

Después se activa la opción *Debugger > Stopwatch*, con esto se consigue abrir la ventana que muestra el tiempo transcurrido y los ciclos máquina empleados en la ejecución de cada instrucción, como puede apreciarse en la figura 12-3.

Es importante recordar que los simuladores software no trabajan en tiempo real. Hay que tener en cuenta que el cronómetro del MPLAB trabaja mucho más lento que la realidad. Ahora bien, el tiempo que muestra será el tiempo real que tardan en ejecutarse las instrucciones.

## 12.3 INSTRUCCIÓN “NOP”

La instrucción *nop* (*No Operation*) no realiza operación alguna. En realidad, consume un ciclo máquina sin hacer nada. Se utiliza para hacer gastar tiempo al

microprocesador sin alterar el estado de los registros ni dc los flags. Esta instrucción tarda 1 ciclo máquina en ejecutarse. Así, para un sistema con un oscilador a cristal de cuarzo de 4 MHz, tendrá una duración de 1  $\mu$ s (4 ciclos de reloj).

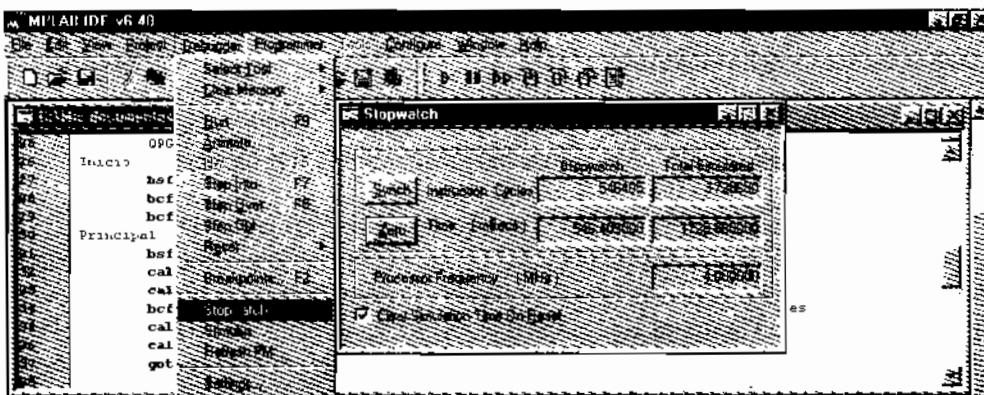


Figura 12-3 Ventana con el contenido del tiempo transcurrido

## 12.4 RETARDOS MEDIANTE LAZO SIMPLE

En muchas aplicaciones y proyectos con microcontroladores resulta necesario generar tiempos de espera, denominados **tiempo de retardo**. Estos intervalos pueden conseguirse mediante una **subrutina de retardo**, basada en un lazo simple de algunas instrucciones que se repiten tantas veces como sea necesario, hasta conseguir el retardo pretendido, figura 12-4(A). Como el tiempo de ejecución de cada instrucción es conocido lo único que hay que hacer es calcular el valor inicial que debe tener el registro *R\_ContA*, que actúa como contador del número de iteraciones en el lazo, para obtener el tiempo de retardo deseado.

Un ejemplo típico de subrutina de retardo puede ser el siguiente fragmento de programa, donde "cm" significan "números de ciclos máquina" y en la figura 12-4(A) se muestra su diagrama de flujo:

Retardo 1ms	;	La llamada "call" aporta 2 ciclos máquina.
movlw d249'	;	Aporta 1 ciclo máquina. Éste es el valor de "K".
movwf R_ContA	;	Aporta 1 ciclo máquina.
R1ms_BucleInterno		
nop	;	Aporta Kx1 ciclos máquina.
decfsz R_ContA,F	;	(K-1)x1 cm (cuando no salta) + 2 cm (al saltar).
goto R1ms_BucleInterno	;	Aporta (K-1)x2 ciclos máquina.
return	;	El salto de retorno aporta 2 ciclos máquina.

; En total esta subrutina tarda:  
;  $2 + 1 + 1 + Kx1 + (K-1)x1 + 2 + (K-1)x2 + 2 = 5 + 4K = 1001$  ciclos máquina (para K=249).  
; 1001 ciclos máquinas suponen 1 milisegundos (1001  $\mu$ s) para una cristal de 4 MHz.

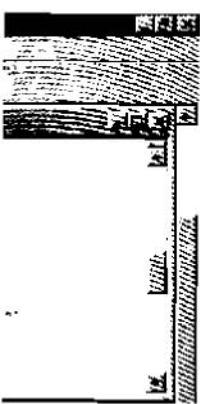
En el e  
R1ms\_BucleIn  
R\_ContA, en e  
retardo.

Carga co  
v  
Peque  
Decre

A). ESTRUCT  
DE RETAR

Es fácil de  
inicialmente el co  
viene expresado e

sta instrucción tarda  
cristal de cuarzo de



urrido

es resultado necesario  
os intervalos pueden  
simple de algunas  
conseguir el retardo  
rucción es conocido  
el registro R\_ContA,  
ibtenw el tiempo de

uiente fragmento de  
la figura 12-4(A) se

máquina.  
el valor de "6".

2 cm (al saltar).

máquina.

(K=249).

En el ejemplo anterior el bucle de las instrucciones `decfsz R_ContA,F` y `goto Rlms_BucleInterno` se repetirá tantas veces como determine el valor con el que se carga R\_ContA, en este caso 249. Cuanto mayor sea esta constante, mayor será el tiempo de retardo.

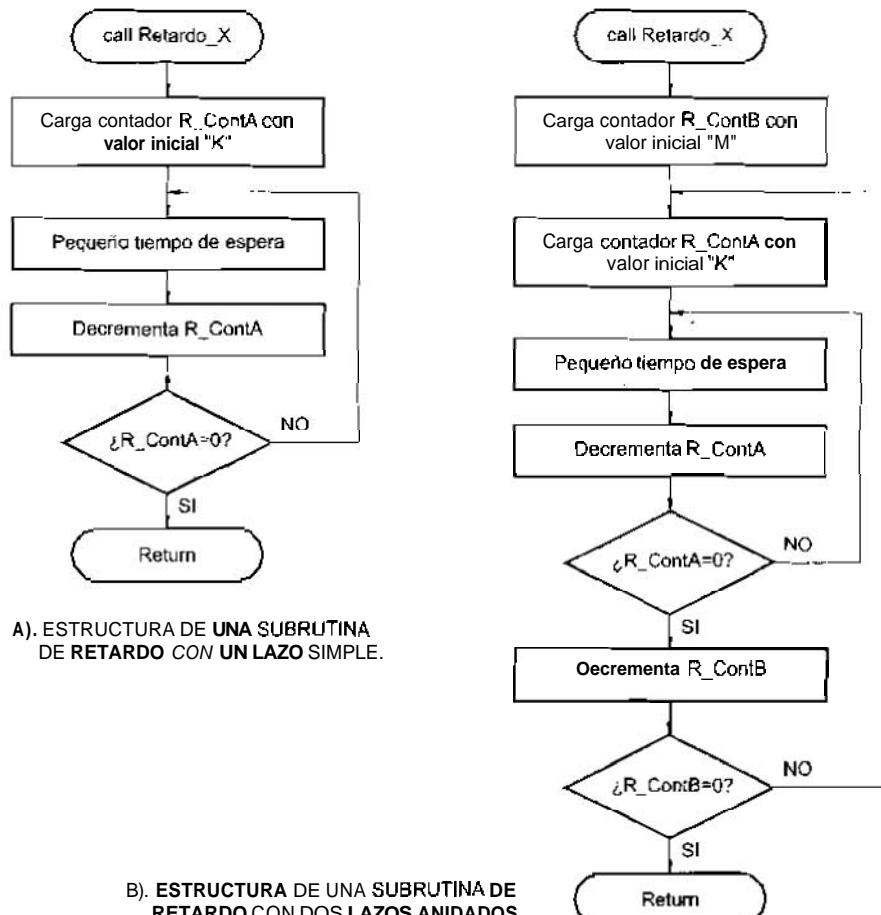


Figura 12-4 Estructura de las subrutinas de retardo

Es fácil deducir que el valor de la constante "K" con el que se ha cargado inicialmente el contador R\_ContA vendrá dado por la siguiente ecuación, donde el tiempo viene expresado en  $\mu s$ :

$$Tiempo = 5 + 4K \quad K = \frac{Tiempo - 5}{4}$$

**EJEMPLO:** Calcular el valor de la constante K, para obtener una subrutina de retardo de 500  $\mu$ s con la estructura de la figura 12-4(A).

**Solución:** Aplicando la ecuación se obtiene:

$$K = \frac{\text{Tiempo} - 5}{4} = \frac{500 - 5}{4} = 123,7$$

Así pues se elige K=123, obteniéndose un tiempo de retardo real de:

$$\text{Tiempo} = 5 + 4K = 5 + 4 \cdot 123 = 497 \mu\text{s}$$

El ajuste fino para los 500  $\mu$ s exactos se conseguiría añadiendo 3 instrucciones *nop* al principio de la subrutina de retardo.

## 12.5 RETARDOS MEDIANTE LAZOS ANIDADOS

Para la generación de retardos de mayor duración deben utilizarse lazos anidados, poniendo un lazo de retardo dentro de otro. La forma de hacerlo se explica en las subrutinas "Retardo\_200ms" y "Retardo\_100ms" del siguiente programa ejemplo, donde a partir de la estructura de 1 ms conseguido en la sección anterior se obtienen retardos mayores mediante la realización de lazos anidados, figura 12-4(B). Este programa es una útil aplicación de un LED intermitente para el circuito de la figura 12-5.

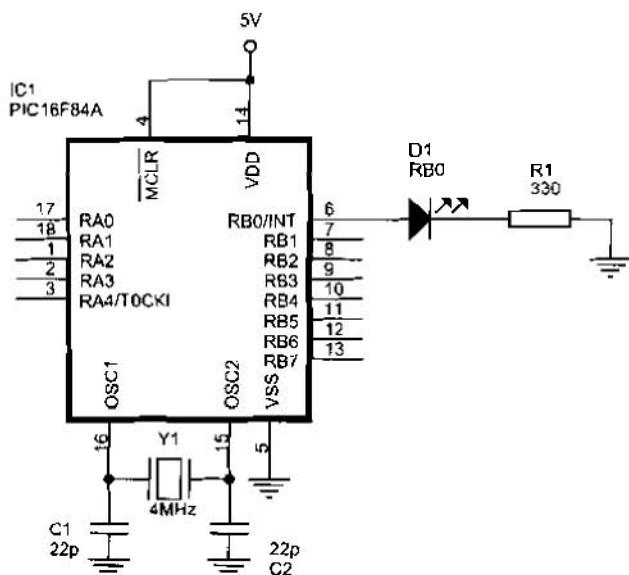


Figura 12-5. Intermitente

```

;*****
;
;< El LED conecta
;< apaga durante
;
; ZONA DE DEDALO
;< CO
;< LIST
;< INCLU
;< CBLOC
;< ENDC
;< #DEFINE LED
;< ; ZONA DE CÓDIGO
;< ORG
;< inicio
;< bsf
;< befc
;< befc
;< Principal
;< bsf
;< call
;< call
;< bcf
;< call
;< call
;< goto
;
;< ; Subrutinas "Retardo"
;< ;
;< CBLOC
;< R_Cond
;< R_Cond
;< ENDC
;< Retardo_200ms
;< movlw
;< goto
;< Retardo_100ms
;< movlw
;< goto
;< Retardo_1ms
;< movlw
;< ;
;< ; El próximo bloqu
;< ; 1 + M + M + Kx
;< ; = (2 + 4M + 4K)
;< ; que a 4 MHz son
;< ;
;< Retardos_ms
;< movwf

```

er una subrutina de

l de:

3 instrucciones *nop*

DOS

arse lazos anidados,  
o se explica en las  
una ejemplo, donde  
se obtienen retardos  
este programa es una  
).

```
;***** Retardo_01.asm *****
;
; El LED conectado a la línea 0 del puerto de salida se enciende durante 400 ms y se
; apaga durante 300 ms.
;
; ZONA DE DATOS *****

LIST P=16F84A
INCLUDE <P16F84A.INC>
CBLOCK 0x0C ; En esta posición empieza la RAM de usuario.
ENDC

#DEFINE LED PORTB,0

; ZONA DE CÓDIGOS *****
ORG 0
Inicio
    bsf STATUS,RP0 ; Acceso al Banco 1.
    bcf LED ; Línea del LED configurada como salida.
    bcf STATUS,RP0 ; Acceso al Banco 0.

Principal
    bsf LED ; Enciende el LED
    call Retardo_200ms ; durante la suma de este tiempo.
    call Retardo_200ms
    bcf LED ; Lo apaga durante la suma de los siguientes
    call Retardo_200ms ; retardos.
    call Retardo_100ms
    goto Principal

; Subrutinas "Retardo_200ms" y "Retardo_100ms"
;
CBLOCK
R_ContA ; Contadores para los retardos.
R_ContB
ENDC

Retardo_200ms
    movlw d'200' ; La llamada "call" aporta 2 ciclos máquina.
    goto Retardos_ms ; Aporta 1 ciclo máquina. Este es el valor de "M".
Retardo_100ms
    movlw d'100' ; Aporta 2 ciclos máquina.
    goto Retardos_ms ; La llamada "call" aporta 2 ciclos máquina.
                    ; Aporta 1 ciclo máquina. Este es el valor de "M".
Retardo_1ms
    movlw d'1' ; Aporta 2 ciclos máquina.
                    ; La llamada "call" aporta 2 ciclos máquina.
                    ; Aporta 1 ciclo máquina. Este es el valor de "M".
;

; El próximo bloque "Retardos_ms" tarda:
; 1 + M + M + KxM + (K-1)xM + Mx2 + (K-1)Mx2 + (M-1) + 2 + (M-1)x2 + 2 =
; = (2 + 4M + 4KM) ciclos máquina. Para K=249 y M=1 supone 1002 ciclos máquina
; que a 4 MHz son 1002 µs = 1 ms.
;

Retardos_ms
    movwf R_ContB ; Aporta 1 ciclo máquina.
```

```

R1ms_BucleExterno
    moviw d'249'
    movwf R_ContA
; Aporta Mx1 ciclos máquina. Este es el valor de "K".
; Aporta Mx1 ciclos máquina.

R1ms_BucleInterno
    nop
    decfsz R_ContA,F
    goto R1ms_BucleInterno
    decfsz R_ContB,F
    goto R1ms_BucleExterno
    return
; Aporta KxMx1 ciclos máquina.
; (K-1)xMx1 cm (cuando no salta) + Mx2 cm (al saltar).
; Aporta (K-1)xMx2 ciclos máquina.
; (M-1)x1 cm (cuando no salta) + 2 cm (al saltar).
; Aporta (M-1)x2 ciclos máquina.
; El salto de retorno aporta 2 ciclos máquina.

;En total estas subrutinas tardan:
;- Retardo_200ms:  $2 + 1 + 2 + (2 + 4M + 4KM) = 200007$  cm = 200 ms. (M=200 y K=249).
;- Retardo_100ms:  $2 + 1 + 2 + (2 + 4M + 4KM) = 100007$  cm = 100 ms. (M=100 y K=249).
;- Retardo_1ms :  $2 + 1 + (2 + 4M + 4KM) = 1005$  cm = 1 ms. (M= 1 y K=249).

    END

```

La estructura de lazos anidados descrita en la figura 12-4(B) funciona como contadores en cuenta descendente que permiten multiplicar entre sí los retardos producidos por cada bucle. De esta forma se puede obtener una temporización fácilmente controlable mediante el dato que se carga en cada uno de los registros que actúan como contadores.

## 12.6 LIBRERÍA CON SUBRUTINAS DE RETARDOS

Los retardos son probablemente las subrutinas más utilizadas dentro de los programas. Es por ello importante disponer de una buena librería de subrutinas con numerosos tiempos de temporización. A continuación se expone una librería con subrutinas de retardo de 4  $\mu$ s hasta 20 segundos denominada RETARDOS.INC suficientemente documentada y que se utilizará en casi todos los programas restantes de este libro.

```

***** Librería "RETARDOS.INC" *****
;
; Librería con múltiples subrutinas de retardos, desde 4 microsegundos hasta 20 segundos.
; Además se pueden implementar otras subrutinas muy fácilmente.
;
; Se han calculado para un sistema microcontrolador con un PIC trabajando con un cristal
; de cuarzo a 4 MHz. Como cada ciclo máquina son 4 ciclos de reloj, resulta que cada
; ciclo máquina tarda  $4 \times 1/4\text{MHz} = 1 \mu\text{s}$ .
;
; En los comentarios, "cm" significa "ciclos máquina".
;
; ZONA DE DATOS *****

```

```

CBLOCK
R_ContA
R_ContB
R_ContC
ENDC
;
```

; Contadores para los retardos.

```

; RETARDOS de
; ; A continuación
; ; la llamada a sub
; ; "return" toma o
; ; Retardo_10micro
;     nop
;     nop
;     nop
;     nop
;     nop
;     nop
;     Retardo_5micro
;         nop
;     Retardo_4micro
;         return
; ; RETARDOS de
; ; Retardo_500micro
;     nop
;     movlw
;     goto
; Retardo_200micro
;     nop
;     movlw
;     goto
; Retardo_100micro
;     movlw
;     goto
; Retardo_50micro
;     nop
;     movlw
;     goto
; Retardo_20micro
;     movlw
; ; El próximo bloq
; ;  $1 + (K-1) + 2 + ($ 
; ; RetardoMicros
;     movwf
; Rmicros_Bucle
;     decfsz
;     goto
;     return
; ;En total estas sub
; ; - Retardo_500mi
; ; - Retardo_200mi
; ; - Retardo_100mi
; ; - Retardo_50mic
; ; - Retardo_20mic

```

```

; RETARDOS de 4 hasta 10 microsegundos -----
;

; A continuación retardos pequeños teniendo en cuenta que para una frecuencia de 4 MHZ,
; la llamada a subrutina "call" tarda 2 ciclos máquina, el retorno de subrutina
; "return" toma otros 2 ciclos máquina y cada instrucción "nop" tarda 1 ciclo máquina.
;

Retardo_10micros           ; La llamada "call" aporta 2 ciclos máquina.
    nop                      ; Aporta 1 ciclo máquina.
    Retardo_5micros          ; La llamada "call" aporta 2 ciclos máquina.
    nop                      ; Aporta 1 ciclo máquina.
    Retardo_4micros          ; La llamada "call" aporta 2 ciclos máquina.
    return                   ; El salto del retorno aporta 2 ciclos máquina.

;

; RETARDOS de 20 hasta 500 microsegundos -----
;

Retardo_500micros          ; La llamada "call" aporta 2 ciclos máquina.
    nop                      ; Aporta 1 ciclo máquina.
    movlw d'164'             ; Aporta 1 ciclo máquina. Éste es el valor de "K".
    goto RetardoMicros      ; Aporta 2 ciclos máquina.
    Retardo_200micros        ; La llamada "call" aporta 2 ciclos máquina.
    nop                      ; Aporta 1 ciclo máquina.
    movlw d'64'               ; Aporta 1 ciclo máquina. Éste es el valor de "K".
    goto RetardoMicros      ; Aporta 2 ciclos máquina.
    Retardo_100micros        ; La llamada "call" aporta 2 ciclos máquina.
    movlw d'31'               ; Aporta 1 ciclo máquina. Éste es el valor de "K".
    goto RetardoMicros      ; Aporta 2 ciclos máquina.
    Retardo_50micros         ; La llamada "call" aporta 2 ciclos máquina.
    nop                      ; Aporta 1 ciclo máquina.
    movlw d'14'               ; Aporta 1 ciclo máquina. Éste es el valor de "K".
    goto RetardoMicros      ; Aporta 2 ciclos máquina.
    Retardo_20micros         ; La llamada "call" aporta 2 ciclos máquina.
    movlw d'5'                ; Aporta 1 ciclo máquina. Éste es el valor de "K".

;

; El próximo bloque "RetardoMicros" tarda:
;  $1 + (K-1) + 2 + (K-1)x2 + 2 = (2 + 3K)$  ciclos máquina.
;

RetardoMicros
    movwf R_ContA            ; Aporta 1 ciclo máquina.
Rmicros_Bucle
    decfsz R_ContA,F          ;  $(K-1)x1$  cm (cuando no salta) + 2 cm (al saltar).
    goto Rmicros_Bucle        ; Aporta  $(K-1)x2$  ciclos máquina.
    return                    ; El salto del retorno aporta 2 ciclos máquina.

;

; En total estas subrutinas tardan:
; - Retardo_500micros:    $2 + 1 + 1 + 2 + (2 + 3K) = 500$  cm = 500  $\mu$ s. (para K=164 y 4 MHz).
; - Retardo_200micros:    $2 + 1 + 1 + 2 + (2 + 3K) = 200$  cm = 200  $\mu$ s. (para K= 64 y 4 MHz).
; - Retardo_100micros:    $2 + 1 + 1 + 2 + (2 + 3K) = 100$  cm = 100  $\mu$ s. (para K= 31 y 4 MHz).
; - Retardo_50micros :    $2 + 1 + 1 + 2 + (2 + 3K) = 50$  cm = 50  $\mu$ s. (para K= 14 y 4 MHz).
; - Retardo_20micros :    $2 + 1 + 1 + 2 + (2 + 3K) = 20$  cm = 20  $\mu$ s. (para K= 5 y 4 MHz).

```

```

;
; RETARDOS de 1 ms hasta 200 ms.
;

Retardo_200ms
    moviw d'200'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_100ms
    movlw d'100'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_50ms
    movlw d'50'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_20ms
    movlw d'20'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_10ms
    movlw d'10'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_5ms
    movlw d'5'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_2ms
    movlw d'2'
    goto Retardos_ms
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".
; Aporta 2 ciclos máquina.

Retardo_1ms
    moviw d'1'
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "M".

;
; El próximo bloque "Retardos_ms" tarda:
; 1 + M + M + KxM + (K-1)xM + Mx2 + (K-1)Mx2 + (M-1) + 2 + (M-1)x2 + 2 =
; = (2 + 4M + 4KM) ciclos máquina. Para K=249 y M=1 supone 1002 ciclos máquina
; que a 4 MHz son 1002  $\mu$ s = 1 ms.

;
Retardos_ms
    movwf R_ContB
; Aporta 1 ciclo máquina.

R1ms_BucleExterno
    movlw d'249'
    movwf R_ContA
; Aporta Mx1 ciclos máquina. Éste es el valor de "K".
; Aporta Mx1 ciclos máquina.

R1ms_BucleInterno
    nop
; Aporta KxMx1 ciclos máquina.
    decfsz R_ContA,F
; (K-1)xMx1 cm (cuando no salta) + Mx2 cm (al saltar).
    goto R1ms_BucleInterno
; Aporta (K-1)xMx2 ciclos máquina.
    decfsz R_ContB,F
; (M-1)x1 cm (cuando no salta) + 2 cm (al saltar).
    goto R1ms_BucleExterno
; Aporta (M-1)x2 ciclos máquina.
    return
; El salto del retorno aporta 2 ciclos máquina.

;
; En total estas subrutinas tardan:
; - Retardo_200ms: 2 + 1 + 2 + (2 + 4M + 4KM) = 200007 cm = 200 ms. (M=200 y K=249).
; - Retardo_100ms: 2 + 1 + 2 + (2 + 4M + 4KM) = 100007 cm = 100 ms. (M=100 y K=249).
; - Retardo_50ms : 2 + 1 + 2 + (2 + 4M + 4KM) = 50007 cm = 50 ms. (M= 50 y K=249).
; - Retardo_20ms : 2 + 1 + 2 + (2 + 4M + 4KM) = 20007 cm = 20 ms. (M= 20 y K=249).
; - Retardo_10ms : 2 + 1 + 2 + (2 + 4M + 4KM) = 10007 cm = 10 ms. (M= 10 y K=249).
; - Retardo_5ms : 2 + 1 + 2 + (2 + 4M + 4KM) = 5007 cm = 5 ms. (M= 5 y K=249).
; - Retardo_2ms : 2 + 1 + 2 + (2 + 4M + 4KM) = 2007 cm = 2 ms. (M= 2 y K=249).

;
; - Retardo_1s
; - RETARDOS
; - Retardo_20s
;     mov
;     goto
; Retardo_10s
;     mov
;     goto
; Retardo_5s
;     mov
;     goto
; Retardo_2s
;     mov
;     goto
; Retardo_1s
;     mov
;     goto
; Retardo_500m
;     mov
; ;
; ; El próximo b
; ; 1 + N + N + N
; ; + (M-1)xN -
; ; = (2 + 4M + 4
; ; ciclos máqui
; ;
; Retardo_1Deci
;     mov
; R1Decima_Buc
;     mov
;     mov
; R1Decima_Buc
;     mov
;     mov
; R1Decima_Buc
;     nop
;     decfs
;     goto
;     decfs
;     goto
;     decfs
;     goto
;     decfs
;     goto
;     return
; ;
; ; En total estas s
; - Retardo_20s:
; - Retardo_10s:
; - Retardo_5s:

```

```

; - Retardo_1ms : 2 + 1 + (2 + 4M + 4KM) = 1005 cm = 1 ms. (M= 1 y K=249).
;
; RETARDOS de 0.5 hasta 20 segundos
;
Retardo_20s      movlw d'200'
                  goto Retardo_1Decima
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "N".
; Aporta 2 ciclos máquina.
Retardo_10s       movlw d'100'
                  goto Retardo_1Decima
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "N".
; Aporta 2 ciclos máquina.
Retardo_5s        movlw d'50'
                  goto Retardo_1Decima
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "N".
; Aporta 2 ciclos máquina.
Retardo_2s        movlw d'20'
                  goto Retardo_1Decima
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "N".
; Aporta 2 ciclos máquina.
Retardo_1s         movlw d'10'
                  goto Retardo_1Decima
; La llamada "call" aporta 2 ciclos máquina.
; Aporta 1 ciclo máquina. Éste es el valor de "N".
; Aporta 2 ciclos máquina.
Retardo_500ms     movlw d'5'
;
; El próximo bloque "Retardo_1Decima" tarda:
; 1 + N + N + MxN + MxN + KxMxN + (K-1)xMxN + MxNx2 + (K-1)xMxNx2 +
; + (M-1)xN + Nx2 + (M-1)xNx2 + (N-1) + 2 + (N-1)x2 + 2 =
; = (2 + 4M + 4MN + 4KM) ciclos máquina. Para K=249, M=100 y N=1 supone 100011
; ciclos máquina que a 4 MHz son 100011 µs = 100 ms = 0,1 s = 1 decima de segundo.
;
Retardo_1Decima   movwf R_ContC
; Aporta 1 ciclo máquina.
R1Decima_BucleExterno2
                  movlw d'100'
                  movwf R_ContB
; Aporta Nx1 ciclos máquina. Éste es el valor de "M".
; Aporta Nx1 ciclos máquina.
R1Decima_BucleExterno
                  movlw d'249'
                  movwf R_ContA
; Aporta MxNx1 ciclos máquina. Éste es el valor de "K".
; Aporta MxNx1 ciclos máquina.
R1Decima_BucleInterno
                  nop
                  decfsz R_ContA,F
; Aporta KxMxNx1 ciclos máquina.
; (K-1)xMxNx1 cm (si no salta) + MxNx2 cm (al saltar).
                  goto R1Decima_BucleInterno
                  decfsz R_ContB,F
; Aporta (K-1)xMxNx2 ciclos máquina.
; (M-1)xNx1 cm (cuando no salta) + Nx2 cm (al saltar).
                  goto R1Decima_BucleExterno
                  decfsz R_ContC,F
; Aporta (M-1)xNx2 ciclos máquina.
; (N-1)x1 cm (cuando no salta) + 2 cm (al saltar).
                  goto R1Decima_BucleExterno2
                  return
; Aporta (N-1)x2 ciclos máquina.
; El salto del retorno aporta 2 ciclos máquina.

; En total estas subrutinas tardan:
; - Retardo_20s: 2 + 1 + 2 + (2 + 4N + 4MN + 4KM) = 20000807 cm = 20 s.
; (N=200, M=100 y K=249).
; - Retardo_10s: 2 + 1 + 2 + (2 + 4N + 4MN + 4KM) = 10000407 cm = 10 s.
; (N=100, M=100 y K=249).
; - Retardo_5s: 2 + 1 + 2 + (2 + 4N + 4MN + 4KM) = 5000207 cm = 5 s.
; (N= 50, M=100 y K=249).
;
```

; - Retardo\_2s:  $2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) = 2000087 \text{ cm} = 2 \text{ s.}$   
                  (N= 20, M=100 y K=249).  
; - Retardo\_1s:  $2 + 1 + 2 + (2 + 4N + 4MN + 4KMN) = 1000047 \text{ cm} = 1 \text{ s.}$   
                  (N= 10, M=100 y K=249).  
; - Retardo\_500ms:  $2 + 1 + (2 + 4N + 4MN + 4KMN) = 500025 \text{ cm} = 0,5 \text{ s.}$   
                  (N= 5, M=100 y K=249).

Como ejemplo de aplicación se resuelve el programa Retardo\_01.asm de la sección anterior referido al intermitente de la figura 12-5 utilizando esta librería:

```
***** Retardo_02.asm *****
;
; El LED conectado a la línea 0 del puerto de salida se enciende durante 400 ms y se
; apaga durante 300 ms. Utiliza las subrutinas de la librería RETARDOS.INC.
;
; ZONA DE DATOS *****
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

#DEFINE LED PORTB,0

; ZONA DE CÓDIGOS *****
ORG 0
Inicio
    bsf STATUS,RP0      ; Acceso al Banco 1.
    bcf LED             ; Línea del LED configurada como salida.
    bcf STATUS,RP0      ; Acceso al Banco 0.

Principal
    bsf LED             ; Enciende el LED
    call Retardo_200ms  ; durante la suma de este tiempo.
    call Retardo_200ms
    bcf LED             ; Lo apaga durante la suma de los siguientes
    call Retardo_200ms  ; retardos.
    call Retardo_100ms
    goto Principal

INCLUDE <RETARDOS.INC>      ; Libreria con subrutinas de retardo.
END                         ; Fin del programa.
```

Se observa que el programa gana en claridad gracias a la subrutinas y a la directiva `INCLUDE <RETARDOS.INC>`. Además, es mucho más reducido que si las subrutinas se hubieran incluido directamente, aunque en la memoria de programa ocupan lo mismo.

En algunas introducidas por el problema durante un punto y comprobado el código las subrutinas eliminadas.

## 12.7 REBO

En casi todos los mecanismos para la normalmente abierta



Figura 12-7

El propósito de este circuito dependiendo si está abierto o cerrado el interruptor S1 figura 12-7 donde se observa:

- Cuando el interruptor S1 está cerrado.
- Cuando el interruptor S1 está abierto.

punto A

En algunas simulaciones de programas con MPLAB, las temporizaciones introducidas por los retardos de tiempo pueden resultar muy incomodas. Para evitar este problema durante la simulación hay que inhabilitar las subrutinas de retardo anteponiendo un punto y coma, “;”, delante de cada llamada a subrutina de retardo. Una vez comprobado el correcto funcionamiento del programa habría que volver a habilitar dichas subrutinas eliminando los puntos y comas insertados.

sm de la sección

## 12.7 REBOTES EN LOS PULSADORES

En casi todos los proyectos de sistemas digitales habrá ocasión de utilizar contactos mecánicos para la generación de impulsos eléctricos. Ejemplo de esto, es el pulsador normalmente abierto utilizado en el circuito de la figura 12-6.

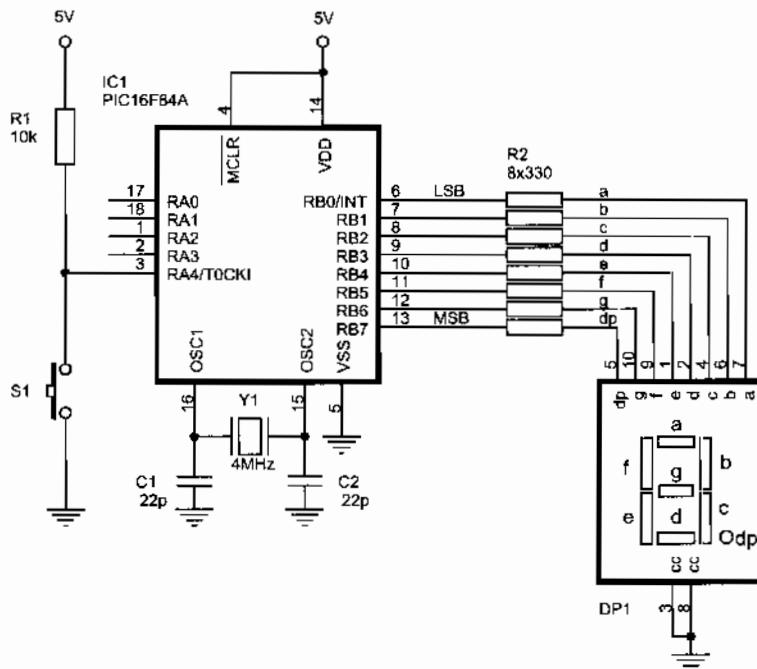


Figura 12-6 Pulsador a una entrada del microcontrolador

El propósito de este pulsador es aplicar un nivel lógico bajo o un nivel lógico alto, dependiendo si está o no pulsado. La señal producida por el pulsador se detalla en la figura 12-7 donde se aprecia como:

- Cuando no se actúa sobre el pulsador la tensión en el punto A es +5 V.
- Cuando el pulsador es presionado los contactos se cierran y la tensión en el punto A debe ser 0 V.

Idealmente, la forma de onda de tensión en A debe aparecer como se indica en la forma de onda superior de la figura 12-7. Sin embargo, en realidad la forma de onda del punto A aparecerá con una serie de **rebotes** tal como se especifica en la misma figura. Esto es debido al fenómeno de **rebote de contacto**. Cualquier dispositivo mecánico de conmutación consiste en un brazo móvil de contacto dotado de alguna clase de sistema de muelle. Cuando el brazo se mueve de una posición estable a otra el brazo rebota, lo mismo que cuando una bola de materia dura choca con una superficie también dura. El número de rebotes que ocurren en el período de rebote difiere en cada dispositivo de conmutación y de la fuerza con que se actúa sobre el pulsador.

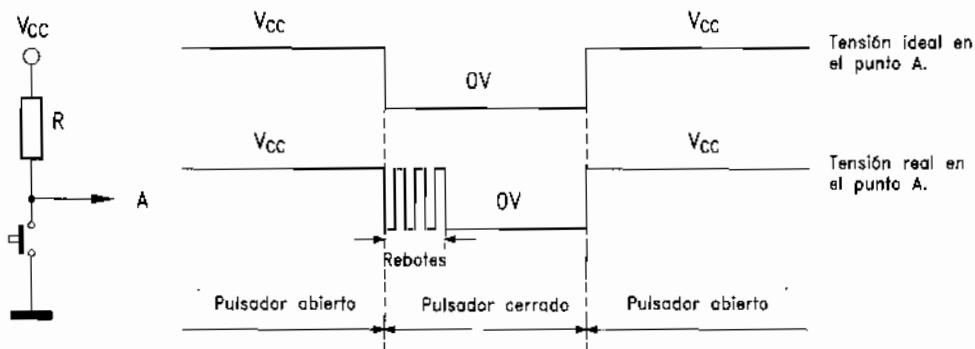


Figura 12-7 Todo pulsador produce rebotes

Si la tensión en el punto A se aplica a la entrada de un circuito digital el circuito responderá adecuadamente cuando el pulsador esté abierto, ya que no ocurre rebote alguno del contacto. Sin embargo, cuando el pulsador se presiona, el circuito responderá como si fuesen aplicadas señales múltiples, en vez de haber un solo cierre del pulsador, con resultados impredecibles e indeseables.

Estos rebotes se pueden eliminar mediante circuitos hardware (figura 32-25), sin embargo, la mayoría de las veces es suficiente con añadir un pequeño retardo en el programa cada vez que se detecta un cambio de posición en el pulsador y volver a leer el pulsador una vez pasado este tiempo. El siguiente programa es un ejemplo de ello para el circuito de la figura 12-6. La figura 12-8 detalla el funcionamiento mediante su diagrama de flujo.

```
***** Pulsador_01.asm *****
;
; Cada vez que presione el pulsador conectado al pin RA4 incrementa un contador visualizado en el display.
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

CBL  
Cont  
END  
  
#DEFINE Pulse  
#DEFINE Disp  
; ZONA DE CO  
  
ORG  
Inicio  
bsf  
clr  
bsf  
bcf  
call  
call  
Principal  
btfs  
goto  
call  
btfs  
goto  
call  
EsperaDejePulsa  
btfs  
goto  
Fin  
goto  
;  
Subrutina "Incre  
IncrementaVisual  
inef  
movlw  
subwf  
btfs  
InicializaContador  
cirf  
Visualiza  
movf  
call  
movwf  
return  
INCLU  
INCLU  
END  
  
Hay otra  
carrera, detecta

se indica en la  
na de onda del  
mismo figura.  
o mecánico de  
e de sistema de  
bota, lo mismo  
ura. El número  
e conmutación

tensión ideal en  
l punto A.

tensión real en  
l punto A.

ital el circuito  
ocurre rebote  
ito responderá  
e del pulsador,

ra 32-25), sin  
retardo en el  
olver a leer el  
de ello para el  
e su diagrama

\*\*\*\*\*  
en el display.  
\*\*\*\*\*

```

CBLOCK 0x0C
Contador
ENDC

; El contador a visualizar.

#DEFINE Pulsador PORTA,4
#DEFINE Display PORTB
; Pulsador conectado a RA4.
; El display está conectado al Puerto B.

; ZONA DE CÓDIGOS ****
; ****

ORG 0
; El programa comienza en la dirección 0.

Inicio
    bsf STATUS,RP0
    ; Acceso al Banco 1.
    clrf Display
    ; Estas líneas configuradas como salidas.
    bsf Pulsador
    bcf STATUS,RP0
    ; Línea del pulsador configurada como entrada.
    call InicializaContador
    ; Acceso al Banco 0.
    call Visualiza
    ; Inicializa el Contador y lo visualiza.

Principal
    btfsc Pulsador
    ; ¿Pulsador presionado?, ¿(Pulsador) = 0?
    goto Fin
    ; No. Vuelve a leerlo.
    call Retardo_20ms
    ; Espera que se establezcan los niveles de tensión.
    btfsc Pulsador
    ; Comprueba si es un rebote.
    goto Fin
    ; Era un rebote y sale fuera.
    call IncrementaVisualiza
    ; Incrementa el contador y lo visualiza.

EsperaDejePulsar
    btfss Pulsador
    ; ¿Dejó de pulsar?, ¿(Pulsador) = 1?
    goto EsperaDejePulsar
    ; No. Espera que deje de pulsar.

Fin
    goto Principal

; Subrutina "IncrementaVisualiza"
; ----

IncrementaVisualiza
    incf Contador,F
    moviw d'10'
    subwf Contador,W
    btfsc STATUS,C
    ; Incrementa el contador y comprueba si ha
    ; llegado a su valor máximo mediante una
    ; resta. (W)=(Contador)-10.
    ; ¿C=0?, ¿(W) negativo?, ¿(Contador)<10?

InicializaContador
    clrf Contador
    ; No, era igual o mayor. Por tanto, resetea.

Visualiza
    movf Contador,W
    call Numero_a_7Segmentos
    movwf Display
    ; Lo pasa a siete segmento para poder ser
    ; visualizado en el display.
    return

INCLUDE <DISPLAY_7S.INC>
INCLUDE <RETARDOS.INC>
END
; Subrutina Numero_a_7Segmentos.
; Subrutinas de retardo.
; Fin del programa.

```

Hay otros muchos dispositivos como interruptores, conmutadores, finales de carrera, detectores digitales, etc, que funcionan de forma similar a los pulsadores.

## 12.8 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular, grabar el microcontrolador y comprobar los siguientes programas para el esquema de la figura 1-2. El lector puede introducir todas las mejoras que considere conveniente.

**Retardo\_01.asm:** El LED conectado a la línea 0 del Puerto B se enciende durante 400 ms y se apaga durante 300 ms.

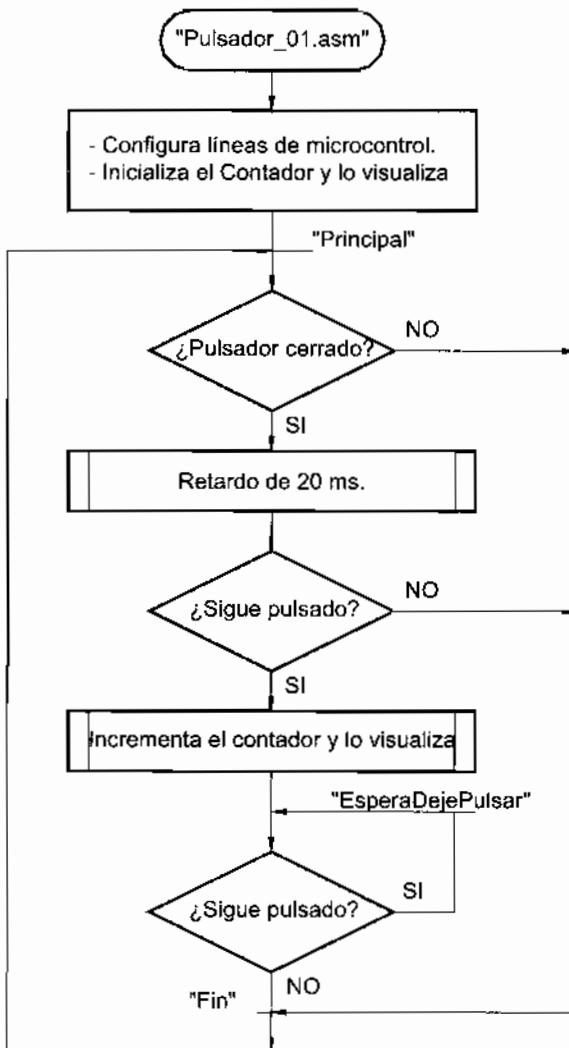


Figura 12-8 Diagrama de flujo del programa para pulsador antirrebote

iar, ensamblar,  
ramas para el  
que considere

ciede durante

rebote

**Retardo\_02.asm:** Repetir el programa anterior utilizando las subrutinas de la librería RETARDOS.INC.

**Retardo\_03.asm:** Los diodos pares conectados al Puerto B se encienden durante 0,5 segundos y los impares permanecen apagados. Después se invierten durante el mismo tiempo y se repite el ciclo indefinidamente.

**Retardo\_04.asm:** Por la barra de LEDs conectada al Puerto B un LED encendido rota a la izquierda durante 0,3 s en cada posición. Cuando llega al final se apagan todos los LEDs y de nuevo repite la operación. Observe el curioso efecto visualizado en el display, ¿le dá nuevas ideas al lector?

**Retardo\_05.asm:** Por la barra de LEDs conectada al Puerto B un LED encendido rota a la izquierda durante 0,3 s en cada posición. Cuando llega al final comienza a rotar a derechas durante 0,5 s en cada posición. La operación se repite indefinidamente.

**Retardo\_06.asm:** Por la barra de LEDs conectada al Puerto B un LED encendido rota a la izquierda durante 0,5 s en cada posición empezando por la línea D0. El número de posiciones a desplazar lo fija el valor de las tres primeras líneas del puerto de entrada. Así por ejemplo, si por el puerto de entrada se lee el dato “--00011” (binario) (3 decimal), la secuencia de salida sería .., 00000000, 00000001, 00000010, 00000100, 00000000, 00000001, 00000010,... ( y se repite indefinidamente).

**Retardo\_07.asm:** Por la barra de LEDs conectada al Puerto B un LED encendido rota a la izquierda durante 0,2 s cada posición. Cuando llega al final se apagan todos los LEDs y de nuevo repite la operación. Hay que realizarlo mediante el manejo de una tabla. Este ejercicio se ha implementado anteriormente utilizando la instrucción *rif*.

**Retardo\_08.asm:** Por la barra de LEDs conectada al puerto de salida se visualiza un juegos de luces que al lector le pueda resultar divertido. Hay que utilizar una tabla de datos. Observe el curioso efecto visualizado en el display, ¿le dá nuevas ideas al lector?

**Retardo\_09.asm:** El display debe visualizar un mensaje grabado en la memoria ROM mediante la directiva *DT*. Se utilizará la misma técnica que para la visualización de los juegos de luces de los programas anteriores.

**Retardo\_10.asm:** El display debe visualizarse un mensaje grabado en la memoria ROM mediante la directiva *DT*. En lugar de medir la longitud como en programas anteriores, detectará el fin de mensaje mediante el código 0x00 grabado al final.

**Retardo\_11.asm:** El display visualiza un contador descendente que cuenta desde la cantidad leída por el Puerto A hasta cero y vuelve a repetir. Cada dígito se visualizará durante un segundo. Por ejemplo, si por el puerto de entrada se lee “--00101” en el

display se visualizarán las cantidades: 5, 4, 3, 2, 1, 0, 5, 4, 3, ... Si en la entrada se lee una cantidad mayor de 9 o un 0 se encenderá únicamente el punto decimal.

**Retardo\_12.asm:** Si el bit 0 del puerto de entrada es “0” lógico por el display se visualizará un contador descendente (9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 9, ..) con una cadencia de 0,5 s aproximadamente, si es “1” por el display se visualizará un contador ascendente (0, 1, 2, 3, 5, 6, 7, 8, 9, 0, 1, 2, ..) con una cadencia de 0,5 s.

**Retardo\_13.asm:** Por la barra de LEDs conectada al puerto de salida se visualiza un juegos de luces con la secuencia que el lector determine. La velocidad del movimiento será fijada por la lectura de las tres líneas conectadas al puerto A, de manera que se visualice cada posición durante un tiempo:

- Si (PORTA)=0, cada posición se visualiza durante  $0 \times 100 \text{ ms} = 0 \text{ ms}$ . (Apagado).
- Si (PORTA)=1, cada posición se visualiza durante  $1 \times 100 \text{ ms} = 100 \text{ ms}$  aproximadamente.
- Si (PORTA)=2, cada posición se visualiza durante  $2 \times 100 \text{ ms} = 200 \text{ ms}$  aproximadamente.
- .... y así sucesivamente hasta...
- Si (PORTA)=7, cada posición se visualiza durante  $7 \times 100 \text{ ms} = 700 \text{ ms}$  aproximadamente.

**Pulsador\_01.asm:** Por el display aparecerá las veces que se activa el pulsador conectado al pin RA4. Cuando llegue a 10 la cuenta se reseteará y comenzará de nuevo. Debe haber un pequeño tiempo de temporización que evite el efecto rebote del pulsador. (Hay que asegurarse que el interruptor que está en paralelo con el pulsador no está cerrado).

**Pulsador\_02.asm:** Mientras se mantenga activado el pulsador del pin RA4, en el display contará de 0 a 9 continuamente, manteniéndose 200 ms en cada valor. Cuando deje de estar pulsador permanecerá el último valor visualizado. (Asegurarse que el interruptor que está en paralelo con el pulsador no está cerrado).

**Pulsador\_03.asm:** Simula un dado electrónico que cuenta de 1 a 6. Mientras se mantenga activado un pulsador del pin RA4, el display contará de 1 a 6 continuamente, manteniéndose un instante en cada valor. Cuando deje de estar pulsador permanecerá el último valor visualizado. En este caso no es importante el tema de los rebotes. (Hay que asegurarse que el interruptor que está en paralelo con el pulsador no está cerrado).

En muchos  
de bajo coste. La  
este capítulo.

## 13.1 VISUALIZACIÓN

Las pantallas de tipo *Dot Matrix* (*Display*) tienen la ventaja de ser más económicas que las matrices de líneas. Representan la información en forma de imágenes (figuras, textos, etc.) mediante un grupo de puntos (figura 13.1). Una matriz de 5x7 puntos) distribuidos en una sola línea. El proceso de generación de la imagen es más sencillo que en la pantalla de líneas, siendo el tiempo de respuesta menor.

Falsa

**CAPÍTULO 13****LCD**

En muchos proyectos es necesario visualizar información a través de una pantalla de bajo coste. La forma más utilizada es mediante un display LCD tal como se explica en este capítulo.

### 13.1 VISUALIZADOR LCD

Las pantallas de cristal líquido o display **LCD** para mensajes (*Liquid Crystal Display*) tienen la capacidad de mostrar cualquier carácter alfanumérico, permitiendo representar la información que genera cualquier equipo electrónico de una forma fácil y económica (figura 13-1). La pantalla consta de una matriz de caracteres (normalmente de 5x7 puntos) distribuidos en una, dos, tres o cuatro líneas de 16 hasta 40 caracteres cada línea. El proceso de visualización es gobernado por un microcontrolador incorporado a la pantalla, siendo el Hitachi 44780 el modelo más utilizado.

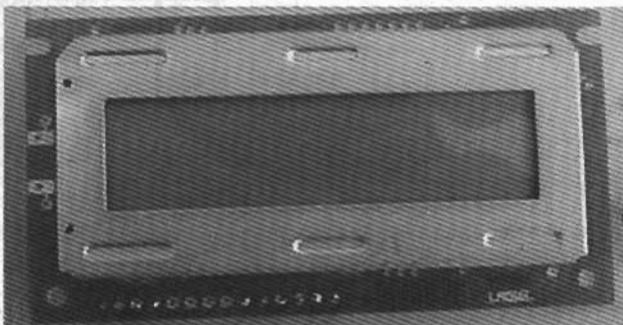


Figura 13-1 Aspecto de un módulo LM016L

Distintos fabricantes ofrecen multitud de versiones de visualizadores de cristal líquido. El modelo utilizado en este libro es el LM016L, que es un módulo LCD de dos líneas de 16 caracteres cada una. Su fácil manejo lo hace ideal para dispositivos que necesitan una capacidad de visualización pequeña o media. Las características generales de un módulo LM016L son:

- Consumo muy reducido, del orden de 7,5 mW.
  - Pantalla de caracteres ASCII, además de los caracteres japoneses Kanji, caracteres griegos y símbolos matemáticos.
  - Desplazamiento de los caracteres hacia la izquierda o a la derecha.
  - Memoria de 40 caracteres por línea de pantalla, visualizándose 16 caracteres por línea.
  - Movimiento del cursor y cambio de su aspecto.
  - Permite que el usuario pueda programar ocho caracteres.
  - Pueden ser gobernados de dos formas principales:
    - Conexión con bus de 4 bits.
    - Conexión con bus de 8 bits.

Lo que a continuación se explica se refiere al modelo LM016L y con pequeñas variaciones es también válido para cualquier otro.

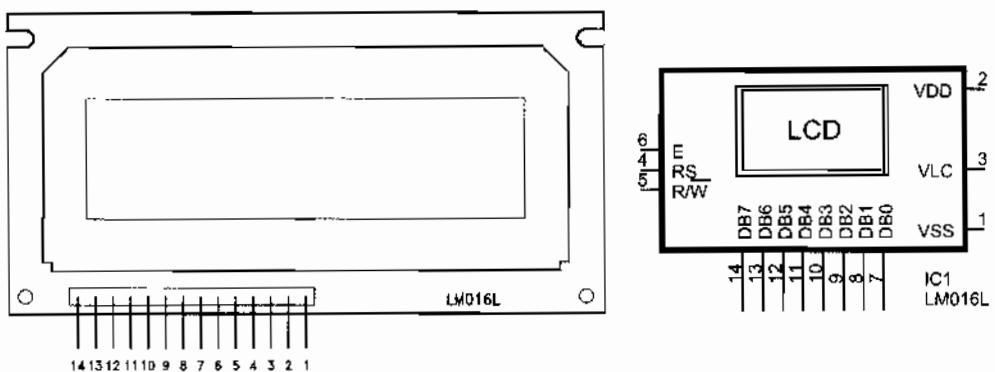


Figura 13-2 Patillaje del LCD LM016L

La alí voltaje obteni V <sub>LC</sub> , como de y conecta V <sub>LC</sub>
SEÑAL
DB0...DE
E
R/W
RS
V <sub>LC</sub>
V <sub>DD</sub>
V <sub>SS</sub>

### 13.3 DDI

El LMC  
RAM) donde se  
de 80 bytes, 40  
bytes por línea  
que aparecerán  
posiciones con

En pantalla			
00	01	02	03
40	41	42	43

La DDE

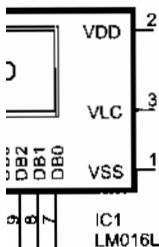
dores de cristal  
ulo LCD de dos  
dispositivos que  
ísticas generales

pones Kanji,

6 caracteres por

y con pequeñas

tabla 13-1. Se  
estra en la figura  
ancia cuando el



La alimentación es de + 5V. La regulación de contraste se realiza mediante el voltaje obtenido al dividir los 5 V con una resistencia ajustable de 10 k y aplicárselo al pin  $V_{LC}$ , como describe la figura 13-5. En algunos proyectos se elimina la resistencia ajustable y conecta  $V_{LC}$  a masa fijando el máximo contraste permanentemente.

SEÑAL	DEFINICIÓN	PINES	FUNCIÓN
DB0....DB7	<i>Data Bus</i>	7...14	Bus de Datos.
E	<i>Enable</i>	6	$E=0$ , LCD no habilitado. $E=1$ , LCD habilitado.
R/W	<i>Read/Write</i>	5	$R/W=0$ , escribe en LCD. $R/W=1$ , lee del LCD.
RS	<i>Register Select</i>	4	$R/S=0$ , Modo Comando. $R/S=1$ , Modo Carácter.
$V_{LC}$	<i>Liquid Crystal driving Voltage</i>	3	Tensión para ajustar el contraste.
V <sub>DD</sub>	<i>Power Supply Voltage</i>	2	Tensión de alimentación, +5V
V <sub>SS</sub>	<i>Ground</i>	1	Masa.

Tabla 13-1 Función de los pines en un LM016L

### 13.3 DDRAM

El LM016L posee una zona de memoria RAM llamada **DDRAM** (*Data Display RAM*) donde se almacenan los caracteres que se pueden representar. Tiene una capacidad de 80 bytes, 40 por cada línea, de los cuales sólo 32 se pueden visualizar a la vez (16 bytes por línea), figura 13-3. La DDRAM almacena los códigos ASCII de los caracteres que aparecerán en pantalla y existe una correspondencia entre las filas de la pantalla y las posiciones consecutivas de memoria.

En pantalla se visualizan 32 caracteres: 16 de cada fila															
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

← FILA 0

← FILA 1

La DDRAM tiene un tamaño de 80 bytes (40 en cada fila), de los cuales se visualizan 32

Figura 13-3 DDRAM

De las 80 posibles, las dos direcciones más importantes de la DDRAM son:

- Dirección 00h, que es el comienzo de la primera línea.
- Dirección 40h, que es el comienzo de la segunda línea.

Cada vez que se escribe un dato en la DDRAM automáticamente se apunta a la siguiente posición, donde se realizará la escritura del próximo carácter.

Lower C pos: 4 bits	Upper C pos: 4 bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)	0	ä	P	^	F				-	タ	ミ	。	o	p		
xxxx0001	(2)	!	1	A	Q	a	q			。	ア	チ	カ	ä	q		
xxxx0010	(3)	"	2	B	R	b	r			「	イ	ツ	×	β	θ		
xxxx0011	(4)	#	3	C	S	c	s			」	ウ	テ	モ	ε	ω		
xxxx0100	(5)	\$	4	D	T	d	t			、	エ	ト	ト	μ	Ω		
xxxx0101	(6)	%	5	E	U	e	u			・	オ	ナ	।	ç	Ü		
xxxx0110	(7)	&	6	F	U	f	v			ヲ	カ	ニ	ヨ	ρ	Σ		
xxxx0111	(8)	'	7	G	W	g	w			ア	キ	ヌ	ラ	q	π		
xxxx1000	(1)	(	8	H	X	h	x			イ	ク	ネ	リ	τ	χ		
xxxx1001	(2)	)	9	I	Y	i	y			カ	ケ	ノ	ル	~	ψ		
xxxx1010	(3)	*	:	J	Z	j	z			エ	コ	ハ	レ	j	ヰ		
xxxx1011	(4)	+	;	K	C	k	{			オ	サ	ヒ	ロ	*	¤		
xxxx1100	(5)	,	<	L	¥	l	}			ヤ	シ	フ	ワ	Φ	円		
xxxx1101	(6)	-	=	M	』	m	›			ュ	ズ	ヘ	ン	モ	÷		
xxxx1110	(7)	.	>	N	^	n	†			ヨ	セ	ホ	^	ñ			
xxxx1111	(8)	/	?	O	_	o	←			弔	リ	マ	¶	ö	■		

Figura 13-4 Caracteres definidos dentro de la tabla CGROM (cortesía de Hitachi)

## 13.4 CARA

El módulo CGROM donde se (figura 13-4). Cada visualizar un carácter ejemplo, para visual b'01000001'.

También per tabla interna. Estos (Character Generat

## 13.5 MODO

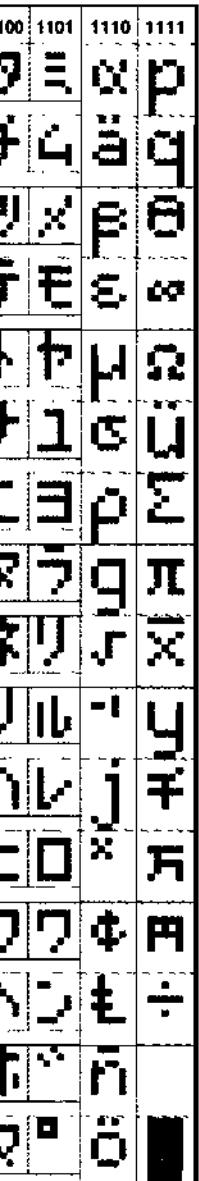
El LM016L,

- **Modo Com** "Borrar Dis modo coma indicar que modo tarda
- **Modo Cara** un carácter ASCII a vis R/W debe s También se en este mod
- **Modo lectu** LCD inform Busy Flag). de datos es internas y no el Busy Flag

El modo de o problemas de tiempo comprobar que no es Busy Flag que sólo instrucción que se le

RAM son:

nte se apunta a la



## 13.4 CARACTERES DEFINIDOS EN LA CGROM

El módulo LM016L posee una zona de memoria interna no volátil llamada CGROM donde se almacena una tabla con los 192 caracteres que pueden ser visualizados (figura 13-4). Cada uno de los caracteres tiene su representación binaria de ocho bits. Para visualizar un carácter debe recibir por el bus de datos el código correspondiente. Por ejemplo, para visualizar el carácter "A" el LCD debe recibir por su bus de datos el código b'01000001'.

También permite definir ocho nuevos caracteres de usuario, no incluidos en su tabla interna. Estos caracteres se guardan en una zona de RAM denominada CGRAM (*Character Generator RAM*).

## 13.5 MODOS DE FUNCIONAMIENTO

El LM016L, tiene tres modos de funcionamiento principales:

- **Modo Comando.** Cuando por el bus de datos el LCD recibe instrucciones como "Borrar Display", "Mover Cursor", "Desplazar a izquierda", etc. Para trabajar en modo comando, el pin RS debe estar a "0". El pin R/W también debe ser "0" para indicar que se está realizando una operación de escritura. Una operación en este modo tarda un máximo de 1,64 ms.
- **Modo Carácter o Dato.** Cuando por el bus de datos el visualizador LCD recibe un carácter a escribir en la DDRAM. Es decir, cuando se envía al LCD el carácter ASCII a visualizar. Para trabajar en este modo, el pin RS debe estar a "1". El pin R/W debe ser "0" para indicar que está realizando una operación de escritura. También se le puede llamar "modo carácter" o "modo registro". Una operación en este modo tarda un máximo de 40  $\mu$ s.
- **Modo lectura del "Busy Flag" o LCD Ocupada.** En el bit 7 del bus de dato el LCD informa al microcontrolador de que está ocupado, (este bit es denominado *Busy Flag*). Para ello se lee el bus de dato con RS=0 y R/W=1, si el bit 7 del bus de datos es "1" indica que la pantalla LCD está ocupada realizando operaciones internas y no puede aceptar nuevas instrucciones ni datos. Hay que esperar a que el *Busy Flag* valga "0" para enviarle la siguiente instrucción o carácter.

El modo de operación de lectura del *Busy Flag* se ha ideado para evitar posibles problemas de tiempo, de manera que no se realiza ninguna operación con el LCD hasta comprobar que no está ocupado. El pin R/W permite leer el registro de estado en el modo *Busy Flag* que sólo sirve para comprobar si el controlador ha terminado de realizar la instrucción que se le ha enviado y así poder enviar más.

(esia de Hitachi)

Para un control sencillo, se pueden realizar pausas después de cada instrucción o envío de datos para no tener que leer el registro de estado, con ello se evita el modo de lectura del *Busy Flag*. La principal ventaja de esto es que se logra ahorrar un pin del microcontrolador porque la línea R/W no es necesaria y se puede conectar directamente a masa, tal como se ilustra en la figura 13-5. La detección del *Busy Flag* se sustituye entonces por un pequeño retardo antes de realizar cualquier nueva operación con el display LCD. Este retardo debe ser mayor de 1,64 ms si trabaja en modo comando y mayor de 40  $\mu$ s si trabaja en modo dato.

## 13.6 COMANDOS DE CONTROL

Los comandos que admite el módulo LM016L se resumen en la tabla 13-2.

COMANDO	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
<i>Clear Display</i>	0	0	0	0	0	0	0	0	0	1
<i>Return Home</i>	0	0	0	0	0	0	0	0	1	*
<i>Entry Mode Set</i>	0	0	0	0	0	0	0	1	I/D	S
<i>Display Control</i>	0	0	0	0	0	0	1	D	C	B
<i>Cursor and display Shift</i>	0	0	0	0	0	1	S/C	R/L	*	*
<i>Function Set</i>	0	0	0	0	1	DL	N	F	*	*
<i>Set CGRAM Address</i>	0	0	0	1	<i>CGRAM Address</i>					
<i>Set DDRAM Address</i>	0	0	1	<i>DDRAM Address</i>						
<i>Read Busy Flag</i>	0	1	BF	<i>DDRAM Address</i>						
<i>Write RAM</i>	1	0	<i>Write Data</i>							
<i>Read RAM</i>	1	1	<i>Read Data</i>							

Tabla 13-2 Comandos del visualizador LCD LM016L

Los comandos se envían a través del bus de datos. Para que el LCD los reconozca hay que poner la señal RS a nivel bajo. A continuación se detallan los comandos y símbolos de esta tabla:

- *Clear Display* (0 0 0 0 0 0 1). Borra pantalla y devuelve el cursor a la posición inicial (dirección 0 de la DDRAM)
- *Return Home* (0 0 0 0 0 0 1 x). Cursor a dirección origen. Devuelve el cursor a la posición original de la DDRAM (dirección 00h) quedando intacto su contenido.
- *Entry Mode Set* (0 0 0 0 0 1 I/D S). Modo Entrada. Establece las características de escritura de los datos *Shift* e *Increment/Decrement*:
  - S = 0. La información visualizada en pantalla no se desplaza al escribir un nuevo carácter.

- S =
- I/D =
- I/D =
- *Display Control*
  - B =
  - B =
  - C =
  - C =
  - D =
  - D =
- *Cursor and display Shift*
  - R/L =
  - R/L =
  - S/C =
  - S/C =
- *Function Set*
  - F = 0
  - F = 1
  - N = 0
  - N = 1
  - DL =

## 13.7 CONEXIONES

La figura

instrucción o  
el modo de  
ar un pin del  
rectamente a  
se sustituye  
ación con el  
comando y

3-2.

DB1	DB0
0	1
1	*
I/D	S
C	B
*	*
*	*

os reconozca  
comandos y

la posición

el cursor la  
contenido.  
terísticas de

desplaza al

- S = 1. La información visualizada se desplaza al escribir un nuevo carácter. La pantalla se desplaza en el sentido indicado por el bit I/D cuando el cursor llega al filo de la pantalla.
- I/D = 1. Incremento automático de la posición del cursor. La posición de la DDRAM se incrementa automáticamente tras cada lectura o escritura a la misma,
- I/D = 0. Decremento de la posición del cursor. Se decrementa el puntero de la DDRAM.
- *Display Control* (0 0 0 0 1 D C B). Control de la pantalla:
  - B = 0. *Blink OFF*, no hay efecto de parpadeo del cursor.
  - B = 1. *Blink ON*, efecto de parpadeo con un cursor rectangular.
  - C = 0. *Cursor OFF*, el cursor no se visualiza.
  - C = 1. *Cursor ON*, el cursor es visualizado.
  - D = 0. *Display OFF*, el display se apaga.
  - D = 1. *Display ON*, el display se enciende.
- *Cursor and Display Shift* (0 0 0 1 S/C R/L x x). Control de los desplazamientos del cursor y de la pantalla:
  - R/L = 0. *Left*. A la izquierda.
  - R/L = 1. *Right*. A la derecha.
  - S/C = 0. El efecto de desplazamiento se aplica sólo sobre el cursor sin alterar el contenido de la DDRAM.
  - S/C = 1. El efecto de desplazamiento se aplica sobre todo el display.
- *Function Set* (0 0 1 DL N F x x). Características de control hardware:
  - F = 0. *Font*. Caracteres de 5 x 7 puntos.
  - F = 1. *Font*. Caracteres de 5 x 10 puntos.
  - N = 0. *Number Line*. Pantalla de 1 línea.
  - N = 1. *Number Line*. Pantalla de 2 líneas.
  - DL = 0. *Data Length*. Comunicación con 4 bits. Indica al display LCD que solamente se van a utilizar las líneas DB7, DB6, DB5 y DB4 para enviarle los datos y que se hará enviando primero el nibble alto, y a continuación el nibble bajo del dato
  - DL = 1. *Data Length*. Comunicación con 8 bits.
- *Set CGRAM Address*. Se va a escribir sobre la dirección CGRAM señalada.
- *Set DDRAM Address* (1 d d d d d d). Esta instrucción se utiliza para modificar el puntero a la DDRAM. Así por ejemplo, si la dirección es la 08h se escribirá en el centro de la primera línea.
- *Read Busy Flag*. Lee el BF indicando si hay una operación interna en curso y lee, además, el contenido de la dirección DDRAM apuntada.

## 13.7 CONEXIÓN DE LCD MEDIANTE 4 BITS

La figura 13-5 ilustra la forma de conectar el LCD al Puerto B del microcontrolador mediante 4 líneas y sin lectura del *Busy flag* por lo que también se

ahorra la línea R/W que se conecta a masa. La principal ventaja de este circuito es que utiliza el mínimo posible de pines (6 líneas) del microcontrolador para el control del display LCD.

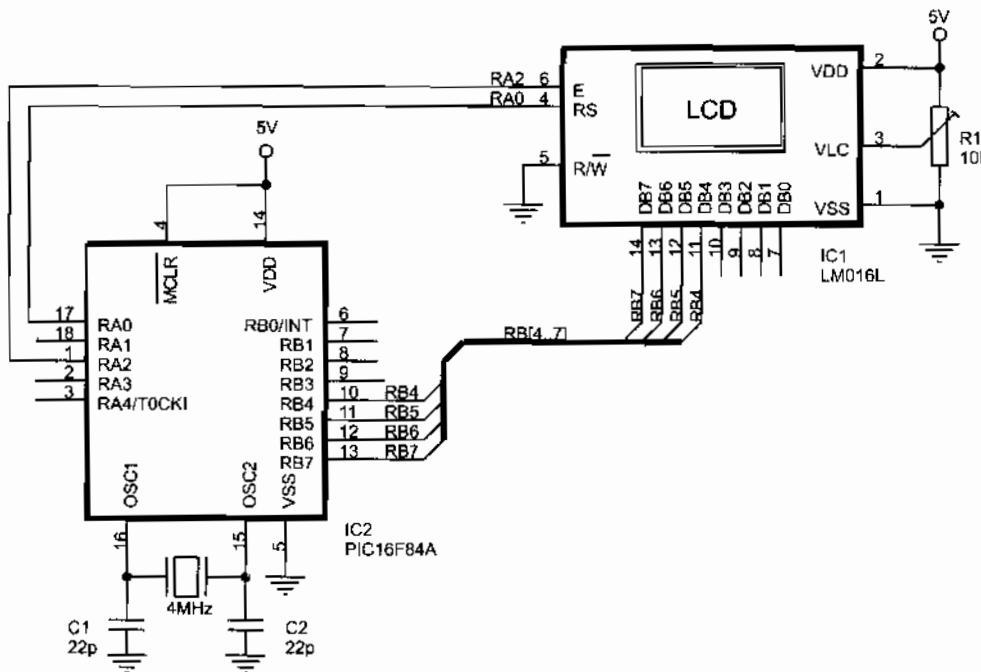


Figura 13-5 Conexión del módulo LCD al PIC16F84A mediante bus de 4 líneas

## 13.8 LIBRERÍA DE SUBRUTINAS

La librería LCD\_4BIT.INC contiene las subrutinas de control que permiten realizar las tareas básicas de control de un módulo LCD conectado según el circuito de la figura 13-5.

Aunque esta librería está suficientemente documentada, se destacan algunas de sus subrutinas principales. Otras serán comentadas más adelante.

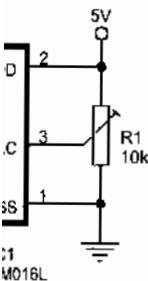
- “LCD\_Inicializa”. Inicializa el módulo LCD para su correcto funcionamiento. Configura funciones del LCD, produce un reset por software, borra la memoria DDRAM y enciende la pantalla. Es necesario ejecutar esta subrutina al principio de los programas que vayan a utilizar la visualización mediante LCD. El fabricante especifica que para garantizar una correcta inicialización debe realizarse como indica la figura 13-6.
- “LCD\_Carácter”. Visualiza en la posición actual del cursor el código ASCII del dato contenido en el registro W.

Figura 13-6 P

- “LCD\_B
- “LCD\_L
- “LCD\_L
- “LCD\_P
- (W). Por
- de la líne
- “LCD\_P
- “LCD\_L
- “LCD\_D

Esta librería figura 13-10, don

circuito es que  
a el control del



de 4 líneas

permiten realizar  
el circuito de la figura

n algunas de su

funcionamiento.  
borra la memoria  
tina al principio  
ante LCD. El  
alización debe

digo ASCII del

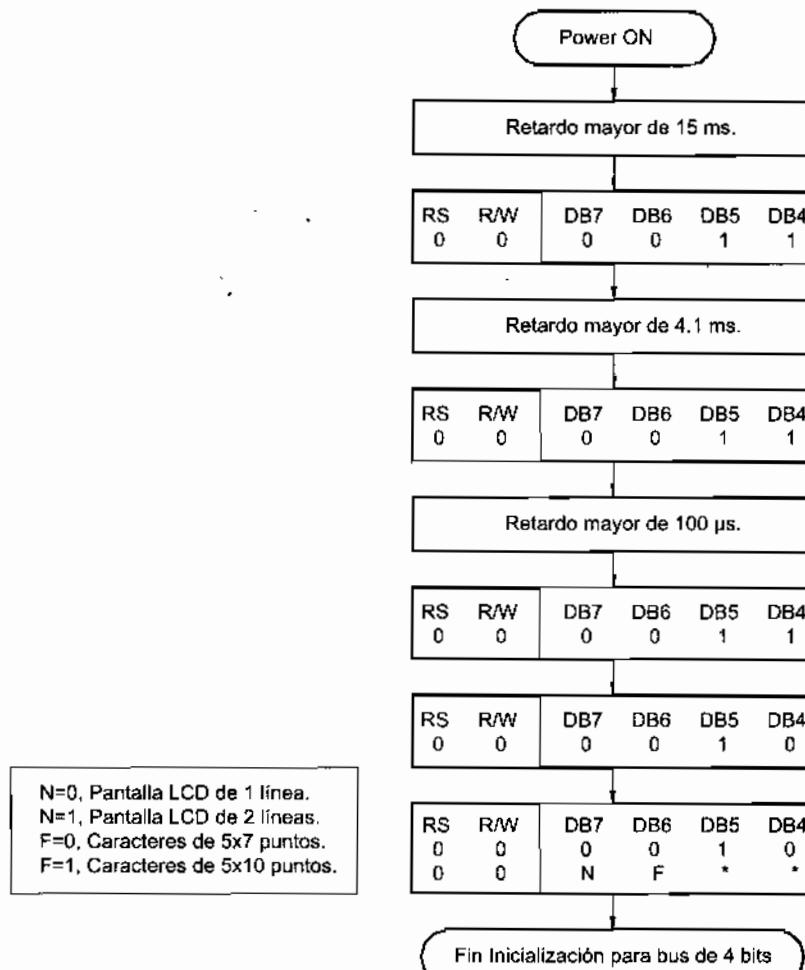


Figura 13-6 Proceso de inicialización del LM016L para conexión con bus de 4 líneas

- “LCD\_Borra”. Borra toda la pantalla y pone el cursor a principio de la línea 1.
- “LCD\_Linea1”. Envía el cursor al principio de la línea 1.
- “LCD\_Linea2”. Envía el cursor al principio de la línea 2.
- “LCD\_PosicionLinea1”. Envía el cursor a la posición de la línea 1 indicada por (W). Por ejemplo, si (W)=0x08, al ejecutar esta subrutina el cursor se irá al centro de la línea en una pantalla de 16 caracteres.
- “LCD\_PosicionLinea2”. Igual que el anterior para la línea 2.
- “LCD\_LineaEnBlanco”. Visualiza una línea en blanco.
- “LCD\_DosEspaciosBlanco”. Visualiza dos espacios en blanco.

Esta librería ha sido diseñada para que funcione correctamente con el circuito de la figura 13-10, donde a las líneas del bus de datos se han conectado otros dispositivos. Por

ello no deben alterar el contenido de las líneas de la parte baja del Puerto B que no son utilizadas para el LCD (pines RB3:RB0), primero se lee estas líneas y después se vuelve a enviar este dato sin cambiarlo. También debe mantener la configuración de las líneas del Puerto B cuando no se utilizan para enviar datos al LCD. De estas funciones cuida la subrutina "LCD\_EscribeLCD".

El listado completo de la librería LCD\_4BIT.INC es el siguiente:

```
***** Libreria "LCD_4BIT.INC" *****
;
; Estas subrutinas permiten realizar las tareas básicas de control de un módulo LCD de 2
; líneas por 16 caracteres, compatible con el modelo LM016L.
;
; El visualizador LCD está conectado al Puerto B del PIC mediante un bus de 4 bits. Las
; conexiones son:
; - Las 4 líneas superiores del módulo LCD, pines <DB7:DB4> se conectan a las 4
;   líneas superiores del Puerto B del PIC, pines <RB7:RB4>.
; - Pin RS del LCD a la linea RA0 del PIC.
; - Pin R/W del LCD a la línea RA1 del PIC, o a masa.
; - Pin Enable del LCD a la línea RA2 del PIC.
;
; Se utilizan llamadas a subrutinas de retardo de tiempo localizadas en la librería RETARDOS.INC.
;
; ZONA DE DATOS *****
;
; CBLOCK
LCD_Dato
LCD_GuardaDatos
LCD_GuardaTRISB
LCD_Auxiliar1
LCD_Auxiliar2
ENDC

LCD_CaracteresPorLinea EQU .16 ; Número de caracteres por linea de la pantalla.

#DEFINE LCD_PinRS PORTA,0
#DEFINE LCD_PinRW PORTA,1
#DEFINE LCD_PinEnable PORTA,2
#DEFINE LCD_BusDatos PORTB

; Subrutina "LCD_Inicializa"
;
; Inicialización del módulo LCD: Configura funciones del LCD, produce reset por software,
; borra memoria y enciende pantalla. El fabricante especifica que para garantizar la
; configuración inicial hay que hacerla como sigue:
;
LCD_Inicializa
    bsf STATUS,RP0      ; Configura las líneas conectadas al pines RS,
    bcf LCD_PinRS      ; R/W y E.
    bcf LCD_PinEnable
    bcf LCD_PinRW
    bcf STATUS,RP0
```

puerto B que no son  
después se vuelve a  
ón de las líneas del  
funciones cuida la

```

bcf    LCD_PinRW      ; En caso de que esté conectado le indica
bcf    LCD_PinEnable   ; que se va a escribir en el LCD.
bcf    LCD_PinRS       ; Impide funcionamiento del LCD poniendo E=0.
call   Retardo_20ms    ; Activa el Modo Comando poniendo RS=0.
movlw  b'00110000'
call   LCD_EscribeLCD ; Escribe el dato en el LCD.
call   Retardo_5ms
movlw  b'00110000'
call   LCD_EscribeLCD
call   Retardo_200micros
movlw  b'00110000'
call   LCD_EscribeLCD
movlw  b'00100000'
call   LCD_EscribeLCD
; Interface de 4 bits.

```

; Ahora configura el resto de los parámetros:

```

call   LCD_2Lineas4Bits5x7 ; LCD de 2 líneas y caracteres de 5x7 puntos.
call   LCD_Borra          ; Pantalla encendida y limpia. Cursor al principio
call   LCD_CursorOFF      ; de la línea 1. Cursor apagado.
call   LCD_CursorIncr     ; Cursor en modo incrementar.
return

```

; Subrutina "LCD\_EscribeLCD"

```

; Envía el dato del registro de trabajo W al bus de dato y produce un pequeño pulso en el pin
; Enable del LCD. Para no alterar el contenido de las líneas de la parte baja del Puerto B que
; no son utilizadas para el LCD (pines RB3:RB0), primero se lee estas líneas y después se
; vuelve a enviar este dato sin cambiarlo.

```

LCD\_EscribeLCD

```

andlw b'11110000'      ; Se queda con el nibble alto del dato que es el
movwf LCD_Dato          ; que hay que enviar y lo guarda.
movf  LCD_BusDatos,W    ; Lee la información actual de la parte baja
andlw b'00001111'        ; del Puerto B, que no se debe alterar.
iorwf LCD_Dato,F        ; Envíará la parte alta del dato de entrada
                        ; y en la parte baja lo que había antes.
bsf   STATUS,RP0         ; Acceso al Banco 1.
movf  TRISB,W            ; Guarda la configuración que tenía antes TRISB.
movwf LCD_GuardaTRISB   ; Las 4 líneas inferiores del Puerto B se dejan
movlw  b'00001111'        ; como estaban y las 4 superiores como salida.
andwf PORTB,F            ; Acceso al Banco 0.
bcf   STATUS,RP0
movf  LCD_Dato,W          ; Recupera el dato a enviar.
movwf LCD_BusDatos        ; Envía el dato al módulo LCD.
bsf   LCD_PinEnable       ; Permite funcionamiento del LCD mediante un pequeño
bcf   LCD_PinEnable       ; pulso y termina impidiendo el funcionamiento del LCD.
bsf   STATUS,RP0           ; Acceso al Banco 1. Restaura el antiguo valor en
movf  LCD_GuardaTRISB,W   ; la configuración del Puerto B.
movwf PORTB               ; Realmente es TRISB.
bcf   STATUS,RP0           ; Acceso al Banco 0.
return

```

; Subrutinas variadas para el control del módulo LCD -----

; Los comandos que pueden ser ejecutados son:

LCD_CursorIncr	movlw b'00000110'	; Cursor en modo incrementar.
	goto LCD_EnviaComando	
LCD_Linea1	movlw b'10000000'	; Cursor al principio de la Línea 1.
	goto LCD_EnviaComando	; Dirección 00h de la DDRAM
LCD_Linea2	movlw b'11000000'	; Cursor al principio de la Línea 2.
	goto LCD_EnviaComando	; Dirección 40h de la DDRAM
LCD_PosicionLinea1	iorlw b'10000000'	; Cursor a posición de la Línea 1, a partir de la
	goto LCD_EnviaComando	; dirección 00h de la DDRAM más el valor del
LCD_PosicionLinea2	iorlw b'11000000'	; Cursor a posición de la Línea 2, a partir de la
	goto LCD_EnviaComando	; dirección 40h de la DDRAM más el valor del
LCD_OFF	movlw b'00001000'	; registro W.
	goto LCD_EnviaComando	; Pantalla apagada.
LCD_CursorON	movlw b'00001110'	; Pantalla encendida y cursor encendido.
	goto LCD_EnviaComando	
LCD_CursorOFF	movlw b'00001100'	; Pantalla encendida y cursor apagado.
	goto LCD_EnviaComando	
LCD_Borra	movlw b'00000001'	; Borra toda la pantalla, memoria DDRAM y pone el
	goto LCD_EnviaComando	; cursor a principio de la línea 1.
LCD_2Lineas4Bits5x7	movlw b'00101000'	; Define la pantalla de 2 líneas, con caracteres
	goto LCD_EnviaComando	; de 5x7 puntos y conexión al PIC mediante bus de
		; 4 bits.
; Subrutinas "LCD_EnviaComando" y "LCD_Caracter" -----		
;"LCD_EnviaComando". Escribe un comando en el registro del módulo LCD. La palabra de		
; comando ha sido entregada a través del registro W. Trabaja en Modo Comando.		
;"LCD_Caracter". Escribe en la memoria DDRAM del LCD el carácter ASCII introducido a		
; a través del registro W. Trabaja en Modo Dato.		
LCD_EnviaComando	bcf LCD_PinRS	; Activa el Modo Comando, poniendo RS=0.
	goto LCD_Envia	
LCD_Caracter	bsf LCD_PinRS	; Activa el "Modo Dato", poniendo RS=1.
	call LCD_CodigoCGROM	; Obtiene el código para correcta visualización.
LCD_Envia	movwf LCD_GuardaDato	; Guarda el dato a enviar.
	call LCD_EscribeLCD	; Primero envía el nibble alto.

swapf

call

btfs

call

call

return

; Subrutina "LCD\_C

;

; A partir del carácter

; tabla CGROM del

; ASCII de la "N" en

;

; Esta subrutina conv

; que puedan ser visu

;

; Entrada: en (W) el c

; Salida: en (W) el c

LCD\_CodigoCGRO

movwf

LCD\_EnheMinuscula

subiw

btfs

goto

movlw

movwf

goto

LCD\_EnheMayuscula

movf

sublw

btfs

goto

movlw

movwf

goto

LCD\_Grado

movf

sublw

btfs

goto

movlw

movwf

LCD\_FinCGROM

movf

return

; Subrutina "LCD\_D

;

; Visualiza espacios en

LCD\_LineaEnBlanco

```

swapf  LCD_GuardaDatos,W      ; Ahora envía el nibble bajo. Para ello pasa el
call    LCD_EscribeLCD        ; nibble bajo del dato a enviar a parte alta del byte.
btfs  LCD_PinRS              ; Se envía al visualizador LCD.
call    Retardo_2ms           ; Debe garantizar una correcta escritura manteniendo
call    Retardo_50micros       ; 2 ms en modo comando y 50 µs en modo carácter.
return

```

; Subrutina "LCD\_CodigoCGROM" -----

; A partir del carácter ASCII número 127 los códigos de los caracteres definidos en la  
; tabla CGROM del LM016L no coinciden con los códigos ASCII. Así por ejemplo, el código  
; ASCII de la "Ñ" en la tabla CGRAM del LM016L es EEh.

; Esta subrutina convierte los códigos ASCII de la "Ñ", ";" y otros, a códigos CGROM para que  
; que puedan ser visualizado en el módulo LM016L.

; Entrada: en (W) el código ASCII del carácter que se desea visualizar.  
; Salida: en (W) el código definido en la tabla CGROM.

```

LCD_CodigoCGROM
    movwf  LCD_Dato          ; Guarda el valor del carácter y comprueba si es
LCD_EnheMinuscula
    sublw  'í'               ; un carácter especial.
    btfs  STATUS,Z           ; ¿Es la "í"?
    goto  LCD_EnheMayuscula ; No es "í".
    movlw  b'11101110'        ; Código CGROM de la "í".
    movwf  LCD_Dato
    goto  LCD_FinCGROM
LCD_EnheMayuscula
    movf   LCD_Dato,W        ; Recupera el código ASCII de entrada.
    sublw  'Ñ'               ; ¿Es la "Ñ"?
    btfs  STATUS,Z           ; No es "Ñ".
    goto  LCD_Grado          ; Código CGROM de la "ñ". (No hay símbolo para
                                ; la "Ñ" mayúscula en la CGROM).
    movlw  b'11101110'
    movwf  LCD_Dato
    goto  LCD_FinCGROM
LCD_Grado
    movf   LCD_Dato,W        ; Recupera el código ASCII de entrada.
    sublw  ';'               ; ¿Es el símbolo ";"?
    btfs  STATUS,Z           ; No es ";".
    goto  LCD_FinCGROM       ; Código CGROM del símbolo ";".
    movlw  b'11011111'
    movwf  LCD_Dato
LCD_FinCGROM
    movf   LCD_Dato,W        ; En (W) el código buscado.
    return

```

; Subrutina "LCD\_DosEspaciosBlancos" y "LCD\_LineaBlanco" -----

; Visualiza espacios en blanco.

LCD\_LineaEnBlanco

```

        movlw  LCD_CaracteresPorLinea
        goto   LCD_EnviaBlancos

LCD_UnEspacioBlanco
        movlw  .1
        goto   LCD_EnviaBlancos

LCD_DosEspaciosBlancos
        movlw  .2
        goto   LCD_EnviaBlancos

LCD_TresEspaciosBlancos
        movlw  .3

LCD_EnviaBlancos
        movwf  LCD_Auxiliar1 ; (LCD_Auxiliar1) se utiliza como contador.

LCD_EnviaOtroBlanco
        movlw  ''
        call   LCD_Caracter
        decfsz LCD_Auxiliar1,F ; Visualiza tanto espacios en blanco como se
        goto   LCD_EnviaOtroBlanco ; haya cargado en (LCD_Auxiliar1).

        return

```

```

; - Si (W)=b'010101
; - Si (W)=b'101011
;
LCD_Nibble
    andlw
    mcvwf
    sublw
    btfs
    goto
    movf
    addlw
    goto
LCD_EnviaByteLet
    movf
    addlw
LCD_FinVisualizaD
    goto

```

### 13.9 VISU

Como ejer-  
aplicación muy se-  
un mensaje.

; Subrutinas "LCD\_ByteCompleto" y "LCD\_Byte" -----  
;  
; Subrutina "LCD\_ByteCompleto", visualiza el byte que almacena el registro W en el  
; lugar actual de la pantalla. Por ejemplo, si (W)=b'10101110' visualiza "AE".  
;  
; Subrutina "LCD\_Byte" igual que la anterior, pero en caso de que el nibble alto sea cero  
; visualiza en su lugar un espacio en blanco. Por ejemplo si (W)=b'10101110' visualiza "AE"  
; y si (W)=b'00001110', visualiza " E" (un espacio blanco delante).  
;  
Utilizan la subrutina "LCD\_Nibble" que se analiza más adelante.

```

LCD_Byte
    movwf LCD_Auxiliar2 ; Guarda el valor de entrada.
    andlw b'11110000' ; Analiza si el nibble alto es cero.
    btfss STATUS,Z ; Si es cero lo apaga.
    goto LCD_VisualizaAlto ; No es cero y lo visualiza.
    movlw '' ; Visualiza un espacio en blanco.

    call LCD_Caracter
    goto LCD_VisualizaBajo

```

```

LCD_ByteCompleto
    movwf LCD_Auxiliar2 ; Guarda el valor de entrada.
LCD_VisualizaAlto
    swapf LCD_Auxiliar2,W ; Pone el nibble alto en la parte baja.
    call LCD_Nibble ; Lo visualiza.
LCD_VisualizaBajo
    movf LCD_Auxiliar2,W ; Repite el proceso con el nibble bajo.
    call LCD_Nibble ; Lo visualiza.
;
```

; Subrutina "LCD\_Nibble"  
; Visualiza en el lugar actual de la pantalla, el valor hexadecimal que almacena en el nibble  
; bajo del registro W. El nibble alto de W no es tenido en cuenta. Ejemplo:  
;

```

;*****  

;  

; El módulo LCD visu  

;  

; ZONA DE DATOS  

;  

; CONFIG  

; LIST  

; INCLUDE  

;  

; CBLOCK 0  

; ENDC  

;  

; ZONA DE CÓDIGOS  

;  

ORG    0  

Inicio  

call   LC  

movlw 'H'  

call   LC  

movlw 'o'  

call   LC  

movlw 'T'  

call   LC  

movlw 'a'  

call   LC

```

```

; - Si (W)=b'01010110', se visualizará "6".
; - Si (W)=b'10101110', se visualizará "E".
;
LCD_Nibble
    andlw  b'00001111'          ; Se queda con la parte baja.
    movwf  LCD_Auxiliar1        ; Lo guarda.
    sublw  0x09                 ; Comprueba si hay que representarlo con letra.
    btfss  STATUS,C
    goto   LCD_EviaByteLetra
    movf   LCD_Auxiliar1,W
    addlw  '0'                  ; El número se pasa a carácter ASCII sumándole
    goto   LCD_FinVisualizaDigito ; el ASCII del cero y lo visualiza.

LCD_EviaByteLetra
    movf   LCD_Auxiliar1,W
    addlw  'A'-0x0A            ; Si, por tanto, se le suma el ASCII de la 'A'.
    LCD_FinVisualizaDigito
    goto   LCD_Caracter         ; Y visualiza el carácter. Se hace con un "goto"
                                ; para no sobrecargar la pila.

```

## 13.9 VISUALIZACIÓN DE CARACTERES

Como ejemplo de aplicación de la librería anterior se detalla un programa de aplicación muy sencillo donde se indica el procedimiento para visualizar los caracteres de un mensaje.

```

***** LCD_01.asm *****
;
; El módulo LCD visualiza el mensaje "Hola".
;
; ZONA DE DATOS *****
;
; CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

; ZONA DE CÓDIGOS *****
;
ORG     0
Inicio
    call   LCD_Inicializa
    movlw  'H'
    call   LCD_Caracter
    movlw  'o'
    call   LCD_Caracter
    movlw  'T'
    call   LCD_Caracter
    movlw  'a'
    call   LCD_Caracter

```

```

sleep           ; Entra en modo de bajo consumo.

INCLUDE <LCD_4BIT.INC>      ; Subrutinas de control del módulo LCD.
INCLUDE <RETARDOS.INC>      ; Subrutinas de retardo.
END             ; Fin del programa.

```

Numero\_NoCeroAlto  
Numero\_CeroAlto  
; ZONA DE CÓDIGO

## 13.10 VISUALIZACIÓN DE VALORES NUMÉRICOS

La librería LCD\_4BIT.INC contiene algunas subrutinas que permiten la visualización de valores numéricos de 8 bits:

- "LCD\_BytCompleto". Visualiza el valor hexadecimal del byte que almacena el registro W en el lugar actual del cursor. Por ejemplo, si (W)=b'00001110', visualiza "0E".
- "LCD\_Byt". Igual que el anterior pero si el nibble alto es un cero visualiza un blanco en su lugar. Por ejemplo si (W)=00001110, visualiza " E" (con espacio en blanco en el nibble alto); si (W)=b'10101110', visualiza "AE".
- "LCD\_Nibble". Visualiza el valor hexadecimal que almacena el nibble bajo del registro W. El nibble alto no es tenido en cuenta. Por ejemplo, si (W)=b'01010110' visualiza "6", si (W)=b'10101110' visualiza "E".

El siguiente programa ejemplo permite apreciar la diferencia de funcionamiento entre cada una de estas subrutinas.

```

***** LCD_03.asm *****
;
; Programa ejemplo para comprender la utilización de las subrutinas de visualización
; de valores numéricos.
;
; Para ello se van a utilizar sucesivamente y en este orden las subrutinas:
; LCD_BytCompleto, LCD_DosEspaciosBlancos, LCD_Byt, "LCD_DosEspaciosBlancos"
; y LCD_Nibble para dos números que serán:
;
; - En la primera línea del LCD un número con el nibble alto no cero. Por ejemplo: 1Dh.
; - En la segunda línea del LCD un número con el nibble alto igual a cero. Ejemplo: 0Dh.
;
; Así por ejemplo, para los números "1D" y "0D" se visualizaría (donde "#" viene a significar
; espacio en blanco):
; "1D##1D##D"    (Primera linea).
; "0D##D##D"     (Segunda linea).
;
; ZONA DE DATOS *****
;
; CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
; LIST      P=16F84A
; INCLUDE <P16F84A.INC>
;
; CBLOCK 0x0C
; ENDC

```

ORG  
Inicio  
Principal  
movlw  
call  
call  
sleep

INCLUDE  
INCLUDE  
END

## 13.11 CONEXIÓN AL LCD

El módulo de LCD se conecta a la placa de desarrollo. Se tratarlo como un periférico de tipo I2C. También se utiliza la conexión RA1 del microcontrolador para la generación de pulsos de sincronización, tal como se hizo en el apartado anterior.

La ventaja de esta conexión es la rapidez al enviar los datos en forma de 4 bits. Además,

Como inconveniente, el microcontrolador tiene que manejar la interfaz de 4 bits. Además, el microcontrolador tiene que manejar la interfaz de 4 bits.

módulo LCD.

**OS**

ue permiten la

que almacena el  
N)=b'00001110',

cero visualiza un  
" (con espacio en

el nibble bajo del  
Por ejemplo, si

e funcionamiento

\*\*\*\*\*

alización

icos"

car

\*\*\*\*\*

```
Numero_NoCeroAlto      EQU    0x1D ; Número ejemplo nibble alto no cero.
Numero_CeroAlto        EQU    0x0D ; Número ejemplo nibble alto cero.
```

; ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG 0
Inicio
Principal
    call LCD_Inicializa
    movlw Numero_NoCeroAlto
    call LCD_BytCompleto
    call LCD_DosEspaciosBlancos
    movlw Numero_NoCeroAlto
    call LCD_Byt
    call LCD_DosEspaciosBlancos
    movlw Numero_NoCeroAlto
    call LCD_Nibble
    call LCD_Linea2           ; Se sitúa en la segunda línea.
    movlw Numero_CeroAlto
    call LCD_BytCompleto
    call LCD_DosEspaciosBlancos
    moviw Numero_CeroAlto
    call LCD_Byt
    call LCD_DosEspaciosBlancos
    movlw Numero_CeroAlto
    call LCD_Nibble
    sleep

INCLUDE <LCD_4BIT.INC>
INCLUDE <RETARDOS.INC>
END
```

### 13.11 CONEXIÓN DE LCD MEDIANTE 8 BITS

El módulo visualizador LCD también puede ser conectado a un puerto de 8 bits y tratarlo como un periférico más, tal como se indica en la figura 13-7. En este ejemplo también se utiliza el modo *Busy Flag*, por ello se conecta el pin R/W del LCD a la línea RA1 del microcontrolador, aunque se podía haber sustituido por un retardo de 2 ms ó 50 µs, tal como se hizo con el control de 4 bits.

La ventaja fundamental del control mediante 8 bit respecto de 4 bit, es una mayor rapidez al enviar los 8 bits por el puerto de una vez, en lugar de empaquetados en bloques de 4 bits. Además, el software del control es algo más sencillo.

Como inconveniente importante destaca el aumento de las líneas ocupadas del microcontrolador. Esto es muy importante en el caso del PIC16F84 que es un microcontrolador al que precisamente no le sobran líneas de entrada/salida.

Hay que hacer observar que la librería LCD\_4BIT.INC también funciona para el esquema de la figura anterior. Evidentemente, aunque la disposición de las patillas es de conexión a 8 bits, funcionará como si sólo estuviesen conectadas 4 líneas.

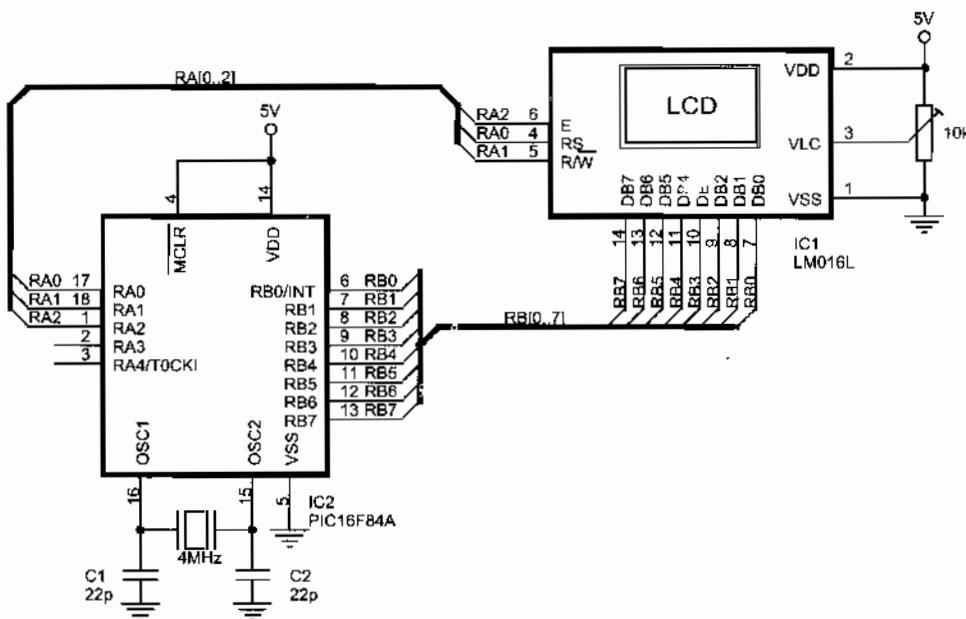


Figura 13-7 Conexión del LM016L al PIC16F84 mediante 8 bits y modo Busy Flag

### 13.12 VISUALIZACIÓN DE MENSAJES FIJOS

Muchos proyectos requieren visualizar mensajes más o menos largos en la pantalla de un LCD. La siguiente librería LCD\_MENS.INC describe dos subrutinas para realizar esta tarea de forma muy sencilla:

- Subrutina “LCD\_Mensaje”, que visualiza mensajes fijos.
- Subrutina “LCD\_Movimiento”, que visualiza mensajes en movimiento.

Estas subrutinas están ampliamente documentada como siguen:

```
***** Librería "LCD_MENS.INC" *****
;
; Librería de subrutinas para el manejo de mensajes a visualizar en un visualizador LCD.
```

CBLOCK

LCD\_ApuntaCaracter

: Indica la posición del carácter a visualizar  
; respecto del comienzo de todos los mensajes,  
; (posición de la etiqueta "Mensajes").

LCD\_ValorCaracter

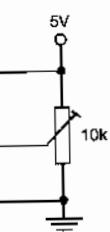
; Código ASCII del carácter a  
; visualizar.

ENDC

```
; Los mensajes tienen
; memoria de program
; Subrutina "LCD_M
; Visualiza por pantalla
; Los mensajes deben
; tenga la siguiente es
; Mensajes
; addwf P
; Mensaje0
; DT "...", 0
; Mensaje1
; ...
; FinMensajes
; La llamada a esta sub
; movlw M
; call L
LCD_Mensaje
movwf LO
movlw M
subwf LO
decf LO
LCD_VisualizaOtroCar
movf L
call M
movwf LO
movf L
btfc ST
goto LO
LCD_NoUltimoCaracte
call LC
incf LC
goto LC
LCD_FinMensaje
return
```

```
; Subrutina "LCD_Mens
; Visualiza un mensaje de
; en una linea, por tanto se
; En el mensaje debe dejar
; conseguir que el desplaza
; CBLOCK
```

ción para el  
pantillas es de



; Los mensajes tienen que estar situados dentro de las 256 primeras posiciones de la memoria de programa, es decir, no pueden superar la dirección 0FFh.

; Subrutina "LCD\_Mensaje" -----

; Visualiza por pantalla el mensaje apuntado por el registro W.

; Los mensajes deben localizarse dentro de una zona encabezada por la etiqueta "Mensajes" y que tenga la siguiente estructura:

; Mensajes ; ; Etiqueta obligatoria!

; addwf PCL,F

; Mensaje0

; DT "...", 0x00

; Mensaje1

; ...

; ...

; FinMensajes

; La llamada a esta subrutina se realizará siguiendo este ejemplo:

; movlw Mensaje0 ; Carga la posición del mensaje.  
; call LCD\_Mensaje ; Visualiza el mensaje.

LCD\_Mensaje

movwf LCD\_ApuntaCaracter

movlw Mensajes

subwf LCD\_ApuntaCaracter,F

decf LCD\_ApuntaCaracter,F

; Posición del primer carácter del mensaje.

; Halla la posición relativa del primer carácter

; del mensaje respecto de etiqueta "Mensajes".

; Compensa la posición que ocupa "addwf PCL,F".

LCD\_VisualizaOtroCaracter

movf LCD\_ApuntaCaracter,W

call Mensajes

movwf LCD\_ValorCaracter

movf LCD\_ValorCaracter,F

btfsc STATUS,Z

goto LCD\_FinMensaje

; Obtiene el código ASCII del carácter apuntado.

; Guarda el valor de carácter.

; Lo único que hace es posicionar flag Z. En caso

; que sea "0x00", que es código indicador final

; de mensaje, sale fuera.

LCD\_NoUltimoCaracter

call LCD\_Caracter

incf LCD\_ApuntaCaracter,F

goto LCD\_VisualizaOtroCaracter

; Visualiza el carácter ASCII leído.

; Apunta a la posición del siguiente carácter

; dentro del mensaje.

LCD\_FinMensaje

return

; Vuelve al programa principal.

; Subrutina "LCD\_MensajeMovimiento" -----

; Visualiza un mensaje de mayor longitud que los 16 caracteres que pueden representarse  
; en una línea, por tanto se desplaza a través de la pantalla.

; En el mensaje debe dejarse 16 espacios en blanco, al principio y al final para  
; conseguir que el desplazamiento del mensaje sea lo más legible posible.

CBLOCK

LCD_CursorPosicion ENDC	; Contabiliza la posición del cursor dentro de la ; pantalla LCD	CBLOCK 0 ENDC
LCD_MensajeMovimiento		; ZONA DE CÓDigos
movwf LCD_ApuntaCaracter movlw Mensajes subwf LCD_ApuntaCaracter,F decf LCD_ApuntaCaracter,F	; Posición del primer carácter del mensaje. ; Halla la posición relativa del primer carácter ; del mensaje respecto de la etiqueta "Mensajes". ; Compensa la posición que ocupa "addwf PCL,F".	ORG 0 Inicio call LCD_ApuntaCaracter movlw Mensajes subwf LCD_ApuntaCaracter,F decf LCD_ApuntaCaracter,F
LCD_PrimeraPosicion		
clrfl LCD_CursorPosicion call LCD_Borra	; El cursor en la posición 0 de la línea. ; Se sitúa en la primera posición de la línea 1 y ; borra la pantalla.	
LCD_VisualizaCaracter		
movlw LCD_CaracteresPorLinea subwf LCD_CursorPosicion,W btfs STATUS,Z goto LCD_NoEsFinalLinea	; ; ¿Ha llegado a final de linea?	; Mensajes ----- ; Mensajes addwf PCL,F Mensaje0 DT "Hola!, qu
LCD_EsFinalLinea		INCLUDE < INCLUDE < INCLUDE < END
call Retardo_200ms call Retardo_200ms	; Lo mantiene visualizado durante este tiempo.	
movlw LCD_CaracteresPorLinea-1 subwf LCD_ApuntaCaracter,F goto LCD_PrimeraPosicion	; Apunta a la posición del segundo carácter visualizado ; en pantalla, que será el primero en la siguiente ; visualización de linea, para producir el efecto ; de desplazamiento hacia la izquierda.	
LCD_NoEsFinalLinea		
movf LCD_ApuntaCaracter,W call Mensajes movwf LCD_ValorCaracter movf LCD_ValorCaracter,F btfc STATUS,Z goto LCD_FinMovimiento	; Obtiene el ASCII del carácter apuntado. ; Guarda el valor de carácter. ; Lo único que hace es posicionar flag Z. En caso ; que sea "0x00", que es código indicador final ; de mensaje, sale fuera.	
LCD_NoUltimoCaracter2		
call LCD_Caracter incf LCD_CursorPosicion,F	; Visualiza el carácter ASCII leído. ; Contabiliza el incremento de posición del ; cursor en la pantalla.	
incf LCD_ApuntaCaracter,F goto LCD_VisualizaCaracter	; Apunta a la siguiente posición por visualizar. ; Vuelve a visualizar el siguiente carácter ; de la linea.	
LCD_FinMovimiento		
return	; Vuelve al programa principal.	

Un ejemplo de aplicación para la visualización de un mensaje fijo por la pantalla, podría ser el siguiente programa donde se aprecia la sencillez de utilización de la subrutina LCD\_Mensaje.

```
***** Mensaje_02.asm *****
;
; En la pantalla del módulo LCD se visualiza un mensaje de menos de 16 caracteres grabado
; en la memoria ROM mediante la directiva "DT". Utiliza la subrutina "LCD_Mensaje" de la
; librería LCD_MENS.INC
;
; ZONA DE DATOS *****
```

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

"Mensaje" es la etiqueta que se define para todos los mensajes.  
"LCD\_ApuntaCaracter", es un register que indica la posición relativa del carácter a visualizar respecto de la etiqueta Mensaje.

Figura 13-8 Diagrama de flujo de ejecución del programa Mensaje\_02.asm.

```

CBLOCK 0x0C
ENDC

; ZONA DE CÓDIGOS ****
ORG 0
Inicio
    call LCD_Inicializa
    movlw Mensaje0
    call LCD_Mensaje
    sleep

; Mensajes -----
; Mensajes
    addwf PCL,F
Mensaje0
    DT "Hola!, que tal? ", 0x00

INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
INCLUDE <RETARDOS.INC>
END

```

La figura 13-8 explica el funcionamiento de la subrutina para la visualización de mensajes fijos en pantalla.

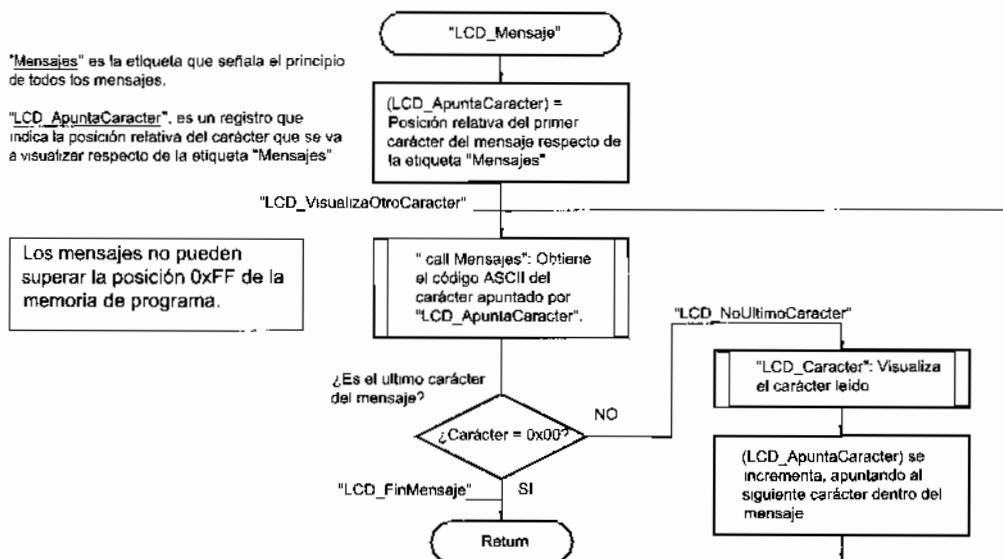


Figura 13-8 Diagrama de flujo para la subrutina de visualización de mensajes fijos

## 13.13 VISUALIZACIÓN DE MENSAJES EN MOVIMIENTO

La subrutina LCD\_MensajeMovimiento de la librería LCD\_MENS.INC, expuesta anteriormente, permite visualizar mensajes en movimiento por la pantalla. El diagrama de flujo que explica su funcionamiento se muestra en figura 13-9.

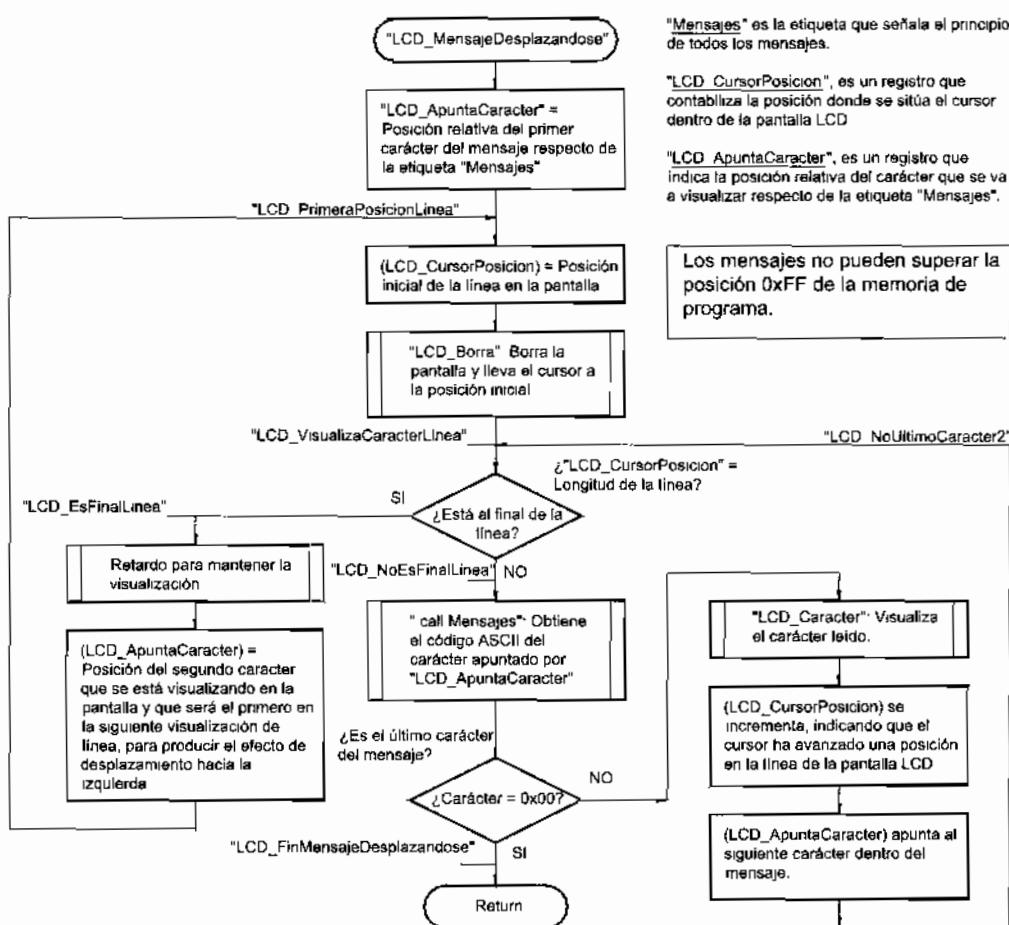


Figura 13-9 Subrutina de visualización de mensajes en movimiento

Un ejemplo de aplicación podría ser el siguiente:

```
***** Mensaje_07.asm *****
;
; El módulo LCD visualiza un mensaje largo (más de 16 caracteres) que se desplaza a lo largo
; de la pantalla. Se utiliza la subrutina LCD_MensajeMovimiento de la librería LCD_MENS.INC.
;
; ZONA DE DATOS *****
```

## 13.14 PR

Respe  
simular y g  
correcto fun  
todas las me

LCD  
terminar de e

LCD  
carácter se i

## MOVIMIENTO

ENS.INC, expuesta  
lla. El diagrama de

que señala el principio  
es

"n", es un registro que  
n donde se sitúa el cursor  
LCD

"er", es un registro que  
ativa del carácter que se va  
de la etiqueta "Mensajes"

no pueden superar la  
de la memoria de

"LCD\_NoUltimoCaracter2"

carácter". Visualiza  
ter leído.

orPosición) se  
indicando que el  
avançado una posición  
de la pantalla LCD.

taCaracter) apunta al  
rácter dentro del

imiento

\*\*\*\*\*

largo  
ENS.INC.

\*\*\*\*\*

```

    CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

; ZONA DE CÓDIGOS ****
; ****
; ****

ORG 0
Inicio call LCD_Inicializa ; Prepara el display LCD.
Principal movlw Mensaje0 ; Apunta al mensaje.
call LCD_MensajeMovimiento
goto Principal ; Repite la visualización.

; "Mensajes"
;
Mensajes addwf PCL,F
Mensaje0 DT " " ; Posición inicial del mensaje.
DT "Estudia un Ciclo Formativo "
DT "de ELECTRONICA."
DT " ", 0x0 ; Espacios en blanco al final.

; INCLUDE <LCD_MENS.INC> ; Subrutina "LCD_MensajeMovimiento".
INCLUDE <LCD_4BIT.INC> ; Subrutas de control del LCD.
INCLUDE <RETARDOS.INC> ; Subrutas de retardos.
END ; Fin del programa.
;
```

Para que aparezcan las comillas, el mensaje hay que grabarlo precediendo las comillas visibles con una barra, como en el siguiente ejemplo:

DT "Estudia \"Desarrollo de Productos Electronicos\". Es tu futuro!"

## 13.14 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador con los siguientes programas. Comprobar su correcto funcionamiento con el esquema de la figura 13-10. El lector puede introducir todas las mejoras que considere conveniente.

**LCD\_01.asm:** En la pantalla del módulo LCD se visualiza el mensaje "Hola". Al terminar de escribir la frase el PIC entrará en modo de bajo consumo.

**LCD\_02.asm:** En la pantalla se visualiza el mensaje "Hola". La escritura de cada carácter se irá realizando cada 500 ms. Después se borrará y comenzará de nuevo.

**LCD\_03.asm:** Programa ejemplo para comprender la utilización de las subrutinas para la visualización de datos numéricos. Para ello se van a utilizar sucesivamente y en este orden las subrutinas: "LCD\_BytCompleto", "LCD\_DosEspaciosBlancos", "LCD\_Byte", "LCD\_DosEspaciosBlancos" y "LCD\_Nibble" para dos números que serán:

- En la primera línea un número con el nibble alto no cero. Por ejemplo: 1Dh.
- En la segunda línea un número con el nibble alto igual a cero. Ejemplo: 0Dh.

Así por ejemplo, para los números "1D" y "0D" se visualizaría (donde "#" viene a significar espacio en blanco):

- "1D##1D##D" (Primera línea).
- "0D###D##D" (Segunda línea).

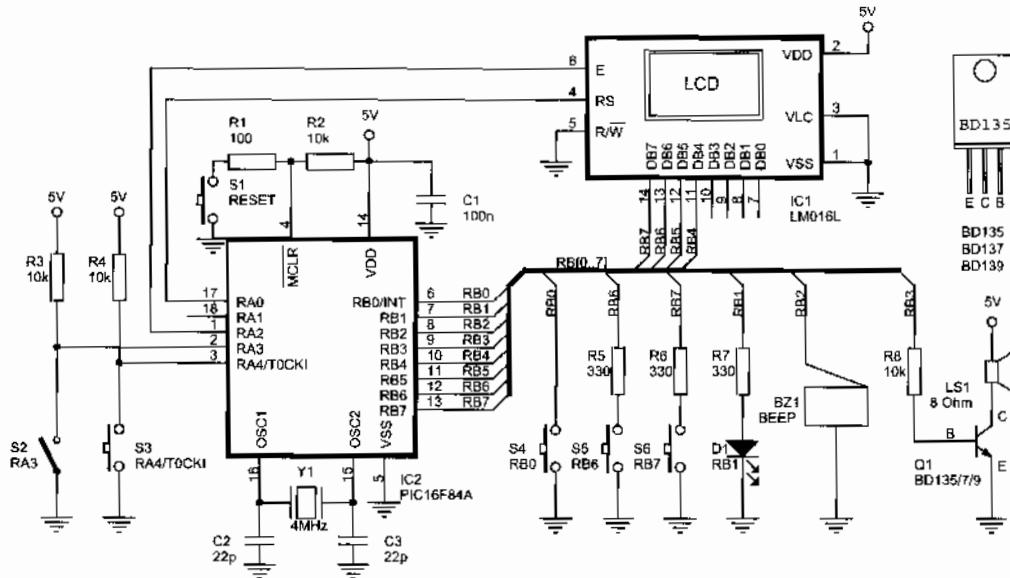


Figura 13-10 Circuito para la realización de las prácticas de este capítulo y próximos

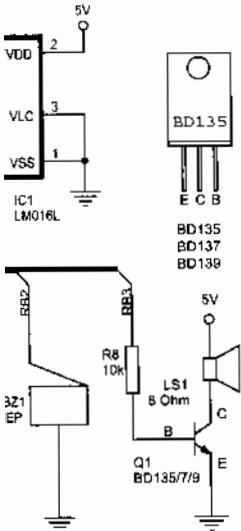
**LCD\_04.asm:** La pantalla visualiza un contador descendente desde '59 hasta 0 y vuelve a repetir la cuenta ininterrumpidamente. En cada valor estará unos 500 ms.

**LCD\_05.asm:** Cada vez que presiona el pulsador conectado al pin RA4 se incrementa un contador que se visualiza en el centro de la primera línea de la pantalla.

**LCD\_06.asm:** Igual que el anterior pero cuando llegue a su valor máximo (por ejemplo 6) se resetea y comienza de nuevo la cuenta.

ción de las subrutinas r sucesivamente y en "DosEspaciosBlancos", os números que serán:  
r ejemplo: 1Dh.  
o Ejemplo: 0Dh.

ia (donde "#" viene a



capítulo y próximos

te desde '59 hasta 0 y unos 500 ms.

tado al pin RA4 se nea de la pantalla.

su valor máximo (por

**LCD\_07.asm:** Igual que el anterior, pero se incrementa mientras se mantenga presionado el pulsador una cuenta cada 200ms.

**LCD\_08.asm:** Mientras se mantenga presionado el pulsador conectado al pin RA4 se incrementa un contador y visualiza en la pantalla en tres formatos: decimal, hexadecimal y binario. Un ejemplo:

- Primera Línea: "CE 206"
- Segunda Línea: "11001110"

**Mensaje\_01.asm:** En pantalla se visualiza un mensaje de menos de 16 caracteres grabado en la memoria ROM mediante la directiva DT.

**Mensaje\_02.asm.** Repetir el anterior utilizando la subrutina LCD\_Mensaje de la librería LCD\_MEN.INC.

**Mensaje\_03.asm:** En pantalla se visualizan varios mensajes, uno detrás de otro. Cada mensaje permanece visualizado durante 2 segundos. Entre mensaje y mensaje la pantalla se mantiene apagada durante 200 ms.

**Mensaje\_04.asm:** En las dos líneas de la pantalla aparecerán dos mensajes parpadeantes.

**Mensaje\_05.asm:** En la primera línea de la pantalla aparecerá un mensaje fijo. En la segunda línea aparecerá un mensaje parpadeante.

**Mensaje\_06.asm:** En la pantalla se visualizarán varios mensajes diferentes. El paso de uno a otro se realiza al actuar sobre el pulsador conectado a la línea RA4. En pantalla aparecerá por ejemplo:

"Mensaje 2" (primera linea).  
"COSLADA Moderna." (segunda linea).

**Mensaje\_07.asm:** En la pantalla se visualizará un mensaje largo (de más de 16 caracteres) que se va desplazando a lo largo de la pantalla. Se utilizará la subrutina LCD\_MensajeMovimiento de la librería LCD\_MENS.INC.

**Mensaje\_08.asm:** Programa para el juego de la Quiniela: Al presionar sobre el pulsador conectado al pin RA4 en la pantalla aparecerá rápidamente "1", "X", "2". Cuando suelta el pulsador, permanece el signo seleccionado.

**Mensaje\_09.asm:** En pantalla visualiza "Cerrado" o "Abierto" según si un pulsador está presionado o no.

## CAPÍTULO 14

# EEPROM DE DATOS

En algunos proyectos es necesario guardar la información que se genera durante el proceso de una forma permanente, es decir, esos datos han de permanecer incluso cuando el sistema se desconecta de la alimentación. Para realizar esta función los microcontroladores PIC disponen de un área de datos EEPROM no volátil que se describe en este capítulo.

### 14.1 MEMORIA EEPROM DE DATOS

El PIC16F84 dispone de una zona con 64 bytes de memoria EEPROM para almacenar datos que no se pierden al desconectar la alimentación. Esto es muy útil ya que permite guardar datos permanentemente. La figura 14-1 (que es un detalle de la arquitectura interna completa de la figura 4-1) muestra la estructura de esta memoria y los registros asociados.

Como en cualquier otra memoria EEPROM se pueden realizar dos tipos de operaciones:

- Operación de lectura.
- Operación de escritura o grabación.

Un ciclo de grabación en una posición EEPROM de datos dura unos 10 ms, un tiempo muy elevado para la velocidad del procesador, que se controla mediante un temporizador interno. Al escribir en una posición de memoria ya ocupada, automáticamente se borra el contenido que había y se introduce el nuevo dato, por lo que no hay comando de borrado.

El PIC16F84A soporta un millón de ciclos de escritura/borrado de su memoria EEPROM de datos y es capaz de guardar la información inalterada durante más de 40 años.

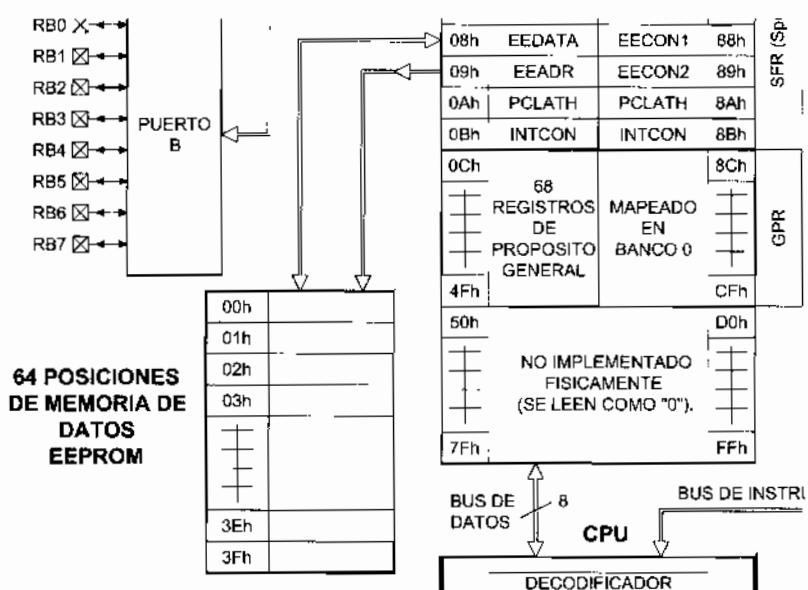


Figura 14-1 Memoria EEPROM de datos en el PIC16F84

Esta memoria no forma parte del espacio direccionable y sólo es accesible para lectura y escritura a través de registros. Los registros relacionados con esta memoria EEPROM de datos son:

- **EEDATA (EEPROM Data Register).** Contiene los bytes que se van a escribir o que se han leído de la EEPROM de datos.
- **EEADR (EEPROM Address Register).** Contiene la dirección de la EEPROM de datos a la que acceder para leer o escribir. Las 64 posiciones de memoria EEPROM ocupan las direcciones de un mapa que comienza en la posición 00h y termina en la 3Fh, por lo que los dos bits de más peso de este registro siempre valen 0.
- **EECON1 (EEPROM Control Register 1).** Los bits de este registro definen el modo de funcionamiento de esta memoria (tabla 14-1). Los bits RD y WR del EECON1 indican respectivamente lectura o escritura. No hay que ponerlos a "0", sólo a "1", ya que se borran automáticamente cuando la operación de lectura o escritura ha sido completada.
- **EECON2 (EEPROM Control Register 2).** Este registro no está implementado físicamente, por lo que es imposible leerlo (si se intenta leer, todos sus bits se leen como ceros). Se emplea como dispositivo de seguridad durante el proceso

de escritura d  
tiempo que p

## 14.2 REGISTROS

Es el registro p  
posición 88h del Banco

Bit 7	Bit 6	Bit 5
-------	-------	-------

- **RD (Read Control).** En "1" se inicia la lectura y se pone a "0" para la escritura. RD = 0: EEPROM. RD = 1: ROM.
- **WR (Write Control).** En "1" se inicia la escritura y se limpia (se ponen a "0") la EEPROM hasta el final. WR = 0: EEPROM. WR = 1: ROM.
- **WREN (EEPROM Write Enable).** WREN = 0: se ignora la escritura. WREN = 1: se permite la escritura.
- **WRERR (EEPROM Write Error).** WRERR = 0: se ignora el error de escritura. WRERR = 1: se posiciona a "1" si se produce un error de escritura en cualquier comando. WRERR = 0: se ignora el error de escritura. WRERR = 1: se posiciona a "1" si se produce un error de escritura en cualquier comando.
- **EIF (EEPROM End of Interrupt).** EIF = 0: se ignora la interrupción pendiente de la operación de escritura. EIF = 1: comienza la operación de escritura. EIF = 0: se ignora la interrupción pendiente de la operación de escritura. EIF = 1: se borra la interrupción pendiente de la operación de escritura.
- **Bits 5, 6 y 7 (EEPROM Address).** Estos bits determinan la dirección de la EEPROM de datos. Los bits 5 y 6 determinan la dirección de la EEPROM de datos. El bit 7 determina si la EEPROM de datos es de 8 o 16 bits.

o de su memoria  
durante más de 40

88h (Sp)  
89h  
8Ah  
8Bh  
3Ch  
+  
+  
+  
+  
+  
+  
GPR  
CFh  
00h  
+  
+  
+  
FFh

DE INSTRU

84

es accesible para  
on esta memoria

e van a escribir o

e la EEPROM de  
ones de memoria  
la posición 00h y  
el registro siempre

registro definen el  
bits RD y WR del  
que ponerlos a "0",  
ación de lectura o

stá implementado  
todos sus bits se  
urante el proceso

de escritura de la EEPROM, para evitar las interferencias en el largo intervalo de tiempo que precisa su desarrollo.

## 14.2 REGISTRO EECON1

Es el registro para el control de la memoria EEPROM de datos. Se encuentra en la posición 88h del Banco 1. Sólo destina cinco bits para este control:

			EEIF	WRERR	WREN	WR	RD
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 14-1 Registro EECON1

- **RD (Read Control Bit).** Bit de control de lectura en la EEPROM. Al ponerlo en "1" se inicia la lectura de un byte en la EEPROM de datos. Este bit se limpia (se pone a "0") por hardware automáticamente al finalizar la lectura de la posición EEPROM.
  - RD = 0. No inicia la lectura de la EEPROM o la misma ha terminado.
  - RD = 1. Inicia la lectura de la EEPROM. Se borra por hardware.
- **WR (Write Control Bit).** Bit de control de escritura en la EEPROM. Al ponerlo en "1" se inicia una escritura de un byte en la EEPROM de datos. Este bit se limpia (se pone en "0") por hardware automáticamente una vez la escritura de la EEPROM ha terminado.
  - WR = 0. No inicia la escritura de la EEPROM o la misma ha terminado.
  - WR = 1. Inicia la escritura de la EEPROM. Se borra por hardware.
- **WREN (EEPROM Write Enable bit).** Permiso de escritura en la EEPROM.
  - WREN = 0. Prohibe la escritura de la EEPROM.
  - WREN = 1. Permite la escritura de la EEPROM.
- **WRERR (EEPROM Write Error Flag Bit).** Flag de error en la escritura. Se posiciona a "1" cuando la operación de escritura termina prematuramente debido a cualquier condición de reset .
  - WRERR = 0. La operación de escritura se ha completado correctamente.
  - WRERR = 1. La operación de escritura ha terminado prematuramente.
- **EEIF (EEPROM Write Operation Interrupt Flag Bit).** Flag de estado de interrupción por finalización de escritura en EEPROM. Señala el final con éxito de la operación de escritura de un byte en la EEPROM.
  - EEIF = 0. La operación de escritura de la EEPROM no ha terminado o no comenzó.
  - EEIF = 1. La operación de escritura de la EEPROM ha terminado. Debe borrarse por software.
- **Bits 5, 6 y 7 (Unimplemented).** No implementados físicamente. Se leen como "0".

## 14.3 LIBRERÍA DE SUBRUTINAS

Una posible librería con las subrutinas que permiten realizar las tareas básicas de control en la memoria EEPROM es la EEPROM.INC donde destacan las subrutinas:

- “EEPROM\_LeeDatos”. El microcontrolador lee el dato que hay escrito en la posición de la EEPROM de datos del PIC apuntada por el contenido del registro de trabajo W. El resultado se proporciona en el registro de trabajo W.
- “EEPROM\_EscribeDatos”. Escribe el dato introducido en el registro de trabajo W en la posición de memoria EEPROM apuntada por el registro EEADR.

Esta librería suficientemente documentada se detalla a continuación, aunque es necesario poseer algunos conocimientos sobre interrupciones que se explican en el capítulo 17 para comprenderla en su integridad.

```
;***** Librería "EEPROM.INC" *****
```

```
; Estas subrutinas permiten realizar las tareas básicas de escritura y lectura de la memoria EEPROM de datos del PIC.
```

```
; Subrutina "EEPROM_LeeDatos"
```

```
; El microcontrolador lee el dato que hay escrito en la posición de la EEPROM del PIC apuntada por el contenido del registro de trabajo W. El resultado se proporciona en el mismo W.
```

```
; Entrada: En (W) la dirección de la memoria EEPROM a leer.
```

```
; Salida : En (W) el byte leído.
```

```
EEPROM_LeeDatos
```

bcf STATUS,RP0	; Asegura que trabaja con el Banco 0.
movwf EEADR	; Dirección a leer.
bsf STATUS,RP0	; Banco 1.
bsf EECON1,RD	; Orden de lectura.

```
EEPROM_SigueLeyendo
```

btfsc EECON1,RD	; El PIC indica que ha terminado la lectura
goto EEPROM_SigueLeyendo	; poniendo en bajo este bit.
bcf STATUS,RP0	; Banco 0.
movf EEDATA,W	; El byte leído al registro W.
return	

```
; Subrutina "EEPROM_EscribeDatos"
```

```
; Escribe el dato introducido en el registro de trabajo W en la posición de memoria EEPROM del PIC apuntada por el registro EEADR.
```

```
; Como altera el valor del registro INTCON al posicionar el flag GIE, éste se debe guardar al principio de la subrutina y restaurarlo al final.
```

```
; Entradas: En el registro EEADR la dirección de la memoria EEPROM a escribir.  
; En el registro W el byte a escribir.
```

```
CBLOCK  
EEPROM_Guard  
ENDC
```

```
EEPROM_EscribeDatos
```

bcf STATUS	STATU
movwf EEDAT	EEAD
movf INTCC	INTCO
movwf EEPROM	EEPRO
bsf STATUS	STATU
bcf INTCC	INTCO
bsf EECOM	EECOM

```
; El fabricante especifica que
```

movlw 0x55	
movwf EECOM	
movlw 0xAA	
movwf EECOM	
bsf EECOM	

```
EEPROM_TerminaEscribir
```

btfsc EECOM	EECON
goto EEPROM	EEPRO
bcf EECOM	EECON
bcf STATUS	STATU
movf EEPROM	EEPRO
movwf INTCC	INTCO
return	

## 14.4 LECTURA

El proceso de especificado en la subru

- 1º El registro EEA
- 2º Puesta a 1 del b
- 3º Espera a que te a “0” el bit RD.
- 4º Lectura en el re

Teniendo en cuenta disponible en EEDATA leerlo mediante la instru

## 14.5 ESCRITURA

La escritura en el trabajando con una E

tareas básicas de subrutinas:

ay escrito en la 1 contenido del de trabajo W. gistro de trabajo o EEADR.

ción, aunque es explican en el

\*\*\*\*\*

untada

ctura

OM del

al

ir.

```

CBLOCK
EEPROM_GuardaINTCON
ENDC

EEPROM_EscribeDatos
    bcf    STATUS,RP0           ; Asegura que trabaja con el Banco 0.
    movwf EEDATA              ; El byte a escribir.
    movf   INTCON,W            ; Reserva el valor anterior de INTCON
    movwf EEPROM_GuardaINTCON
    bsf    STATUS,RP0           ; Acceso al Banco 1.
    bcf    INTCON,GIE          ; Deshabilita todas las interrupciones.
    bsf    EECON1,WREN         ; Habilita escritura.

;
; El fabricante especifica que hay que seguir esta secuencia para escritura en EEPROM:
;

    movlw 0x55
    movwf EECON2
    movlw 0xAA
    movwf EECON2
    bsf    EECON1,WR           ; Inicia la escritura.

EEPROM_TerminaEscribir
    btfsc EECON1,WR           ; Comprueba que termina de escribir en la EEPROM.
    goto  EEPROM_TerminaEscribir
    bcf    EECON1,WREN         ; Desautoriza la escritura en EEPROM.
    bcf    EECON1,EEIF          ; Limpia este flag.
    bcf    STATUS,RP0           ; Acceso al Banco 0.
    movf   EEPROM_GuardaINTCON,W ; Restaura el valor anterior de INTCON.
    movwf INTCON
    return

```

## 14.4 LECTURA DE LA EEPROM DE DATOS

El proceso de lectura de una posición de memoria de la EEPROM está especificado en la subrutina EEPROM\_LeeDatos y comprende los siguientes pasos:

- 1º El registro EEADR debe contener la posición de memoria a leer.
- 2º Puesta a 1 del bit RD del registro EECON1.
- 3º Espera a que termine la operación de lectura, en la cual el microcontrolador pone a “0” el bit RD.
- 4º Lectura en el registro EEDATA del dato direccionado.

Teniendo en cuenta la velocidad de funcionamiento de esta memoria, el dato está disponible en EEDATA después de que el bit RD se ponga a “1”, por lo que es posible leerlo mediante la instrucción que sigue inmediatamente después.

## 14.5 ESCRITURA EN LA EEPROM DE DATOS

La escritura en esta memoria consiste en realidad en una grabación ya que se está trabajando con una EEPROM, es algo más complicada por evidentes razones de

seguridad. Este proceso está especificado en la subrutina EEPROM\_EscribeDatos y comprende los siguientes pasos:

- 1º Se carga en el registro EEDATA el dato a grabar.
  - 2º Se carga en EEADR la dirección de la posición a escribir.
  - 3º Microchip recomienda deshabilitar las interrupciones durante la secuencia de escritura. (En el capítulo 17 se profundizará en esto).
  - 4º Se ejecuta la siguiente secuencia para iniciar la escritura de cada byte:

movlw	0x55
movwf	EECON2
movlw	0xAA
movwf	EECON2
bsf	EECON1,WR

5º Esta última instrucción inicia el proceso de escritura propiamente dicho. Cuando se termina la escritura, el bit EEIF del registro EECON1 se pone a “1”. Al acabar el proceso, el bit WR se pone a “0” automáticamente.

6º Mediante programa, hay que deshabilitar la escritura y poner a “0” el bit EEIF del EECON1.

## 14.6 DIRECTIVA “DE”

La directiva **DE** (*Declare EEPROM Data Byte*) reserva palabras de memoria de 8 bits dentro de la memoria EEPROM de datos, asignándole un valor. Cada expresión reserva un valor de 8 bits y cada carácter de una frase se guarda en una posición separada. Cuando se usa esta directiva debe fijarse un origen en 2100h para su uso con grabadores. Ejemplo:

**ORG** 0x2100 ; Corresponde a la dirección 0 de la zona EEPROM  
; de datos.  
**DE** "Programa EEPROM 04. Versión 2.5. 15-08-2003" .0x00

## 14.7 VENTANA EEPROM EN EL MPLAB

En el simulador del MPLAB el contenido de la memoria EEPROM puede verse seleccionando la opción *View > EEPROM*. Así, por ejemplo, una vez ensamblado el fragmento de programa del ejemplo de la sección anterior, se visualizaría como se indica en la figura 14-2.

## 14.8 PROGRAMA EJEMPLO

El siguiente programa es un ejemplo para una mejor comprensión de la utilización de este recurso, aplicado al esquema de la figura 14-3.

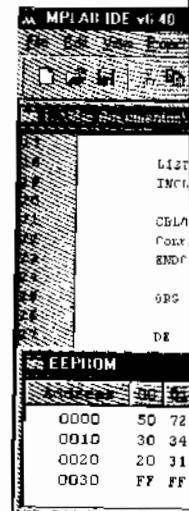


Figura 14-.

M\_EscribeDatos y

e la secuencia de

a byte:

opiamente dicho.  
ECON1 se pone a  
amente.

ra "0" el bit EEIF

s de memoria de 8  
or. Cada expresión  
posición separada.  
uso con grabadores.

zona EEPROM

0x00

'ROM puede verse  
vez ensamblado el  
aría como se indica

ón de la utilización

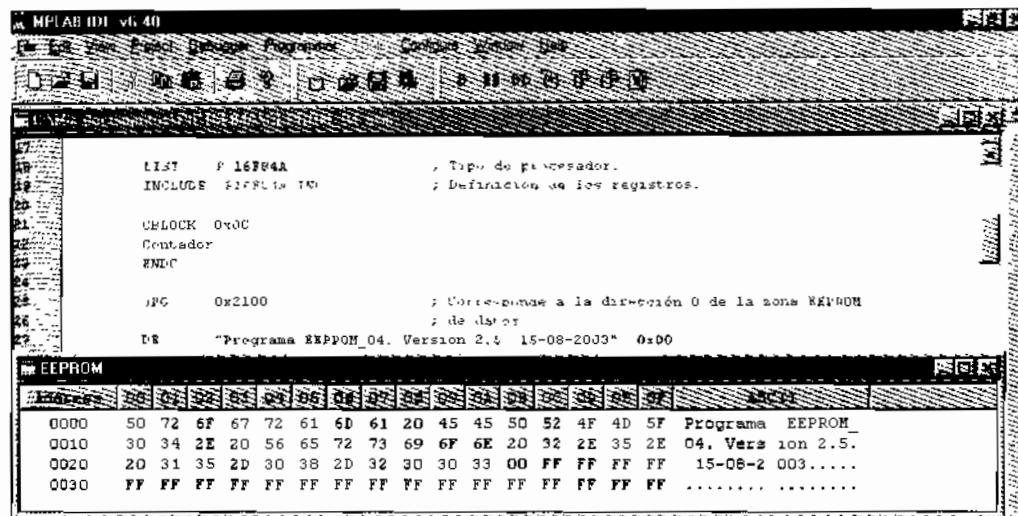


Figura 14-2 Visualización del contenido de la EEPROM de datos en el MPLAB

```
;***** EEPROM_01.asm *****
;
;Este programa comprueba el funcionamiento de la lectura y escritura en la memoria EEPROM de
;datos. Cada vez que el sistema es reseteado se incrementa un contador que se guarda en la
;primera posición de la memoria EEPROM de datos del PIC y es visualizado en el módulo LCD.
;
;ZONA DE DATOS *****
;
;CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
Contador
ENDC

ORG 0x2100 ;Corresponde a la dirección 0 de la zona EEPROM
DE 0x00 ;El contador en principio a cero.

;ZONA DE CÓDIGOS *****
;
Inicio ORG 0 ;El programa comienza en la dirección 0.
    call LCD_Inicializa
    clrw ;Leará la primera posición de memoria EEPROM.
    call EEPROM_LeeDatos
    movwf Contador ;Se guarda en "Contador".
    call BIN_a_BCD ;Se visualiza en BCD
    call LCD_Byt ;con nibble alto apagado si es cero.
    movlw MensajeResetead
    call LCD_Mensaje
```

```

incf    Contador,F      ; Se incrementa.
movf    Contador,W      ; Ahora se graba en la EEPROM de datos.
call    EEPROM_EscribeDatos
Principal
sleep
goto   Principal        ; Pasa a modo de reposo.

```

; "Mensajes" -----

;

Mensajes

```

addwf  PCL,F
MensajeReseteado
    DT " reseteados. ", 0x00
FinMensajes

```

```

INCLUDE <EEPROM.INC>          ; Subrutinas básicas de control de la EEPROM de
INCLUDE <RETARDOS.INC>        ; datos del PIC.
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

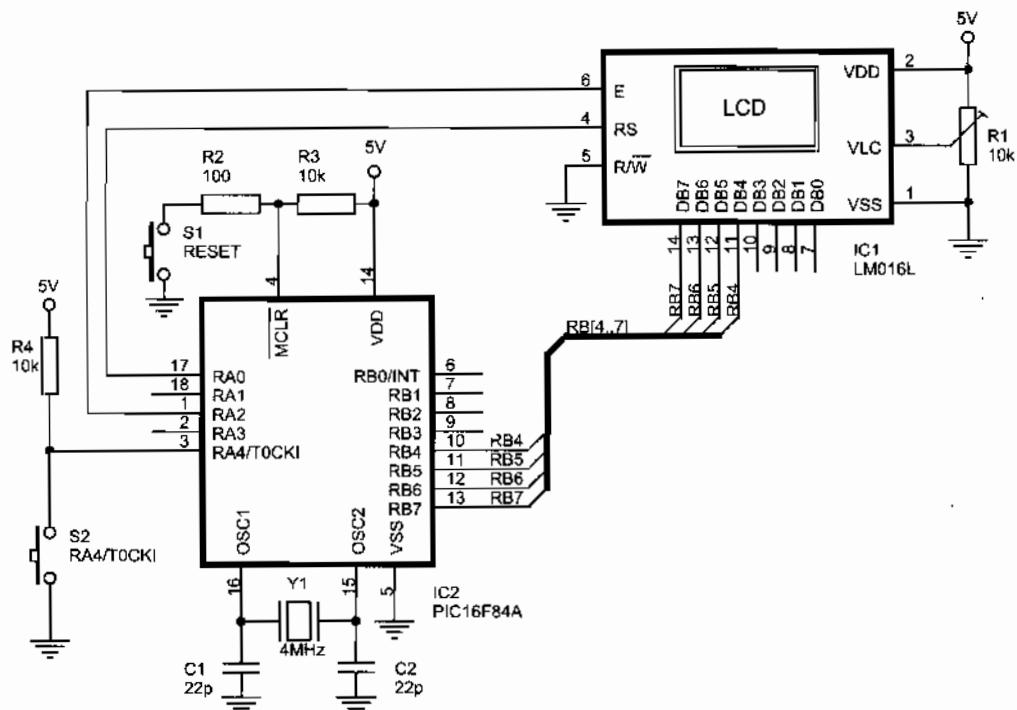


Figura 14-3 Circuito para comprobar los programas de este capítulo

## 14.9 BL

En alg  
número de ve  
crea oportuno  
ejemplifica cr

\*\*\*\*\*

Este programa i  
número determin

Cada vez que el  
posición de la m  
un máximo de n  
que se vuelve a  
a funcionar hay

ZONA DE DAT

CON  
LIST  
INCLU

CBLOCK  
Contad  
ENDC

ORG

DE

NumeroSecreto

ZONA DE CÓD

ORG

Inicio

call  
clrw  
call  
movwf  
movlw  
subwf  
btfs  
goto  
movf  
call  
call  
movlw  
call  
incf  
movf

## 14.9 BLOQUEAR UN CIRCUITO

En algunos proyectos interesa que un circuito funcione sólo un determinado número de veces, es decir, bloquear el funcionamiento del mismo cuando el diseñador lo crea oportuno. Esto es fácil de realizar con ayuda de la EEPROM de datos, tal como se exemplifica en el siguiente programa suficientemente documentado.

```
;***** EEPROM_02.asm *****
;
;Este programa indica un procedimiento para que un prototipo con PIC funcione sólo un
;número determinado de veces fijado por el diseñador.
;
;Cada vez que el sistema es reseteado incrementa un contador que se guarda en la primera
;posición de la memoria EEPROM de datos del PIC y visualiza en la pantalla. El sistema admite
;un máximo de reseteados (por ejemplo 13), a partir del cual ya no funcionará más. Cada vez
;que se vuelva a alimentar el circuito aparecerá un mensaje de bloqueo. Para que el PIC vuelva
;a funcionar hay que volverlo a grabar.
;
;ZONA DE DATOS *****

```

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
Contador
ENDC

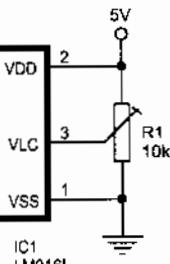
ORG 0x2100 ;Corresponde a la dirección 0 de la zona EEPROM
DE 0x00 ;El contador en principio a cero.

NumeroSecreto EQU .13

;ZONA DE CÓDIGOS *****

```

Inicio call LCD_Inicializa clrw call EEPROM_LeeDatos movwf Contador movlw NumeroSecreto subwf Contador,W btfsc STATUS,C goto ModoBloqueado movf Contador,W call BIN_a_BCD call LCD_Byt movlw MensajeResetead call LCD_Mensaje incf Contador,F movf Contador,W	; El programa comienza en la dirección 0. ; Leerá la primera posición de memoria EEPROM. ; Lo guarda. ; Ahora lo compara con el número secreto. ; Si llega al máximo pasa al modo bloqueado. ; Se visualiza. ; Incrementa el contador. ; Ahora se graba en la EEPROM de datos.
--	---



```

Principal    call    EEPROM_EscribeDatos
sleep
goto    Principal           ; Pasa a modo de bajo consumo.

ModoBloqueado
movlw   MensajeBloqueado
call    LCD_Mensaje
sleep
goto    ModoBloqueado       ; La única forma de salir de este bloqueo
                                ; es volver a grabar el PIC.

Mensajes
addwf   PCL,F
MensajeReseteado
DT "reseteados.", 0x00
MensajeBloqueado
DT "Estoy BLOQUEADO.", 0x00

INCLUDE <EEPROM.INC>          ; Control de la EEPROM de datos del PIC.
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

## 14.10 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador con los siguientes programas. Comprobar su correcto funcionamiento con el esquema de la figura 13-10. El lector puede introducir todas las mejoras que considere convenientes.

**EEPROM\_01.asm:** Cada vez que el sistema es reseteado incrementa un contador que se guarda en la primera posición de la memoria EEPROM de datos del PIC y es visualizado en el LCD.

**EEPROM\_02.asm:** Cada vez que el sistema es reseteado incrementa un contador, que se guarda en la primera posición de la memoria EEPROM de datos del PIC, y se visualiza en la pantalla. El sistema admitirá un máximo de reseteados (por ejemplo 13), a partir del cual ya no funcionará más y, si se vuelve a alimentar el circuito, aparece un mensaje de bloqueo. Para que el PIC vuelva a funcionar habrá que volverlo a grabar.

**EEPROM\_03.asm:** Control de las máquinas de tipo "Su Turno" habituales en múltiples comercios. En el módulo LCD se visualiza el número de turno actual. Este se incrementa cada vez que se presiona el pulsador conectado al pin RA4. La memoria EEPROM de datos del PIC almacena el último número visualizado, de forma que ante un fallo de alimentación se reanude la cuenta en el último número.

Muchos  
distintas acciones  
mediante subrutinas  
utilización de un

## 15.1 EL TIEMPO

Un **timer** preciso entre el **desbordamiento**. Consiste en un inicializado con llegar a su valor

Impulso  
de entrada

El PIC16F84 tiene un contador ascendente que se reinicia cada impulso de

## CAPÍTULO 15

### TIMER 0

Muchos sistemas necesitan un estricto control de los tiempos que duran sus distintas acciones. En todos los programas diseñados hasta ahora ésto se ha realizado mediante subrutinas de retardo. Otro procedimiento más eficaz y preciso consiste en la utilización de un *timer*.

#### 15.1 EL TIMER 0 (TMR0)

Un *timer* se implementa por medio de un contador que determina un tiempo preciso entre el momento en que el valor es cargado y el instante en el que se produce su **desbordamiento**. Un timer típico se describe de forma simplificada en la figura 15-1. Consiste en un contador ascendente (también podría ser descendente) que, una vez inicializado con un valor, su contenido se incrementa con cada impulso de entrada hasta llegar a su valor máximo b'11....11', desbordando y volviendo a comenzar desde cero.



Figura 15-1 Esquema simplificado de un timer

El PIC16F84 dispone de un timer principal denominado *Timer 0* o *TMR0* que es un contador ascendente de 8 bits. El TMR0 se inicializa con un valor, que se incrementa con cada impulso de entrada hasta su valor máximo b'1111111'; con el siguiente impulso de

entrada el contador se **desborda** pasando a valer b'00000000', circunstancia que se advierte mediante la activación del flag de fin de conteo **T0IF** localizado en el registro INTCON.

Los impulsos aplicados al TMR0, pueden provenir de los pulsos aplicados al pin T0CKI o de la señal de reloj interna (Fosc/4), lo que le permite actuar de dos formas diferentes (figura 15-2):

- Como contador de los impulsos que le llegan por el pin RA4/T0CKI.
- Como temporizador de tiempos.

El actuar de una u otra forma depende del bit T0CS del registro OPTION:

- Si T0CS = 1, el TMR0 actúa como contador.
- Si T0CS = 0, el TMR0 actúa como temporizador.

## 15.2 TMR0 COMO CONTADOR

Cuando el TMR0 trabaja como contador se le introducen los impulsos desde el exterior por el pin RA4/T0CKI (*TMR0 External Clock Input*). Su misión es "contar" el número de acontecimientos externos representados por los impulsos que se aplican al pin T0CKI. La figura 15-3 muestra un ejemplo.

El tipo de flanco activo se elige mediante el bit TOSE del registro OPTION:

- Si TOSE = 1, el flanco activo es descendente.
- Si TOSE = 0, el flanco activo es ascendente.

## 15.3 TMR0 COMO TEMPORIZADOR

Cuando el TMR0 trabaja como temporizador cuenta los impulsos de Fosc/4. Se usa para determinar intervalos de tiempo concretos. Estos impulsos tienen una duración conocida de un ciclo máquina que es cuatro veces el periodo de la señal de reloj. Para una frecuencia de reloj igual a 4 MHz el TMR0 se incrementa cada 1  $\mu$ s, tal como se calculó en el capítulo 12.

Como se trata de un contador ascendente el TMR0 debe ser cargado con el valor de los impulsos que se desean contar restados de 256 que es el valor de desbordamiento. Por ejemplo, para contar cuatro impulsos, se carga al TMR0 con  $256 - 4 = 252$ :

- Número de pulsos a contar:  $4_{10} = b'00000100'$ .
- Número a cargar:  $256_{10} - 4_{10} = 252_{10} = b'11111100'$ .
- Incremento a cada ciclo de instrucción:  $b'11111100'$ ,  $b'11111101'$ ,  $b'11111110'$ ,  $b'11111111'$ , aquí se desborda pasando a b'00000000' y activando el flag T0IF.

De esta manera se alcanzando el valor de 4  $\mu$ s si los impulsos son de 1  $\mu$ s.

## 15.4 EL TMR0

El TMR0 es un temporizador de 16 bits que se alimenta de la RAM de datos y que se conecta al bus de datos.

La figura 15-3 muestra el circuito de control de cualquier momento de la señal sobre TMR0 para que se retrase durante un tiempo deseado.

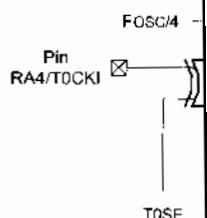


Figura 15-3

## 15.5 DIVISORES

A veces es necesario dividir la frecuencia de los impulsos que incrementa el TMR0. Un divisor programable llamado Prescaler es utilizada por diverso.

En realidad

- El TMR0 es un temporizador de 16 bits.
- El Watchdog es un temporizador de 8 bits que, cuando no recibe impulsos, activa el flag T0IF.

El Prescaler es un divisor de frecuencia que reduce la frecuencia de los impulsos que incrementa el TMR0. Cuando el prescaler es activado, la frecuencia de los impulsos que incrementa el TMR0 es dividida entre el valor de prescaler.

circunstancia que se  
ado en el registro

s aplicados al pin  
ar de dos formas

CKI.

PPTION:

impulsos desde el  
ción es "contar" el  
e se aplican al pin

OPTION:

de Fosc/4. Se usa  
en una duración  
de reloj. Para una  
il como se calculó

ido con el valor de  
sboradamiento. Por  
2:

01', b'11111110',  
do el flag T0IF.

De esta manera, con la llegada de cuatro impulsos, el timer se ha desbordado alcanzando el valor b'00000000' que determina el tiempo de temporización, en este caso 4  $\mu$ s si los impulsos se hubieran aplicado cada microsegundo.

## 15.4 EL TMR0 ES UN REGISTRO DEL SFR

El TMR0 es un registro de propósito especial ubicado en la posición 1 del área SFR de la RAM de datos (figura 4-1). Puede ser leído y escrito al estar conectado directamente al bus de datos.

La figura 15-2 ofrece el esquema de funcionamiento del TMR0. Se puede leer en cualquier momento para conocer el estado de la cuenta. Cuando se escribe un nuevo valor sobre TMR0 para comenzar una nueva temporización, el siguiente incremento del mismo se retrasa durante los dos ciclos de reloj posteriores.

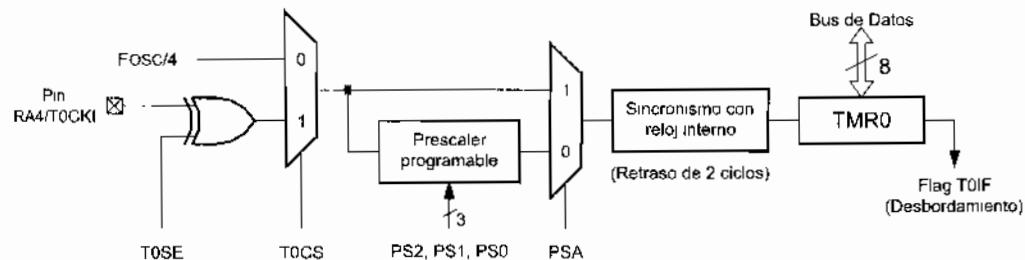


Figura 15-2 Esquema de funcionamiento del timer principal TMR0

## 15.5 DIVISOR DE FRECUENCIA (PRESCALER)

A veces es necesario controlar tiempos largos y aumentar la duración de los impulsos que incrementan el TMR0. Para cubrir esta necesidad se dispone de un circuito programable llamado **Divisor de Frecuencia** o **Prescaler** que divide la frecuencia utilizada por diversos rangos para poder conseguir temporizaciones más largas (fig. 15-2).

En realidad el PIC16F84 dispone de dos temporizadores:

- El TMR0, que actúa como temporizador principal.
- El *Watchdog* (perro guardián), que vigila que el programa no se "cuelgue". Para ello, cada cierto tiempo comprueba que el programa está ejecutándose normalmente y, si no es así, reinicializa todo el sistema. El *Watchdog* se analizará en el próximo capítulo.

El *Prescaler* puede aplicarse a uno de los dos temporizadores, al TMR0 o al *Watchdog*. Cuando se asigna al TMR0 los impulsos pasan primero por el divisor de frecuencia y una vez aumentada su duración se aplican a TMR0 (figura 15-2).

## 15.6 BITS DE CONFIGURACIÓN DEL TMR0

Para controlar el comportamiento del TMR0 se utilizan algunos bits de los registros OPTION e INTCON.

### 15.6.1 Del registro INTCON

El registro INTCON es un registro localizado en la dirección 0Bh del Banco 0 y duplicado en la 8Bh del Banco 1. Contiene los 8 bits que se muestran en la tabla 15-1, de los cuales se utilizará por ahora únicamente el T0IF (figura 15-2):

GIE	EEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 15-1 Bit del registro INTCON utilizado por el Timer 0

- **T0IF (TMR0 Overflow Interrupt Flag bit).** Flag de interrupción del TMR0. Indica que se ha producido un desbordamiento del Timer 0, que ha pasado de b'11111111' a b'00000000'.
  - T0IF = 0: El TMR0 no se ha desbordado.
  - T0IF = 1: El TMR0 se ha desbordado. (Debe borrarse por software).

### 15.6.2 Del registro OPTION

La misión principal del registro OPTION es gobernar el comportamiento del TMR0. Algunos microcontroladores PIC tienen una instrucción denominada también *option*, por ello, el fabricante *Microchip* recomienda darle otro nombre a este registro. Así en el fichero de definición de etiquetas P16F84A.INC se le nombra como OPTION\_REG. Los bits utilizados por los timers son (tabla 15-2):

/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 15-2 Bits del registro OPTION utilizados por el Timer 0

- **PS2:PS0. (Prescaler Rate Select bits).** Bits para seleccionar los valores del Prescaler o rango con el que actúa el divisor de frecuencia, según la tabla 15-3.
- **PSA (Prescaler Assignment bit).** Asignación del divisor de frecuencia
  - PSA = 0: El divisor de frecuencia se asigna al TMR0.
  - PSA = 1: El divisor de frecuencia se asigna al Watchdog.

## 15.7 EJEMPLOS

Un ejemplo de código que muestra la generación de impulsos que se proporcionan por medio de un timer.

```
*****
; Este programa comprueba los impulsos generados por el timer
; aplicados a la línea RA4/INT0
; el pulsador se incrementa el divisor de frecuencia
; Como es un incremento permanente, el divisor de frecuencia al final
; de la ejecución es de 1024
; ZONA DE DATOS ****
```

```
CONFIG _C
LIST P=1
INCLUDE <P16F84A.H>
```

```
CBLOCK 0x0C
```

PS2 PS1 PS0	Divisor del TMR0	Divisor 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

Tabla 15-3 Selección del rango del divisor de frecuencia

- **T0SE (TMR0 Source Edge Select bit).** Selecciona flanco de la señal de entrada del TMR0.
  - T0SE = 0. TMR0 se incrementa en cada flanco ascendente de la señal aplicada al pin RA4/T0CKI.
  - T0SE = 1. TMR0 se incrementa en cada flanco descendente de la señal aplicada al pin RA4/T0CKI.
- **T0CS (TMR0 Clock Source Select bit).** Selecciona la fuente de señal del TMR0.
  - T0CS = 0. Pulsos de reloj interno Fosc/4 (TMR0 como temporizador).
  - T0CS = 1. Pulsos introducidos a través del pin RA4/T0CKI (TMR0 como contador).

## 15.7 EJEMPLO DEL TMR0 COMO CONTADOR

Un ejemplo de programación del TMR0 como contador se describe a continuación, referido al esquema de la figura 15-3. Se trata de visualizar en el display LCD el número de impulsos que se aplican al pin TOCKI (*Timer 0 Clock Input*). Estos pulsos se proporcionan por medio de la pulsación de S1.

```
***** Timer0_01.asm *****
;
; Este programa comprueba el funcionamiento del Timer 0 como contador de los impulsos
; aplicados a la línea RA4/T0CKI, donde se ha conectado un pulsador. Cada vez que presiona
; el pulsador se incrementa un contador visualizado en el display LCD.
; Como es un incremento por cada impulso aplicado al pin TOCKI no es necesario asignarle
; divisor de frecuencia al TMR0, por tanto, el Prescaler se asigna al Watchdog.
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
```

ENDC

; ZONA DE CÓDIGOS \*\*\*\*

```

ORG 0
Inicio
    call LCD_Inicializa
    bsf STATUS,RP0           ; Acceso al Banco 1.
    movlw b'00111000'         ; TMR0 como contador por flanco descendente de
    movwf OPTION_REG          ; RA4/T0CKI. Prescaler asignado al Watchdog.
    bcf STATUS,RP0           ; Acceso al Banco 0.
    clrf TMR0                ; Inicializa el contador.

; La sección "Principal" es de mantenimiento. Sólo se dedica a visualizar el Timer 0.

Principal
    call LCD_Lineal          ; Se pone al principio de la línea 1.
    movf TMR0,W               ; Lee el Timer 0.
    call BIN_a_BCD            ; Se debe visualizar en BCD.
    call LCD_Byt              ; Visualiza apagando las decenas en caso de que sean 0.
    goto Principal

INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END

```

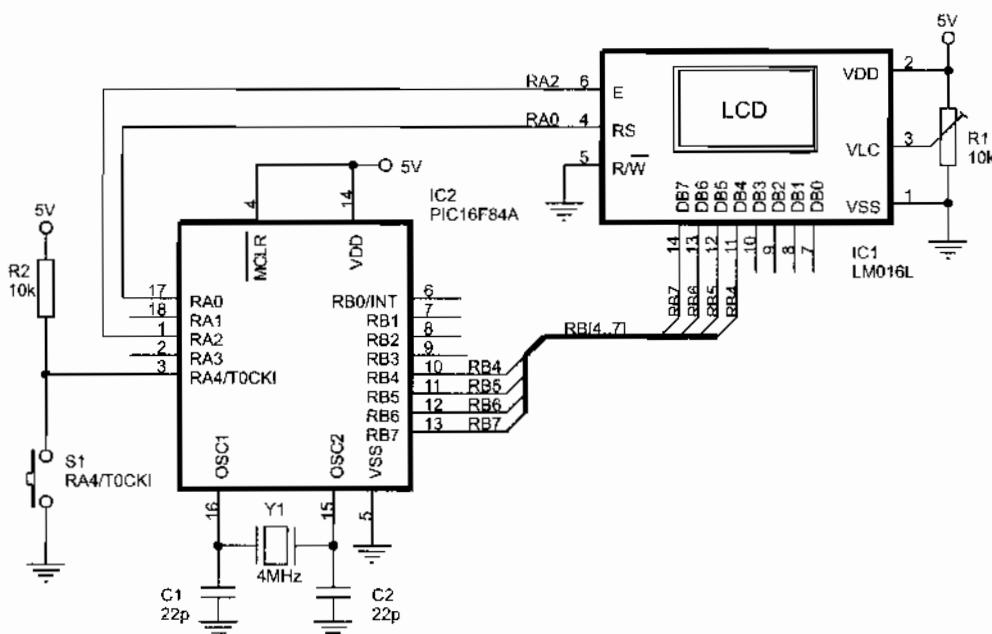


Figura 15-3 Contador de los impulsos introducidos por el pin T0CKI

## 15.8 EJEMPLO

El principal  
cálculo de los tiem

Te

Donde:

- *Temporizad*
- $T_{CM}$  es el p
- ha calculado
- *Prescaler*, e
- (256-Carga
- de desborda
- inicialmente

EJEMPLO.  
us si se utiliza un Pre

Solución: Sus

Ter

de donde se de

Así pues el val

TMRO\_Carga500us EQ

O también se p

TMRO\_Carga500us EQ

Al trabajar con

TMRO\_Carga500us EQ

En esta ecuaci  
ejemplo pudiera ser el

Por la línea 3 del puerto B  
semiperiodo dura 500  $\mu$ s.  
utilización del Timer 0 de

## 15.8 EJEMPLO DEL TMR0 COMO TEMPORIZADOR

El principal problema cuando se configura el TMR0 como temporizador es el cálculo de los tiempos de temporización. Se puede utilizar la siguiente fórmula:

$$\text{Temporización} = T_{CM} \cdot \text{Prescaler} \cdot (256 - \text{Carga TMR0})$$

Donde:

- *Temporización*, es el tiempo deseado.
- $T_{CM}$ , es el periodo de un ciclo máquina e igual a  $T_{CM} = 4 T_{OSC}$ . Para 4 MHz ya se ha calculado en anteriores ocasiones  $T_{CM} = 4 \cdot 1/f = 4 \cdot \frac{1}{4} = 1 \mu\text{s}$ .
- *Prescaler*, es el rango de divisor de frecuencia elegido.
- $(256 - \text{Carga TMR0})$ , es el número total de impulsos a contar por el TMR0 antes de desbordarse en la cuenta ascendente. “*Carga TMR0*” es el valor cargado inicialmente en el TMR0 tal como se ha explicado en la sección 15.3

**EJEMPLO.** ¿Qué valor hay que cargar en el TMR0 para lograr un tiempo de 500  $\mu\text{s}$  si se utiliza un Prescaler de 2?

**Solución:** Sustituyendo en la ecuación anterior queda:

$$\begin{aligned} \text{Temporización} &= T_{CM} \cdot \text{Prescaler} \cdot (256 - \text{Carga TMR0}) \\ 500 &= 1 \times 2 \cdot (256 - \text{Carga TMR0}) \end{aligned}$$

de donde se deduce que el valor a cargar tiene que ser:

$$\text{Carga TMR0} = 6$$

Así pues el valor de carga se podría poner en el programa:

TMRO\_Carga500us EQU d'6' ; Número a cargar para contar 250 pulsos antes de desbordarse.

O también se puede poner:

TMRO\_Carga500us EQU d'256-d'250' ; Igual a d'6' = b'00000110'

Al trabajar con 8 bits es mejor ponerlo de esta forma más rápida:

TMRO\_Carga500us EQU -d'250' ; d'250' = b'11111010' y -d'250' = b'00000110'

En esta ecuación se desprecian los pequeños tiempos de carga, saltos, etc. Un ejemplo pudiera ser el siguiente programa para el esquema de la figura 15-4.

\*\*\*\*\* Timer0\_02.asm \*\*\*\*\*

```
; Por la linea 3 del puerto B se genera una onda cuadrada de 1 kHz, por tanto, cada
; semiperiodo dura 500 µs. Los tiempos de temporización se consiguen mediante la
; utilización del Timer 0 del PIC.
```

; A la línea de salida se puede conectar un altavoz, tal como se indica en el esquema ; correspondiente, con lo que se escuchará un pitido.

; El cálculo de la carga del TMR0 se hará de forma simple despreciando el tiempo que ; tardan en ejecutarse las instrucciones.

; ZONA DE DATOS \*\*\*\*\*

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK0x0C
ENDC
```

```
#DEFINE     Salida    PORTB,3
```

; ZONA DE CÓDIGOS \*\*\*\*\*

Inicio	ORG 0	
	bsf STATUS,RP0	; Acceso al Banco 1.
	bcf Salida	; Esta línea se configura como salida.
	moviw b'00000000'	
	movwf OPTION_REG	; Prescaler de 2 asignado al TMR0
	bcf STATUS,RP0	; Acceso al Banco 0.
Principal	bsf Salida	; La salida pasa a nivel alto
	call Timer0_500us	; durante este tiempo.
	bcf Salida	; La salida pasa a nivel bajo
	call Timer0_500us	; durante este tiempo.
	goto Principal	

; Subrutina "Timer0\_500us"

; Como el PIC trabaja a una frecuencia de 4 MHz, el TMR0 evoluciona cada microsegundo.  
; Para conseguir un retardo de 500 µs con un prescaler de 2 el TMR0 debe contar 250  
; impulsos. Efectivamente: 1 µs x 250 x 2 = 500 µs.

; Comprobando con la ventana Stopwatch" del simulador se obtienen unos tiempos para la onda  
; cuadrada de 511 µs para el nivel alto y 513 µs para el bajo.

```
TMRO_Carga500us EQU d'256-d'250'
```

Timer0_500us	movlw TMRO_Carga500us	; Carga el Timer 0.
	movwf TMRO	
	bcf INTCON,T0IF	; Resetea el flag de desbordamiento del TMR0.
Timer0_Rebosamiento	btfss INTCON,T0IF	; ¿Se ha producido desbordamiento?
	goto Timer0_Rebosamiento	; Todavía no. Repite.
	return	

IC1  
PIC16F

17  
18  
1  
2  
3

C1  
22p

Figura

El procedimiento para la mayoría de las tareas precisas hay que tener en cuenta el valor de carga del prescaler. En el caso de las instrucciones *nop*, el resultado es que el simulador del MPLAB no ejecuta el siguiente programa:

```
*****  
;  
; Por la línea 3 del puerto de salida, la señal de salida tiene un semiperíodo de 500 µs de duración, lo que implica una utilización del Timer 0 de 500 µs.  
;  
; A esta línea de salida se le aplica la configuración de salida correspondiente, con lo que el resultado es que la señal de salida tiene un periodo de 1 ms.  
;  
; El cálculo de la carga del prescaler es simple: para conseguir un retardo de 500 µs, las instrucciones para cargar el prescaler del TMR0 hay que ayudarlas a que se ejecuten.
```

; Comprobando con la ventana Stopwatch del simulador se obtienen unos tiempos para la onda cuadrada de 511  $\mu$ s para el nivel alto y 513  $\mu$ s para el bajo.

END

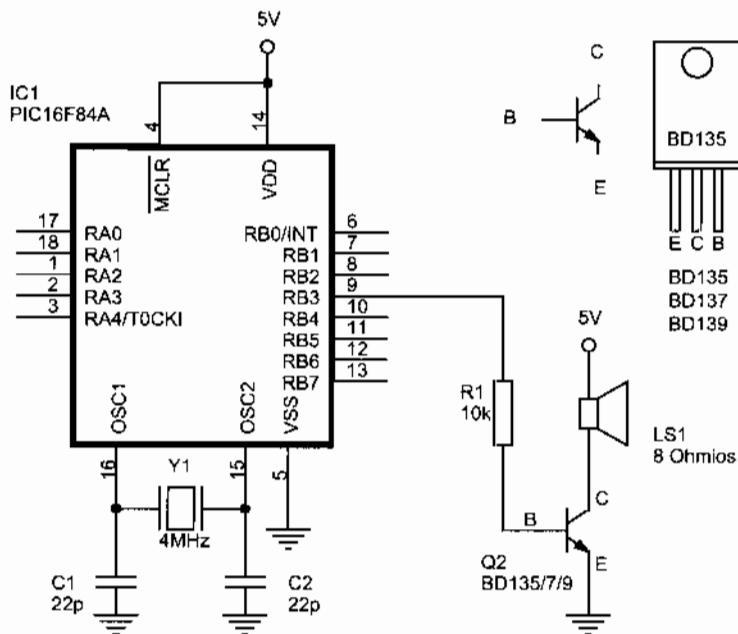


Figura 15-4 Generador de ondas cuadradas utilizando el Timer 0

El procedimiento explicado para hallar el tiempo de temporización es suficiente para la mayoría de las aplicaciones. Sin embargo en caso de requerir temporizaciones precisas hay que tener en cuenta el tiempo de ejecución de las instrucciones, saltos, etc. El valor de carga del TMR0 será algo menor y habrá que hacer un ajuste fino con instrucciones *nop*. La forma más sencilla de calcularlo es experimentando con el simulador del MPLAB y el reloj de la ventana *Debugger > Stopwatch* (figura 12-3). El siguiente programa es un ejemplo de ello.

```
;***** Timer0_03.asm *****  
;  
; Por la línea 3 del puerto B se genera una onda cuadrada de 1 kHz, por tanto, cada  
; semiperíodo dura 500 µs. Los tiempos de temporización se consiguen mediante la  
; utilización del Timer 0 del PIC.  
;  
; A esta línea de salida se puede conectar un altavoz, tal como se indica en el esquema  
; correspondiente, con lo que se escuchará un pitido.  
;  
; El cálculo de la carga del TMR0 se hará de forma que se tenga en cuenta los tiempos de  
; las instrucciones para conseguir tiempos exactos. Para calcular los valores de carga  
; del TMR0 hay que ayudarse del simulador del MPLAB y de la ventana de reloj Stopwatch.
```

; ZONA DE DATOS \*\*\*\*\*

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK0x0C
ENDC
```

```
#DEFINE Salida PORTB,3
```

; ZONA DE CÓDIGOS \*\*\*\*\*

	ORG	0	
Inicio	bsf	STATUS,RP0	; Acceso al Banco 1.
	bcf	Salida	; Esta línea se configura como salida.
	movlw	b'00000000'	
	movwf	OPTION_REG	; Prescaler de 2 para el TMR0
	bcf	STATUS,RP0	; Acceso al Banco 0.
Principal	bsf	Salida	; La salida pasa a nivel alto
	call	Timer0_500us	; durante este tiempo.
	nop		; Dos ciclos mediante "nop" para compensar
	nop		; la instrucción "goto Principal" del nivel bajo.
	bcf	Salida	; La salida pasa a nivel bajo
	call	Timer0_500us	; durante este tiempo.
	goto	Principal	

; Subrutina "Timer0\_500us" -----

; Con el simulador se comprueba que se obtienen unos tiempos para la onda cuadrada  
; de 1kHz exactos, 500 µs tanto para el nivel alto como para el bajo.

TMR0_Carga500us EQU	-d'242'	; Este valor se ha obtenido experimentalmente ; con ayuda del simulador del MPLAB.
---------------------	---------	---

Timer0_500us	nop	; Algunos "nop" para ajustar a 500 µs exactos.
	nop	
	movlw	TMR0_Carga500us ; Carga el Timer 0.
	movwf	TMR0
	bcf	INTCON,T0IF ; Resetea el flag de desbordamiento del TMR0.
Timer0_Rebosamiento	btfs	INTCON,T0IF ; ¿Se ha producido desbordamiento?
	goto	Timer0_Rebosamiento ; Todavia no. Repite.
	return	

```
END
```

## 15.9 PRÁCTICA

Respectando el resumen visto en clase, se simulará y grabará el programa para comprobar su correcto funcionamiento. Se podrán aplicar todas las mejoras que se consideren necesarias.

**Timer0\_01.asm**  
RA4/T0CKI se incrementa cada 100 µs.

**Timer0\_02.asm**  
RA4/T0CKI se incrementa cada 1 kHz, así pues, cada 100 µs se incrementará en 1. Se realizarán mediante la instrucción ADDWF RA4, F0. Se conectará un altavoz, que escuchará un pitido cada 100 µs. Se despreciando el tiempo entre la ejecución de la instrucción ADDWF y la generación del pitido.

**Timer0\_03.asm**  
RA4/T0CKI se incrementa cada 100 µs. El programa hará de forma que se generen tiempos exactos. Para ello, se utilizará el simulador del MPLAB para comprobar los tiempos reales obtenidos.

**Timer0\_04.asm**  
RA4/T0CKI se incrementa cada 100 µs. Se generan 500 µs en alto y 300 µs en bajo.

**Timer0\_05.asm**  
RA4/T0CKI se incrementa cada 1 segundo, rebotes incluidos.

**Timer0\_06.asm**  
RA4/T0CKI se incrementa cada 100 µs. La frecuencia máxima a la que se puede incrementar es de 1 kHz. Se realizará un frecuencímetro que permita desarrollarlo como proyecto.

## 15.9 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador con los siguientes programas. Comprobar su correcto funcionamiento con el esquema de la figura 13-10. El lector puede introducir todas las mejoras que considere conveniente.

**Timer0\_01.asm:** Cada vez que se actúe sobre el pulsador conectado a la línea RA4/TOCKI se incrementa un contador que se visualiza en la pantalla del módulo LCD.

**Timer0\_02.asm:** Por la línea 3 del Puerto B se genera una onda cuadrada de 1 kHz, así pues, cada semiperíodo durará 500  $\mu$ s. Los tiempos de temporización se realizarán mediante la utilización del Timer 0 del PIC. A esta línea de salida se puede conectar un altavoz, tal como se indica en el esquema correspondiente, con lo que se escuchará un pitido. El cálculo de la carga del TMR0 se hará de forma simple, despreciando el tiempo que tarda en ejecutarse las instrucciones. Compruébelo con el MPLAB los tiempos reales obtenidos en alto y en bajo.

**Timer0\_03.asm:** Igual que el anterior pero el cálculo de la carga del TMR0 se hará de forma que se tenga en cuenta los tiempos de las instrucciones para conseguir tiempos exactos. Para calcular los valores de carga del TMR0 hay que ayudarse del simulador del MPLAB y de la ventana de reloj *Stopwatch*. Compruebe con el MPLAB los tiempos reales obtenidos en alto y en bajo.

**Timer0\_04.asm:** Por la línea 3 del Puerto B se genera una onda rectangular de 500  $\mu$ s en alto y 300  $\mu$ s en bajo.

**Timer0\_05.asm:** Cuenta el número de veces que se presiona un pulsador durante 1 segundo, rebotes incluidos.

**Timer0\_06.asm:** Frecuencímetro elemental para la señal aplicada al pin RA4. La frecuencia máxima a medir será 255 Hz. Este programa muestra el fundamento para realizar un frecuencímetro digital completo con diferentes escalas. Se anima al lector a desarrollarlo como proyecto.

pensar  
vel bajo.

talmente  
3.

exactos.

el TMR0.

*Hernán Romero*

## CAPÍTULO 16

# OTROS RECURSOS

El PIC16F84 posee algunos **recursos especiales** que le confieren una gran potencia:

- La memoria EEPROM de datos (capítulo 14).
- El *Timer 0* (capítulo 15).
- El *Watchdog* (WDT).
- El modo bajo consumo o *sleep*.
- El direccionamiento indirecto.
- Resistencias de *Pull-Up* en el Puerto B.
- Las interrupciones (capítulo 17).

El ensamblador MPASM ofrece también otros recursos como las macros que facilitan las tareas de programación y diseño.

En este tema se analizarán el *Watchdog*, la instrucción *sleep*, el direccionamiento indirecto, las macros y las resistencias de *Pull-Up* del Puerto B.

### 16.1 EL WATCHDOG (WDT)

El nombre del *Watchdog* (perro guardián) pretende reflejar la función de este recurso: "vigilar" que el programa no se "cuelgue" y dejen por esto de ejecutarse las instrucciones, tal como lo ha previsto el diseñador. Para realizar esta labor de vigilancia, el Watchdog da un paseo por la CPU cada cierto tiempo, asegurándose que el programa se ejecuta normalmente; en caso contrario (por ejemplo si el control está detenido en un bucle infinito o a la espera de algún acontecimiento que no se produce), el Watchdog "ladrá" y provoca un reset, reinicializando todo el sistema.

El Watchdog es un temporizador de 8 bits cuyo objetivo es generar un reset general cuando se desborda su cuenta (figura 16-1). Su control de tiempo es independiente del oscilador principal del microcontrolador y se basa en una red RC interna que fija un periodo de oscilación de 18 ms. Este periodo puede ampliarse haciendo uso del Prescaler con el que se puede conseguir una relación de hasta 1:128 que corresponde a 2,3 segundos de temporización según la tabla 15-3.

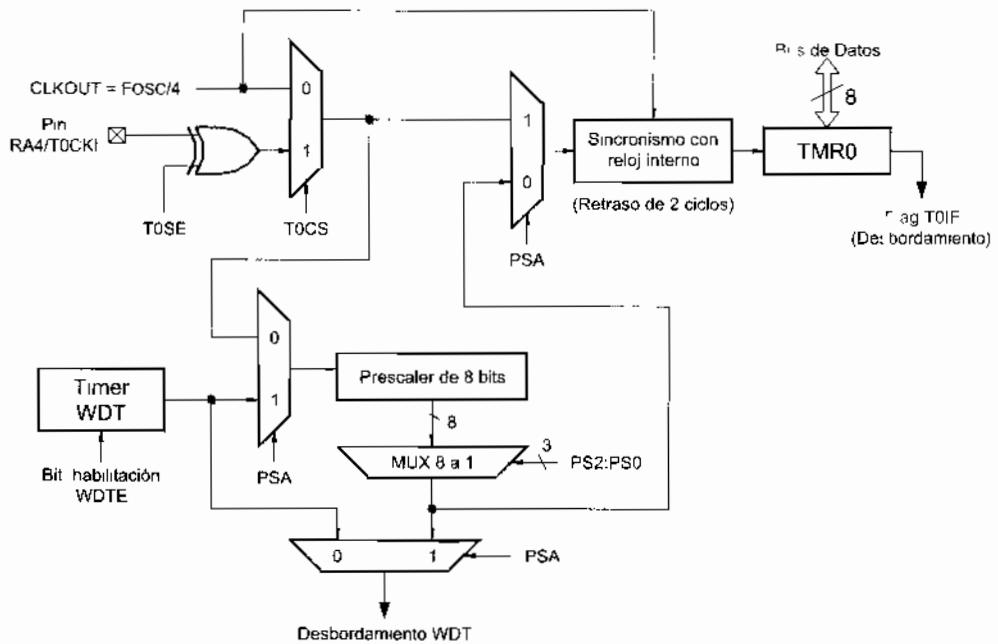


Figura 16-1 Diagrama de bloques del Prescales para Timer 0 y Watchdog

El **refresco** del Watchdog consiste en la puesta a cero del mismo para comenzar la cuenta, evitando así su desbordamiento y el reset que esto provoca. Esta puesta a cero sólo se puede realizar por software mediante la instrucción `clrwdt`. El diseñador deberá analizar el programa y situar esta instrucción en los lugares más adecuados por los que transcurre el flujo del programa antes de que finalice el tiempo que controla el Watchdog. De esta manera si el programa se “cuelga” (bucle infinito, espera de acontecimiento que no se produce, etc.) el Watchdog no se refrescará y al completar su temporización provocará el reset del sistema.

El Watchdog se habilita, o no, durante el proceso de grabación del microcontrolador. Prácticamente todos los software para grabadores de microcontroladores PIC tienen en sus menús la opción específica de activar o desactivar el Watchdog. En la figura 16-2 se aprecia para el IC-Prog.

IC-Prog 1.95	
Archivo	Edición
...	...
Dirección - Cód.	
0000: 206	
0008: 205	
0010: 000	
0018: 304	
0020: 008	
0028: 103	
0030: 300	
0038: 281	
0040: 303	
0048: 306	
0050: 284	
0058: 0212	
Dirección - Dato	
10000: FF E	

Figura 16-2 D...

La habilitación de la opción `WDT ON`

#### CONFIG

La utilización del Watchdog se debe utilizarse en aplicaciones que requieren funcionamiento no interrumpible y no exageradamente largo.

Un error de programación en el Watchdog inconsciente puede hacer que los programas no funcionen correctamente. Si simplemente el programa se ejecuta sin problemas, el Watchdog está desabilitado. Si el Watchdog hay que comprobarlo, se debe deshabilitar la función.

## 16.2 MODO DE PROGRAMACIÓN

Si se analiza el modo de programación durante muchos intentos de programación, se observa que el microcontrolador permanece en modo de programación incluso después de un reinicio. Esto es debido a que el microcontrolador permanece alimentado por la energía llamada modo de programación.

El microcontrolador permanece alimentado por la energía llamada modo de programación.

un reset general independiente del sistema que fija un uso del Prescaler de a 2.3 segundos

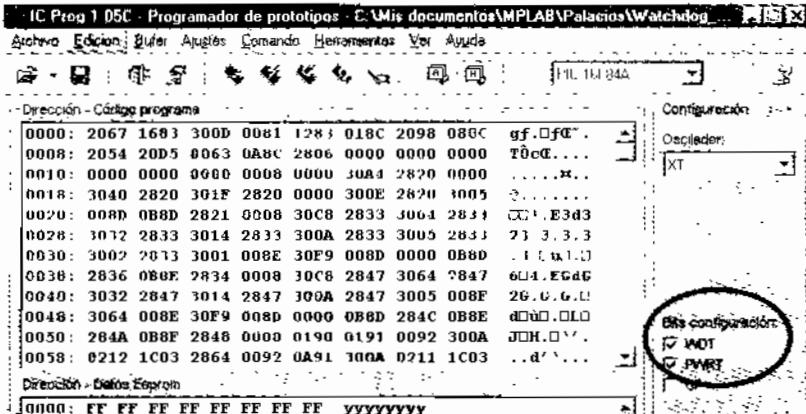
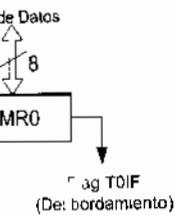


Figura 16-2 Detalle del IC-Prog donde se aprecia la habilitación del Watchdog

La habilitación del Watchdog también se puede hacer desde el programa mediante la opción *WDT\_ON* de la directiva *\_CONFIG*, tal como se indica:

```
_CONFIG _CP_OFF & _WDT_ON & _PWRTE_ON & _XT_OSC
```

La utilización del Watchdog es decisión del diseñador. Este recurso no suele utilizarse en aplicaciones sencillas. Sirve para evitar posibles problemas de funcionamiento no controlables como, por ejemplo, bucles infinitos, esperas exageradamente largas de alguna determinada entrada, finales de carrera rotos, etc.

Un error de programación frecuente en diseñadores noveles consiste en habilitar el Watchdog inconscientemente o por accidente. Con desesperación comprueban como sus programas no funcionan correctamente, se resetean nada más comenzar a funcionar o simplemente el programa diseñado no realiza operación alguna. El problema es que el Watchdog está desbordándose cada poco tiempo y el sistema nunca llega a funcionar más allá de las primeras instrucciones. La solución es sencilla: si no se va a utilizar el Watchdog hay que comprobar siempre que la casilla WDT del software del grabador está deshabilitada.

## 16.2 MODO DE BAJO CONSUMO O “SLEEP”

Si se analiza un sistema en una situación real en que es fácil comprobar que, durante muchos intervalos de tiempo el microcontrolador está a la espera de un acontecimiento exterior, está parado. Durante estos períodos se podría “apagar” el microcontrolador para lograr un ahorro de energía. Esto es muy importante sobre todo en los sistemas alimentados por medio de pilas.

El microcontrolador PIC dispone de un modo de funcionamiento de ahorro de energía llamado modo de bajo consumo, reposo, *standby* o **SLEEP**. Para entrar en este

modo de funcionamiento se dispone de la instrucción homónima al estado que produce denominada *sleep*.

En este estado el microcontrolador queda “dormido” (*sleep* viene a significar “sueño” en inglés), deteniendo el reloj principal y sus circuitos asociados con el consiguiente ahorro de energía. Al activarse una “interrupción”, ocasionada por algún acontecimiento esperado, el microcontrolador se despierta y reanuda su trabajo.

Así pues, el modo de bajo consumo es un modo especial de funcionamiento al que se entra por medio de la instrucción *sleep* provocando los siguientes acontecimientos:

- El consumo del microcontrolador baja a unos pocos microamperios.
- Si el Watchdog está habilitado se refresca su valor, pero seguirá funcionando normalmente.
- El oscilador principal del sistema deja de funcionar.
- El timer TMR0 tampoco funciona.
- Los puertos de entrada/salida mantienen el mismo estado que tenían antes de ejecutar *sleep*.
- Los bits TO y PD del registro de estado STATUS toman los valores “1” y “0” respectivamente.

Tras una instrucción *sleep* el microcontrolador permanece en modo bajo consumo hasta que “despierte” por alguna de las razones siguientes:

- El pin de reset MCLR es activado a nivel bajo, generándose un reset.
- El Watchdog que estaba habilitado cuando se ejecutó *sleep* se desborda y genera un reset.
- Por una “interrupción” (que se explicará en el próximo capítulo) que no sea por desbordamiento de TMR0, ya que este no funciona en modo *sleep*.

La causa de un reset se puede identificar examinando los bits TO y PD del registro de estado o STATUS. La tabla 16-1 detalla estos dos bits:

IRP	RPI	RP0	/TO	/PD	Z	DC	C
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 16-1 Bits del registro STATUS utilizados por el Watchdog y el modo SLEEP

- **/PD (Power Down).** Flag de bajo consumo. Es un bit de sólo lectura, no puede ser escrito por el usuario. Sirve para detectar el modo de bajo consumo.
  - /PD = 0. Al ejecutar la instrucción *sleep* y entrar en reposo.
  - /PD = 1. Tras conectar la alimentación V<sub>DD</sub> o al ejecutar la instrucción *clrwdt*.

- **/TO (Timer Output).** Bit de sólo lectura, indica el resultado del circuito de vigilancia. Se activa cuando una condición de overflow en el timer TMR0 es detectada.
  - /TO = 0. No se ha detectado una condición de overflow.
  - /TO = 1. Se ha detectado una condición de overflow.

Cuando sale de la instrucción *sleep*, el bit /TO se establece en función de si el Watchdog o por una interrupción.

En el siguiente programa se muestra cómo se puede probar en el circuito.

```
*****  
;  
; Este programa comprueba el modo de reposo.  
; La forma de detectar la causa de despertar:  
;  
; Se conectará un pulsador al pin P1.0.  
; que se visualizará en el monitor de depuración.  
; despertará y en la pantalla aparecerá:  
;  
; Con un Prescaler de 64 el timer TMR0  
; 18 x 64 = 1152 ms, es decir, 1.152 segundos.  
;  
; ZONA DE DATOS *****
```

#### CONFIG \_CP

; Observad que el Watchdog se habilita.

```
LIST P=16  
INCLUDE <PI16F84.H>
```

```
CBLOCK 0x0C  
Contador  
ENDC
```

#DEFINE Pulsador PORTA.0

; ZONA DE CÓDIGOS \*\*\*

ORG	0
Inicio	
call	LCD Init
btfss	STATUS, 0
goto	ResetPort
movlw	Mensaje
call	LCD Message
bsf	STATUS, 0

- **/TO (Timer Out).** Flag indicador de fin temporización del Watchdog. Es un bit de sólo lectura, no puede ser escrito por el usuario. Se activa en “0” cuando el circuito de vigilancia Watchdog finaliza la temporización. Sirve para detectar si una condición de reset fue producida por el *Watchdog Timer*.
  - /TO = 0. Al desbordar el temporizador del Watchdog.
  - /TO = 1. Tras conectar V<sub>DD</sub> (funcionamiento normal) o al ejecutar las instrucciones *clrwdt* o *sleep*.

Cuando sale del modo de bajo consumo, por desbordamiento del temporizador Watchdog o por una interrupción, se ejecuta la instrucción siguiente a la *sleep*.

En el siguiente programa se muestra un ejemplo de aplicación de todo esto, que se puede probar en el circuito de la figura 14-3.

```
***** Watchdog_02.asm *****
;
; Este programa comprueba el funcionamiento del Watchdog, de la instrucción "sleep" y
; la forma de detectar la causa de un reset.
;
; Se conectará un pulsador al pin RA4 y cada vez que se presione se incrementará un contador
; que se visualizará en el modulo LCD. Si tarda más de 1 segundo en pulsar el Watchdog se
; despertará y en la pantalla aparecerá un mensaje.
;
; Con un Prescaler de 64 el desbordamiento del Watchdog se producirá cada
; 18 x 64 = 1152 ms, es decir, cada segundo aproximadamente.
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_ON & _PWRTE_ON & _XT_OSC

; Observad que el Watchdog está habilitado.

LIST      P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK0x0C
Contador
ENDC

#define Pulsador PORTA,4

; ZONA DE CÓDIGOS *****

ORG      0
Inicio
call    LCD_Inicializa
btfs   STATUS,NOT_TO
goto   ResetPorWatchdog ; Si.
movlw  MensajePulsacion
call    LCD_Mensaje
bsf    STATUS,RP0
;
; ¿El reset fué producido por el Watchdog?
; No. Pues entonces aparece el texto
; " pulsaciones".
; Acceso al Banco 1.
```

```

bsf     Pulsador          ; La línea RA4 configurada como entrada.
movlw   b'00001110'        ; Un prescaler de 64 es asignado al Watchdog.
movwf   OPTION_REG
bcf    STATUS,RP0          ; Acceso al Banco 0.
clrf   Contador           ; Inicializa el contador y lo visualiza.
call   VisualizaContador
call   Retardo_2s          ; Retardo inicial de un par de segundos.

Principal
btfsf  Pulsador           ; Lee el pulsador.
call   IncrementaVisualiza ; Salta a incrementar y visualizar el contador.
goto   Principal           ; Si se produce un reset por desbordamiento del Watchdog aparece un mensaje
; significativo y sólo puede salir de aquí apagando o resemando el sistema.

ResetPorWatchdog
movlw   MensajeWatchdog
call   LCD_Mensaje
sleep

; Subrutina "IncrementaVisualiza"
IncrementaVisualiza
clrwdt                         ; Resetea el Watchdog.
call   Retardo_20ms
btfsf  Pulsador
goto   FinIncrementa
incf   Contador,F             ; Incrementa el contador.

VisualizaContador
call   LCD_Linea1              ; Se pone al principio de la línea 1.
movf   Contador,W
call   BIN_a_BCD               ; Se debe visualizar en BCD.
call   LCD_Byt

EsperaDejePulsar
clrwdt                         ; Resetea el Watchdog mientras mantenga pulsado.
btfsf  Pulsador
goto   EsperaDejePulsar

FinIncrementa
return

; Mensajes
addwf   PCL,F
MensajePulsacion
DT " pulsaciones ",0x00
MensajeWatchdog
DT " Eres LENTO",0x00
;

INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

## 16.3 DIRE

Las instrucciones tienen cinco modos de dirección:

- Dirección directa: contiene una constante.
- Dirección indirecta: misma constante.
- Dirección por ejemplos.
- Dirección por instrucción.
- Dirección por registro: dirección de un registro.

Los direcciones utilizadas hasta el momento en los microcontroladores.

Para el dirección dentro de la zona de Banco 0, aunque es

El registro de cualquier instrucción tiene dirección donde apunta al valor 14h una instrucción. Se puede decir en forma indirecta a través del contenido del registro F.

Siguiendo con el contenido de la página en la figura 16-3(B).

### 16.3 DIRECCIONAMIENTO INDIRECTO

Las instrucciones del PIC16F84 pueden especificar los datos u operandos mediante cinco modos de direccionamiento:

- **Direccionamiento Inmediato.** El valor del dato inmediato (su valor *literal*) lo contiene el mismo código de operación que, en la ejecución de la instrucción, se carga en el registro W para su posterior procesamiento. Por ejemplo *iorlw Constante*.
- **Direccionamiento Directo.** La dirección de memoria RAM se encuentra en el mismo código de operación. Por ejemplo *movwf TMR0* o *addwf Variable*.
- **Direccionamiento de Bit.** Procesa datos de un bit. La dirección del dato es un bit. Por ejemplo *bcf STATUS,RP0*.
- **Direccionamiento Indexado.** Utilizado para el manejo de tablas mediante la instrucción *addwf PCL,F*.
- **Direccionamiento Indirecto.** La dirección del dato se encuentra contenida en el registro *INDF*. Cada vez que se hace referencia a éste, se utiliza el contenido del registro *FSR* para dirigir al operando.

Los direccionamientos inmediato, directo, de bit e indexado han sido ampliamente utilizados hasta ahora. El direccionamiento indirecto es un recurso de los microcontroladores analizado a continuación para el caso del PIC16F84.

Para el direccionamiento indirecto se utilizan los registros *INDF* y *FSR* ubicados dentro de la zona de registros especiales, concretamente en las posiciones 00h y 04h del Banco 0, aunque el registro *INDF* no está implementado físicamente (figura 4-1).

El registro *FSR* sirve como **puntero** para el direccionamiento indirecto. Si en cualquier instrucción se opera con el registro *INDF*, en realidad se estará operando con la dirección donde apunte el contenido del registro *FSR*. Por ejemplo, si el *FSR* contiene el valor 14h una instrucción que opere sobre *INDF*, opera en realidad sobre la dirección 14h. Se puede decir en este ejemplo que la posición 14h de memoria fue direccionada en forma indirecta a través del puntero *FSR*. Si *FSR* = 14h, la instrucción *movwf INDF* carga el contenido del registro W en la posición 14h de la RAM, tal como se muestra en la figura 16-3(A). Simbólicamente se representa:

$$(W) \rightarrow ((FSR)), \text{ en este ejemplo, } (W) \rightarrow (14h)$$

Siguiendo con otro ejemplo, si *FSR* = 14h, la instrucción *movf INDF,W* carga el contenido de la posición 14h de la RAM en el registro de trabajo W, tal como se muestra en la figura 16-3(B). Simbólicamente se representa:

$$((FSR)) \rightarrow (W), \text{ en este ejemplo, } (14h) \rightarrow (W)$$

Esta simbología quiere representar que el contenido de la posición de memoria especificada por el registro FSR pasa al registro de trabajo (direcciónamiento indirecto). Con los dos paréntesis se indica que el contenido del registro no es un dato sino una dirección.

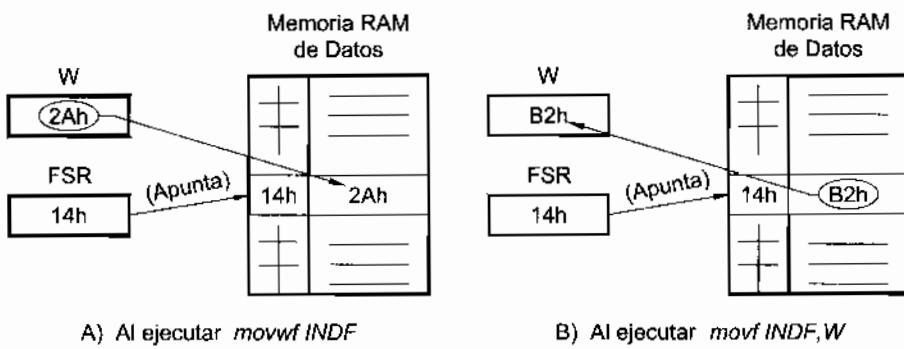


Figura 16-3 Funcionamiento de las instrucciones de Direccionamiento Indirecto

El direccionamiento indirecto es muy útil para el procesamiento de posiciones consecutivas de memoria RAM de datos, como se muestra en el siguiente programa ejemplo, para cuya comprobación habrá que utilizar el simulador del MPLAB.

```
***** Indirecto_01.asm *****
;
; Este programa comprueba el funcionamiento del direccionamiento indirecto. Se trata de
; escribir con el valor de una constante a partir de la última dirección ocupada de
; la memoria RAM de datos hasta el final.
; Su correcto funcionamiento debe comprobarse con el simulador del MPLAB.
;
; ZONA DE DATOS *****
;
; CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
RAM_Contador
ENDC

Constante EQU 2Ah
RAM_UltimaDireccion EQU 4Fh
; Por ejemplo.
; Última dirección de RAM de datos utilizada para
; el PIC16F84A.

; ZONA DE CÓDIGOS *****
;
ORG 0
Inicio
    movlw RAM_UltimaDireccion-RAM_Contador ; Número de posiciones a escribir.
    movwf RAM_Contador
```

- |           |                   |
|-----------|-------------------|
| movlw     | RAM_EscribeConsta |
| movwf     | movlw             |
| incf      | movwf             |
| decfsz    | incf              |
| goto      | decfsz            |
| Principal | goto              |
| sleep     | Principal         |
| END       | sleep             |
- La posición 00h.
  - A partir de la última posición 2AEh.
  - Las posiciones corresponden al contenido de memoria RAM de datos.

MPLAB IDE v8.30	
0000	2A
0010	2A
0020	2A
0030	2A
0040	2A
0050	2A
0060	2A
0070	2A
0080	2A
0090	2A
00A0	2A
00B0	2A
00C0	2A
00D0	2A
00E0	2A
00F0	2A

Figura 16-4 Memoria RAM de datos

## 16.4 MACROS

Las **macros** simplifican la elaboración de programas.

```

        movlw  RAM_Contador+1      ; Primera posición de RAM libre.
        movwf  FSR                  ; Primera dirección de memoria RAM a escribir
RAM_EscribeConstante
        movlw  Constante           ; Escribe el valor de la constante en la posición
        movwf  INDF                ; apuntada por FSR. (W) -> ((FSR))
        incf   FSR,F               ; Apunta a la siguiente dirección de memoria.
        decfsz RAM_Contador,F
        goto   RAM_EscribeConstante

Principal
        sleep                         ; Pasa a reposo.

        END

```

Al ejecutar este programa y comprobar su funcionamiento con el simulador se obtiene el resultado que muestra la figura 16-4 donde se aprecia el contenido de la memoria RAM de usuario en la ventana *File Registers* del simulador:

- La posición 0Ch la variable RAM\_Contador que al finalizar el programa vale 00h.
- A partir de la primera posición libre que es la 0Dh hasta la posición 4Fh, que es la última posición de RAM en el Banco 0, se llena con el valor de la constante 2AEh.
- Las posiciones desde 8Ch hasta CFh del Banco 1 están mapeadas con las correspondiente del Banco 0 (ver figura 4-1) y por tanto almacenan el mismo contenido.

MPLAB IDE v6.40 [File Registers]																
	REG															
0000	--	00	0A	10	50	00	00	--	00	00	00	00	2A	2A	2A	----
0010	2A	*****														
0020	2A	*****														
0030	2A	*****														
0040	2A	*****														
0050	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0060	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0070	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
0080	--	FF	0A	10	50	1F	FF	--	00	00	00	00	2A	2A	2A	----
0090	2A	*****														
00A0	2A	*****														
00B0	2A	*****														
00C0	2A	*****														
00D0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
00E0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--
00F0	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Figura 16-4 Memoria RAM de datos después de ejecutar el programa *Indirecto\_01.asm*

## 16.4 MACROS

Las **macros** son una potente herramienta del ensamblador que facilita la elaboración de programas. Una macro consiste en una serie de instrucciones y directivas

que se agrupan en una sola orden mayor de forma que se comporta como una única instrucción cuando es invocada. Suele utilizarse para automatizar la utilización de grupos de instrucciones usadas con frecuencia. Las macros pueden aceptar argumentos, lo que las hacen muy flexibles.

Antes de que una macro pueda ser invocada en una línea del programa fuente como si se tratase de una instrucción, debe ser definida por el diseñador de acuerdo a la siguiente sintaxis:

```
<label> MACRO [<arg>, <arg> ..., <arg>]
;
;
;
ENDM
```

Donde *<label>* es una etiqueta que define a la macro y *<arg>* es cualquier número de argumentos opcionales que se le proporciona a la macro. Los valores asignados a estos argumentos cuando se invoca la macro serán posteriormente sustituidos por los nombres de los argumentos que se encuentran dentro de la macro.

El cuerpo de la macro puede contener instrucciones o directivas. El ensamblador ejecuta el procesado de la macro hasta que encuentra alguna directiva de fin de la macro, generalmente *ENDM*.

Una vez la macro ha sido definida puede ser invocada en cualquier punto del programa utilizando una llamada a la macro con el siguiente formato:

```
<macro_name> [<arg>, ..., <arg>]
```

Donde *<macro\_name>* es el nombre de la macro previamente definida. Hay que proporcionarle los argumentos *<arg>* solicitados en la definición de la macro.

La macro no ocupa código máquina alguno mientras no sea invocada, cuando esto ocurre la macro se expandirá en el lugar del programa en que se encuentre.

Un ejemplo de definición de macro podría ser:

```
; Macro "SaltaSiIgual"
;
;
;
; Compara el valor del (Registro) con una "Constante". Si:
; - (Registro) = Constante, salta a la dirección indicada por el argumento "Salto" y el
;   flag Z se pone a "1".
; - (Registro) ≠ Constante, no salta y el flag Z se pone a "0".
```

```
SaltaSiIgual MACRO Registro, Constante, Salto
    movlw Constante           ; Va a realizar la comparación mediante resta.
    subwf Registro,W          ; (W)=(Registro)-Constante
```

btfsc  
goto  
ENDM

Un ejemp

SaltaSiIgua

Lo que pro

moviw 0
subwf C
btfsc S
goto V

El diseñado  
programas más uti  
librería con macros

```
*****
;
; Éstas son algunas mac
;
; Macro "Incrementa" -
;
; Incrementa el valor del
; - Que (Registro)
;   por el argume
; - Que (Registro)
;   del argumento
```

Incrementa	MA
incf	Reg
movf	Reg
sublw	Max
btfsc	STA
goto	Salt
movlw	Min
movwf	Reg
ENDM	

;	Macro "SaltaSiIgual" -
;	Compara el valor del (Re
;	- (Registro)=Con
;	flag Z se pone a
;	- (Registro)distin

SaltaSiIgual	MAC
movlw Cons	Reg
subwf Reg	

btfsc	STATUS,Z	; ¿Z=0?, ¿(W) distinto 0?, ¿(Registro) distinto Constante?
goto	Salto	; Ha resultado (Registro)=Constante y salta.
ENDM		

Un ejemplo de invocación de esta macro puede ser:

```
SaltaSiIgual Contador,0xF9,Visualiza
```

Lo que produciría el siguiente código en el proceso de ensamblado.

movlw	0xF9	; Va a realizar la comparación mediante resta.
subwf	Contador,W	; (W)=(Registro)-Constante
btfsc	STATUS,Z	; ¿Z=0?, ¿(W) distinto 0?, ¿(Registro) distinto Constante?
goto	Visualiza	; Ha resultado (Registro)=Constante y salta.

El diseñador puede crear su propia librería de macros con los fragmentos de programas más utilizados y tratarla como instrucciones personalizadas. Un ejemplo de útil librería con macros es el siguiente:

```
;***** Librería "MACROS.INC" *****
; Éstas son algunas macros útiles y de frecuente uso.
; Macro "Incrementa"
; Incrementa el valor del (Registro). Pueden ocurrir dos casos:
; - Que (Registro) no supere el valor "Maximo", entonces salta a la etiqueta apuntada
; por el argumento "Salto". El flag Carry se pone a "1".
; - Que (Registro) supere el valor "Maximo", entonces se inicializa con el valor de
; del argumento "Minimo". El flag Carry se pone a "0".
Incrementa MACRO Registro, Minimo, Maximo, Salto
    incf Registro,F ; Incrementa el valor del registro.
    movf Registro,W ; ¿Ha llegado a su valor máximo?
    sublw Maximo ; (W)=Maximo-(Registro).
    btfsc STATUS,C ; ¿C=0?, ¿(W) negativo?, ¿Maximo<(Registro)?
    goto Salto ; No, Es Maximo>=(Registro) y salta.
    movlw Minimo ; Sí, ha resultado Maximo<(Registro), entonces
    movwf Registro ; inicializa el registro.
ENDM

; Macro "SaltaSiIgual"
; Compara el valor del (Registro) con una "Constante". Si:
; - (Registro)=Constante, salta a la dirección indicada por el argumento "Salto" y el
; flag Z se pone a "1".
; - (Registro)distinto "Constante", no salta y el flag Z se pone a "0".
SaltaSiIgual MACRO Registro, Constante, Salto
    movlw Constante ; Va a realizar la comparación mediante resta.
    subwf Registro,W ; (W)=(Registro)-Constante.
```

```

btfc STATUS,Z      ; ¿Z=0?, ¿(W) distinto 0?, ¿(Registro) distinto Constante?
goto Salto         ; Ha resultado (Registro)=Constante y salta.
ENDM

```

; Macro "SaltaSiMayor"

; Compara el valor del (Registro) con una "Constante". Si:  
; - (Registro)>Constante, salta a la dirección indicada por el argumento "Salto" y el  
; flag Carry se pone a "0".  
; - (Registro)<=Constante, no salta y el flag Carry se pone a "1".

SaltaSiMayor MACRO Registro, Constante, Salto

```

movf Registro,W    ; Va a realizar la comparación mediante resta.
sublw Constante     ; (W)=Constante-(Registro).
btfs STATUS,C       ; ¿C=1?, ¿(W) positivo?, ¿Constante>=(Registro)?
goto Salto          ; Ha resultado Constante<=(Registro) y salta.
ENDM

```

; Macro "SaltaSiMenor"

; Compara el valor del (Registro) con una "Constante". Si:  
; - (Registro)<Constante, salta a la dirección indicada por el argumento "Salto" y el  
; flag Carry se pone a "0".  
; - (Registro)>=Constante, no salta y el flag Carry se pone a "1".

SaltaSiMenor MACRO Registro, Constante, Salto

```

movlw Constante     ; Va a realizar la comparación mediante resta.
subwf Registro,W    ; (W)=(Registro)-Constante
btfs STATUS,C       ; ¿C=1?, ¿(W) positivo?, ¿(Registro)>=Constante?
goto Salto          ; Ha resultado (Registro)<Constante y salta.
ENDM

```

; Macro "VisualizaBCD"

; Visualiza en la pantalla del módulo LCD el contenido de un (Registro) expresado en BCD.

VisualizaBCD MACRO Registro

```

movf Registro,W
call BIN_a_BCD      ; Se debe visualizar en BCD.
call LCD_Byt         ; Con decena apagada si es cero.
ENDM

```

Para comprender las ventajas en la utilización de las macros se expone un programa resuelto sin macros tal como se ha hecho hasta ahora y posteriormente con las macros definidas en la librería MACROS.INC.

\*\*\*\*\* Macro\_01.asm \*\*\*\*\*

; Se conectará un pulsador al pin RA4 y mientras se mantenga pulsado se incrementarán dos contadores distintos que se visualizarán en la pantalla del modulo LCD:  
; - El Contador1 se visualiza en la línea 1 y cuenta de 3 a 16.  
; - El Contador2 se visualiza en la línea 2 y cuenta de 7 a 21.

; ZONA DE DATOS

CONFIG  
LIST  
INCLUDE

CBLOCK0  
Contador1  
Contador2  
ENDC

Minimo1 EQU .3  
Maximo1 EQU .1  
Minimo2 EQU .7  
Maximo2 EQU .2

#DEFINE Pulsador P

; ZONA DE CÓDigos

	ORG	0
Inicio	call	LC
	bsf	ST
	bsf	Pu
	bcf	ST
	movlw	Mi
	movwf	Co
	movlw	Mi
	movwf	Co
	call	Vis

	Principal	
	btfs	Pub
	call	Inc
	goto	Prin

; Subrutina "Incrementa"

; Subrutina que incrementa

IncrementaVisualiza

	IncrementaVisualiza	
	incf	Com
	movf	Com
	sublw	Max
	btfc	STA
	goto	Inc
	movlw	Min
	movwf	Com

IncrementaContador2

	IncrementaContador2	
	incf	Com
	movf	Com
	sublw	Max
	btfc	STA

into Constante?  
 sta.  
 egistro)?  
 ta.

esta.  
 onstante?  
 ta.

de un programa  
 con las macros

```
; ZONA DE DATOS ****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK0x0C
Contador1
Contador2
ENDC

Minimo1 EQU    .3
Maximo1 EQU    .16
Minimo2 EQU    .7
Maximo2 EQU    .21

#DEFINE Pulsador PORTA,4           ; Línea donde se conecta el pulsador.

; ZONA DE CÓDIGOS ****
ORG     0
Inicio
  call   LCD_Inicializa
  bsf    STATUS,RP0          ; Acceso al Banco 1.
  bsf    Pulsador            ; Línea del pulsador configurada como entrada.
  bcf    STATUS,RP0          ; Acceso al Banco 0.
  movlw  Minimo1             ; Inicializa los contadores
  movwf  Contador1
  movlw  Minimo2
  movwf  Contador2
  call   VisualizaContadores

Principal
  btfs  Pulsador            ; Lee el pulsador.
  call   IncrementaVisualiza ; Salta a incrementar y visualizar el contador.
  goto  Principal

; Subrutina "IncrementaVisualiza"
;
; Subrutina que incrementa el contador y lo visualiza.
;
IncrementaVisualiza
  incf  Contador1,F          ; Incrementa el contador 1.
  movf  Contador1,W          ; ¿Ha llegado a su valor máximo?
                            ; (W) = Maximo1 - (Contador1).
  sublw Maximo1
  btfs  STATUS,C              ; ¿C=0?, ¿(W) negativo?, ¿Maximo1<(Contador1)?
  goto  IncrementaContador2  ; No, ha resultado Maximo1>=(Contador1)
  movlw  Minimo1
  movwf  Contador1            ; Si, ha resultado Maximo1<(Contador1), entonces
                            ; inicializa el registro.

IncrementaContador2
  incf  Contador2,F          ; Incrementa el contador 2.
  movf  Contador2,W          ; ¿Ha llegado a su valor máximo?
                            ; (W) = Maximo2 - (Contador2).
  sublw Maximo2
  btfs  STATUS,C              ; ¿C=0?, ¿(W) negativo?, ¿Maximo2<(Contador2)?
```

```

goto    VisualizaContadores; No, ha resultado Maximo2>=(Contador2)
movlw   Minimo2           ; Si, ha resultado Maximo2<(Contador2), entonces
movwf   Contador2         ; inicializa el registro.

VisualizaContadores
call    LCD_Lineal
movf    Contador1,W
call    BIN_a_BCD
call    LCD_Byte
call    LCD_Linea2
movf    Contador2,W
call    BIN_a_BCD
call    LCD_Byte
call    Retardo_200ms     ; Se incrementa cada 200 ms.
return

;
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END

```

A continuación se expone el mismo programa pero resueltos con macros.

```

***** Macro_02.asm *****
;
; Es el mismo ejercicio que Macro_01.asm, pero resueltos con macros.
;
; ZONA DE DATOS *****
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST    P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK0x0C
Contador1
Contador2
ENDC

Minimo1 EQU .3
Maximo1 EQU .16
Minimo2 EQU .7
Maximo2 EQU .21

#define Pulsador PORTA,4          ; Línea donde se conecta el pulsador.

;
; ZONA DE CÓDIGOS *****
INCLUDE <MACROS.INC>          ; La definición de las macros se deben realizar
                                ; antes de que éstas sean invocadas.
ORG    0
Inicio
call   LCD_Inicializa
bsf    STATUS,RP0               ; Acceso al Banco 1.

```

bsf	Puls
bcf	STA
movlw	Min
movwf	Con
movlw	Min
movwf	Con
call	Visu
<b>Principal</b>	
btfss	Puls
call	Incre
goto	Princ
;	
; Subrutina "IncrementaV"	
; Subrutina que incrementa	
IncrementaVisualiza	
Incrementa	
IncrementaContador2	
Incrementa	
<b>VisualizaContadores</b>	
call	LCD
VisualizaBCD	
call	LCD
VisualizaBCD	
call	Retar
return	;
INCLUDE <RE	
INCLUDE <BI	
INCLUDE <LO	
END	

La subrutina IncrementaVisualiza es la que se encarga de visualizar los datos en la pantalla LCD. Al acercándose a la definición de las macros se observa que las macros se definen de forma similar a como se hace en C.

Seguro que al final del libro se habrá visto que es posible añadir a la librería que viene por defecto nuevas formas de resolución de problemas. Hacerlo y, además, a modo de ejemplo, es una utilidad de este recurso que es el libro.

## 16.5 RESISTORES

Una forma usual de conectar un resistor es la correspondiente resistencia.

```

bsf      Pulsador          ; Línea del pulsador configurada como entrada.
bcf      STATUS,RP0        ; Acceso al Banco 0.
movlw   Minimo1           ; Inicializa los contadores
movwf   Contador1         ;
movlw   Minimo2           ;
movwf   Contador2         ;
call    VisualizaContadores
Principal
btfsr   Pulsador          ; Lee el pulsador.
call    IncrementaVisualiza ; Salta a incrementar y visualizar el contador.
goto   Principal

;
; Subrutina "IncrementaVisualiza"
;
; Subrutina que incrementa el contador y lo visualiza.
;
IncrementaVisualiza
  Incrementa      Contador1,Minimo1,Maximo1,IncrementaContador2
  IncrementaContador2
    Incrementa    Contador2,Minimo2,Maximo2,VisualizaContadores
  VisualizaContadores
    call    LCD_Linea1
    VisualizaBCD  Contador1
    call    LCD_Linea2
    VisualizaBCD  Contador2
    call    Retardo_200ms
  return

;
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END

```

La subrutina IncrementaVisualiza se ha simplificado mucho ganando en claridad y acercándose a la filosofía del lenguaje estructurado. Es importante señalar que la definición de las macros mediante el *INCLUDE <MACROS.INC>* debe realizarse antes de que las macros sean invocadas, por ello se coloca al principio del programa.

Seguro que al finalizar este ejercicio al lector se le han ocurrido nuevas macros para añadir a la librería que se adapten a su forma de trabajar y le han surgido ideas sobre nuevas formas de resolver los múltiples ejercicios propuestos hasta ahora. Le animamos a hacerlo y, además, a reformar todos los ejercicios que se planteen en adelante mediante la utilización de este recurso. Y no olvide actualizar su librería de macros conforme avanza el libro.

## 16.5 RESISTENCIAS DE PULL-UP DEL PUERTO B

Una forma usual de producir pulsos es mediante pulsadores que necesitan de su correspondiente **resistencia de Pull-Up** (figura 15-3). El PIC16F84 permite configurar

una resistencia de Pull-Up interna en cada una de las líneas del Puerto B, ahorrando estas necesarias resistencias externas cuando se utilizan pulsadores u otros dispositivos externos de lectura (figura 16-5).

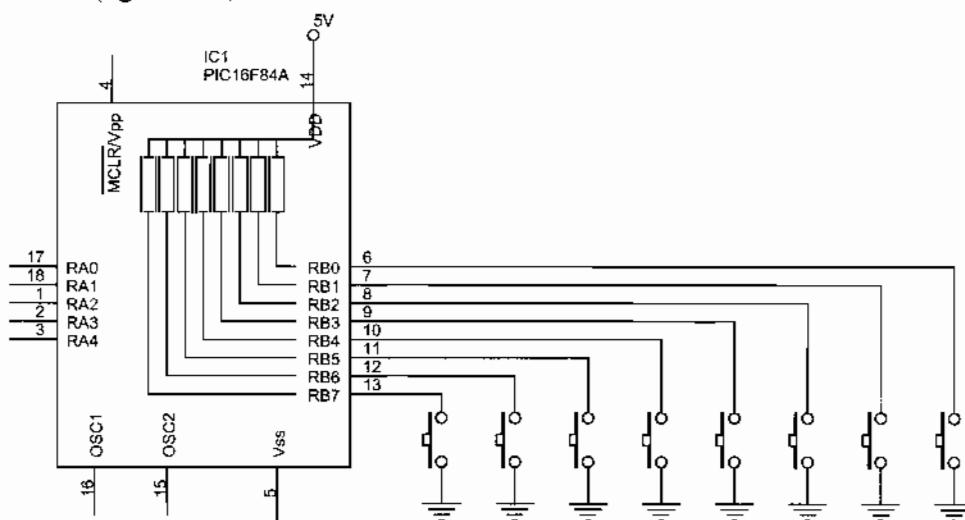


Figura 16-5 Resistencias de Pull-Up internas del PIC16F84A

Para configurar esta resistencia hay que utilizar el bit NOT\_RBPU del registro OPTION:

/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 16-2 Bit para configurar las resistencias de Pull-Up del Puerto B

- **/RBPU, (Resistor Port B Pull-Up Enable bit).** Habilitación de las resistencias de Pull-Up del Puerto B.
  - /RBPU = 0. Habilita las resistencias de Pull-Up del Puerto B.
  - /RBPU = 1. Deshabilita las resistencias de Pull-Up del Puerto B.

Por tanto para activar las resistencias de Pull-Up del Puerto B basta con resetear el bit NOT\_RBPU del registro OPTION utilizando instrucción:

*bcf OPTION\_REG,NOT\_RBPU*

## 16.6 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular y grabar el microcontrolador con los siguientes programas. Comprobar su

correcto funcionamiento todas las mejoras que con-

## WATCHDOG Y SLE

**Watchdog\_01.asm:** mismo se producirá cada producirá un incremento volverá a la situación de aproximadamente.

**Watchdog\_02.asm:** incrementará un contador en pulsar el Watchdog se e

## DIRECCIONAMIENTO

**Indirecto\_01.asm:** direccionamiento indirecto última dirección ocupada funcionamiento debe comp

**Indirecto\_02.asm:** de la memoria RAM de da por ejemplo, en la dirección dirección 22h se escribe incluida, ya que a partir de

A continuación se p desde la dirección 00h ha dirección y su contenido, SFR. Cada visualización se

## MACROS

**Macro\_01.asm:** Se pulsado se incrementarán d

- El Contador 1
- El Contador 2

**Macro\_02.asm:** Re

ahorando estas  
sitivos externos

correcto funcionamiento con el esquema de la figura 13-10. El lector puede introducir todas las mejoras que considere conveniente.

## WATCHDOG Y SLEEP

**Watchdog\_01.asm:** El PIC se pone en modo bajo consumo. El despertar del mismo se producirá cada vez que el Watchdog desborde su cuenta, en ese momento se producirá un incremento de un contador que se visualizará en la pantalla y nuevamente volverá a la situación de bajo consumo. El proceso debe repetirse cada medio segundo aproximadamente.

**Watchdog\_02.asm:** Cada vez que se presione el pulsador conectado al pin RA4, se incrementará un contador que se visualizará en el display LCD. Si tarda más de 1 segundo en pulsar el Watchdog se despertará y en la pantalla aparecerá un mensaje.

## DIRECCIONAMIENTO INDIRECTO

**Indirecto\_01.asm:** Este programa comprobará el funcionamiento del direccionamiento indirecto. Se trata de escribir con el valor de una constante a partir de la última dirección ocupada de la memoria RAM de datos hasta el final. Su correcto funcionamiento debe comprobarse con el simulador.

**Indirecto\_02.asm:** Se trata de escribir, a partir de la de la última dirección ocupada de la memoria RAM de datos hasta el final, el contenido con el valor de la dirección. Así, por ejemplo, en la dirección 20h se escribe "20", en la dirección 21h se escribe "21", en la dirección 22h se escribe "22" y así sucesivamente. Se escribirá hasta la dirección 4Fh incluida, ya que a partir de la dirección 50h no están implementadas en un PIC16F84.

A continuación se procederá a la lectura de la memoria RAM de datos completa, desde la dirección 00h hasta la 4Fh. En la pantalla del display LCD se visualizará la dirección y su contenido. Observar que las direcciones de 00h a 0Bh corresponden al SFR. Cada visualización se mantendrá durante medio segundo en pantalla.

## MACROS

**Macro\_01.asm:** Se conectarán un pulsador al pin RA4 y mientras se mantenga pulsado se incrementarán dos contadores distintos que se visualizarán en la pantalla:

- El Contador 1 se visualizará en la línea 1 y cuenta de 3 a 16.
- El Contador 2 se visualizará en la línea 2 y cuenta de 7 a 21.

**Macro\_02.asm:** Repetir el ejercicio anterior mediante la utilización de macros.

**Macro\_03.asm:** Programa para el Juego de la Bonoloto. Al presionar sobre el pulsador conectado al pin RA4, se incrementará un contador rápidamente de 1 a 49. Cuando se suelta el pulsador aparece el número seleccionado. No es necesario eliminar los rebotes del pulsador que incluso facilitarán la obtención de números pseudoaleatorios.

**Otros:** Aumentar el número de macros de la librería MACROS.INC hasta que se adapte a los gustos del lector. Comprobar después el correcto funcionamiento de las mismas reformando algunos de los programas hechos hasta ahora.

En casi todo  
como pulsadores, in  
modo de realizarlo c

## 17.1 TÉCNIC

Hasta ahora l  
como por ejemplo  
programa del estado  
“por sondeo”. El sig  
ejemplo de su utilizac

```
;*****
; Cada vez que presione e
; que es visualizára en el m
; leyendo constantemente
; ZONA DE DATOS ***
```

```
CONFIG
LIST P=_
INCLUDE <P>
```

```
CBLOCK 0x00
Contador
ENDC
```

ionar sobre el  
ante de 1 a 49.  
rio eliminar los  
aleatorios.

C hasta que se  
amiento de las

Domingo

## CAPÍTULO 17

# INTERRUPCIONES. LECTURA DE ENTRADAS

En casi todos los proyectos es necesario leer alguna entrada de tipo digital tal como pulsadores, interruptores, sensores digitales, o similares. Este capítulo explica el modo de realizarlo con la máxima eficacia.

### 17.1 TÉCNICA POLLING

Hasta ahora la técnica utilizada para la lectura de entradas al microcontrolador, como por ejemplo pulsadores, ha consistido en la comprobación cíclica por parte del programa del estado de la entrada correspondiente, a esta técnica se la llama *Polling* o "por sondeo". El siguiente programa de control para el circuito de la figura 17-1 es un ejemplo de su utilización.

```
***** Int_INT_01.asm *****

; Cada vez que presione el pulsador conectado al pin RB0, se incrementará un contador
; que es visualizará en el módulo LCD. La lectura se realiza mediante la técnica Polling
; leyendo constantemente el estado de la entrada.

; ZONA DE DATOS *****

    CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    LIST      P=16F84A
    INCLUDE <P16F84A.INC>

    CBLOCK 0x0C
    Contador          ; El contador a visualizar.
    ENDC
```

```

#define Pulsador PORTB,0 ; Línea donde se conecta el pulsador.

; ZONA DE CÓDIGOS ****
ORG 0
Inicio
    call LCD_Inicializa
    bsf STATUS,RP0 ; Acceso al Banco 1.
    bsf Pulsador ; La línea RB0/INT se configura como entrada.
    bcf OPTION_REG,NOT_RBPU ; Activa las resistencias de Pull-Up del Puerto B.
    bcf STATUS,RP0 ; Acceso al Banco 0.
    clrf Contador ; Inicializa el contador y lo visualiza.
    call VisualizaContador

; La sección "Principal" es de mantenimiento. Está leyendo constantemente; la línea de entrada
; mediante técnica Polling.

Principal
    btfss Pulsador ; Lee el pulsador.
    call IncrementaVisualiza ; Si pulsa, salta a incrementar y visualizar el contador.
    goto Principal

; Subrutina "IncrementaVisualiza"
; Incrementa un contador y lo visualiza.
; IncrementaVisualiza
    call Retardo_20ms ; Espera a que se estabilice el nivel de tensión.
    btfsc Pulsador ; Confirma si es un rebote.
    goto Fin_Incrementa
    incf Contador,F ; Incrementa el contador y después lo visualiza.

VisualizaContador
    call LCD_Lineal
    movf Contador,W
    call BIN_a_BCD ; Se debe visualizar en BCD.
    call LCD_Byt

EsperaDejePulsar
    btfss Pulsador
    goto EsperaDejePulsar

Fin_Incrementa
    return

INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END

```

La figura 17-2(A) muestra el diagrama básico de funcionamiento de esta técnica. Aunque éste es el método más sencillo para leer una entrada presenta el inconveniente de una pérdida de eficacia, ya que el programa tiene que quedar encerrado en un bucle permanente que permita leer la entrada. Las desventajas de la técnica *Polling* son:

- Hay que invertir la señal de salida.
- Al periférico se le interviene.

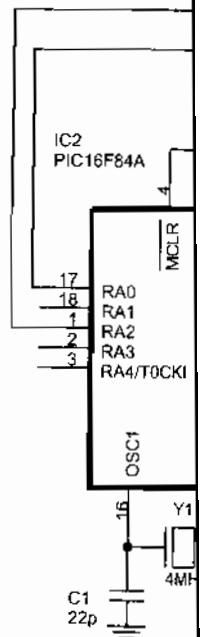


Figura 17-1 Una

## 17.2 INTERRUPCIÓN

Una **interrupción externa** puede interrumpir la ejecución de un programa para que de entonces se produzca una **interrupción**, ésta a su vez ejecutará un fragmento de código que se conoce como **tarea** o **rutina de servicio**. La ejecución de la tarea justo donde la interrupción se produce es una característica de la interfaz de programación de PIC.

Este método es más eficiente que el de la técnica *Polling*, ya que el sistema perderá el tiempo para leer la entrada únicamente cuando se detecta una interrupción.

- Hay que interrogar a las entradas en cada ciclo de programa.
- Al periférico se le atiende después de realizar la consulta y no cuando solicita la intervención del microcontrolador.

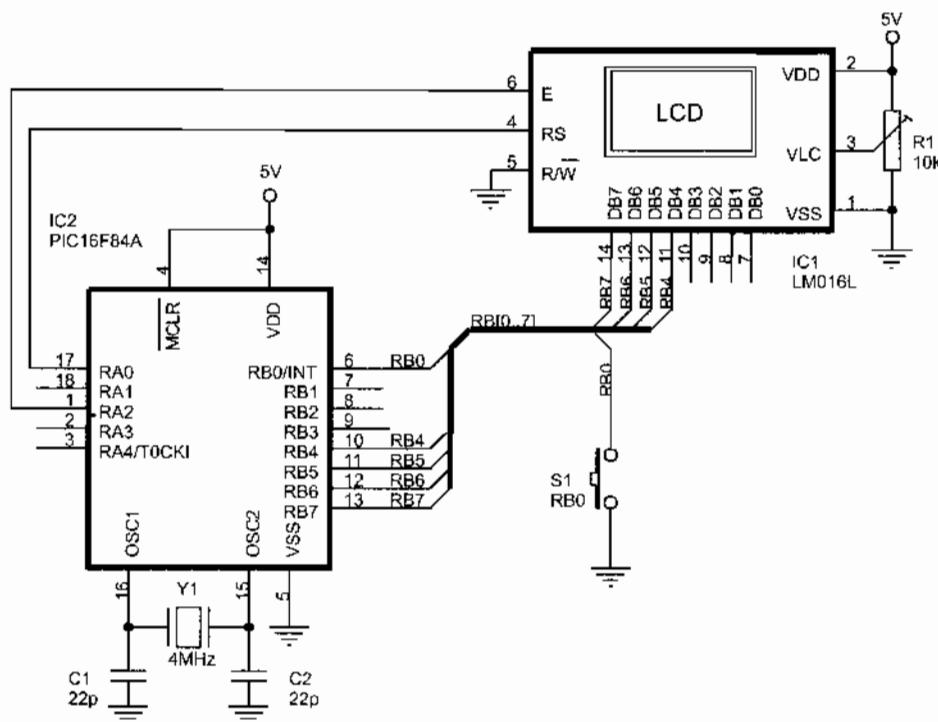


Figura 17-1 Una entrada se puede leer mediante técnica Polling o Interrupciones

## 17.2 INTERRUPCIONES

Una **interrupción** consiste en un mecanismo por el cual un evento interno o externo puede interrumpir la ejecución de un programa en cualquier momento. A partir de entonces se produce automáticamente un salto a una **subrutina de atención a la interrupción**, ésta atiende inmediatamente el evento y retoma luego la ejecución del programa exactamente donde estaba en el momento de ser interrumpido, continuando su tarea justo donde la dejó, tal como ilustra la figura 17-2(B). La interrupción tiene la característica de la inmediatez, nace de la necesidad de ejecutar una subrutina en el instante preciso y, por tanto, se considera su intervención urgente.

Este método es más eficaz que la técnica *Polling* ya que el microcontrolador no perderá el tiempo preguntando a la línea de entrada para leer el estado, sino que únicamente atenderá al periférico cuando éste se lo pida mediante la solicitud de interrupción.

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

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

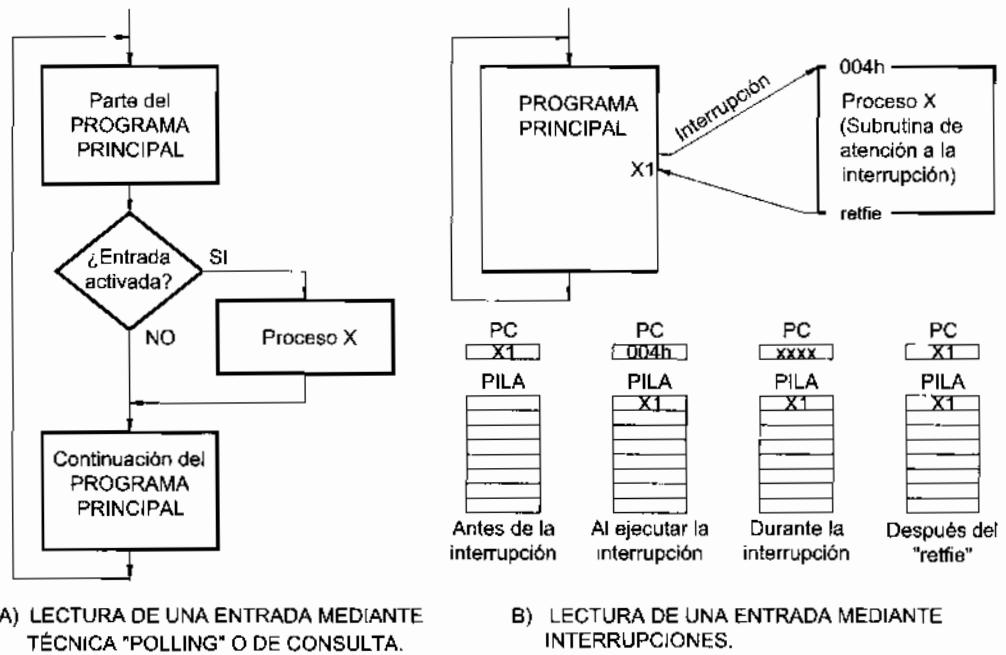


Figura 17-2 Funcionamiento de la técnica Polling frente a las Interrupciones

El PIC16F84 dispone de 4 posibles fuentes de interrupción:

- Interrupción **INT**. Por activación del pin RB0/INT.
- Interrupción **RBI**. Por cambio de estado en una o varias de las 4 líneas de más peso RB7:RB4 del Puerto B.
- Interrupción **T0I**. Por desbordamiento del Timer 0.
- Interrupción **EEI**. Por la finalización de la escritura en la EEPROM de datos.

El programa deberá tener mecanismos que permitan identificar la causa de la interrupción.

### 17.3 FUN

Cuando se recibe una petición de interrupción:

- 1º Salva el contenido de la pila, incluyendo el PC.
- 2º El bit GIE se pone a 1, lo que provoca que el PC sea modificado.
- 3º El PC se dirige a la subrutina de atención a la interrupción.
- 4º Comienza la ejecución de la subrutina.

Los bits de control de las interrupciones. Consultar figura 17-3.

- Uno de estos bits debe estar a 1 para que no la interrupción sea ignorada.
- El otro bit es el INTEN, que habilita las interrupciones.

TOIF	[Caja lógica]
TOIE	[Caja lógica]
INTF	[Caja lógica]
INTE	[Caja lógica]
RBIF	[Caja lógica]
RBIE	[Caja lógica]
EEIF	[Caja lógica]
EEIE	[Caja lógica]
GIE	[Caja lógica]

Hay un único bit de control para cada tipo de interrupción, el cual, cuando se pone a 1, indica que la continuación deberá darse en la subrutina de atención a la interrupción que produce la interrupción.

En el PIC16F84, el bit INTEN habilita las interrupciones. Desde el momento en que se pone a 1, el efecto que produce es que se activa la interrupción.

### 17.3 FUNCIONAMIENTO DE UNA INTERRUPCIÓN

Cuando se produce cualquiera de los sucesos indicados anteriormente se origina una petición de interrupción que, si se acepta, origina el siguiente mecanismo hardware:

- 1º Salva el valor actual del contador de programa (PC) guardando su contenido en la pila, figura 17-2(B).
- 2º El bit GIE (*Global Interrupt Enable*) del registro INTCON es puesto a cero, lo que prohíbe cualquier otra interrupción.
- 3º El PC se carga con el valor 004h, que es la posición del vector de interrupción.
- 4º Comienza a ejecutarse el programa de atención a la interrupción que se encuentra a partir de la dirección 004h.

Los bits de control localizados en el registro INTCON habilitan y configuran las interrupciones. Cada causa de interrupción actúa con dos flags representados en la figura 17-3.

- Uno de ellos actúa como flag de señalización que indica si se ha producido o no la interrupción: T0IF, INTF, RBIF y EEIF.
- El otro funciona como permiso o prohibición de la interrupción en sí: TOIE, INTE, RBIE, EEIE y GIE.

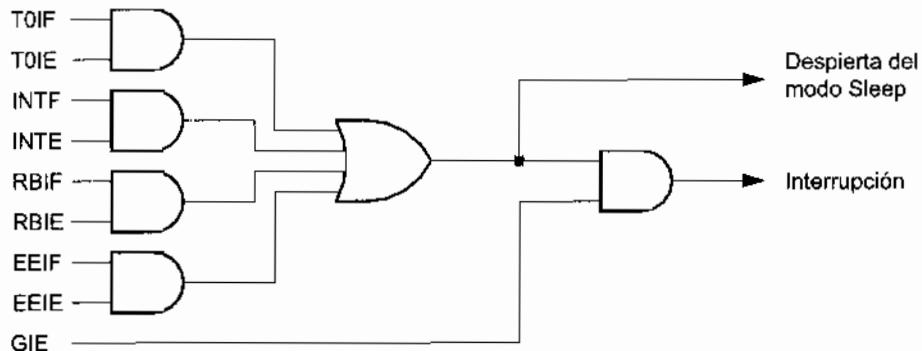


Figura 17-3 Interrupciones en el PIC16F84

Hay un único **vector de interrupción** en la dirección 004h. Sea cual sea la interrupción, el contador de programa se carga con la dirección 004h, figura 17-2(B). A continuación debe comprobar los diferentes indicadores para saber cuál es el dispositivo que produce la interrupción y actuar según sea el caso.

En el PIC16F84 las interrupciones se comportan casi exactamente igual que las subrutinas. Desde el punto de vista del control del programa, una interrupción produce el mismo efecto que ocurriría si el programa tuviese un “call 004h” en el punto en que se produjo la interrupción, ejecutando la **subrutina de atención de interrupción**.

## 17.4 FLAGS RELACIONADOS CON INTERRUPCIONES

Los flags relacionados con las interrupciones se alojan en los registros INTCON y OPTION.

### 17.4.1 Del registro INTCON

El registro INTCON (*Interrupts Control Register*) es el registro para el control de las interrupciones, está localizado en la dirección 0Bh del Banco 0 ó 8Bh del Banco 1. Es el encargado del manejo de las interrupciones y contiene los 8 bits que se muestran en la tabla 17-1, de los cuales unos actúan como flags señaladores del estado de la interrupción y otros como bit de permiso o autorización para que se pueda producir la interrupción:

GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 17-1 Bits del registro INTCON

- **GIE** (*Global Interrupt Enable*). Flag de habilitación global del permiso de interrupción. Se borra automáticamente cuando se reconoce una interrupción para evitar que ninguna otra se produzca mientras se está atendiendo a la primera. Al retornar de la interrupción con una instrucción *retfie*, el bit GIE se vuelve a activar poniéndose a “1”. Para el resto de los flags no se ha previsto mecanismo de puesta a cero, por lo que el programa de atención a la interrupción debe realizar el tratamiento de la correspondiente interrupción y además debe poner el o los flags de indicación de interrupción a “0”.
  - GIE = 0. No autoriza interrupción de ningún tipo.
  - GIE = 1. Autoriza cualquier tipo de interrupción. Se pone a “1” de forma automática con la instrucción *retfie*.
- **EEIE** (*EEPROM Write Complete Interrupt Enable*). Habilitación de la interrupción EEI. Flag que autoriza la interrupción por escritura completada de un byte en la EEPROM de datos del PIC. (El flag EEIF se encuentra en el registro EECON1).
  - EEIE = 0. Interrupción EEI deshabilitada.
  - EEIE = 1. Interrupción EEI habilitada.
- **T0IE** (*TMR0 Interrupt Enable bit*). Habilitación de la interrupción T0I. Flag que autoriza la interrupción por desbordamiento del Timer 0.
  - T0IE = 0. Interrupción T0I deshabilitada.
  - T0IE = 1. Interrupción T0I habilitada.
- **INTE** (*External INT Enable bit*). Habilitación de la interrupción externa INT. Flag que autoriza la interrupción externa a través del pin RB0/INT.
  - INTE = 0. Interrupción INT deshabilitada.
  - INTE = 1. Interrupción INT habilitada.

- **RBI**  
RBI  
RB7:  
◦ ◦
- **T0IF**  
produ  
desbo  
b'000  
◦ ◦
- **INTF**  
INT.  
◦ ◦
- **RBIF**  
Indica  
cualqu  
◦ R  
◦ R  
(I)

## 17.4.2 De...

La misi  
TMR0, como s  
interrupcione

/RBPU	IN
Bit 7	

Tabla

- **INTED**  
INT.  
◦ IN  
◦ IN

## 17.5 INST...

La instruc  
un retorno de

## PCIONES

gistros INTCON y

para el control de  
3h del Banco 1. Es  
se muestran en la  
de la interrupción  
a interrupción:

INTF	RBIF
Bit 1	Bit 0

al del permiso de  
una interrupción  
á atendiendo a la  
*retfie*, el bit GIE se  
no se ha previsto  
de atención a la  
ntre interrupción y  
a "0".

one a "1" de forma

abilitación de la  
ura completada de  
se encuentra en el

rupción T0I. Flag

oción externa INT.  
0/INT.

- **RBIE (RB Port Change Interrupt Enable).** Habilitación de la interrupción RBI. Flag que autoriza la interrupción por cambio de estado de las líneas RB7:RB4 del Puerto B.
  - RBIE = 0. Interrupción RBI deshabilitada.
  - RBIE = 1. Interrupción RBI habilitada.
- **T0IF (TMR0 Overflow Interrupt Flag bit).** Flag de estado de la interrupción producida por el TMR0. Indica que se ha producido una interrupción por desbordamiento del Timer 0, es decir, que ha pasado de b'11111111' (FFh) a b'00000000' (00h).
  - T0IF = 0. El TMR0 no se ha desbordado
  - T0IF = 1. El TMR0 se ha desbordado. (Debe borrarse por software).
- **INTF (External Interrupt Flag bit).** Flag de estado de la interrupción externa INT. Indica que se ha producido una interrupción a través del pin RB0/INT.
  - INTF = 0. No hay interrupción externa por el pin RB0/INT.
  - INTF = 1. Ha ocurrido una interrupción externa por la línea RB0/INT. (Debe borrarse por software).
- **RBIF (RB port change Interrupt Flag).** Flag de estado de la interrupción RBI. Indica que se ha producido una interrupción por cambio de estado de cualquiera de las líneas RB4 a RB7.
  - RBIF = 0. Ninguna de las entradas RB7 a RB4 ha cambiado de estado.
  - RBIF = 1. Cualquiera de las líneas RB7 a RB4 del Puerto B ha cambiado. (Debe borrarse por software).

### 17.4.2 Del registro OPTION

La misión principal del registro OPTION es gobernar el comportamiento del TMR0, como se estudió en el capítulo 15, pero también tiene un bit relacionado con las interrupciones externas:

/RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Tabla 17-2 Bit del registro OPTION utilizados por las interrupciones

- **INTEDG (Interrupt Edge Select bit).** Selector de flanco de la interrupción INT.
  - INTEDG = 0. Interrupción por flanco descendente del pin RB0/INT.
  - INTEDG = 1. Interrupción por flanco ascendente del pin RB0/INT.

## 17.5 INSTRUCCIÓN “RETFIE”

La instrucción *retfie* utilizada al final de la subrutina de interrupción es idéntica a un retorno de subrutina *return*. Además coloca automáticamente a "1" el bit GIE,

volviendo a habilitar las interrupciones. Al terminar la subrutina de servicio a la interrupción el programa principal continúa donde fue interrumpido.

Es importante recalcar que la subrutina de atención a la interrupción debe situarse a partir de la dirección 0x004 y acabar con la instrucción *retfie*.

## 17.6 INTERRUPCIÓN EXTERNA INT

La fuente de interrupciones externa INT es muy importante para atender eventos externos en tiempo real. Cuando en la línea RB0/INT se hace una petición de interrupción el bit INTF del registro INTCON se pone a "1" de forma automática y, si el bit GIE está a "1", se pone en marcha el mecanismo ya comentado de la interrupción.

Mediante el bit INTDEG del registro OPTION es seleccionado el flanco activo de RB0/INT, ya que con éste puesto a "1" el flanco activo es el ascendente y cuando está a "0" el flanco activo es el descendente.

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

A continuación se expone un programa ejemplo para el hardware de la figura 17-1, cuyo pulsador producirá una interrupción externa INT al actuar sobre él. Es un ejemplo típico de lectura de entradas digitales aplicable a multitud de proyectos.

```
***** Int_INT_02.asm *****
; Cada vez que presiona el pulsador conectado al pin RB0/INT se incrementa un contador
; que es visualizado en el módulo LCD. La lectura del pulsador se hará mediante interrupciones.

; ZONA DE DATOS *****

    CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    LIST      P=16F84A
    INCLUDE <P16F84A.INC>

    CBLOCK 0x0C
    Contador          ; El contador a visualizar.
    ENDC

#DEFINE Pulsador     PORTB,0      ; Línea donde se conecta el pulsador.

; ZONA DE CÓDIGOS *****

    ORG    0
    goto  Inicio
    ORG    4          ; Vector de interrupción.
    goto  ServicioInterrupcion
```

Inicio

call  
bsf  
bsf  
bcf  
bcf  
bcf  
bcf  
clrf  
call  
movlw  
movwf

; La sección "Pnn"

Principal

sleep  
goto

; Subrutina "Serv"

; Subrutina de ser

;

ServicioInterrupc

call  
btfs  
goto  
incf

VisualizaContado

call  
movf  
call  
call

FinInterrupcion

bcf  
retfie

INCLU  
INCLU  
INCLU  
END

En este  
respecto de la t  
tiempo en que  
se consigue un

## 17.7 RE

Durante  
contador de p  
contenido de lo

```

e servicio a la
ón debe situarse
atender eventos
una petición de
automática y, si el
interrupción.
flanco activo de
y cuando está a
programa principal
a instrucción de
de interrupción.
de la figura 17-1,
1. Es un ejemplo
*****  

iones.  

*****  

En este ejemplo se aprecia la diferencia del funcionamiento de las interrupciones
respecto de la técnica Polling, o de consulta, utilizada en el programa Int_INT_01.asm. El
tiempo en que el programa no es interrumpido está en modo de bajo consumo, con lo que
se consigue un importante ahorro de energía.

```

---

```

Inicio
    call    LCD_Inicializa
    bsf     STATUS,RP0          ; Acceso al Banco 1.
    bsf     Pulsador           ; La línea RB0/INT se configura como entrada.
    bcf     OPTION_REG,NOT_RBPU ; Activa las resistencias de Pull-Up del Puerto B.
    bcf     OPTION_REG,INTEDG   ; Interrupción INT se activa por flanco de bajada.
    bcf     STATUS,RP0          ; Acceso al Banco 0.
    clrf    Contador           ; Inicializa el contador y lo visualiza.
    call    VisualizaContador
    movlw   b'10010000'         ; Habilita la interrupción INT y la general.
    movwf   INTCON

; La sección "Principal" es de mantenimiento. Sólo espera las interrupciones en modo de bajo consumo.

Principal
    sleep
    goto   Principal          ; Pasa a modo de bajo consumo o reposo.

; Subrutina "ServicioInterrupcion"
;
; Subrutina de servicio a la interrupción. Incrementa un contador y lo visualiza.
;
ServicioInterrupcion
    call    Retardo_20ms        ; Espera a que se establezca el nivel de tensión.
    btfsc  Pulsador            ; Comprueba si es un rebote.
    goto   FinInterrupcion    ; Era un rebote y por tanto sale.
    incf   Contador,F          ; Incrementa el contador y lo visualiza.

VisualizaContador
    call    LCD_Lineal
    movf   Contador,W
    call    BIN_a_BCD
    call    LCD_Byt              ; Se debe visualizar en BCD.

FinInterrupcion
    bcf    INTCON,INTF         ; Limpia flag de reconocimiento (INTF).
    retfie                         ; Retorna y reabilita las interrupciones (GIE=1).

INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END

```

## 17.7 REGISTROS ALTERADOS POR LA INTERRUPCIÓN

Durante la interrupción el único registro que se salvaguarda en la pila es el contador de programa. La subrutina de atención a la interrupción puede modificar el contenido de los registros del microcontrolador. Al iniciarla conviene guardar el valor de

los mismos y restaurarlos en su valor antes de la instrucción *retfie*. No es predecible cuando se producirá una interrupción. Si al regreso de la subrutina de atención a la interrupción los registros no se encuentran en el mismo estado que antes de la interrupción, el programa no funcionará correctamente al usar datos que ya no son correctos.

A primera vista, salvar y restaurar los registros sin modificar sus contenidos parece que es una tarea fácil. El contenido de los registros W y STATUS deben guardarse primero, junto con los demás registros que varíen su valor durante la ejecución de la subrutina de servicio a la interrupción. Sin embargo, aunque en principio parezca correcto el siguiente procedimiento de salvaguarda de registros y restauración, éste no funcionará bien:

```
ORG 0 ; Dirección de comienzo del programa
goto Inicio
ORG 4 ; Vector de interrupción.
goto SevicioInterrupcion.
```

Inicio

...

; Subrutina "ServicioInterrupcion"

;

; Subrutina de atención a la interrupción. Como esta subrutina altera los valores del registro de trabajo W, del STATUS y de otros registros, habrá que preservar su valor previo y después restaurarlo al final.

```
CBLOCK
Guarda_W
Guarda_STATUS
Guarda_RegistroA
Guarda_RegistroB
```

....

ENDC

; Primero preserva los valores que tenían W y STATUS antes de producirse la interrupción.

ServicioInterrupcion

```
movwf Guarda_W ; Guarda W y STATUS.
movf STATUS,W ; Esta instrucción es incorrecta porque corrompe el
                ; contenido del STATUS
movwf Guarda_STATUS ; Guarda el contenido del Registro A alterado en esta
                    ; subrutina.
movf RegistroA,W ; Guarda el contenido del Registro B.
movwf Guarda_RegistroA ; Aquí se desconoce en qué banco de registros se trabaja.
movf RegistroB,W
movwf Guarda_RegistroB.
```

; (Aquí el resto de la subrutina de atención a la interrupción).

....

; Ahora restaura los valores que tenían W y STATUS y de los registros alterados por la interrupción
; antes de producirse ésta.

movf	Gua
movwf	Reg
movf	Gua
movwf	Reg
movf	Gua
movwf	STA
movf	Gua
bcf	INT
retfie	

El fragmento mover el registro de estado STATUS. M restaurar los registros

; Primero preserva los val
; puede utilizar la instrucc

ServicioInterrupcion

movwf	Guar
swapf	STA
movwf	Guard
movf	Regis
movwf	Guard
movf	Regis
movwf	Guard
.....	
bcf	STAT
.....	

(Aquí el resto de la subrut

; Ahora restaura los valores
; antes de producirse ésta.

movf	Guard
movwf	Regist
movf	Guard
movwf	Regist
swapf	Guard
movwf	STAT
swapf	Guard
swapf	Guard
bef	INTC
retfie	

La instrucción STATUS. Aunque los

es predecible  
atención a la  
antes de la  
e ya no son

enidos parece  
ben guardarse  
ecución de la  
rezca correcto  
no funcionará

trabajo  
io al final.

rompe el  
ado en esta  
tos se trabaja.

cción

movf	Guarda_RegistroB,W	; Restaura el contenido del RegistroB antes de producirse
movwf	RegistroB	; la interrupción.
movf	Guarda_RegistroA,W	; Restaura el contenido del RegistroA antes de producirse
movwf	RegistroA	; la interrupción.
movf	Guarda_STATUS,W	; Esta instrucción también es incorrecta porque altera
movwf	STATUS	; el contenido del STATUS.
movf	Guarda_W,W	; Otra instrucción que vuelve a alterar el STATUS.
bcf	INTCON,INTF	; Limpia el flag de reconocimiento de la interrupción.
retfie		; Retorna y rehabilita la interrupción.

El fragmento de programa anterior no funciona correctamente, ya que el hecho de mover el registro de trabajo W a otro registro corrompe el flag Z y modifica el registro de estado STATUS. Microchip recomienda una secuencia de código que permite salvar y restaurar los registros sin modificarlos:

; Primero preserva los valores que tenían W y STATUS antes de producirse la interrupción. No se  
; puede utilizar la instrucción "movf STATUS,W", porque corrompe el contenido del STATUS.

#### ServicioInterrupcion

movwf	Guarda_W	; Guarda W y STATUS.
swapf	STATUS,W	; Ya que "movf STATUS,W" corrompe el bit Z.
movwf	Guarda_STATUS	
movf	RegistroA,W	; Guarda el contenido del Registro A alterado en esta
movwf	Guarda_RegistroA	; subrutina.
movf	RegistroB,W	; Guarda el contenido del Registro B.
movwf	Guarda_RegistroB,	
.....		
bcf	STATUS,RP0	; Para asegurarse que trabaja dentro del Banco 0.
.....		

(Aqui el resto de la subrutina de servicio de atención a la interrupción).

; Ahora restaura los valores que tenían W y STATUS y de los registros alterados por la interrupción  
; antes de producirse ésta.

movf	Guarda_RegistroB,W	; Restaura el contenido del RegistroB antes de producirse
movwf	RegistroB	; la interrupción.
movf	Guarda_RegistroA,W	; Restaura el contenido del RegistroA antes de producirse
movwf	RegistroA	; la interrupción.
swapf	Guarda_STATUS,W	; Restaura STATUS, dejándolo como estaba.
movwf	STATUS	
swapf	Guarda_W,F	; Restaura W como estaba antes de producirse la
swapf	Guarda_W,W	; interrupción.
bcf	INTCON,INTF	; Limpia el flag de reconocimiento de la interrupción.
retfie		; Retorna y rehabilita la interrupción.

La instrucción *swapf STATUS,W* mueve los datos sin afectar al flag Z del registro STATUS. Aunque los conjuntos de 4 bits se invierten en el proceso, posteriormente son

restaurados a su situación inicial. Si se empleara la instrucción *movf STATUS,W* se corrompería el bit Z.

## 17.8 AVERIGUAR LA CAUSA DE LA INTERRUPCIÓN

El microcontrolador sólo dispone de un vector de interrupción en la dirección 004h. Esto quiere decir que, sea cual sea la fuente de la interrupción, el contador de programa se carga con la dirección 004h

Dentro de la subrutina de atención a la interrupción, el programa deberá identificar la causa de la interrupción. Para ello, debe testear el estado de los flags de interrupción de cada una de las fuentes habilitadas, para deducir cual fue la que causó la interrupción y así decidir que acción tomar. En el siguiente fragmento de programa se muestra una forma de hacerlo:

; Pasa a detectar la causa de la interrupción y ejecutar la subrutina correspondiente.

```
btfsc  INTCON,INTF    ; ¿Interrupción por activación línea RB0/INT?
call   Interrupcion_INT ; Ejecuta la subrutina de atención a interrupción INT.
btfsc  INTCON,RBIF    ; ¿Interrupción por cambio en el Puerto B?
call   Interrupcion RBI ; Ejecuta la subrutina de atención a interrupción RBI.
btfsc  INTCON,T0IF    ; ¿Interrupción por desbordamiento TMR0?
call   Interrupcion T0I ; Ejecuta la subrutina de atención a interrupción T0I.
```

## 17.9 FASES DE UNA INTERRUPCIÓN

Como resumen de todo lo explicado hasta ahora, pasamos a enumerar las acciones que realiza automáticamente el microcontrolador y las que el diseñador debe tener en cuenta a la hora de confeccionar el programa:

- 1º El programa debe habilitar las interrupciones correspondientes mediante una instrucción en la inicialización similar a la siguiente:

```
movlw  b'10111000'      ; Activa interrupción del TMR0 (T0IE), por flanco en RB0 (INT),
movwf  INTCON           ; por cambio en líneas del Puerto B (RBIE) y la general (GIE).
```

- 2º Cuando ocurre una interrupción el flag correspondiente (T0IF, INTF o RBIF) se activa. Si el bit de permiso correspondiente (T0IE, INTE o RBIE) está a "1" y el bit de habilitación de todas las interrupciones GIE está a "1", se produce la interrupción.

- 3º Para evitar que se produzca otra interrupción mientras se está atendiendo a otra anterior, el bit GIE se pone automáticamente a "0" por hardware.

- 4º El valor del contador de programa (PC) se guarda en la pila.

5º El PC

6º El pro  
salto  
correspORG  
goto  
ORG  
goto7º Seguid  
modific

; Subrutina "Servici

; Subrutina de aten

; W, del STATUS

CBLOC

Guarda\_V

Guarda\_S

Guarda\_R

Guarda\_F

.....

ENDC

; Primero preserva lo  
; puede utilizar la ins

ServicioInterrupcio

movwf

swapf

movwf

movf

movwf

movf

movwf

.....

bcf

8º A contin  
explorar

btfsc

call

btfsc

call

btfsc

call

call

9º Dependie  
correspon

oyf STATUS,W se

## INTERRUPCIÓN

ón en la dirección  
ón, el contador de

a deberá identificar  
s de interrupción de  
ó la interrupción y  
ma se muestra una

?  
ón INT.

ón RBI.

ón T0I.

umerar las acciones  
ador debe tener en

entes mediante una

flanco en RB0 (INT),  
y la general (GIE).

F, INTF o RBIF) se  
(BIE) está a "1" y el  
"1", se produce la

tá atendiendo a otra  
are.

- 5º El PC se carga con el valor 004h, que es el vector de interrupciones.
- 6º El programa debe comenzar la subrutina de atención a la interrupción con un salto a la posición de memoria, donde se encuentra las instrucciones correspondientes a la interrupción.

```
ORG 0 ; Dirección de comienzo del programa
goto Inicio
ORG 4 ; Vector de interrupción.
goto ServicioInterrupcion.
```

- 7º Seguidamente el programa debe guardar todos los registros que puedan ser modificados por la subrutina de atención a la interrupción.

---

; Subrutina "ServicioInterrupcion"

---

; Subrutina de atención a la interrupción. Como esta subrutina altera los valores del registro de trabajo W, del STATUS y de otros registros, habrá que preservar su valor previo y después restaurarlo al final.

```
CBLOCK
Guarda_W
Guarda_STATUS
Guarda_RegistroA
Guarda_RegistroB
.....
ENDC
```

; Primero preserva los valores que tenían W y STATUS antes de producirse la interrupción. No se puede utilizar la instrucción "mov STATUS,W" porque corrompe el contenido del STATUS.

### ServicioInterrupcion

movwf Guarda_W	; Guarda W y STATUS.
swarf STATUS,W	; Ya que "movf STATUS,W" corrompe el bit Z.
movwf Guarda_STATUS	
movf RegistroA,W	; Guarda el contenido del Registro A alterado en esta subrutina.
movwf Guarda_RegistroA	
movf RegistroB,W	; Guarda el contenido del Registro B.
movwf Guarda_RegistroB.	
.....	
bcf STATUS,RP0	; Para asegurarse que trabaja dentro del Banco 0.

- 8º A continuación, si están habilitadas varias vías de interrupción el programa debe explorar el valor de los flag para determinar la causa de la interrupción.

btfsc INTCON,INTF	; ¿Interrupción por activación línea RB0/INT?
call Interrupcion_JNT	; Ejecuta la subrutina de atención a interrupción INT.
btfsc INTCON,RBIF	; ¿Interrupción por cambio en el Puerto B?
call Interrupcion_RBI	; Ejecuta la subrutina de atención a interrupción RBI.
btfsc INTCON,T0IF	; ¿Interrupción por desbordamiento TMR0?
call Interrupcion_T0I	; Ejecuta la subrutina de atención a interrupción T0I.

- 9º Dependiendo de la causa de la interrupción, se bifurca a la subrutina correspondiente.

- 10º Una vez finalizado el tratamiento de la interrupción el programa debe devolver los valores que tenían los registros antes de producirse la interrupción.

movf	Guarda_RegistroB,W	; Restaura el contenido del RegistroB antes de producirse la interrupción.
movwf	RegistroB	; Restaura el contenido del RegistroA antes de producirse la interrupción.
movf	Guarda_RegistroA,W	; Restaura el contenido del RegistroA antes de producirse la interrupción.
movwf	RegistroA	; Restaura STATUS, dejándolo como estaba.
swapf	Guarda_STATUS,W	; Restaura W como estaba antes de producirse la interrupción.
movwf	STATUS	; Restaura W como estaba antes de producirse la interrupción.
swapf	Guarda_W,F	; Limpia el flag de reconocimiento de la interrupción.
swapf	Guarda_W,W	; Retorna y rehabilita la interrupción.
bcf	INTCON,INTF	
retfie		

- 11º El programa debe borrar los flags que indican las fuentes de las interrupciones (INTF, RBIF, T0IF o EEIF) antes del retorno al programa principal.

bcf	INTCON,INTF	; Limpia los flags de reconocimiento de la interrupción.
bcf	INTCON,RBIF	
bcf	INTCON,T0IF	
retfie		; Retorna y rehabilita la interrupción.

- 12º Cuando llega a la última instrucción de la rutina de interrupción *retfie*, el contador de programa se carga con el valor que se guardó inicialmente en la pila (figura 17-2.B) y el bit GIE se pone automáticamente a “1”.

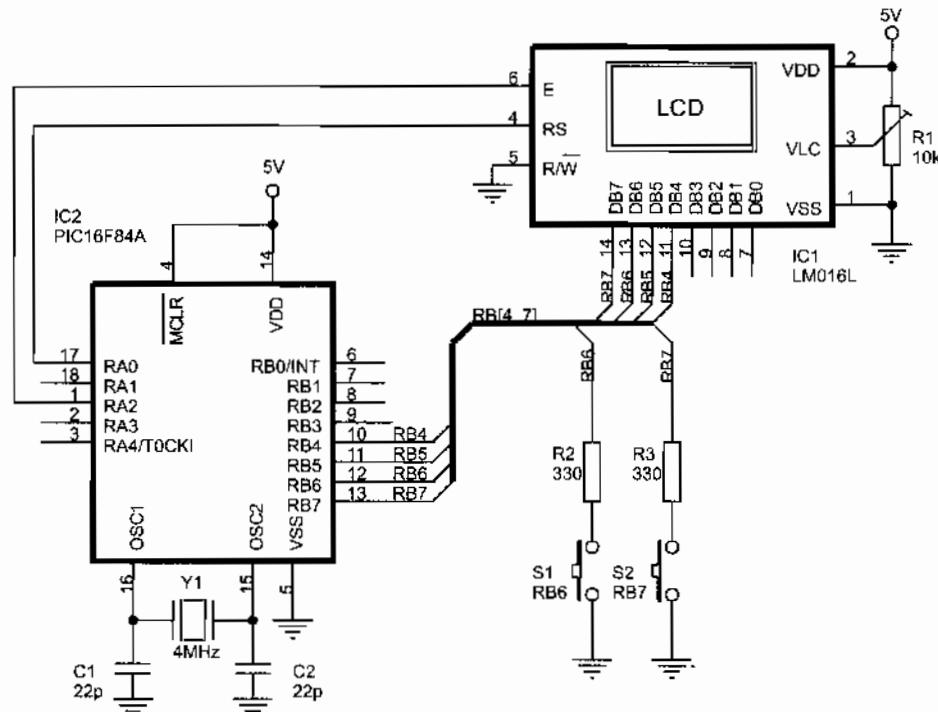


Figura 17-4 Circuito para estudiar las interrupciones RBI

## 17.10 INTERR

Para activar la RBIE y GIE del registrador INTCON se produce un cambio de nivel en el registro INTCON.

A continuación se muestra el circuito en el cual se procede a activar la RBIE y GIE del registrador INTCON con los pulsadores S1 y S2. Los pulsadores S1 y S2 cortocircuitan las líneas de control de estos pulsadores. La conexión de la línea de control de ninguno de los pulsadores se utilizó para garantizar que no se activaran más de una interrupción a la vez.

\*\*\*\*\*  
;  
; Por el zumbador se oirá un pitido.  
;  
; ZONA DE DATOS \*\*\*\*\*

```
CONFIG_C
LIST_P=16
INCLUDE <PI16F84A.H>
```

```
CBLOCK 0x0C
ENDC
```

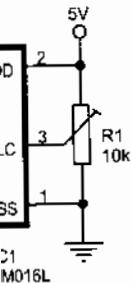
```
#DEFINE EntradaRB7
#DEFINE EntradaRB6
#DEFINE Zumbador
```

;  
; ZONA DE CÓDigos \*\*\*

ORG	0
goto	Inicio
ORG	4
goto	Servicio
<b>Inicio</b>	
bsf	STATUS
bsf	EntradaRB7
bsf	EntradaRB6
bcf	Zumbador
bcf	OPTION
bcf	STATUS
movlw	b'10001000
movwf	INTCON
<b>Principal</b>	
sleep	
goto	Principal

debe devolver  
ión.  
ntes de producirse  
ntes de producirse  
staba.  
ducirse la  
a interrupción.  
s interrupciones  
al.  
e la interrupción.

pción *refsie*, el  
lmente en la pila



## 17.10 INTERRUPCIÓN RBI

Para activar la interrupción por cambio de nivel en los pines <RB7:RB4> los bits RBIE y GIE del registro INTCON deben de estar a “1”, en estas condiciones cuando se produce un cambio de nivel en cualquiera de las líneas RB7 a RB4 se activa el flag RBIF del registro INTCON.

A continuación se desarrolla un programa ejemplo para el circuito de la figura 17-4 en el cual se procede a la lectura de dos pulsadores mediante interrupción RBI. En serie con los pulsadores es necesario conectar una resistencia con la finalidad de que no cortocircuiteen las líneas RB6 y RB7 cuando envía datos a la pantalla y a la vez se activan estos pulsadores. La característica más sobresaliente de este circuito es que no consume ninguna nueva línea del microcontrolador, pues utiliza dos líneas (RB6 y RB7) que ya eran utilizada para gobernar el módulo LCD. Esta técnica puede ser aprovechada en muchos proyectos.

```
***** Int_RBI_03.asm *****
;  
; Por el zumbador se oirá un pitido largo cuando pulse RB7 y uno corto cuando pulse RB6.  
; ZONA DE DATOS *****  
  
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC  
LIST P=16F84A  
INCLUDE <P16F84A.INC>  
  
CBLOCK 0x0C  
ENDC  
  
#DEFINE EntradaRB7 PORTB,7  
#DEFINE EntradaRB6 PORTB,6  
#DEFINE Zumbador PORTB,2  
  
; ZONA DE CÓDIGOS *****  
  
ORG 0  
goto Inicio  
ORG 4  
goto ServicioInterrupcion  
  
Inicio  
bsf STATUS,RP0  
bsf EntradaRB7  
bsf EntradaRB6  
bcf Zumbador  
bcf OPTION_REG,NOT_RBPU  
bcf STATUS,RP0  
movlw b'10001000'  
movwf INTCON  
  
Principal  
sleep  
goto Principal
```

```

; Subrutina "ServicioInterrupcion"
;
ServicioInterrupcion
    call Retardo_20ms
    btfss EntradaRB7           ; ¿Está presionado el pulsador de RB7?
    call PitidoLargo
    btfss EntradaRB6           ; ¿Está presionado el pulsador de RB6?
    call PitidoCorto
    EsperaDejePulsar          ; Espera a que desaparezcan las señales de entrada.
    btfss EntradaRB7
    goto EsperaDejePulsar
    btfss EntradaRB6
    goto EsperaDejePulsar
    bcf INTCON,RBJF
    retfie

; Subrutinas "PitidoCorto" y "PitidoLargo"
;
PitidoLargo
    bsf Zumbador
    call Retardo_200ms
PitidoCorto
    bsf Zumbador
    call Retardo_20ms
    bcf Zumbador
    return

INCLUDE <RETARDOS.INC>
END

```

Este tipo de interrupciones es especialmente idóneo para el control de un teclado matricial de 4 x 4 pulsadores, como se estudiará en el capítulo 19.

## 17.11 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores diseñar, ensamblar, simular, grabar el microcontrolador y comprobar los siguientes programas para el esquema de la figura 13-10. El lector puede introducir todas las mejoras que considere conveniente.

### INTERRUPCIÓN EXTERNA “INT”

**Int\_INT\_01.asm:** Cada vez que presione el pulsador conectado al pin RB0 se incrementará un contador que se visualizará en el módulo LCD. La lectura se realizará mediante la técnica *Polling* consultando constantemente el estado de la entrada.

**Int\_INT\_02.asm:** se incrementará un contador cada vez que se presione el pulsador conectado al pin RB0. Mientras se mantenga presionado el pulsador se hará medir el tiempo que dura la pulsación.

**Int\_INT\_03.asm:** se incrementará un contador cada vez que se presione el pulsador conectado al pin RB0. El valor del contador se incrementará cada 200 ms.

**Int\_INT\_04.asm:** cuando se presione el pulsador conectado al pin RB0/INT se comutará el LED conectado a la salida RB1. Mientras el pulsador se mantenga presionado el LED se encenderá y 200 ms a continuación se apagará.

**Int\_INT\_05.asm:** cuando se presione el pulsador conectado al pin RB0/INT el LED conectado a la salida RB1 se comutará. Mientras el pulsador se mantenga presionado el LED se encenderá y 200 ms a continuación se apagará.

**Int\_INT\_06.asm:** cuando se presione el pulsador conectado al pin RB0/INT el LED conectado a la salida RB1 se comutará. Mientras el pulsador se mantenga presionado el LED se encenderá y 200 ms a continuación se apagará.

**Int\_INT\_07.asm:** cuando se presione el pulsador conectado al pin RB0/INT se comutará el estado de la salida RB1. Mientras el pulsador se mantenga presionado el módulo LCD se visualizará un carácter.

**Int\_INT\_08.asm:** cuando se presione el pulsador conectado al pin RB0/INT se pondrá en marcha el módulo LCD. Se visualizará el carácter “INTERMITENTE” y se apagará cuando el pulsador deje de estar presionado.

### INTERRUPCIÓN “INT”

**Int RBI\_01.asm:** se incrementará un contador cada vez que se presione el pulsador conectado al pin RB7.

**Int RBI\_02.asm:** se incrementará un contador cada vez que se presione el pulsador conectado al pin RB7. Mientras se mantenga presionado el pulsador se comutará el módulo LCD y se visualizará un carácter.

**Int RBI\_03.asm:** se incrementará un contador cada vez que se presione el pulsador conectado al pin RB7. Mientras se mantenga presionado el pulsador se comutará el módulo LCD y se visualizará un carácter.

**Int RBI\_04.asm:** se incrementará un contador cada vez que se presione el pulsador conectado al pin RB7. Mientras se mantenga presionado el pulsador se comutará el módulo LCD y se visualizará un carácter.

**Int\_INT\_02.asm:** Cada vez que presione el pulsador conectado al pin RB0/INT, se incrementará un contador que se visualizará en el módulo LCD. La lectura del pulsador se hará mediante interrupciones.

**Int\_INT\_03.asm:** Cada vez que presione el pulsador conectado al pin RB0/INT se incrementará un contador que se visualiza en el módulo LCD. Si se mantiene pulsado se incrementa cada 200 ms. La lectura del pulsador se hará mediante interrupciones.

**Int\_INT\_04.asm:** Cada vez que presione el pulsador conectado a la línea RB0/INT se comutará el estado de un LED conectado a la línea RB1. La lectura del pulsador se realizará mediante interrupciones.

**Int\_INT\_05.asm:** Mientras se mantenga activado el pulsador conectado al pin RB0/INT el LED conectado a la línea RB1 parpadeará con una cadencia de 500 ms encendido y 200 ms apagado.

**Int\_INT\_06.asm:** Cada tres veces que se actúe sobre el pulsador conectado al pin RB0/INT el LED conectado a RB1 comuta de encendido a apagado y viceversa.

**Int\_INT\_07.asm:** Cada vez que presione el pulsador conectado al pin RB0/INT comutarán el estado de un LED conectado a la línea RB1. Al mismo tiempo en el módulo LCD se visualizará un mensaje desplazándose por la pantalla.

**Int\_INT\_08.asm:** Cada vez que se oprima el pulsador conectado a la línea RB0/INT se pondrá intermitente, o no, un mensaje publicitario que aparece en la pantalla del módulo LCD. En la línea superior aparecerá "Mensaje FIJO" o "M. INTERMITENTE" y en la inferior un mensaje publicitario fijo o intermitente según corresponda.

## INTERRUPCIÓN "RBI" POR CAMBIO EN LAS LÍNEAS <RB7:RB4>

**Int RBI\_01.asm:** Cada vez que presione el pulsador conectado a la línea RB6 se incrementará un contador visualizado en el módulo LCD.

**Int RBI\_02.asm:** Cada vez que se actúe sobre el pulsador conectado al pin RB6 se incrementará un minutero, es decir, contará de 0 a 59 y cuando llegue a 59 volverá de nuevo a 0. Mientras se mantenga pulsado se incrementará cada 200 ms, además, se oirá un pitido procedente de un zumbador.

**Int RBI\_03.asm:** En el zumbador conectado a la línea RB2 se oirá un pitido largo cuando pulse RB7 y uno corto cuando pulse RB6.

**Int RBI\_04.asm:** Cuando se presiona el pulsador conectado al pin RB7 en el módulo LCD se visualiza "Flanco de SUBIDA", cuando deje de oprimir el pulsador en la pantalla del módulo LCD aparecerá "FLANCO de bajada". Este ejercicio pretende

demostrar que la interrupción RBI por cambio en las líneas RB7:RB4 del Puerto B se produce tanto en el flanco de bajada como en el de subida.

**Int\_RBI\_05.asm:** A las líneas RB7 y RB6 se conectan dos pulsadores que producen una interrupción en el PIC cada vez que se pulsan. En el módulo LCD se visualizará el nombre del pulsador activado: "RB7" o "RB6".

En el tema fuentes de interrupciones

- Interrupciones de puertos
- Interrupciones de temporizadores
- Interrupciones de temporizadores <RB7:RB4>
- Interrupciones de temporizadores TMR0
- Interrupciones de temporizadores TMR1

Este tema incluye un apartado con diversas aplicaciones.

## 18.1 INTERRUPCIONES

Para autorizar una interrupción se debe establecer el bit 7 de la dirección del registro INTCON. La interrupción de temporizador TMR0 se activa cuando el temporizador TMR0 genera un溢出 (overflow). La interrupción de temporizador TMR1 se activa cuando el temporizador TMR1 genera un溢出 (overflow).

En el tema temporizadores se explica el funcionamiento de los temporizadores y el tiempo de temporización.

puerto B se

adores que  
lo LCD se

*X-18-353 Alejandro*

## CAPÍTULO 18

# INTERRUPCIÓN POR DESBORDAMIENTO DEL TIMER 0

En el tema anterior se estudió el concepto de interrupción y se citaron las 4 posibles fuentes de interrupción del PIC16F84:

- Interrupción INT. Por activación del pin RB0/INT.
- Interrupción RBI. Por cambio de estado en una de las 4 líneas de más peso <RB7:RB4> del Puerto B.
- Interrupción T0I. Por desbordamiento del TMR0.
- Interrupción EEI. Por la finalización de la escritura en la EEPROM de datos.

Este tema trata de la interrupción por desbordamiento del Timer 0 y la ejemplifica con diversas aplicaciones de gran utilidad para incluir en proyectos.

### 18.1 INTERRUPCIÓN PRODUCIDA POR EL TMR0

Para autorizar la interrupción por desbordamiento del TMR0 los bits T0IE y GIE del registro INTCON deben posicionarse a “1”. En estas condiciones cuando el temporizador TMR0 se desborda, al pasar de b'11111111' (FFh) a b'00000000' (00h), activa el flag T0IF del registro INTCON produciendo una interrupción.

En el tema 15 sobre el Timer 0 se estudió la ecuación que permite calcular el tiempo de temporización de TMR0:

$$\text{Temporización} = T_{CM} \cdot \text{Prescaler} \cdot (256 - \text{Carga TMR0})$$

Un ejemplo aplicable a muchos proyectos es el siguiente programa para el esquema de la figura 15-4, donde se explica una forma sencilla de generar ondas cuadradas, produciendo un pitido en el altavoz.

```
***** Int_TOI_01.asm *****
;
; Por la línea 3 del puerto B se genera una onda cuadrada de 10 kHz, cada semiperiodo dura
; 50 µs. Los tiempos de temporización se lograrán mediante la interrupción por desbordamiento
; del Timer 0. A la línea de salida se puede conectar un altavoz que producirá un pitido.
;
; ZONA DE DATOS *****

    CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    LIST P=16F84A
    INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

TMR0_Carga50us EQU -d'50'
#define Salida PORTB,3

;
; ZONA DE CÓDIGOS *****
ORG 0
goto Inicio
ORG 4
; Vector de interrupción.
goto Timer0_Interrupcion
Inicio
    bsf STATUS,RP0 ; Acceso al Banco 1.
    bcf Salida ; Línea configurada como salida.
    movlw b'00001000'
    movwf OPTION_REG ; Sin prescaler para TMR0 (se asigna al Watchdog).
    bcf STATUS,RP0 ; Acceso al Banco 0.
    movlw TMR0_Carga50us
    movwf TMR0
    movlw b'10100000'
    movwf INTCON ; Autoriza interrupción TOI y la general (GIE).
    ; No puede pasar a modo de bajo consumo
    ; porque detendría el Timer 0.
Principal
    goto $ ; Subrutina "Timer0_Interrupcion"
;
; Como el PIC trabaja a una frecuencia de 4 MHz el TMR0 evoluciona cada microsegundo. Para
; conseguir un retardo de 50 µs con un prescaler de 1 el TMR0 tiene que contar 50 impulsos.
; Efectivamente: 1 µs x 50 x 1 = 50 µs.
;
Timer0_Interrupcion
    movlw TMR0_Carga50us
    movwf TMR0 ; Recarga el timer TMR0.
    btfsc Salida ; Testea el anterior estado de la salida.
    goto EstabaAlto
EstabaBajo

```

```
bsf
goto
EstabaAlto
btfsc
FinInterrupcion
btfsc
retfie
```

```
; Comprobando con
; 56 µs para el nivel
```

```
END
```

El proceso es suficientemente

## 18.2 TEM

En caso de ejecución de las interrupciones se habrá que hacer experimentalmente > Stopwatch. Para ello se genera una señal cuadrada de 10 kHz.

```
*****
```

Por la línea 3 del puerto B se genera una onda cuadrada de 10 kHz, cada semiperiodo dura 50 µs exactos. Los tiempos de temporización se lograrán mediante la interrupción por desbordamiento del Timer 0. A la línea de salida se puede conectar un altavoz que producirá un pitido.

El cálculo de la carga se hace de la siguiente manera: para que el TMR0 execute las instrucciones de carga del TMR0.

### ZONA DE DATOS

```

    CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    LIST P=16F84A
    INCLUDE <P16F84A.INC>

```

```
CBLOCK 0x0C
ENDC
```

```
TMR0_Carga50us
#define Salida
```

### ZONA DE CÓDIGO

```
ORG
```

para el esquema  
ndas cuadradas,

\*\*\*\*\*

ento

\*\*\*\*\*

bsf	Salida	; Estaba bajo y lo pasa a alto.
goto	FinalInterrupcion	
EstabaAlto		
bcf	Salida	; Estaba alto y lo pasa a bajo.
FinalInterrupcion		
bcf	INTCON,T0IF	; Repone flag del TMR0.
retfie		

; Comprobando con el simulador del MPLAB se obtienen unos tiempos para la onda cuadrada de  
; 56 µs para el nivel alto y de 55 µs para el bajo.

END

El procedimiento aplicado para hallar el tiempo de temporización es suficientemente preciso para la mayoría de las aplicaciones.

## 18.2 TEMPORIZACIONES EXACTAS

En caso de requerir temporizaciones exactas, hay que tener en cuenta el tiempo de ejecución de las instrucciones y de los saltos. El valor de carga del TMR0 será algo menor y habrá que hacer un ajuste fino con instrucciones *nop*. Esto normalmente se realiza experimentalmente utilizando el simulador del MPLAB y el reloj de la ventana *Debugger > Stopwatch*. Para comprobarlo se va a repetir el programa anterior sobre la generación de una señal cuadrada de 10 kHz, pero en esta ocasión calculada con exactitud.

\*\*\*\*\* Int\_TOI\_02.asm \*\*\*\*\*

Por la línea 3 del puerto B se genera una onda cuadrada de 10 kHz. Cada semiperíodo dura 50 µs exactos. Los tiempos de temporización se lograrán mediante la interrupción del Timer 0. A la línea de salida se puede conectar un altavoz que producirá un pitido.

El cálculo de la carga del TMR0 se realiza teniendo en cuenta los tiempos que tardan en ejecutarse las instrucciones y saltos para conseguir tiempos exactos. Para calcular el valor de carga del TMR0 se ayuda del simulador del MPLAB y de la ventana de reloj Stopwatch.

ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
ENDC
```

```
TMR0_Carga50us EQU -d'43'
#define Salida PORTB,3
```

; Obtenido experimentalmente con ayuda del  
; simulador del MPLAB.

ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG 0
```

```

    goto Inicio
ORG 4 ; Vector de interrupción.
    goto Timer0_Interrupcion
Inicio
    bsf STATUS,RP0 ; Acceso al Banco 1.
    bcf Salida ; Esta línea se configura como salida.
    movlw b'00001000'
    movwf OPTION_REG ; Sin prescaler para TMR0 (se asigna al Watchdog).
    bcf STATUS,RP0 ; Acceso al Banco 0.
    movlw TMR0_Carga50us ; Carga el TMR0.
    movwf TMR0
    movlw b'10100000'
    movwf INTCON ; Autoriza interrupción T0I y la general (GIE).
; No puede pasar a modo de bajo consumo
; porque detendría el Timer 0.
Principal
    goto $

```

; Subrutina "Timer0\_Interrupcion"

; Como el PIC trabaja a una frecuencia de 4 MHz el TMR0 evoluciona cada microsegundo. Para conseguir un retardo de 50 microsegundos con un prescaler de 1 el TMR0 tiene que contar 43 impulsos. Efectivamente:  $1\mu s \times 1 \times 43 + \text{tiempo de los saltos y otros} = 50\mu s$ .

; Las instrucciones "nop" se ponen para ajustar el tiempo a 50  $\mu s$  exacto y lograr una onda cuadrada perfecta. El simulador del MPLAB comprueba unos tiempos para la onda cuadrada de 10 kHz exactos de 50  $\mu s$  para el nivel alto y otros 50  $\mu s$  para el bajo.

#### Timer0\_Interrupcion

```

    nop
    movlw TMR0_Carga50us
    movwf TMR0 ; Recarga el TMR0.
    btfsc Salida ; Testea el anterior estado de la salida.
    goto EstabaAlto
EstabaBajo
    nop
    bsf Salida ; Estaba bajo y lo pasa a alto.
    goto FinInterrupcion
EstabaAlto
    bcf Salida ; Estaba alto y lo pasa a bajo.
FinInterrupcion
    bcf INTCON,T0IF ; Repone flag del TMR0.
    retfie ; Retorno de interrupción
END

```

### 18.3 TEMPORIZACIONES LARGAS

Con la máxima división de frecuencia posible (Prescaler = 256) la temporización máxima que se puede conseguir para un circuito con cristal de cuarzo de 4 MHz es para una (Carga TMR0 = 0) igual a:

$$\text{Temporización} = T_{CM} \cdot \text{Prescaler} \cdot (256 - \text{Carga TMR0})$$

$$\text{Temporización} = 1 \times 256 \times (256 - 0) = 65.536 \mu s = 65 ms$$

Para lograr de la forma que sea intermitente para proyectos.

\*\*\*\*\*  
;  
; Un LED conectado a la salida de salida durante otros 500 ms de la interrupción por la ZONA DE DATOS

CONFIC  
LIST  
INCLUDE

CBLOCK  
Registro50  
ENDC

Carga500ms I  
Carga800ms I  
TMR0\_Carga50ms I  
#DEFINE I

ZONA DE CÓDIGO

ORG  
goto  
ORG  
goto

Para lograr temporizaciones de tiempo mayores hay que utilizar un registro auxiliar de la forma que se explica en el siguiente programa ejemplo que trata de implementar un intermitente para el circuito del esquema 18-1 y que puede utilizarse en muchos proyectos.

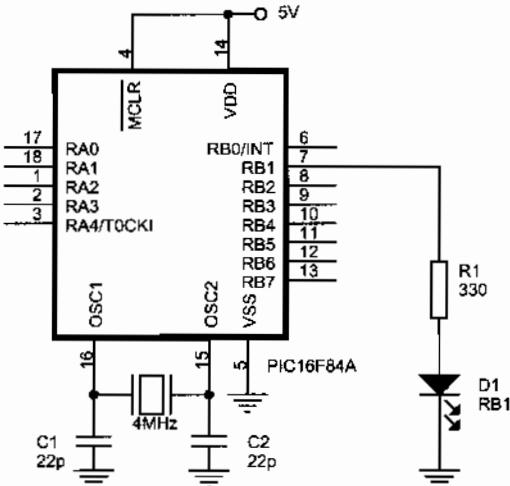


Figura 18-1 Circuito para intermitentes

```
;***** Int_T0I_04.asm *****
;Un LED conectado a la línea 1 del puerto de salida se enciende durante 800 ms y apaga
;durante otros 500 ms. Los tiempos de temporización se realizarán mediante la utilización
;de la interrupción por desbordamiento del Timer 0 del PIC.

;ZONA DE DATOS *****
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
Registro50ms          ; Registro auxiliar para conseguir una
ENDC                  ; temporización mayor.

Carga500ms    EQU    d'10'
Carga800ms    EQU    d'16'
TMR0_Carga50ms EQU    -d'195'
#define      LED     PORTB,1

;ZONA DE CÓDIGOS *****
ORG    0
goto  Inicio
ORG    4
goto  Timer0_Interrupcion ; Vector de interrupción.
```

## Inicio

```

bsf STATUS,RP0 ; Acceso al Banco 1.
bcf LED ; Línea del LED configurada como salida.
movlw b'00000111' ;
movwf OPTION_REG ; Prescaler de 256 asignado al TMR0.
bcf STATUS,RP0 ; Acceso al Banco 0.
movlw TMR0_Carga50ms ; Carga el TMR0.
movwf TMR0
movlw Carga500ms
movwf Registro50ms
movlw b'10100000'
movwf INTCON ; Autoriza interrupción del TMR0 (TOIE) y la GIE.
Principial
goto $ ; No puede pasar a modo bajo consumo porque
; el Timer 0 se detendría.

```

## ; Subrutina "Timer0\_Interrupt"

; Como el PIC trabaja a una frecuencia de 4 MHz el TMR0 evoluciona cada microsegundo.  
; El bucle central se hace en un tiempo de 50 ms. Para ello se utiliza un prescaler  
; de 256 y el TMR0 tiene que contar 195. Efectivamente  $195 \times 256 = 49929 \mu\text{s} = 50 \text{ ms}$ .  
; Para conseguir una temporización de 500 ms habrá que repetir 10 veces el lazo de 50 ms.  
; Para conseguir una temporización de 800 ms habrá que repetir 16 veces el lazo de 50 ms.

## Timer0\_Interrupt

```

movlw TMR0_Carga50ms ; Recarga el TMR0.
movwf TMR0
decfsz Registro50ms,F ; Decrementa el contador.
goto FinInterrupt ; Testea el último estado del LED.
btfs r.0,LED ; Estaba apagado y lo enciende.
goto EstabaApagado ; Repone el contador nuevamente para que esté
; 800 ms encendido.
EstabaApagado
bsf LED ; Estaba encendido y lo apaga.
movlw Carga800ms ; Repone el contador nuevamente para que esté
; 800 ms apagado.
EstabaEncendido
bcf LED ; Estaba encendido y lo apaga.
movlw Carga500ms ; Repone el contador nuevamente para que esté
; 500 ms apagado.
CargaRegistro50ms
movwf Registro50ms
FinInterrupt
bcf INTCON,TOIF ; Repone flag del TMR0.
retfie ; Retorno de interrupción.
END

```

## 18.4 TEMPORIZADOR DIGITAL

Mediante una ajustada elección de la carga del Timer 0 se puede lograr una base de tiempo suficientemente precisa para desarrollar proyectos basados en temporizaciones de tiempo real. Un proyecto típico puede ser un reloj digital propuesto en los ejercicios finales de este capítulo. Otro proyecto típico es un temporizador de precisión como el de

la figura 18-2 donde se muestra el circuito de carga con los 230

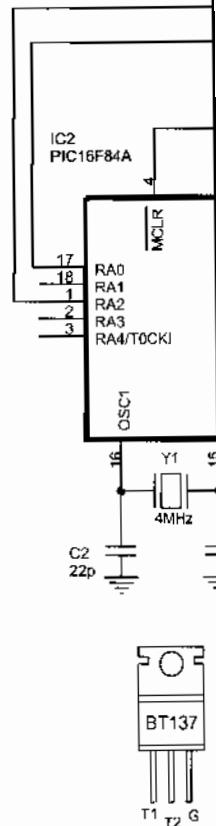


Fig.

El funcionamiento del programa de inserciones, figuras

\*\*\*\*\*  
; Programa de control p  
; de temporización se m  
; En estado de  
; tiempo deseado  
; Si se pulsa "A"  
; Cuando acaba  
; Si pulsa "A"  
; de paro: inte

la figura 18-2 donde a su salida hay conectado un módulo de potencia para alimentar una carga con los 230 V de la red eléctrica, analizado en el capítulo 2.

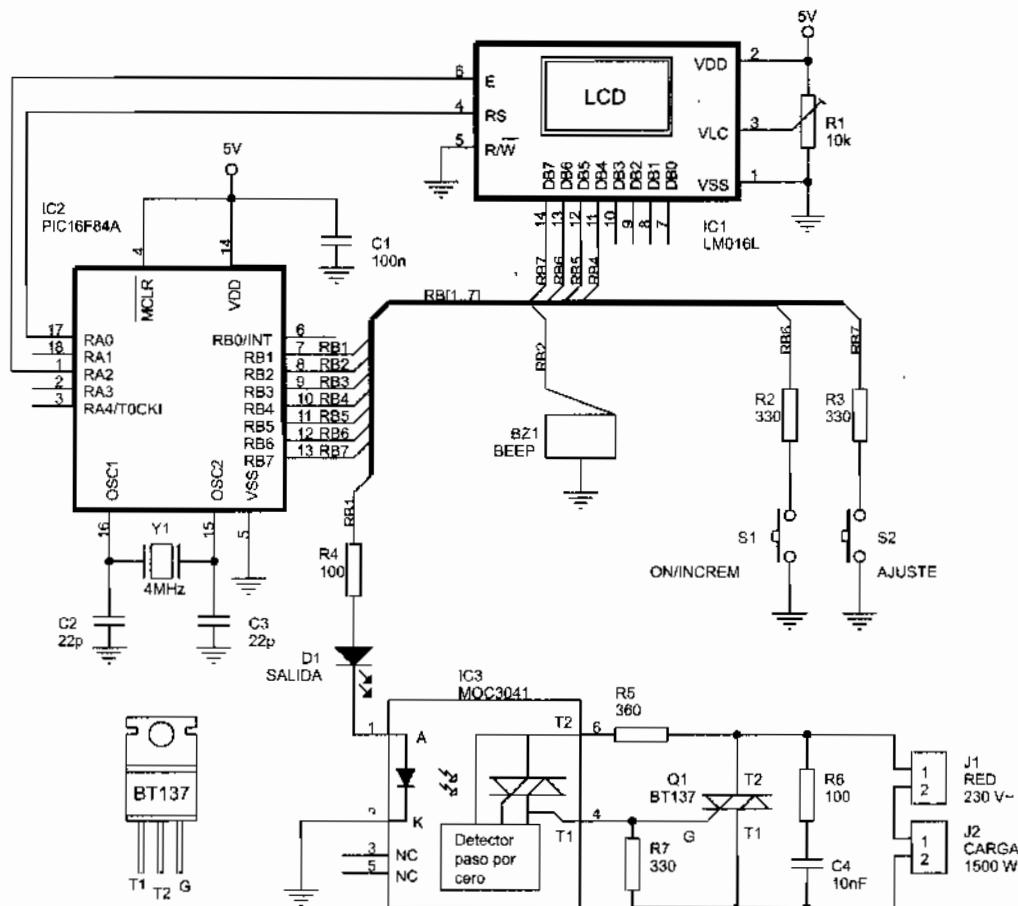


Figura 18-2 Esquema del temporizador digital de precisión

El funcionamiento de este temporizador de precisión se detalla en los comentarios del programa de control expuesto a continuación y en los diagramas descriptivos insertados, figuras 18-3 a 18-6.

```
***** INT_Temporizador.asm *****
;
; Programa de control para un temporizador digital de precisión. La programación del tiempo
; de temporización se realiza mediante dos pulsadores: "AJUSTE" y "ON/INCREMENT". Funcionamiento:
;
; - En estado de reposo la salida del temporizador está apagada y el pantalla aparece el
;   tiempo deseado para la próxima temporización.
; - Si se pulsa "ON/INCREMENT" comienza la temporización.
; - Cuando acaba la temporización pasa otra vez a reposo.
; - Si pulsa "AJUSTE" antes que haya acabado el tiempo de temporización actúa como pulsador
;   de paro: interrumpe la temporización, apaga la carga y pasa al estado de reposo.
```

;

; Para ajustar la temporización al tiempo deseado.

; - Palsa "AJUSTE" y ajusta el tiempo deseado mediante el pulsador "ON/INCREM".

; - Se vuelve a pulsar "AJUSTE" y pasa a modo de reposo.

;

; Al apagar el sistema debe conservar el tiempo de temporización deseado para la próxima vez que se encienda.

;

; ZONA DE DATOS \*\*\*\*\*

```
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST    P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
TiempoDeseado          ; El tiempo deseado de temporización.
Tiempo                  ; Tiempo que resta de temporización.
FlagsModos              ; Guarda los flags con los diferentes
ENDC                   ; modos de funcionamiento.

ORG     0x2100           ; Corresponde a la dirección 0 de la zona
DE      0x00             ; EEPROM de datos. Aquí se va a guardar el
                        ; tiempo de temporización deseado.
```

```
#DEFINE F_Temporizador_ON   FlagsModos,2
#DEFINE F_Temporizador_Ajuste FlagsModos,1
#DEFINE F_Temporizador_OFF  FlagsModos,0

#DEFINE SalidaTemporizador PORTB,1       ; Salida donde se conecta la carga.
#DEFINE Zumbador            PORTB,2       ; Salida donde se conecta el zumbador.
#DEFINE AjustePulsador      PORTB,7       ; Los pulsadores están conectados a estas
#DEFINE IncrementarPulsador PORTB,6       ; líneas del Puerto B.
```

; ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG    0
goto  Inicio
ORG    4
goto  ServicioInterrupcion
```

#### Mensajes

```
addwf  PCL,F
Mensaje_ON
  DT " En MARCHA", 0x00
Mensaje_Ajuste
  DT "Tiempo deseado:", 0x00
Mensaje_OFF
  DT " PARADO", 0x00
```

; Instrucciones de inicialización.

```
;
Inicio  call    LCD_Inicializa
        bsf    STATUS,RP0
```

movlw	b'00000011
movwf	OPTION
bsf	AjustePuls
bsf	Incrementa
bcf	SalidaTem
bcf	Zumbador
bcf	STATUS,J
clrw	
call	EEPROM
movwf	TiempoDe
call	ModoTemp
movlw	b'10001000
movwf	INTCON

Principal  
Principal

goto Principal

Figura 18-

; Subrutina "ServicioInterrupcion"

; Detecta qué ha producido la int

#### ServicioInterrupcion

btfsc	INTCON,T
call	Temporizad
btfss	INTCON,R
goto	FinInterrup
btfss	AjustePuls

movlw	b'00000111'	; Prescaler de 256 asignado al TMR0 habilita
movwf	OPTION_REG	; resistencias de Pull-Up del Puerto B.
bsf	AjustePulsador	; Configurados como entradas.
bsf	IncrementarPulsador	
bcf	SalidaTemporizador	; Configurados como salidas.
bcf	Zumbador	
bcf	STATUS.RP0	
clrw		; Lee la posición 0x00 de memoria EEPROM de datos
call	EEPROM_LeeDatos	; donde se guarda el tiempo deseado de la última vez
movwf	TiempoDeseado	; que se ajustó.
call	ModoTemporizador_OFF	; Modo de funcionamiento inicial.
movlw	b'10001000'	; Activa interrupciones RBI.
movwf	INTCON	
Principal		
goto	Principal	



Figura 18-3 Diagrama de flujo principal del temporizador

; Subrutina "ServicioInterrupcion"  
;  
; Detecta qué ha producido la interrupción y ejecuta la subrutina de atención correspondiente.

ServicioInterrupcion		
btfsr	INTCON,T0IF	
call	Temporizador	
btfsr	INTCON,RBIF	; Si es una interrupción RBI lee los pulsadores.
goto	FinInterrupcion	
btfsr	AjustePulsador	; ¿Está presionado el pulsador de "AJUSTE"?

call	CambiarModo	; Sí, pues salta a la subrutina correspondiente.
btfsc	IncrementarPulsador	; ¿Pulsado "ON/INCREM"?
goto	FinInterrupcion	; No, pues salta al final y sale.
;		
call	Retardo_20ms	; Espera que se estabilice el nivel de tensión.
btfsc	IncrementarPulsador	; Si es un rebote del pulsador "ON/INCREM" sale fuera.
goto	FinInterrupcion	
btfsc	F_Temporizador_OFF	; ¿Estaba en reposo cuando pulsó "ON/INCREM"?
call	ModoTemporizador_ON	; Sí, pues comienza la temporización.
btfsc	F_Temporizador_Ajuste	; ¿Estaba ajustando tiempo?
call	IncrementarTiempoDeseado	; Sí, pues pasa a incrementar el tiempo deseado.
FinInterrupcion		
bcf	INTCON,RBIF	; Limpia los flags de reconocimiento.
bcf	INTCON,T0IF	
retfie		

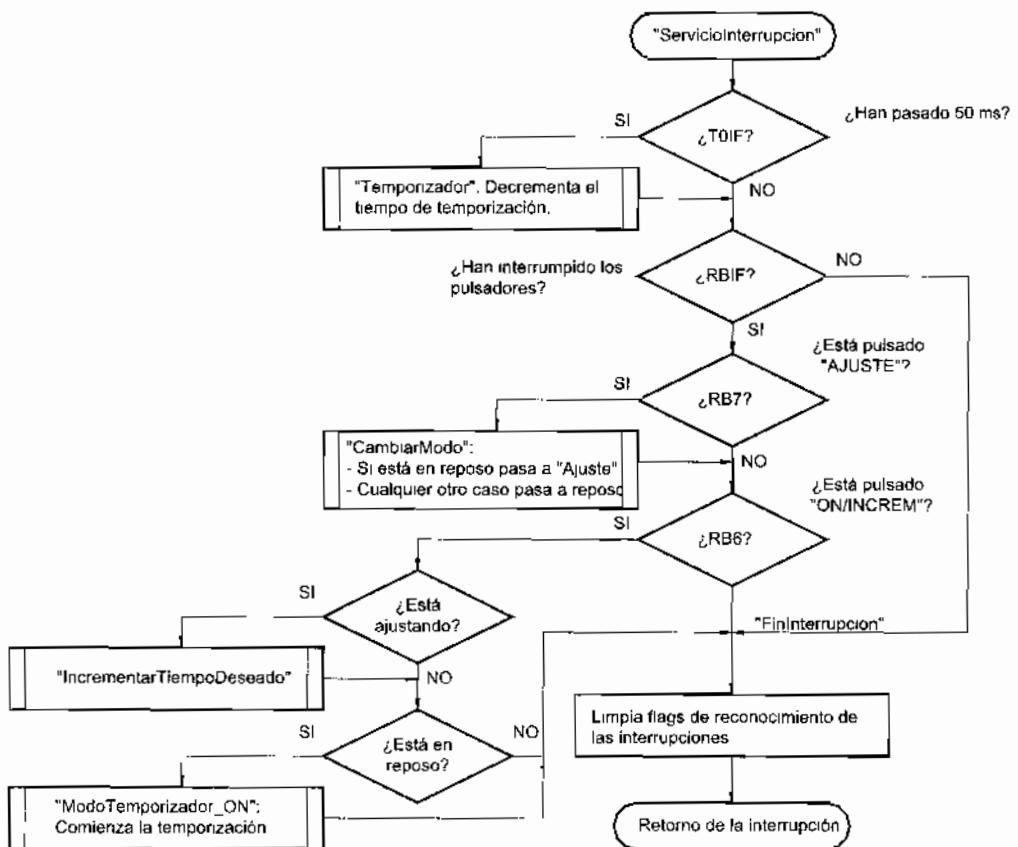


Figura 18-4 Diagrama de flujo de la subrutina de atención a las interrupciones

: Subrutinas "CambiarModo" y todas las de MODO de funcionamiento

; Subrutina de atención a la interrupción producida al presionar el pulsador "AJUSTE" que cambia el modo de funcionamiento.

- ; Hay identificados tr
- ; A) Modo "Tea
- ; que termina
- ; finalice. Re
- ; una tempor
- ; B) Modo "Ten
- ; como temp
- ; C) Modo "Ten
- ; Reconocido
- ; El programa consigu
- ; El contenido del regi
- ; - (FlagsModos)=b'00
- ; - (FlagsModos)=b'00
- ; - (FlagsModos)=b'00
- ; Al pulsar "AJUSTE"
- ; - Si estaba en modo "
- ; - Si estaba en modo "
- ; el tiempo de tempor
- ; - Si estaba en modo "
- ; temporización).

CambiarModo		Pi
call		
btfsc	Aj	
goto	Esp	
btfsc	F	
goto	Mo	
btfss	F	
goto	Mo	
clrw		
movwf	EE	
movf	Tie	
call	EE	
ModoTemporizador_OP		
bcf	Salir	
call	Piti	
movlw	b'00	
movwf	Flag	
bcf	INT	
movf	Tier	
movwf	Tier	
call	LCL	
movlw	Men	
goto	Find	

ModoTemporizador\_Aju  
bcf Salida  
movlw b'0000'

spondiente.

de tensión.  
" sale fuera.

"ON/INCREM"?  
ón.

empo deseado.

nto.

on"

¿Han pasado 50 ms?

NO

¿Está pulsado  
"AJUSTE"?

¿Está pulsado  
"ON/INCREM"?

Interrupcion"

amiento de

rupción

s interrupciones

." que cambia

; Hay identificados tres modos de funcionamiento que se diferencian mediante los tres flags:

- A) Modo "Temporizador\_OFF" o estado inicial. A él se pasa en el estado inicial cada vez que termina una temporización o cuando se aborta la temporización sin esperar a que finalice. Reconocido por el flag F\_Temporizador\_OFF, bit 0 del registro FlagsModos.
- B) Modo "Temporizador\_Ajuste", donde se ajusta la temporización deseada cuando funcione como temporizador. Reconocido por el flag F\_Temporizador\_Ajuste, bit 1 del FlagsModos.
- C) Modo "Temporizador\_ON", la salida está activada mientras dure la temporización. Reconocido por el flag F\_Temporización\_ON, que es el bit 2 del registro FlagsModos.

; El programa consigue que esté activado uno sólo de los flags anteriores.

; El contenido del registro (FlagsModos) diferencia los siguientes modos de funcionamiento:

- (FlagsModos)=b'00000001'. Está en el modo "Temporizador\_OFF", en reposo.
- (FlagsModos)=b'00000010'. Está en el modo "Temporizador\_Ajuste", ajustando tiempo deseado.
- (FlagsModos)=b'00000100'. Está en el modo "Temporizador\_ON", activa la carga y temporizador.

; Al pulsar "AJUSTE" pueden darse tres casos:

- Si estaba en modo "Temporizador\_OFF", pasa a modo "Temporizador\_Ajuste".
- Si estaba en modo "Temporizador\_Ajuste", pasa a modo "Temporizador\_OFF", pero antes salva el tiempo de temporización deseado en la EEPROM de datos.
- Si estaba en modo "Temporizador\_ON", pasa a modo "Temporizador\_OFF". (Interrumpe la temporización).

#### CambiarModo

call	PitidoCorto	; Cada vez que pulsa origina un pitido.
btfc	AjustePulsador	; Si es un rebote sale fuera.
goto	EsperaDejePulsar	
btfc	F_Temporizador_OFF	; ¿Está en reposo?
goto	ModoTemporizador_Ajuste	; Sí, pues pasa a ajustar la temporización.
btfs	F_Temporizador_Ajuste	; ¿Está ajustando?
goto	ModoTemporizador_OFF	; No, pues pasa a reposo.
clrw		; Sí, pues antes de pasar a reposo salva en la
movwf	EEADR	; posición 00h de memoria EEPROM de datos el tiempo
movf	TiempoDeseado,W	; de temporización deseado. Se conserva aunque se
call	EEPROM_EscribeDatos	apague la alimentación.

#### ModoTemporizador\_OFF

bcf	SalidaTemporizador	; Apaga la carga y resetea tiempo deseado.
call	Pitido	
movlw	b'00000001'	; Actualiza el registro FlagsModos pasando al
movwf	FlagsModos	; modo inicial "Temporizador_OFF".
bcf	INTCON,T0IE	; Prohibe las interrupciones del TMR0.
movf	TiempoDeseado,W	; Repone otra vez el tiempo que se desea para la
movwf	Tiempo	; próxima temporización.
call	LCD_Borra	; Borra la pantalla.
movlw	Mensaje_OFF	; En pantalla el mensaje correspondiente.
goto	FinCambiarModo	

#### ModoTemporizador\_Ajuste

bcf	SalidaTemporizador	; Apaga la carga
movlw	b'00000010'	; Actualiza el registro FlagsModos pasando al

movwf	FlagsModos	; modo "Temporizador_Ajuste".
clrf	Tiempo	; Resetea el tiempo.
cirf	TiempoDeseado	
bcf	INTCON,T0IE	; Prohibe las interrupciones del TMR0.
call	LCD_Borra	
movlw	Mensaje_Ajuste	; En pantalla el mensaje correspondiente.
goto	FinCambiarModo	

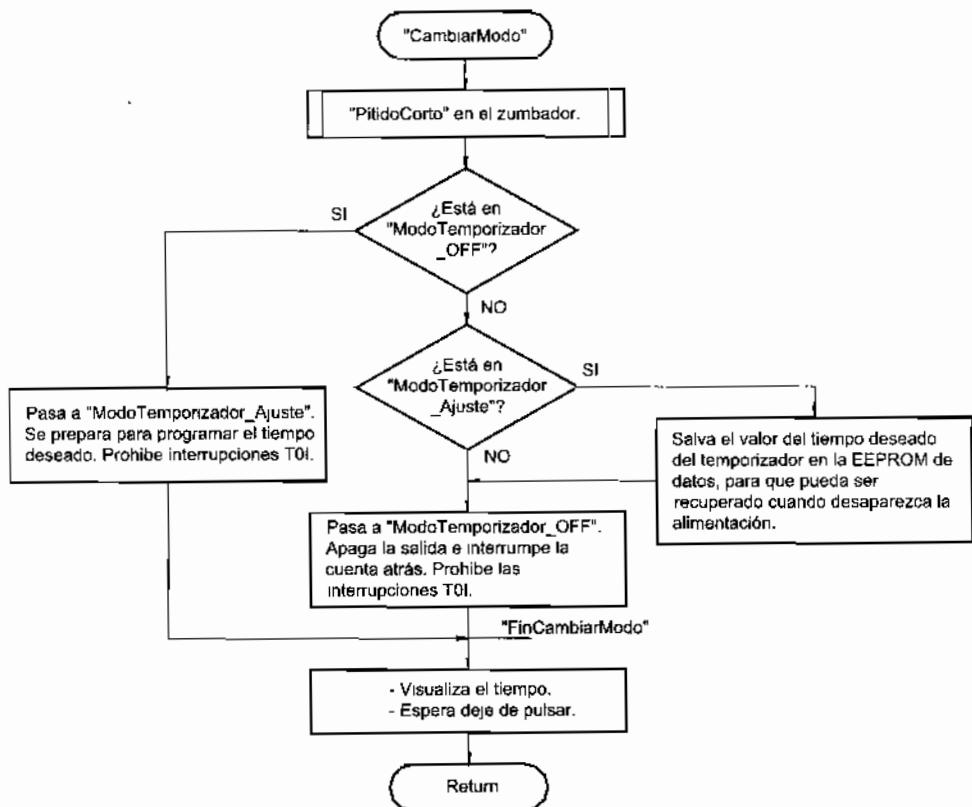


Figura 18-5 Diagrama de flujos de la subrutina CambiarModo

## ModoTemporizador\_ON

movf	TiempoDeseado,W	; Si el tiempo deseado es cero pasa a modo
btfsz	STATUS,Z	; de trabajo "Temporizador_OFF".
goto	ModoTemporizador_OFF	
movwf	Tiempo	
call	PitidoCorto	
movlw	b'00000100'	; Actualiza el registro FlagsModos pasando al
movwf	FlagsModos	; modo "Temporizador_ON".
movlw	TMR0_Carga50ms	; Carga el TMR0.
movwf	TMR0	
movlw	Carga_1s	; Y el registro cuyo decremento contará los
movwf	Registro50ms	; segundos.
bsf	INTCON,T0IE	; Autoriza las interrupciones de TMR0.

call	LC
bsf	Sal
movlw	Me
FinCambiarModo	
call	LC
call	Vis
EsperaDejePulsar	
btfs	Aju
goto	Esp
return	
; Subrutina "Temporizado"	
;	
; Esta subrutina va decremen-	
; Se ejecuta debido a la pa-	
; experimentalmente con	
CBLOCK	
Registro50ms	
ENDC	
TMR0_Carga50ms	EQU
Carga_1s	EQU
d'20'	
Temporizador	
call	Retar
call	Retar
nop	
movlw	TMR
movwf	TMR
decfsz	Regis
goto	FinTe
movlw	Carga
movwf	Regis
btfs	F_Te
goto	FinTe
decfsz	Tiemp
goto	Visua
bef	Salida
call	Visual
call	Retar
Pitido	Pitido
call	Retar
call	Pitido
call	Retar
call	Modo
call	FinTe
VisualizaContador	
call	Visua
FinTemporizador	
return	
; Subrutina "VisualizaTiem	

```

call    LCD_Borra
bsf     SalidaTemporizador
movlw   Mensaje_ON          ; Enciende la carga.
FinCambiarModo
call    LCD_Mensaje
call    VisualizaTiempo
EsperaDejePulsar
btfss  AjusteFulsador      ; Espera deje de pulsar.
goto   EsperaDejePulsar
return

```

; Subrutina "Temporizador"

;

; Esta subrutina va decrementando el tiempo de temporización y visualizándolo en la pantalla.  
; Se ejecuta debido a la petición de interrupción del Timer 0 cada 50 ms exactos, comprobado  
; experimentalmente con la ventana Stopwatch del simulador del MPLAB.

<b>CBLOCK</b> Registro50ms <b>ENDC</b> TMRO_Carga50ms EQU -d'195' Carga_1s EQU d'20'	; Guarda los incrementos cada 50 ms. ; Para conseguir la interrupción cada 50 ms. ; Leerá cada segundo (20 x 50ms = 1000 ms).
--	---

<b>Temporizador</b>		
call	Retardo_50micros	; Ajuste fino de 71 microsegundos para
call	Retardo_20micros	; ajustar a 50 milisegundos exactos.
nop		
movlw	TMRO_Carga50ms	; Carga el Timer0.
movwwf	TMR0	
decfsz	Registro50ms,F	; Decrementa el contador.
goto	FinTemporizador	; No ha pasado 1 segundo y por tanto sale.
movlw	Carga_1s	; Repone el contador nuevamente.
movwwf	Registro50ms	
btfss	F_Temporizador_ON	; Si no está en modo "Temporizador_ON" sale
goto	FinTemporizador	; fuera.
decfsz	Tiempo,F	
goto	VisualizaContador	; Visualiza el tiempo restante.
bcf	SalidaTemporizador	; Apaga la salida
call	VisualizaTiempo	; Visualiza cero segundos en la pantalla.
call	Pitido	; Tres pitidos indican final de la temporización.
call	Retardo_500ms	
call	Pitido	
call	Retardo_500ms	
call	PitidoLargo	
call	Retardo_500ms	
call	ModoTemporizador_OFF	; Acabó la temporización.
goto	FinTemporizador	
VisualizaContador		
call	VisualizaTiempo	
FinTemporizador		
return		

; Subrutina "VisualizaTiempo"

;  
; Visualiza el registro Tiempo en formato "Minutos:Segundos". Así por ejemplo, si  
;(Tiempo)=124 segundos, en la segunda línea de la pantalla visualiza " 2:04", ya que 124  
; segundos es igual a 2 minutos más 4 segundos.  
;

**VisualizaTiempo**

```
    movlw .5          ; Para centrar visualización en la
    call LCD_PosicionLinea2 ; segunda línea.
    movf Tiempo,W      ; Convierte el tiempo deseado (y expresado sólo en
    call MinutosSegundos ; segundos) a minutos y segundos.
    movf TemporizadorMinutos,W ; Visualiza los minutos.
    call BIN_a_BCD     ; Lo pasa a BCD.
    call LCD_BytE       ; Visualiza dos puntos.
    movlw '!'          ; Visualiza los segundos.
    call LCD_Caracter   ; Visualiza los segundos.
    movf TemporizadorSegundos,W ; Lo pasa a BCD.
    call BIN_a_BCD
    goto LCD_BytECompleto
    return
```

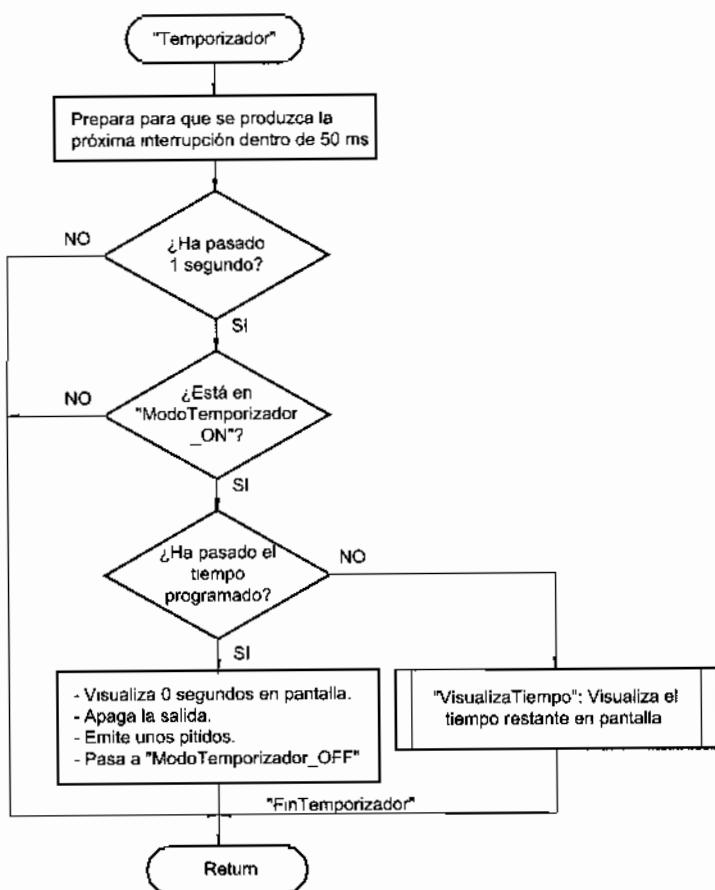


Figura 18-6 Diagrama de flujos de la subrutina Temporizador

**Subrutina "MinutosSegundos"**

Una cantidad expresada en segundos convertida a minutos y segundos llamadas Temporizadas.

El máximo número a visualizar es un número binario de 16 bits.

El procedimiento utilizado es un ejemplo que trata de la conversión de 124 segundos = 2 minutos y 4 segundos.

**Minutos**      **Segundos**

0

1

2

Entrada: En el registro Tiempo  
Salidas: En (Temporizador)

CBLOCK  
Temporizado  
Temporizado  
ENDC

**MinutosSegundos**

```
    movwf TemporizadorMinutos
    clrf TemporizadorSegundos
    movlw .60
    subwf TemporizadorSegundos
    btfss ST,0
    goto FinMinutosSegundos
    movwf TemporizadorSegundos
    incf TemporizadorMinutos
    goto Resta60
    
```

**Resta60**

```
    movlw 60
    subwf TemporizadorSegundos
    btfss ST,0
    goto FinMinutosSegundos
    incf TemporizadorMinutos
    goto Resta60
    
```

**FinMinutosSegundos**

```
    return
```

**Subrutina "IncrementaTiempoDes"**

Subrutina de atención a un pulsador conectado al pin P1.0. Muestra el tiempo restante en segundos.

**SaltoIncremento**      **EQ**

**IncrementarTiempoDes**

```
    call Pit
    movlw Segundo
    addwf TemporizadorSegundos
    
```

124

expresado sólo en  
s.

; Subrutina "MinutosSegundos"  
; Una cantidad expresada exclusivamente en segundos y contenida en el registro W es  
; convertida a minutos y segundos. El resultado se guarda en dos posiciones de memoria  
; llamadas TemporizadorMinutos y TemporizadorSegundos.

; El máximo número a convertir será el 255 que es el máximo valor que puede adquirir el  
; número binario de entrada de 8 bits. (255 segundos = 4 minutos + 15 segundos)  
; El procedimiento utilizado es mediante restas de 60 tal como se explica en el siguiente  
; ejemplo que trata de la conversión del 124 segundos a minutos y segundos.  
; 124 segundos = 2 minutos + 4 segundos.

Minutos	Segundos	$\{\text{Segundos}\} < 60?$
0	124	NO. Resta 60 a (Segundos) e incrementa (Minutos).
1	64	NO. Resta 60 e (Segundos) e incrementa (Minutos).
2	4	Si, se acabó.

; Entrada: En el registro W el número de segundos a convertir.

; Salidas: En (TemporizadorMinutos) y (TemporizadorSegundos) el resultado.

```

CBLOCK
TemporizadorMinutos
TemporizadorSegundos
ENDC

MinutosSegundos
    movwf TemporizadorSegundos      ; Carga el número de segundos a convertir.
    clrf TemporizadorMinutos        ; Carga los registros con el resultado inicial.
Resta60   movlw .60                  ; Resta 60 en cada pasada.
    subwf TemporizadorSegundos,W   ; (W)=(TemporizadorSegundos)-60.
    btfss STATUS,C                 ; ¿(W) positivo?, ¿(TemporizadorSegundos)>=60?
    goto FinMinutosSegundos        ; No, es menor de 60. Acabó.
    movwf TemporizadorSegundos      ; Si, por tanto, recupera lo que queda por restar.
    incf TemporizadorMinutos,F     ; Incrementa los minutos.
    goto Resta60                   ; Y vuelve a dar otra pasada.

FinMinutosSegundos
    return

```

; Subrutina "IncrementarTiempoDeseado"

; Subrutina de atención a la interrupción por cambio de la línea RB6 a la cual se ha  
; conectado el pulsador "INCREMENTAR".  
; Estando en el modo "Temporizador\_Ajustar" incrementa el valor del tiempo deseado  
; expresado en segundos en intervalos de 5 segundos y hasta un máximo de 255 segundos.

SaltoIncremento EQU .5

IncrementarTiempoDeseado

call	PitidoCorto	; Cada vez que pulsa se oye un pitido.
movlw	SaltoIncremento	; Incrementa el tiempo deseado de temporización
addwf	Tiempo,F	; saltos de "SaltoIncremento" segundos.

aliza el  
talla.

orizador

```

btfc  STATUS,C          ; Si pasa del valor máximo lo inicializa.
clrf  Tiempo
call  VisualizaTiempo   ; Visualiza mientras espera que deje de pulsar.
call  Retardo_100ms
btfs  IncrementarPulsador; Mientras permanezca pulsado,
goto  IncrementarTiempoDeseado ; incrementa el dígito.
movf  Tiempo,W          ; Actualiza el tiempo deseado.
movwf TiempoDeseado     ; Este es el tiempo deseado.
return

; Subrutinas "PitidoLargo", "Pitido" y "PitidoCorto"
;

PitidoLargo
    bsf   Zumbador
    call  Retardo_500ms
Pitido  bsf   Zumbador
    call  Retardo_200ms
PitidoCorto
    bsf   Zumbador
    call  Retardo_20ms
    bcf   Zumbador
    return

;
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
INCLUDE <EEPROM.INC>
END

```

## 18.5 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular, grabar el microcontrolador y comprobar los siguientes programas para el esquema de la figura 13-10.

### INTERRUPCIÓN POR DESBORDAMIENTO DEL TMR0

**Int\_T0I\_01.asm:** Por la línea 3 del puerto B se genera una onda cuadrada de 10 kHz, es decir, cada semiperíodo durará 50 µs. Los tiempos de temporización se realizarán mediante la utilización de la interrupción por desbordamiento del Timer 0 del PIC y el cálculo se hará de modo aproximado.

**Int\_T0I\_02.asm:** Repetir el ejercicio anterior para un valor exacto de tiempo.

**Int\_T0I\_03.asm:** Un LED conectado a la línea 1 del puerto de salida se enciende durante 500 ms y apaga durante otros 500 ms.

**Int\_T0**  
durante 800 ms

**Int\_T0**  
que se desplaza  
se enciende de

**Int\_T0**  
fijo. En la linea  
segundos y apaga

**GENERAC**

**Int\_Cu**  
El valor de la  
del Puerto B es

PUL  
(  
P  
Se  
T  
C  
O  
S  
Se  
Ta

Al conec  
pulsador por  
sucesivamente

**Int\_Cu**  
será fijado co  
frecuencia ini  
valor inicial y a

**Int\_Cu**  
produce la act  
"Sirena ACTIV  
la linea RB6 y

**Int\_TOI\_04.asm:** Un LED conectado a la línea 1 del puerto de salida se enciende durante 800 ms y apaga durante 500 ms

**Int\_TOI\_05.asm:** En el módulo LCD se visualiza constantemente un mensaje largo que se desplaza por la pantalla. Al mismo tiempo el diodo LED conectado a la línea RB1 se enciende durante 500 ms y apaga durante otros 500 ms a modo de segundero.

**Int\_TOI\_06.asm:** En la línea superior del visualizador LCD aparece un mensaje fijo. En la línea inferior aparece un mensaje intermitente que se enciende durante 500 ms segundos y apaga durante 300 ms.

## GENERACIÓN DE ONDAS CUADRADAS

**Int\_Cuadrada\_01.asm:** Por la línea 3 del puerto B se genera una onda cuadrada. El valor de la onda cuadrada cambia mediante activación del pulsador conectado al pin 7 del Puerto B como indica la tabla 18-1.

PULSACIÓN	FRECUENCIA	SEMIPERÍODO
(Inicial)	10 kHz	50 $\mu$ s. = 1 x 50 $\mu$ s
Primera	5 kHz	100 $\mu$ s. = 2 x 50 $\mu$ s
Segunda	2 kHz	250 $\mu$ s. = 5 x 50 $\mu$ s
Tercera	1 kHz	500 $\mu$ s. = 10 x 50 $\mu$ s
Cuarta	500 Hz	1000 $\mu$ s. = 20 x 50 $\mu$ s
Quinta	200 Hz	2500 $\mu$ s. = 50 x 50 $\mu$ s
Sexta	100 Hz	5000 $\mu$ s. = 100 x 50 $\mu$ s
Séptima	50 Hz	10000 $\mu$ s. = 200 x 50 $\mu$ s

Tabla 18-1 Funcionamiento del programa Int\_Cuadrada\_01.asm

Al conectarlo a la alimentación genera una frecuencia de 10 kHz. Al presionar el pulsador por primera vez cambia a 5 kHz, al actuar una segunda vez 2 kHz y así sucesivamente. El módulo LCD visualiza la frecuencia generada.

**Int\_Cuadrada\_02.asm:** En un altavoz se producirá el sonido de una sirena que será fijado como deseé el diseñador. En la posible solución se ha decidido que la frecuencia inicial sea de 300 Hz, subiendo hasta 4 kHz y volviendo a bajar de nuevo a su valor inicial y así continuamente.

**Int\_Cuadrada\_03.asm:** Al presionar el pulsador conectado a la línea RB7, produce la activación de una sirena y en el módulo LCD aparece un mensaje del tipo "Sirena ACTIVADA". Para apagar la sirena hay que actuar sobre el pulsador conectado a la línea RB6 y en el visualizador LCD aparece un mensaje del tipo "Sirena APAGADA".

**Int\_Cuadrada\_04.asm:** Ejemplo de control del ciclo de trabajo de una onda cuadrada. Por la línea 3 del Puerto B se genera una onda cuadrada de frecuencia constante a 100 Hz y ciclo de trabajo variable desde 0% a 100%, es decir, el tiempo en alto varía entre 0  $\mu$ s (0%) y 10000  $\mu$ s (100%). El valor del ciclo de trabajo cambia mediante activación del pulsador conectado al pin 7 del Puerto B: al conectarlo por primera vez se genera un ciclo de trabajo del 0%, al presionar el pulsador cambia al 10%, al presionar una segunda vez al 20% y así sucesivamente hasta la décima pulsación que alcanzará el 100% para volver a repetir el ciclo. El módulo LCD visualiza el ciclo de trabajo vigente.

## RELOJ DIGITAL Y TEMPORIZADOR DE PRECISIÓN

**Int\_Reloj\_01.asm:** Programa para un reloj digital en tiempo real sin puesta en hora. Se visualiza en un formato: " 8:47:39. Las temporizaciones necesarias del reloj se logran mediante el Timer 0, que produce una interrupción cada 50 ms.

**Int\_Reloj\_02.asm:** Programa para un reloj digital en cuanto al ajuste manual de las horas, es decir, sólo se van a ajustar las horas, el resto del reloj digital no funciona. Esto se realiza mediante el pulsador "INCREMENTAR" conectado al pin RB6 mediante una resistencia de 330  $\Omega$ . El reloj se visualiza en un formato: " 8:00:00" (segunda línea), donde los minutos y segundos siempre valdrán cero y el dígito de las horas se mantiene intermitente. Cada vez que se pulse INCREMENTAR, el dígito de las horas se incrementará. Las temporizaciones necesarias del reloj se logran mediante el Timer 0, que produce una interrupción cada 50 milisegundos. Con esto se logra el tiempo base necesario para obtener los 500 ms de la intermitencia.

**Int\_Reloj\_03.asm:** Programa para un reloj digital en cuanto al ajuste manual de las horas y minutos, más concretamente en cuanto al cambio entre ajuste de horas y minutos. Esto se realiza mediante el pulsador "MODO" conectado a la línea RB7 a través de una resistencia de 330  $\Omega$ . El reloj se visualiza en formato: " 0:00:00", donde las horas, minutos y segundos siempre valen cero. Se mantiene intermitente el dígito seleccionado por el pulsador "MODO" de la siguiente forma:

- 1º Pulsa "MODO", las "Horas" se ponen intermitente.
- 2º Pulsa "MODO", pasa a ajustar los "Minutos" de forma similar.
- 3º Pulsa "MODO", acaba la puesta en hora, pasando a visualización normal, (por ahora todo a cero).

**Int\_Reloj\_04.asm:** Programa para un reloj digital completo con puesta en hora resuelto a partir de los tres ejercicios anteriores. En el capítulo 24 se explica otro reloj en tiempo real cuyos diagramas de flujo descriptivos se pueden aplicar aquí.

**Int\_Reloj\_05.asm:** Visualiza el tiempo que mantiene presionado un pulsador.

**Int\_Temporizador.asm:** Programa de control para el temporizador digital de precisión de la figura 18-2.

Los sistemas  
Estos se obtienen  
sensores que mi-  
suministrados p-  
información es  
mayor o menor

En este c  
utilizados en la  
desarrollarán int

jo de una onda uencia constante ipo en alto varía mbita mediante r primera vez se 0%, al presionar que alcanzará el rabajo vigente.

al sin puesta en arias del reloj se

ste manual de las funcióna. Esto se 6 mediante una (segunda línea), horas se mantiene de las horas se e el Timer 0, que el tiempo base

ste manual de las horas y minutos. a través de una s horas, minutos ccionado por el

ión normal, (por

puesta en hora lica otro reloj en

n pulsador.

zador digital de

• Las columnas del teclado se crean con el pin 13 de la placa de desarrollo. El pin 12 es utilizada para leer los pulsos que se generan en las filas. Los pulsos se generan en las filas de acuerdo a la configuración de los resistores de fila. El valor de los resistores de fila es de 10 kΩ. Si se aplica una tensión de 5 V a la fila, el pulso se genera en la fila cuando se presiona una tecla. Si se aplica una tensión de 0 V a la fila, el pulso se genera cuando se suelta la tecla. La figura 19-1 muestra el teclado matricial.

## CAPÍTULO 19

### TECLADO MATRICIAL

Los sistemas con microcontroladores tienen como finalidad el proceso de datos. Éstos se obtienen de formas muy variadas, puede ser de manera automática por medio de sensores que midan parámetros físicos o de manera manual, en cuyo caso tienen que ser suministrados por los usuarios. Para este último caso se pueden usar pulsadores cuando la información es muy simple, tal como se ha hecho hasta ahora, o mediante teclados de mayor o menor complejidad.

En este capítulo estudiaremos los teclados matriciales, que sin duda son los más utilizados en la realización de proyectos con microcontroladores (figura 19-1), y se desarrollarán interesantes aplicaciones con ellos.

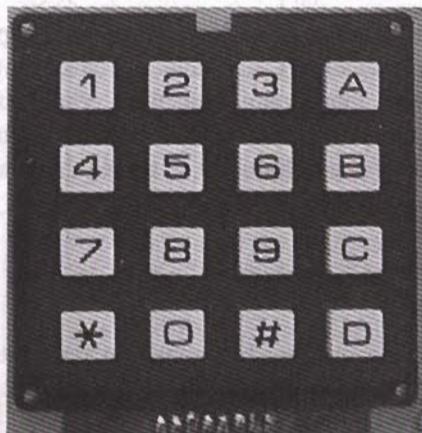


Figura 19-1 Teclado matricial

## 19.1 TECLADO HEXADECIMAL

Un **teclado matricial** está constituido por una matriz de pulsadores dispuestos en filas y columnas, figuras 19-2 y 19-3. Su intención es reducir el número de líneas necesarias para su conexionado.

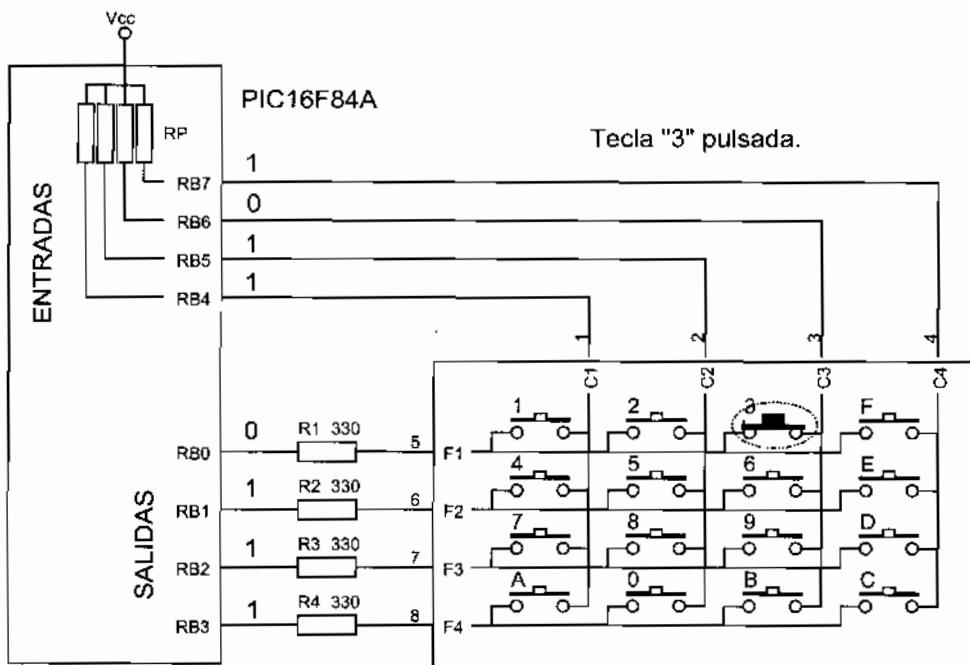


Figura 19-2 Conexión de un teclado matricial a un microcontrolador

La figura 19-3 muestra la constitución interna y el aspecto físico de un teclado matricial hexadecimal o de 16 teclas. Cada tecla se conecta a una fila y a una columna. Las 16 teclas necesitan sólo 8 líneas para conectarse con el exterior, en lugar de las 16 líneas que hubieran necesitado las teclas independientes. Cuando una tecla es pulsada, queda en contacto una fila con una columna y, si no hay tecla alguna presionada, las filas están desconectadas de las columnas.

## 19.2 CONEXIÓN DE UN TECLADO A UN PIC16F84

La figura 19-4 describe una posible conexión de un teclado hexadecimal a un sistema con microcontrolador PIC16F84A. En la figura 19-2 se destacan los aspectos principales de su conexión:

- Las filas del teclado se conectan a las líneas de la parte baja del Puerto B configuradas como salida.

- Las columnas del teclado se conectan a las líneas de la parte alta del Puerto B configuradas como entrada.

Para detectar altos (figura 19-2). Si ese nivel bajo aparece en el ejemplo de la F1, este cero aparece

Si en una fila flotante, razón por la configuración del Puerto B para el Puerto B.

Leyendo los niveles a RB7 se podrá deducir si hay tecla pulsada. Si no hay pulsada, escribiendo secuencias conectadas a las filas pulsada en su caso.

Las resistencias cortocircuitos entre el microcontrolador y el teclado.

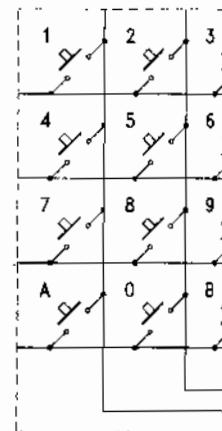


Figura 19-3 Constitución interna y aspecto físico de un teclado matricial

- Las columnas del teclado se conectan a las líneas de la parte alta del Puerto B del PIC16F84A configuradas como entradas.

Para detectar si hay alguna tecla pulsada se aplica a las filas un nivel bajo y tres altos (figura 19-2). Si se presiona alguna tecla en la fila por la que se aplica el nivel bajo, ese nivel bajo aparecerá en la columna correspondiente con la que ha hecho el contacto. En el ejemplo de la figura se pulsa la tecla "3" cuando el microcontrolador explora la fila F1, este cero aparece en la columna C3, y es leído por la línea RB6.

Si en una fila no hay pulsada tecla alguna, las entradas se encuentran en estado flotante, razón por la que son necesarias resistencias de Pull-Up. Por ello en la configuración del PIC16F84A es necesario activar las resistencias internas de Pull-Up para el Puerto B.

Leyendo los niveles de los terminales de las columnas conectados a las líneas RB4 a RB7 se podrá deducir si hay alguna tecla pulsada en la fila donde está aplicado el nivel bajo. Si no hay pulsada tecla alguna, en todas las columnas se lee un nivel alto. Escribiendo secuencialmente el nivel bajo por cada una de las líneas RB0 a RB3 conectadas a las filas y leyendo los niveles en las columnas, se podrá determinar la tecla pulsada en su caso.

Las resistencias de  $330\ \Omega$  en serie con las filas tienen como misión evitar cortocircuitos entre las líneas de la parte baja y alta del Puerto B cuando el microcontrolador utilice estas líneas para funciones distintas de la exploración del teclado.

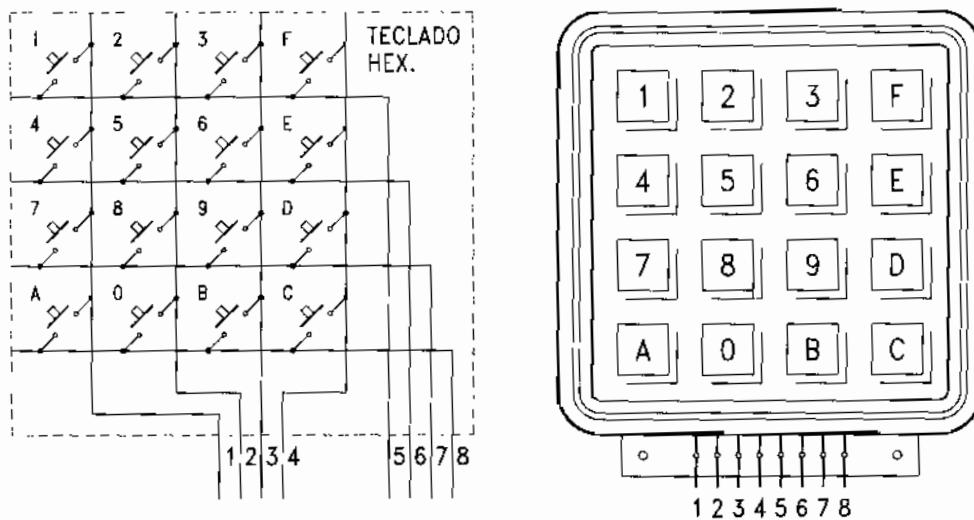


Figura 19-3 Constitución interna y aspecto físico de un teclado matricial hexadecimal

### 19.3 ALGORITMO DE PROGRAMACIÓN

La técnica de programación usada para explorar este teclado está esquematizada en el diagrama de flujo de la subrutina Teclado\_LeeOrdenTecla que lee el orden de la tecla pulsada (figura 19-6). Se entiende como "orden" el número que le corresponde a cada tecla leyéndola de izquierda a derecha y de arriba hacia abajo, independientemente de la serigrafía dibujada sobre ella, figura 19-5(A).

La exploración del teclado debe deducir el "orden" de la tecla pulsada sin importar la serigrafía sobre ésta. Cada tecla tiene asignado el número de orden indicado en la figura y que se va contabilizando en la variable Tecl\_TeclaOrden. El programa utiliza una tabla de conversión para convertir el orden de la tecla a su valor real para cada tipo de teclado.

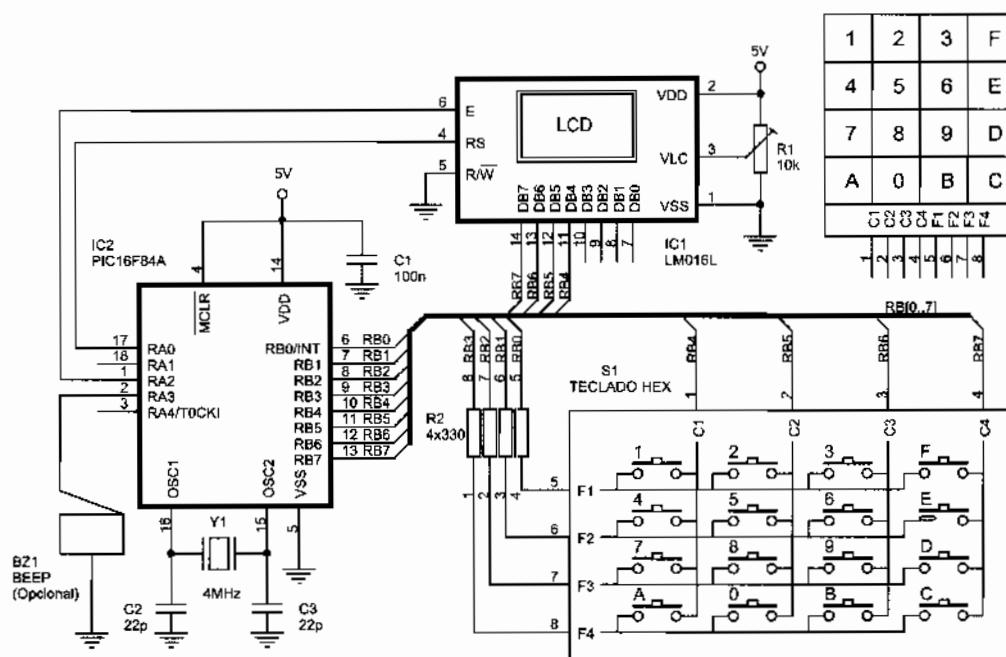


Figura 19-4 Conexión de un teclado matricial hexadecimal a un PIC16F84A

La figura 19-5 expone la relación entre el número de orden de las teclas y los valores correspondientes para un teclado hexadecimal. Así, la tecla "7" ocupa el orden 8, la tecla "F" ocupa el orden 3 y la tecla "9" el orden 10. Si cambia el teclado es necesario cambiar la tabla de conversión.

En el diagrama de flujo de la figura 19-6 se aprecia que la exploración del teclado comienza colocando a "0" la primera fila (línea RB0 del microcontrolador) y las restantes a "1". Se va testeando el estado de cada una de las columnas e incrementando al mismo

tiempo el valor de orden de la tecla exploradas de forma la fila siguiente incrementando el valor de orden de la tecla pulsada y a "0" si

Como con se expone una lib

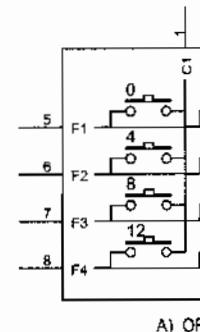


Figura 19-5 Rel

### 19.4 LIBRERIA

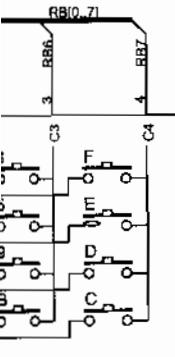
La librería hexadecimal. Par

- "Teclado" teclado re
- "Teclado" la tecla. S
- "Teclado" de la te presionad
- "Teclado" aparece e para indi

schematizada en orden de la tecla responde a cada intentemente de la

ada sin importar n indicado en la grama utiliza una para cada tipo de

1	2	3	F
4	5	6	E
7	8	9	D
A	0	B	C
5	8	3	E
6	7	2	F
1	2	3	4
5	6	7	8



PIC16F84A

e las teclas y los ocupa el orden 8, el teclado es necesario

ación del teclado (el primero) y las restantes dependiendo al mismo

tiempo el valor de la variable Tecl\_TeclaOrden, de manera tal que su valor coincide con el orden de la tecla cuya columna se lee y a cuya fila se aplica un cero. Las columnas son exploradas de forma secuencial, comprobando si hay un "0". Si no lo encuentra pone a "0" la fila siguiente y la anterior a "1", comprobando nuevamente las columnas e incrementando el valor de la variable Tecl\_TeclaOrden. Si alguna tecla es pulsada, el valor de orden de la tecla es guardado en el registro de trabajo y acaba la exploración del teclado. El bit de Carry del STATUS se activa a "1" indicando que hay alguna tecla pulsada y a "0" si no ha pulsado tecla alguna.

Como conclusión de la técnica de lectura de un teclado hexadecimal a continuación se expone una librería de subrutinas para su control.

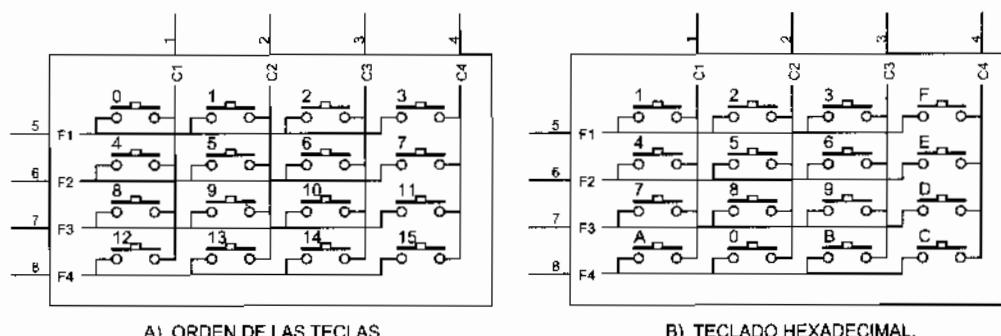


Figura 19-5 Relación entre el orden de teclas y la serigrafía de un teclado hexadecimal

## 19.4 LIBRERÍA DE SUBRUTINAS

La librería TECLADOS.INC contiene las subrutinas de control del teclado hexadecimal. Para la confección de los programas se utilizarán las siguientes subrutinas:

- "Teclado\_Inicializa". Configura las líneas del Puerto B según la conexión del teclado realizada y comprueba que no hay pulsada tecla alguna al principio.
- "Teclado\_EsperaDejePulsar". Permanece en esta subrutina mientras siga pulsada la tecla. Se utiliza para que no repita la misma lectura varias veces.
- "Teclado\_LeeOrdenTecla". Lee el teclado obteniendo en el registro W el orden de la tecla pulsada. Además, posiciona el flag Carry para indicar si se ha presionado alguna tecla.
- "Teclado\_LeeHex". Lee un teclado hexadecimal. En el registro de trabajo W aparece el valor hexadecimal de la tecla pulsada. Además posiciona el flag Carry para indicar si se ha presionado alguna tecla.

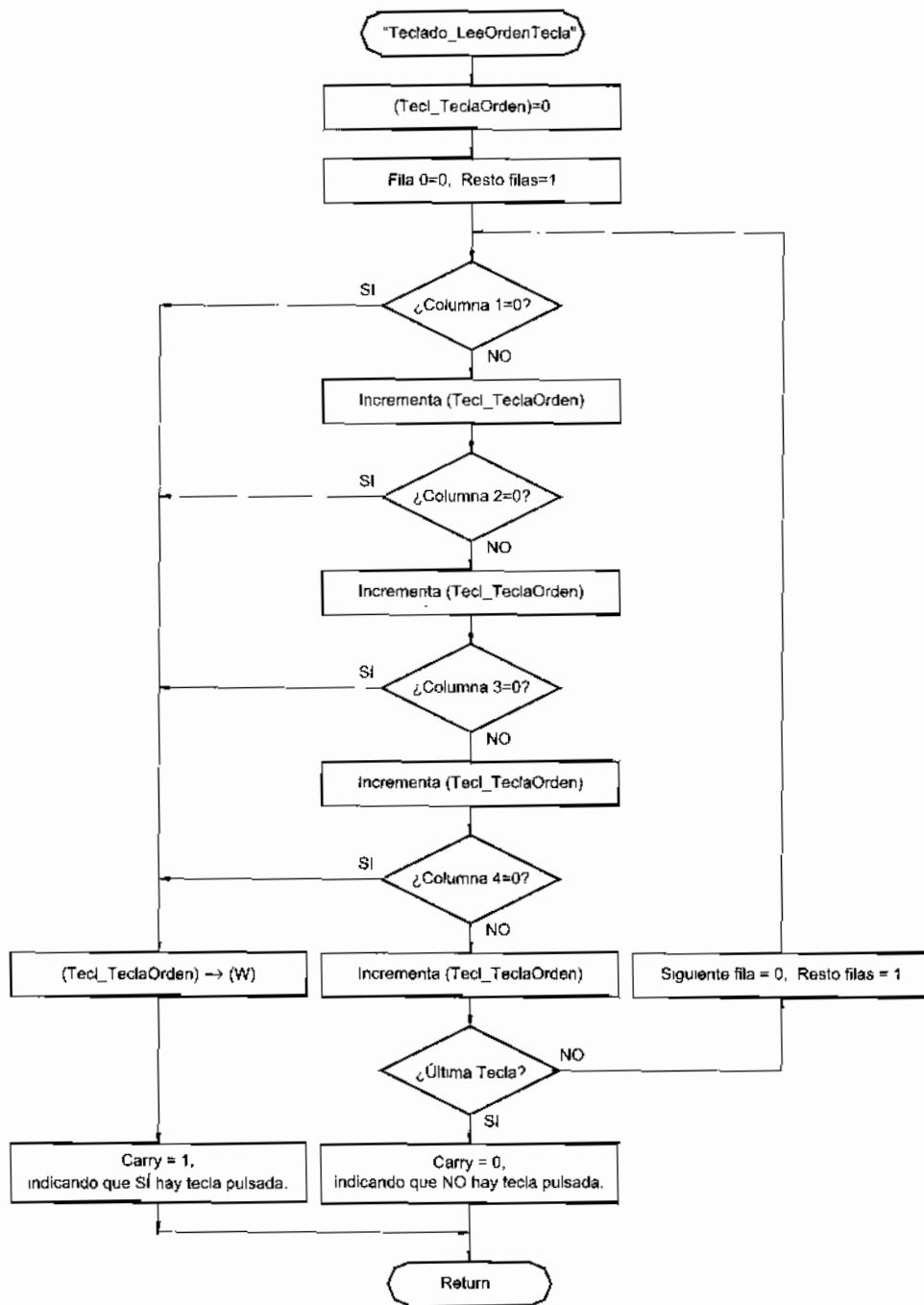


Figura 19-6 Diagrama de flujo para la exploración de un teclado hexadecimal

```

*****
; Librería de subrutina
; conectado al Puerto
;
; RB0 -->
; RB1 -->
; RB2 -->
; RB3 -->
;
; Los números que se
; que no tiene por qué
; de la tecla al valor q
;
; ZONA DE DATOS
;
; CBLOCK
; Tec1_Tecla
; ENDC
;
; Tec1_UltimaTecla
;
; Subrutina "Teclado"
;
; Cada tecla tiene asig
; Tec1_TeclaOrden. Pa
; utiliza una tabla de c
; A continuación se ex
; valores correspondie
;
; ORDEN
; 0
; 4
; 8
; 1
;
; Así, en este ejemplo,
; tecla "9" el orden 10.
;
; Si cambia el teclado
;
; Entrada: En (W) el o
; Salida: En (W) el v
;
; Teclado_LeeHex
; call T

```

```
***** Librería "TECLADO.INC" *****
;
; Librería de subrutinas para la gestión de un teclado organizado en una matriz de 4 x 4 y
; conectado al Puerto B según la disposición siguiente y explicada en la figura 19-2 del libro:
;
; RB4 RB5 RB6 RB7
;   ^ ^ ^ ^
; RB0 --> | 0 | 1 | 2 | 3 |
; RB1 --> | 4 | 5 | 6 | 7 |
; RB2 --> | 8 | 9 | 10 | 11 |
; RB3 --> | 12 | 13 | 14 | 15 |
;
; Los números que se han dibujado dentro de cada cuadrado corresponde al orden de las teclas,
; que no tiene por qué coincidir con lo serigrafiado sobre ellas. El paso del número de orden
; de la tecla al valor que hay serigrafiado sobre la misma se hace con una tabla de conversión.
;
; ZONA DE DATOS *****
;
; CBLOCK
; Tecl_TeclaOrden          ; Orden de la tecla a chequear.
; ENDC
;
; Tecl_UltimaTecla EQU    d'15'      ; Valor de orden de la última tecla utilizada.
;
; Subrutina "Teclado_LeeHex" *****
;
; Cada tecla tiene asignado un número de orden que es contabilizado en la variable
; Tecl_TeclaOrden. Para convertir a su valor según el tipo de teclado en concreto se
; utiliza una tabla de conversión.
; A continuación se expone la relación entre el número de orden de la tecla y los
; valores correspondientes para el teclado hexadecimal más utilizado.
;
; ORDEN DE TECLA:           TECLADO HEX. UTILIZADO:
;   0 1 2 3                 1 2 3 F
;   4 5 6 7                 4 5 6 E
;   8 9 10 11                7 8 9 D
;   12 13 14 15               A 0 B C
;
; Así, en este ejemplo, la tecla "7" ocupa el orden 8, la tecla "F" ocupa el orden 3 y la
; tecla "9" el orden 10.
;
; Si cambia el teclado también hay cambiar de tabla de conversión.
;
; Entrada: En (W) el orden de la tecla pulsada.
; Salida: En (W) el valor hexadecimal para este teclado concreto.
;
; Teclado_LeeHex
; call      Teclado_LeeOrdenTecla    ; Lee el Orden de la tecla pulsada.
```

Testo filas = 1

adecimal

```

        btfss STATUS,C           ; ¿Pulsa alguna tecla?, ¿C=1?
        goto Tecl_FinLeeHex    ; No, por tanto sale.
        call Tecl_ConvierteOrdenEnHex ; Lo convierte en su valor real mediante tabla.
        bsf STATUS,C            ; Vuelve a posicionar el Carry, porque la
        ; instrucción "addwf PCL,F" lo pone a "0".
Tecl_FinLeeHex
        return

;
; Tecl_ConvierteOrdenEnHex          ; Según el teclado utilizado resulta:
        addwf PCL,F
        DT 1h,2h,3h,0Fh          ; Primera fila del teclado.
        DT 4h,5h,6h,0Eh          ; Segunda fila del teclado
        DT 7h,8h,9h,0Dh          ; Tercera fila del teclado.
        DT 0Ah,0h,0Bh,0Ch        ; Cuarta fila del teclado.

Teclado_FinTablaHex
;
; Esta tabla se sitúa al principio de la librería con el propósito de que no supere la
; posición OFFh de memoria ROM de programa. De todas formas, en caso que así fuera
; visualizaría el siguiente mensaje de error en el proceso de ensamblado:
;
        IF (Teclado_FinTablaHex > 0xFF)
                ERROR "Atención: La tabla ha superado el tamaño de la página de los"
                MESSG "primeros 256 bytes de memoria ROM. NO funcionará correctamente."
ENDIF

;
; Subrutina "Teclado_Inicializa"
;
; Esta subrutina configura las líneas del Puerto B según la conexión del teclado realizada
; y comprueba que no hay pulsada tecla alguna al principio.

Teclado_Inicializa
        bsf STATUS,RP0             ; Configura las líneas del puerto:
        movlw b'11110000'           ; <RB7:RB4> entradas, <RB3:RB0> salidas
        movwf PORTB
        bcf OPTION_REG,NOT_RBPU   ; Habilita resistencia de Pull-Up del Puerto B.
        bcf STATUS,RP0              ; Acceso al banco 0.
        call Teclado_EsperaDejePulsar
        return

;
; Subrutina "Teclado_EsperaDejePulsar"
;
; Permanece en esta subrutina mientras siga pulsada la tecla.

Teclado_Comprobacion EQU b'11110000'

Teclado_EsperaDejePulsar:
        movlw Teclado_Comprobacion ; Pone a cero las cuatro líneas de salida del
        movwf PORTB                ; Puerto B.

Teclado_SigueEsperando
        call Retardo_20ms          ; Espera a que se estabilicen los niveles de tensión.
        movf PORTB,W                ; Lee el Puerto B.
        sublw Teclado_Comprobacion ; Si es lo mismo que escribió es que ya no pulsa
        btfss STATUS,Z              ; tecla alguna.
        goto Teclado_SigueEsperando

```

```

        return
;
; Subrutina "Teclado_Lee"
;
; Lee el teclado, obteniendo
; Salida: En (W) el número
; se pone
;
Teclado_LeeOrdenTecla
        clrf Tecl_W
        movlw b'11110000'
;
; Tecl_ChequeaFila
        movwf PORTE
;
; Tecl_Columna1
        btfss PORTB,7
        goto Tecl_Col1
        incf Tecl_Col1
;
; Tecl_Columna2
        btfss PORTB,6
        goto Tecl_Col2
        incf Tecl_Col2
;
; Tecl_Columna3
        btfss PORTB,5
        goto Tecl_Col3
        incf Tecl_Col3
;
; Tecl_Columna4
        btfss PORTB,4
        goto Tecl_Col4
        incf Tecl_Col4
;
; Comprueba si ha chequeado
; el contenido del registro
;
Tecl_TerminaColumnas
        movlw Tecl_Col4
        subwf Tecl_Col1
        btfsc STATUS,0
        goto Tecl_Col1
        bsf STATUS,0
        rlf Tecl_Col1,f
        goto Tecl_Col1
;
; Tecl_NoPulsada
        bcf STATUS,0
        goto Tecl_Col1
;
; Tecl_GuardaValor
        movf Tecl_W,f
        bsf STATUS,0
;
; Tecl_PinTecladoLee
        return

```

```

        return
;
; Subrutina "Teclado_LeeOrdenTecla"
;
; Lee el teclado, obteniendo el orden de la tecla pulsada.
;
; Salida: En (W) el número de orden de la tecla pulsada. Además Carry se pone a "1" si
; se pulsa una tecla ó a "0" si no se pulsa tecla alguna.
;
; Teclado_LeeOrdenTecla:
        clrf    Tecl_TeclaOrden      ; Todavia no ha empezado a chequear el teclado.
        movlw   b'11111110'          ; Va a chequear primera fila.
        Tecl_ChequeaFila           ; (Ver esquema de conexión).
        movwf   PORTB              ; Activa la fila correspondiente.
;
        Tecl_Columna1
        btfss   PORTB,4            ; Chequea la 1ª columna buscando un cero.
        goto    Tecl_GuardaValor   ; Si, es cero y por tanto guarda su valor y sale.
        incf    Tecl_TeclaOrden,F
;
        Tecl_Columna2
        btfss   PORTB,5            ; Va a chequear la siguiente tecla.
        goto    Tecl_GuardaValor   ; Repite proceso para las siguientes
        incf    Tecl_TeclaOrden,F ; columnas.
;
        Tecl_Columna3
        btfss   PORTB,6            ;
        goto    Tecl_GuardaValor   ;
        incf    Tecl_TeclaOrden,F
;
        Tecl_Columna4
        btfss   PORTB,7            ;
        goto    Tecl_GuardaValor   ;
        incf    Tecl_TeclaOrden,F
;
;
; Comprueba si ha chequeado la última tecla, en cuyo caso sale. Para ello testea si
; el contenido del registro Tecl_TeclaOrden es igual al número de teclas del teclado.
;
; Tecl_TerminaColumnas
        movlw   Tecl_UltimaTecla   ; (W) = (Tecl_TeclaOrden)-Tecl_UltimaTecla.
        subwf   Tecl_TeclaOrden,W   ; ¿C=0?, ¿(W) negativo?, ¿(Tecl_TeclaOrden)<15?
        btfsc   STATUS,C           ; No, se ha llegado al final del chequeo.
        goto    Tecl_NoPulsada     ; Si. Va a chequear la siguiente fila.
        bsf    STATUS,C             ; Apunta a la siguiente fila.
        rlf    PORTB,W
        goto    Tecl_ChequeaFila
;
        Tecl_NoPulsada
        bcf    STATUS,C             ; Posiciona C=0, indicando que no ha pulsado
        goto    Tecl_FinTecladoLee ; tecla alguna y sale.
;
        Tecl_GuardaValor
        movf    Tecl_TeclaOrden,W   ; El orden de la tecla pulsada en (W) y sale.
        bsf    STATUS,C             ; Como hay tecla pulsada, pone C=1.
;
        Tecl_FinTecladoLee
        return

```

## 19.5 EJEMPLO DE APLICACIÓN

A continuación se detalla un ejemplo de programa típico para la lectura de un teclado por parte de un PIC16F84A, referido al esquema de la figura 19-4 y librería de subrutinas anterior y que puede ser utilizado en múltiples proyectos.

```
***** Teclado_03.asm *****
; El módulo LCD visualiza el valor hexadecimal de la tecla pulsada. Para la lectura del
; teclado se utiliza la interrupción por cambio en las líneas <RB7:RB4> del Puerto B (RBI).
; ZONA DE DATOS *****
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

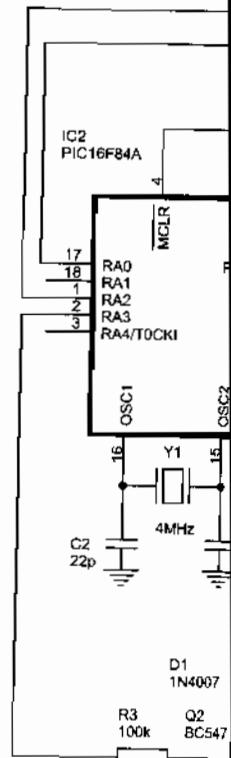
; ZONA DE CÓDIGOS *****
ORG 0
goto Inicio
ORG 4
goto ServicioInterrupcion
Inicio
call LCD_Inicializa
call Teclado_Inicializa ; Configura las líneas del teclado.
movlw b'10001000' ; Habilita la interrupción RBI y la general.
movwf INTCON
Principal
sleep ; Espera en modo bajo consumo que pulse teclado.
goto Principal

; Subrutina "ServicioInterrupcion"
;
ServicioInterrupcion
call Teclado_LeeHex ; Obtiene el valor hexadecimal de la tecla pulsada.
call LCD_Nibble ; Visualiza el valor en pantalla.
call Teclado_EsperaDejePulsar ; Para que no se repita el mismo carácter
bcf INTCON,RBIF ; mientras permanece pulsado. Limpia flag.
retfie

INCLUDE <TECLADO.INC> ; Subrutinas de control del teclado.
INCLUDE <LCD_4BIT.INC>
INCLUDE <RETARDOS.INC>
END
```

## 19.6 CERRADURA ELECTRÓNICA

Un proyecto de cerradura electrónica que se activa por teclado.



El programa

diagrama de flujo qu

```
*****
;
; Cerradura Electrónica: la
; por teclado sea correcta.
;
```

## 19.6 CERRADURA ELECTRÓNICA

Un proyecto clásico a realizar con teclado hexadecimal trata de la realización de una cerradura electrónica cuyo esquema se muestra en la figura 19-7.

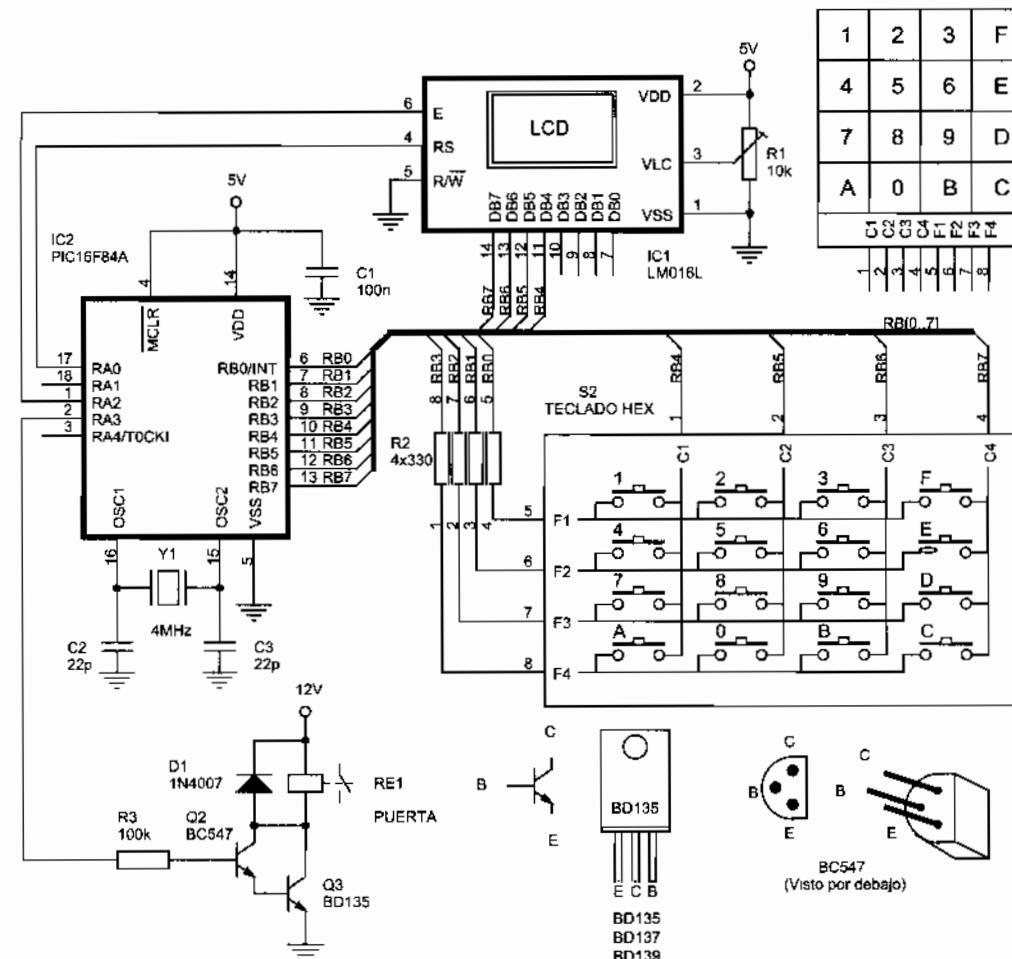


Figura 19-7 Cerradura electrónica

El programa de control se expone a continuación. La figura 19-8 detalla el diagrama de flujo que describe el funcionamiento de la subrutina más importante.

```
***** Teclado_09.asm *****
;
; Cerradura Electrónica: la salida se activa cuando una clave de varios dígitos introducida
; por teclado sea correcta. El esquema se describe en la figura 19-7.
;
```

; Tiene una salida "CerraduraSalida" que, cuando se habilita, activa durante unos segundos el electroimán de la cerradura permitiendo la apertura de la puerta:  
; - Si (CerraduraSalida) = 1, la puerta se puede abrir.  
; - Si (CerraduraSalida) = 0, la puerta no se puede abrir.

## ;Funcionamiento:

; - En pantalla visualiza "Introduzca CLAVE". Según se va escribiendo, visualiza asteriscos '\*'.  
; - Cuando termine de escribir la clave pueden darse dos posibilidades:  
; - Si la clave es incorrecta la cerradura sigue inactivada, en pantalla aparece el mensaje "Clave INCORRECTA" durante unos segundos y tiene que repetir de nuevo el proceso.  
; - Si la clave es correcta la cerradura se activa durante unos segundos y la puerta puede ser abierta. En pantalla aparece: "Clave CORRECTA" (primera linea) y "Abra la puerta" (segunda linea). Pasados unos segundos, se repite el proceso.

## ;ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC
```

; La clave puede tener cualquier tamaño y su longitud se calcula:

```
#DEFINE LongitudClave    (FinClaveSecreta-ClaveSecreta)
#DEFINE CerraduraSalida  PORTA,3
```

## ;ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG    0
goto  Inicio
ORG    4
goto  ServicioInterrupcion
```

## Mensajes

```
addwf PCL,F
```

## MensajeTeclee

```
DT    "Teclee CLAVE:", 0x00
```

## MensajeClaveCorrecta

```
DT    "Clave CORRECTA", 0x00
```

## MensajeAbraPuerta

```
DT    "Abra la puerta", 0x00
```

## MensajeClaveIncorrecta

```
DT    "Clave INCORRECTA", 0x00
```

;

## LeeClaveSecreta

```
addwf PCL,F
```

## ClaveSecreta

```
DT    4h,5h,6h,0Eh
```

; Ejemplo de clave secreta.

```
DT    7h,8h
```

## FinClaveSecreta

Inicio	call	LCD
	btf	STAT
	bef	Cerrad
	bef	STAT
	call	Teclad
	call	Iniciar
	movlw	b'1000
	movwf	INTC
Principal		
	sleep	
	goto	Princip
		Subrutina "ServicioInterru
	CBLOCK	
	ContadorCaracter	
	GuardaClaveTec	
	ENDC	
		ServicioInterrupcion
	call	Teclad
	Según va introduciendo los	
	las posiciones RAM "Clave"	
	FSR como apuntador. Por	
	movwf	INDF
	movlw	*
	call	LCD_C
	incf	FSRF
	incf	Contado
	movlw	Longitu
	subwf	Contado
	btfss	STATU
	goto	Finaliz
	Si ha llegado aquí es porque	
	procede a comprobar si la c	
	dígitos almacenados en las s	
	correcto de la clave almacen	
	Para acceder a las posicio	
	direcciónamiento indirecto	
	Para acceder a memoria RO	
	el registro ContadorCaracter	
	call	LCD_B
	clr	Contado
	movlw	ClaveTe
	movwf	FSR
		CompararClaves

```

Inicio    call    LCD_Inicializa
         bsf    STATUS,RFO
         bcf    CerraduraSalida
         bcf    STATUS,RFO
         call    Teclado_Inicializa
         call    InicializaTodo
         movlw  b'10001000'
         movwf  INTCON

Principal   sleep
             goto   Principal

; Subrutina "ServicioInterrupcion"
;
; CBLOCK
; ContadorCaracteres
; GuardaClaveTecleada
; ENDC

ServicioInterrupcion
    call    Teclado_LeeHex      ; Obtiene el valor hexadecimal de la tecla pulsada.

; Según va introduciendo los dígitos de la clave, estos van siendo almacenados a partir de
; las posiciones RAM "ClaveTecleada" mediante direccionamiento indirecto y utilizando el
; FSR como apuntador. Por cada dígito leído en pantalla se visualiza un asterisco.

; movwf  INDF      ; Almacena ese dígito en memoria RAM con
; movlw   **       ; direcciónamiento indirecto apuntado por FSR.
; call    LCD_Caracter  ; Visualiza asterisco.

; incf   FSR,F      ; Apunta a la próxima posición de RAM.
; incf   ContadorCaracteres,F ; Cuenta el número de teclas pulsadas.
; movlw  LongitudClave ; Comprueba si ha introducido tantos caracteres
; subwf  ContadorCaracteres,W ; como longitud tiene la clave secreta.
; btfs   STATUS,C    ; ¿Ha terminado de introducir caracteres?
; goto   FinInterrupcion ; No, pues lee el siguiente carácter tecleado.

; Si ha llegado aquí es porque ha terminado de introducir el máximo de dígitos. Ahora
; procede a comprobar si la clave es correcta. Para ello va comparando cada uno de los
; dígitos almacenados en las posiciones RAM a partir de "ClaveTecleada" con el valor
; correcto de la clave almacenado en la posición ROM "ClaveSecreta".

; Para acceder a las posiciones de memoria RAM a partir de "ClaveTecleada" utiliza
; direccionamiento indirecto siendo FSR el apuntador.

; Para acceder a memoria ROM "ClaveSecreta" se utiliza direccionamiento indexado con el
; el registro ContadorCaracteres como apuntador.

; call    LCD_Borra
; clrf   ContadorCaracteres
; movlw  ClaveSecreta
; movwf  FSR

CompararClaves

```

```

movf INDF,W           ; Lee la clave tecleada y guardada en RAM.
movwf GuardaClaveTecleada ; La guarda para compararla después.
movf ContadorCaracteres,W ; Apunta al carácter de ROM a leer.
call LeeClaveSecreta   ; En (W) el carácter de la clave secreta.
subwf GuardaClaveTecleada,W ; Se comparan.
btfs STATUS,Z          ; ¿Son iguales?, ¿Z=1?
goto ClaveIncorrecta  ; No, pues la clave tecleada es incorrecta.
incf FSR,F             ; Apunta a la próxima posición de RAM.
incf ContadorCaracteres,F ; Apunta a la próxima posición de ROM.
movlw LongitudClave    ; Comprueba si ha comparado tantos caracteres
subwf ContadorCaracteres,W ; como longitud tiene la clave secreta.
btfs STATUS,C            ; ¿Ha terminado de comparar caracteres?
goto ComparaClaves     ; No, pues compara el siguiente carácter.
                                ; La clave ha sido correcta. Aparecen los mensajes
ClaveCorrecta          ; correspondientes y permite la apertura de la
                        ; puerta durante unos segundos.

        movlw MensajeClaveCorrecta
        call LCD_Mensaje
        call LCD_Linea2
        movlw MensajeAbraPuerta
        call LCD_Mensaje
        bsf CerraduraSalida      ; Activa la cerradura durante unos segundos.
        goto Retardo

ClaveIncorrecta         movlw MensajeClaveIncorrecta
                        call LCD_Mensaje

Retardo                 call Retardo_2s
                        call Retardo_1s

InicializaTodo          bcf CerraduraSalida      ; Desactiva la cerradura.
                        clrf ContadorCaracteres ; Inicializa este contador.
                        movlw ClaveTecleada    ; FSR apunta a la primera dirección de la RAM
                        movwf FSR               ; donde se va a almacenar la clave tecleada.
                        call LCD_Borra          ; Borra la pantalla.
                        movlw MensajeTeclee      ; Aparece el mensaje para que introduzca la clave.
                        call LCD_Mensaje
                        call LCD_Linea2          ; Los asteriscos se visualizan en la segunda línea.

FinInterrupcion         call Teclado_EsperaDejePulsar
                        bcf INTCON,RBIF
                        retfie

INCLUDE <TECLADO.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
INCLUDE <RETARDOS.INC>

; Las posiciones de memoria RAM donde se guardará la clave leída se definen al final, después
; de los Includes, ya que van a ocupar varias posiciones de memoria mediante el
; direccionamiento indirecto utilizado.

CBLOCK
ClaveTecleada
ENDC

END                      ; Fin del programa.

```

Abre puerta y visualiza  
"Clave CORRECTA".

Figura 19-8

## 19.7 PRÁCTICA

Respetando el esquema de la figura, simular, grabar el esquema de la figura convenientemente, por ejem-

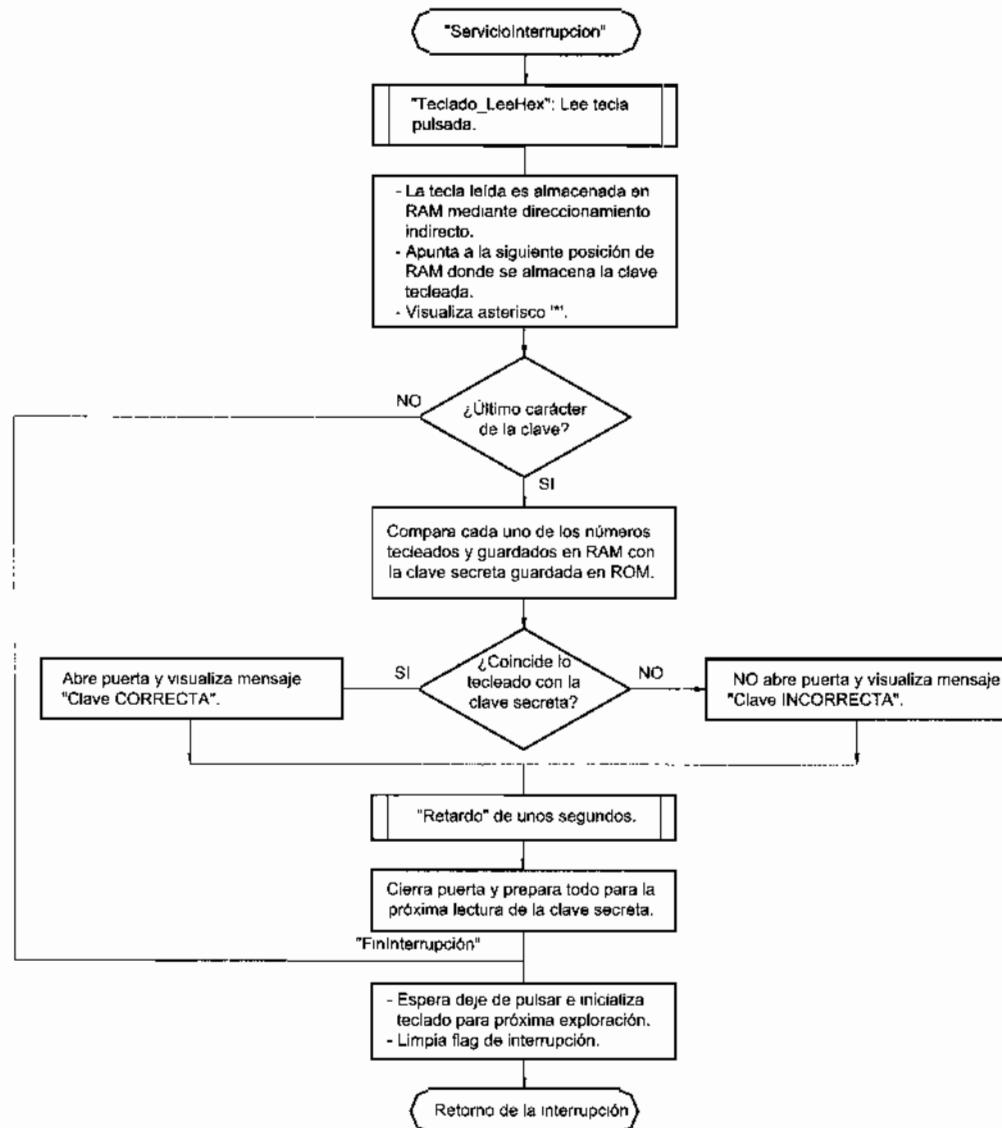


Figura 19-8 Diagrama de flujo principal para la cerradura electrónica

## 19.7 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular, grabar el microcontrolador y comprobar los siguientes programas para el esquema de la figura 19-4. El lector puede introducir todas las mejoras que considere conveniente, por ejemplo añadirle el sonido de un pitido cada vez que pulse una tecla.

**Teclado\_01.asm:** En el visualizador LCD aparece el orden de la tecla pulsada. Se utiliza la técnica Polling o lectura constante del puerto al que se ha conectado el teclado.

**Teclado\_02.asm:** En la pantalla se visualiza el valor hexadecimal de la tecla pulsada. Se utiliza la técnica lectura constante del puerto al que se ha conectado el teclado.

**Teclado\_03.asm:** Lo que se va escribiendo por el teclado aparece en la pantalla del display LCD. Utiliza la interrupción de las líneas <RB7:RB4> del Puerto B donde está conectado el teclado.

**Teclado\_04.asm:** Supone un teclado con los 16 primeros caracteres del alfabeto occidental, por tanto habrá que cambiar la tabla respecto de los ejercicios anteriores. Lo que se va escribiendo por el teclado aparece por pantalla como si fuera un teclado de caracteres alfabéticos.

**Teclado\_05.asm:** Primero aparece un mensaje de presentación que se desplaza a lo largo de la pantalla. Luego en la primera línea aparece un mensaje estático y lo que se va escribiendo por el teclado aparece en el visualizador LCD.

**Teclado\_06.asm:** Lo que se está escribiendo por el teclado aparece en la pantalla. Cuando llega al final de la primera línea pasa a la segunda. Cuando llega al final de la segunda línea borra todo y comienza de nuevo al principio de la primera.

**Teclado\_07.asm:** Los valores decimales que se escriben por el teclado aparecen en la pantalla del display LCD. Si pulsa cualquier otra tecla que no sea número lo interpreta como tecla de borrado de la pantalla.

**Teclado\_08.asm:** Suma el valor de tres teclas consecutivas pulsadas. En la primera línea aparece en hexadecimal y en la segunda en decimal. Así por ejemplo si pulsa "A", "6" y "F" en pantalla aparecerá:

- "Hex: A+6+F=1F " (Primera Línea).
- "Dec: 10+06+15=31" (Segunda Línea).

**Teclado\_09.asm:** Programa de una cerradura electrónica cuya salida se activa cuando una clave de varios dígitos introducida por el teclado sea correcta (figura 19-7).

Algunos ci  
cada cierto tiempo  
(estadístico por e  
ordenadores perso  
dotar al sistema de  
tema del presente d

## 20.1 PUERTOS

La forma n  
ordenador es a trav  
**RS232** (o **EIA232**)  
normalmente deno

Los puertos  
dos tipos de cone  
hembras, figura 20-  
modem, por lo que  
aplicaciones no se

Cada una de  
norma. Hay unos  
controlan el estable  
patillas del DB-9.

tecla pulsada. Se  
ctado el teclado.

ecimal de la tecla  
ectado el teclado.

en la pantalla del  
erto B donde está

teres del alfabeto  
ios anteriores. Lo  
era un teclado de

ue se desplaza a lo  
ico y lo que se va

ce en la pantalla.  
lega al final de la

clado aparecen en  
mero lo interpreta

das. En la primera  
mplo si pulsa "A",

salida se activa  
(figura 19-7).

Gabriel

Consejero

## CAPÍTULO 20

# COMUNICACIÓN CON ORDENADOR

Algunos circuitos microprogramados se encargan de almacenar parámetros que cada cierto tiempo han de ser recogidos o "volcados" en otros sistemas para su estudio (estadístico por ejemplo). Es frecuente que los sistemas de recogida de datos sean ordenadores personales, por lo que en la realización de algunos proyectos será necesario dotar al sistema de la capacidad de intercambiar información con ordenadores. Este es el tema del presente capítulo.

### 20.1 PUERTO SERIE RS232

La forma más común y sencilla de comunicar cualquier dispositivo con un ordenador es a través de su **puerto serie**, que es compatible con el denominado estándar RS232 (*o EIA232 Standard*). En un ordenador puede haber varios puertos series, normalmente denominados COM 1, COM 2, etc.

Los puertos serie son accesibles mediante conectores. La norma RS232 establece dos tipos de conectores llamados DB-25 (de 25 pines) y DB-9 (de 9 pines), machos y hembras, figura 20-1. La norma RS232 se estableció para conectar un ordenador con un modem, por lo que aparecen muchas patillas en los conectores DB-25 que en otro tipo de aplicaciones no se utilizan y en las que es más común utilizar el conector tipo DB-9.

Cada una de las patillas del conector RS232 tiene una función especificada por la norma. Hay unos terminales por los que se transmiten y reciben datos y otros que controlan el establecimiento, flujo y cierre de la comunicación. La figura 20-2 describe las patillas del DB-9.

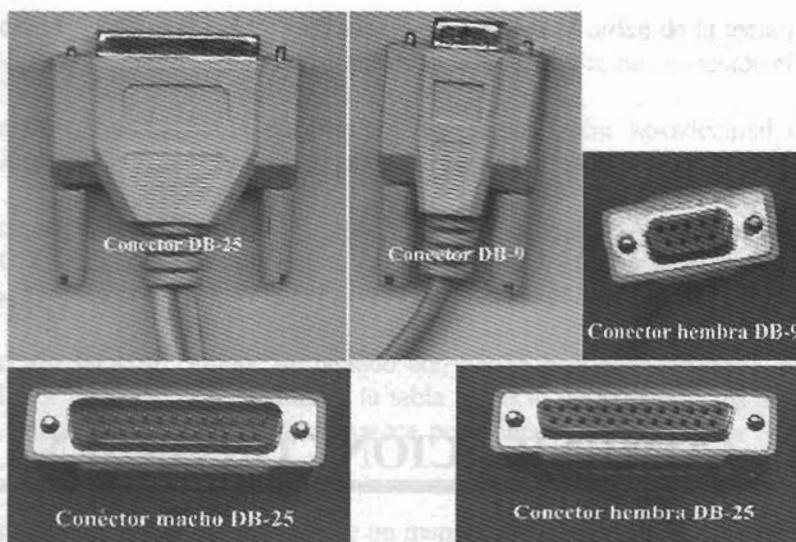


Figura 20-1 Conectores RS232

Para comunicarse con un microcontrolador es suficiente con tres líneas:

- Línea de transmisión (TxD), pin 3 (*Transmitted Data*).
- Línea de recepción (RxD), pin 2 (*Received Data*).
- Pin de masa (SG), pin 5 (*Signal Ground*).

Las especificaciones del puerto serie están contenidas en la norma RS232 (ó *EIA232 Standard*). En este capítulo se expondrán aquellas necesarias para comunicarse con un microcontrolador.

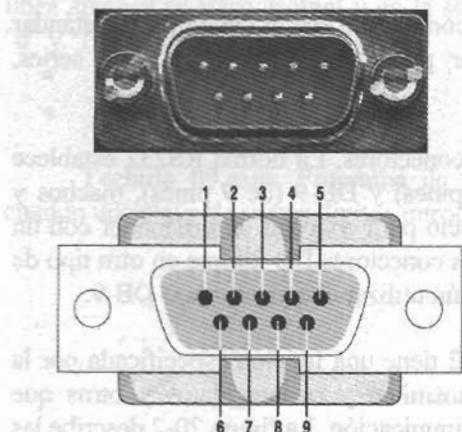


Figura 20-2 Patillaje del conector DB-9

PIN	SEÑAL
1	<i>Data Carrier Detect (DCD)</i>
2	<i>Received Data (RxD)</i>
3	<i>Transmitted Data (TxD)</i>
4	<i>Data Terminal Ready (DTR)</i>
5	<i>Signal Ground (SG)</i>
6	<i>Data Set Ready (DSR)</i>
7	<i>Request to Send (RTS)</i>
8	<i>Clear to Send (CTS)</i>
9	<i>Ring Indicator (RI)</i>

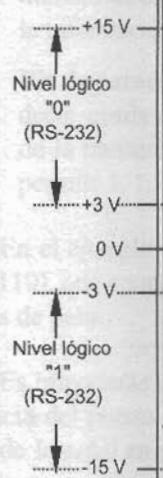
## 20.2 EL BAUDI

Un dato importante es la velocidad de transmisión, que es la cantidad de información enviada en un período de tiempo. Hay una medida llamada Baudio, que es proporcional a la velocidad de transmisión.

La velocidad de transmisión se normaliza a 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 y 57600 baudios. Los datos se envían en una multitud de aplicaciones.

## 20.3 NIVELES DE VOLTAJE

La figura 20-3 muestra los niveles de voltaje para una transmisión serie RS-232.



- Los datos se transmiten en forma de señal de conexión representada por niveles de voltaje.
- Para garantizar la calidad de la señal, los niveles de voltaje deben ser de al menos 5 V.
- Del mismo modo, el voltaje de tierra debe ser de al menos 5 V.
- Los voltajes negativos se usan para representar el nivel lógico "1".

## 20.2 EL BAUDIO

Un dato importante a tener en cuenta en cualquier comunicación es la **velocidad de transmisión**, que es la cantidad de información enviada por la línea de transmisión en la unidad de tiempo. Hay distintas unidades para expresar esta medida, la más utilizada es el **Baudio**, que es proporcional a los Bits/segundo (bps), definidos como el número de bits de información enviados por segundo.

La velocidad a la que pueden trabajar los puertos COM de un ordenador está normalizada a 75, 150, 300, 600, 1200, 2400, 4800, 9600 Baudios, etc. Estos valores son demasiado pequeños para los estándares de hoy en día, pero suficientemente rápidos para multitud de aplicaciones.

## 20.3 NIVELES LÓGICOS RS232

La figura 20-3 ilustra los requisitos en cuanto a niveles lógicos que debe cumplir una transmisión serie según la norma RS232. A destacar:

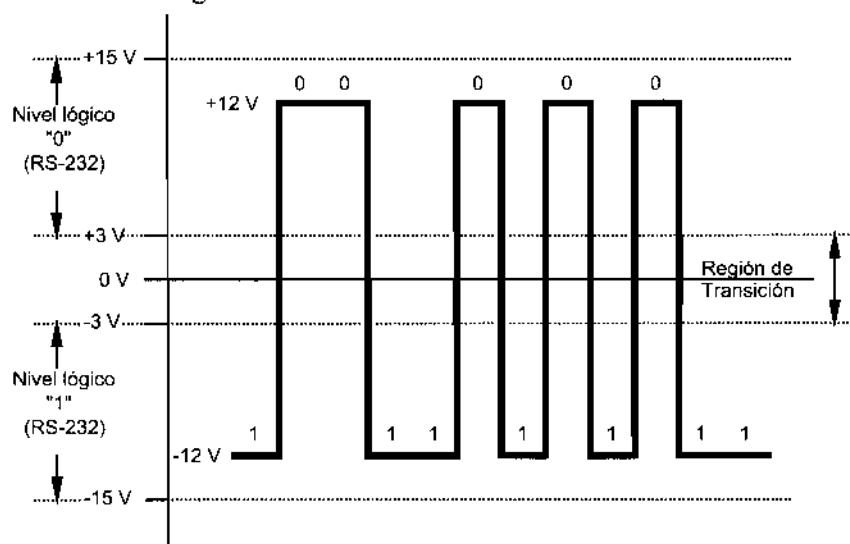


Figura 20-3 Niveles de tensión lógicos para RS232

- Los datos se transmiten con lógica negativa, es decir, un voltaje positivo en la conexión representa un "0", mientras que un voltaje negativo representa un "1".
- Para garantizar un "0" lógico una línea debe mantener un voltaje entre +3 y +15 V.
- Del mismo modo para "1" lógico garantizado debe estar entre -3 y -15 V.
- Los voltajes más usados son +12 V para el "0" y -12 V para el "1".

- Es importante resaltar que cuando un puerto serie no está transmitiendo mantiene el terminal de transmisión a "1" lógico a -12 V, normalmente.
- La banda muerta entre +3 V y -3 V se conoce como la región de transición donde los niveles lógicos no están definidos. Esto significa que cualquier valor entre +3 y -3 voltios puede interpretarse ambiguamente como "0" ó "1".

Si se aumenta la velocidad de transmisión las señales de datos se vuelven susceptibles a pérdidas de voltaje causadas por la capacidad, resistencia e inductancia del cable. Estas pérdidas son conocidas como efectos de alta frecuencia y aumentan con la longitud del cable.

Estos valores de tensión proporcionan un amplio margen de seguridad que es de gran utilidad cuando los cables deben pasar por zonas cercanas a elementos que generan interferencias eléctricas: motores, transformadores, equipos de comunicación, etc. Estos elementos, unidos a la longitud del cable, pueden hacer disminuir la señal hasta en varios voltios, sin que afecte adversamente al nivel lógico de la entrada.

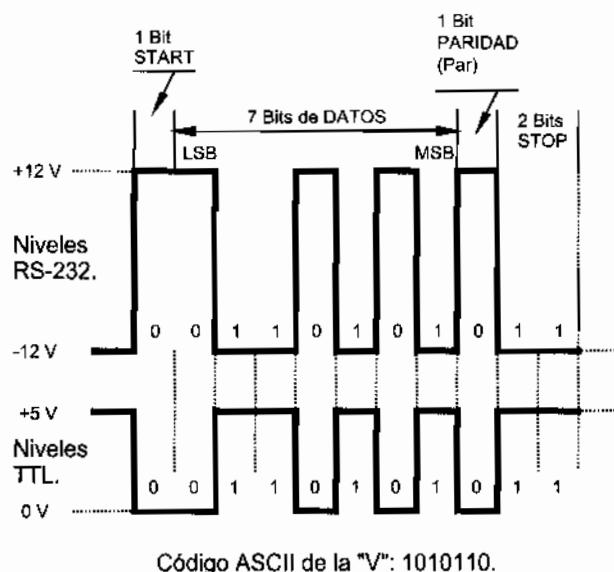


Figura 20-4 Ejemplo de envío de un byte según normas RS232

## 20.4 FORMATO DE UN BYTE

La comunicación de datos en un puerto serie RS232 se usa normalmente para efectuar comunicaciones asíncronas sin tiempo preestablecido para iniciarse. Los datos llegan en paquetes de información normalmente de 8 bits. Algunos equipos envían carácter por carácter, otros guardan muchos caracteres en la memoria y cuando les toca enviarlos los envían uno tras otro.

El protocolo es 4 partes, ilustradas en

- **Bit de inicio**: "1" a un "0" detecta el bit de inicio, entonces, define la función de la señal.
- **Bits de datos**: de *Start*. El bit de *Start* y el de *mayor* suele consistir en un solo bit de datos.
- **Bit de paridad**: bit de paridad sencillas no se transmiten. Se transmite una palabra de datos y el número de bits de la información.
- **Bit de parada**: decir, queda al final de la transmisión. Permite 1, 1.5 o 2 bits de parada.

En el ejemplo de la figura, el código ASCII de la letra "V" (101 0110), que corresponde a 7 bits de datos más 1 bit de paridad y 2 bits de parada.

Es importante tener en cuenta la diferencia del puerto serie en la forma de la señal en el receptor, tal como se muestra en la figura.

## 20.5 MAX232

En el mercado existen chips que convierten niveles TTL y niveles RS-232. Uno de estos chips es el MAX232 fabricado por Dallas Semiconductor.

El MAX232 consiste en un driver TTL → RS232 (que convierte niveles TTL a niveles RS-232) y un receptor RS232 → TTL (que convierte niveles RS-232 a niveles TTL).

mitiendo mantiene  
e transición donde  
uer valor entre +3

datos se vuelven  
a e inductancia del  
r aumentan con la

guridad que es de  
entos que generan  
icación, etc. Estos  
ial hasta en varios

5232

ormalmente para  
ciarse. Los datos  
s equipos envían  
y cuando les toca

El protocolo establecido por la norma RS232 envía la información estructurada en 4 partes, ilustradas en la figura 20-4 con un ejemplo:

- **Bit de inicio o arranque (*Start*)**. Es un paso de -12 V a +12 V, es decir, de un "1" a un "0" lógico en la lógica negativa de la norma RS232. Cuando el receptor detecta el bit de inicio sabe que la transmisión ha comenzado y, a partir de entonces, debe leer las señales de la línea a distancias concretas de tiempo en función de la velocidad fijada por emisor y receptor.
- **Bits de datos (*Data*s)**. Los bits de datos son enviados al receptor después del bit de *Start*. El bit de menos peso LSB (*Least Significant Bit*) es transmitido primero y el de mayor peso MSB (*Most Significant Bit*) el último. Un carácter de datos suele consistir en 7 u 8 bits. En el ejemplo de la figura 20-4 trabaja con 7 bits.
- **Bit de paridad (*Parity*)**. Dependiendo de la configuración de la transmisión un bit de paridad puede ser enviado después de los bits de datos. En aplicaciones sencillas no suele utilizarse. Con este bit se pueden descubrir errores en la transmisión. Se puede dar paridad par o impar. En la paridad par, por ejemplo, la palabra de datos a transmitir se completa con el bit de paridad de manera que el número de bits "1" enviados sea par, como en el ejemplo de la figura 20-4, donde la información del byte de datos más paridad es b'01010110' (cuatro "1").
- **Bit de parada (*STOP*)**. La línea queda a -12 V después del último bit enviado, es decir, queda a "1" en la lógica negativa de la norma RS232. Indica la finalización de la transmisión de una palabra de datos. El protocolo de transmisión de datos permite 1, 1.5 ó 2 bits de parada.

En el ejemplo de la figura anterior se envía un bit de inicio, una palabra de 7 bits (101 0110), que corresponde a la letra "V" en código ASCII, un bit de paridad par y luego dos bits de paro.

Es importante hacer observar, que el microcontrolador trabaja con lógica positiva a diferencia del puerto RS232 del ordenador que trabaja con lógica negativa. Por tanto la forma de la señal en el microcontrolador (niveles TTL) será inversa a los niveles RS232 tal como se muestra en la figura 20-4.

## 20.5 MAX232

En el mercado hay muchos circuitos integrados que permiten la conversión entre niveles TTL y niveles RS232. Entre ellos destaca el transceptor MAX232 (figura 20-5), fabricado por *Dallas Semiconductor-MAXIM*, [www.dalsemi.com](http://www.dalsemi.com).

El MAX232 convierte los niveles RS232 (cerca de +12 y -12 V) a voltajes TTL (0 a +5 V) y viceversa sin requerir nada más que una fuente de +5 V. El chip contiene dos drivers TTL → RS232 y dos RS232 → TTL. Necesita cuatro condensadores externos de

unos pocos microfaradios para generar el voltaje RS232 internamente tal como se muestra en las figuras 20-5 y 20-6.

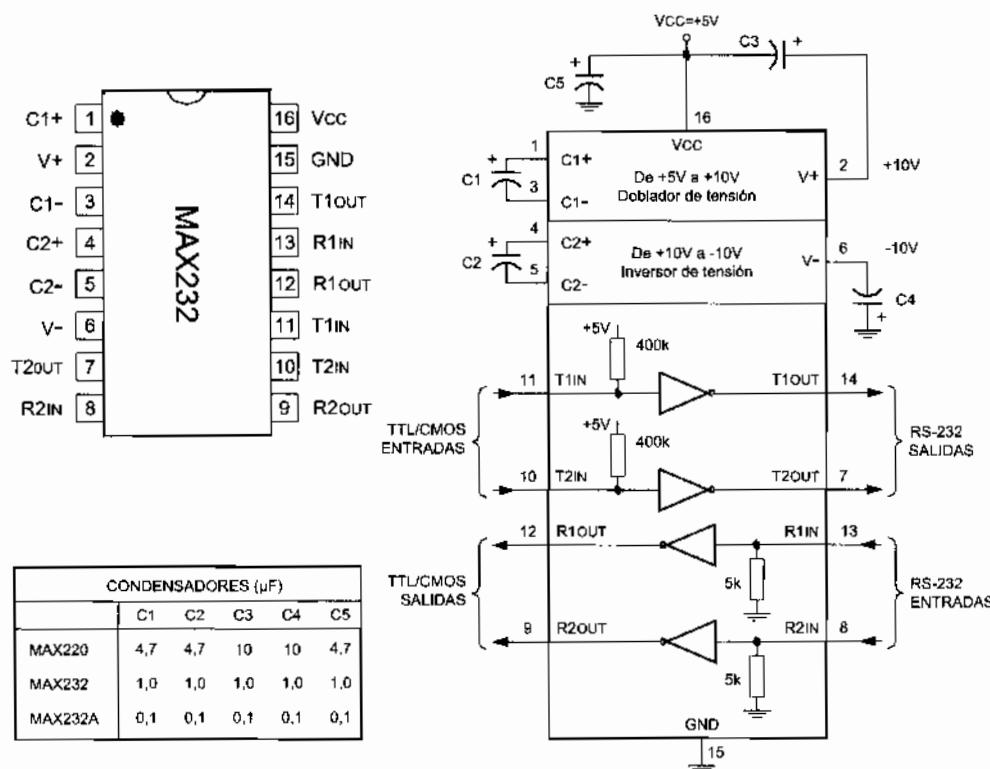


Figura 20-5 MAX232

## 20.6 CONEXIÓN PUERTO RS232 Y PIC16F84

Para comunicarse con un microcontrolador se pueden utilizar sólo tres patillas del puerto serie RS232. Éstas son:

- Pin de transmisión (Tx).
- Pin de recepción (Rx).
- Pin de masa (SG).

La figura 20-7 muestra un ejemplo de circuito para comunicar un ordenador y un microcontrolador PIC16F84A a través del puerto serie RS232.

El primer problema a resolver es que los niveles lógicos TTL que salen del microcontrolador no son compatibles con los niveles lógicos RS232 del puerto serie del ordenador, razón por la cual se debe introducir en el circuito un interface que traduzca los

datos del microcontrolador a niveles RS232.

En el esquema se muestran cuatro disponibles:

- La línea de datos informa los niveles para conectar a la terminal 2 de la patilla RxD.
- Del mismo modo, en la terminal 3 se conecta la línea de datos recibida DB9.

Hay que tener en cuenta que los condensadores electrolíticos al lado de la terminal 2 y 3.

Para el correcto funcionamiento es necesario el siguiente circuito:

- Unas resistencias de 1 k $\Omega$  se deben utilizar en las líneas de datos.
- Un programa de HyperTerminal se debe utilizar para configurar el puerto serie.

tal como se muestra

datos del microcontrolador al puerto y viceversa. Este chip puente puede ser el MAX232 o alguno similar.

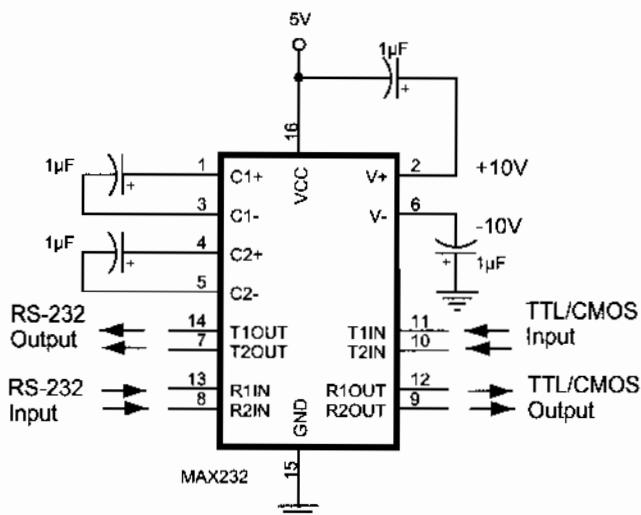
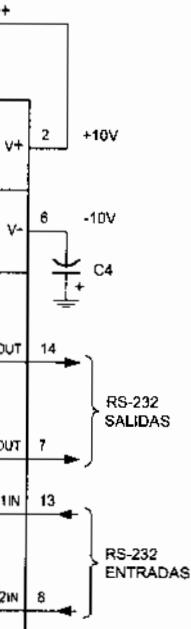


Figura 20-6 Conexión típica del MAX232

En el esquema de la figura 20-7 se aprecia como se utiliza un par de drivers de los cuatro disponibles.

- La línea TxD (pin 3 del conector DB9) del puerto serie RS232 transmite información con niveles RS232, por tanto se conecta al pin R1IN del MAX232 para convertir estos niveles a TTL y transmitírselos al PIC16F84A a través de la patilla R1OUT.
- Del mismo modo la información que envía el PIC16F84A con niveles TTL entra en la línea T1IN del MAX232 para convertirla en niveles RS232 y poder ser recibida por el puerto serie RS232 a través de su línea RxD, pin 2 del conector DB9.

Hay que tener cuidado para no equivocarse con la polaridad de los condensadores electrolíticos al conectarlos en el circuito.

Para el control de la comunicación del ordenador con el microcontrolador es necesario el siguiente software:

- Unas subrutinas que gestionen el software del microcontrolador, como las utilizadas en la librería RS232.INC que se explicará en la próxima sección.
- Un programa para el ordenador que gestione el control del sistema, tal como el *HyperTerminal* de Windows o similar.

A continuación se explican ambos.

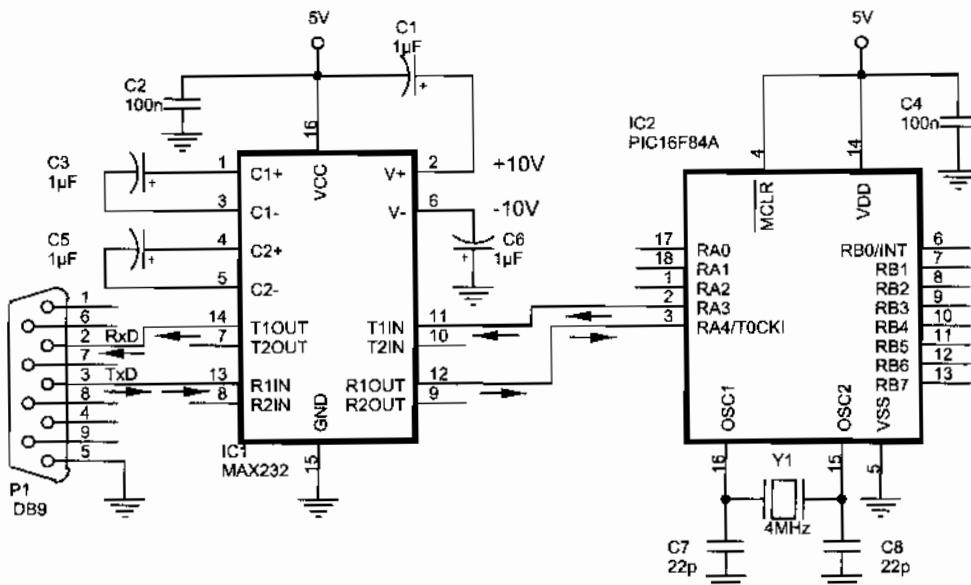


Figura 20-7 Conexión del MAX232 a un PIC16F84A

## 20.7 LIBRERÍA DE SUBRUTINAS PARA RS232

La siguiente librería denominada RS232.INC recoge unas subrutinas de control muy sencillas para la comunicación de un PIC16F84 con un ordenador. Estas subrutinas permiten realizar las tareas básicas de comunicación con el puerto serie RS232 conectado según el esquema de la figura 20-7. Las subrutinas principales son:

- "RS232\_Inicializa", configura las líneas de salida y entrada del PIC.
- "RS232\_LeeDatos", el microcontrolador lee el dato por la línea de entrada.
- "RS232\_EnviaDatos", el microcontrolador envía un dato por la línea de salida.

Esta librería está suficientemente documentada y se detalla a continuación:

```
***** Librería "RS232.INC" *****
;
; Estas subrutinas permiten realizar las tareas básicas de control de la transmisión
; serie asíncrona según normas RS-232.
;
; Los parámetros adoptados para la comunicación son los siguientes:
; - Velocidad de transmisión de 9600 baudios. La duración de cada bit será 104 µs.
; - Un bit de inicio o Start a nivel bajo.
; - Dato de 8 bits.
; - Sin paridad.
; - Dos bits de final o Stop a nivel alto.
```

El tiempo entre bit y bit debe ser de 104 µs. Como la velocidad de transmisión es de 9600 Baudios ≈ 104 µs. Se cumple.

CBLOCK  
RS232\_CodificadorBit  
RS232\_Dato  
ENDC

#DEFINE RS232\_Entrada  
#DEFINE RS232\_Salida

Subrutina "RS232\_Inicializa"

Configura las líneas de salida

```
RS232_Inicializa
    bsf    STATUS, RS232_E
    bcf    RS232_E
    bcf    RS232_S
    bcf    STATUS, RS232_D
    return
```

Subrutina "RS232\_LeeDatos"

El microcontrolador lee el dato por la línea de entrada. El dato leído se envía al ordenador.

El ordenador parte siempre de que el microcontrolador envía información. La secuencia es la siguiente:

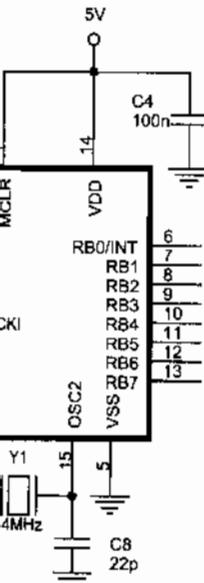
- 1º Espera que se ejecute la subrutina.
- 2º Deja pasar un tiempo y saltarse el bit de Start.
- 3º Lee el resto de los bits.

Salida: En el registro de trabajo.

RS232\_LeeDatos
 movlw d'8
 movwf RS232\_C

RS232\_EspereBitStart
 bfc RS232\_E
 goto RS232\_E
 call Retardo\_1ms
 call Retardo\_1ms

RS232\_LeeBit
 bcf RS232\_E
 bfc RS232\_E
 bcf STATUS, RS232\_S
 rrf RS232\_D
 call Retardo\_1ms



; El tiempo entre bit y bit debe coincidir con el periodo de la señal leída o enviada.  
; Como la velocidad de transmisión o recepción es de 9600 baudios, el periodo será:  
; 1/9600 Baudios = 104 µs. Se utilizará pues la subrutina Retardos\_100micros.

```
CBLOCK
RS232_ContadorBits
RS232_Dato
ENDC
```

```
#DEFINE RS232_EstablecerEntrada PORTA,3 ; Línea por la que se reciben los datos.
#DEFINE RS232_EstablecerSalida PORTA,4 ; Línea por la que se envían los datos.
```

; Subrutina "RS232\_Inicializa"

; Configura las líneas de salida y entrada del microcontrolador.

RS232\_Inicializa

```
bsf STATUS,RP0
bsf RS232_EstablecerEntrada ; Esta línea se configura como entrada.
bcf RS232_EstablecerSalida ; Esta línea se configura como salida.
bcf STATUS,RP0
return
```

; Subrutina "RS232\_LeeDatos"

; El microcontrolador lee el dato por la línea de entrada comenzando por el bit de menor peso. El dato leído se envía finalmente en el registro de trabajo W.

; El ordenador parte siempre de un nivel alto, que es el estado que tiene cuando no envía información. La secuencia utilizada es:

- ; 1º Espera que se ejecute el pulso negativo del bit Start o flanco de bajada.
- ; 2º Deja pasar un tiempo una y media veces mayor que el periodo de transmisión para saltarse el bit de Start y lee el primer bit en su mitad.
- ; 3º Lee el resto de los bits de datos, esperando un tiempo igual a la duración del periodo entre lectura y lectura para testearlos en mitad del bit.

; Salida: En el registro de trabajo W el byte leído.

RS232\_LeeDatos

```
movlw d'8' ; Número de bits a recibir.
movwf RS232_ContadorBits
```

RS232\_EspereBitStart

```
btfsC RS232_EstablecerEntrada ; Lee la entrada y espera a que sea "0".
goto RS232_EspereBitStart ; No, pues espera el nivel bajo.
call Retardo_100micros ; El primer bit debe leerlo un tiempo igual a una vez y media el periodo de transmisión.
```

RS232\_LeeBit

```
bcf STATUS,C ; Ahora lee el pin. En principio supone que es 0.
btfsC RS232_EstablecerEntrada ; ¿Realmente es cero?
bsf STATUS,C ; No, pues cambia a "1".
rrf RS232_Dato,F ; Introduce el bit en el registro de lectura.
call Retardo_100micros ; Los siguientes bits los lee un periodo más tarde.
```

```

decfsz RS232_ContadorBits,F ; Comprueba que es el último bit.
goto RS232_LeeBit ; Si no es el último bit pasa a leer el siguiente.
call Retardo_200micros ; Espera un tiempo igual al los 2 bits de Stop.
movf RS232_Dato,W ; El resultado en el registro W.
return

```

; Subrutinas "RS232\_EnviaDato" y "RS232\_EnviaNúmero" -----

; El microcontrolador envía un dato por la línea de salida comenzando por el bit de menor peso. En dato enviado será el que le llegue a través del registro de trabajo W.

- ; 1º Envía un "0" durante un tiempo igual al periodo de la velocidad de transmisión. Este es el bit de "Start".
- ; 2º Envía el bit correspondiente:
  - Si va a enviar un "0" permanece en bajo durante el periodo correspondiente.
  - Si va a escribir un "1" permanece en alto durante el periodo correspondiente.
- ; 3º Envía dos bits "1" durante un tiempo igual al periodo de la velocidad de transmisión cada uno. Estos son los dos bits de Stop.

; Entrada: En (W) el dato a enviar.

RS232_EnviaNúmero		;
addlw '0'		Envía el código ASCII de un número.
RS232_EnviaDato		;
movwf RS232_Dato		Lo pasa a código ASCII sumándole el ASCII del 0.
movlw d'8'		
movwf RS232_ContadorBits		;
bcf RS232_Salida		Guarda el contenido del byte a transmitir.
call Retardo_100micros		;
RS232_EnviaBit		Éste es el número de bits a transmitir.
rrf RS232_Dato,F		;
btfs STATUS,C		Bit de Start.
goto RS232_EnviaCero		;
RS232_EnviaUno		Comienza a enviar datos.
bsf RS232_Salida		;
goto RS232_FinEnviaBit		Lleva el bit que se quiere enviar al Carry para deducir su valor. ¿Es un "1" el bit a transmitir?
RS232_EnviaCero		;
bcf RS232_Salida		No, pues envía un "0".
RS232_FinEnviaBit		;
call Retardo_100micros		Transmite un "1".
decfsz RS232_ContadorBits,F		;
goto RS232_EnviaBit		Transmite un "0".
bsf RS232_Salida		;
call Retardo_200micros		Éste es el tiempo que estará en alto o bajo.
return		Comprueba que es el último bit.
		;
		Como no es el último bit repite la operación.
		Envía dos bits de Stop.

El programa necesario para gestionar los datos con el ordenador puede ser uno cualquiera que controle el puerto serie. Uno muy común es el *HyperTerminal* de Windows que es muy sencillo de utilizar, aunque los hay mejores.

Es importante resaltar que la configuración en el formato de transmisión y recepción debe ser igual para el programa de comunicaciones del ordenador que para las

subrutinas utilizadas por la configuración:

- Velocidad de transmisión
- Bits de datos: 8.
- Paridad: Ninguna.
- Bits de parada: 1.

## 20.8 EL HYPERTERMINAL

*HyperTerminal* es un programa de comunicación resulta válido para conectar el ordenador con comunicaciones mucho más rápidas que el serial, por su facilidad de uso y configuración.

Puede ocurrir que el ordenador no siga el mismo procedimiento en el siguiente orden: *Windows > Comunicaciones > Puerto serie*.

Para ejecutar *HyperTerminal* debemos:

- 1º Dentro de Windows, pulsamos sobre el icono de *Comunicaciones*.
- 2º Se abre la ventana de *Comunicaciones*. Allí elegimos el puerto serie que queremos utilizar.
- 3º Se abre la ventana de *Configuración de puerto serie*.
- 4º Se configura el puerto serie.
- 5º Se pulsa el botón *Aplicar*.
- 6º Se pulsa el botón *OK*.
- 7º Se pulsa el botón *Aplicar*.
- 8º Se pulsa el botón *OK*.
- 9º Se pulsa el botón *Aplicar*.
- 10º Se pulsa el botón *OK*.
- 11º Se pulsa el botón *Aplicar*.
- 12º Se pulsa el botón *OK*.
- 13º Se pulsa el botón *Aplicar*.
- 14º Se pulsa el botón *OK*.
- 15º Se pulsa el botón *Aplicar*.
- 16º Se pulsa el botón *OK*.
- 17º Se pulsa el botón *Aplicar*.
- 18º Se pulsa el botón *OK*.
- 19º Se pulsa el botón *Aplicar*.
- 20º Se pulsa el botón *OK*.
- 21º Se pulsa el botón *Aplicar*.
- 22º Se pulsa el botón *OK*.
- 23º Se pulsa el botón *Aplicar*.
- 24º Se pulsa el botón *OK*.
- 25º Se pulsa el botón *Aplicar*.
- 26º Se pulsa el botón *OK*.
- 27º Se pulsa el botón *Aplicar*.
- 28º Se pulsa el botón *OK*.
- 29º Se pulsa el botón *Aplicar*.
- 30º Se pulsa el botón *OK*.
- 31º Se pulsa el botón *Aplicar*.
- 32º Se pulsa el botón *OK*.
- 33º Se pulsa el botón *Aplicar*.
- 34º Se pulsa el botón *OK*.
- 35º Se pulsa el botón *Aplicar*.
- 36º Se pulsa el botón *OK*.
- 37º Se pulsa el botón *Aplicar*.
- 38º Se pulsa el botón *OK*.
- 39º Se pulsa el botón *Aplicar*.
- 40º Se pulsa el botón *OK*.
- 41º Se pulsa el botón *Aplicar*.
- 42º Se pulsa el botón *OK*.
- 43º Se pulsa el botón *Aplicar*.
- 44º Se pulsa el botón *OK*.
- 45º Se pulsa el botón *Aplicar*.
- 46º Se pulsa el botón *OK*.
- 47º Se pulsa el botón *Aplicar*.
- 48º Se pulsa el botón *OK*.
- 49º Se pulsa el botón *Aplicar*.
- 50º Se pulsa el botón *OK*.
- 51º Se pulsa el botón *Aplicar*.
- 52º Se pulsa el botón *OK*.
- 53º Se pulsa el botón *Aplicar*.
- 54º Se pulsa el botón *OK*.
- 55º Se pulsa el botón *Aplicar*.
- 56º Se pulsa el botón *OK*.
- 57º Se pulsa el botón *Aplicar*.
- 58º Se pulsa el botón *OK*.
- 59º Se pulsa el botón *Aplicar*.
- 60º Se pulsa el botón *OK*.
- 61º Se pulsa el botón *Aplicar*.
- 62º Se pulsa el botón *OK*.
- 63º Se pulsa el botón *Aplicar*.
- 64º Se pulsa el botón *OK*.
- 65º Se pulsa el botón *Aplicar*.
- 66º Se pulsa el botón *OK*.
- 67º Se pulsa el botón *Aplicar*.
- 68º Se pulsa el botón *OK*.
- 69º Se pulsa el botón *Aplicar*.
- 70º Se pulsa el botón *OK*.
- 71º Se pulsa el botón *Aplicar*.
- 72º Se pulsa el botón *OK*.
- 73º Se pulsa el botón *Aplicar*.
- 74º Se pulsa el botón *OK*.
- 75º Se pulsa el botón *Aplicar*.
- 76º Se pulsa el botón *OK*.
- 77º Se pulsa el botón *Aplicar*.
- 78º Se pulsa el botón *OK*.
- 79º Se pulsa el botón *Aplicar*.
- 80º Se pulsa el botón *OK*.
- 81º Se pulsa el botón *Aplicar*.
- 82º Se pulsa el botón *OK*.
- 83º Se pulsa el botón *Aplicar*.
- 84º Se pulsa el botón *OK*.
- 85º Se pulsa el botón *Aplicar*.
- 86º Se pulsa el botón *OK*.
- 87º Se pulsa el botón *Aplicar*.
- 88º Se pulsa el botón *OK*.
- 89º Se pulsa el botón *Aplicar*.
- 90º Se pulsa el botón *OK*.
- 91º Se pulsa el botón *Aplicar*.
- 92º Se pulsa el botón *OK*.
- 93º Se pulsa el botón *Aplicar*.
- 94º Se pulsa el botón *OK*.
- 95º Se pulsa el botón *Aplicar*.
- 96º Se pulsa el botón *OK*.
- 97º Se pulsa el botón *Aplicar*.
- 98º Se pulsa el botón *OK*.
- 99º Se pulsa el botón *Aplicar*.
- 100º Se pulsa el botón *OK*.
- 101º Se pulsa el botón *Aplicar*.
- 102º Se pulsa el botón *OK*.
- 103º Se pulsa el botón *Aplicar*.
- 104º Se pulsa el botón *OK*.
- 105º Se pulsa el botón *Aplicar*.
- 106º Se pulsa el botón *OK*.
- 107º Se pulsa el botón *Aplicar*.
- 108º Se pulsa el botón *OK*.
- 109º Se pulsa el botón *Aplicar*.
- 110º Se pulsa el botón *OK*.
- 111º Se pulsa el botón *Aplicar*.
- 112º Se pulsa el botón *OK*.
- 113º Se pulsa el botón *Aplicar*.
- 114º Se pulsa el botón *OK*.
- 115º Se pulsa el botón *Aplicar*.
- 116º Se pulsa el botón *OK*.
- 117º Se pulsa el botón *Aplicar*.
- 118º Se pulsa el botón *OK*.
- 119º Se pulsa el botón *Aplicar*.
- 120º Se pulsa el botón *OK*.
- 121º Se pulsa el botón *Aplicar*.
- 122º Se pulsa el botón *OK*.
- 123º Se pulsa el botón *Aplicar*.
- 124º Se pulsa el botón *OK*.
- 125º Se pulsa el botón *Aplicar*.
- 126º Se pulsa el botón *OK*.
- 127º Se pulsa el botón *Aplicar*.
- 128º Se pulsa el botón *OK*.
- 129º Se pulsa el botón *Aplicar*.
- 130º Se pulsa el botón *OK*.
- 131º Se pulsa el botón *Aplicar*.
- 132º Se pulsa el botón *OK*.
- 133º Se pulsa el botón *Aplicar*.
- 134º Se pulsa el botón *OK*.
- 135º Se pulsa el botón *Aplicar*.
- 136º Se pulsa el botón *OK*.
- 137º Se pulsa el botón *Aplicar*.
- 138º Se pulsa el botón *OK*.
- 139º Se pulsa el botón *Aplicar*.
- 140º Se pulsa el botón *OK*.
- 141º Se pulsa el botón *Aplicar*.
- 142º Se pulsa el botón *OK*.
- 143º Se pulsa el botón *Aplicar*.
- 144º Se pulsa el botón *OK*.
- 145º Se pulsa el botón *Aplicar*.
- 146º Se pulsa el botón *OK*.
- 147º Se pulsa el botón *Aplicar*.
- 148º Se pulsa el botón *OK*.
- 149º Se pulsa el botón *Aplicar*.
- 150º Se pulsa el botón *OK*.
- 151º Se pulsa el botón *Aplicar*.
- 152º Se pulsa el botón *OK*.
- 153º Se pulsa el botón *Aplicar*.
- 154º Se pulsa el botón *OK*.
- 155º Se pulsa el botón *Aplicar*.
- 156º Se pulsa el botón *OK*.
- 157º Se pulsa el botón *Aplicar*.
- 158º Se pulsa el botón *OK*.
- 159º Se pulsa el botón *Aplicar*.
- 160º Se pulsa el botón *OK*.
- 161º Se pulsa el botón *Aplicar*.
- 162º Se pulsa el botón *OK*.
- 163º Se pulsa el botón *Aplicar*.
- 164º Se pulsa el botón *OK*.
- 165º Se pulsa el botón *Aplicar*.
- 166º Se pulsa el botón *OK*.
- 167º Se pulsa el botón *Aplicar*.
- 168º Se pulsa el botón *OK*.
- 169º Se pulsa el botón *Aplicar*.
- 170º Se pulsa el botón *OK*.
- 171º Se pulsa el botón *Aplicar*.
- 172º Se pulsa el botón *OK*.
- 173º Se pulsa el botón *Aplicar*.
- 174º Se pulsa el botón *OK*.
- 175º Se pulsa el botón *Aplicar*.
- 176º Se pulsa el botón *OK*.
- 177º Se pulsa el botón *Aplicar*.
- 178º Se pulsa el botón *OK*.
- 179º Se pulsa el botón *Aplicar*.
- 180º Se pulsa el botón *OK*.
- 181º Se pulsa el botón *Aplicar*.
- 182º Se pulsa el botón *OK*.
- 183º Se pulsa el botón *Aplicar*.
- 184º Se pulsa el botón *OK*.
- 185º Se pulsa el botón *Aplicar*.
- 186º Se pulsa el botón *OK*.
- 187º Se pulsa el botón *Aplicar*.
- 188º Se pulsa el botón *OK*.
- 189º Se pulsa el botón *Aplicar*.
- 190º Se pulsa el botón *OK*.
- 191º Se pulsa el botón *Aplicar*.
- 192º Se pulsa el botón *OK*.
- 193º Se pulsa el botón *Aplicar*.
- 194º Se pulsa el botón *OK*.
- 195º Se pulsa el botón *Aplicar*.
- 196º Se pulsa el botón *OK*.
- 197º Se pulsa el botón *Aplicar*.
- 198º Se pulsa el botón *OK*.
- 199º Se pulsa el botón *Aplicar*.
- 200º Se pulsa el botón *OK*.
- 201º Se pulsa el botón *Aplicar*.
- 202º Se pulsa el botón *OK*.
- 203º Se pulsa el botón *Aplicar*.
- 204º Se pulsa el botón *OK*.
- 205º Se pulsa el botón *Aplicar*.
- 206º Se pulsa el botón *OK*.
- 207º Se pulsa el botón *Aplicar*.
- 208º Se pulsa el botón *OK*.
- 209º Se pulsa el botón *Aplicar*.
- 210º Se pulsa el botón *OK*.
- 211º Se pulsa el botón *Aplicar*.
- 212º Se pulsa el botón *OK*.
- 213º Se pulsa el botón *Aplicar*.
- 214º Se pulsa el botón *OK*.
- 215º Se pulsa el botón *Aplicar*.
- 216º Se pulsa el botón *OK*.
- 217º Se pulsa el botón *Aplicar*.
- 218º Se pulsa el botón *OK*.
- 219º Se pulsa el botón *Aplicar*.
- 220º Se pulsa el botón *OK*.
- 221º Se pulsa el botón *Aplicar*.
- 222º Se pulsa el botón *OK*.
- 223º Se pulsa el botón *Aplicar*.
- 224º Se pulsa el botón *OK*.
- 225º Se pulsa el botón *Aplicar*.
- 226º Se pulsa el botón *OK*.
- 227º Se pulsa el botón *Aplicar*.
- 228º Se pulsa el botón *OK*.
- 229º Se pulsa el botón *Aplicar*.
- 230º Se pulsa el botón *OK*.
- 231º Se pulsa el botón *Aplicar*.
- 232º Se pulsa el botón *OK*.
- 233º Se pulsa el botón *Aplicar*.
- 234º Se pulsa el botón *OK*.
- 235º Se pulsa el botón *Aplicar*.
- 236º Se pulsa el botón *OK*.
- 237º Se pulsa el botón *Aplicar*.
- 238º Se pulsa el botón *OK*.
- 239º Se pulsa el botón *Aplicar*.
- 240º Se pulsa el botón *OK*.
- 241º Se pulsa el botón *Aplicar*.
- 242º Se pulsa el botón *OK*.
- 243º Se pulsa el botón *Aplicar*.
- 244º Se pulsa el botón *OK*.
- 245º Se pulsa el botón *Aplicar*.
- 246º Se pulsa el botón *OK*.
- 247º Se pulsa el botón *Aplicar*.
- 248º Se pulsa el botón *OK*.
- 249º Se pulsa el botón *Aplicar*.
- 250º Se pulsa el botón *OK*.
- 251º Se pulsa el botón *Aplicar*.
- 252º Se pulsa el botón *OK*.
- 253º Se pulsa el botón *Aplicar*.
- 254º Se pulsa el botón *OK*.
- 255º Se pulsa el botón *Aplicar*.
- 256º Se pulsa el botón *OK*.
- 257º Se pulsa el botón *Aplicar*.
- 258º Se pulsa el botón *OK*.
- 259º Se pulsa el botón *Aplicar*.
- 260º Se pulsa el botón *OK*.
- 261º Se pulsa el botón *Aplicar*.
- 262º Se pulsa el botón *OK*.
- 263º Se pulsa el botón *Aplicar*.
- 264º Se pulsa el botón *OK*.
- 265º Se pulsa el botón *Aplicar*.
- 266º Se pulsa el botón *OK*.
- 267º Se pulsa el botón *Aplicar*.
- 268º Se pulsa el botón *OK*.
- 269º Se pulsa el botón *Aplicar*.
- 270º Se pulsa el botón *OK*.
- 271º Se pulsa el botón *Aplicar*.
- 272º Se pulsa el botón *OK*.
- 273º Se pulsa el botón *Aplicar*.
- 274º Se pulsa el botón *OK*.
- 275º Se pulsa el botón *Aplicar*.
- 276º Se pulsa el botón *OK*.
- 277º Se pulsa el botón *Aplicar*.
- 278º Se pulsa el botón *OK*.
- 279º Se pulsa el botón *Aplicar*.
- 280º Se pulsa el botón *OK*.
- 281º Se pulsa el botón *Aplicar*.
- 282º Se pulsa el botón *OK*.
- 283º Se pulsa el botón *Aplicar*.
- 284º Se pulsa el botón *OK*.
- 285º Se pulsa el botón *Aplicar*.
- 286º Se pulsa el botón *OK*.
- 287º Se pulsa el botón *Aplicar*.
- 288º Se pulsa el botón *OK*.
- 289º Se pulsa el botón *Aplicar*.
- 290º Se pulsa el botón *OK*.
- 291º Se pulsa el botón *Aplicar*.
- 292º Se pulsa el botón *OK*.
- 293º Se pulsa el botón *Aplicar*.
- 294º Se pulsa el botón *OK*.
- 295º Se pulsa el botón *Aplicar*.
- 296º Se pulsa el botón *OK*.
- 297º Se pulsa el botón *Aplicar*.
- 298º Se pulsa el botón *OK*.
- 299º Se pulsa el botón *Aplicar*.
- 300º Se pulsa el botón *OK*.
- 301º Se pulsa el botón *Aplicar*.
- 302º Se pulsa el botón *OK*.
- 303º Se pulsa el botón *Aplicar*.
- 304º Se pulsa el botón *OK*.
- 305º Se pulsa el botón *Aplicar*.
- 306º Se pulsa el botón *OK*.
- 307º Se pulsa el botón *Aplicar*.
- 308º Se pulsa el botón *OK*.
- 309º Se pulsa el botón *Aplicar*.
- 310º Se pulsa el botón *OK*.
- 311º Se pulsa el botón *Aplicar*.
- 312º Se pulsa el botón *OK*.
- 313º Se pulsa el botón *Aplicar*.
- 314º Se pulsa el botón *OK*.
- 315º Se pulsa el botón *Aplicar*.
- 316º Se pulsa el botón *OK*.
- 317º Se pulsa el botón *Aplicar*.
- 318º Se pulsa el botón *OK*.
- 319º Se pulsa el botón *Aplicar*.
- 320º Se pulsa el botón *OK*.
- 321º Se pulsa el botón *Aplicar*.
- 322º Se pulsa el botón *OK*.
- 323º Se pulsa el botón *Aplicar*.
- 324º Se pulsa el botón *OK*.
- 325º Se pulsa el botón *Aplicar*.
- 326º Se pulsa el botón *OK*.
- 327º Se pulsa el botón *Aplicar*.
- 328º Se pulsa el botón *OK*.
- 329º Se pulsa el botón *Aplicar*.
- 330º Se pulsa el botón *OK*.
- 331º Se pulsa el botón *Aplicar*.
- 332º Se pulsa el botón *OK*.
- 333º Se pulsa el botón *Aplicar*.
- 334º Se pulsa el botón *OK*.
- 335º Se pulsa el botón *Aplicar*.
- 336º Se pulsa el botón *OK*.
- 337º Se pulsa el botón *Aplicar*.
- 338º Se pulsa el botón *OK*.
- 339º Se pulsa el botón *Aplicar*.
- 340º Se pulsa el botón *OK*.
- 341º Se pulsa el botón *Aplicar*.
- 342º Se pulsa el botón *OK*.
- 343º Se pulsa el botón *Aplicar*.
- 344º Se pulsa el botón *OK*.
- 345º Se pulsa el botón *Aplicar*.
- 346º Se pulsa el botón *OK*.
- 347º Se pulsa el botón *Aplicar*.
- 348º Se pulsa el botón *OK*.
- 349º Se pulsa el botón *Aplicar*.
- 350º Se pulsa el botón *OK*.
- 351º Se pulsa el botón *Aplicar*.
- 352º Se pulsa el botón *OK*.
- 353º Se pulsa el botón *Aplicar*.
- 354º Se pulsa el botón *OK*.
- 355º Se pulsa el botón *Aplicar*.
- 356º Se pulsa el botón *OK*.
- 357º Se pulsa el botón *Aplicar*.
- 358º Se pulsa el botón *OK*.
- 359º Se pulsa el botón *Aplicar*.
- 360º Se pulsa el botón *OK*.
- 361º Se pulsa el botón *Aplicar*.
- 362º Se pulsa el botón *OK*.
- 363º Se pulsa el botón *Aplicar*.
- 364º Se pulsa el botón *OK*.
- 365º Se pulsa el botón *Aplicar*.
- 366º Se pulsa el botón *OK*.
- 367º Se pulsa el botón *Aplicar*.
- 368º Se pulsa el botón *OK*.
- 369º Se pulsa el botón *Aplicar*.
- 370º Se pulsa el botón *OK*.
- 371º Se pulsa el botón *Aplicar*.
- 372º Se pulsa el botón *OK*.
- 373º Se pulsa el botón *Aplicar*.
- 374º Se pulsa el botón *OK*.
- 375º Se pulsa el botón *Aplicar*.
- 376º Se pulsa el botón *OK*.
- 377º Se pulsa el botón *Aplicar*.
- 378º Se pulsa el botón *OK*.
- 379º Se pulsa el botón *Aplicar*.
- 380º Se pulsa el botón *OK*.
- 381º Se pulsa el botón *Aplicar*.
- 382º Se pulsa el botón *OK*.
- 383º Se pulsa el botón *Aplicar*.
- 384º Se pulsa el botón *OK*.
- 385º Se pulsa el botón *Aplicar*.
- 386º Se pulsa el botón *OK*.
- 387º Se pulsa el botón *Aplicar*.
- 388º Se pulsa el botón *OK*.
- 389º Se pulsa el botón *Aplicar*.
- 390º Se pulsa el botón *OK*.
- 391º Se pulsa el botón *Aplicar*.
- 392º Se pulsa el botón *OK*.
- 393º Se pulsa el botón *Aplicar*.
- 394º Se pulsa el botón *OK*.
- 395º Se pulsa el botón *Aplicar*.
- 396º Se pulsa el botón *OK*.
- 397º Se pulsa el botón *Aplicar*.
- 398º Se pulsa el botón *OK*.
- 399º Se pulsa el botón *Aplicar*.
- 400º Se pulsa el botón *OK*.
- 401º Se pulsa el botón *Aplicar*.
- 402º Se pulsa el botón *OK*.
- 403º Se pulsa el botón *Aplicar*.
- 404º Se pulsa el botón *OK*.
- 405º Se pulsa el botón *Aplicar*.
- 406º Se pulsa el botón *OK*.
- 407º Se pulsa el botón *Aplicar*.
- 408º Se pulsa el botón *OK*.
- 409º Se pulsa el botón *Aplicar*.
- 410º Se pulsa el botón *OK*.
- 411º Se pulsa el botón *Aplicar*.
- 412º Se pulsa el botón *OK*.
- 413º Se pulsa el botón *Aplicar*.
- 414º Se pulsa el botón *OK*.
- 415º Se pulsa el botón *Aplicar*.
- 416º Se pulsa el botón *OK*.
- 417º Se pulsa el botón *Aplicar*.
- 418º Se pulsa el botón *OK*.
- 419º Se pulsa el botón *Aplicar*.
- 420º Se pulsa el botón *OK*.
- 421º Se pulsa el botón *Aplicar*.
- 422º Se pulsa el botón *OK*.
- 423º Se pulsa el botón *Aplicar*.
- 424º Se pulsa el botón *OK*.
- 425º Se pulsa el botón *Aplicar*.
- 426º Se pulsa el botón *OK*.
- 427º Se pulsa el botón *Aplicar*.
- 428º Se pulsa el botón *OK*.
- 429º Se pulsa el botón *Aplicar*.
- 430º Se pulsa el botón *OK*.
- 431º Se pulsa el botón *Aplicar*.</

subrutinas utilizadas por el microcontrolador. En nuestro caso utilizaremos la siguiente configuración:

- Velocidad de transmisión: 9600 baudios.
- Bits de datos: 8.
- Paridad: Ninguna.
- Bits de parada: 1.

## 20.8 EL HYPERTERMINAL

*HyperTerminal* es un programa general de comunicaciones de Windows que resulta válido para conectarse con otros ordenadores o dispositivos. Hay programas de comunicaciones mucho más potentes que éste, sin embargo, *HyperTerminal* se caracteriza por su facilidad de uso y su fácil adquisición, ya que viene integrado dentro de Windows.

Puede ocurrir que el programa *HyperTerminal* no esté cargado, para ello hay que seguir el mismo procedimiento que para cualquier otro programa de Windows, ejecutando en el siguiente orden: *Panel de Control > Agregar o Quitar programas > Instalación de Windows > Comunicaciones* y activar la casilla correspondiente a *HyperTerminal*.

Para ejecutar *HyperTerminal* hay que seguir los siguientes pasos:

- 1º Dentro de Windows hay que activar *Inicio > Programas > Accesorios > Comunicaciones* y seleccionar *HyperTerminal*. Se advierte de que no se trata de un programa sino de una carpeta cuyo contenido es un único programa denominada *Hypertrm*. Cada vez que se ejecuta el programa *Hypertrm* se le pide al usuario la información suficiente para crear en la carpeta *HyperTerminal* una nueva conexión, es decir, un lugar de destino con el que conectarse.

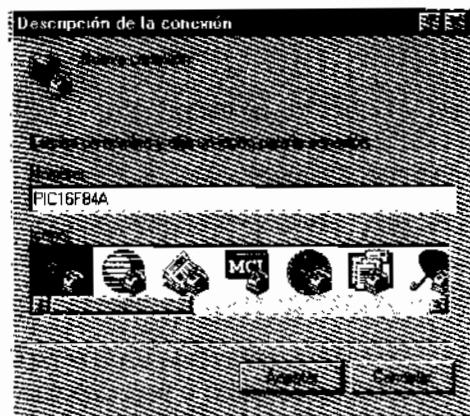


Figura 20-8 *HyperTerminal*

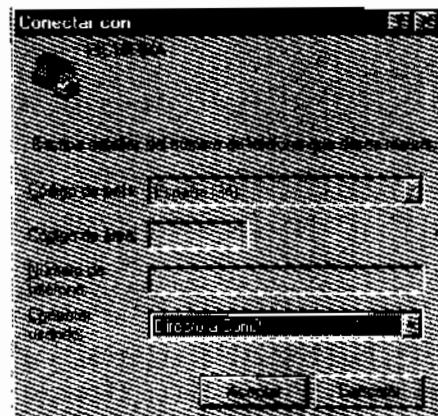


Figura 20-9 *Conectar con...*

- 2º Ejecutar el programa *Hyperterm* para crear una nueva conexión, para ésto se le pregunta el nombre que asignará a la conexión, se teclea por ejemplo "PIC16F84A" y se elige un ícono con el que se quiera representar la conexión (figura 20-8).
- 3º En la siguiente pantalla *Conecta con* se selecciona la opción *Conectar usando: Directo a Com2* (figura 20-9). Se va a utilizar el puerto serie COM 2 para comunicar el ordenador y el microcontrolador. También se puede elegir el COM 1 si le es más conveniente al lector.
- 4º Aparece una pantalla con las *Propiedades de COM2* donde se establecen las apropiadas para el sistema utilizado. En nuestro caso hay que elegir las reflejadas en la pantalla de la figura 20-10.

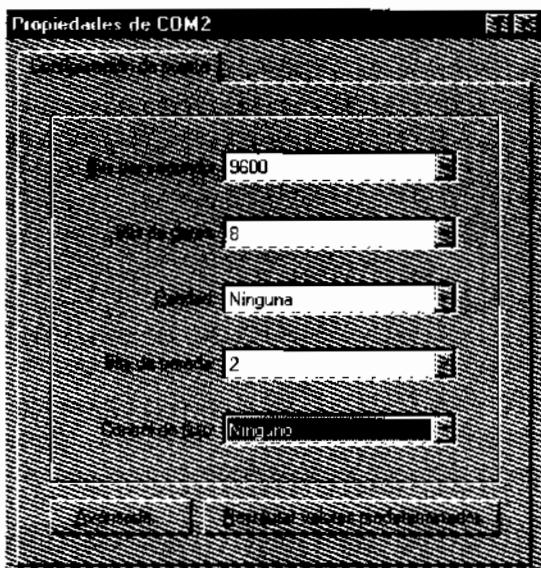
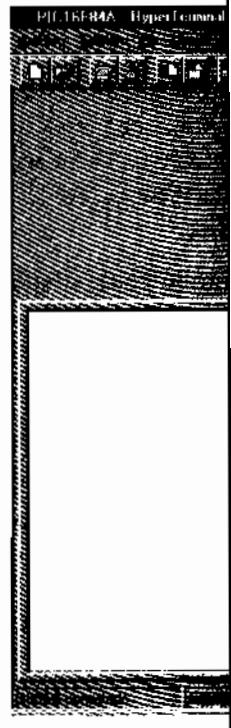


Figura 20-10 *HyperTerminal. Configuración del puerto*

- 5º A continuación aparece la pantalla de trabajo (figura 20-11).
- 6º Convendría seleccionar una fuente de letra grande para un trabajo más cómodo. Para ello, dentro de la pantalla de trabajo activar la opción *Ver > Fuente* y seleccionar, por ejemplo, el tipo de letra *Lucida Console 14* o alguna parecida.

Para conectar el ordenador y el microcontrolador a través del puerto serie COM 2, basta con activar la opción *Llama* (o ícono de teléfono colgado) y para desconectar hay que activar opción *Desconecta* (o ícono de teléfono descolgado). Evidentemente la conexión entre el puerto COM y el microcontrolador debe estar cableada de forma correcta mediante un cable RS232 como el de la figura 3-7 o similar.



Una vez configurado el puerto, se ha de ejecutar el icono *hticons.dll* (figura 20-12).



Figura 20-

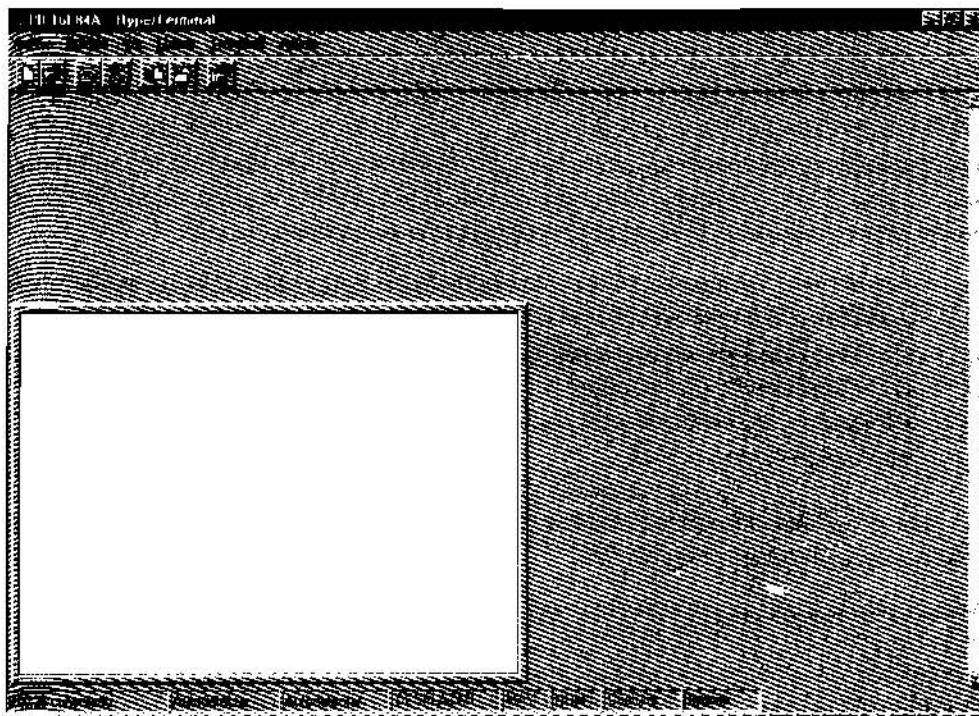


Figura 20-11 HyperTerminal. Pantalla de trabajo

Una vez configurada una conexión, para comenzar la comunicación simplemente ha de ejecutar el ícono de la conexión previamente configurada, tal como se muestra en la figura 20-12.



Figura 20-12 HyperTerminal. Ejemplos de conexiones configuradas

## 20.9 PROGRAMA EJEMPLO

Escribir por el teclado de un ordenador y que esta información se visualice en la pantalla del módulo LCD de un sistema con microcontrolador es un ejemplo inmediato de las aplicaciones que la conexión entre microcontrolador y puerto RS232 de un ordenador puede ofrecer. Para el correcto funcionamiento del siguiente programa ejemplo hay que cumplir los tres requisitos fundamentales de este tipo de comunicaciones:

- **Hardware:** Se ejecuta sobre el circuito de la figura 20-13, no se ha de olvidar conectarlo al puerto COM 1 o COM 2 del ordenador a través del cable RS232.
  - Un programa de comunicaciones que se debe abrir en el ordenador, como el *HyperTerminal* u otro similar.
  - Programa de control del microcontrolador que se debe ejecutar en el microcontrolador.

Un programa de control grabado en el microcontrolador podría ser el siguiente ejemplo suficientemente **documentado**:

\*\*\*\*\* RS232\_02.asm \*\*\*\*\*

; En el módulo LCD se visualizan los caracteres que se escriban en el teclado del ordenador ; y se transmiten a través de su puerto serie. Estos datos volverán a ser enviados por el ; microcontrolador al ordenador, por lo que también se visualizarán en su monitor.

• Se utilizará un programa de comunicaciones para que el ordenador pueda enviar datos a través de su puerto serie, como el Hyper Terminal de Windows o alguno similar.

; Concluyendo, lo que se escriba en el teclado del ordenador aparecerá en la pantalla del módulo LCD y en el monitor del HyperTerminal.

## ZONA DE DATOS \*\*\*\*

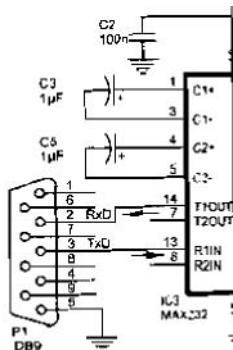
```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC  
LIST    P=16F84A  
INCLUDE <P16F84A.INC>
```

CBLOCK 0x0C  
GuardaDato  
ENDC

	ORG	0	
Inicio	call	LCD_Inicializa	; Inicializa el LCD y las líneas que se
	call	RS232_Inicializa	; van a utilizar en la comunicación con el puerto
Principal	call	RS232_LeeDatos	; serie RS232.
	movwf	GuardaDatos	; Espera recibir un carácter.
	call	LCD_Caracter	; Guarda el dato recibido.
			; Lo visualiza.

movf	Guad
call	RS2
goto	Prin

```
INCLUDE <R  
INCLUDE <L  
INCLUDE <R  
END
```



**Figura 2**

20-10 LIBRER

## **Una aplicació hacia el ordenador. RS232 MEN INC.**

- "RS232\_M" el microcom
  - "RS232\_Li" del ordenad

• Estas subrutinas permiten al ordenador a través del paralelo.

• Subrutina "RS232 Me

```

movf  GuardaData,W
call  RS232_EnviaDatos
goto Principal
;
INCLUDE <RS232.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <RETARDOS.INC>
END

```

ón se visualice en la  
jemplo inmediato de  
232 de un ordenador  
na ejemplo hay que  
nes:

no se ha de olvidar  
del cable RS232.  
ordenador, como el  
ebe ejecutar en el  
dría ser el siguiente

\*\*\*\*\*

rador

s

el

\*\*\*\*\*

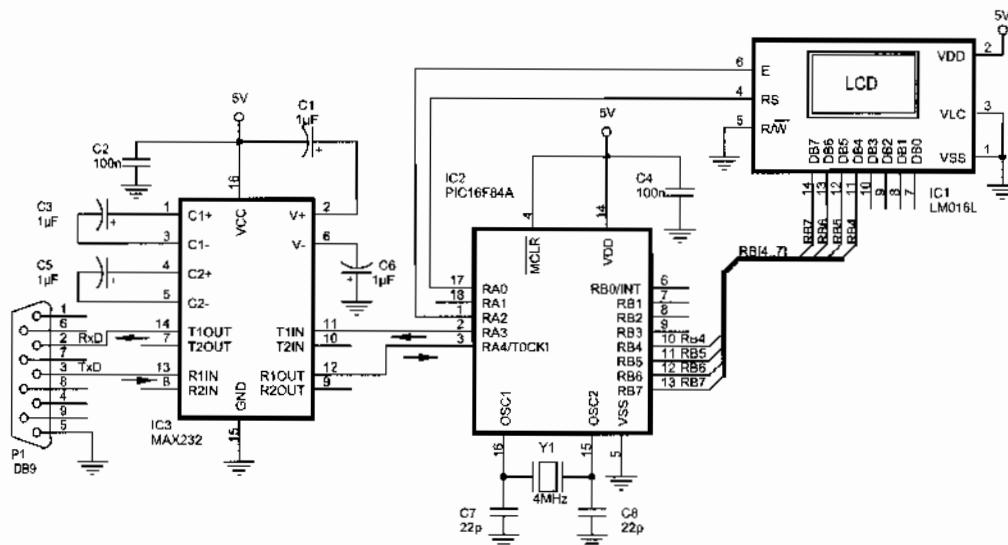


Figura 20-13 Hardware para el programa RS232\_02.asm y otros

## 20.10 LIBRERÍA RS232\_MEN.INC

Una aplicación muy útil es enviar mensajes desde el circuito del microcontrolador hacia el ordenador. Una librería con subrutinas para conseguir este objetivo puede ser la RS232\_MEN.INC, con las subrutinas:

- "RS232\_Mensaje". Visualiza en el monitor del ordenador un mensaje grabado en el microcontrolador.
- "RS232\_LíneasBlanco". Visualiza unas cuantas líneas en blanco en el monitor del ordenador.

```

***** Librería "RS232MEN.INC" *****
;
; Estas subrutinas permiten transmitir mensajes desde el microcontrolador hacia el
; ordenador a través del puerto serie RS232.
;
; Subrutina "RS232_Mensaje"

```

;  
; Envía por el puerto serie el mensaje apuntado por el registro W y grabado en memoria  
; de programa del PIC. El mensaje se visualizará en el monitor del ordenador. Esta  
; subrutina sigue la estructura de funcionamiento similar a la ya conocida "LCD\_Mensaje"  
; para visualización de mensajes en pantallas de modulos LCD.  
;

```
CBLOCK
RS232_ApuntaCaracter
RS232_ValorCaracter
ENDC
```

RS232\_ApuntaCaracter ; Apunta al carácter a visualizar.
RS232\_ValorCaracter ; Valor ASCII del carácter a visualizar.

RS232\_Mensaje
movwf RS232\_ApuntaCaracter ; Posición del primer carácter del mensaje.
movlw Mensajes ; Halla la posición relativa del primer carácter
subwf RS232\_ApuntaCaracter,F ; del mensaje respecto del comienzo de todos los
; mensajes (identificados mediante la etiqueta
; "Mensajes").  
decf RS232\_ApuntaCaracter,F ; Para compensar la posición que ocupa la
RS232\_VisualizaOtroCaracter ; instrucción "addwf PCL,F".  
movf RS232\_ApuntaCaracter,W ; Apunta al carácter a visualizar.
call Mensajes ; Obtiene el código ASCII del carácter apuntado.
movwf RS232\_ValorCaracter ; Guarda el valor de carácter.
movf RS232\_ValorCaracter,F ; Lo único que hace es posicionar flag Z. En caso
btfsr STATUS,Z ; que sea "0x00", que es código indicador final
goto RS232\_FinMensaje ; de mensaje, sale fuera.  
RS232\_NoUltimoCaracter
call RS232\_EnviaDatos ; Visualiza el carácter ASCII leído.
incf RS232\_ApuntaCaracter,F ; Apunta a la posición del siguiente carácter
goto RS232\_VisualizaOtroCaracter ; dentro del mensaje.  
RS232\_FinMensaje
return

#### ; Subrutina "RS232\_LineasBlanco"

;  
; Visualiza unas cuantas líneas en blanco en el monitor del ordenador.

```
CBLOCK
RS232_ContadorLineas
ENDC
```

RS232\_LineasBlanco
movlw d'10' ; Por ejemplo este número de líneas en
movwf RS232\_ContadorLineas ; blanco.  
R232\_LineasBlancoLazo
movlw .10 ; Código del salto de línea
call RS232\_EnviaDatos
decfsz RS232\_ContadorLineas,F
goto R232\_LineasBlancoLazo
movlw .13 ; Código del retorno de carro.
call RS232\_EnviaDatos
return

El siguiente  
*HyperTerminal*.

\*\*\*\*\*  
;  
; Este programa envía un  
; ordenador. Es decir, en  
; ZONA DE DATOS \*\*\*

CONFIG
LIST P=16
INCLUDE <PIC16F84.H>
CBLOCK 0x00
ENDC

RetornoCarro EQU 0xd
CambioLinea EQU 0xa

; ZONA DE CÓDIGOS \*

ORG	0
Inicio	call LCD
	call RS232
Principal	movlw Mensaje0
	call RS232
	call RetornoCarro
	goto Principal

"Mensajes" —

Mensajes	addwf PCL
Mensaje0	DT RetornoCarro
	DT "En el IES"
	DT RetornoCarro
	DT " se pue
	DT RetornoCarro
	DT " DESA
	DT CambioLin

INCLUDE <STDIO.H>
INCLUDE <REGS.H>
INCLUDE <REGS.H>
INCLUDE <REGS.H>
END

El siguiente programa ejemplo lanza un mensaje publicitario a la ventana del *HyperTerminal*.

```
***** RS232_05.asm *****
```

Este programa envía un mensaje grabado en la memoria de programa del microcontrolador al ordenador. Es decir, en el monitor del ordenador aparecerá el mensaje grabado en el PIC.

```
ZONA DE DATOS *****
```

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
ENDC
```

RetomoCarro	EQU	.13	; Código de tecla "Enter" o "Retorno de Carro".
CambioLinea	EQU	.10	; Código para el cambio de línea.

```
ZONA DE CÓDIGOS *****
```

ORG	0		
Inicio	call	LCD_Inicializa	; Inicializa el módulo LCD y las líneas que se
	call	RS232_Inicializa	; van a utilizar en la comunicación con el puerto
Principal	movlw	Mensaje0	; serie RS232.
	call	RS232_Mensaje	; Carga la primera posición del mensaje.
	call	Retardo_5s	; Lo visualiza en el ordenador.
	goto	Principal	

```
"Mensajes" -----
```

```
Mensajes
```

```
addwf PCL,F
```

```
Mensaje0
```

```
DT RetomoCarro
DT "En el I.E.S. ISAAC PERAL de Torrejón de Ardoz"
DT RetomoCarro, CambioLinea
DT "se puede estudiar el Ciclo Formativo"
DT RetomoCarro, CambioLinea
DT "DESARROLLO de PRODUCTOS ELECTRONICOS."
DT CambioLinea, CambioLinea, CambioLinea, 0x00
```

```
INCLUDE <RS232.INC>
INCLUDE <RS232MEN.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <RETARDOS.INC>
END
```

## 20.11 SISTEMA DE MONITORIZACIÓN

Un sistema de monitorización de entradas permite visualizar en el monitor del ordenador el estado de unas entradas conectadas al PIC16F84A, como en el ejemplo de la figura 20-14. A continuación se expone un ejemplo de programa de control suficientemente documentado.

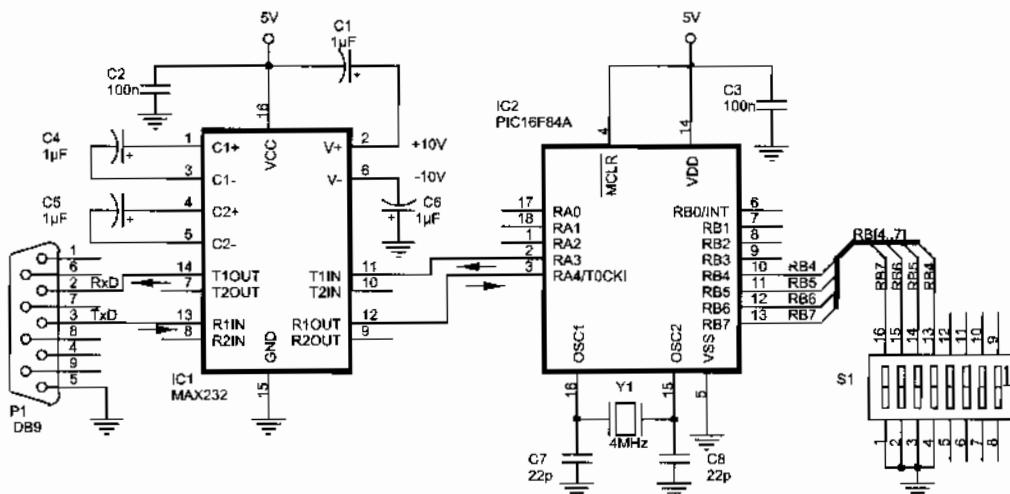


Figura 20-14 Sistema de monitorización

```
***** RS232_10.asm *****
;
; SISTEMA DE MONITORIZACIÓN: Se trata de leer el estado de las entradas conectadas a las
; líneas <RB7:RB4> del Puerto B y se envía por el puerto RS232 a un terminal para monitorizar
; el estado de los mismos.
; Se utilizará las interrupciones por cambio de nivel en una línea del Puerto B, por ello
; las entradas deben conectarse a la parte alta del Puerto B.
;
; ZONA DE DATOS *****
;
; CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

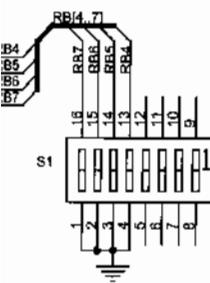
CBLOCK 0x0C
ENDC

RetornoCarro EQU .13 ; Código de tecla "Enter" o "Retorno de Carro".
CambioLinea EQU .10 ; Código para el cambio de linea.

#define Entrada0 PORTB,4 ; Define dónde se sitúan las entradas.
#define Entrada1 PORTB,5
#define Entrada2 PORTB,6
```

#DEFINE Entrada3	P
; ZONA DE CODIGO	
ORG 0	In
goto 4	Le
Mensajes	
addwf PC	
MensajeEntradas	
DT RetornoC	
DT "Entrada	
DT RetornoC	
DT -----	
DT RetornoC	
MensajeAbierto	
DT " Abierto	
MensajeCerrado	
DT " Cerrado	
Inicio	
call RS	
bsf ST	
bsf En	
bcf OP	
bcf ST	
call RS	
call RS	
call Le	
movlw b'1	
movwf IN	
Principal	
sleep	
goto Pri	
; Subrutina de Servicio	
; Lee el estado de las en	
LeeEntradasVisualiza	
call RS	
movfw Me	
call RS	
LeeEntrada3	
btfs En	
goto Em	
call Vi	
goto Le	

r en el monitor del  
o en el ejemplo de la  
ograma de control



adas a las  
onitorizar

o de Carro".

as.

```
#DEFINE Entrada3 PORTB,7

; ZONA DE CODIGOS ****
; Aquí se sitúa el vector de interrupción para
; atender las subrutinas de interrupción.

ORG 0
goto Inicio
ORG 4
goto LeeEntradasVisualiza

Mensajes
addwf PCL,F ; Los mensajes no deben sobrepasar las 256
; primeras posiciones de memoria de programa.

MensajeEntradas
DT RetomoCarro, CambioLinea
DT "Entrada 3 Entrada 2 Entrada 1 Entrada 0"
DT RetomoCarro, CambioLinea
DT "-----"
DT RetornoCarro, CambioLinea, 0x00

MensajeAbierto
DT " Abierto ", 0x00

MensajeCerrado
DT " Cerrado ", 0x00

Inicio
call RS232_Inicializa
bsf STATUS,RP0 ; Configura como entrada las 4 líneas correspondientes
bsf Entrada0 ; del Puerto B respetando la configuración del
bsf Entrada1 ; resto de las líneas.
bsf Entrada2
bsf Entrada3
bcf OPTION_REG,NOT_RBPU ; Activa las resistencias de Pull-Up del Puerto B.
bcf STATUS,RP0
call RS232_LineaBlanco ; Visualiza unas cuantas líneas en blanco
call RS232_LineaBlanco ; para limpiar la pantalla.
call LeeEntradasVisualiza ; Lee las entradas y visualiza por primera vez.
movlw b'10001000' ; Habilita la interrupción RBI y la general.
movwf INTCON

Principal
sleep ; Espera en modo de bajo consumo que se
goto Principal ; modifique una entrada.

; Subrutina de Servicio a la Interrupcion" -----
; Lee el estado de las entradas y las monitoriza en la pantalla del HyperTerminal.

LeeEntradasVisualiza
call RS232_LineaBlanco
movlw MensajeEntradas ; Nombre de las entradas.
call RS232_Mensaje ; Lo visualiza en el ordenador.

LeeEntrada3
btfs Entrada3 ; ¿Entrada = 1? , ¿Entrada = Abierta?
goto Entrada3Cerrado ; No, está cerrada.
call VisualizaAbierto
goto LeeEntrada2
```

```

Entrada3Cerrado
    call    VisualizaCerrado
LeeEntrada2
    btfs  Entrada2
    goto   Entrada2Cerrado
    call    VisualizaAbierto
    goto   LeeEntrada1
Entrada2Cerrado
    call    VisualizaCerrado
LeeEntrada1
    btfs  Entrada1
    goto   Entrada1Cerrado
    call    VisualizaAbierto
    goto   LeeEntrada0
Entrada1Cerrado
    call    VisualizaCerrado
LeeEntrada0
    btfs  Entrada0
    goto   Entrada0Cerrado
    call    VisualizaAbierto
    goto   FinVisualiza
Entrada0Cerrado
    call    VisualizaCerrado
FinVisualiza
    call    RS232_LineaBlanco
    bcf    INTCON,RBIF
    retfie ; Limpia el flag de reconocimiento de la
; interrupción.
;
VisualizaAbierto
    movlw  MensajeAbierto
    call   RS232_Mensaje
    return ; Visualiza el mensaje "Abierto"
; en el HyperTerminal.
VisualizaCerrado
    movlw  MensajeCerrado
    call   RS232_Mensaje
    return ; Visualiza el mensaje "Cerrado"
; en el HyperTerminal.

INCLUDE <RS232.INC>
INCLUDE <RS232MEN.INC>
INCLUDE <RETARDOS.INC>
END

```

## **20.12 SISTEMA DE GOBIERNO DESDE ORDENADOR**

Una de las aplicaciones más interesantes es el control mediante ordenador de dispositivos conectados a un microcontrolador. En el siguiente ejemplo, mediante el teclado de un ordenador, se van a controlar cuatro salidas simuladas con cuatro diodos LEDs. El hardware utilizado es el descrito en la figura 20-15 y el programa de control a cargar en el PIC16F84A podría ser:

SISTEMA DE GOBIERNO		
el movimiento de una tecla		
TECLA (Por)		
	t	
	b	
	a	
	l	
		Espacio
La pulsación de cualquier		
RB3 (Adelante), RB2 (Atrás)		
El movimiento que se		
realiza es el de la tecla del sistema y también		
El programa debe permitir la salida. Es decir, para		
en lugar de por byte (u)		
ZONA DE DATOS		
	CONFIG	
	LIST	P
	INCLUDE	<
CBLOCK 0		
TeclaPulsada		
MensajeApun		
ENDC		
#DEFINE SalidaAdelante		
#DEFINE SalidaAtras		
#DEFINE SalidaIzquierda		
#DEFINE SalidaDerecha		
TeclaAdelante	EQ	
TeclaAtras	EQ	
TeclaIzquierda	EQ	
TeclaDerecha	EQ	
TeclaParada	EQ	
ZONA DE CÓDIGOS		
	ORG	0
Inicio	call	LC
	call	RS
	bsf	ST
	baf	Sa

\*\*\*\*\* RS232\_11.asm \*\*\*\*\*

; SISTEMA DE GOBIERNO DESDE ORDENADOR: Desde el teclado de un ordenador se desea comandar ; el movimiento de una estructura móvil, según la siguiente tabla:

TECLA (Por ejemplo)	MOVIMIENTO
t	Adelante
b	Atrás
a	Izquierda
i	Derecha
Espacio	Parada

; La pulsación de cualquiera de estas teclas activa el estado de las salidas correspondiente ; RB3 (Adelante), RB2 (Atrás), RB1 (Izquierda), RB0 (Derecha) y apaga el resto.

; El movimiento que se está realizando aparece reflejado en un mensaje en el visualizador LCD ; del sistema y también en la pantalla del ordenador.

; El programa debe permitir modificar fácilmente en posteriores revisiones en el hardware de ; la salida. Es decir, para activar las salidas conviene utilizar el direccionamiento por bit ; en lugar de por byte (utilizar instrucciones "bsf" y "bcf", en lugar de "mov").

; ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

CBLOCK 0x0C

TeclaPulsada

; Va a guardar el contenido de la tecla pulsada.

MensajeApuntado

; Va a guarda la dirección del mensaje apuntado.

ENDC

```
#DEFINE SalidaAdelante PORTB,3
```

; Define dónde se sitúan las salidas.

```
#DEFINE SalidaAtras PORTB,2
```

```
#DEFINE SalidaIzquierda PORTB,1
```

```
#DEFINE SalidaDerecha PORTB,0
```

TeclaAdelante EQU 't'

; Código de las teclas utilizadas.

TeclaAtras EQU 'b'

Teclal Izquierda EQU 'a'

TeclaDerecha EQU 'i'

TeclaParada EQU ''

; Código de la tecla espaciadora, (hay un espacio, ; tened cuidado al teclear el programa).

; ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG 0
```

Inicio

call LCD\_Inicializa

call RS232\_Inicializa

bsf STATUS,RP0

bcf SalidaAdelante

; Configura como salidas las 4 líneas del

; del Puerto B respetando la configuración del

```

        bcf    SalidaAtras          ; resto de las líneas.
        bcf    SalidaIzquierda
        bcf    SalidaDerecha
        bcf    STATUS,RP0
        call   Parado               ; En principio todas las salidas deben estar
        Principal
        call   RS232_LeeDatos      ; apagadas.
        call   TesteaTeclado       ; Espera a recibir un carácter.
        goto  Principal

; "Mensajes"
;
Mensajes
        addwf  PCL,F
MensajeParado
        DT "Sistema PARADO", 0x00
MensajeAdelante
        DT "Marcha ADELANTE", 0x00
MensajeAtras
        DT "Marcha ATRAS", 0x00
MensajeIzquierda
        DT "Hacia IZQUIERDA", 0x00
MensajeDerecha
        DT "Hacia DERECHA", 0x00

; Subrutina "TesteaTeclado"
;
; Testea el teclado y actúa en consecuencia.

TesteaTeclado
        movwf  TeclaPulsada         ; Guarda el contenido de la tecla pulsada.
        xorlw  TeclaAdelante        ; ¿Es la tecla del movimiento hacia adelante?
        btfsc  STATUS,Z
        goto  Adelante             ; Sí, se desea movimiento hacia adelante.

;
        movf   TeclaPulsada,W      ; Recupera el contenido de la tecla pulsada.
        xorlw  TeclaAtras           ; ¿Es la tecla del movimiento hacia atrás?
        btfsc  STATUS,Z
        goto  Atras                 ; Sí, se desea movimiento hacia atrás.

;
        movf   TeclaPulsada,W      ; Recupera el contenido de la tecla pulsada.
        xorlw  TeclaIzquierda       ; ¿Es la tecla del movimiento hacia la izquierda?
        btfsc  STATUS,Z
        goto  Izquierda              ; Sí, se desea movimiento hacia la izquierda.

;
        movf   TeclaPulsada,W      ; Recupera el contenido de la tecla pulsada.
        xorlw  TeclaDerecha         ; ¿Es tecla del movimiento hacia la derecha?
        btfsc  STATUS,Z
        goto  Derecha                ; Sí, se desea movimiento hacia la derecha.

;
        movf   TeclaPulsada,W      ; Recupera el contenido de la tecla pulsada.
        xorlw  TeclaParada          ; ¿Es la tecla de parada?.
        btfss  STATUS,Z

```

	goto	Fin
Parado	bcf	Salid
	movlw	Men
	goto	Visu
Adelante	bcf	Salid
	bsf	Salid
	bcf	Salid
	bcf	Salid
	movlw	Men
	goto	Visu
Atras	bcf	Salid
	bsf	Salid
	bcf	Salid
	bcf	Salid
	movlw	Mens
	goto	Visu
Izquierda	bcf	Salid
	bcf	Salid
	bsf	Salid
	bcf	Salid
	movlw	Mens
	goto	Visu
Derecha	bcf	Salid
	bcf	Salid
	bcf	Salid
	bsf	Salid
	moviw	Mens
	goto	Visu
	bcf	Salid
	bcf	Salid
	bsf	Salid
	bcf	Salid
	movlw	Mens
	goto	Visu
	bcf	Salid
	bcf	Salid
	bsf	Salid
	bcf	Salid
	movlw	Mens
	goto	Visu
	bcf	Salid
	bcf	Salid
	bsf	Salid
	bcf	Salid
	movlw	Mens
	goto	Visu
	bcf	Salid
	bcf	Salid
	bsf	Salid
	bcf	Salid
	movlw	Mens
	call	LCD
	movf	Mens
	call	LCD
	call	RS23
	movf	Mens
	call	RS23
	call	RS23
Fin	INCLUDE <RS	
	INCLUDE <RS	
	INCLUDE <L	
	INCLUDE <L	
	INCLUDE <R	
	END	

	goto	Fin	; No es ninguna tecla de movimiento. Sale.
en estar	bcf	SalidaAdelante	
	bcf	SalidaAtras	; Como se ha pulsado la tecla de parada se
	bcf	SalidaIzquierda	; desactivan todas las salidas.
	bcf	SalidaDerecha	
	movlw	MensajeParado	
	goto	Visualiza	
	Adelante		
	bcf	SalidaAtras	
	bsf	SalidaAdelante	
	bcf	SalidaIzquierda	
	bcf	SalidaDerecha	
	movlw	MensajeAdelante	
	goto	Visualiza	
	Atras		
	bcf	SalidaAdelante	
	bsf	SalidaAtras	
	bcf	SalidaIzquierda	
	bcf	SalidaDerecha	
	movlw	MensajeAtras	
	goto	Visualiza	
	Izquierda		
	bcf	SalidaAdelante	
	bcf	SalidaAtras	
	bsf	SalidaIzquierda	
	bcf	SalidaDerecha	
	movlw	MensajeIzquierda	
	goto	Visualiza	
	Derecha		
	bcf	SalidaAdelante	
	bcf	SalidaAtras	
	bcf	SalidaIzquierda	
	bsf	SalidaDerecha	
	movlw	MensajeDerecha	
	Visualiza		
	movwf	MensajeApuntado	; Guarda la posición del mensaje.
	call	LCD_Borra	; Borra la pantalla del modulo LCD.
	movf	MensajeApuntado,W	; Visualiza el mensaje en la pantalla
	call	LCD_Mensaje	; del visualizador LCD.
	call	RS232_LineasBlanco	; Borra la pantalla del ordenador.
	movf	MensajeApuntado,W	
	call	RS232_Mensaje	; Lo visualiza en el HyperTerminal.
	call	RS232_LineasBlanco	
	Fin	return	
		INCLUDE <RS232.INC>	
		INCLUDE <RS232MEN.INC>	
		INCLUDE <LCD_4BIT.INC>	
		INCLUDE <LCD_MENS.INC>	
		INCLUDE <RETARDOS.INC>	
		END	

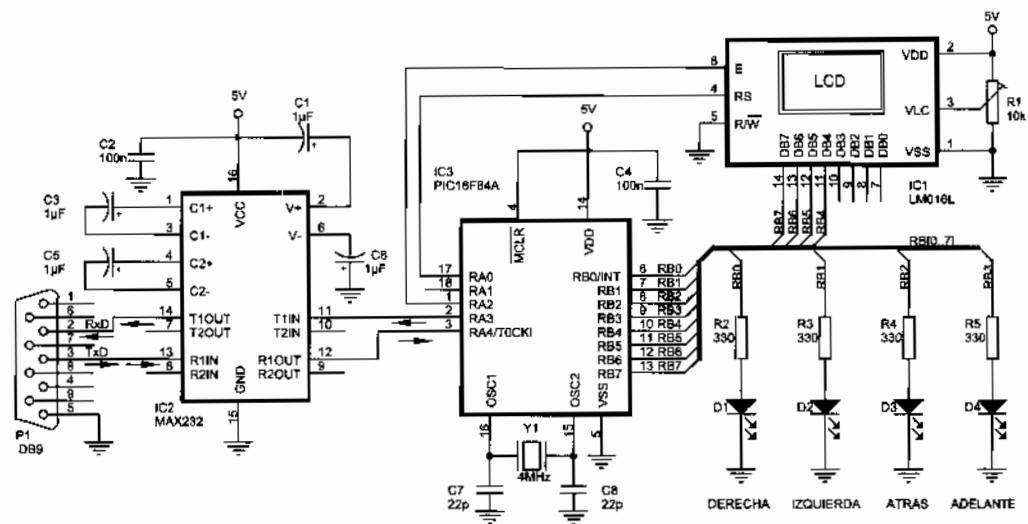


Figura 20-15 Sistema de gobierno mediante ordenador

## 20.13 PRÁCTICAS DE LABORATORIO

Respetando el procedimiento descrito en los temas anteriores, diseñar, ensamblar, simular, grabar el microcontrolador y comprobar los siguientes programas utilizando el hardware apropiado para cada uno de ellos. El lector puede introducir todas las mejoras que considere conveniente.

**RS232\_01.asm:** Lo que se escriba por el teclado del ordenador se visualizará en el módulo LCD.

**RS232\_02.asm:** Programa igual que en anterior con la diferencia de que lo escrito por el teclado del ordenador se visualizará en el módulo LCD y volverá a ser enviado al ordenador por lo que se visualizará, también en el monitor de este.

**RS232\_03.asm:** Lo que se escriba por el teclado se visualizará en el módulo LCD y en el monitor del ordenador, pero en este último se visualizará un solo carácter por línea.

**RS232\_04.asm:** Este programa es igual que el RS232\_02.asm pero con la diferencia de que si pulsa el retorno de carro comienza a escribir en la segunda línea del módulo LCD.

**RS232\_05.asm:**  
microcontrolador al o  
subrutina RS232\_M  
RS232MEN.INC.

**RS232\_06.asm:**  
microcontrolador al n  
durante unos segundos

**RS232\_07.asm:**  
memoria de programa  
que se pulse la tecla E

**RS232\_08.asm:**  
formato hexadecimal  
formato decimal. Por e

- Monitor del o código ASCII
- Pantalla LCD: en formato hex

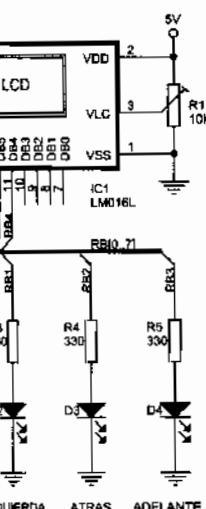
**RS232\_09.asm:**  
esquema de la figura 2  
líneas RB4 a RB7 del P  
monitorizar el estado  
segundos.

**RS232\_10.asm:**  
interrupciones por camb

**RS232\_11.asm:**  
figura 20.15. Desde el t  
estructura móvil, segú  
estado de las salidas co  
(Derecha) y apaga el res

El movimiento o  
pantalla del módulo LC

El programa deb  
posteriores revisiones. P  
en lugar de por byte, y e



**RS232\_05.asm:** Envía un mensaje grabado en la memoria de programa del microcontrolador al ordenador que aparece en la pantalla del ordenador. Se utilizará una subrutina RS232\_Mensaje que previamente se ha realizado en la librería RS232MEN.INC.

**RS232\_06.asm:** Envía varios mensajes grabados en la memoria de programa del microcontrolador al monitor del ordenador. Cada mensaje permanecerá en el monitor durante unos segundos hasta que aparezca el siguiente.

**RS232\_07.asm:** En el monitor del ordenador se visualizan mensajes grabados en la memoria de programa del microcontrolador. El cambio de mensaje se ejecuta cada vez que se pulse la tecla *Enter*.

**RS232\_08.asm:** Visualiza en la pantalla del módulo LCD el código ASCII en formato hexadecimal de la tecla pulsada y también en el monitor del ordenador en formato decimal. Por ejemplo:

- Monitor del ordenador: "k-107", donde la "k" es la tecla pulsada y el "107" el código ASCII en formato decimal.
- Pantalla LCD: "k-6B", donde la "k" es la tecla pulsada y el "6B" el código ASCII en formato hexadecimal.

**RS232\_09.asm:** Sistema de monitorización. Se utilizará el circuito descrito en esquema de la figura 20-14. Se trata de leer el estado de las entradas conectadas a las líneas RB4 a RB7 del Puerto B y se envía vía serie por el puerto RS232 al ordenador para monitorizar el estado de los mismos. El estado de las entradas se mostrará cada 5 segundos.

**RS232\_10.asm:** Igual que el ejercicio anterior pero realizado mediante interrupciones por cambio en una de las líneas altas del Puerto B.

**RS232\_11.asm:** Sistema de gobierno desde ordenador. Se utilizará el montaje de la figura 20-15. Desde el teclado de un ordenador se desea gobernar el movimiento de una estructura móvil, según la tabla 20-1. La pulsación de cualquiera de estas teclas activa el estado de las salidas correspondiente RB3 (Adelante), RB2 (Atrás), RB1 (Izquierda), RB0 (Derecha) y apaga el resto.

El movimiento que se está realizando aparece reflejado en un mensaje en la pantalla del módulo LCD del sistema y también en el monitor del ordenador.

El programa debe permitir una fácil modificación en el hardware de salida para posteriores revisiones. Para activar las salidas conviene utilizar el direccionamiento por bit en lugar de por byte, y emplear instrucciones *bsf* y *bcf*, en lugar de las "mov".

TECLA	MOVIMIENTO
T	Adelante
B	Atrás
A	Izquierda
L	Derecha
(Espacio)	Parada

Tabla 20-1

Actualmente hay un bus serie desarrollado por estos dispositivos para sistemas completos, como puede ser el bus I2C, así como algunos otros.

## 21.1 EL BUS

Numerosos protocolos cumplen con los criterios de diseños modernos.

- El sistema es más económico que los periféricos individuales.
- La conexión es más sencilla y más fácil de realizar.
- Estos sistemas tienen una mayor velocidad de transferencia de datos.
- El sistema no es tan complejo como para que su implementación no sería posible.

Para implementar un sistema de control de movimiento se necesitan buses serie, ya que los sistemas de control paralelos, requieren poder manejar directamente todos los protocolos para la corrección de errores.

*Hans*

## CAPÍTULO 21

# BUS I2C

Actualmente hay en el mercado multitud de dispositivos que son gobernados por un bus serie desarrollado por la empresa *Philips* que es conocido como bus I2C. Entre estos dispositivos podemos encontrar desde un simple circuito integrado hasta sistemas completos, como puede ser un sintonizador. En este capítulos y posteriores se describe el bus I2C, así como algunos de los dispositivos que se pueden utilizar.

### 21.1 EL BUS I2C

Numerosos proyectos que utilizan microcontroladores de 8 bits suelen regirse por criterios de diseños muy reiterados:

- El sistema consta de, al menos, un microcontrolador y varios dispositivos periféricos como memorias, LCD, convertidores ADC, etc.
- La conexión entre los distintos dispositivos que componen el sistema debe ser fácil de realizar y su coste mínimo.
- Estos sistemas suelen realizar funciones que no requieren una alta tasa de transferencia de datos, generalmente no superior a 100 kbits por segundo.
- El sistema no debe depender de los dispositivos conectados a él. De otro modo no sería posible realizar modificaciones o mejoras.

Para implementar un sistema que satisfaga estos criterios se necesita una estructura de **bus serie**, ya que aunque no tienen ni la capacidad ni la velocidad de los buses paralelos, requieren poco hardware y un mínimo de cableado. Este bus serie no debe ser simplemente un hilo de conexión, debe incorporar una serie de procedimientos o protocolos para la correcta comunicación entre los componentes del sistema.

Todos estos criterios son la base sobre la que se fundamentan las especificaciones del **bus serie I<sup>2</sup>C** para la interconexión de circuitos integrados o bus I<sup>2</sup>C (*IIC, Inter Integrated Circuit Bus*) desarrollado por *Philips Semiconductors* ([www.semiconductors.philips.com](http://www.semiconductors.philips.com)) y que es ampliamente utilizado en la industria electrónica.

El I<sup>2</sup>C es un bus serie, formado por dos hilos, que puede conectar varios dispositivos mediante hardware muy simple, tal como ilustra la figura 21-1. Por esos dos hilos se produce una comunicación serie, bit a bit. Se transmiten dos señales, una por cada línea:

- **SCL, (Serial Clock).** Es la señal de reloj que se utiliza para la sincronización de los datos.
- **SDA, (Serial Data).** Es la línea para la transferencia serie de los datos.

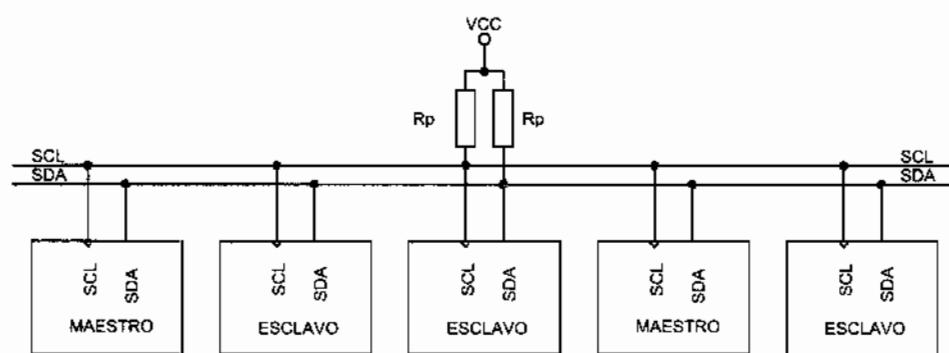


Figura 21-1 Estructura de un bus I<sup>2</sup>C

Los dispositivos conectados al bus I<sup>2</sup>C mantienen un protocolo de comunicaciones del tipo maestro/esclavo (o *master/slave*). Las funciones del maestro y del esclavo se diferencian en:

- El circuito maestro inicia y termina la transferencia de información, además de controlar la señal de reloj. Normalmente es un microcontrolador.
- El esclavo es el circuito direccionado por el maestro.

La línea SDA es bidireccional, es decir, tanto el maestro como los esclavos pueden actuar como transmisores o receptores de datos, dependiendo de la función del dispositivo. Así por ejemplo, un display es sólo un receptor de datos mientras que una memoria recibe y transmite datos.

La generación de señales de reloj (SCL) es siempre responsabilidad del maestro.

Cada dispositivo conectado al bus I<sup>2</sup>C es reconocido por una única dirección que lo diferencia del resto de los circuitos conectados. Los dispositivos compatibles con bus

I<sup>2</sup>C suelen llevar 2 dispositores para evitar la dirección.

El bus I<sup>2</sup>C es un dispositivo capaz de un solo microcontrolador.

## 21.2 HARDWARE

El hardware de salida de los dispositivos para poder realizar la

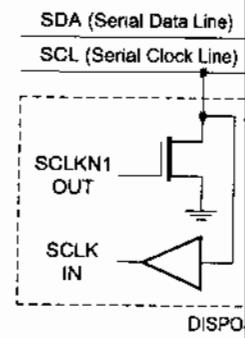


Figura 21-2

Las líneas SDA y SCL de unas resistencias de cada dispositivo pueden

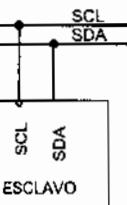
- Que el transistor correspondiente, decir, el bus es
- Que el transistor de la línea depa hay ningún otro través de la res

El cálculo de las la capacidad del bus y

especificaciones  
I<sup>2</sup>C (IIC, Inter  
Semiconductors  
en la industria

conectar varios  
21-1. Por esos  
señales, una por

sincronización  
datos.



comunicaciones  
del esclavo se

ión, además de

esclavos pueden  
la función del  
ientras que una

del maestro.

a dirección que  
atibles con bus

I2C suelen llevar 2 ó 3 pines para poder modificar esta dirección de modo que el diseñador pueda evitar que en un mismo diseño haya 2 o más esclavos con la misma dirección.

El bus I2C puede ser *multi-master*, esto significa que puede soportar más de un dispositivo capaz de controlar el bus. Los sistemas más comunes están constituidos por un solo microcontrolador maestro.

## 21.2 HARDWARE DEL BUS I2C

El hardware del bus I2C se basa en la And cableada (figura 21-2). Las etapas de salida de los dispositivos conectados al bus deben ser drenador abierto, o colector abierto, para poder realizar la función And cableada.

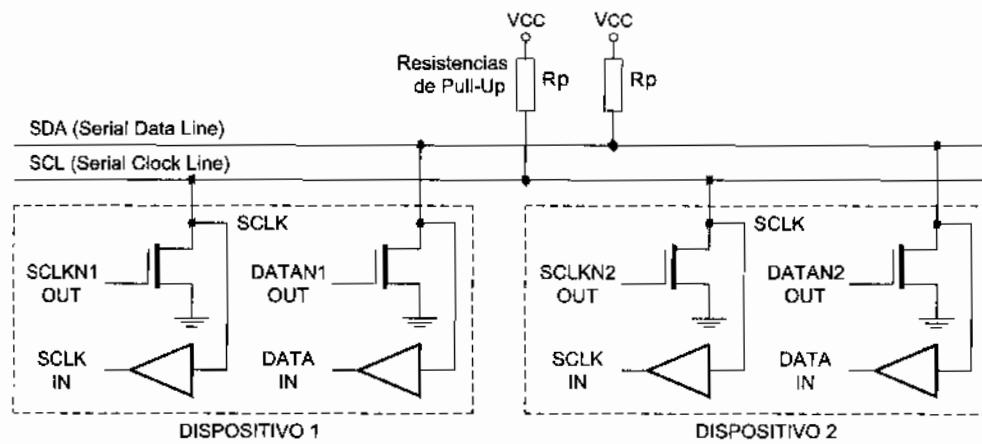


Figura 21-2 Etapa de salida de los dispositivos I2C

Las líneas SDA y SCL están conectadas a tensión positiva de alimentación a través de unas resistencias de Pull-Up, Rp. Dependiendo del estado del transistor de salida de cada dispositivo puede ocurrir alguno de estos dos casos:

- Que el transistor esté saturado, con lo cual lleva a nivel bajo o “0”, a la línea correspondiente, independientemente del estado de los otros transistores. Es decir, el bus está ocupado a nivel bajo.
- Que el transistor esté en corte (estado de alta impedancia) con lo cual el estado de la línea depende de los otros transistores. Es decir, el bus está libre y, si no hay ningún otro transistor saturado, la línea se encuentran en estado alto a través de la resistencia de Pull-Up conectada a la alimentación.

El cálculo de las resistencias de Pull-Up depende de la tensión de alimentación, de la capacidad del bus y del número de dispositivos conectados. Esto se tabula en unas

tablas que facilita el fabricante *Philips Semiconductors*. De todas formas, un valor de  $4k7$  es satisfactorio para la mayoría de las aplicaciones.

Se pueden transferir datos por el bus a una velocidad máxima de 100 kbytes por segundo (modo *standard*). El número de dispositivos conectados al bus viene limitado por la capacidad máxima admitida, que es de 400 pF.

### 21.3 TRANSFERENCIA DE UN BIT POR LA LÍNEA SDA

Para transferir un bit por la línea de datos SDA debe ser generado un pulso de reloj por la línea SCL (figura 21-3). Los bits de datos transferidos por la línea SDA deben mantenerse estables mientras la línea SCL esté a nivel alto. El estado de la línea SDA sólo puede cambiar cuando la línea SCL está a nivel bajo.

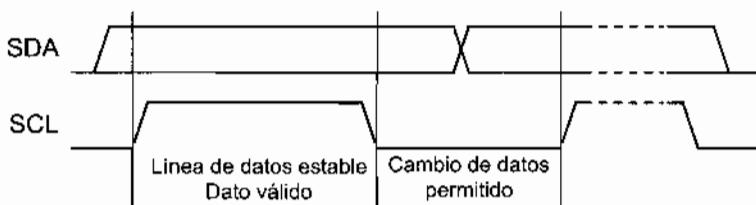


Figura 21-3 Transferencia por un bit por el bus I2C

Si la línea SDA cambia mientras SCL está a nivel alto, no se interpreta como dato, sino como una condición especial (*Start* o *Stop*), explicada a continuación.

#### 21.4 CONDICIONES DE START Y STOP

Para que la transferencia de información pueda ser iniciada el bus no debe estar ocupado. Esto quiere decir que los transistores de salida de todos los dispositivos conectados al bus I2C deben estar en alta impedancia. Para indicar que el bus está libre (no ocupado) las líneas del reloj (SCL) y datos (SDA) deben estar a nivel alto. Una vez que se ha verificado esto, el transmisor procederá a enviar un bit cada pulso de reloj.

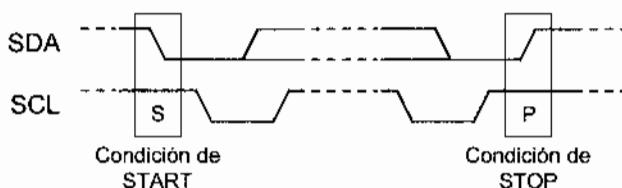


Figura 21-4 Condiciones de Start y Stop

### En la transmisión

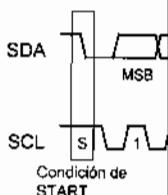
- 1º Condición de transmisión** permanece a un nivel bajo. Los datos.
  - 2º Transmisión**
  - 3º Condición de transmisión** permanece a un nivel alto. Ésta es la condición deseada.

Las condiciones de maestro. El bus I2C se considera libre de奴 dispositivos esclavos y es necesario que le permitan

## 21.5 TRANSF

Cada dato enviado tiene 8 bytes que se puede enviar por el bit 7 que es de muestreo.

Una vez que se asentimiento o reconocimiento, el receptor ejecuta este resultado. Cada grupo de 8 bits de



Figs

El bit de reconocimiento de reloj SCL corresponde al momento en que el transmisor deja libre la línea. El receptor debe hacer que la línea permanezca en alto durante el impulso de reloj SCL (figura 10-1).

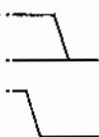
Si el esclavo-receptor debe detectar la situación

mas, un valor de 4k7

na de 100 kbits ·por  
l bus viene limitado

## LÍNEA SDA

ado un pulso de reloj  
la línea SDA deben  
ado de la línea SDA



2C

interpreta como dato,  
ción.

el bus no debe estar  
dos los dispositivos  
que el bus está libre  
a nivel alto. Una vez  
pulso de reloj.

---

---

En la transmisión de datos se da la siguiente secuencia de sucesos (figura 21-4):

- 1º **Condición de Start.** SDA debe estar en flanco de bajada mientras que SCL permanece a nivel alto. Esta condición señala el comienzo de la transferencia de datos.
- 2º **Transmisión de los bits de información a cada pulso de reloj.**
- 3º **Condición de Stop.** SDA en flanco de subida mientras que SCL permanece a nivel alto. Ésta es la condición que indica el fin de la transferencia.

Las condiciones de Start y Stop son siempre generadas por el microcontrolador maestro. El bus I2C se considera ocupado después de la condición de Start. El bus se considera libre de nuevo después de un cierto tiempo tras la condición de Stop. Los dispositivos esclavos compatibles con bus I2C deben poseer el hardware de acoplamiento necesario que le permita detectar las condiciones de Start y Stop.

## 21.5 TRANSFERENCIA DE DATOS

Cada dato enviado por la línea SDA debe tener 8 bits (un byte). El número de bytes que se puede enviar no tiene restricción. El byte de datos se transfiere empezando por el bit 7 que es de mayor peso, denominado MSB (*Most Significant Bit*).

Una vez que se han transmitido los 8 bits el receptor deberá mandar un bit de asentimiento o reconocimiento (*acknowledgement*) en el noveno pulso de reloj. El receptor ejecuta este reconocimiento poniendo la señal SDA a un nivel bajo (figura 21-5). Cada grupo de 8 bits debe ser asentido.

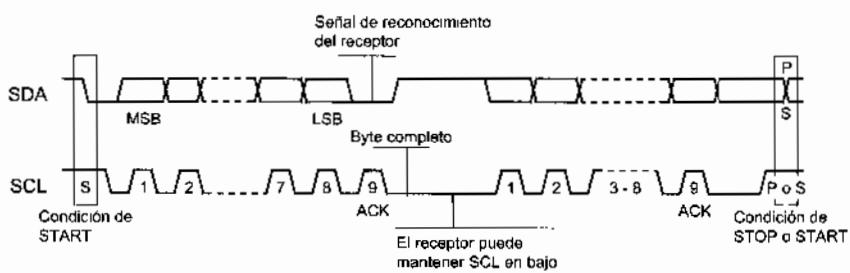


Figura 21-5 Transferencia de datos por el bus I2C

El bit de reconocimiento ACK es obligatorio en la transferencia de datos. El pulso de reloj SCL correspondiente al bit de reconocimiento es generado por el maestro. El transmisor deja libre la línea SDA (la pone en alta impedancia o a nivel alto). El receptor ha de hacer que la línea SDA pase a nivel bajo estable durante el periodo alto del noveno impulso de reloj SCL (figura 21-6).

Si el esclavo-receptor que esta direccionado no desea recibir mas bytes el maestro debe detectar la situación y no enviarle más datos. Esto se indica porque el esclavo no

genera el bit ACK del último byte quedando la línea SDA a nivel alto, lo cual es detectado por el maestro que puede generar la condición Stop y aborta la transferencia de datos.

No es imprescindible enviar una condición de Stop para abortar una transferencia de datos. Si se repite la condición de Start se aborta la transferencia de datos anterior y se comienza una nueva.

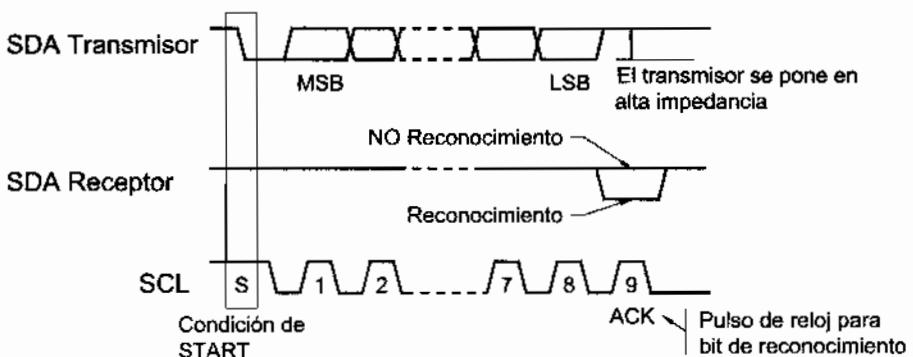


Figura 21-6 Reconocimiento de datos en el bus I2C

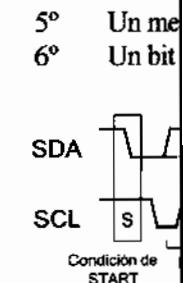
Si un maestro-receptor está recibiendo datos de un esclavo-transmisor, debe generar un bit ACK tras cada byte recibido de transmisor. Para finalizar la transferencia de datos no debe generar el bit ACK tras el último byte enviado por el esclavo. El esclavo-transmisor debe dejar libre la línea de datos SDA para permitir que el maestro genere la condición de Stop o de Start.

Si un dispositivo esclavo no puede recibir o transmitir un byte de datos completo hasta que haya acabado alguna de sus operaciones internas, puede mantener la línea SCL a nivel bajo lo que fuerza al maestro a permanecer en un estado de espera (figura 21-5). La transferencia de datos se reanudará cuando el dispositivo esclavo se encuentre listo para otro byte de datos y desbloquee la línea de reloj SCL.

## 21.6 FORMATO DE UNA TRANSFERENCIA DE DATOS

Los datos transferidos tienen la forma de la figura 21-7, donde se aprecia que para operar un esclavo sobre el bus I2C son necesarios seis pasos para enviar o recibir información:

- 1º Un bit de Start, que señala el inicio de la transferencia de datos.
- 2º Siete bits de direccionamiento de un esclavo.
- 3º Un bit de lectura/escritura (R/W) que define si el esclavo es transmisor o receptor.
- 4º Un bit de reconocimiento ACK (*acknowledgement*).



Puesto q  
mediante una d  
(*slave address*)  
los esclavos e  
se dirige el m  
primer byte de  
maestro (figura

Los 7 pr  
(R/W) determin

- Si R/W informa
- Si R/W esclavo

Cuando  
de Start cada d  
propia. El que  
un esclavo-rece

## 21.7 TIPO

Los posi

vel alto, lo cual es  
a la transferencia de

ar una transferencia  
e datos anterior y se

se pone en  
ia

de reloj para  
reconocimiento

o-transmisor, debe  
zar la transferencia  
por el esclavo. El  
uir que el maestro

de datos completo  
ntener la línea SCL  
spera (figura 21-5).  
se encuentre listo

## DE DATOS

se aprecia que para  
ra enviar o recibir

os.  
o es transmisor o

- 5º Un mensaje dividido en bytes (8 bits).
- 6º Un bit de Stop, que indica el fin de la comunicación.

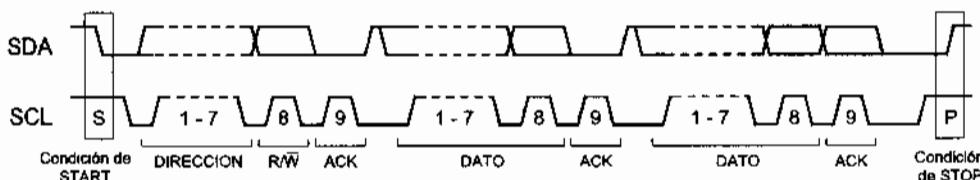


Figura 21-7 Transferencia completa de datos

Puesto que puede haber varios esclavos conectados al bus, el maestro debe indicar mediante una dirección de esclavo a cuál de ellos va destinada la información a transferir, (*slave address*). Cada esclavo tiene una única dirección con la que es identificado. Todos los esclavos están escuchando continuamente la línea para reconocer si es a ellos a quien se dirige el maestro. El procedimiento de dirección para el bus I2C establece que el primer byte después de la condición de Start determina el esclavo seleccionado por el maestro (figura 21-8).



Figura 21-8 Primer byte después del bit de Start

Los 7 primeros bits del primer byte marcan la dirección del esclavo. El octavo bit (R/W) determina la dirección de los datos:

- Si R/W=0, el esclavo es receptor. Significa que el maestro escribirá información en el esclavo seleccionado.
- Si R/W=1, el esclavo es emisor. Significa que el maestro leerá información del esclavo seleccionado.

Cuando un microcontrolador maestro envía una dirección después de la condición de Start cada dispositivo comprueba los siete primeros bits de la dirección con la suya propia. El que coincide se considera el dispositivo seleccionado por el maestro, que será un esclavo-receptor o esclavo-emisor dependiendo del bit R/W.

## 21.7 TIPOS DE FORMATOS DE TRANSFERENCIA

Los posibles formatos de transferencia son:

- 1º **Maestro-emisor transmite al esclavo-receptor.** Si el bit 8 es de escritura, (R/W=0), la secuencia será la descrita en la figura 21-9:

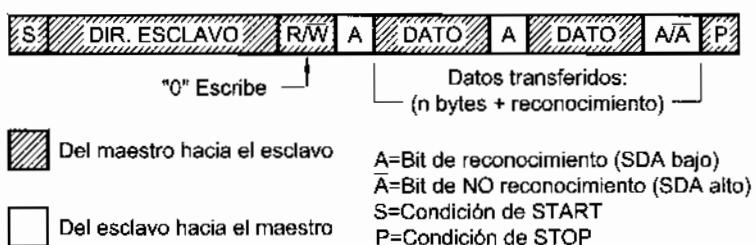


Figura 21-9 Maestro escribe datos en el esclavo

- 2º **Maestro-receptor lee de un esclavo-emisor** inmediatamente después del primer byte. Si el bit 8 es de lectura, (R/W=1), la secuencia será la descrita en la figura 21-10:

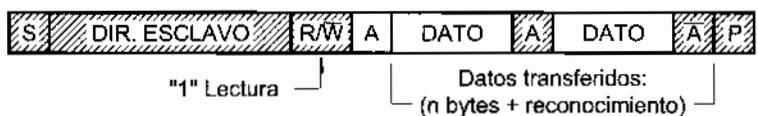


Figura 21-10 Maestro lee datos del esclavo

- 3º **Formato combinado.** Una transferencia de datos siempre acaba con una condición de Stop generada por el maestro. Sin embargo, si un maestro todavía desea comunicarse con el bus puede generar repetidamente condiciones de Start y dirigir a otro esclavo sin generar primero la condición de Stop (figura 21-11). Durante un cambio de dirección dentro de una transferencia, la condición de Start y la dirección del esclavo se repiten, pero con el bit R/W invertido.

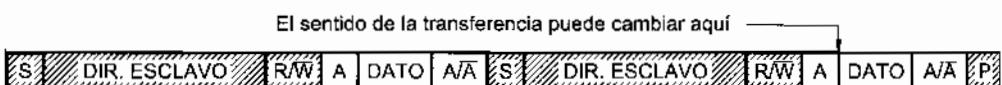


Figura 21-11 Formato combinado

Varias combinaciones de lectura y escritura son posibles dentro de una misma transferencia de datos.

## 21.8 TEMPORIZACIÓN

Los tipos de transferencia de datos en el bus se clasifican:

- *Standard Mode.* Modo estándar, aproximadamente a 100 kbits por segundo.

- *Fast*
  - *High Speed*
- El modo de alta velocidad se describe en la figura 21-12.
- $t_{HIGH}$
  - $t_{LOW}$
  - $t_{HDS}$
  - $t_{SU;S}$
  - $t_{HD;D}$
  - $t_{baja}$
  - $t_{SU;D}$
  - $t_{en l}$
  - $t_{desp}$
  - $t_{BUF}$
  - $t_{nuev}$

SDA

SCL

## 21.9 COMUNICACIONES I2C

La figura 21-12 muestra la secuencia de operación de un sistema I2C y un PIC16F84. El maestro (SDA) controla el bus mediante sus líneas SDA y SCL.

Para comunicarse con el esclavo, el maestro debe permitir la ejecución de las subrutinas correspondientes.

es de escritura,

P  
]

jo)  
A alto)

después del primer  
ta en la figura 21-

A P

acaba con una  
maestro todavía  
ciones de Start y  
op (figura 21-11).  
condición de Start

DATO AA P

ro de una misma

por segundo.

- *Fast Mode*. Modo rápido, aproximadamente a 400 kb/s.
- *High-Speed Mode*. Modo alta velocidad, más de 3,4 Mb/s.

El modo estándar es el más común y sus tiempos requeridos vienen reflejados en la figura 21-12. Los valores mínimos que pueden tener estos tiempos son:

- $t_{HIGH} > 4 \mu s$ . Duración del semiperíodo alto del reloj SCL.
- $t_{LOW} > 4,7 \mu s$ . Duración del semiperíodo bajo del reloj SCL.
- $t_{HD,STA} > 4 \mu s$ . En condición de *Start*, tiempo que tiene que transcurrir entre el flanco de bajada de la línea SDA y el flanco de bajada de la línea SCL.
- $t_{SU,STO} > 4,7 \mu s$ . En la condición de *Stop*, tiempo entre el flanco de subida de la línea SCL y el flanco de subida de la línea SDA.
- $t_{HD,DAT} > 5 \mu s$ . Tiempo de mantenimiento del dato. Tiempo entre el flanco de bajada del reloj SCL y el cambio del dato en la línea SDA.
- $t_{SU,DAT} > 250 \text{ ns}$ . Tiempo de puesta del dato. Tiempo entre el cambio de datos en la línea SDA y el próximo flanco de subida del reloj SCL. Se suele despreciar.
- $t_{BUF} > 4,7 \mu s$ . Tiempo en que el bus debe estar libre antes de comenzar una nueva transmisión.

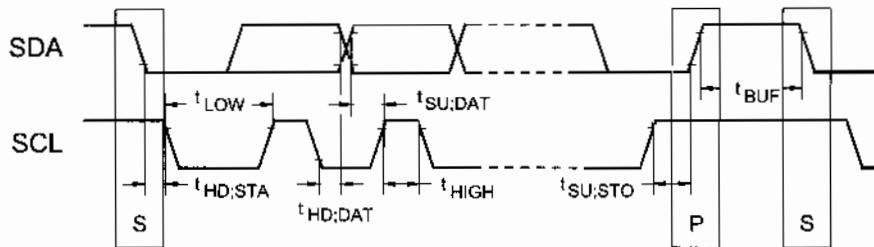


Figura 21-12 Definición de los tiempos para el bus I2C

## 21.9 CONEXIÓN DE BUS I2C A UN PIC16F84

La figura 21-13 describe un ejemplo de conexión de dispositivos conectables a bus I2C y un PIC16F84A configurado como maestro que utiliza las líneas RA3 (SCL) y RA4 (SDA) como líneas del bus I2C. Los esclavos periféricos se conectan al bus a través de sus líneas SCL y SDA.

Para que funcione correctamente el sistema, el programa del microcontrolador debe permitir la comunicación mediante el protocolo para bus I2C recogido en las subrutinas de una librería, como la explicada a continuación.

## 21.10 LIBRERÍA DE SUBRUTINAS PARA BUS I2C

Como resumen de todo el protocolo que rige el bus I2C a continuación se expone la librería BUS\_I2C.INC, con subrutinas sencillas para su control y suficientemente documentadas. El hardware aplicable debe tener la estructura de la figura 21-13. Las subrutinas principales son:

- “I2C\_EviaStart”. Envía una condición de Start o inicio.
- “I2C\_EviaStop”. Envía un condición de Stop o parada.
- “I2C\_EviaByte”. El microcontrolador maestro transmite un byte por el bus I2C que se escribe en el esclavo. El byte a transmitir debe estar cargado previamente en el registro de trabajo W.
- “I2C\_LeeByte”. El microcontrolador lee un byte del esclavo conectado al bus I2C. El dato recibido se envía a la subrutina superior a través del registro W.

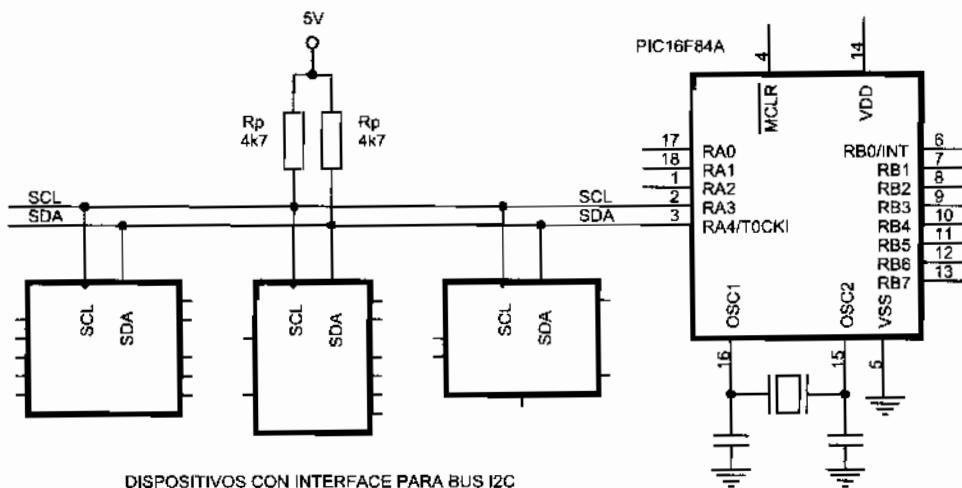


Figura 21-13 El PIC16F84A como maestro de un bus I2C

\*\*\*\*\* Librería "BUS\_I2C.INC" \*\*\*\*\*

; Estas subrutinas permiten realizar las tareas básicas de control del bus serie I2C,  
; por parte de un solo microcontrolador maestro.

\*\*\*\*\* ZONA DE DATOS \*\*\*\*\*

CBLOCK	
I2C_ContadorBits	; Cuenta los bits a transmitir o a recibir.
I2C_Dato	; Dato a transmitir o recibido.
I2C_Flags	; Guarda la información del estado del bus I2C.
ENDC	

#DEFINE I2C\_UltimoByteLeer I2C\_Flags,0  
; - (I2C\_UltimoByteLeer)=0, NO es el último byte a leer por el maestro.

```

; - (I2C_UltimoByteLeer)
; La definición de las líneas
; necesidades del hardware

#define SCL    PO
#define SDA    PO

; Subrutina "SDA_Bajo"
SDA_Bajo
bsf    ST
bcf    SD
bcf    ST
bcf    SD
return

; Subrutina "SDA_Alta"
SDA_Alta
bsf    STA
bsf    SD
bcf    STA
return

; Subrutina "SCL_Bajo"
SCL_Bajo
bsf    STA
bcf    SCL
bcf    STA
bcf    SCL
return

; Subrutina "SCL_Alta"
SCL_Alta
bsf    STA
bsf    SCL
bcf    STA
return

; Subrutina "I2C_EviaStart"
I2C_EviaStart
call   SD
call   SCI
call   Ref
call   SD

***** ZONA DE DATOS *****

; Esta subrutina envía un byte al dispositivo esclavo
; I2C_EviaByte
I2C_EviaByte
call   SD
call   SCI
call   Ref
call   SD

***** ZONA DE DATOS *****

; Subrutina "I2C_LeeByte"
I2C_LeeByte
call   SD
call   SCI
call   Ref
call   SD

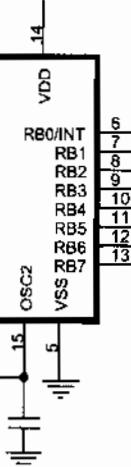
```

2C

uación se expone suficientemente figura 21-13. Las

byte por el bus e estar cargado

conectado al bus el registro W.



\*\*\*\*\*  
\*\*\*\*\*

bus I2C.

```
; - (I2C_UltimoByteLeer)=1, Si es el último byte a leer por el maestro.

; La definición de las líneas SCL y SDA del bus I2C se puede cambiar según las
; necesidades del hardware.

#DEFINE SCL    PORTA,3           ; Línea SCL del bus I2C.
#DEFINE SDA    PORTA,4           ; Línea SDA del bus I2C.

;

; Subrutina "SDA_Bajo" ----

; SDA_Bajo
    bsf    STATUS,RP0          ; Configura la línea SDA como salida.
    bcf    SDA
    bcf    STATUS,RP0
    bcf    SDA
    return ; SDA en bajo.

;

; Subrutina "SDA_AltaImpedancia" ----

; SDA_AltaImpedancia
    bsf    STATUS,RP0          ; Configura la línea SDA entrada.
    bcf    SDA
    bcf    STATUS,RP0
    return ; Lo pone en alta impedancia y, gracias a la
            ; Rp de esta línea, se mantiene a nivel alto.

;

; Subrutina "SCL_Bajo" ----

; SCL_Bajo
    bsf    STATUS,RP0          ; Configura la línea SCL como salida.
    bcf    SCL
    bcf    STATUS,RP0
    bcf    SCL
    return ; La linea de reloj SCL en bajo.

;

; Subrutina "SCL_AltaImpedancia" ----

; SCL_AltaImpedancia
    bsf    STATUS,RP0          ; Configura la linea SCL entrada.
    bcf    SCL
    bcf    STATUS,RP0
    return ; Lo pone en alta impedancia y, gracias a la Rp
            ; de esta línea, se mantiene a nivel alto.

;

; Subrutina "SCL_EsperaNivelAlto"
    btfs  SCL
    goto SCL_EsperaNivelAlto ; Si algún esclavo mantiene esta línea en bajo
    return ; hay que esperar.

;

; Subrutina "I2C_EnviaStart" ----

; Esta subrutina envia una condición de Start o inicio.

; I2C_EnviaStart
    call  SDA_AltaImpedancia ; Línea SDA en alto.
    call  SCL_AltaImpedancia ; Línea SCL en alto.
    call  Retardo_4micros   ; Tiempo tBUF del protocolo.
    call  SDA_Bajo          ; Flanco de bajada de SDA mientras SCL está alto.
```

```

call    Retardo_4micros      ; Tiempo tHD;STA del protocolo.
call    SCL_Bajo              ; Flanco de bajada del reloj SCL.
call    Retardo_4micros      ; De alguna de las
return                                         ; o esta misma I2C

; Subrutina "I2C_EviaStop" -----
; Esta subrutina envía un condición de Stop o parada.

I2C_EviaStop
    call    SDA_Bajo
    call    SCL_AltaImpedancia   ; Flanco de subida de SCL.
    call    Retardo_4micros     ; Tiempo tSU;STO del protocolo.
    call    SDA_AltaImpedancia   ; Flanco de subida de SDA.
    call    Retardo_4micros     ; Tiempo tBUF del protocolo.
    return

; Subrutina "I2C_EviaByte" -----
; El microcontrolador maestro transmite un byte por el bus I2C, comenzando por el bit
; MSB. El byte a transmitir debe estar cargado previamente en el registro de trabajo W.
; De la subrutina ejecutada anteriormente I2C_EviaStart o esta misma I2C_EviaByte,
; la linea SCL se debe encontrar a nivel bajo al menos durante 5 µs.

I2C_EviaByte
    movwf  I2C_Dato          ; Almacena el byte a transmitir.
    movlw  0x08                ; A transmitir 8 bits.
    movwf  I2C_ContadorBits

I2C_EviaBit
    rlf    I2C_Dato,F        ; Chequea el bit, llevándolo previamente al Carry.
    btfsc STATUS,C
    goto  I2C_EviaUno

I2C_EviaCero
    call    SDA_Bajo          ; Si es "0" envía un nivel bajo.

I2C_EviaUno
    call    SDA_AltaImpedancia ; Si es "1" lo activará a alto.

I2C_FlancoSCL
    call    SCL_AltaImpedancia ; Flanco de subida del SCL.
    call    Retardo_4micros    ; Tiempo tHIGH del protocolo.
    call    SCL_Bajo           ; Termina el semiperiodo positivo del reloj.
    call    Retardo_4micros    ; Tiempo tHD;DAT del protocolo.
    decfsz I2C_ContadorBits,F ; Lazo para los ocho bits.
    goto  I2C_EviaBit

; 
    call    SDA_AltaImpedancia ; Libera la linea de datos.
    call    SCL_AltaImpedancia ; Pulso en alto de reloj para que el esclavo
    call    Retardo_4micros    ; pueda enviar el bit ACK.

    call    SCL_Bajo
    call    Retardo_4micros
    return

; Subrutina "I2C_LeeByte" -----
;
```

```

; El microcontrolador
; recibido se carga
; del registro W. S
; De alguna de las
; o esta misma I2C

I2C_LeeByte
    movlw
    movwf
    call

I2C_LeeBit
    call
    bcf
    btfsc
    bsf
    rif
    call
    call
    decfsz
    goto

; Chequea si este es
; ACK en consecuencia
; de la transmisión.

btfs
call
call
call
call
call
movf
return

```

## 21.11 DISPOSITIVOS

Se fabrican y venden:  
ellos destacan:

- 24LC256
- DS1624
- DS1307
- SAA106
- PCF8574
- PCF8591
- PCF8575
- LM76, t<sub>A</sub>
- DS18B20  
En los proyectos
- 74LS244
- ↑ 1 Mbit

; El microcontrolador maestro lee un byte desde el esclavo conectado al bus I2C. El dato recibido se carga en el registro I2C\_Dato y lo envía a la subrutina superior a través del registro W. Se empieza a leer por el bit de mayor peso MSB.  
; De alguna de las subrutinas ejecutadas anteriormente I2C\_EnviaStart, I2C\_EnviaByte o esta misma I2C\_LeeByte, la línea SCL lleva en bajo al menos 5  $\mu$ s.

```
I2C_LeeByte
    movlw 0x08          ; A recibir 8 bits.
    movwf I2C_ContadorBits
    call SDA_AltalImpedancia ; Deja libre la línea de datos.

I2C_LeeBit
    call SCL_AltalImpedancia ; Flanco de subida del reloj.
    bcf STATUS,C           ; En principio supone que es "0".
    btfsc SDA             ; Lee el bit
    bsf STATUS,C           ; Si es "1" carga 1 en el Carry.
    rlf I2C_Dato,F         ; Lo introduce en el registro.
    call SCL_Bajo          ; Termina el semiperíodo positivo del reloj.
    call Retardo_4micros   ; Tiempo tHD;DAT del protocolo.
    decfsz I2C_ContadorBits,F ; Lazo para los 8 bits.

    goto I2C_LeeBit

; Chequea si este es el último byte a leer para enviar o no el bit de reconocimiento ACK en consecuencia.

    btfss I2C_UltimoByteLeer ; Si es el último, no debe enviar
    call SDA_Bajo            ; el bit de reconocimiento ACK.
    call SCL_AltalImpedancia ; Envía el bit de reconocimiento ACK
    call Retardo_4micros     ; porque todavía no es el último byte a leer.
    call SCL_Bajo             ; Pulso en alto del SCL para transmitir el
    call Retardo_4micros     ; bit ACK de reconocimiento. Este es tHIGH.
    movf I2C_Dato,W           ; Pulso de bajada del SCL.

    return                    ; El resultado se manda en el registro de
                            ; de trabajo W.
```

## 21.11 DISPOSITIVOS I2C

Se fabrican multitud de dispositivos que pueden conectarse a un bus I2C. Entre ellos destacan:

- 24LC256, memoria EEPROM serie de 32 kbytes. → 24LC3041
- DS1624, termómetro digital. → LM0161 LCD 5
- DS1307, reloj y calendario de tiempo real.
- SAA1064, driver de display del tipo LED de 4 dígitos. → Sentec CNV70 SRF 04
- PCF8574, interface entre un puerto de 8 líneas y el bus I2C.
- PCF8591, conversor ADC de cuatro canales de 8 bits más 1 canal DAC.
- PCF8576, driver para display LCD. → MAX232
- LM76, termómetro digital y termostato. → 4 canales 4MHz
- D51320  
 En los próximos capítulos se estudiarán con detenimiento algunos de ellos.
- 74LS2138 → 16 filas de salida
- 1 Hobby DC

## CAPÍTULO 22

### 24LC256, MEMORIA EEPROM EN BUS I2C

En algunos proyectos es necesario almacenar gran cantidad de datos en memoria EEPROM, sin embargo, la disponible por los microcontroladores es limitada. Las memorias EEPROM serie permiten solucionar este problema con gran sencillez de hardware ya que en un solo chip de 8 pines pueden almacenar gran cantidad de información.

#### 22.1 MEMORIA EEPROM SERIE 24LC256

La memoria 24LC256 es una popular EEPROM serie que permite almacenar hasta 32 kbytes. Este capítulo se va a desarrollar en torno a este chip en particular, pero todo lo aquí expuesto es fácilmente aplicables a otros modelos de memorias EEPROM serie. Las principales características de la memoria 24LC256 son:

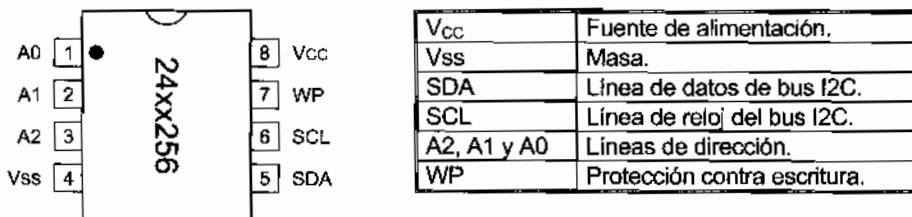


Figura 22-1 Patillaje de la memoria serie 24LC256

- Es una memoria EEPROM serie con interface I<sub>2</sub>C fabricada por *Microchip*, ([www.microchip.com](http://www.microchip.com)) en un encapsulado de 8 pines mostrado en la figura 22-1.
- Tiene una capacidad de 256 kbits o 32 kbytes, es decir, está estructurada en 32 k posiciones de memoria con una longitud de 8 bits cada una.

- Los 32 kbytes están organizados en 128 bloques de 256 bytes cada uno (figura 22-2).
- Es una memoria serie con interfaz I2C. Los datos se escriben y leen en serie a través de los pines SCL y SDA.
- Se puede alimentar con un voltaje de entre 2,5 a 5,5 V, siendo su valor típico 5V.
- Su consumo es muy reducido: 3 mA en proceso de escritura, 400  $\mu$ A en lectura y 100 nA en *standby* o modo de bajo consumo.
- El fabricante garantiza un millón de operaciones de borrado y escritura.
- La retención de datos está garantizada durante 200 años.
- Las líneas de dirección A2, A1 y A0 permiten conectar varias memorias en el mismo circuito, variando la dirección de cada una de ellas.
- Tiene un sistema de protección contra escrituras accidentales mediante el pin WP (*Write Protection*):
  - Si WP se conecta a masa el funcionamiento del chip es normal.
  - Si WP se conecta a  $V_{CC}$  no es posible la escritura de la memoria.

## 22.2 PAGINACIÓN DE LA MEMORIA 24LC256

La memoria está constituida por palabras de 8 bits (1 byte). El número total de bits es de  $32k \times 8$  bits = 256 kbytes. De ahí el “256” de la designación “24LC256”.

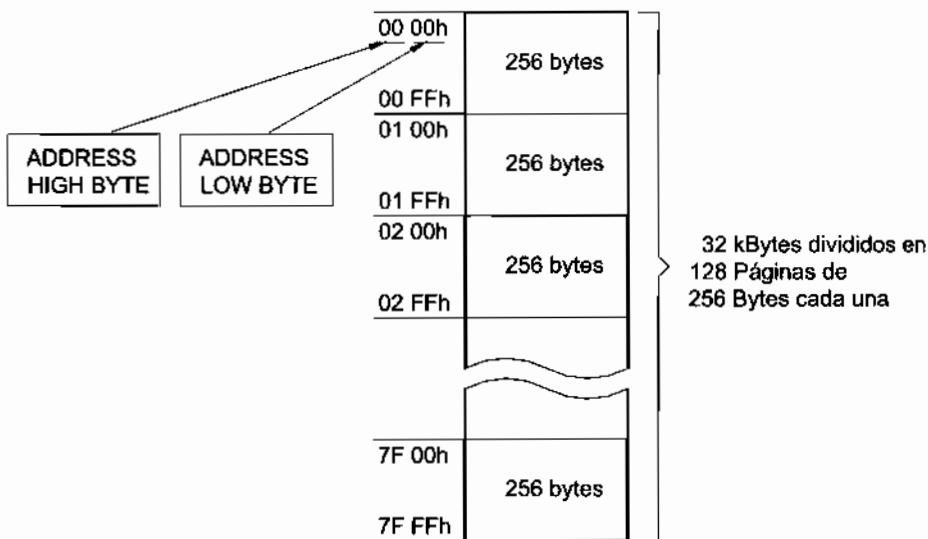


Figura 22-2 Paginación del contenido de la memoria EEPROM 24LC256

Las 32 k de posiciones de memoria están divididas en 128 bloques denominados páginas de 256 posiciones cada una, tal como ilustra la figura 22-2. Cada posición de memoria está identificada por dos bytes de dirección (*Address Byte*):

- El Ad
- El Ad
- págin

Así por  
(02ECh) se lo  
Byte) dentro d

## 22.3 DIR

El direc  
enviarle el mi  
22-3. Esta dire  
define el conex

- Despu
- A cor
- definic
- Finalm
- S
- m
- S

Figura 2

Así por  
en lectura y b  
direcccionamien

## 22.4 COM

La figur  
microcontrolad  
de los pines SC

bytes cada uno (figura  
ben y leen en serie a  
lo su valor típico 5V.  
400  $\mu$ A en lectura y  
y escritura.

arias memorias en el  
s mediante el pin WP  
normal.  
memoria.

56

l número total de bits  
C256".

Bytes divididos en  
páginas de  
bytes cada una

OM 24LC256

bloques denominados  
2. Cada posición de

- El *Address High Byte* que identifica el número de página.
- El *Address Low Byte* que identifica la posición de esa palabra dentro de la página.

Así por ejemplo, la palabra identificada por la posición b'00000010 11101100' (02ECh) se localiza en la página 02h (*Address High Byte*), posición ECh (*Address Low Byte*) dentro de dicha página.

## 22.3 DIRECCIONAMIENTO COMO ESCLAVO

El direccionamiento de esta memoria se rige por el byte de control que debe enviarle el microcontrolador maestro, después de la condición Start, descrito en la figura 22-3. Esta dirección consta de una parte fija y otra programable. La parte programable la define el conexionado de los pines A0, A1 y A2. Consta de:

- Despues del bit de Start, el byte de control comienza por los bits "1010".
- A continuación viene la parte programable que son los bits A2, A1 y A0 definidos por el conexionado de los pines del mismo nombre.
- Finalmente el bit R/W, para indicar si la memoria es escrita o leída:
  - o Si R/W=0, significa que el microcontrolador maestro escribirá en la memoria 24LC256.
  - o Si R/W=1, significa que el maestro leerá un dato de la memoria 24LC256.

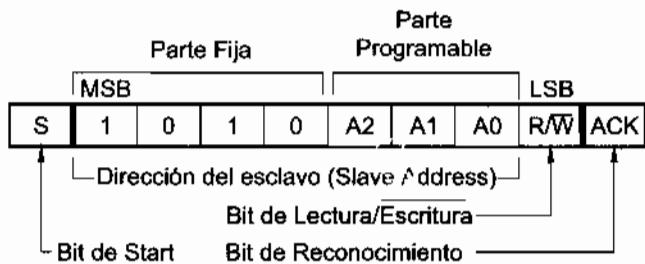


Figura 22-3 Dirección de la memoria 24LC256 como esclavo en un bus I2C

Así por ejemplo, si A2, A1 y A0 se conectan a masa, la dirección es b'10100001' en lectura y b'10100000' en escritura. Esta programación mediante los tres hilos de direccionamiento hardware (A0, A1 y A2) permite conectar hasta ocho memorias.

## 22.4 CONEXIÓN DE UNA 24LC256 A UN PIC16F84

La figura 22-4 ilustra un ejemplo de conexión de una memoria 24LC256 a un microcontrolador PIC16F84A. Se aprecia como el bus I2C lo conforman las conexiones de los pines SCL y SDA de la memoria a líneas RA3 y RA4 del microcontrolador.

Como A2, A1 y A0 se conectan a masa, la dirección para la memoria es b'10100001' en lectura y b'10100000' en escritura.

El pin Wp (*Write Protection*) se conecta a masa para poder realizar la operación de escritura. Si se conectase a V<sub>CC</sub> se evitaría la sobreescritura accidental de la memoria.

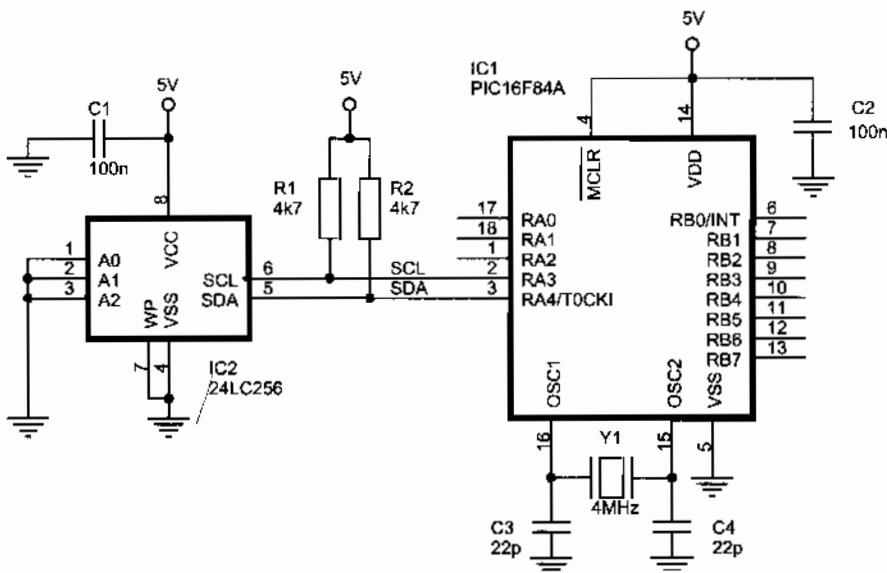


Figura 22-4 Conexión de una memoria EEPROM serie 24LC256 a un PIC16F84A

## 22.5 ESCRITURA EN LA MEMORIA 24LC256

La escritura de una memoria 24LC256 por parte del PIC16F84 sigue el procedimiento de transferencia de datos del maestro al esclavo de un bus I<sub>2</sub>C, descrito en el tema anterior e ilustrado en la figura 22-5. El protocolo de escritura comprende los siguientes pasos:

- Primero el microcontrolador maestro envía la condición de Start.
- Luego envía el byte de control en modo escritura según se ha explicado antes.
- A continuación el maestro envía el número de la página donde se va a escribir, *Address High Byte* (figura 22-2).
- Despues el maestro envía el número de posición dentro de la página seleccionada anteriormente, *Address Low Byte* (figura 22-2).
- Despues se transmiten los datos a escribir. La dirección de la posición de memoria a escribir se incrementa automáticamente.
- Por ultimo el maestro envía la condición de Stop.

La memoria tarda un tiempo máximo de 5 ms en escribir el byte en su interior, por tanto el software debe permitir este tiempo entre la escritura de cada byte.

El número má  
escribir más de 64 by

Para que pued  
no debe estar conecta

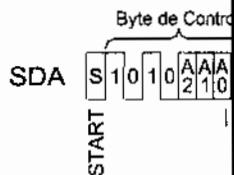


Figura 22-5 P

## 22.6 LECTURA

La lectura de  
protocolo descrito en  
posición a leer, que s  
se desconoce prime  
exactamente su valor

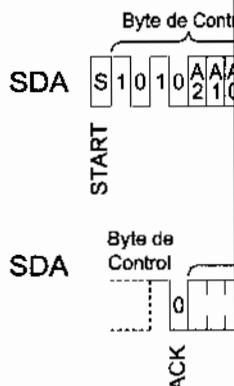
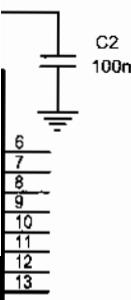


Figura 22-6

- Primero el m  
a escribir (ta  
indicar la dir
- Luego el ma
- A continua

a la memoria es  
zar la operación de  
la memoria.



un PIC16F84A

C16F84 sigue el  
us I2C, descrito en  
ura comprende los

aplicado antes.  
e se va a escribir,

pagina seleccionada

de la posición de

en su interior, por

El número máximo de bytes que se pueden escribir cada vez es de 64. Si se desea escribir más de 64 bytes hay que repetir el procedimiento descrito.

Para que pueda realizarse la operación de escritura, el pin WP (*Write Protection*) no debe estar conectado a V<sub>CC</sub>.

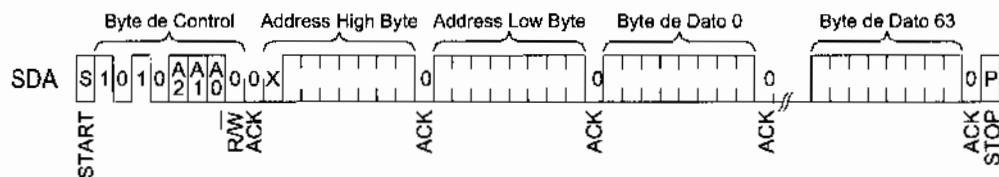


Figura 22-5 Protocolo de escritura en la memoria EEPROM serie 24LC256

## 22.6 LECTURA DE LA MEMORIA 24LC256

La lectura de los datos de la 24LC256 por parte del microcontrolador sigue el protocolo descrito en la figura 22-6 que, en principio, no fija la dirección de la primera posición a leer, que será la siguiente a la última operación realizada. Por ello, si este dato se desconoce primero habrá que realizar una operación de escritura para conocer exactamente su valor. El protocolo de lectura comprende los siguientes pasos:

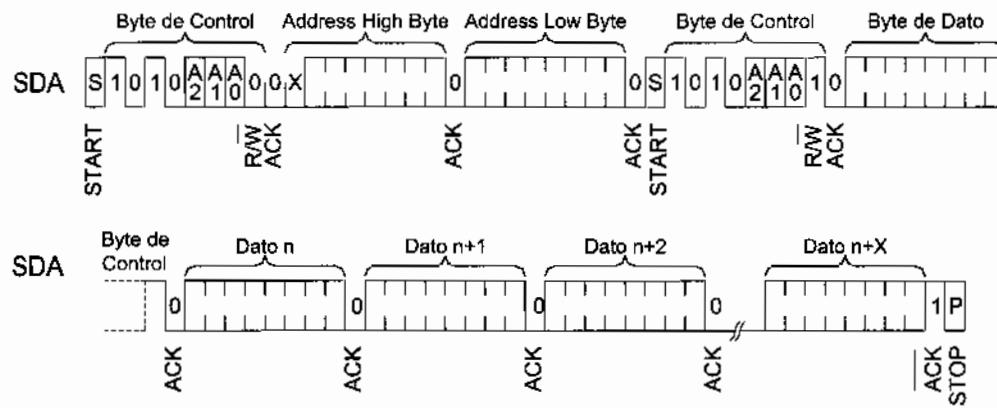


Figura 22-6 Protocolo de lectura de la memoria EEPROM serie 24LC256

- Primero el microcontrolador maestro realiza un envío de dirección como si fuera a escribir (tal como se ha explicado en la sección anterior) con el objeto de indicar la dirección a partir de la cual se va a proceder a la lectura.
- Luego el maestro vuelve a enviar la condición de Start.
- A continuación envía el byte de control en modo lectura.

- Después el maestro lee el contenido de la memoria. La dirección de la posición a leer se incrementa automáticamente.
- Por último el maestro envía la condición de Stop.

## 22.7 LIBRERÍA DE SUBRUTINAS

El protocolo de escritura y lectura en la memoria 24LC256 se concreta en las siguientes subrutinas de control descritas en la librería M24LC256.INC:

- "M24LC256\_InicializaEscritura". Prepara la memoria para iniciar su escritura a partir de una posición de memoria determinada.
- "M24LC256\_InicializaLectura". Prepara la memoria para iniciar su lectura a partir de una posición de memoria determinada.
- "M24LC256\_Mensaje\_a\_LCD". Lee el mensaje grabado en la memoria y lo visualiza por la pantalla del visualizador LCD.

```
;***** Librería "M24LC256.INC" *****
```

```
; Estas subrutinas permiten realizar las tareas de manejo de la memoria EEPROM serie  
; 24LC256 que transmite y recibe la información vía serie a través de un bus I2C.
```

```
; Subrutina "M24LC256_InicializaEscritura"
```

```
; Prepara la memoria para iniciar su escritura a partir de la posición de memoria fijada  
; por los registros:  
; - (M24LC256_AddressHigh), indica el número del bloque o página de memoria a escribir.  
; - (M24LC256_AddressLow), indica posición a escribir dentro del bloque.
```

```
CBLOCK  
M24LC256_AddressHigh ; Guarda el valor de la dirección de memoria a  
M24LC256_AddressLow ; escribir o leer.  
M24LC256_Dato  
ENDC
```

```
M24LC256_DireccionEscritura EQU b'10100000'; Dirección de la memoria 24LC256 para  
M24LC256_DireccionLectura EQU b'10100001'; escritura y lectura respectivamente.
```

```
M24LC256_InicializaEscritura
```

```
call I2C_EnviaStart ; Envía condición de Start.  
movlw M24LC256_DireccionEscritura; Envía dirección de escritura del  
call I2C_EnviaByte ; esclavo.  
movf M24LC256_AddressHigh,W ; A partir de la dirección apuntada por los  
call I2C_EnviaByte ; registros M24LC256_AddressHigh y  
movf M24LC256_AddressLow,W ; M24LC256_AddressLow.  
call I2C_EnviaByte  
return
```

```
; Subrutina "M24LC256_InicializaLectura"
```

```
; Prepara la memoria para iniciar su lectura a partir de la posición de memoria fijada
```

```
; por los registros:  
; - (M24LC256_AddressHigh), indica el número del bloque o página de memoria a leer.  
; - (M24LC256_AddressLow), indica posición a leer dentro del bloque.
```

```
M24LC256_InicializaLectura
```

```
bcf I2C_L  
call I2C_E  
movlw M24LC256_AddressHigh  
call I2C_E  
movf M24LC256_AddressLow  
call I2C_E  
movf M24LC256_AddressHigh  
call I2C_E  
call I2C_E  
;  
call I2C_E  
movlw M24LC256_AddressLow  
call I2C_E  
return
```

```
; Subrutina "M24LC256_Mensaje_a_LCD"
```

```
; Lee el mensaje grabado en la memoria  
; En el registro de trabajo W se almacena la  
; posición se va a leer. La lectura se hace  
; si (W)=2Fh lee el mensaje que se almacena  
; que es la posición 2F00h.
```

```
CBLOCK  
M24LC256_Value  
ENDC
```

```
M24LC256_Mensaje_a_LCD
```

```
movwf M24LC256_Value  
clrf M24LC256_Value  
call M24LC256_Value
```

```
M24LC256_LeeOtroByte
```

```
call I2C_E  
movwf M24LC256_Value  
movf M24LC256_Value  
btfs STAT,5 ; Si el bit 5 de STAT es 1, se lee el byte  
goto M24LC256_Value  
movf M24LC256_Value  
call LCD_Display  
incf M24LC256_Value  
goto M24LC256_Value
```

```
M24LC256_FinMensaje
```

```
call M24LC256_Value  
return
```

```
; Subrutina "M24LC256_FinMensaje"
```

```
; Activa el bit I2C_UltimoBit  
; impide la lectura de la línea SDA y
```

cción de la posición a

56 se concreta en las  
NC:

a iniciar su escritura a

a iniciar su lectura a

o en la memoria y lo

\*\*\*\*\*

eric

da

escribir.

n de memoria a

LC256 para  
mente.

el  
da por los  
High y

; por los registros:  
; - (M24LC256\_AddressHigh), indica el número del bloque o página de memoria a leer.  
; - (M24LC256\_AddressLow), indica posición a escribir dentro del bloque.

#### M24LC256\_InicializaLectura

```
bcf    I2C_UltimoByteLeer      ; Todavía no ha comenzado a leer ningún dato.
call   I2C_EnviaStart        ; Envía condición de Start.
movlw  M24LC256_DireccionEscritura; Envía dirección de escritura del
call   I2C_EnviaByte         ; esclavo.
movf   M24LC256_AddressHigh,W ; A partir de la dirección apuntada por los
call   I2C_EnviaByte         ; registros M24LC256_AddressHigh y
movf   M24LC256_AddressLow,W ; M24LC256_AddressLow.
call   I2C_EnviaByte
call   I2C_EnviaStop

;
call   I2C_EnviaStart        ; Envía condición de Start.
movlw  M24LC256_DireccionLectura ; Indica a la memoria 24LC256 que va a
call   I2C_EnviaByte         ; proceder a la lectura.
return
```

#### ; Subrutina "M24LC256\_Mensaje\_a\_LCD"

```
; Lee el mensaje grabado en la memoria 24LC256 y lo visualiza en el módulo LCD.
; En el registro de trabajo W se introduce la página de la memoria a partir de cuya primera
; posición se va a leer. La lectura termina cuando encuentre el código 0x00. Así por ejemplo,
; si (W)=2Fh lee el mensaje que comienza en la posición 0 de la página 2Fh de la memoria,
; que es la posición 2F00h absoluta.
```

```
; CBLOCK
M24LC256_VvalorCaracter      ; Valor ASCII del carácter leído.
ENDC
```

#### M24LC256\_Mensaje\_a\_LCD

```
movwf  M24LC256_AddressHigh    ; Apunta al inicio de la página correspondiente.
clrf   M24LC256_AddressLow
call   M24LC256_InicializaLectura
```

#### M24LC256\_LeeOtroByte

```
call   I2C_LeeByte            ; Lee la memoria 24LC256.
movwf  M24LC256_VvalorCaracter ; Guarda el valor de carácter.
movf   M24LC256_VvalorCaracter,F ; Lo único que hace es posicionar flag Z. En caso
btfsr  STATUS,Z              ; que sea "0x00", que es código indicador final
goto   M24LC256_FinMensaje
movf   M24LC256_VvalorCaracter,W ; del mensaje, sale de la subrutina.
call   LCD_Caracter          ; Recupera el valor leído.
call   LCD_Caracter          ; Lo visualiza en la pantalla del LCD.
incf   M24LC256_AddressLow,F ; Apunta a la siguiente posición,
goto   M24LC256_LeeOtroByte
```

#### M24LC256\_FinMensaje

```
; call   M24LC_256_FinalizaLectura
; return
```

#### ; Subrutina "M24LC256\_FinalizaLectura"

```
; Activa el bit I2C_UltimoByteLeer para que la subrutina I2C_LeeByte ponga en alta
; impedancia la línea SDA y pueda ejecutarse posteriormente la condición de Start o Stop
```

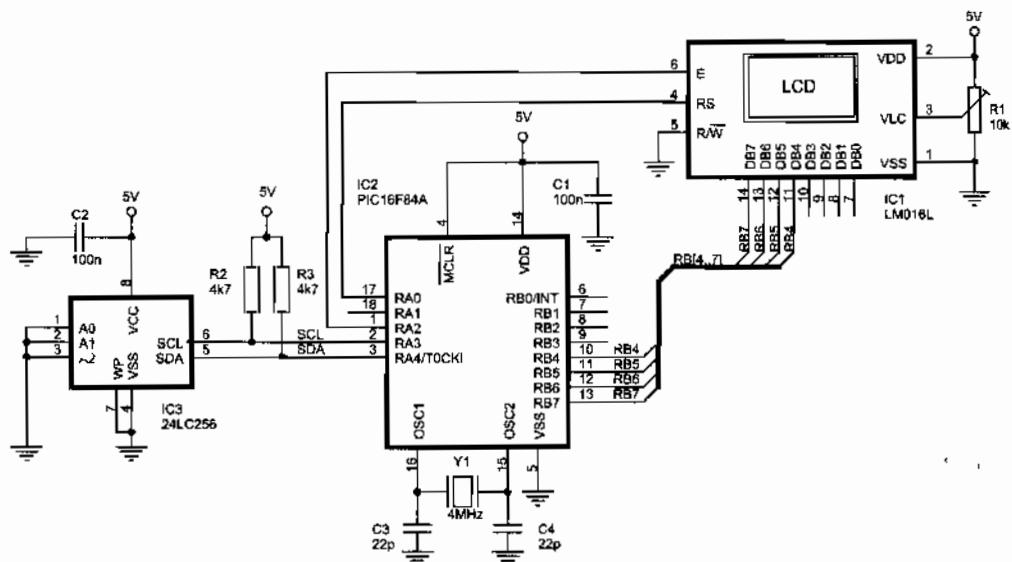
; que fija el protocolo del bus I2C.

#### M24LC256\_FinalizaLectura

```
bsf    I2C_UltimoByteLeer      ; Con estas dos instrucciones se pone en
call   I2C_LeeByte          ; alta impedancia la linea SDA. No importa el
call   I2C_EnviaStop         ; resultado de la lectura realizada.
return
```

## 22.8 EJEMPLO TÍPICO DE APLICACIÓN

El siguiente programa es un ejemplo de aplicación en el manejo de la memoria EEPROM serie 24LC256, para el circuito de la figura 22-7, donde se utiliza el procedimiento típico de escritura y de lectura.



```
LIST      P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK0x0C
ENDC
```

```
Pagina EQU 0x3B ; Escribe y lee en esta página, por ejemplo.
```

```
; ZONA DE CÓDigos ****
```

```
ORG 0
```

Inicio

```
call LCD_Inicializa
```

Principal

```
movlw Pagina ; Escribe a partir de la dirección 0 de esta
call M24LC256_EscribeMensaje ; página de la memoria.
movlw Pagina ; Lee a partir de la dirección 0 de esta página
call M24LC256_Mensaje_a_LCD ; de la memoria.
sleep ; Pasa a reposo.
```

```
; Subrutina "M24LC256_EscribeMensaje"
```

```
; Escribe la memoria M24LC256 con el mensaje grabado al final de este programa a partir
; de la posición 0 de la página de la memoria 24LC256, señalada por el contenido del
; registro de trabajo W.
```

```
; El tamaño de memoria que se puede grabar por este procedimiento es de 64 bytes como máximo.
```

```
CBLOCK
```

```
ValorCaracter
```

```
; Valor ASCII del carácter a escribir.
```

```
M24LC256_EscribeMensaje
```

```
movwf M24LC256_AddressHigh ; El registro W apunta a la página donde va a
clrf M24LC256_AddressLow ; grabar el mensaje a partir de la dirección 0.
call M24LC256_InicializaEscritura
```

```
M24LC256_EscribeOtroByte
```

```
movf M24LC256_AddressLow,W ; Obtiene el código ASCII del carácter.
call Mensaje ; Guarda el valor de carácter que ha leído del
movwf ValorCaracter ; mensaje y lo escribe en la 24LC256.
call I2C_EnviaByte ; Tiempo de escritura entre byte y byte.
call Retardo_5ms ; Lo único que hace es posicionar flag Z. En caso
movf ValorCaracter,F ; de que sea "0x00", que es código indicador final
btfs STATUS,Z ; del mensaje sale de la subrutina.
goto FinEscribeMensaje ; Lo único que hace es posicionar flag Z. En caso
incf M24LC256_AddressLow,F ; de que sea "0x00", que es código indicador final
goto M24LC256_EscribeOtroByte ; del mensaje sale de la subrutina.
```

```
FinEscribeMensaje
```

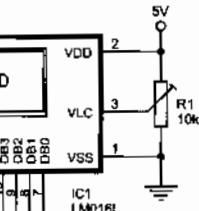
```
call I2C_EnviaStop ; Termina la escritura.
call Retardo_5ms
return
```

```
Mensaje
```

```
addwf PCL,F
DT "Estudia D.P.E.",0x0
```

en  
importa el

rejo de la memoria  
onde se utiliza el



l.asm

ición

en la

\*\*\*\*\*

```

INCLUDE <BUS_I2C.INC> ; Subrutinas de control del bus I2C.
INCLUDE <M24LC256.INC> ; Subrutinas de control de la memoria 24LC256.
INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
END

```

## 22.9 GRABACIÓN DE DATOS MEDIANTE EL IC-PROG

En el programa anterior se explica como grabar datos en la memoria. Sin embargo, puede resultar más rápido grabarlo por el mismo procedimiento utilizado para programar el microcontrolador, usando el grabador TE20-SE y el software IC-Prog.



Figura 22-8 Grabación de mensajes en la 24LC256 mediante el IC-Prog

La figura 22-8 describe el procedimiento para grabar mensajes al comienzo de cada uno de los bloques de la memoria con el software IC-Prog. A destacar que:

- Previamente se ha tenido que elegir el dispositivo adecuado mediante su selección en el menú *Ajustes > Dispositivo > I2C EEPROM > M24C256*.
- Cada uno de los mensajes comienza al inicio cada página de memoria. Es decir, cada mensaje debe comenzar a partir de las direcciones 0000h, 0100h, 0200h, 0300h, ... 7E00h y 7F00h.
- Es más cómodo grabar los mensajes en la zona ASCII de la pantalla de edición.
- Cada uno de los mensajes tiene que acabar con el código 00h. Este código es más cómodo grabarlo en la zona hexadecimal de la pantalla de edición.

Los datos a grabar pueden ser guardados en un fichero mediante la selección del menú *Archivo > Guarda como..* siguiendo el procedimiento típico de Windows.

## 22.10 VISU

Para poder cambiar la subrutina esta otra que se similar analizada

```

; Subrutina "M24LC
; 
; Lee un mensaje gra
; que sea más largo q
; de movimiento. En
; de cuya primera po
; 0x00. Así por ejem
; página 2Fh de la me
;
```

```

CBLOCK
M24LC25
M24LC25
ENDC

```

```

M24LC256_Mensaje
bcf
movwf
clrf
call
M24LC256_Primeran
clrf
call
M24LC256_Visualiza
movlw
subwf
btfs
goto
M24LC256_EsFinal
call
call
call
incf
call
goto
M24LC256_NoEsFin
call
movwf
movf
btfc
goto
M24LC256_NoUltim
call
incf
goto
M24LC256_FinMens

```

goto M

## 22.10 VISUALIZACIÓN DE MENSAJES LARGOS

Para poder visualizar mensajes largos desplazándose por la pantalla hay que cambiar la subrutina M24LC256\_Mensaje\_a\_LCD de la librería M24LC256.INC, por esta otra que se explica a continuación y cuyo funcionamiento coincide con una subrutina similar analizada en el capítulo 13 para la visualización de mensajes en el LCD.

```
; Subrutina "M24LC256_Mensaje_a_LCD"
;
; Lee un mensaje grabado en la memoria 24LC256 y lo visualiza por el módulo LCD. En caso de
; que sea más largo que la longitud de la pantalla se desplaza hacia la izquierda con sensación
; de movimiento. En el registro de trabajo W se introduce la página de la memoria a partir
; de cuya primera posición va a leer. La visualización termina cuando encuentre el código
; 0x00. Así por ejemplo si (W)=2Fh lee el mensaje que comienza en la posición 0 de la
; página 2Fh de la memoria, que es la posición 2F00h absoluta.
;
; CBLOCK
    M24LC256_ValorCaracter          ; Valor ASCII del carácter leído.
    M24LC256_CursorPosicion
    ENDC

M24LC256_Mensaje_a_LCD
    bcf    I2C_UltimoByteLeer
    movwf M24LC256_AddressHigh      ; Apunta al inicio de la página correspondiente.
    clrf    M24LC256_AddressLow
    call    M24LC256_InicializaLectura
M24LC256_PrimeraPosicion
    clrf    M24LC256_CursorPosicion ; El cursor en la posición 0 de la línea.
    call    LCD_Borra              ; Se sitúa en la primera posición de la línea 1 y
                                    ; borra la pantalla.
M24LC256_VisualizaCaracter
    movlw  LCD_CaracteresPorLinea ; ¿Ha llegado a final de línea?
    subwf  M24LC256_CursorPosicion,W
    btfss  STATUS,C
    goto   M24LC256_NoEsFinalLinea
M24LC256_EsFinalLinea
    call    Retardo_200ms          ; Lo mantiene visualizado durante este tiempo.
    call    Retardo_200ms
    call    M24LC256_FinalizaLectura
    incf   M24LC256_AddressLow,F
    call    M24LC256_InicializaLectura
    goto   M24LC256_PrimeraPosicion
M24LC256_NoEsFinalLinea
    call    I2C_LeeByte            ; Obtiene el ASCII del carácter apuntado.
    movwf M24LC256_ValorCaracter ; Guarda el valor de carácter.
    movf   M24LC256_ValorCaracter,F ; Lo único que hace es posicionar flag Z. En caso
    btfsc  STATUS,Z              ; que sea "0x00", que es código indicador final
    goto   M24LC256_FinMensaje   ; de mensaje, sale fuera.
M24LC256_NoUltimoCaracter
    call    LCD_Caracter           ; Visualiza el carácter ASCII leído.
    incf   M24LC256_CursorPosicion,F ; Contabiliza el incremento de posición del
                                    ; cursor en la pantalla.
    goto   M24LC256_VisualizaCaracter ; Vuelve a visualizar el siguiente carácter
M24LC256_FinMensaje
                                    ; de la línea.
```

```
call      M24LC256_FinalizaLectura
return
```

Un ejemplo de aplicación para la subrutina anterior se detalla en el siguiente ejemplo de sorprendente sencillez para la función realizada.

```
***** I2C_EEPROM_02.asm *****
; Lee un mensaje almacenado en la memoria 24LC256 y lo visualiza por la pantalla del módulo
; LCD desplazándose hacia la izquierda.
; El mensaje tiene que haber sido grabado previamente mediante el IC-Prog o un software similar.
; También se puede grabar mediante el procedimiento explicado en el programa I2C_EEPROM_01.asm.
; ZONA DE DATOS *****

_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK0x0C
ENDC

Pagina EQU 0x00
; ZONA DE CÓDIGOS *****

ORG 0
Inicio
Principal
    call LCD_Inicializa
    movlw Pagina ; Lee a partir de la dirección 0 de esta página
    call M24LC256_Mensaje_a_LCD ; de la memoria.
    call Retardo_1s
    goto Principal

INCLUDE <BUS_I2C.INC> ; Subrutinas de control del bus I2C.
INCLUDE <M24LC256.INC> ; Subrutinas de control de la memoria 24LC256.
INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
END
```

## 22.11 CONTROL DE MUCHOS MENSAJES

Con un solo microcontrolador PIC16F84 el número de mensajes que se pueden almacenar es limitado. Una memoria EEPROM serie permite trabajar con gran una cantidad de mensajes, así la 24LC256 permite almacenar hasta 32 k de caracteres, ¡32.768 caracteres!

El siguiente programa ejemplo explica como trabajar hasta con 128 mensajes de 256 caracteres cada uno. Para ello, todos los mensajes tienen que haber sido grabados

previamente en programa I2C mensajes haya figura 22-8. Des el programa en f

```
*****
; Lee los mensajes g
; en la pantalla del m
; Los mensajes han t
; software específico
; I2C_EEPROM_01.
; - El mensaje 0, a p
; - El mensaje 1, a p
; - El mensaje 2, a p
; (y así suce
; - El mensaje 127, a
; La longitud de cada
; una página de la me
; ZONA DE DATOS
_CONFIG
LIST
INCLUDE
CBLOCK0
Apuntador
ENDC

UltimoMensaje B
; ZONA DE CÓDIGO
ORG 0
Inicio
Principal
    call L
    clrf A
    movf A
    call M
    call R
    incf A
    movf A
    sublw U
    btfss ST
    clrf A
    goto PR
```

previamente en la memoria, bien con el IC-Prog o por el procedimiento explicado en el programa I2C\_EEPROM\_01.asm. En este ejemplo es necesario que cada uno de los mensajes haya sido grabado al inicio de cada bloque de memoria, tal como detalla la figura 22-8. Destaca la aparente sencillez de programación que permite variar fácilmente el programa en función de las necesidades del proyecto.

\*\*\*\*\* [2C EEPROM 03.asm] \*\*\*\*\*

Lee los mensajes grabados en las 10 primeras páginas de la memoria 28LC256 y los visualiza en la pantalla del módulo LCD.

Los mensajes han tenido que ser previamente grabados en la memoria 24LC256 con algún software específico como el IC-Prog, o mediante el procedimiento explicado en el programa I2C\_EEPROM\_01.asm, de la siguiente forma:

- El mensaje 0, a partir de la dirección 0000h.
  - El mensaje 1, a partir de la dirección 0100h.
  - El mensaje 2, a partir de la dirección 0200h.

(y así sucesivamente hasta un máximo de..)

- ; - El mensaje 127, a partir de la dirección 7F00h.

La longitud de cada mensaje está limitada por la extensión de los 128 bytes que ocupa una página de la memoria 24LC256. Cada mensaje debe terminar con el código 0x00.

## ZONA DE DATOS \*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC  
LIST    P16F84A  
INCLUDE <P16F84A.INC>
```

**CBLOCK0x0C** Apuntador ; Apunta al número de página donde se almacenan  
ENDC ; cada uno de los mensajes en la 24LC256.

**UltimoMensaje** EQU .10 ; Señala el número del último mensaje.

## ZONA DE CÓDIGOS \*\*\*\*

	ORG	0	
Inicio	call	LCD_Inicializa	
	clrf	Apuntador	; Inicializa el contador
Principal	movf	Apuntador,W	; Apunta al inicio de cada mensaje ya que esta
	call	M24LC256_Mensaje_a_LCD	; subrutina lo carga en (M24LC256_AddressHigh).
	call	Retardo_2s	; Visualiza el mensaje durante este tiempo.
	incf	Apuntador,F	; Apunta al siguiente mensaje.
	movf	Apuntador,W	; Comprueba si ha llegado al último mensaje.
	sublw	UltimoMensaje	; (W)=UltimoMensaje-(Apuntador)
	btfss	STATUS,C	; ¿C=1?, ¿(W) positivo?
	clrf	Apuntador	; Ha resultado UltimoMensaje<(Apuntador).
	goto	Principal	

```

INCLUDE <BUS_I2C.INC>      ; Subrutinas de control del bus I2C.
INCLUDE <M24LC256.INC>    ; Subrutinas de control de la memoria 24LC256.
INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
END

```

Una necesidad  
continuación se expli-  
En el capítulo 28 se e-

### 23.1 EL SENSOR DS1624

El DS1624 es un dispositivo que mide la temperatura leída en grados Celsius. Sus principales son:

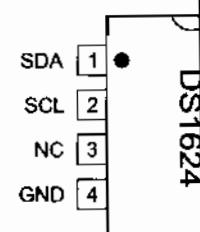


Figura 23.1

- Fabricado por tecnología de silicio en óxido de aluminio.
- Tiene ocho pines con conexión directa a través de una matriz de conexión.
- Es un dispositivo de memoria de alta velocidad.
- No requiere de alimentación externa.
- La temperatura de operación es de -40 a 85°C.

$\text{O}^{\beta 1} \text{lo}^3$

## CAPÍTULO 23

# DS1624, TERMÓMETRO EN BUS I2C

Una necesidad típica de muchos proyectos es la medida de temperatura. A continuación se explica un procedimiento para realizarla con un simple chip de 8 pines. En el capítulo 28 se explicará otro dispositivo con la misma función.

### 23.1 EL SENSOR DE TEMPERATURA DS1624

El DS1624 es un sensor de temperatura que transmite el valor digital de la temperatura leída en el lugar donde se encuentre el dispositivo. Sus características principales son:

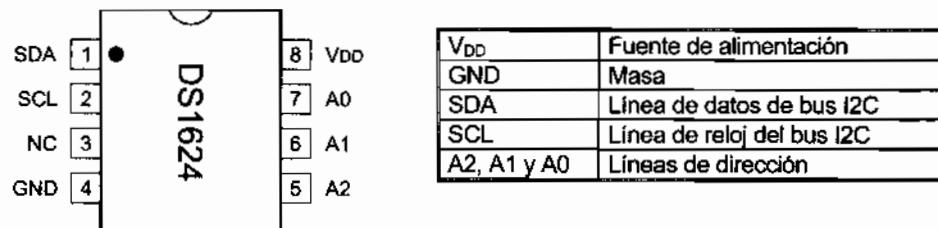


Figura 23-1 Patillaje del sensor de temperatura DS1624

- Fabricado por *Dallas Semiconductors* ([www.dalsemi.com](http://www.dalsemi.com)) en encapsulado de ocho pines como se muestra en la figura 23-1.
- Es un dispositivo conectable a bus I2C, es decir, los datos se escriben y leen en serie a través de los pines SCL y SDA.
- No requiere de componentes externos para realizar la medida de la temperatura.
- La temperatura es leída como un valor digital de 2 bytes que incluye el signo.

- El rango de temperatura que puede medir varía desde -55°C hasta +125°C con variaciones de 0,0625 °C.
- Realiza la conversión de temperatura en un tiempo máximo de un segundo.
- Posee 256 bytes de memoria EEPROM para almacenar datos.
- Se puede alimentar con un voltaje de entre 2,7 a 5,5 V, siendo su valor típico 5V.
- Las entradas de dirección A2, A1 y A0 permiten conectar varios DS1624 en el mismo circuito, variando la dirección de cada uno de ellos.

## 23.2 DIRECCIONAMIENTO COMO ESCLAVO

Como cualquier dispositivo compatible con bus I2C, el DS1624 se activa cuando recibe la dirección válida indicada en la figura 23-2. Esta dirección consta de una parte fija y otra programable. La parte programable la define el conexionado de los pines A2, A1 y A0.

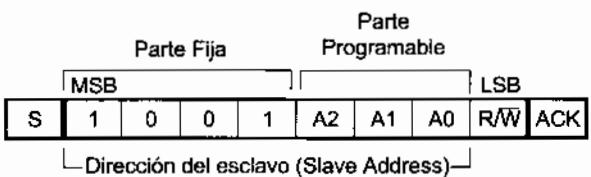


Figura 23-2 Dirección del DS1624 como esclavo en un bus I2C

Así por ejemplo, si A2, A1 y A0 se conectan a masa, la dirección es b'10010001' en lectura y b'10010000' en escritura. Esta programación mediante tres hilos de direccionamiento hardware permite conectar hasta ocho sensores de temperatura DS1624.

## 23.3 LECTURA DE LA TEMPERATURA

El DS1624 realiza la lectura de la temperatura mediante dos bytes en complemento a 2. Este dato es transmitido por el bus I2C, siendo primero el bit MSB en formato descrito en el ejemplo de la figura 23-3.

MSB	LSB							
0	0	0	1	1	0	0	1	0
=+25,0625°C								

Figura 23-3 Formato de lectura de la temperatura

La tabla 23-1 muestra algunos ejemplos que relacionan la temperatura y el contenido de estos dos registros. Se deduce que:

- La temperatura es almacenada en dos bytes de los cuales el byte de los más significativos es igual a "1".
- Los cuatro bytes de los más significativos que son: 01101011.
- La parte decimal de la temperatura es igual a 0,0625.



- La temperatura viene expresada en grados centígrados con cuatro decimales.
- Se almacena en 2 bytes donde el byte superior es la parte entera y el inferior la parte decimal.
- La temperatura está expresada en complemento a 2, por tanto, el bit más alto del byte de los enteros indica el signo, siendo igual a "0" para temperaturas positivas e igual a "1" para temperaturas negativas.
- Los cuatro bits más bajos del byte de la parte decimal están siempre a cero.
- La parte decimal se obtiene sumando el valor posicional de los 4 bits más alto que son: 0,5 °C, 0,25 °C, 0,125 °C y 0,0625 °C, tal como se calcula en los ejemplos de la tabla 23-1.

TEMPERATURA	CÓDIGO DS1624	
	(Parte Entera)	(Parte Decimal)
+125.0000 °C	01111101	00000000
+ 55.0000 °C	00110111	00000000
+ 25.5000 °C	00011001	10000000
+ 25.0625 °C	00011001	00010000
+ 25.0000 °C	00011001	00000000
+ 0.9375 °C	00000000	11110000
+ 0.7500 °C	00000000	11000000
+ 0.6250 °C	00000000	10100000
+ 0.5625 °C	00000000	10010000
+ 0.5000 °C	00000000	10000000
+ 0.2500 °C	00000000	01000000
+ 0.1250 °C	00000000	00100000
+ 0.0625 °C	00000000	00010000
+ 0.0000 °C	00000000	00000000
- 0.5000 °C	11111111	10000000
- 25.0000 °C	11100110	00000000
- 25.0625 °C	11100110	11110000
- 55.0000 °C	11001001	00000000

Tabla 23-1 Ejemplo de lectura de temperaturas en el DS1624

**EJEMPLO 1:** Un DS1624 proporciona el siguiente valor en sus dos bytes: b'00010110 10110000', ¿cuál es el valor de la temperatura medida?

**Solución:** Hay que diferenciar los dos bytes:

• Byte de parte entera: b'00010110' =

$$\begin{aligned}
 &= 128 \times 0 + 64 \times 0 + 32 \times 0 + 16 \times 1 + 8 \times 0 + 4 \times 1 + 2 \times 1 + 1 \times 0 = \\
 &+ 22^{\circ}\text{C}
 \end{aligned}$$

- Byte de la parte decimal:

$$b'10110000' = 0,5 \times 1 + 0,25 \times 0 + 0,125 \times 1 + 0,0625 \times 1 = 0,6875 \text{ } ^\circ\text{C}.$$

Así pues, la temperatura medida será:  $+22,6875 \text{ } ^\circ\text{C}$

**EJEMPLO 2:** Una vez medida la temperatura el DS1624 proporciona en siguiente valor en sus dos bytes:  $b'11110110\ 10010000'$ . ¿Cuál es el valor de la temperatura?

**Solución:** Como el bit más alto es "1" significa que la temperatura es negativa. Para obtener el valor absoluto hay que negar aritméticamente el dato:

Negando  $11110110\ 10010000$  resulta  $00001001\ 01110000$ . Por tanto:

- Byte de parte entera:  $b'00001001' =$

$$= 128 \times 0 + 64 \times 0 + 32 \times 0 + 16 \times 0 + 8 \times 1 + 4 \times 0 + 2 \times 0 + 1 \times 1 = \\ + 9 \text{ } ^\circ\text{C}$$

- Byte de la parte decimal:

$$b'01110000' = 0,5 \times 0 + 0,25 \times 1 + 0,125 \times 1 + 0,0625 \times 1 = 0,4375 \text{ } ^\circ\text{C}$$

Así pues, la temperatura medida será:  $-9,4375 \text{ } ^\circ\text{C}$

## 23.4 REGISTRO DE CONTROL

El DS1624 posee el denominado **registro de control** cuyo contenido determina el modo de trabajo del DS1624 en función de cómo se fijen los bits de la figura 23-4.

DONE	1	0	0	1	0	1	1SHOT
------	---	---	---	---	---	---	-------

Figura 23-4 Registro de control

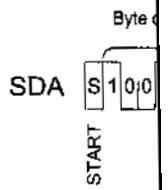
- Bit *DONE*. Es el bit de conversión:
  - o Si *DONE*=0, la medida de temperatura todavía está en proceso.
  - o Si *DONE*=1, la conversión ha finalizado
- Bit *1SHOT*. Es el bit que determina el modo de conversión:
  - o Si *1SHOT*=0, el DS1624 realiza constantemente la conversión de temperatura.
  - o Si *1SHOT*=1, sólo se realiza una conversión desde que se recibe el comando de comienzo de conversión *Start Convert*.

Este registro está implementado en EEPROM, por tanto, la escritura del mismo requiere de unos 10 ms para que se considere completada.

## 23.5 COMANDOS

Los bytes siguientes corresponden al protocolo del DS1624. Los bytes se envían después de enviar el comando *Start Convert*. El DS1624 no responde a estos bytes.

Escribir en el registro de control:



Leer del DS1624:

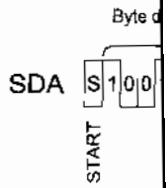


Figura 23-5 Comandos

### INSTRUCCIONES

Read Temperature

Start Convert T

Stop Convert T

Access Memory

Access Config

Se distingue

= 0,6875 °C.

24 proporciona en  
es el valor de la

eratura es negativa.

tanto:

0 + 1 x 1 =

= 0,4375 °C

tenido determina el  
figura 23-4.

1SHOT

oceso.

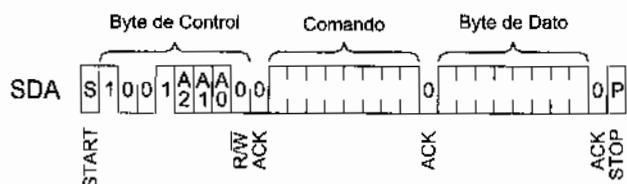
la conversión de  
que se recibe el

escritura del mismo

## 23.5 COMANDOS

Los bytes de datos y control son leídos y escritos en el DS1624 siguiendo el protocolo del bus I2C, según el formato descrito en la figura 23-5. En la escritura, después de enviar el byte de dirección, el microcontrolador maestro debe enviar al DS1624 alguno de los bytes de "comando" indicados en la tabla 23-2.

Escribir en el DS1624



Ler del DS1624

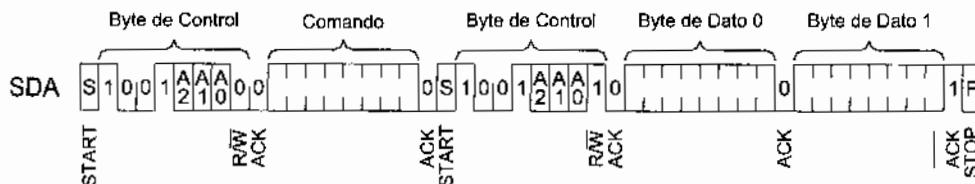


Figura 23-5 Protocolo de comunicación del DS1624 mediante bus I2C

INSTRUCCIÓN	DESCRIPCIÓN	COMANDO
<i>Read Temperature</i>	Lee el valor de temperatura realizado en la última conversión.	0AAh
<i>Start Convert T</i>	Inicia la conversión de temperatura.	0EEh
<i>Stop Convert T</i>	Detiene la conversión de temperatura.	22h
<i>Access Memory</i>	Lee o escribe en los 256 bytes de memoria interna EEPROM	17h
<i>Access Config</i>	Lee o escribe la palabra de configuración en el registro de control. En escritura, la palabra de control escrita es la siguiente al byte de comando.	0ACh

Tabla 23-2 Comandos del DS1624

Se distinguen dos casos para realizar la lectura de la temperatura:

- En el modo de conversión **continuo**, la conversión comienza a ejecutarse desde que se envía el comando *Start Convert T*. Las conversiones se realizan una detrás de otra y pueden ser leídas en cualquier momento. Para finalizar hay que enviar el comando *Stop Convert T*.
- En el modo de conversión **One-Shot**, se debe enviar un comando *Start Convert T* cada vez que se desee realizar la lectura de la temperatura. Cuando acaba la conversión el DS1624 pasa a modo de bajo consumo.

Una vez iniciada la conversión de temperatura mediante el comando *Start Convert T*, el microcontrolador maestro debe leer el bit DONE del registro de control para conocer si ha terminado la conversión y proceder a la lectura del DS1624. Esto no es necesario si la lectura se realiza en un tiempo mayor de un segundo después de iniciada la conversión.

## 23.6 LIBRERÍA DE SUBRUTINAS

El control del sensor de temperatura DS1624 puede concretarse en las subrutinas incluidas en la librería DS1624.INC:

- "DS1624\_Inicializa". Configura el DS1624 para que trabaje en modo *One-Shot*.
- "DS1624\_IniciaConversion" Inicializa el DS1624 para que comience la conversión de temperatura.
- "DS1624\_LeeTemperatura". Proporciona tres datos:
  - Valor absoluto de la temperatura en el registro DS1624\_Temperatura.
  - Parte decimal de la temperatura en el registro DS1624\_Decimal.
  - Signo de la temperatura leída en el registro DS1624\_Signo.

```
***** Librería "DS1624.INC" *****
;
; Estas subrutinas permiten realizar las tareas de manejo del sensor de temperatura
; DS1624. Este sensor transmite la información vía serie a través de un bus I2C.
;
; ZONA DE DATOS *****
CBLOCK
  DS1624_Temperatura          ; Parte entera de la temperatura medida.
  DS1624_Decimal              ; Parte decimal de la temperatura medida.
  DS1624_Signo                ; Signo de la temperatura medida.
ENDC

DS1624_DireccionEscritura EQU b'10010000'
DS1624_DireccionLectura   EQU b'10010001'

Comando_ReadTemperature  EQU 0AAh      ; Comandos del DS1624.
Comando_StartConvert_T   EQU 0EEh
Comando_AccessConfig     EQU 0ACH
```

```
; Subrutina "DS1624_Inicializa"
; Esta subrutina redondea el resultado de la conversión
; con un decimal indicado por el parámetro d
DS1624_Inicializa
    addwf PC, f
    retlw d0
    retlw d1
    retlw d2
    retlw d3
    retlw d4
    retlw d5
    retlw d6
    retlw d7
    retlw d8
    retlw d9
    retlw d10
    retlw d11
    retlw d12
    retlw d13
    retlw d14
    retlw d15
    retlw d16
    retlw d17
    retlw d18
    retlw d19
    retlw d20
    DS1624_FinTablaRedondeo
    IF (DS1624_Error = 0)
        MBR = DS1624_Modo
    ENDIF
;
; Subrutina "DS1624_IniciaConversion"
; Configura el DS1624 para modo One-Shot
DS1624_IniciaConversion
    movlw DS1624_DireccionEscritura
    call I2C_Start
    movlw Comando_StartConvert_T
    call I2C_Write
    movlw b'01
    call I2C_Stop
    call I2C_Start
    call DS1624_SetMode
    ;
    ;
    return
;
; Subrutina "DS1624_LeeTemperatura"
; Inicializa el DS1624 para modo One-Shot
; Configura el DS1624 para modo One-Shot
DS1624_LeeTemperatura
    call I2C_Start
    movlw DS1624_DireccionLectura
    call I2C_Write
    movlw Comando_ReadTemperature
    call I2C_Stop
    call I2C_Start
    call DS1624_SetMode
    ;
    ;
    return
;
```

a ejecutarse desde  
realizan una detrás  
zar hay que enviar

do Start Convert T  
Cuando acaba la

ndo Start Convert  
o de control para  
S1624. Esto no es  
pués de iniciada la

e en las subrutinas

modo One-Shot.  
que comience la

temperatura.  
cimal.

\*\*\*\*\*

ratura medida.  
eratura medida.  
medida.

; Subrutina "DS1624\_RedondeaDecimal"  
; Esta subrutina redondea el valor del contenido del registro de trabajo W al valor más cercano  
; con un decimal indicado en una tabla.

#### DS1624\_RedondeoDecimal

```
addwf PCL,F
retlw d'0' ; (DS1624_Decimal)=b'0000'. Mide 0,0000°C. Redondeado a 0,0°C.
retlw d'1' ; (DS1624_Decimal)=b'0001'. Mide 0,0625°C. Redondeado a 0,1°C.
retlw d'1' ; (DS1624_Decimal)=b'0010'. Mide 0,1250°C. Redondeado a 0,1°C.
retlw d'2' ; (DS1624_Decimal)=b'0011'. Mide 0,1875°C. Redondeado a 0,2°C.
retlw d'3' ; (DS1624_Decimal)=b'0100'. Mide 0,2500°C. Redondeado a 0,3°C.
retlw d'3' ; (DS1624_Decimal)=b'0101'. Mide 0,3125°C. Redondeado a 0,3°C.
retlw d'4' ; (DS1624_Decimal)=b'0110'. Mide 0,3750°C. Redondeado a 0,4°C.
retlw d'4' ; (DS1624_Decimal)=b'0111'. Mide 0,4375°C. Redondeado a 0,4°C.
retlw d'5' ; (DS1624_Decimal)=b'1000'. Mide 0,5000°C. Redondeado a 0,5°C.
retlw d'6' ; (DS1624_Decimal)=b'1001'. Mide 0,5625°C. Redondeado a 0,6°C.
retlw d'6' ; (DS1624_Decimal)=b'1010'. Mide 0,6250°C. Redondeado a 0,6°C.
retlw d'7' ; (DS1624_Decimal)=b'1011'. Mide 0,6875°C. Redondeado a 0,7°C.
retlw d'8' ; (DS1624_Decimal)=b'1100'. Mide 0,7500°C. Redondeado a 0,8°C.
retlw d'8' ; (DS1624_Decimal)=b'1101'. Mide 0,8125°C. Redondeado a 0,8°C.
retlw d'9' ; (DS1624_Decimal)=b'1110'. Mide 0,8750°C. Redondeado a 0,9°C.
retlw d'9' ; (DS1624_Decimal)=b'1111'. Mide 0,9375°C. Redondeado a 0,9°C.
```

#### DS1624\_FinTablaRedondeo

```
IF (DS1624_FinTablaRedondeo > 0xFF)
    ERROR "¡CUIDADO!: La tabla ha superado el tamaño de la página de los"
    MESSG "primeros 256 bytes de memoria ROM. NO funcionará correctamente."
ENDIF
```

#### DS1624\_Inicializa

```
movlw DS1624_DireccionEscritura ; Apunta al dispositivo.
call I2C_EnviaByte
movlw Comando_AccessConfig ; Comando indicando que el próximo byte es la
call I2C_EnviaByte ; palabra de control.
movlw b'01001011' ; Carga la palabra de control para modo One-Shot.
call I2C_EnviaByte
call I2C_EnviaStop
; call DS1624_IniciaConversion ; Inicia la conversión de temperatura.
; return
```

#### DS1624\_IniciaConversion

```
call I2C_EnviaStart ; Configura el DS1624.
movlw DS1624_DireccionEscritura ; Apunta al dispositivo.
call I2C_EnviaByte
movlw Comando_StartConvert_T ; Comando que ordena el comienzo de la conversión
```

```

call    I2C_EEnviaByte      ; de la temperatura.
call    I2C_EEnviaStop
return

```

; Subrutina "DS1624\_LeeTemperatura"

; El DS1624 lee la temperatura en un formato ejemplificado como sigue:

; +125.0000 °C	01111101 00000000
; + 25.0625 °C	00011001 00010000
; + 0.5000 °C	00000000 10000000
; + 0.0000 °C	00000000 00000000
; - 0.5000 °C	11111111 10000000
; - 25.0625 °C	11100110 11110000
; - 55.0000 °C	11001001 00000000

; Se observa que:

- El formato es de dos bytes. El primer byte es la parte entera y el segundo la parte decimal.
- Las temperaturas vienen dadas en complemento a 2.
- Las temperaturas positivas comienzan con el bit MSB a cero: b'0xxxxxxxx xxxx xxxx'.
- Las temperaturas negativas comienzan con el bit MSB a uno: b'1xxxxxxxx xxxx xxxx'.

; Esta subrutina lee la temperatura, proporcionando tres datos:

; Salida: - En (DS1624\_Temperatura) la parte entera del valor de la temperatura medida.  
; - En (DS1624\_Decimal) la parte decimal del valor de la temperatura.  
; - En (DS1624\_Signo)=b'1111111' si la temperatura es negativa y  
; (DS1624\_Signo)=b'00000000' si es positiva.

DS1624\_LeeTemperatura

```

bcf    I2C_UltimoByteLeer
call   I2C_EEnviaStart
movlw  DS1624_DireccionEscritura ; Apunta al dispositivo.
call   I2C_EEnviaByte
movlw  Comando_ReadTemperature ; Comando de lectura de la temperatura.
call   I2C_EEnviaByte
call   I2C_EEnviaStart          ; Comienza a leer.
movlw  DS1624_DireccionLectura ; Apunta al dispositivo.
call   I2C_EEnviaByte
call   I2C_ELeeByte            ; Lee el primer byte
movwf  DS1624_Temperatura      ; y lo guarda.
bsf    I2C_UltimoByteLeer      ; El próximo es el último byte a leer.
call   I2C_ELeeByte            ; Lee el segundo byte
movwf  DS1624_Decimal          ; y lo guarda.
call   I2C_EEnviaStop

```

; Ahora deduce si la temperatura es positiva o negativa y halla su valor absoluto.

```

clrf  DS1624_Signo           ; Supone que la temperatura es positiva.
btfs  DS1624_Temperatura,7   ; Si el bit MSB es "1", la temperatura es negativa.
goto  DS1624_FinLeeTemperatura ; La temperatura es positiva y salta.

```

DS1624\_TemperaturaNegativa

```

comf  DS1624_Signo,F        ; Indica que la temperatura es negativa.
comf  DS1624_Decimal,F      ; Para hallar el valor absoluto de la

```

```

comf  DS
movlw .1
addwf DS
btfsc ST
inef  DS
DS1624_FinLeeTemperatura
swapf DS
andlw b00
call  DS
movwf DS
movf  DS
return

```

## 23.7 TERMÓMETRO

Una aplicación de un termómetro digital DS1624 se detalla a continuación.

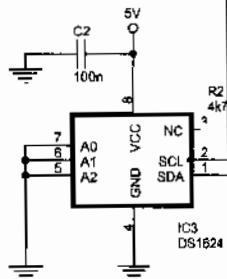


Figura 23-6

\*\*\*\*\*

Programa de control para

ZONA DE DATOS \*\*\*

CONFIG  
LIST P  
INCLUDE <

```

comf   DS1624_Temperatura,F
movlw  .1
addwf  DS1624_Decimal,F
btfc   STATUS,C
incf   DS1624_Temperatura,F
DS1624_FinLeeTemperatura
swapf  DS1624_Decimal,W
andiw  b'00001111'
call   DS1624_RedondeoDecimal
movwf  DS1624_Decimal
movf   DS1624_Temperatura,W
return

```

; temperatura invierte los dos registros y le  
; suma una unidad.

; Si hay acarreo tiene que llevarlo al byte superior.  
; Le suma 1.

; Para las temperaturas es suficiente trabajar  
; con un solo dígito decimal, por lo que  
; primero se queda con los 4 bits menos  
; significativo y pasa a redondear su valor.  
; En (W) la parte entera del valor absoluto de la  
; temperatura.

## 23.7 TERMÓMETRO DIGITAL

Una aplicación de todo lo explicado hasta ahora, es la realización del proyecto de un termómetro digital como el representado en la figura 23-6. Su programa de control se detalla a continuación.

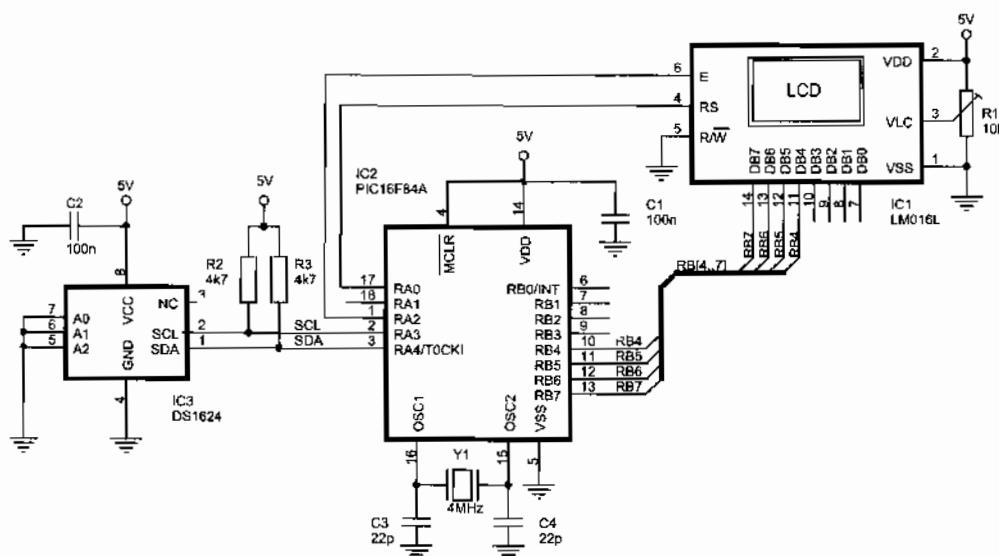


Figura 23-6. Conexión de un DS1624 a un PIC16F84A. Termómetro digital

```

***** I2C_Termometro_01.asm *****
;
; Programa de control para un termómetro digital con el sensor de temperatura DS1624.
;
; ZONA DE DATOS *****

```

```

_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST    P=16F84A
INCLUDE <P16F84A.INC>

```

```

CBLOCK 0x0C
Registro50ms      ; Guarda los incrementos cada 50 ms.
    END C

TMR0_Carga50ms EQU -d'195'   ; Para conseguir interrupción cada 50 ms.
Carga2s          EQU d'40'     ; Leerá cada 2s = 40 x 50ms = 2000ms.

; ZONA DE CÓDIGOS ****
; *****

ORG 0
goto Inicio
ORG 4
goto ServicioInterrupcion

Mensajes
    addwf PCL,F
MensajePublicitario
    DT "IES. ISAAC PERAL", 0x00
MensajeTemperatura
    DT "Temperatura: ", 0x00
MensajeGradoCentigrado
    DT "°C ", 0x00           ; En la pantalla: "°C "

Inicio call DS1624_Inicializa ; Configura el DS1624 e inicia la conversión.
call LCD_Inicializa
bsf STATUS,RP0
movlw b'00000111'
movwf OPTION_REG
bcf STATUS,RP0
movlw MensajePublicitario ; Visualiza un mensaje publicitario en la primera
call LCD_Mensaje           ; línea del LCD.
movlw TMR0_Carga50ms       ; Carga el TMR0 en complemento a 2.
movwf TMR0
movlw Carga2s              ; Y el registro cuyo decremento consigue los 2 s.
movwf Registro50ms         ; Activa interrupción del TMR0 (T0IE), por cambio en
movlw b'10101000'           ; las líneas del Puerto B (RBIE) y la general (GIE).
movwf INTCON

Principal
    goto Principal

; Subrutina "ServicioInterrupcion"
; Detecta qué ha producido la interrupción y ejecuta la subrutina de atención correspondiente.

ServicioInterrupcion
    btfscc INTCON,T0IF
    call Termometro
    bcf INTCON,T0IF
    retfie

; Subrutina "Termometro"
; Esta subrutina lee y visualiza el termómetro cada 2 segundos aproximadamente.

```

```

; Se ejecuta debido
; Para conseguir un
; 50 ms (40x50ms =
;
Termometro
    movlw
    movwf
    decfsz
    goto
    movlw
    movwf
    call
    call
    call
    Fin_Termometro
    return

; Subrutina "Visualizar"
; Visualiza la temperatura
; Entradas: - (DS)
;           - (DS)
;           - (DS)

VisualizaTermometro
    movlw
    call
    btfss
    goto
    TemperaturaNegativa
    movlw
    call
    TemperaturaPositiva
    movf
    call
    call
    movlw
    call
    movf
    call
    movlw
    call
    return

INCLUDE
INCLUDE
INCLUDE
INCLUDE
INCLUDE
INCLUDE
INCLUDE
END

```

```

; Se ejecuta debido a la petición de interrupción del Timer 0, cada 50 ms.
; Para conseguir una temporización de 2 s habrá que repetir 40 veces el lazo de
; 50 ms (40x50ms = 2000ms = 2s).
;

Termometro
    movlw   TMRO_Carga50ms
    movwf   TMRO
    decfsz  Registro50ms,F
    goto   Fin_Termometro
    movlw   Carga2s
    movwf   Registro50ms
    call    DS1624_LecTemperatura
    call    DS1624_IniciaConversion
    call    VisualizaTermometro
Fin_Termometro
return

; Subrutina "VisualizaTermometro"
;
; Visualiza la temperatura en la segunda línea de la pantalla LCD.
;
; Entradas: - (DS1624_Temperatura), temperatura medida en valor absoluto.
;           - (DS1624_Decimal), parte decimal de la temperatura medida.
;           - (DS1624_Signo), indica signo de la temperatura.
;

VisualizaTermometro
    movlw   .5
    call    LCD_PosicionLinea2
    btfss  DS1624_Signo,7
    goto   TemperaturaPositiva
TemperaturaNegativa:
    movlw   '-'
    call    LCD_Caracter
TemperaturaPositiva
    movf   DS1624_Temperatura,W
    call    BIN_a_BCD
    call    LCD_BytE
    movlw   '.'
    call    LCD_Caracter
    movf   DS1624_Decimal,W
    call    LCD_Nibble
    movlw   MensajeGradoCentigrado
    call    LCD_Mensaje
    return

INCLUDE <BUS_I2C.INC>
INCLUDE <DS1624.INC>
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

## CAPÍTULO 24

Examen

### DS1307, RELOJ-CALENDARIO EN BUS I2C

En muchos proyectos es necesario trabajar en tiempo real, por lo que en este capítulo se explicará una forma muy sencilla de hacerlo.

#### 24.1 EL RELOJ-CALENDARIO DS1307

El DS1307 es un reloj en tiempo real RTC (*Real Time Clock*) con líneas de conexión a un bus I2C. Este circuito integrado es un poderoso reloj y calendario de tiempo real, que cumple perfectamente con muchas de las necesidades normales en la adquisición y registro del tiempo. Sus características más destacadas son:

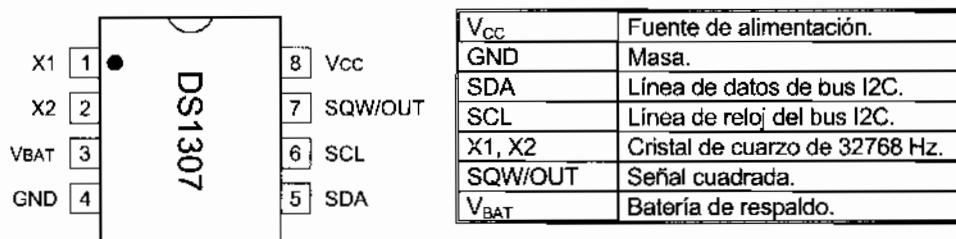


Figura 24-1 Patillaje del reloj-calendario DS1307

- Fabricado por *Dallas Semiconductors* ([www.dalsemi.com](http://www.dalsemi.com)), en encapsulado de 8 pines (figura 24-1)
- El DS1307 es un reloj y calendario de tiempo real que cuenta los segundos, los minutos, las horas, los días de la semana (lunes, martes, etc.), los días del mes, los meses y los años, válido hasta el año 2100.

- Almacena los datos en formato BCD para que se pueda trabajar directamente con ellos.
- Tiene 56 bytes de RAM no volátil para almacenamiento de datos.
- En su pin SQW/OUT proporciona una onda cuadrada programable.
- Tiene una circuitería interna de "respaldo" para alimentación en caso de fallo de la alimentación principal, por tanto, es capaz de mantener el tiempo y la fecha actualizados aún cuando el sistema esté apagado.
- Se puede alimentar entre 4,5 a 5,5 V, siendo su valor típico 5V.
- Posee un bajo consumo, menos a 500 nA en el modo de respaldo.
- Utiliza un cristal de cuarzo propio de 32.768 Hz para lograr tiempos exactos y no depender del microcontrolador.
- El último día del mes es automáticamente ajustado a 28, 29, 30 ó 31 días según corresponda, tiene en cuenta los años bisiestos.
- Puede trabajar en formato europeo de 24 horas o el americano de 12 con indicador de AM/PM.
- Se activa cuando recibe la dirección válida indicada en la figura 24-2.

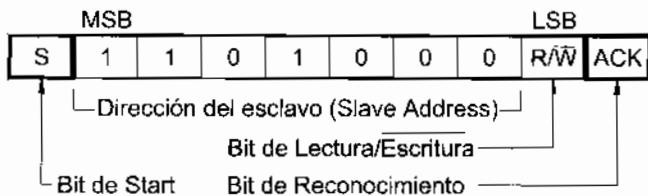
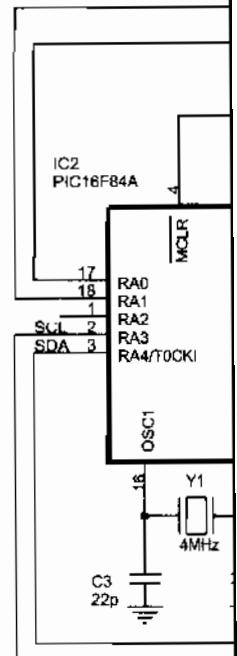


Figura 24-2 Dirección del DS1307 como esclavo en bus I2C



## 24.2 CONEXIÓN DE UN DS1307 A UN PIC16F84

La figura 24-3 ilustra un posible esquema de conexión con un microcontrolador PIC16F84A. Los pines SDA y SDL del DS1307 se conectan a dos líneas del puerto A del microcontrolador conformando el bus I2C. Las resistencias de Pull-Up tienen el valor característico de 4k7.

El DS1307 tiene su propio cristal de cuarzo de 32.768 Hz para lograr tiempos exactos y no depender del microcontrolador. Un condensador ajustable opcional se puede poner en paralelo con el cristal para ajustar el tiempo a valores exactos. Para este ajuste será necesaria la medida con osciloscopio de la señal cuadrada del pin SQW/OUT del DS1307. En la mayoría de las aplicaciones, la precisión del DS1307 y del cristal son suficientes y este condensador no es necesario.

Si falla la alimentación principal, el DS1307 pasa a alimentarse de la batería de respaldo BT1, manteniendo la información del tiempo y la fecha mientras la alimentación principal se mantenga apagada. Si esta batería no se utiliza, los datos de calendario del

DS1307 no se activa. Debe estar comprendido que el fabricante recomienda la conservación de la

salida de interrupción RB0/INT1 configurar esta señal sea interrumpido cuando sea necesaria una LED D1 parpadeando.

La puesta en marcha e "INCREMENTAR" basa en interrupciones.

DS1307 no se actualizarán en caso de fallo de la alimentación. El valor de esta batería debe estar comprendido entre los 2 y 3,5 V. Puede usarse cualquier tipo de pila, aunque el fabricante recomienda una batería de litio de al menos 48 mAh que garantiza una conservación de la información en el DS1307 para más de 10 años.

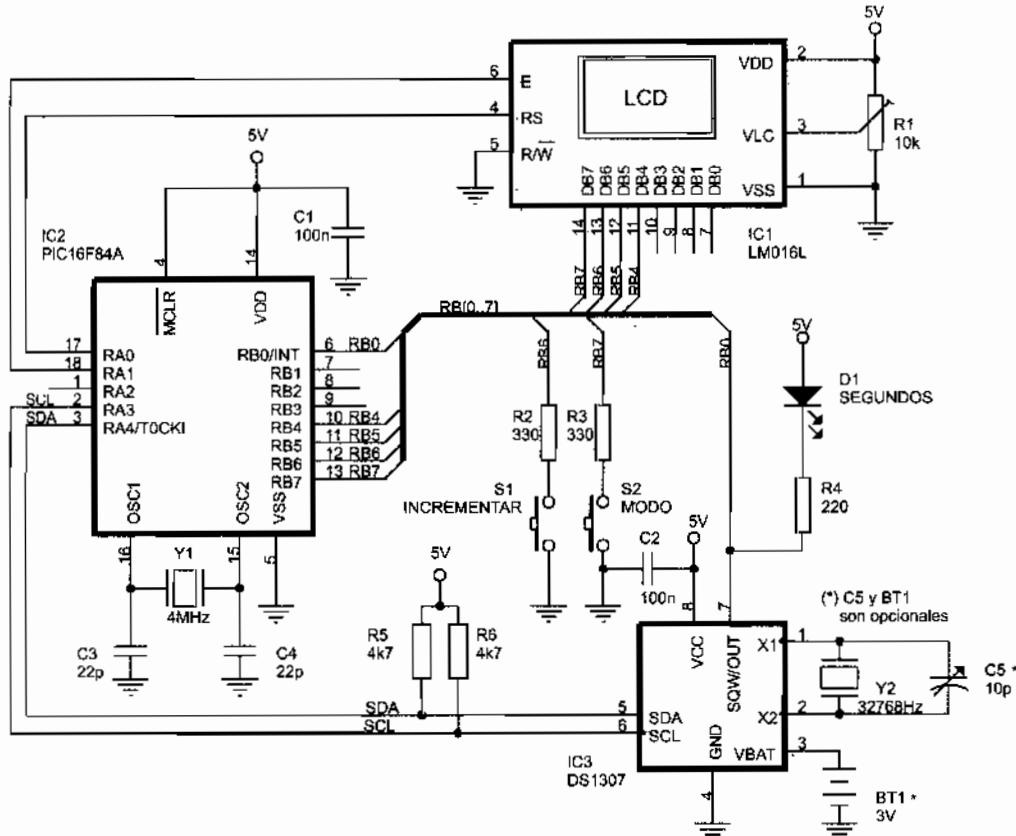


Figura 24-3 Conexión de un DS1307 a un PIC16F84A

La salida de onda cuadrada del DS1307 (SQW/OUT) se conecta a la línea de interrupción RB0/INT del microcontrolador. El programa de control del PIC16F84A debe configurar esta señal cuadrada con una frecuencia de 1 Hz para que el microcontrolador sea interrumpido una vez por segundo. La salida de esta línea es en drenador abierto siendo necesaria una resistencia de Pull-Up para su correcto funcionamiento. El diodo LED D1 parpadeará al ritmo de los segundos.

La puesta en hora de este reloj digital se realiza mediante los pulsadores "MODO" e "INCREMENTAR" que se conectan a pines 7 y 6 del Puerto B. Su funcionamiento se basa en interrupción por cambio en la línea del Puerto B. En serie con estos pulsadores

ectamente con

uso de fallo de  
pó y la fecha

s exactos y no

31 días según

o de 12 con

2.

recontrolador  
l puerto A del  
enen el valor

regar tiempos  
onal se puede  
ra este ajuste  
QW/OUT del  
el cristal son

la batería de  
alimentación  
alendario del

deben conectar las resistencias con la finalidad de que no cortocircuite las líneas RB6 y RB7 cuando se envíen datos al módulo LCD y a la vez se active estos pulsadores.

## 24.3 REGISTROS DEL DS1307

La figura 24-4 muestra el mapa de direcciones de la memoria RAM del DS1307. Los registros del calendario se localizan en las direcciones 00h hasta la 07h. Desde la 08h hasta la 3Fh hay 56 posiciones de memoria RAM que pueden ser utilizadas para almacenar datos.

00h	SECONDS
01h	MINUTES
02h	HOURS
03h	DAY
04h	DATE
05h	MONTH
06h	YEAR
07h	CONTROL
08h	RAM
	56 x 8
3Fh	

Figura 24-4 Mapa de memoria del DS1307

El valor del tiempo y calendario se obtiene mediante la lectura de los registros apropiados. La figura 24-5 muestra los registros del reloj y calendario en detalle, a destacar:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0		
00h	CH	10 SECONDS			SECONDS				00-59	
01h	X	10 MINUTES			MINUTES				00-59	
02h	X	12 24	10 HR A/P	10 HR	HOURS				01-12 00-23	
03h	X	X	X	X	X	DAY			1-7 01-28/29	
04h	X	X	10 DATE			DATE			01-30 01-31	
05h	X	X	X	10 MONTH	MONTH				01-12	
06h	10 YEAR				YEAR					00-99
07h	OUT	X	X	SQWE	X	X	RS1	RS0		

Figura 24-5 Registros del DS1307

- El formato
- El bit 7 de
  - Si C
  - Si Cl
- El formato controlado
  - Si (B)
  - Si (B)
- En el modo determina
- En el modo hora.
- El contenido como se ex

## 24.4 REGIS

El registro gabinete de la onda bits es:

- Bits 0 y 1, cuando esta



Tabla 24-

- Bit 4, SQW
  - Si SQW
  - Si SQW
- Bit 7, OUT cuando esta
  - Si OUT
  - Si OUT

Así, por ejemplo, la palabra de control s

circuenten las líneas  
estos pulsadores.

RAM del DS1307.  
07h. Desde la 08h  
ser utilizadas para

a de los registros  
ario en detalle, a

00-59  
00-59  
01-12  
00-23  
1-7  
01-28/29  
01-30  
01-31  
01-12  
00-99

- El formato de todos los datos está en BCD.
- El bit 7 del registro 00h es el bit de puesta en marcha *Clock Halt* (CH):
  - Si CH=0, pone en marcha el reloj.
  - Si CH=1, impide el funcionamiento del reloj, el cual permanecerá parado.
- El formato de las horas puede seguir el modelo americano o el europeo controlado por el bit 6 del registro 02h (Bit 12/24).
  - Si (Bit 12/24) = 0, elige el modo europeo de 24 horas (0..23 h).
  - Si (Bit 12/24) = 1, elige el modo americano de 12 horas (1..12h).
- En el modo americano de 12 horas, el bit 5 (A/P) del registro 02h es el bit que determina si la hora es AM (bit A/P = 0) o PM (bit A/P = 1).
- En el modo europeo de 24 horas, el bit 5 es el de mayor peso de las decenas de hora.
- El contenido del registro 07h controla la señal cuadrada del pin SQW/OUT tal como se explica a continuación.

## 24.4 REGISTRO DE CONTROL

El registro de control del DS1307 está en la posición 07h y se utiliza para el gobierno de la onda cuadrada que se obtiene en el pin SQW/OUT. El significado de sus bits es:

- Bits 0 y 1, **RS** (*Rate Select*). Estos bits fijan la frecuencia de la onda cuadrada cuando está habilitada según la tabla 24-1.

RS1	RS0	FRECUENCIA
0	0	1 Hz
0	1	4.096 Hz
1	0	8.192 Hz
1	1	32.768 Hz

Tabla 24-1 Frecuencia de la onda cuadrada por la patilla SQW/OUT

- Bit 4, **SQWE** (*Square Wave Enable*). Habilita la onda cuadrada de salida:
  - Si SQWE = 0, en el pin SQW/OUT no hay una onda cuadrada.
  - Si SQWE = 1, en el pin SQW/OUT hay una onda cuadrada.
- Bit 7, **OUT** (*Output Control*). Indica el nivel lógico de la salida SQW/OUT cuando está deshabilitada la onda cuadrada (es decir si SQWE = 0):
  - Si OUT = 0, el pin SQW/OUT está a "0".
  - Si OUT = 1, el pin SQW/OUT está a "1".

Así, por ejemplo, para generar una onda cuadrada de 1 Hz por el pin SQWE, la palabra de control sería b'00010000' (SQWE = 1, RS1 = RS0 = 0).

## 24.5 ESCRITURA EN EL DS1307

La transferencia de datos desde el microcontrolador al DS1307 sigue el procedimiento de escritura del maestro al esclavo de un bus I<sub>2</sub>C ya conocido y esquematizado en la figura 24-6. La transferencia de datos se efectúa en el siguiente orden:

- Primero el microcontrolador maestro envía la condición de Start.
- Luego envía la dirección del DS1307 (*Slave Address*) en modo escritura que es la b'11010000' o D0h.
- A continuación el maestro envía un puntero con la primera dirección del registro a escribir (*Word Address*).
- Despues se transmiten los datos a escribir. La dirección del registro a escribir se incrementa automáticamente.
- Cuando termina de escribir el microcontrolador maestro envía la condición de Stop.

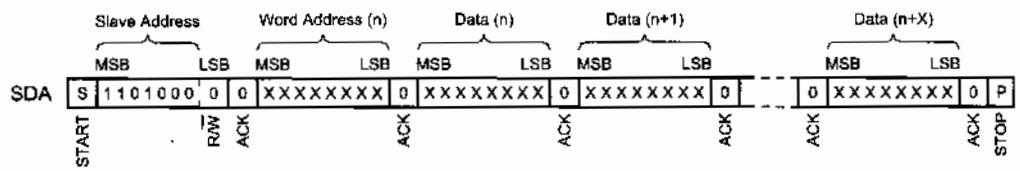


Figura 24-6 Protocolo de escritura del maestro sobre el DS1307

## 24.6 LECTURA DEL DS1307

La lectura de los datos del DS1307 por parte del microcontrolador sigue el procedimiento de lectura del esclavo por parte del maestro ya conocido y esquematizado en la figura 24-7. La transferencia de datos se efectúa en el siguiente orden:

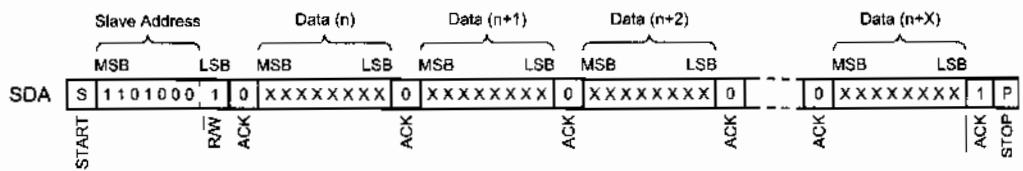


Figura 24-7 Protocolo de lectura del DS1307 por parte del maestro

- Primero el microcontrolador maestro envía la condición de Start.
- Luego envía la dirección del DS1307 (*Slave Address*) en modo lectura que es la b'11010001' o D1h.

- Después el se incrementa.
- Por último el

El primer registro que este dato se desconoce exactamente el valor es el DS1624.

## 24.7 LIBREAS

El protocolo de las subrutinas de control

- "DS1307\_Iniciar": Configura el pin SQW/O.
- "DS1307\_Lectura": Lee los registros de memoria.
- "DS1307\_Escritura": Escriben dentro del DS1307.
- "DS1307\_Config": Configura el reloj-calendario.

\*\*\*\*\*  
Estas subrutinas permiten leer y escribir el dispositivo transmisor y receptor.

ZONA DE DATOS \*\*\*

CBLOCK  
Anho  
Mes  
Dia  
DiaSemana  
Hora  
Minuto  
Segundo  
ENDC

DS1307\_DireccionEscritura  
DS1307\_DireccionLectura

; Subrutina "DS1307\_Iniciar"  
; Configura la señal cuadrangular de salida.

DS1307\_Inicializar

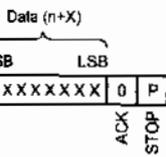
DS1307 sigue el  
ya conocido y  
en el siguiente

escritura que es la

acción del registro

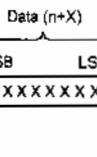
stro a escribir se

la condición de



307

olador sigue el  
esquematizado



estro

lectura que es la

- Después el maestro lee los datos de los registros. La dirección del registro a leer se incrementa automáticamente.
- Por último el microcontrolador maestro envía la condición de Stop.

El primer registro leído será el señalado en la última operación anterior realizada. Si este dato se desconoce, primero habrá que realizar una operación de escritura para conocer exactamente el valor de este puntero, de forma similar como se explicó para el termómetro DS1624.

## 24.7 LIBRERÍA DE SUBRUTINAS

El protocolo de escritura y lectura en el DS1307 se concreta en las siguientes subrutinas de control descritas en la librería DS1307.INC:

- "DS1307\_Inicializa". Configura la señal cuadrada que genera el DS1307 en su pin SQW/OUT a 1 Hz.
- "DS1307\_Lee". Lee las variables de tiempo del DS1307 y las guarda en los registros de tiempo correspondientes (Anho, Mes, Día, etc.).
- "DS1307\_Escribe". Carga los datos de los registros de tiempo (Anho, Mes, etc), dentro del DS1307.
- "DS1307\_CargaInicial". Realiza una carga inicial en los registros internos del reloj-calendario DS1307 a fecha: Lunes, 1 de Enero de 2004 a las 0:00:00.

```
***** Librería DS1307.INC *****
;
; Estas subrutinas permiten realizar las tareas de manejo del reloj-calendario DS1307.
; Este dispositivo transmite la información vía serie a través de un bus I2C.
;
; ZONA DE DATOS *****

CBLOCK
Anho ; Guarda el año.
Mes ; Guarda el mes.
Dia ; Guarda el día.
DiaSemana ; Guarda el día de la semana: lunes, etc.
Hora ; Guarda las horas.
Minuto ; Guarda los minutos.
Segundo ; Guarda los segundos.
ENDC

DS1307_DireccionEscritura EQU 0xD0 ; Dirección del DS1307.
DS1307_DireccionLectura EQU 0xD1

; Subrutina "DS1307_Inicializa"
;
; Configura la señal cuadrada que genera el DS1307 en su pin SQW/OUT a 1 Hz.

DS1307_Inicializa
```

```

call    I2C_EviaStart      ; Envía condición de Start.
movlw  DS1307_DireccionEscritura ; Indica al DS1307 que el byte a escribir,
call    I2C_EviaByte        ; está en la posición 07h, que corresponde
movlw  0x07                  ; al control de la señal cuadrada.
call    I2C_EviaByte        ;
movlw  b'00010000'          ; Escribe en el registro de control para
call    I2C_EviaByte        ; configurar onda cuadrada del DS1307 a 1 Hz.
call    I2C_EviaStop         ; Termina de enviar datos.
return

```

; Subrutina "DS1307\_Lee"

; Se leen las variables de tiempo del DS1307 y se guardan en los registros correspondientes.

DS1307\_Lee

```

bcf    I2C_UltimoByteLeer   ; Envía condición de Start.
call    I2C_EviaStart       ; Indica al DS1307 que el primer byte
movlw  DS1307_DireccionEscritura ; a leer está en la posición 00H, que corresponde
call    I2C_EviaByte        ; a los segundos.
movlw  0x00                  ; Envía condición de Stop.
call    I2C_EviaStop         ; Ahora va a leer el DS1307.
call    I2C_EviaStart       ; Envía condición de Start.
movlw  DS1307_DireccionLectura ; Lee los segundos.
call    I2C_EviaByte        ; Lo carga en el registro correspondiente.
call    I2C_LeeByte          ; Lee el resto de los registros utilizando
movwf  Segundo               ; el mismo procedimiento.
call    I2C_LeeByte          ;
movwf  Minuto                ;
call    I2C_LeeByte          ;
movwf  Hora                  ;
call    I2C_LeeByte          ;
movwf  DiaSemana             ;
call    I2C_LeeByte          ;
movwf  Dia                   ;
call    I2C_LeeByte          ;
movwf  Mes                   ;
bsf    I2C_UltimoByteLeer   ; Para terminar.
call    I2C_LeeByte          ;
movwf  Anho                  ;
call    I2C_EviaStop          ; Acaba de leer.
return

```

; Subrutina "DS1307\_CargaInicial"

; Realiza una carga inicial en los registros internos del reloj-calendario DS1307 a fecha  
; Lunes, 1 de Enero de 2004 a las 0:00:00.

DS1307\_CargaInicial

```

movlw  .1                  ; Inicializa todos los datos del reloj: año, mes,
movwf  Dia                 ; día, día de la semana, hora, minuto y segundo.
movwf  Mes

```

```

movwf  Di
movlw  .4
movwf  An
clrf   Ho
clrf   Mi
clrf   Seg
call   DS
;
;
; Subrutina "DS1307_E
;
; Carga los datos de los
DS1307_Escrive
call   I2C
movlw DS
call   I2C
movlw 0x0
call   I2C
movf  Seg
call   I2C
movf  Min
call   I2C
movf  Hor
call   I2C
movf  Dia
call   I2C
movf  Dia
call   I2C
movf  Dia
call   I2C
movf  Mes
call   I2C
movf  Anh
call   I2C
call   I2C
return

```

## 24.8 PROGRAMACIÓN

La figura 24-24 muestra la programación del DS1307 y controlador PIC16F84. Los comentarios de los programas están en los comentarios de los diagramas de los bloques.

A partir de este punto, es posible programar el DS1307 y controlador PIC16F84 en reloj-calendario con un programador de memoria programable fácilmente.

```

        movwf  DiaSemana
        movlw  4
        movwf  Anho          ; Inicializa en el año 2004.
        clrf   Hora
        clrf   Minuto
        clrf   Segundo
        call   DS1307_Escribe ; Despues lo graba en el DS1307 para
        return                         ; ponerlo en marcha.

; Subrutina "DS1307_Escribe" -----
;
```

Carga los datos de los registros Anho, Mes, etc., dentro del DS1307.

```

DS1307_Escribe
    call   I2C_EnviaStart      ; Envía condición de Start.
    movlw  DS1307_DireccionEscritura ; Indica al DS1307 que el primer byte a escribir
    call   I2C_EnviaByte        ; está en la posición 00h que corresponde
    movlw  0x00                 ; a los segundos.
    call   I2C_EnviaByte
    movf   Segundo,W           ; Pasa los segundos de la memoria del PIC16F84A al
    call   I2C_EnviaByte        ; DS1307.
    movf   Minuto,W            ; Y se repite el proceso para el resto.
    call   I2C_EnviaByte
    movf   Hora,W
    call   I2C_EnviaByte
    movf   DiaSemana,W
    call   I2C_EnviaByte
    movf   Dia,W
    call   I2C_EnviaByte
    movf   Mes,W
    call   I2C_EnviaByte
    movf   Anho,W
    call   I2C_EnviaByte
    call   I2C_EnviaStop        ; Termina de enviar datos.
    return

```

## 24.8 PROGRAMA DEL RELOJ CALENDARIO DIGITAL

La figura 24-3 muestra el esquema de un reloj digital y calendario basado en el DS1307 y controlado por un microcontrolador PIC16F84A. Su funcionamiento se detalla en los comentarios del documentado programa de control que se expone a continuación y en los diagramas descriptivos que se intercalan.

A partir de este esquema y este programa es fácil implementar proyectos basados en reloj-calendario de tiempo real tales como temporizadores programables, cronómetros, programadores de tiempo, etc. Por ejemplo en el capítulo 18 se explica un temporizador programable fácilmente adaptable a este dispositivo.

\*\*\*\*\* I2C\_Relej\_01.asm \*\*\*\*\*

; Programa para un reloj digital en tiempo real con puesta en hora. Visualiza los datos del reloj en formato: "Día Mes Año" (Primera linea)  
; "Día\_de\_la\_Semana Horas:Minutos:Segundos", (Segunda linea)  
; (por ejemplo " 7 Diciemb. 2004" (Primera linea).  
; "Martes 8:47:39" (Segunda linea).

; La actualización del reloj se consigue leyendo el chip DS1307, que es un reloj y calendario en tiempo real compatible con bus I2C.

; El DS1307 se configura para que genere una señal cuadrada de un 1 Hz por su pin SQW/OUT.  
; Esta señal se aplica al pin de interrupción INT del PIC16F84A de manera que genera una interrupción cada segundo, que es cuando realiza la lectura del DS1307 y la visualiza.

; La "PuestaEnHora" se logra mediante dos pulsadores: "MODO" e "INCREMENTAR".  
; Su modo de operación es:  
; 1º Pulsa MODO, los "Años" se ponen intermitente y se ajustan mediante el pulsador INCREMENTAR.  
; 2º Pulsa MODO y pasa a ajustar los "Meses" de forma similar.  
; 3º Se va pulsando MODO secuencialmente para ajustar del modo anteriormente explicado los días del mes, los días de la semana, las horas y los minutos.  
; 4º Pulsa MODO y se acabó la "PuestaEnHora", pasando a visualización normal.

; Cuando se ajusta una variable de tiempo, ésta debe aparecer en intermitencia.

; Los pulsadores MODO e INCREMENTAR se conectan a líneas del Puerto B y su funcionamiento se basa en interrupción por cambio en la línea del Puerto B:

\*\*\*\*\* ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
Auxiliar          ; Registro auxiliar.
Intermitencia    ; Para lograr la intermitencia.
                  ; Si es 0 apaga en intermitencia.
FlagsAjuste       ; Guarda los flags para establecer los
ENDC              ; ajustes de día, mes, año, hora, etc.

#define ModoPulsador PORTB,7      ; Los pulsadores se conectan a estos
#define IncrementarPulsador PORTB,6 ; pinos del Puerto B.
#define OndaCuadrada_DS1307 PORTB,0 ; La onda cuadrada al pin RBO/INT.
#define F_AjusteAnho   FlagsAjuste,5 ; Flags para los diferentes modos de
#define F_AjusteMes    FlagsAjuste,4 ; ajustes.
#define F_AjusteDia    FlagsAjuste,3
#define F_AjusteDiaSemana FlagsAjuste,2
#define F_AjusteHora   FlagsAjuste,1
#define F_AjusteMinuto FlagsAjuste,0
```

Habilita las

Mensajes	addwf	PCI
MensajeLunes	DT "Lunes"	
MensajeMartes	DT "Martes"	
MensajeMiércoles	DT "Mierc."	

Fig

## ZONA DE CÓDIGOS \*\*\*\*

```

ORG    0
goto   Inicio
ORG    4
goto   ServicioInterrupcion

```

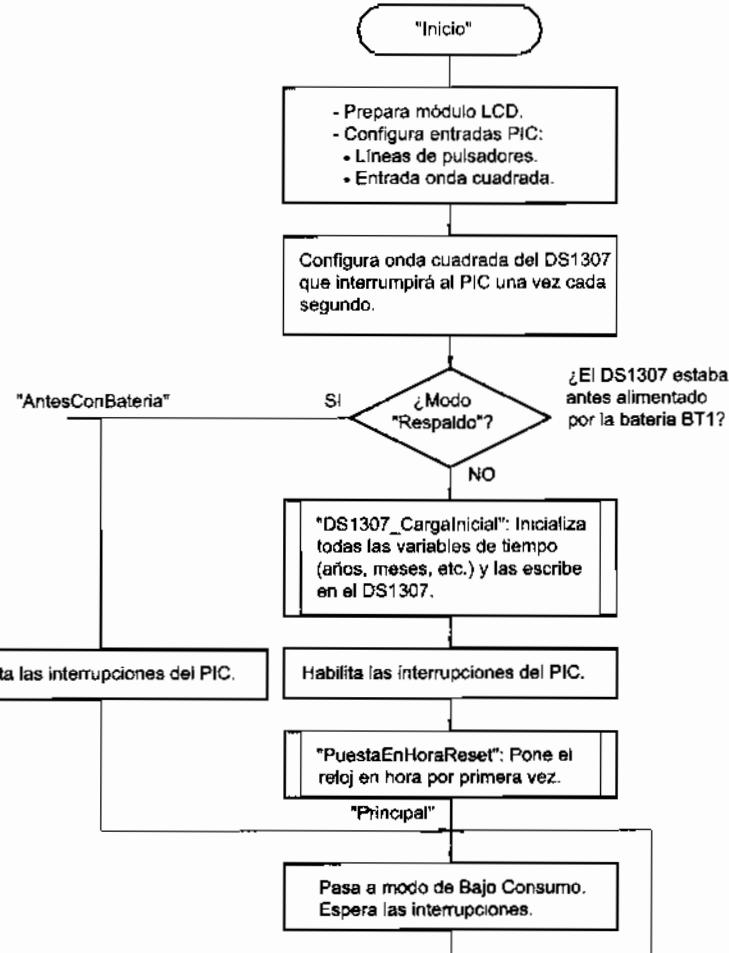


Figura 24-8 Diagrama de flujo principal del reloj digital

```
Mensajes      addwf    PCL,F  
MensajeLunes   DT "Lunes ",0x00  
MensajeMartes  DT "Martes ",0x00  
MensajeMiércoles DT "Miérco.",0x00
```

```

MensajeJueves           retlw
                        DT "Jueves ", 0x00
MensajeViernes          retlw
                        DT "Viernes ", 0x00
MensajeSabado            retlw
                        DT "Sabado ", 0x00
MensajeDomingo          retlw
                        DT "Domingo ", 0x00
;
MensajeEnero             retlw
                        DT " Enero ", 0x00
MensajeFebrero           retlw
                        DT " Febrero", 0x00
MensajeMarzo              retlw
                        DT " Marzo ", 0x00
MensajeAbril              retlw
                        DT " Abril ", 0x00
MensajeMayo               retlw
                        DT " Mayo ", 0x00
MensajeJunio              retlw
                        DT " Junio ", 0x00
MensajeJulio              retlw
                        DT " Julio ", 0x00
MensajeAgosto             retlw
                        DT " Agosto ", 0x00
MensajeSeptiembre         retlw
                        DT "Septiem.", 0x00
MensajeOctubre            retlw
                        DT "Octubre ", 0x00
MensajeNoviembre          retlw
                        DT "Noviemb.", 0x00
MensajeDiciembre          retlw
                        DT "Diciemb.", 0x00
MensajeBlanco             retlw
                        DT "      ", 0x00 ; Ocho espacios en blanco.

;
DiasSemana
    addwf    PCL,F
    nop
    retlw    MensajeLunes
    retlw    MensajeMartes
    retlw    MensajeMiercoles
    retlw    MensajeJueves
    retlw    MensajeViernes
    retlw    MensajeSabado
    retlw    MensajeDomingo
; No hay dia 0 de la semana. Empiezan en
; el día 1 de la semana (Lunes).
; Día 2 de la semana.
; Día 3 dc la semana.
; Día 4 de la semana.
; Día 5 de la semana.
; Día 6 de la semana.
; Día 7 de la semana.

;
Meses
    addwf    PCL,F
    nop
    retlw    MensajeEnero
    retlw    MensajeFebrero
    retlw    MensajeMarzo
    retlw    MensajeAbril
    retlw    MensajeMayo
; No hay mes 0x00.
; Empiezan en el mes 1 (Enero).
; Mes 0x02
; Mes 0x03
; Mes 0x04
; Mes 0x05

;
FinTablas
; Estas tablas y mens
; supere la posición 0
; así fuera, se visualiz
;
IF (FinTablas)
    ENDIF
; Instrucciones de inici
;
Inicio   call
        bcf
        bcf
        bcf
        bcf
        bcf
        bcf
        bcf
        bcf
        clrf
        call

;
; Ahora lee el bit 7 de
; reset. Este bit es el C
; reloj (CH=1). Tras la
; - Si CH está
; vez y está e
; alimentand
; la pantalla.
; DS1307 qu
;
; - Si CH está
;
call
        btfss
        goto
        call

```

retlw	MensajeJunio	; Mes 0x06	
retlw	MensajeJulio	; Mes 0x07	
retlw	MensajeAgosto	; Mes 0x08	
retlw	MensajeSeptiembre	; Mes 0x09	
nop		; Mes 0xA	(Como el DS1307 trabaja en BCD,
nop		; Mes 0xB	estos meses se llenan con
nop		; Mes 0xC	"nop").
nop		; Mes 0xD	
nop		; Mes 0xE	
nop		; Mes 0xF	
retlw	MensajeOctubre	; Mes 0x10	
retlw	MensajeNoviembre	; Mes 0x11	
retlw	MensajeDiciembre	; Mes 0x12	

FinTablas

; Estas tablas y mensajes se sitúan al principio del programa con el propósito que no  
; supere la posición OFFh de memoria ROM de programa. De todas formas, en caso de que  
; así fuera, se visualizaría el siguiente mensaje de error en el proceso de ensamblado:

;  
; IF (FinTablas > 0xFF)  
;     ERROR "Atención: La tabla ha superado el tamaño de la página de los"  
;     MESSG "primeros 256 bytes de memoria ROM. NO funcionará correctamente."  
ENDIF

; Instrucciones de inicialización.

;  
; Inicio call LCD\_Inicializa  
bsf STATUS,RP0 ; Acceso banco 1.  
bcf OPTION\_REG,NOT\_RBPU ; Se activan las resistencias de Pull-Up del Puerto B.  
bsf ModoPulsador ; Los pulsadores se configuran como entrada.  
bsf IncrementarPulsador  
bsf OndaCuadrada\_DS1307  
bcf OPTION\_REG,JNTEDG ; Interrupción INT activa por flanco de bajada.  
bcf STATUS,RP0 ; Acceso banco 0.  
clrf FlagsAjuste ; Inicializa todos los flags de la puesta en hora.  
call DS1307\_Inicializa ; Configura la señal cuadrada que genera el  
; DS1307 en su pin SQW/OUT a 1 Hz.

;  
; Ahora lee el bit 7 de los segundos para saber el estado anterior del reloj antes del  
; reset. Este bit es el CH (Control Halt) y permite poner en marcha (CH=0) o parar el  
; reloj (CH=1). Tras la lectura pueden ocurrir dos casos:

;- Si CH está a "0" quiere decir que anteriormente el reloj ya se ha puesto en hora alguna  
vez y está en modo "respaldo". Es decir, ha fallado la alimentación y el DS1307 se está  
alimentando por la batería conectada al pin "VBAT", aunque el reloj no se visualice en  
la pantalla. Por tanto, no debe inicializarse porque se corrompería el contenido del  
DS1307 que ya está en marcha.  
;- Si CH está a "1" quiere decir que estaba parado y es la primera vez que se pone en hora.

call DS1307\_Lee ; Lee los segundos en el DS1307 y comprueba el  
btfs Segundo,7 ; valor del bit CH. Si es "1" salta toda la  
goto AntesConBateria ; inicialización de los registros y la puesta  
; en hora inicial.  
call DS1307\_CargaInicial ; Realiza la carga inicial a 1 Enero de 2004.

```

movlw b'100111000' ; Las interrupciones se deben habilitar después
movwf INTCON        ; de la carga inicial del DS1307.
call PuestaEnHoraReset ; Puesta en hora por primera vez.
goto Principal

```

AntesConBateria

**movlw b'10011100'** ; Habilita las interrupciones INT, RBI y la  
**movwf INTCON** ; general GIE.

; La sección "Principal" es de mantenimiento. Pasa al modo reposo y sólo espera las interrupciones procedentes de la línea "SQW/OUT" del DS1307 que se producen cada segundo en funcionamiento normal y cada 500 ms cuando esté en la "PuestaEnHora".

## Principal

sleep  
goto Principal

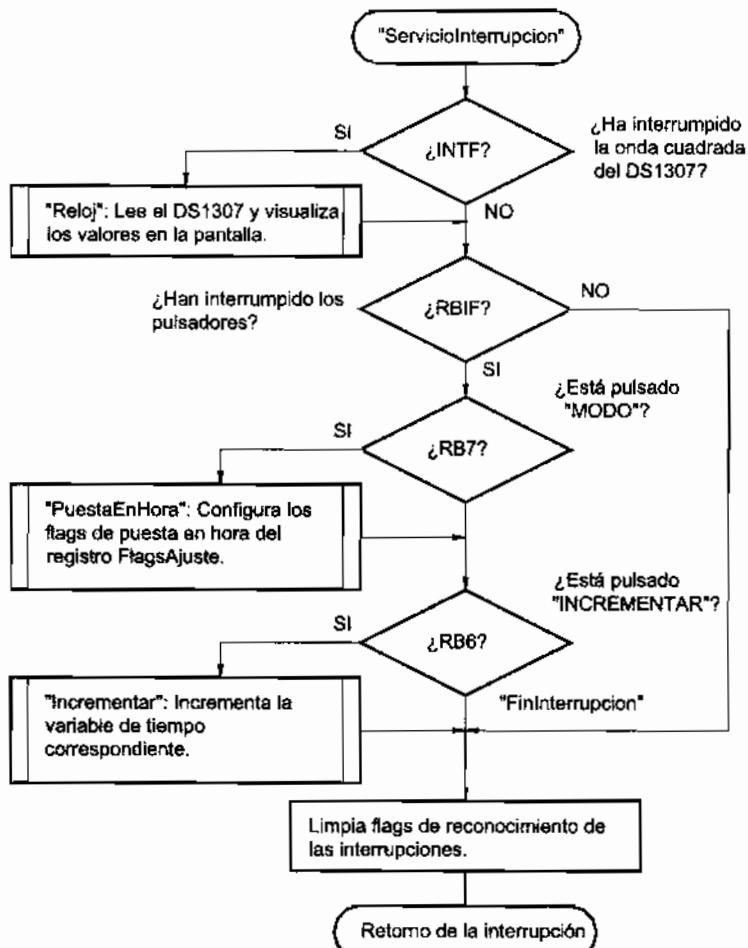


Figura 24-9 Diagrama de flujo de la subrutina de servicio a las interrupciones

; Subrutina "ServicioInterrupcion"

; Detecta qué ha producido la interrupción y ejecuta la subrutina de atención correspondiente.

#### ServicioInterrupcion

btfsc	INTCON,INTF	; Si es una interrupción procedente de la onda
call	Reloj	; cuadrada del DS1307, actualiza la visualización.
btfss	INTCON,RBIF	; Si es una interrupción RBI, lee los pulsadores.
goto	FinInterrupcion	
btfss	ModoPulsador	; ¿Está presionado el pulsador de "MODO"?
call	PuestaEnHora	; Sí, pasa a poner en hora.
btfss	IncrementarPulsador; Pulsado "INCREMENTAR"?	
call	Incrementar	; Sí, pasa a incrementar el registro de tiempo
		; correspondiente.
		; Limpia los flags de reconocimiento de la
FinInterrupcion		; interrupción.
bcf	INTCON,RBIF	
bcf	INTCON,INTF	
retfie		

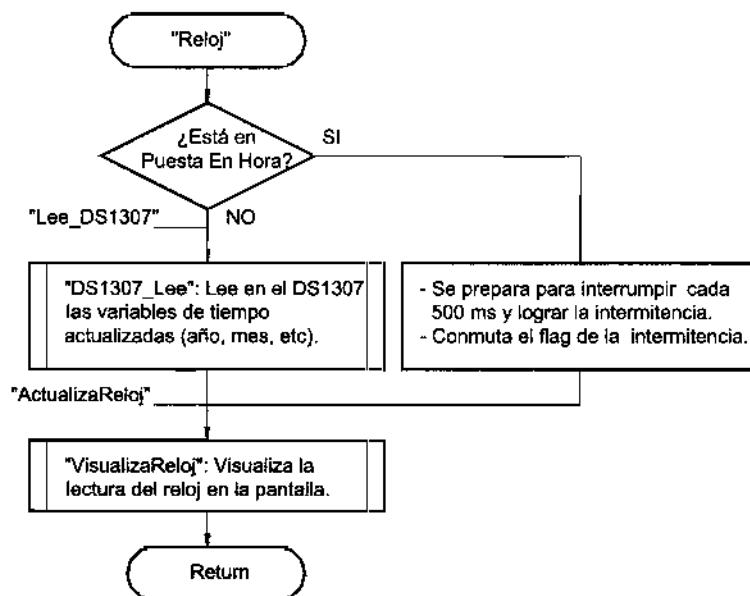


Figura 24-10 Diagrama de flujo de la subrutina Reloj

; Subrutina "Reloj"

; Esta subrutina actualiza los registros Anho, Mes, Dia, DiaSemana, Hora, Minuto y Segundo leyendo el DS1307 a través del bus I2C. Se ejecuta debido a la petición de interrupción de la señal cuadrada de 1 Hz, que procede del pin "SQW/OUT" del DS1307 y se ha conectado al pin RB0/INT del PIC16F84A.

; Esta interrupción ocurre cada segundo si está en visualización normal y cada 500 ms si está en puesta en hora con el objetivo de conseguir la intermitencia.

call  
goto  
ApagaDia  
call

; Si está en algún modo de ajuste debe interrumpir cada 500 ms para lograr la intermitencia ; alternando la interrupción por flanco de subida o de bajada. Esto lo consigue mediante la ; operación XOR del flag INTEDG con un "1", lo cual invierte este bit cada vez que ejecuta la ; instrucción y por tanto commuta entre flanco ascendente y descendente.

```

bsf    STATUS,RP0          ; Acceso al Banco 1.
movlw  b'01000000'          ; El bit INTEDG está en el lugar 6 del registro.
xorwf  OPTION_REG,F
bcf    STATUS,RP0          ; Acceso al Banco 0.

```

; Además complementa el registro Intermitencia para que se produzca la intermitencia cada 500 milisegundos cuando está en la puesta en hora.

**comf**    Intermitecia,F  
**goto**    ActualizaReloj ; Visualiza el reloj y sale.

Leer DS1307

; No está en ningún modo de ajuste, sino en funcionamiento normal de reloj y pasa a leer los  
; registros del reloj-calendario DS1307 a través del bus I2C. Los registros del DS1307 ya  
; trabajan en formato BCD, por tanto, no hay que hacer ningún tipo de conversión al leerlos.

```
call DS1307_Lee ; Estas dos instrucciones se pueden
ActualizaReloj ; obviar, ya que a continuación efectivamente
; ; ejecuta la subrutina VisualizaReloj y
; ; retorna.
```

; Subrutina "VisualizaReloj"

- Visualiza el reloj en formato "Día Mes Año" (Primera Línea)  
"Día\_de\_la\_Semana Horas:Minutos:Segundos", (Segunda Línea).
- Por ejemplo  
" 7 Diciemb. 2004" (Primera Línea).  
"Martes 8:47:39" (Segunda Línea).

; La variable ajustada debe aparecer en intermitencia. Esto se logra con ayuda de cualquier bit del registro Intermitencia que commuta cada 500 ms en la subrutina Reloj.

Visu

<b>VisualizaReloj</b>		
call	LCD_Lineal	; Se sitúa en la primera línea.
btfs	F_AjusteDia	; ¿Está en la puesta en hora?
goto	EnciendeDia	; No. Pasa a visualización normal.
btfs	Intermitencia,0	; Sí. Pasa a intermitencia si procede.
goto	ApagaDia	; Apaga en la intermitencia.
<b>EnciendeDia</b>		
movf	Dia,W	; Lo visualiza. El DS1307 ya lo da en BCD.

```

VisualizaMes
->    call
        btfs
        goto
        btfs
        goto
EnciendeMes
        movf
        call
        goto
ApagaMes
        movlw
        call

```

```

call LCD_Byt      ; Visualiza rechazando cero de las decenas.
goto VisualizaMes
ApagaDia
call LCD_DosEspaciosBlancos ; Visualiza dos espacios en blanco.

```

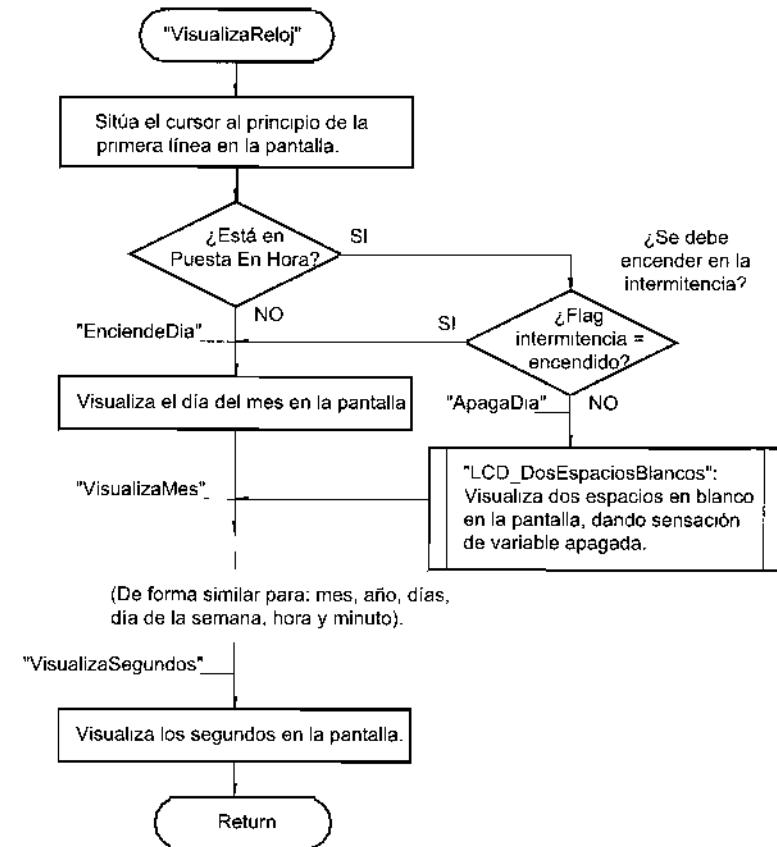


Figura 24-11 Diagrama de flujo de la subrutina VisualizaReloj

VisualizaMes		
call LCD_UnEspacioBlanco	;	Visualiza un espacio en blanco.
btfs F_AjusteMes	;	¿Está en la puesta en hora?
goto EnciendeMes	;	No. Visualización normal.
btfs Intermitencia,0	;	Sí. Intermitencia si procede.
goto ApagaMes	;	Apaga en la intermitencia.
EnciendeMes		
movf Mes,W	;	Lo visualiza.
call EscribeMes		
goto VisualizaAnho		
ApagaMes		
movlw MensajeBlanco		
call LCD_Mensaje	;	Visualiza varios espacios en blanco.
	;	

VisualizaAnho	btfs	F_AjusteAnho	; ¿Está en la puesta en hora?
	goto	EnciendeAnho	; No. Visualización normal.
	btfs	Intermitencia,0	; Sí. Intermitencia si procede.
	goto	ApagaAnho	; Apaga en la intermitencia.
EnciendeAnho	call	LCD_UnEspacioBlanco	; Visualiza un espacio en blanco.
	movlw	0x20	; Visualiza el "20xx", del año "dos mil ..."
	call	LCD_Byte	
	movf	Anho,W	
	call	LCD_BytCompleto	
	goto	VisualizaDiaSemana	
ApagaAnho	movlw	MensajeBlanco	
	call	LCD_Mensaje	; Visualiza varios espacios en blanco.
;			
VisualizaDiaSemana	call	LCD_Linea2	
	btfs	F_AjusteDiaSemana	; Se sitúa en la segunda línea.
	goto	EnciendeDiaSemana	; ¿Está en la puesta en hora?
	btfs	Intermitencia,0	; No. Visualización normal.
	goto	ApagaDiaSemana	; Sí. Intermitencia si procede.
EnciendeDiaSemana	movf	DiaSemana,W	; Apaga en la intermitencia.
	call	EscribeDiaSemana	
	goto	VisualizaHoras	
ApagaDiaSemana	movlw	MensajeBlanco	
	call	LCD_Mensaje	
;			
VisualizaHoras	btfs	F_AjusteHora	
	goto	EnciendeHoras	; ¿Está en la puesta en hora?
	btfs	Intermitencia,0	; No. Visualización normal.
	goto	ApagaHoras	; Sí. Intermitencia si procede.
EnciendeHoras	movf	Hora,W	; Apaga las horas en la intermitencia.
	call	LCD_Byt	
	goto	VisualizaMinutos	
ApagaHoras	call	LCD_DosEspacioBlancos	
;			
VisualizaMinutos	movfw	'.'	; Visualiza dos espacios en blanco.
	call	LCD_Caracter	
	btfs	F_AjusteMinuto	
	goto	EnciendeMinutos	; Envía ":" para separar datos.
	btfs	Intermitencia,0	
	goto	ApagaMinutos	; ¿Está en la puesta en hora?
EnciendeMinutos	movf	Minuto,W	
	call	LCD_BytCompleto	; Visualiza minutos.
	goto	VisualizaSegundos	

```

ApagaMinutos          call    LCD
;
VisualizaSegundos     movlw   '1'
                      call    LCD
                      movf    Segu
                      call    LCD
                      return
;
; Subrutina "EscribeDiaSemana"
;
; Escribe el día de la semana
; "DiaSemana". Supone que
; En el registro W se le indica
; dia de la semana en letra
;
; Primero comprueba que no estén
; erráticos con la llamada
; dato por parte del DS1307
;
EscribeDiaSemana      sublw   0x07
                      btfsc   STA
                      goto   Llamar
                      movlw   0x01
                      movwf   DiaSemana
Llamar_a_DiasSemana   movf    DiaSemana
                      call    DiasSemana
                      call    LCD
                      return

; Subrutina "EscribeMes"
;
; Escribe el mes del año en letra
;
; Primero comprueba que no estén
; erráticos con la llamada
; dato por parte del DS1307.
;
; El DS1307 trabaja en BCD
; solución con una corrección
;
EscribeMes              sublw   0x12
                          btfsc  STA
                          goto   Llamar
                          movlw   0x01
                          movwf   Mes
Llamar_a_Meses          movf    Mes
                          call    Mes

```

```

ApagaMinutos
    call    LCD_DosEspaciosBlancos ; Visualiza dos espacios en blanco.

;
VisualizaSegundos
    movlw   ":"                     ; Envía ":" para separar datos.
    call    LCD_Caracter
    movf    Segundo,W               ; Visualiza segundos.
    call    LCD_ByteCompleto
    return

;
; Subrutina "EscribeDiaSemana"
;
; Escribe el dia de la semana en la posición actual de la pantalla, utilizando la tabla
; "DiaSemana". Supone que el Lunes es el día 1 y el Domingo el 7.
; En el registro W se le introduce el día de la semana numérico y en la pantalla aparece el
; día de la semana en letras. Así por ejemplo si (W)=0x02 en la pantalla aparecerá "Martes".
;
; Primero comprueba que no ha superado el valor máximo para evitar problemas de saltos
; erráticos con la llamada "call DiasSemana", en caso de una lectura defectuosa de este
; dato por parte del DS1307.
;
EscribeDiaSemana
    sublw  0x07                   ; ¿Ha superado su valor máximo?
    btfsr STATUS,C
    goto   Llamada_a_DiasSemana
    movlw  0x01                   ; Lo inicializa si ha superado su valor máximo.
    movwf  DiaSemana
Llamada_a_DiasSemana
    movf   DiaSemana,W
    call   DiasSemana
    call   LCD_Mensaje
    return

;
; Subrutina "EscribeMes"
;
; Escribe el mes del año en la posición actual de la pantalla, utilizando la tabla "Meses".
;
; Primero comprueba que no ha superado el valor máximo para evitar problemas de saltos
; erráticos con la llamada "call Meses", en caso de una lectura defectuosa de este dato
; por parte del DS1307.
;
; El DS1307 trabaja en BCD y la tabla en binario natural. Esto es un problema que se
; soluciona con una correcta distribución de la tabla "Meses".
;
EscribeMes
    sublw  0x12                   ; ¿Ha superado su valor máximo? (Observad que
    btfsr STATUS,C               ; trabaja en BCD)
    goto   Llamada_a_Meses
    movlw  0x01                   ; Lo inicializa si ha superado su valor máximo.
    movwf  Mes
Llamada_a_Meses
    movf   Mes,W                 ; Retorna el resultado en el registro W.
    call   Meses

```

```
call      LCD_Mensaje
return
```

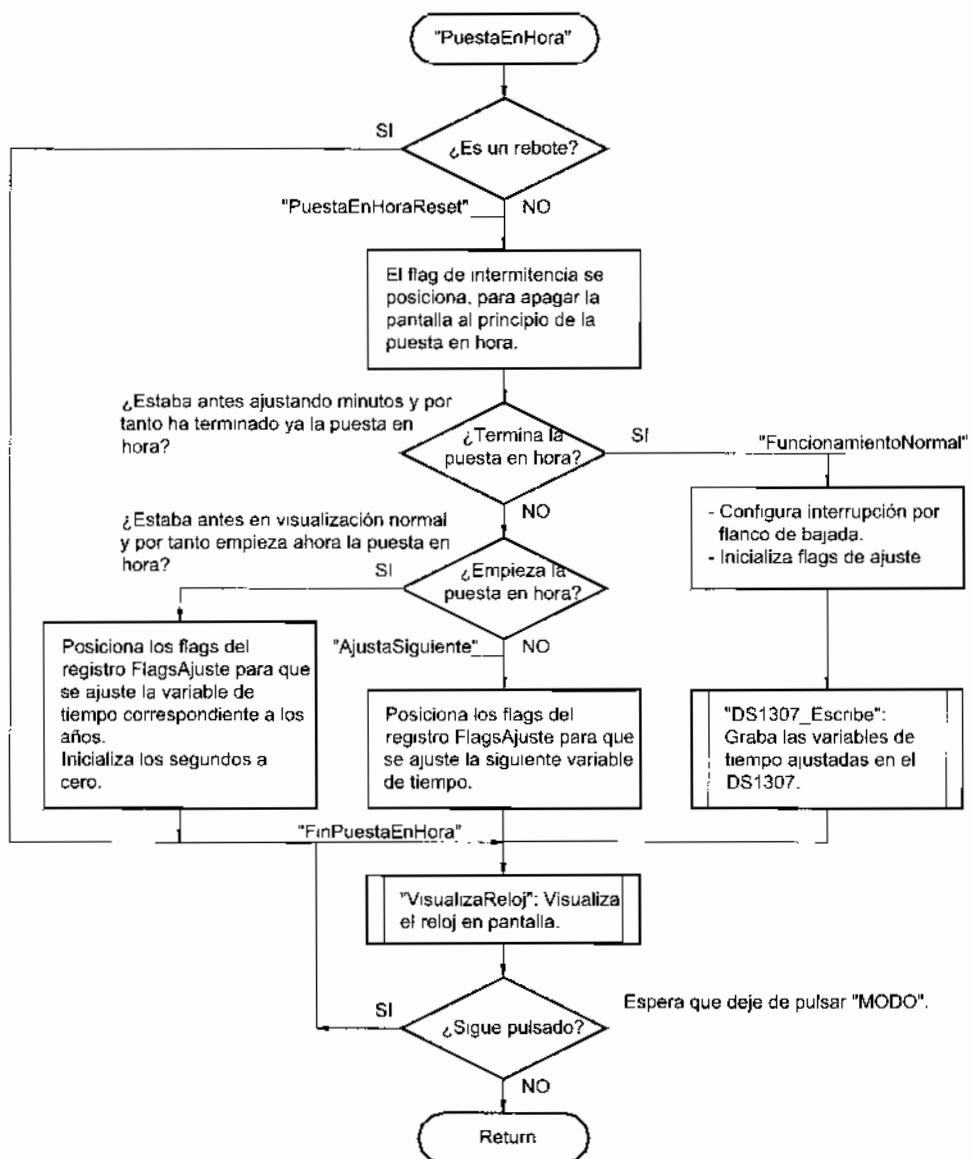


Figura 24-12 Diagrama de flujo de la subrutina PuestaEnHora

```

; Subrutina "PuestaEnHora"
;
; Subrutina de atención a la interrupción producida por el pulsador MODO que pone en hora el
; reloj. Cada vez que pulsa se desplaza el "1" a través del registro FlagsAjuste, pasando a
; ajustar secuencialmente: años, meses, días, días de la semana, horas y minutos.

```

```

;
; Para comprender el funcio
; contiene 5 flags que pem
; - "F_AjusteAnho":
; - "F_AjusteMes":
; - "F_AjusteDia":
; - "F_AjusteDiaSemana":
; - "F_AjusteHora":
; - "F_AjusteMinuto":
;
; Así pues el contenido del
; - (FlagsAjuste)=b'0001000
; - (FlagsAjuste)=b'000100
; - (FlagsAjuste)=b'0000010
; - (FlagsAjuste)=b'0000001
; - (FlagsAjuste)=b'0000000
; - (FlagsAjuste)=b'0000000
; - (FlagsAjuste)=b'0000000
;
; Pueden ocurrir tres casos:
; - Que pulse "MOD"
;   (FlagsAjuste)=b'0
;   carga (FlagsAjuste)
;   FlagsAjuste.
; - Que pulse "MOD" y
;   ajuste del siguiente
;   a derechas. Así p
;   (FlagsAjuste)=b'1
;   como ajuste de lo
; - Que pulse "MOD" y
;   (FlagsAjuste)=b'0
;   puesta en hora ha
;
```

#### PuestaEnHora

```

    call      Retardo
    btfsc   ModoP
    goto   FinPue

```

#### PuestaEnHoraReset

```

    clrf   Intermit
    btfsc   F_Ajus
    goto   Funcio
    movf   FlagsA
    btfss   STATU
    goto   Ajusta
    bsf   F_Ajus
    clrf   Segund
    goto   FinPue

```

#### AjustaSiguiente

```

    bcf   STATU
    rrf   FlagsA
    goto   FinPue

```

```

;
; Lo siguiente se ejecuta si y
;
```

```

;
; Para comprender el funcionamiento de esta subrutina hay que saber que el registro FlagsModos
; contiene 5 flags que permite diferenciar cada uno de los ajustes de registros de tiempo:
; - "F_AjusteAnho": bit 5 de FlagsAjuste, para ajustar los años.
; - "F_AjusteMes": bit 4 de FlagsAjuste, para ajustar los meses.
; - "F_AjusteDia": bit 3 de FlagsAjuste, para ajustar los días del mes.
; - "F_AjusteDiaSemana": bit 2 de FlagsAjuste, para ajustar los días de la semana.
; - "F_AjusteHora": bit 1 de FlagsAjuste, para ajustar las horas.
; - "F_AjusteMinuto": bit 0 de FlagsAjuste, para ajustar los minutos.
;

; Así pues el contenido del registro FlagAjuste identifica los siguientes ajustes:
; - (FlagsAjuste)=b'00100000'. Está ajustando el registro Anho (Años).
; - (FlagsAjuste)=b'00010000'. Está ajustando el registro Mes.
; - (FlagsAjuste)=b'00001000'. Está ajustando el registro Dia.
; - (FlagsAjuste)=b'00000100'. Está ajustando el registro DiaSemana (Lunes, Martes, etc).
; - (FlagsAjuste)=b'00000010'. Está ajustando el registro Hora.
; - (FlagsAjuste)=b'00000001'. Está ajustando el registro Minuto.
; - (FlagsAjuste)=b'00000000'. Está en visualización normal del reloj en tiempo real.
;
```

; Pueden ocurrir tres casos:

- Que pulse "MODO" estando en modo de visualización normal identificado porque (FlagsAjuste)=b'00000000'. En este caso debe activar el flag F\_AjusteAnho, es decir, carga (FlagsAjuste)=b'00100000', ya que el flag F\_AjusteAnho es el bit 5 del registro FlagsAjuste.
- Que pulse "MODO" estando ya en la puesta en hora, en cuyo caso debe pasar al ajuste del siguiente registro de tiempo. Ésto lo hace mediante un desplazamiento a derechas. Así por ejemplo, si antes estaba ajustando los meses, es decir: (FlagsAjuste)=b'00010000', pasará a (FlagsAjuste)=b'00001000' que se identifica como ajuste de los días del mes.
- Que pulse "MODO" estando en el último ajuste correspondiente a los minutos, (FlagsAjuste)=b'00000001', pasará a (FlagsAjuste)=b'00000000', indicando que la puesta en hora ha terminado y pasa a visualización normal del reloj en tiempo real.

#### PuestaEnHora

call	Retardo_20ms	; Espera a que se establece el nivel de tensión.
btfsc	ModoPulsador	; Si es un rebote sale fuera.
goto	FinPuestaEnHora	

#### PuestaEnHoraReset

clrf	Intermitencia	; Al pulsar "MODO" se apaga la variable de
btfsc	F_AjusteMinuto	; tiempo que se va a ajustar.
goto	FuncionamientoNormal	; Si antes estaba en ajuste de minutos es que
movf	FlagsAjuste,F	; ha terminado. Graba datos en el DS1307 y sale.
btfss	STATUS,Z	; Si antes estaba en funcionamiento normal ahora
goto	AjustaSiguiente	; pasa a ajustar el año.
bsf	F_AjusteAnho	; Sino pasa a ajustar la variable de tiempo siguiente.
clrf	Segundo	; Pasa a ajustar el año.
goto	FinPuestaEnHora	; Inicializa contador de segundos.

#### AjustaSiguiente

bcf	STATUS,C	; Desplaza un uno a la derecha del registro
rff	FlagsAjuste,F	; FlagsAjuste para ajustar secuencialmente cada
goto	FinPuestaEnHora	; uno de los registros de tiempo: año, mes, día,
		; día de la semana, hora y minuto.

; Lo siguiente se ejecuta si ya ha acabado el ajuste de la hora, es decir, pasa a

; funcionamiento normal. En este caso hay que realizar tres operaciones:

- ; - Fijar la interrupción INT sólo por flanco de bajada.
- ; - Inicializar a cero todos los flags de ajuste contenidos en (FlagsAjuste).
- ; - Escribir el DS1307 con los datos de las variables de tiempo contenidas en la memoria RAM del microcontrolador.

#### FuncionamientoNormal

```

bsf    STATUS,RP0          ; Acceso al Banco 1.
bcf    OPTION_REG,INTEDG   ; Interrupción INT activa por flanco de bajada.
bcf    STATUS,RP0          ; Acceso al Banco 0.
clrf   FlagsAjuste        ; Inicializa los flags de ajuste.
call   DS1307_Escribe     ; Graba los datos en el DS1307.

FinPuestaEnHora
call   VisualizaReloj      ; Visualiza los datos del reloj digital.
btfsf  ModoPulsador       ; Ahora espera deje de pulsar.
goto   FinPuestaEnHora
return

```

#### ; Subrutina "Incrementar"

- ; Subrutina de atención a la interrupción por cambio de la línea RB6 al cual se ha conectado el pulsador "INCREMENTAR".
- ; Incrementa según corresponda una sola de las siguientes variables: (Anho), (Mes), (Dia), (DiaSemana), (Hora) o (Minuto).

#### Incrementar

```

call   Retardo_20ms         ; Espera a que se establezca el nivel de tensión.
btfsf  IncrementarPulsador; Si es un rebote sale fuera.
goto   FinIncrementar
btfsf  F_AjusteAnho
call   IncrementaAnhos
btfsf  F_AjusteMes
call   IncrementaMeses
btfsf  F_AjusteDia
call   IncrementaDias
btfsf  F_AjusteDiaSemana
call   IncrementaDiasSemana
btfsf  F_AjusteHora
call   IncrementaHoras
btfsf  F_AjusteMinuto
call   IncrementaMinutos
movlw  b'11111111'
movwf  Intermitencia       ; Visualiza siempre mientras incrementa.
call   VisualizaReloj       ; Visualiza mientras espera que deje
call   Retardo_200ms         ; de pulsar.
btfsf  IncrementarPulsador; Mientras permanezca pulsado
goto   Incrementar          ; incrementará el dígito.

FinIncrementar
return

```

"Incre  
variable  
corresp  
ajusta e

(La sub  
que se e  
variable

Figu

#### Subrutina "Incrementar"

Incrementa el valor de  
Este incremento se de

IncrementaMinutos
incf Mi
movf Mi
call Aj

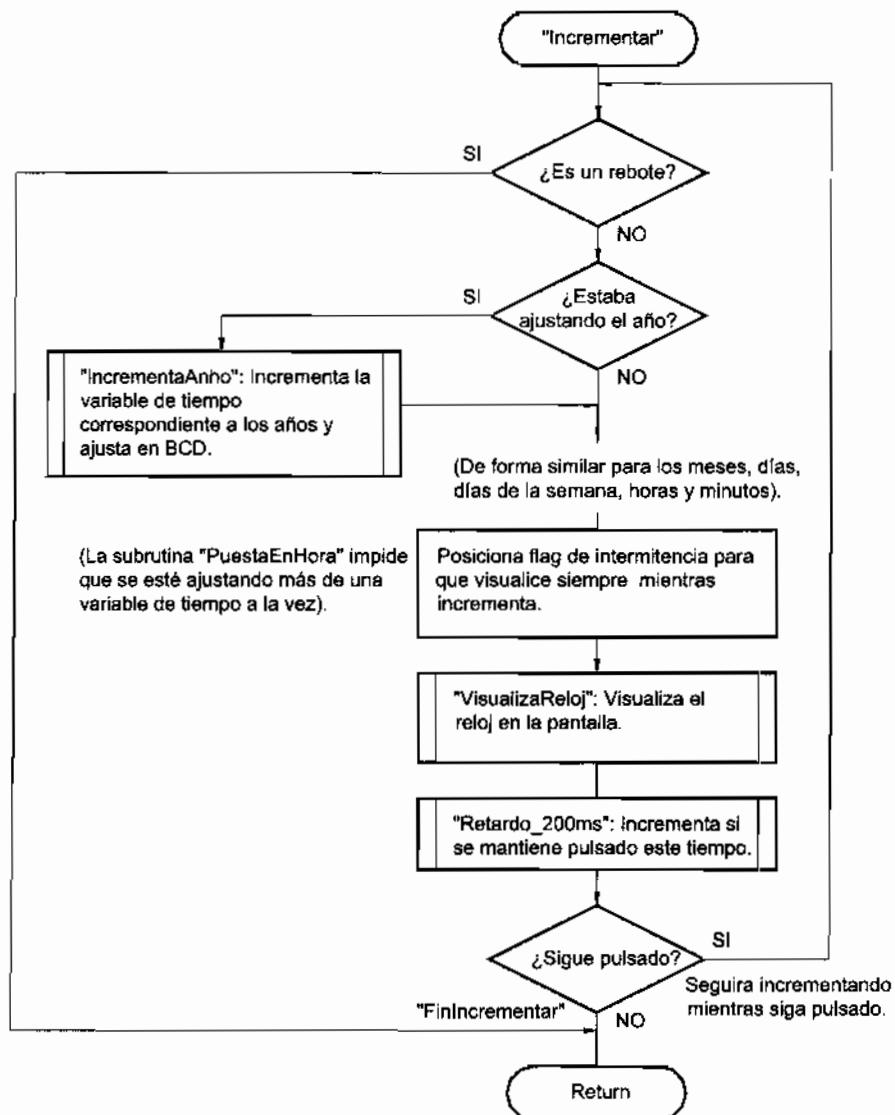


Figura 24-13 Diagrama de flujo de la subrutina Incrementar

; Subrutina "IncrementaMinutos"

; Incrementa el valor de la variable Minutos. Si supera los 0x59, lo resetea.  
; Este incremento se debe realizar en BCD para ello utiliza la subrutina AjusteBCD.

IncrementaMinutos

incf	Minuto,F	;	Incrementa los minutos,
movf	Minuto,W	;	Lo pasa a BCD.
call	AjusteBCD		

```

movwf Minuto           ; Lo guarda.
sublw 0x59             ; ¿Ha superado su valor máximo?
btfs STATUS,C          ; Lo inicializa si ha superado su valor máximo.
clrf Minuto
return

```

; Subrutina "IncrementaHoras"

IncrementaHoras

```

incf Hora,F            ; Incrementa las Horas.
movf Hora,W             ; Ahora hace el ajuste BCD.
call AjusteBCD
movwf Hora              ; Lo guarda.
sublw 0x23              ; ¿Ha superado su valor máximo?
btfs STATUS,C          ; Lo inicializa si ha superado su valor máximo.
clrf Hora
return

```

; Subrutina "IncrementaDiasSemana"

IncrementaDiasSemana

```

incf DiaSemana,F       ; Incrementa.
movf DiaSemana,W        ; Ahora hace el ajuste BCD.
sublw .7                ; ¿Ha superado su valor máximo?
btfs STATUS,C          ; Lo inicializa si ha superado su valor máximo.
goto FinIncrementaDiasSemana
movlw .1
movwf DiaSemana

```

FinIncrementaDiasSemana

return

; Subrutina "IncrementaDias"

;

IncrementaDias

```

incf Dia,F              ; Incrementa.
movf Dia,W               ; Ahora hace el ajuste BCD.
call AjusteBCD
movwf Dia                ; Lo guarda.
sublw 0x31              ; ¿Ha superado su valor máximo?
btfs STATUS,C          ; Lo inicializa si ha superado su valor máximo.
goto FinIncrementaDias
movlw .1
movwf Dia

```

FinIncrementaDias

return

; Subrutina "IncrementaMeses"

;

IncrementaMeses

```

incf Mes,F              ; Incrementa.
movf Mes,W               ; Ajusta a BCD.
call AjusteBCD
movwf Mes                ; Lo guarda.

```

```

sublw 0x1
btfs STA
goto Fin
movlw .1
movwf Mes
FinIncrementaMeses
return

```

; Subrutina "IncrementaA"

```

; Incrementa el valor de la hora.
; Es decir, llega hasta el año.
; Este incremento se debe hacer cada año.
;
```

IncrementaAnhos

```

incf Anho
movf Anho
call Ajus
movwf Anho
sublw 0x30
btfs STA
clrf Anho
return

```

; Subrutina "AjusteBCD"

```

; Esta subrutina pasa a BCD el valor "9", en cuyo caso
; (Minuto)=0x19, al incrementarle 6 resulta (Minuto)=0x1F.
;
```

```

; Entrada: En (W) el código decimal.
; Salida: En (W) el código BCD.
;
```

AjusteBCD

```

movwf Auxil
andlw b'0000
sublw .9
btfs STA
goto NoSuperaNueve
movlw .6
addwf Auxil
movf Auxil
return

```

INCLUDE <BU

INCLUDE <DS

INCLUDE <RE

INCLUDE <LC

INCLUDE <LC

END

```

        sublw  0x12          ; ¿Ha superado su valor máximo?
        btfsc STATUS,C
        goto  FinIncrementaMeses
        movlw  .i
        movwf  Mes
FinIncrementaMeses
        return

; Subrutina "IncrementaAnhos"
;
; Incrementa el valor de la variable (Anhos). Cuando llega a 0x30 (BCD), lo resetea.
; Es decir, llega hasta el año 2030, que se ha considerado suficientemente alto.
; Este incremento se debe realizar en BCD para ello utiliza la subrutina AjusteBCD.
;
IncrementaAnhos
        incf   Anho,F        ; Incrementa.
        movf   Anho,W
        call   AjusteBCD
        movwf  Anho
        sublw  0x30
        btfss STATUS,C
        clrf   Anho
        return

; Subrutina "AjusteBCD"
;
; Esta subrutina pasa a BCD el dato de entrada. Para ello detecta si las unidades superan
; el valor "9", en cuyo caso le suman 6 para pasarlo a BCD. Por ejemplo, si previamente
; (Minuto)=0x19, al incrementarle resulta (Minuto)=0x1A que no es un código válido BCD, si se
; le suma 6 resulta (Minuto)=0x1A+0x06=0x20 que es un código válido BCD.
;
; Entrada: En (W) el código a convertir
; Salida: En (W) el código convertido en BCD.
;
AjusteBCD
        movwf  Auxiliar        ; Guarda el valor del número a convertir.
        andlw  b'00001111'
        subiw  .9
        btfsc STATUS,C
        goto  NoSuperaNueve
        movlw  .6
        addwf  Auxiliar,F
NoSuperaNueve
        movf   Auxiliar,W
        return
; Retorna el resultado en el registro W.

INCLUDE <BUS_I2C.INC>      ; Subrutinas de control del bus I2C.
INCLUDE <DS1307.INC>        ; Subrutinas de control del DS1307.
INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

flotando

## CAPÍTULO 25

# SAA1064, CONTROLADOR DE DISPLAY

En muchos proyectos es necesario representar la información en displays de siete segmentos cuya conexión directa a un microcontrolador requiere de gran cantidad de líneas. También hay otros dispositivos que permiten conectarlos a un bus I2C con una utilización mínima de líneas como el que se estudiará en este capítulo.

## 25.1 SAA1064, CONTROLADOR DE DISPLAY

El SAA1064 es un controlador para cuatro displays de 7 segmentos a diodos LED compatible con bus I2C. Sus características más sobresalientes son:

- Es fabricado con el patillaje que se muestra en la figura 25-1 por *Philips Semiconductors*, ([www.semiconductors.philips.com](http://www.semiconductors.philips.com)).
- Está especialmente diseñado para gobernar cuatro displays de 7 segmentos de ánodo común tipo LED.
- La corriente por los segmentos puede ser controlada por software desde 0 mA (apagado) hasta 21 mA (máxima luminosidad).
- La alimentación puede variar entre 4,5 a 15 V, siendo 5 V su valor típico.
- Permite la multiplexación de dos pares de dígitos.

## 25.2 CIRCUITO TÍPICO PARA MODO ESTÁTICO

El SAA1064 puede trabajar en un modo denominado **estático** alimentando a dos displays cuyo circuito típico se muestra en la figura 25-2.

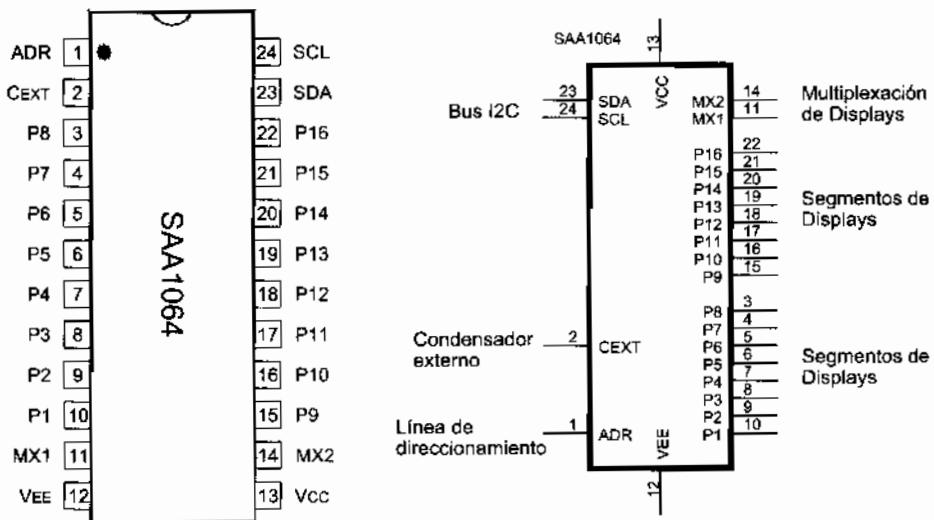


Figura 25-1 Patillaje del controlador de display SAA1064

El bus I2C se logra mediante las conexiones de los pines SCL y SDA del SAA1064 a las líneas RA3 y RA4 del microcontrolador. Las resistencias de Pull-Up tienen el valor característico de 4k7.

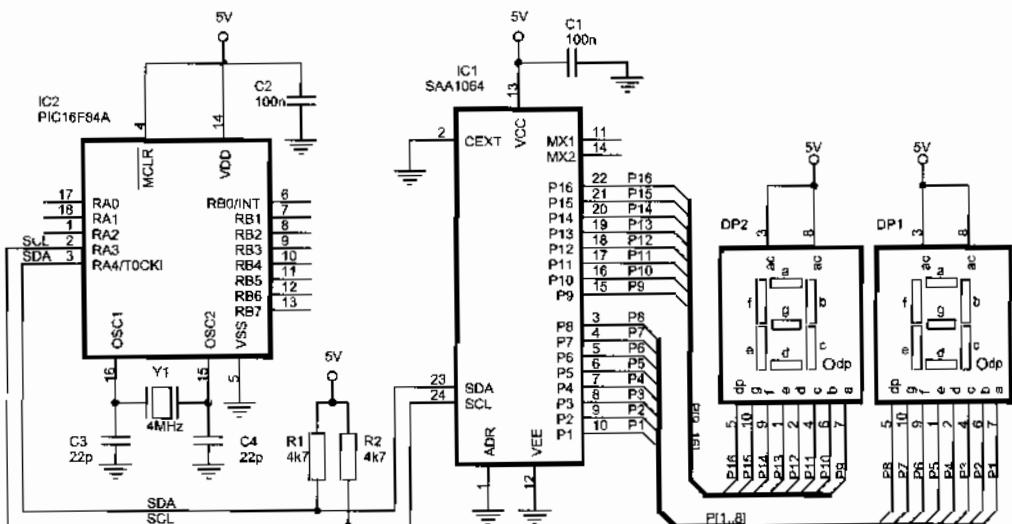


Figura 25-2 Circuito típico para el SAA1064 en modo de funcionamiento estático

Los displays tienen que ser de ánodo común, que se alimentan a  $V_{CC}$ . Los segmentos del display DP1 se conectan a las líneas P1 a P8 y los del display DP2 se

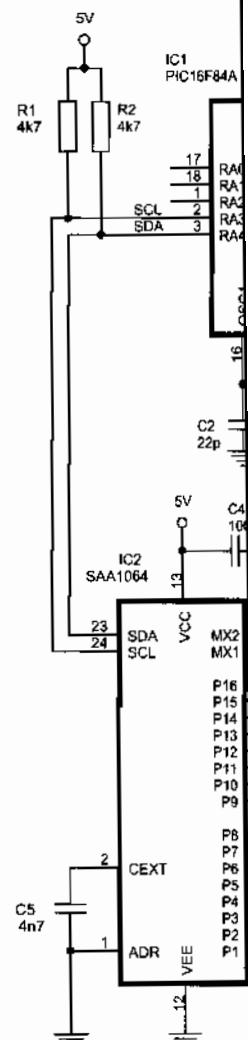


Figura 2.5-3 Circuito

conectan a las líneas P9 a P16. Un segmento se enciende si su correspondiente bit está a "1" y no necesita resistencia limitadora ya que el SAA1064 limita internamente la corriente.

Este circuito no es muy utilizado porque sólo permite gobernar dos displays que se encienden al mismo tiempo.

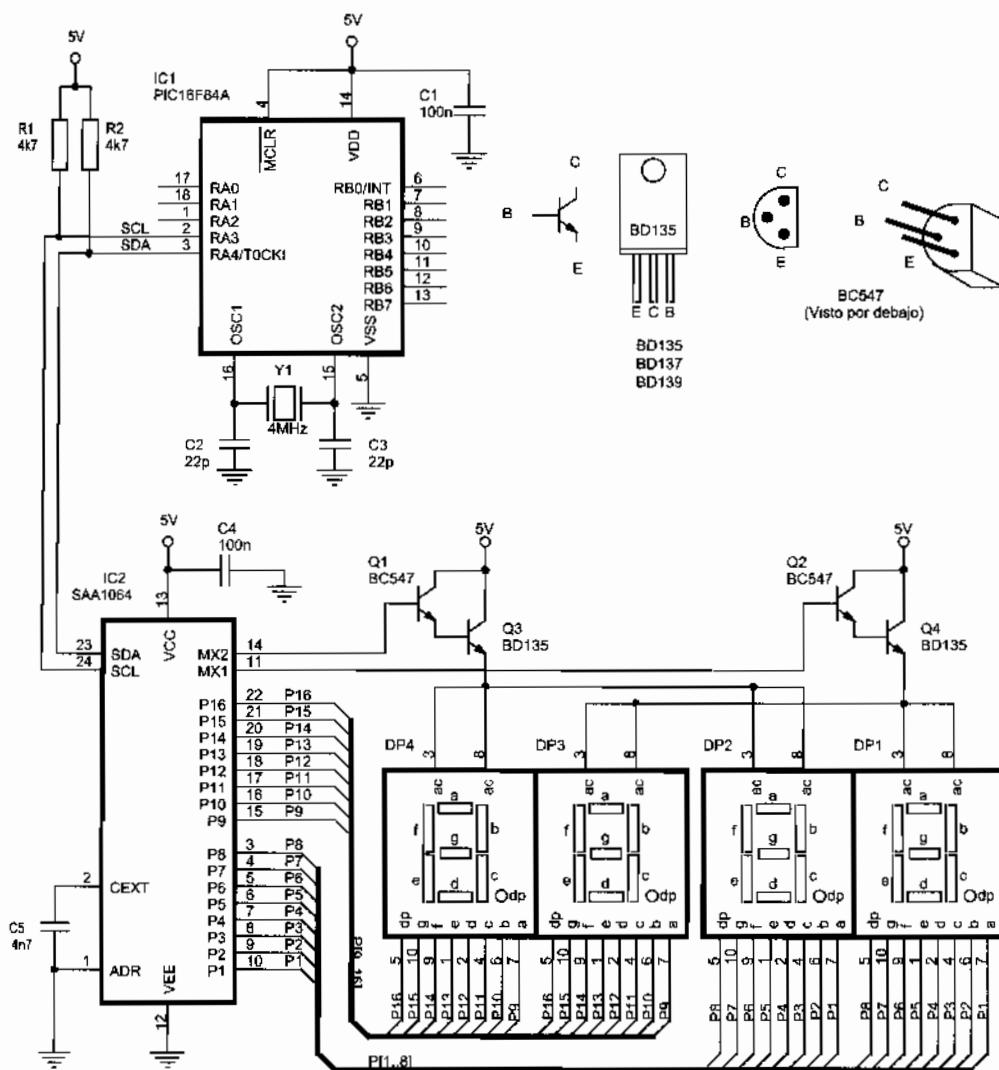


Figura 25-3 Circuito típico para el SAA1064 en modo de funcionamiento dinámico

## 25.3 CIRCUITO TÍPICO PARA MODO DINÁMICO

Para gobernar cuatro displays se utiliza el circuito típico de la figura 25-3. Se observa que el ánodo común de cada display se alimenta a  $V_{CC}$  a través de un par de transistores Darlington. Los segmentos de los displays 1 y 2 se conectan de la línea P1 a la P8 y los de los displays 3 y 4 se conectan de la línea P9 a la P16 del SAA1064. Un segmento se enciende si su correspondiente bit está a "1" y, además, los transistores conectados al terminal de ánodo común de los displays correspondientes están saturados.

Una pantalla con varios displays presenta el problema de su alto consumo, que puede ser excesivo si todos los segmentos de todos los displays están encendidos a la vez. Para solucionar esto se aplica el concepto de **visualización dinámica**, también conocido como **multiplexación de displays**.

La multiplexación de display consiste en activar sólo parte de los displays que conforman el conjunto de la pantalla, cambiando el encendido de unos a otros displays de forma secuencial de manera que no estén encendidos todos simultáneamente. Esto debe hacerse a velocidad tal que el ojo humano no sea capaz de detectarlo, aparentando así que todos los visualizadores están encendidos a la vez.

Este modo de funcionamiento del SAA1064 se llama **dinámico**, donde la multiplexación se consigue mediante los pines MX1 y MX2 con ayuda de dos pares de transistores Darlington, de los cuales sólo conduce un par a la vez, tal como se aclara en la tabla 25-1.

MX1	MX2	Q1 y Q3	Q2 y Q4	DP1 y DP3	DP2 y DP4	LÍNEAS P1 A P8	LÍNEAS P9 A P16
1	0	OFF	ON	Encendidos	Apagados	Datos para DP1	Datos para DP3
0	1	ON	OFF	Apagados	Encendidos	Datos para DP2	Datos para DP4
0	0	OFF	OFF	Apagados	Apagados	—	—

Tabla 25-1 Funcionamiento del modo dinámico o multiplexado

La conmutación de los pines MX1 y MX2 se gobierna mediante un oscilador interno y el condensador conectado a la patilla CEXT de un valor típico de 4,7 nF, que logra una frecuencia de multiplexación suficientemente elevada para evitar el parpadeo.

## 25.4 DIRECCIONAMIENTO COMO ESCLAVO

La figura 25-4 describe la dirección de esclavo para escritura del SAA1064, siendo A1 y A0 dos bits cuyo valor depende de la tensión aplicada al pin ADR según la tabla 25-2.

Figura

Tensión
0 V (0)
3/8 V
5/8 V
V

Tabla 25-2

En las figuras 25-3 y 25-4 se muestra la configuración para la dirección del SAA1064.

Se pueden programar los bits A1 y A0 directamente en el pin ADR, lo que simplifica la interfaz.

## 25.5 REGISTROS DE DATOS

El SAA1064 tiene 16 registros de datos. Los registros 0, 1, 2 y 3 se utilizan para la configuración de los displays y los demás para la escritura de los datos.

Reservados

Tabla

figura 25-3. Se  
s de un par de  
e la línea P1 a la  
SAA1064. Un  
los transistores  
tán saturados.

consumo, que  
ndidos a la vez.  
también conocido

s displays que  
ros displays de  
nte. Esto debe  
ntando así que

ico, donde la  
e dos pares de  
se aclara en la

#### LÍNEAS P9 A P16

Datos para DP3  
Datos para DP4

un oscilador  
e 4,7 nF, que  
l parpadeo.

1064, siendo  
según la tabla

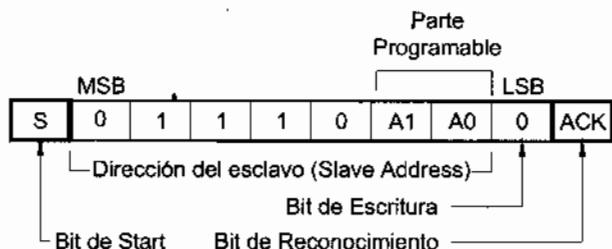


Figura 25-4 Dirección del SAA1064 como esclavo en un bus I2C

Tensión pin ADR	A1	A0	Dirección como esclavo para escritura b'0 1 1 1 0 (A1) (A0) 0'
0 V (masa).	0	0	b'01110000'
3/8 V <sub>CC</sub> .	0	1	b'01110010'
5/8 V <sub>CC</sub> .	1	0	b'01110100'
V <sub>CC</sub>	1	1	b'01110110'

Tabla 25-2 Direccionamiento del SAA1064 en función de la tensión en el pin ADR

En las figura 25-2 y 25-3 el pin ADR es llevado a masa, en consecuencia la dirección del SAA1064 será b'01110000' ó 70h.

Se pueden direccionar hasta cuatro diferentes SAA1064 mediante la conexión del pin ADR, lo que supone un total de 16 displays de 7 segmentos como máximo.

## 25.5 REGISTROS INTERNOS

El SAA1064 posee 5 registros internos con el direccionamiento indicado en la tabla 25-3. A partir de la dirección 05 se localizan registros de funciones reservadas no utilizados.

REGISTRO	DIRECCIÓN	SC	SB	SA
Registro de Control	00	0	0	0
Dígito 1	01	0	0	1
Dígito 2	02	0	1	0
Dígito 3	03	0	1	1
Dígito 4	04	1	0	0

Tabla 25-3 Registros internos del SAA1064 y su direccionamiento

Los registros "Digito x" contienen la información visualizada en los displays correspondientes.

El registro de control contiene 7 bits ( $b'x\ C6\ C5\ C4\ C3\ C2\ C1\ C0'$ ) cuyo significado es el siguiente: (El bit de mayor peso no se utiliza).

- $C0=0$ . El SAA1064 trabaja en modo estático o continuo, sólo puede controlar dos displays, como se muestra en la figura 25-2.
- $C0=1$ . El SAA1064 trabaja en modo dinámico controlando 4 displays multiplexando dos pares, como se muestra en la figura 25-3. Es el modo usual de funcionamiento.
- $C1=0$ . Los dígitos 1 y 3 están apagados.
- $C1=1$ . Los dígitos 1 y 3 no están apagados.
- $C2=0$ . Los dígitos 2 y 4 están apagados.
- $C2=1$ . Los dígitos 2 y 4 no están apagados.
- $C3=1$ . Todos los segmentos se encienden para testear su correcto funcionamiento.
- $C3=0$ . Funcionamiento normal.
- $C4=1$ . Añade 3 mA de corriente a cada segmento.
- $C5=1$ . Añade 6 mA de corriente a cada segmento.
- $C6=1$ . Añade 12 mA de corriente a cada segmento.

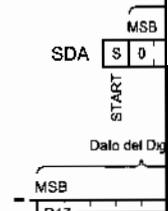
Para configurar el modo de funcionamiento dinámico el contenido del registro de control debe ser igual a  $b'x\ C6\ C5\ C4\ C3\ C2\ C1\ C0' = b'0\ C6\ C5\ C4\ 0\ 1\ 1\ 1'$ . La corriente que circula por cada segmento viene determinado por los bits de control C6, C5 y C4 según la tabla 25-4.

<b>C6 (Añade 12 mA)</b>	<b>C5 (Añade 6 mA)</b>	<b>C4 (Añade 3 mA)</b>	<b>Corriente por segmento</b>	<b>Registro de Control: <math>b'0\ C6\ C5\ C4\ 0\ 0\ 1\ 1\ 1'</math></b>
0	0	0	0 mA	$b'00000111'$
0	0	1	3 mA	$b'00010111'$
0	1	0	6 mA	$b'00100111'$
0	1	1	9 mA	$b'00110111'$
1	0	0	12 mA	$b'01000111'$
1	0	1	15 mA	$b'01010111'$
1	1	0	18 mA	$b'01100111'$
1	1	1	21 mA	$b'01110111'$

Tabla 25-4 Control de la corriente por segmento

## 25.6 ES

La tr  
procedimient  
la figura 25-5.



S = Cond  
P = Cond  
A = Acus  
X = Indife

- Prime
- Despu
- b'011
- A com
- de los
- inmed
- Si el r
- de co
- displa
- Por úl

En caso  
direcciones e  
permite al mi  
este esclavo ca

## 25.7 PR

Como d  
del SAA1064  
proyectos.

los displays

C0') cuyo

de controlar

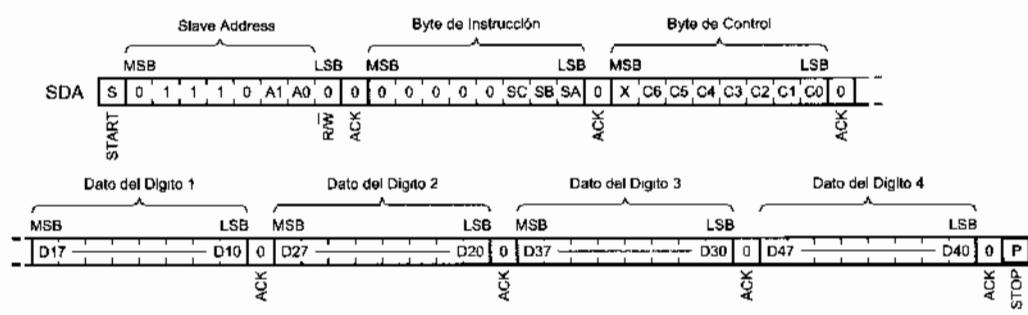
4 displays  
Es el modo

u correcto

registro de  
111'. La  
trol C6, C5Control:  
00111'111'  
111'  
111'  
111'  
111'  
111'  
111'

## 25.6 ESCRITURA EN EL SAA1064

La transferencia de datos desde el microcontrolador al SAA1064 sigue el procedimiento de escritura del maestro al esclavo de un bus I<sup>2</sup>C ya conocido y descrito en la figura 25-5.



S = Condición de Start.  
P = Condición de Stop.  
A = Acuse de recepción.  
X = Indiferente.

A1, A0 = Bits de dirección programable.  
SC, SB, SA = Bits de dirección de los registros internos.  
C6 a C0 = Bits de control  
D10 a D47 = Información para los displays.

Figura 25-5 Protocolo del proceso de escritura en el SAA1064

- Primero el microcontrolador maestro envía la condición de Start.
- Despues el byte de direccionamiento descrito en la sección anterior b'01110(A1)(A0)0'.
- A continuación el byte de instrucción donde los bits SC, SB y SA apuntan a uno de los ocho registros internos en el cual se escribirá el byte de datos que le sigue inmediatamente detrás, según la tabla 25-3.
- Si el registro direccionado es el de control a continuación debe enviarse la palabra de control. Despues se transmite la información que aparecerá en cada uno de los displays.
- Por ultimo, el maestro envía la condición de Stop.

En caso de enviar más de tres bytes los siguientes se escribirán en registros con direcciones consecutivas. Este hecho se denomina autoincremento de la subdirección y permite al microcontrolador maestro modificar rápidamente un contenido específico de este esclavo cambiando únicamente lo que desea cambiar.

## 25.7 PROGRAMA EJEMPLO

Como conclusión de lo explicado, el siguiente programa es un ejemplo del manejo del SAA1064 para el circuito de la figura 25-3 y de inmediata aplicación en múltiples proyectos.

```
***** I2C_Display_01.asm *****
; Visualiza la palabra "HOLA" en los displays de 7 segmentos conectados al SAA1064.
; ZONA DE DATOS *****

__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

SAA1064_Direccion EQU b'01110000' ; SAA1064 como esclavo en bus I2C

; ZONA DE CÓDIGOS *****

ORG 0
Inicio:
    call I2C_EnviaStart ; Envía condición de Start.
    movlw SAA1064_Direccion ; Apunta al SAA1064.
    call I2C_EnviaByte ; El registro de control está en la posición 0.
    cirw
    call I2C_EnviaByte ; Palabra de control para luminosidad media
    moviw b'91000111' ; (12 mA) y visualización dinámica multiplexada.
    call I2C_EnviaByte ; Escribe la palabra "HOLA" empezando por la
    moviw 'A' ; última letra debido a la disposición de los
    call ASCII_a_7Segmentos ; displays.
    call I2C_EnviaByte
    moviw 'L'
    call ASCII_a_7Segmentos
    call I2C_EnviaByte
    moviw 'O'
    call ASCII_a_7Segmentos
    call I2C_EnviaByte
    moviw 'H'
    call ASCII_a_7Segmentos
    call I2C_EnviaByte
    call I2C_EnviaStop ; Termina de enviar datos.

Principal:
    sleep ; Pasa a reposo.
    goto Principal

INCLUDE <DISPLAY_7S.INC>
INCLUDE <BUS_I2C.INC>
INCLUDE <RETARDOS.INC>
END
```

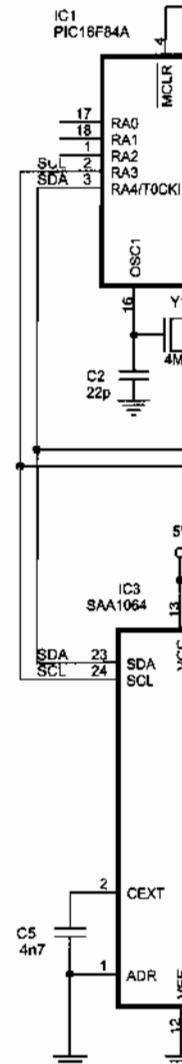


Figura 25-6

En el capítulo del DS1624 se muestra la utilización de los dispositivos para la visualización de la temperatura y un ejemplo de su aplicación.

## 25.8 TERMÓMETRO DE VISUALIZACIÓN EN DISPLAYS

Un proyecto clásico consiste en la visualización de la temperatura en cuatro displays de siete segmentos. La figura 25-6 muestra la forma de resolverlo mediante

dispositivos compatibles con bus I<sub>2</sub>C, en este caso un sensor DS1624 para medir la temperatura y un SAA1064 para controlar la visualización sobre displays de siete segmentos.

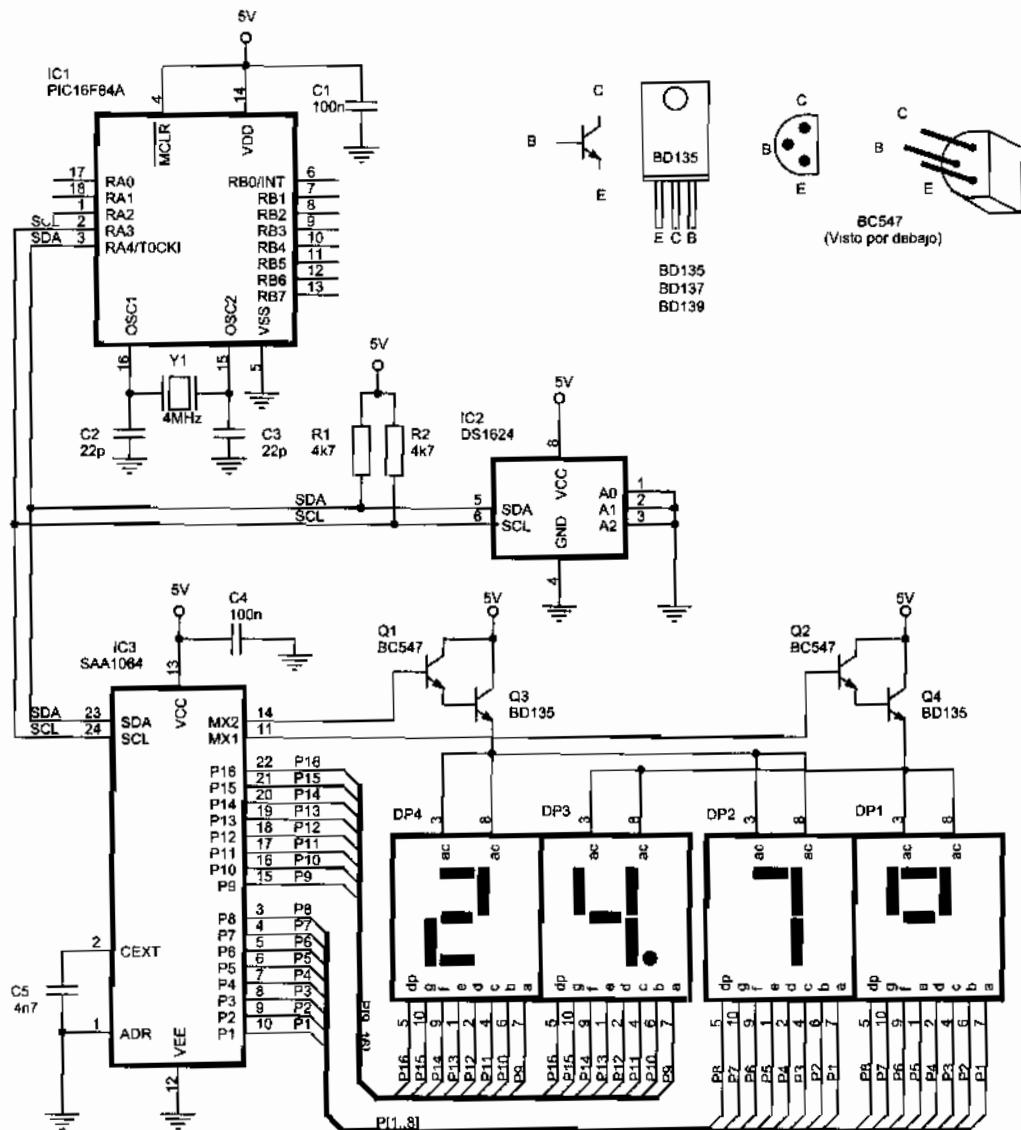


Figura 25-6 Termómetro con visualización en displays de siete segmentos

En el programa de control se aplican también los procedimientos explicados en el capítulo del DS1624. Sorprende la aparente sencillez del programa, lograda gracias a la utilización de los ficheros *Include*.

## DISPLAYS

eratura en cuatro  
solverlo mediante

```
***** I2C_Display_Termometro_01.asm *****
;
; Programa de control para un termómetro digital con el sensor de temperatura DS1624,
; y visualización en cuatro displays.
;
; ZONA DE DATOS *****

LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
Registro50ms           ; Guarda los incrementos cada 50 ms.
ENDC

TMR0_Carga50ms EQU    d'195'      ; Para conseguir interrupción cada 50 ms.
Carga2s          EQU    d'40'       ; Leerá cada 2s = 40 x 50ms = 2000ms.

;
; ZONA DE CÓDIGOS *****
ORG 0
goto Inicio
ORG 4
goto Termometro
Inicio
call DS1624_Inicializa ; Configura el DS1624 e inicia la conversión.
bsf STATUS,RP0
movlw b'00000111'        ; Preescaler de 256 para el TMR0.
movwf OPTION_REG
bcf STATUS,RP0
movlw TMR0_Carga50ms   ; Carga el TMR0 en complemento a 2.
movwf TMR0
movlw Carga2s          ; Y el registro cuyo decremento contará los 2 s.
movwf Registro50ms
movlw b'10100000'        ; Activa interrupción del TMR0 (T0IE),
movwf INTCON            ; y la general (GIE).
Principal
goto Principal
;
; Subrutina "Termometro"
;
; Esta subrutina lee y visualiza el termómetro cada 2 segundos aproximadamente. Se ejecuta
; debido a la petición de interrupción del Timer 0, cada 50 ms. Para conseguir una
; temporización de 2 s habrá que repetir 40 veces el lazo de 50 ms (40x50ms=2000ms=2s).
;
Termometro
movlw TMR0_Carga50ms
movwf TMR0
decfsz Registro50ms,F   ; Recarga el TMR0.
goto Fin_Termometro     ; Decrementa el contador.
; No han pasado 2 segundos y por lo tanto sale.
movlw Carga2s
movwf Registro50ms
call DS1624_LeeTemperatura ; Repone el contador nuevamente.
; Lee la temperatura.
```

call	DS1
call	Vis
<b>Fia_Termometro</b>	
bcf	INT
retfie	
;	
; Subrutina "VisualizaTe"	
;	
; Visualiza la temperatura	
; Se escribe comenzando	
;	
Entradas:	- (D)
;	- (D)
;	- (D)
SAA1064_Direccion	EQ
;	
<b>VisualizaTermometro</b>	
movf	DS1
call	BIN
movwf	DS1
call	I2C
movlw	SAA
call	I2C
clrw	
call	I2C
movlw	b'010
call	I2C
movlw	"
call	ASC
call	I2C
btfss	DS1
goto	Temp
;	
<b>TemperaturaNegativa:</b>	
movf	DS1
call	Num
call	I2C
swapf	DS1
call	Num
call	I2C
movlw	'1'
call	ASC
call	I2C
goto	Fin_V
;	
<b>TemperaturaPositiva</b>	
movf	DS1
call	Num
call	I2C
movf	DS1
call	Num
iorlw	b'1000
call	I2C
swapf	DS1

```
*****
    call    DS1624_IniciaConversion ; Comienza conversión para la siguiente lectura.
    call    VisualizaTermometro   ; Visualiza el resultado de la lectura.
Fin_Termometro
    bcf    INTCON,T0IF
    retfie
```

```
*****
; Subrutina "VisualizaTermometro"
;
; Visualiza la temperatura en los cuatro displays de siete segmentos controlados por el SAA1064.
; Se escribe comenzando por el final, por el display de la derecha.
;
; Entradas:      - (DS1624_Temperatura), temperatura medida en valor absoluto.
;                  - (DS1624_Decimal), parte decimal de la temperatura medida.
;                  - (DS1624_Signo), indica signo de la temperatura.
```

```
SAA1064_Direccion EQU b'01110000' ; SAA1064 como esclavo en bus I2C.
```

#### VisualizaTermometro

```
    movf   DS1624_Temperatura,W ; Hay que visualizar el valor absoluto de la
    call    BIN_a_BCD          ; temperatura en BCD.
    movwf  DS1624_Temperatura ; Guarda el valor de la temperatura en BCD.
    call    I2C_EnviaStart     ; Envía condición de Start.
    movlw  SAA1064_Direccion  ; Apunta al SAA1064.
    call    I2C_EnviaByte
    clrw
    call    I2C_EnviaByte      ; El registro de control está en la posición 0.
    movlw  b'01000111'         ; Palabra de control para luminosidad media
    call    I2C_EnviaByte      ; (12 mA) y visualización dinámica multiplexada.
    movlw  " "
    call    ASCII_a_7Segmentos ; Escribe el símbolo de grados.
    call    I2C_EnviaByte      ; Último carácter debido a la disposición de los
    btfss  DS1624_Signo,7     ; displays.
    goto   TemperaturaPositiva ; ¿Temperatura negativa?
                                ; No, la temperatura es positiva.
```

#### TemperaturaNegativa:

```
    movf   DS1624_Temperatura,W ; Recupera el valor de la temperatura en BCD.
    call    Numero_a_7Segmentos ; Las unidades se pasan a código siete segmentos.
    call    I2C_EnviaByte       ; Lo envía al display.
    swapf  DS1624_Temperatura,W ; Las decenas se pasan a código de siete
    call    Numero_a_7Segmentos ; segmentos.
    call    I2C_EnviaByte       ; Lo envía al display.
    movlw  "'"
    call    ASCII_a_7Segmentos ; Visualiza el signo "-" de temperatura negativa.
    call    I2C_EnviaByte       ; Lo pasa a siete segmentos.
    goto   Fin_VisualizaTemperatura ; Lo envía al display.
```

#### TemperaturaPositiva

```
    movf   DS1624_Decimal,W   ; Visualiza la parte decimal.
    call    Numero_a_7Segmentos ; Lo envía al display.
    call    I2C_EnviaByte
    movf   DS1624_Temperatura,W ; Recupera el valor de la temperatura en BCD.
    call    Numero_a_7Segmentos ; Las unidades se pasan a código siete segmentos.
    iorlw  b'10000000'
    call    I2C_EnviaByte       ; Le añade el punto decimal.
    swapf  DS1624_Temperatura,W ; Las decenas se pasan a código de siete
```

```

call    Numero_a_7Segmentos ; segmentos.
call    I2C_EnviaByte        ; Lo envía al display.

Fin_VisualizaTemperatura
return

INCLUDE <DISPLAY_7S.INC>
INCLUDE <BUS_I2C.INC>
INCLUDE <DS1624.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <RETARDOS.INC>
END

```

En algunos proyectos se necesitan conectar varias líneas a un bus serie I2C. Se hace necesario conectar el bus paralelo y el bus serie I2C. El chip PCF8574.

## 26.1 EL EXPANSOR

El chip PCF8574 es un dispositivo que permite la interconexión entre un bus serie I2C y un bus paralelo.

- Convertir datos de serie a paralelo
- Convertir los datos de paralelo a serie

Las características principales del PCF8574 son:

- Es fabricado por la firma Dallas Semiconductor.
- El encapsulado es tipo DIP-24.
- En las líneas P0, P1 y P2 se pueden configurar tres dispositivos en el mismo circuito.
- La salida está lógicamente invertida con respecto a la información hasta el dispositivo.
- Salida de alta corriente para conectar una corriente máxima de 20mA.

Custom Cards

## CAPÍTULO 26

# PCF8574, EXPANSOR DE BUS I2C

En algunos proyectos interesa conectar dispositivos que utilizan bus paralelo de varias líneas a un bus serie I2C para comunicarlos con otros utilizando únicamente dos líneas. Se hace necesario un interface que permita hacer la interconexión entre un bus paralelo y el bus serie I2C. A este dispositivo se le denomina **expansor** de bus I2C, como el PCF8574.

### 26.1 EL EXPANSOR DE BUS I2C PCF8574

El chip PCF8574 dispone de un puerto de entrada/salida y permite una interconexión entre un bus paralelo de ocho bits y un bus serie I2C, es decir, puede:

- Convertir datos paralelos de 8 bits a datos serie conectables a un bus I2C
- Convertir los datos serie de un bus I2C a datos paralelos de 8 bits.

Las características más sobresalientes del PCF8574 son:

- Es fabricado por *Philips Semiconductors*, ([www.semiconductors.philips.com](http://www.semiconductors.philips.com)) con el encapsulado que se muestra en la figura 26-1.
- En las líneas P0 a P7 se dispone de un puerto de 8 líneas de entrada/salida, que se pueden configurar todas como salidas o todas como entradas.
- Las tres líneas de dirección A0, A1 y A2 permiten conectar varios PCF8574 en el mismo circuito, variando la dirección de cada uno de ellos.
- La salida está latchead, cuando el puerto se configura como salida mantiene la información hasta una nueva escritura.
- Salida de alta corriente a nivel bajo, que le permite gobernar diodos LEDs con una corriente máxima de 25 mA.

- Con la salida en alto la corriente tiene un valor máximo de 300  $\mu$ A, incapaz de pilotar un diodo LED.
  - La alimentación puede variar entre 2,5 y 6 V, siendo 5 V su valor típico.
  - Consumo de corriente muy bajo. En modo bajo consumo, 10  $\mu$ A máximo.
  - Al aplicarle alimentación un dispositivo de puesta a cero coloca al puerto de salida en estado alto.
  - El pin INT es una salida en drenador abierto para interrupciones. Cuando cambia una línea en el puerto configurado como entrada, se activa el pin INT permitiendo solicitar una interrupción al microcontrolador.

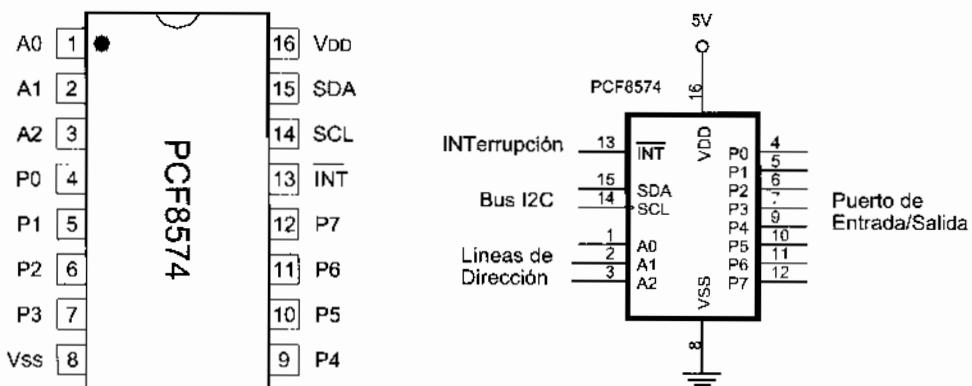


Figura 26-1 Patillaje del expansor de bus I2C PCF8574

## 26.2 DIRECCIONAMIENTO COMO ESCLAVO

Como otros dispositivos compatibles con bus I<sub>2</sub>C, el PCF8574 se activa cuando recibe la dirección válida ilustrada en la figura 26-2, para las dos versiones disponibles. Esta dirección consta de una parte fija y otra programable. La parte programable la define el conexionado de los pines A<sub>0</sub>, A<sub>1</sub> y A<sub>2</sub>.

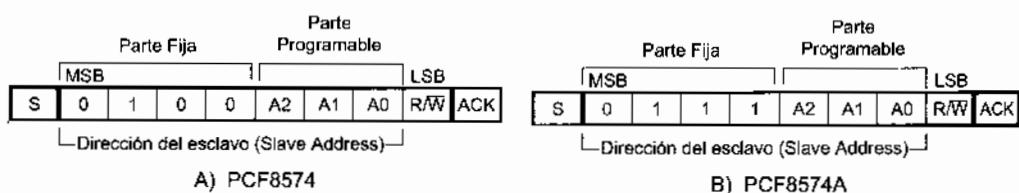


Figura 26-2 El PCF8574 y el PCF8574A se diferencian en su dirección de esclavo

Los tres hilos de direccionamiento hardware (A0, A1 y A2) del PCF8574 permiten conectar hasta ocho dispositivos (64 líneas de entrada/salida). Si se añaden otros ocho PCF8574A hacen un total posible de 128 líneas de entrada/salida.

### **26.3 ESCRIT**

La transferencia procedimiento de evaluación esquematizado en la



Fig

## La transferencia

- Primero el m
  - Luego envia  
binario es: 0  
líneas de dire
  - Después se t  
como salida.
  - Por ultimo el

## 26.4 LECTURE

La lectura de procedimiento de lectura en la figura 26-4

## La transference

- Primero el m
  - Luego envía binario es: 0 líneas de dire
  - Después el m
  - Por último el

10  $\mu$ A, incapaz de

or típico.

máximo.

loca al puerto de

s. Cuando cambia

INT permitiendo

Puerto de  
Entrada/Salida

se activa cuando  
ones disponibles.  
ramable la define

Parte  
ramable

LSB

A1 A0 R/W ACK

Address)

ón de esclavo

CF8574 permiten  
aden otros ocho

## 26.3 ESCRITURA EN EL PCF8574

La transferencia de datos desde el microcontrolador al PCF8574 sigue el procedimiento de escritura del maestro al esclavo de un bus I2C ya conocido y esquematizado en la figura 26-3

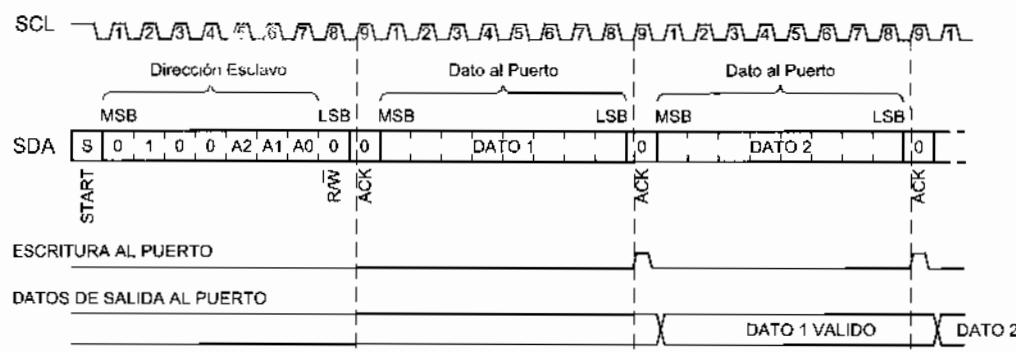


Figura 26-3 El maestro escribe datos en el PCF8574

La transferencia de datos se efectúa en el siguiente orden:

- Primero el microcontrolador maestro envía la condición de Start.
- Luego envía la dirección del PCF8574 (*Slave Address*) en modo escritura que en binario es: 0100(A2)(A1)(A0)0, siendo (A2), (A1) y (A0) el estado lógico de las líneas de direccionamiento del PCF8574.
- Despues se transmiten los datos a escribir hacia el puerto del PCF8574 que actúa como salida.
- Por último el microcontrolador maestro envía la condición de Stop.

## 26.4 LECTURA DEL PCF8574

La lectura de los datos del PCF8574 por parte del microcontrolador sigue el procedimiento de lectura del esclavo por parte del maestro ya conocido y esquematizado en la figura 26-4.

La transferencia de datos se efectúa en el siguiente orden:

- Primero el microcontrolador maestro envía la condición de Start.
- Luego envía la dirección del PCF8574 (*Slave Address*) en modo lectura que en binario es: 0100(A2)(A1)(A0)1, siendo (A2), (A1) y (A0) el estado lógico de las líneas de direccionamiento del PCF8574.
- Despues el maestro lee los datos del puerto del PCF8574 que actúa como entrada.
- Por último el microcontrolador maestro envía la condición de Stop.

Los datos del puerto de entrada no pueden variar más rápidamente que la velocidad de lectura del bus porque en ese caso la lectura quedaría corrompida. Se considera que los datos presentados al puerto de entrada no deben variar por encima de un ritmo aproximado de la décima parte de la señal SCL del bus.

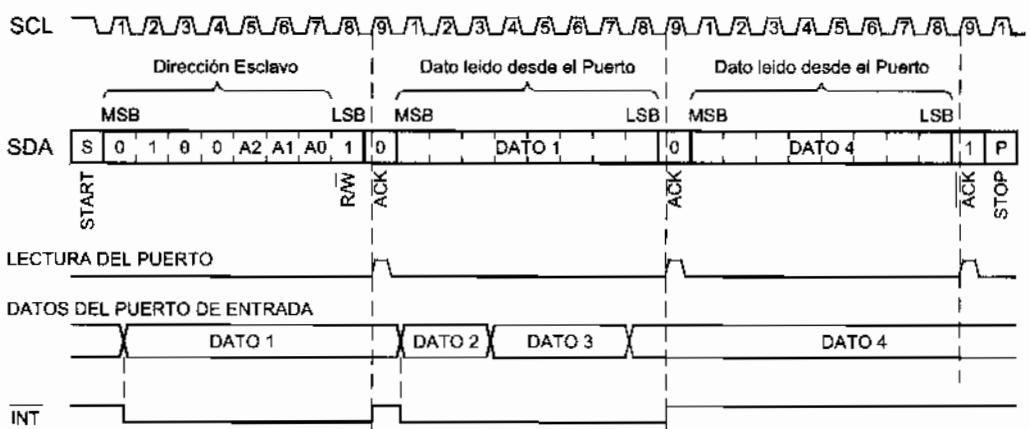


Figura 26-4 El maestro lee datos del PCF8574

## 26.5 LIBRERÍA DE SUBRUTINAS

El protocolo de escritura y lectura en el expansor PCF8574 se concreta en las siguientes subrutinas de control descritas en la librería PCF8574.INC:

- "PCF8574\_Escribe". El dato que le llega por el registro de trabajo W es enviado al puerto paralelo del expansor PCF8574 que actúa como salida.
- "PCF8574\_Lee". Lee el periférico conectado al puerto paralelo del PCF8574 que actúa como entrada.

```
***** Librería "PCF8574.INC" *****
;
; Estas subrutinas permiten realizar las tareas de manejo del expansor de bus I2C PCF8574
; que convierte un bus de 8 líneas paralelo a bus serie I2C y viceversa.
;
; Como es posible que haya más de un expansor en el mismo circuito, se deja a la subrutina
; de llamada a éstas que determine las direcciones de escritura y lectura mediante las
; etiquetas "PCF8574_DireccionLectura" y "PCF8574_DireccionEscritura", atendiendo a las
; características del hardware utilizado.
;
; ZONA DE DATOS *****
```

```
CBLOCK
PCF8574_Dato
ENDC
```

```
; Aquí guarda el resultado del dato leído o que
; se va a escribir.
```

```
; Subrutina "PCF8574_Lee"
;
; Lee el periférico conectado al puerto paralelo del PCF8574 que actúa como entrada y el resultado lo
; devolverá en el registro de trabajo W.
```

```
PCF8574_Lee
call I2C_
movlw PCF8574_
call I2C_
bsf I2C_
call I2C_
movwf PCF8574_
call I2C_
movf PCF8574_
return
```

```
; Subrutina "PCF8574_Escribe"
;
; Escribe en el periférico conectado al puerto paralelo del PCF8574 que actúa como salida, el dato que
; es recuperado finalmente se devolverá en el registro de trabajo W.
```

```
PCF8574_Escribe
movwf PCF8574_
call I2C_
movlw PCF8574_
call I2C_
movf PCF8574_
call I2C_
call I2C_
movf PCF8574_
return
```

MICROCONTR

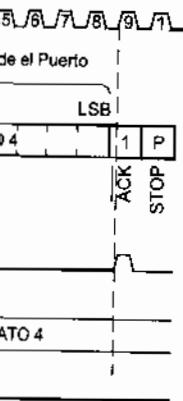
INT

Figura 26-

## 26.6 INTERRUPCIÓN

El PCF8574 posee una interfaz de interrupción que puede conectarse a la línea INT del PIC16F84. Al detectar un pulso bajo en la línea INT, el PCF8574 genera una interrupción que el PIC16F84 puede capturar para leer el nuevo dato.

que la velocidad  
considera que los  
rta de un ritmo



concreta en las

o W es enviado

el PCF8574 que

\*\*\*\*\*

o que

; Subrutina "PCF8574\_Lee" -----

;

; Lee el periférico conectado al bus paralelo de 8 líneas del PCF8574 que actúa como  
; entrada y el resultado lo proporciona en el registro de trabajo W.

PCF8574\_Lee

```

    call  I2C_EnviaStart          ; Envía la condición de Start.
    movlw PCF8574_DireccionLectura ; Apunta al expansor de lectura.
    call  I2C_EnviaByte
    bsf   I2C_UltimoByteLeer
    call  I2C_LeeByte            ; Lee el puerto.
    movwf PCF8574_Dato           ; Lo guarda en el registro correspondiente.
    call  I2C_EnviaStop
    movf  PCF8574_Dato,W         ; Acaba de leer.
    return

```

; Subrutina "PCF8574\_Escribe" -----

;

; Escribe en el periférico conectado al bus paralelo de 8 líneas del expansor PCF8574 que  
; actúa como salida, el dato que le llega por el registro de trabajo W. El dato escrito  
; es recuperado finalmente en el registro W.

PCF8574\_Escribe

```

    movwf PCF8574_Dato          ; Guarda el dato a escribir.
    call  I2C_EnviaStart          ; Envía condición de Start.
    movlw PCF8574_DireccionEscritura ; Apunta al expansor que actúa como salida.
    call  I2C_EnviaByte
    movf  PCF8574_Dato,W         ; Recupera el valor del dato a enviar.
    call  I2C_EnviaByte          ; Envía el dato.
    call  I2C_EnviaStop          ; Termina.
    movf  PCF8574_Dato,W         ; Recupera el valor del dato escrito.
    return

```

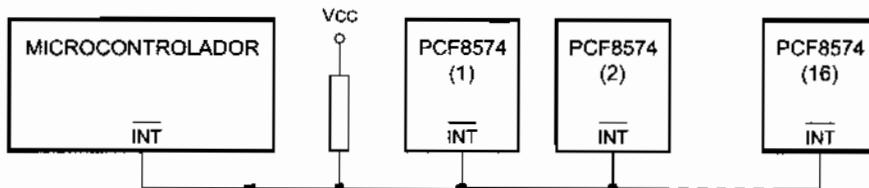


Figura 26-5 Conexión de la señal de interrupción de varios PCF8574

## 26.6 INTERRUPCIÓN

El PCF8574 proporciona una salida de interrupción (línea INT) en drenador abierto que puede conectarse a una entrada externa de interrupción del microcontrolador. Cuando el PCF8574 detecta un cambio en cualquiera de las líneas de entrada del puerto, se genera un pulso bajo como señal de interrupción por la línea INT para que el microcontrolador pueda leer el nuevo dato por el bus I2C. Esta señal solamente puede ser generada cuando

el puerto está configurado en modo entrada. Las señales INT de varios PCF8574 se conectan formando una And cableada, tal como indica la figura 26-5.

## 26.7 CONEXIÓN ENTRE PCF8574 Y PIC16F84

La figura 26-6 muestra un ejemplo de conexión entre un microcontrolador PIC16F84A y dos PCF8574, cuyos pines SDA y SCL se conectan a dos líneas del puerto A del microcontrolador conformando el bus I2C. Las resistencias de Pull-Up del bus I2C tienen un valor característico de 4k7.

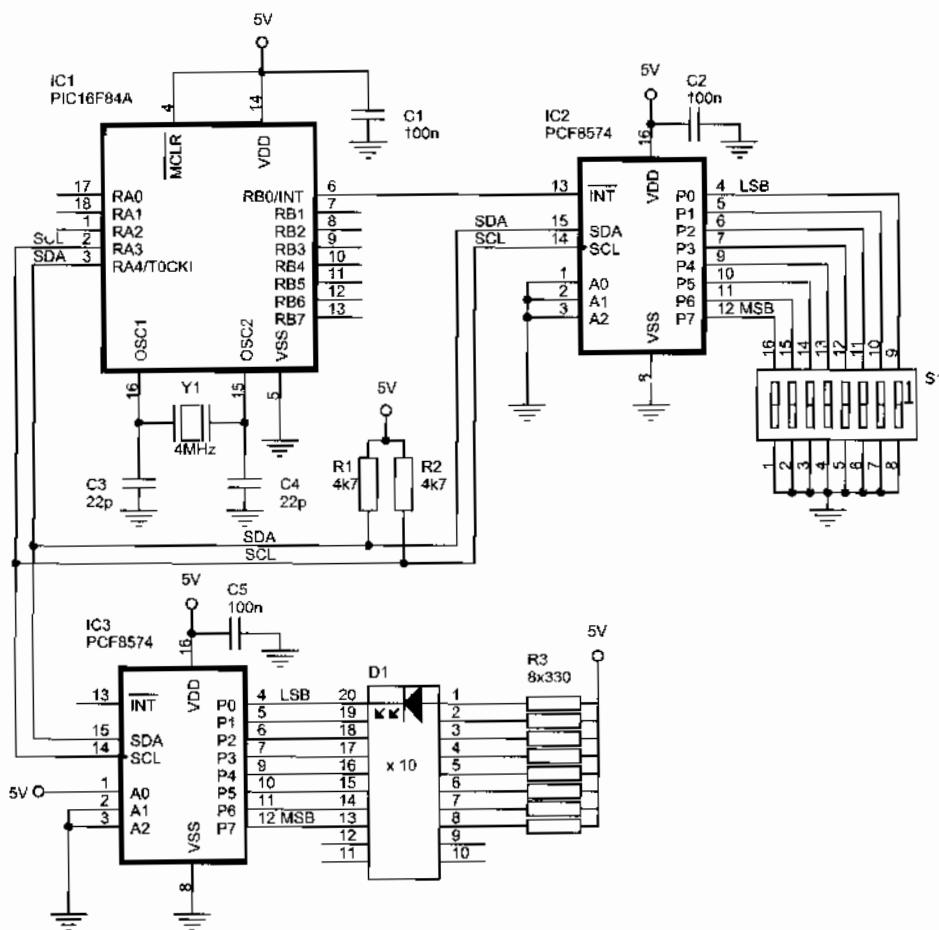


Figura 26-6 Conexión de dos PCF8574, uno como entrada y otro como salida de datos

El circuito integrado IC2 trabaja como entrada leyendo un array de interruptores y transmitiendo la información al PIC16F84A vía serie a través del bus I2C. Las líneas A2, A1 y A0 se conectan a masa por lo que su dirección de lectura es b'01000001'.

IC3 trabaja e información le llega a paralelo hacia los su dirección de esc

La línea de i línea RB0/INT del algún interruptor de del Puerto B media

## 26.8 EJEMPLO

Como ejemp para el circuito de l

```
*****
;
; Programa para compro
; Utiliza dos circuitos in
; - Uno como entrada, le
; - Otro como salida, vis
; conecta a Vcc.
;
; Este programa lee los
; visualiza en los LEDs
;
; ZONA DE DATOS *
```

```
CONFIG
LIST
INCLUDE
```

```
CBLOCK 0
Dato
ENDC
```

```
PCF8574_DireccionL
PCF8574_DireccionE
```

```
#DEFINE PCB8574
```

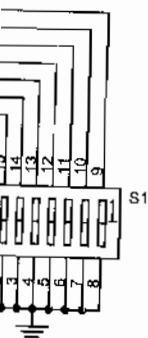
```
; ZONA DE CÓDIGO
```

```
ORG 0
goto Inicio
ORG 4
goto Inicio
ORG 8
bsf S
ORG 10
bsf S
ORG 12
bsf S
ORG 14
bsf S
ORG 16
bsf S
ORG 18
bsf S
ORG 20
bsf S
ORG 22
bsf S
ORG 24
bsf S
ORG 26
bsf S
ORG 28
bsf S
ORG 30
bsf S
ORG 32
bsf S
ORG 34
bsf S
ORG 36
bsf S
ORG 38
bsf S
ORG 40
bsf S
ORG 42
bsf S
ORG 44
bsf S
ORG 46
bsf S
ORG 48
bsf S
ORG 50
bsf S
ORG 52
bsf S
ORG 54
bsf S
ORG 56
bsf S
ORG 58
bsf S
ORG 60
bsf S
ORG 62
bsf S
ORG 64
bsf S
ORG 66
bsf S
ORG 68
bsf S
ORG 70
bsf S
ORG 72
bsf S
ORG 74
bsf S
ORG 76
bsf S
ORG 78
bsf S
ORG 80
bsf S
ORG 82
bsf S
ORG 84
bsf S
ORG 86
bsf S
ORG 88
bsf S
ORG 90
bsf S
ORG 92
bsf S
ORG 94
bsf S
ORG 96
bsf S
ORG 98
bsf S
ORG 100
bsf S
ORG 102
bsf S
ORG 104
bsf S
ORG 106
bsf S
ORG 108
bsf S
ORG 110
bsf S
ORG 112
bsf S
ORG 114
bsf S
ORG 116
bsf S
ORG 118
bsf S
ORG 120
bsf S
ORG 122
bsf S
ORG 124
bsf S
ORG 126
bsf S
ORG 128
bsf S
ORG 130
bsf S
ORG 132
bsf S
ORG 134
bsf S
ORG 136
bsf S
ORG 138
bsf S
ORG 140
bsf S
ORG 142
bsf S
ORG 144
bsf S
ORG 146
bsf S
ORG 148
bsf S
ORG 150
bsf S
ORG 152
bsf S
ORG 154
bsf S
ORG 156
bsf S
ORG 158
bsf S
ORG 160
bsf S
ORG 162
bsf S
ORG 164
bsf S
ORG 166
bsf S
ORG 168
bsf S
ORG 170
bsf S
ORG 172
bsf S
ORG 174
bsf S
ORG 176
bsf S
ORG 178
bsf S
ORG 180
bsf S
ORG 182
bsf S
ORG 184
bsf S
ORG 186
bsf S
ORG 188
bsf S
ORG 190
bsf S
ORG 192
bsf S
ORG 194
bsf S
ORG 196
bsf S
ORG 198
bsf S
ORG 200
bsf S
ORG 202
bsf S
ORG 204
bsf S
ORG 206
bsf S
ORG 208
bsf S
ORG 210
bsf S
ORG 212
bsf S
ORG 214
bsf S
ORG 216
bsf S
ORG 218
bsf S
ORG 220
bsf S
ORG 222
bsf S
ORG 224
bsf S
ORG 226
bsf S
ORG 228
bsf S
ORG 230
bsf S
ORG 232
bsf S
ORG 234
bsf S
ORG 236
bsf S
ORG 238
bsf S
ORG 240
bsf S
ORG 242
bsf S
ORG 244
bsf S
ORG 246
bsf S
ORG 248
bsf S
ORG 250
bsf S
ORG 252
bsf S
ORG 254
bsf S
ORG 256
bsf S
ORG 258
bsf S
ORG 260
bsf S
ORG 262
bsf S
ORG 264
bsf S
ORG 266
bsf S
ORG 268
bsf S
ORG 270
bsf S
ORG 272
bsf S
ORG 274
bsf S
ORG 276
bsf S
ORG 278
bsf S
ORG 280
bsf S
ORG 282
bsf S
ORG 284
bsf S
ORG 286
bsf S
ORG 288
bsf S
ORG 290
bsf S
ORG 292
bsf S
ORG 294
bsf S
ORG 296
bsf S
ORG 298
bsf S
ORG 300
bsf S
ORG 302
bsf S
ORG 304
bsf S
ORG 306
bsf S
ORG 308
bsf S
ORG 310
bsf S
ORG 312
bsf S
ORG 314
bsf S
ORG 316
bsf S
ORG 318
bsf S
ORG 320
bsf S
ORG 322
bsf S
ORG 324
bsf S
ORG 326
bsf S
ORG 328
bsf S
ORG 330
bsf S
ORG 332
bsf S
ORG 334
bsf S
ORG 336
bsf S
ORG 338
bsf S
ORG 340
bsf S
ORG 342
bsf S
ORG 344
bsf S
ORG 346
bsf S
ORG 348
bsf S
ORG 350
bsf S
ORG 352
bsf S
ORG 354
bsf S
ORG 356
bsf S
ORG 358
bsf S
ORG 360
bsf S
ORG 362
bsf S
ORG 364
bsf S
ORG 366
bsf S
ORG 368
bsf S
ORG 370
bsf S
ORG 372
bsf S
ORG 374
bsf S
ORG 376
bsf S
ORG 378
bsf S
ORG 380
bsf S
ORG 382
bsf S
ORG 384
bsf S
ORG 386
bsf S
ORG 388
bsf S
ORG 390
bsf S
ORG 392
bsf S
ORG 394
bsf S
ORG 396
bsf S
ORG 398
bsf S
ORG 400
bsf S
ORG 402
bsf S
ORG 404
bsf S
ORG 406
bsf S
ORG 408
bsf S
ORG 410
bsf S
ORG 412
bsf S
ORG 414
bsf S
ORG 416
bsf S
ORG 418
bsf S
ORG 420
bsf S
ORG 422
bsf S
ORG 424
bsf S
ORG 426
bsf S
ORG 428
bsf S
ORG 430
bsf S
ORG 432
bsf S
ORG 434
bsf S
ORG 436
bsf S
ORG 438
bsf S
ORG 440
bsf S
ORG 442
bsf S
ORG 444
bsf S
ORG 446
bsf S
ORG 448
bsf S
ORG 450
bsf S
ORG 452
bsf S
ORG 454
bsf S
ORG 456
bsf S
ORG 458
bsf S
ORG 460
bsf S
ORG 462
bsf S
ORG 464
bsf S
ORG 466
bsf S
ORG 468
bsf S
ORG 470
bsf S
ORG 472
bsf S
ORG 474
bsf S
ORG 476
bsf S
ORG 478
bsf S
ORG 480
bsf S
ORG 482
bsf S
ORG 484
bsf S
ORG 486
bsf S
ORG 488
bsf S
ORG 490
bsf S
ORG 492
bsf S
ORG 494
bsf S
ORG 496
bsf S
ORG 498
bsf S
ORG 500
bsf S
ORG 502
bsf S
ORG 504
bsf S
ORG 506
bsf S
ORG 508
bsf S
ORG 510
bsf S
ORG 512
bsf S
ORG 514
bsf S
ORG 516
bsf S
ORG 518
bsf S
ORG 520
bsf S
ORG 522
bsf S
ORG 524
bsf S
ORG 526
bsf S
ORG 528
bsf S
ORG 530
bsf S
ORG 532
bsf S
ORG 534
bsf S
ORG 536
bsf S
ORG 538
bsf S
ORG 540
bsf S
ORG 542
bsf S
ORG 544
bsf S
ORG 546
bsf S
ORG 548
bsf S
ORG 550
bsf S
ORG 552
bsf S
ORG 554
bsf S
ORG 556
bsf S
ORG 558
bsf S
ORG 560
bsf S
ORG 562
bsf S
ORG 564
bsf S
ORG 566
bsf S
ORG 568
bsf S
ORG 570
bsf S
ORG 572
bsf S
ORG 574
bsf S
ORG 576
bsf S
ORG 578
bsf S
ORG 580
bsf S
ORG 582
bsf S
ORG 584
bsf S
ORG 586
bsf S
ORG 588
bsf S
ORG 590
bsf S
ORG 592
bsf S
ORG 594
bsf S
ORG 596
bsf S
ORG 598
bsf S
ORG 600
bsf S
ORG 602
bsf S
ORG 604
bsf S
ORG 606
bsf S
ORG 608
bsf S
ORG 610
bsf S
ORG 612
bsf S
ORG 614
bsf S
ORG 616
bsf S
ORG 618
bsf S
ORG 620
bsf S
ORG 622
bsf S
ORG 624
bsf S
ORG 626
bsf S
ORG 628
bsf S
ORG 630
bsf S
ORG 632
bsf S
ORG 634
bsf S
ORG 636
bsf S
ORG 638
bsf S
ORG 640
bsf S
ORG 642
bsf S
ORG 644
bsf S
ORG 646
bsf S
ORG 648
bsf S
ORG 650
bsf S
ORG 652
bsf S
ORG 654
bsf S
ORG 656
bsf S
ORG 658
bsf S
ORG 660
bsf S
ORG 662
bsf S
ORG 664
bsf S
ORG 666
bsf S
ORG 668
bsf S
ORG 670
bsf S
ORG 672
bsf S
ORG 674
bsf S
ORG 676
bsf S
ORG 678
bsf S
ORG 680
bsf S
ORG 682
bsf S
ORG 684
bsf S
ORG 686
bsf S
ORG 688
bsf S
ORG 690
bsf S
ORG 692
bsf S
ORG 694
bsf S
ORG 696
bsf S
ORG 698
bsf S
ORG 700
bsf S
ORG 702
bsf S
ORG 704
bsf S
ORG 706
bsf S
ORG 708
bsf S
ORG 710
bsf S
ORG 712
bsf S
ORG 714
bsf S
ORG 716
bsf S
ORG 718
bsf S
ORG 720
bsf S
ORG 722
bsf S
ORG 724
bsf S
ORG 726
bsf S
ORG 728
bsf S
ORG 730
bsf S
ORG 732
bsf S
ORG 734
bsf S
ORG 736
bsf S
ORG 738
bsf S
ORG 740
bsf S
ORG 742
bsf S
ORG 744
bsf S
ORG 746
bsf S
ORG 748
bsf S
ORG 750
bsf S
ORG 752
bsf S
ORG 754
bsf S
ORG 756
bsf S
ORG 758
bsf S
ORG 760
bsf S
ORG 762
bsf S
ORG 764
bsf S
ORG 766
bsf S
ORG 768
bsf S
ORG 770
bsf S
ORG 772
bsf S
ORG 774
bsf S
ORG 776
bsf S
ORG 778
bsf S
ORG 780
bsf S
ORG 782
bsf S
ORG 784
bsf S
ORG 786
bsf S
ORG 788
bsf S
ORG 790
bsf S
ORG 792
bsf S
ORG 794
bsf S
ORG 796
bsf S
ORG 798
bsf S
ORG 800
bsf S
ORG 802
bsf S
ORG 804
bsf S
ORG 806
bsf S
ORG 808
bsf S
ORG 810
bsf S
ORG 812
bsf S
ORG 814
bsf S
ORG 816
bsf S
ORG 818
bsf S
ORG 820
bsf S
ORG 822
bsf S
ORG 824
bsf S
ORG 826
bsf S
ORG 828
bsf S
ORG 830
bsf S
ORG 832
bsf S
ORG 834
bsf S
ORG 836
bsf S
ORG 838
bsf S
ORG 840
bsf S
ORG 842
bsf S
ORG 844
bsf S
ORG 846
bsf S
ORG 848
bsf S
ORG 850
bsf S
ORG 852
bsf S
ORG 854
bsf S
ORG 856
bsf S
ORG 858
bsf S
ORG 860
bsf S
ORG 862
bsf S
ORG 864
bsf S
ORG 866
bsf S
ORG 868
bsf S
ORG 870
bsf S
ORG 872
bsf S
ORG 874
bsf S
ORG 876
bsf S
ORG 878
bsf S
ORG 880
bsf S
ORG 882
bsf S
ORG 884
bsf S
ORG 886
bsf S
ORG 888
bsf S
ORG 890
bsf S
ORG 892
bsf S
ORG 894
bsf S
ORG 896
bsf S
ORG 898
bsf S
ORG 900
bsf S
ORG 902
bsf S
ORG 904
bsf S
ORG 906
bsf S
ORG 908
bsf S
ORG 910
bsf S
ORG 912
bsf S
ORG 914
bsf S
ORG 916
bsf S
ORG 918
bsf S
ORG 920
bsf S
ORG 922
bsf S
ORG 924
bsf S
ORG 926
bsf S
ORG 928
bsf S
ORG 930
bsf S
ORG 932
bsf S
ORG 934
bsf S
ORG 936
bsf S
ORG 938
bsf S
ORG 940
bsf S
ORG 942
bsf S
ORG 944
bsf S
ORG 946
bsf S
ORG 948
bsf S
ORG 950
bsf S
ORG 952
bsf S
ORG 954
bsf S
ORG 956
bsf S
ORG 958
bsf S
ORG 960
bsf S
ORG 962
bsf S
ORG 964
bsf S
ORG 966
bsf S
ORG 968
bsf S
ORG 970
bsf S
ORG 972
bsf S
ORG 974
bsf S
ORG 976
bsf S
ORG 978
bsf S
ORG 980
bsf S
ORG 982
bsf S
ORG 984
bsf S
ORG 986
bsf S
ORG 988
bsf S
ORG 990
bsf S
ORG 992
bsf S
ORG 994
bsf S
ORG 996
bsf S
ORG 998
bsf S
ORG 1000
bsf S
```

varios PCF8574 se

4

microcontrolador  
os líneas del puerto  
ull-Up del bus I2C



IC3 trabaja en modo salida controlando la visualización de un array de LEDs. La información le llega procedente del PIC16F84 vía serie a través del bus I2C y la convierte a paralelo hacia los LEDs. Las líneas A2 y A1 se conectan a masa y A0 a V<sub>CC</sub>, por tanto, su dirección de escritura es b'010000010'.

La línea de interrupción del PCF8574 que actúa como entrada (IC2) se conecta a la línea RB0/INT del PIC16F84A produciendo una interrupción cada vez que se actúa sobre algún interruptor de entrada. La necesaria resistencia de Pull-Up se obtiene internamente del Puerto B mediante la adecuada programación.

## 26.8 EJEMPLO DE PROGRAMA

Como ejemplo de aplicación en el manejo del PCF8574 se expone un programa para el circuito de la figura 26-6, que es fácilmente adaptable a múltiples proyectos.

```
***** J2C_Expansor_01.asm *****
;
; Programa para comprobar el funcionamiento del PCF8574 que es un expansor de bus I2C.
; Utiliza dos circuitos integrados tal como se indica en el esquema correspondiente:
; - Uno como entrada, leyendo un array de 8 interruptores. Su pin A0 se conecta a masa.
; - Otro como salida, visualizando los datos en un array de diodos LEDs. Su pin A0 se
; conecta a Vcc.
;
; Este programa lee los switches conectados al PCF8574 que actúa como entrada y su valor se
; visualiza en los LEDs conectados al PCF8574 de salida.
;
; ZONA DE DATOS *****
;
; CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A
INCLUDE  <P16F84A.INC>

CBLOCK 0x0C
Dato
ENDC

PCF8574_DireccionLectura EQU    b'01000001'
PCF8574_DireccionEscritura EQU    b'01000010'

#DEFINE PCB8574_INT    PORTB,0           ; Línea donde se conecta la línea de
; interrupción del expansor de bus I2C.
;
; ZONA DE CÓDIGOS *****
;
ORG      0
goto    Inicio
ORG      4
goto    ServicioInterrupcion
Inicio
bsf     STATUS,RP0          ; Acceso al Banco 1.
bsf     PCB8574_INT         ; La línea RB0/INT se configura como entrada.
```

```

        bcf    OPTION_REG,NOT_RBPU ; Habilita la resistencia de Pull-Up del Puerto B.
        bcf    OPTION_REG,INTEDG  ; Interrupción INT activa por flanco de bajada.
        bcf    STATUS,RP0         ; Acceso al Banco 0.
        call   ServicioInterrupcion ; Para que lea y escriba por primera vez.
        movlw  b'10010000'       ; Habilita la interrupción INT y la general.
        movwf  INTCON

Principal
        sleep                         ; Pasa a modo de reposo.
        goto  Principal

; Subrutina "ServicioInterrupcion"
;
; Lee los interruptores conectados al PCF8574 que actúa como entrada y el resultado lo visualiza
; en los diodos LEDs conectados al PCF8574 configurado como salida.

ServicioInterrupcion
        call   Retardo_20ms          ; Espera unos instantes a que se estabilicen los
        call   PCF8574_Lee           ; niveles de tensión y lee la entrada.
        movwf  Dato                 ; Complementa el dato leído porque los diodos
        comaf  Dato,W               ; se activan con nivel bajo, (ver esquema).
        call   PCF8574_Escribe      ; Lo visualiza en los diodos LEDs.

        bcf    INTCON,INTF
        retfie

INCLUDE <BUS_I2C.INC>
INCLUDE <PCF8574.INC>
INCLUDE <RETARDOS.INC>
END

```

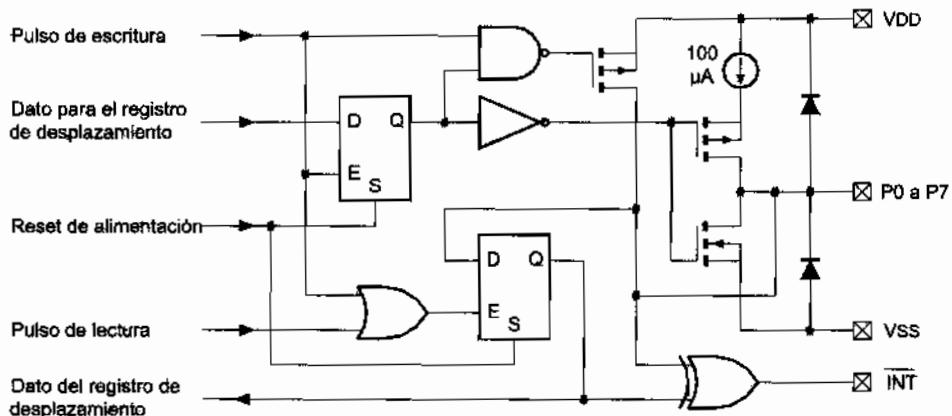


Figura 26-7 Constitución interna de las líneas del puerto del PCF8574

## 26.9 CONSTITUCIÓN INTERNA DEL PUERTO

La figura 26-7 muestra la constitución interna de cada una de las líneas del puerto del PCF8574. En la salida se observa una fuente de corriente de 100  $\mu$ A que hace

innecesaria la entrada.

Al conectar a nivel alto.

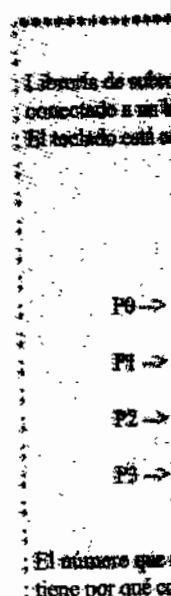
## 26.10 TECNICAS

Un tema de microcontroladores es la forma de resolución de conflictos entre los microcontroladores que comparten la misma interfaz que el PCF8574.

Un ejemplo de montaje es el que se muestra en la figura 26-8.

- Las filas P0 a P3 corresponden a la conexión P3 con P0 a P3.
- Las filas P4 a P7 corresponden a la conexión P4 a P7 con P4 a P7.

La técnica dedicada al tema de la figura 26-8 concluye con la figura 26-9.



El número que aparece en la figura 26-9 tiene por qué ser:

innecesaria la conexión de resistencias de Pull-Up cuando estas líneas actúan como entrada.

Al conectar la alimentación, las líneas del puerto quedan configuradas como salida a nivel alto.

## 26.10 TECLADO HEXADECIMAL EN BUS I2C

Un teclado hexadecimal ocupa ocho líneas, en algunos proyectos el microcontrolador no siempre dispone de suficientes pines libres para esta conexión. Una forma de resolver este problema es la conexión del teclado hexadecimal al microcontrolador mediante un bus I2C, utilizando para ello un expander PCF8574 como interfaz que adapte las ocho líneas del teclado hexadecimal al bus serie.

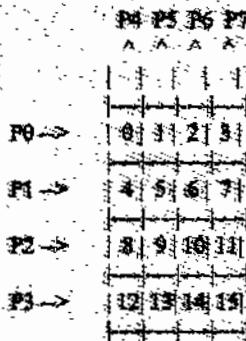
Un ejemplo de esta disposición se muestra en la figura 26-8. Se observa que el montaje es bastante parecido al de la conexión de un teclado matricial a un microcontrolador PIC16F84A, explicado en el capítulo 19. En este caso:

- Las filas del teclado se conectan al la parte baja del puerto PCF8574, líneas P0 a P3 configuradas como salida.
- Las columnas del teclado se conectan al la parte alta del puerto PCF8574, líneas P4 a P7, que actúan como entrada.

La técnica de exploración del teclado es muy similar a la explicada en el capítulo dedicado al teclado matricial, por lo que se remite a ese tema para su análisis y se concluye con la librería PCF8574\_TECLADO.INC que se expone a continuación.

\*\*\*\*\* Librería "PCF8574\_Teclado.INC" \*\*\*\*\*

Libería de subrutinas para la gestión de un teclado organizado en una matriz de 4 x 4 y conectado a un bus I2C a través de un PCF8574 tal como se muestra en el esquema.  
El teclado está conectado al puerto del expander PCF8574, siendo su disposición:



El número que se indica dentro de cada cuadrado es el número de orden de la tecla que no tiene por qué coincidir con lo serigrafiado sobre ella. El número de orden de la tecla

```

; se convierte al valor serigrafiado mediante una tabla de conversión.

; ZONA DE DATOS ****
;

CBLOCK
Tecl_TeclaOrden          ; Orden de la tecla a chequear.
ENDC

PCF8574_DireccionLectura EQU    b'01000001'
PCF8574_DireccionEscritura EQU    b'01000000'

Tecl_NumeroTeclas EQU    d'16'      ; Número de pulsadores del teclado.

#DEFINE PCF8574_INT     PORTB,0    ; Línea donde se conecta la línea de
; interrupción del expansor de bus I2C.

; Subrutina "Teclado_LeeHex"
;

; A cada tecla se le asigna un número de orden que se va contabilizando en la variable
; Tecl_TeclaOrden. Para convertir a su valor cada tipo de teclado se utiliza una tabla de
; conversión.

; A continuación se expone la relación entre el número de orden de la tecla y los valores
; correspondientes para el teclado hexadecimal más utilizado.

; ORDEN DE TECLA:           TECLADO HEX. UTILIZADO:
;   0 1 2 3                 1 2 3 F
;   4 5 6 7                 4 5 6 E
;   8 9 10 11                7 8 9 D
;  12 13 14 15               A 0 B C

; En este ejemplo, la tecla "7" ocupa el orden 8, la tecla "F" ocupa el orden 3 y la tecla
; "9" el orden 10.

; En teclados con otra serigrafía basta con cambiar de tabla de conversión.

; Entrada: En (W) el orden de la tecla pulsada.
; Salida: En (W) el valor hexadecimal para este teclado concreto.

; Teclado_LeeHex
call    Tecl_LeeOrdenTecla    ; Lee el Orden de la tecla pulsada
btfs s STATUS,C              ; ¿Pulsa alguna tecla?, ;C=1?
goto   Tecl_FinLeeHex       ; No, por tanto sale.
call    Tecl_ConvierteOrdenEnHex ; Lo convierte en su valor real mediante tabla.
bsf   STATUS,C               ; Vuelve a posicionar el Carry porque la
; instrucción "addwf PCL,F" lo pone a "0".
Tecl_FinLeeHex
return

; Tecl_ConvierteOrdenEnHex:
addwf  PCL,F
DT     1h,2h,3h,0Fh          ; Primera fila del teclado.
DT     4h,5h,6h,0Eh          ; Segunda fila del teclado
DT     7h,8h,9h,0Dh          ; Tercera fila del teclado.
DT     0Ah,0h,0Bh,0Ch        ; Cuarta fila del teclado.
;
```

```

Teclado_FinTablaHex
;
; Esta tabla se sitúa al principio OFF de memoria para
; visualizarla el siguiente programa.
;

IF (Teclado_Estado & ER_MIDI)
ENDIF

; Subrutina "Teclado_Inicializa"
;

Teclado_Inicializa
bsf   STI,0
bsf   PCI,0
bcf  ORI,0
bcf  OBI,0
bcf  STI,0
call  TECI
;
return

; Subrutina "Teclado_EsperaDejePulse"
;

; Permanece en esta subrutina hasta que se detecta la combinación que permite
; activar el teclado.

; Teclado_Comprobacion
;

Teclado_EsperaDejePulse
movlw  Teclado_Estado
call   PULSE
;
Teclado_SigueEsperando
call  READING
call  PULSE
sublw  TECI
btfs s STI,0
goto  TECI
;
return

; Subrutina "Teclado_LeeHex"
;

; Lee el teclado, obteniendo el orden de la tecla pulsada.

; Salida: En (W) el orden de la tecla pulsada.
; Además el bit C se pone a 1 si se pulsa una tecla.
; Si no se pulsa se pone a 0.

; CBLOCK
; Tecl_I2C_0
; Tecl_I2C_1
; ENDC
;
```

## Teclado\_FinTablaHex

```

*****  

;  

; Esta tabla se sitúa al principio de la librería con el propósito de que no supere la  

; posición OFF de memoria ROM de programa. De todas formas, en caso de que así fuera se  

; visualizaría el siguiente mensaje de error en el proceso de ensamblado:  

;  

    IF (Teclado_FinTablaHex-1 > 0xFF)  

        ERROR  "¡Cuidado!. La tabla ha superado el tamaño de la página de los"  

        MESSG  "primeros 256 bytes de memoria ROM. NO funcionará correctamente."  

    ENDIF

```

## ; Subrutina "Teclado\_Inicializa" -----

;

## Teclado\_Inicializa

```

    bsf    STATUS,RP0           ; Acceso al Banco 1.  

    bsf    PCF8574_INT         ; La línea RB0/INT se configura como entrada.  

    bcf    OPTION_REG,NOT_RBPU ; Habilita las resistencias de Pull-Up del Puerto B.  

    bcf    OPTION_REG,INTEDG   ; Interrupción INT activa por flanco de bajada.  

    bcf    STATUS,RP0           ; Acceso al Banco 0.  

;  

    call   Teclado_EsperaDejePulsar  

;  

    return

```

## ; Subrutina "Teclado\_EsperaDejePulsar" -----

;

; Permanece en esta subrutina mientras siga pulsada la tecla y deja en el puerto una  
; combinación que permite la interrupción de la próxima pulsación.

## Teclado\_Comprobacion EQU b'11110000'

## Teclado\_EsperaDejePulsar:

```

    movlw  Teclado_Comprobacion ; Pone a cero las cuatro líneas de salida del puerto
    call   PCF8574_Escribe      ; para que se produzca una interrupción cuando se
                                ; presione alguna tecla.

```

## Teclado\_SigueEsperando

```

    call   Retardo_20ms         ; Lee el puerto.
    call   PCF8574_Lee          ; Si es lo mismo que lanzó es que ya no pulsa
    sublw  Teclado_Comprobacion ; tecla alguna.
    btfss  STATUS,Z             ; 
    goto   Teclado_SigueEsperando
    return

```

## ; Subrutina "Teclado\_LeeOrdenTecla" -----

;

; Lee el teclado, obteniendo el orden de la tecla pulsada.

; Salida: En (W) el número de orden de la tecla pulsada.

; Además el flag Carry indica si se ha pulsado o no una tecla:

- Si se pulsa una tecla, (Carry)=1.

- Si no se pulsa tecla alguna, (Carry)=0.

## CBLOCK

```

    Tecl_I2C_DatoLeido          ; Guarda el dato leído.
    Tecl_I2C_FilaChequear       ; Fila que va a ser chequeada.

```

ENDC

Teclado Lee Orden Tecla:

	clrf	Tecl_TeclaOrden	; Todavía no ha empezado a chequear el teclado.
	movlw	b'11111110'	; Va a chequear primera fila.
	movwf	Tecl_I2C_FilaChequear	
Tecl_ChequeaFila			; (Ver esquema de conexión).
	call	PCF8574_Escribe	; Activa la fila correspondiente.
	call	PCF8574_Lee	; Lee el puerto.
	movwf	Tecl_I2C_DatoLeido	; Guarda la lectura efectuada.
Tecl_Columna1			
	btfss	Tecl_I2C_DatoLeido,4	; Chequea 1ª columna buscando un cero.
	goto	Tecl_GuardaValor	; Si, es cero. Salta a guardar su valor y sale.
	incf	Tecl_TeclaOrden,F	; No es cero. Va a chequear la siguiente tecla.
Tecl_Columna2			; Repite proceso para las siguientes columnas.
	btfss	Tecl_I2C_DatoLeido,5	
	goto	Tecl_GuardaValor	
	incf	Tecl_TeclaOrden,F	
Tecl_Columna3			
	btfss	Tecl_I2C_DatoLeido,6	
	goto	Tecl_GuardaValor	
	incf	Tecl_TeclaOrden,F	
Tecl_Columna4			
	btfss	Tecl_I2C_DatoLeido,7	
	goto	Tecl_GuardaValor	
	incf	Tecl_TeclaOrden,F	

Comprueba si ha chequeado la última tecla en cuyo caso sale. Para ello comprueba si el contenido del registro Tec1\_TeclaOrden es igual al número de teclas del teclado.

### Test Terms Columns

movlw	Tec1_NumeroTeclas	
subwf	Tec1_TeclaOrden,W	
bsf	STATUS,Z	
goto	Tec1_NoPulsada	
btf	STATUS,C	
rf	Tec1 IDC_FilaChequear,F	
movwf	Tec1 IDC_FilaChequear,W	
goto	Tec1_ChequearFila	
Tec1_NoPulsada		
btf	STATUS,C	
goto	Tec1_FinTecladolee	
Tec1_CuestionaValor		
movwf	Tec1_TeclaOrden,W	
btf	STATUS,C	
Tec1_FinTecladolee		
return		

Un ejemplo de aplicación puede ser el siguiente programa, de sorprendente sencillez:

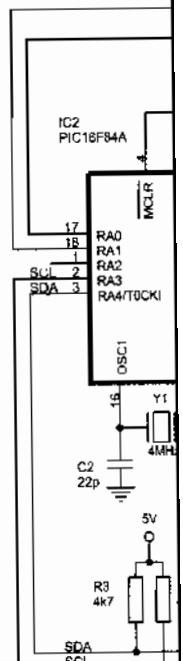


Figura 2

```
*****  
;  
;  
; Programa para co-  
; bus I2C a través  
; El módulo LCD  
;  
;  
; ZONA DE DATOS
```

CON  
LIST  
INCLUDE

CBLO  
ENDC

#DEFINE Zumb

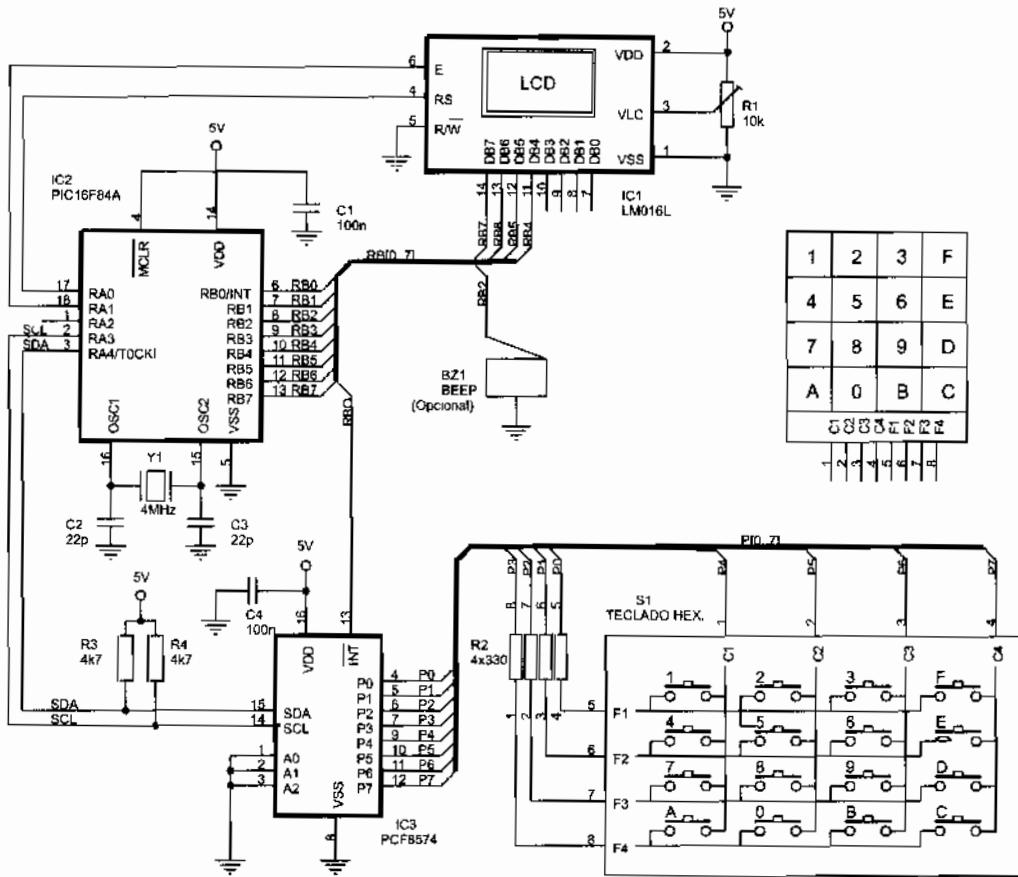


Figura 26-8 Conexión de teclado hexadecimal a bus I2C mediante PCF8574

\*\*\*\*\* I2C\_Teclado\_01.asm \*\*\*\*\*

; Programa para comprobar el funcionamiento y control de un teclado hexadecimal conectado a un  
; bus I2C a través de un expander PCF8574 tal como se indica en el esquema correspondiente.  
; El módulo LCD visualiza el valor hexadecimal de la tecla pulsada.

\*\*\*\*\* ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST    P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
ENDC
```

```
#DEFINE Zumbador      PORTB,2          ; Aquí se conecta el zumbador.
```

\*\*\*\*\* ZONA DE CÓDIGOS \*\*\*\*\*

```

ORG    0
goto   Inicio
ORG    4
goto   ServicioInterrupcion
Inicio
    call  LCD_Inicializa
    bsf   STATUS,RP0           ; Acceso al Banco 1.
    bcf   Zumbador            ; La linea del zumbador se define como salida.
    bcf   STATUS,RP0           ; Acceso al Banco 0.
    call  PitidoCorto          ; Pitido al conectarlo.
    call  Teclado_Inicializa
    movlw  b'10010000'          ; Prepara al PCF8574 para interrumpir.
    movwf  INTCON              ; Habilita la interrupción INT y la general.

Principal
    sleep                         ; Pasa a modo de bajo consumo.
    goto  Principal

; Subrutina "ServicioInterrupcion"
;
; Lee el teclado conectado al PCF8574 y el resultado se visualiza en la pantalla.

ServicioInterrupcion
    call  Teclado_LeeHex         ; Obtiene el valor hex. de la tecla pulsada.
    call  LCD_Nibble             ; Si, ha pulsado. Visualiza el valor en la
    call  PitidoCorto            ; pantalla y suena un pitido en el zumbador.
    call  Teclado_EspereDejePulsar ; Para que no se repita el mismo carácter.
    bcf   INTCON,INTF
    retfie

; Subrutina "PitidoCorto"
;
PitidoCorto
    bsf   Zumbador
    call Retardo_20ms
    bcf   Zumbador
    return

INCLUDE <PCF8574_Teclado.INC>
INCLUDE <BUS_I2C.INC>
INCLUDE <PCF8574.INC>
INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
END

```

La com  
de proyectos c  
integrado que  
analógico.

## 27.1 PIC

El PCF

- Un co
- Un co
- con cu

AINO
AIN1
AIN2
AIN3
A0
A1
A2
VSS

o salida.

eral.

**CAPÍTULO 27****PCF8591, ADC Y DAC EN BUS I2C**

La comunicación con el mundo analógico es uno de los temas que todo diseñador de proyectos con microcontroladores debe conocer. En este capítulo se explica un circuito integrado que mediante sólo dos líneas comunica al microcontrolador con el mundo analógico.

**27.1 PCF8591**

El PCF8591 es un dispositivo conectable a bus I<sub>2</sub>C que incluye:

- Un convertidor Digital-Analógico DAC (*Digital to Analog Converter*) de 8 bits.
- Un convertidor Analógico-Digital ADC (*Analog to Digital Converter*) de 8 bits, con cuatro entradas analógicas.

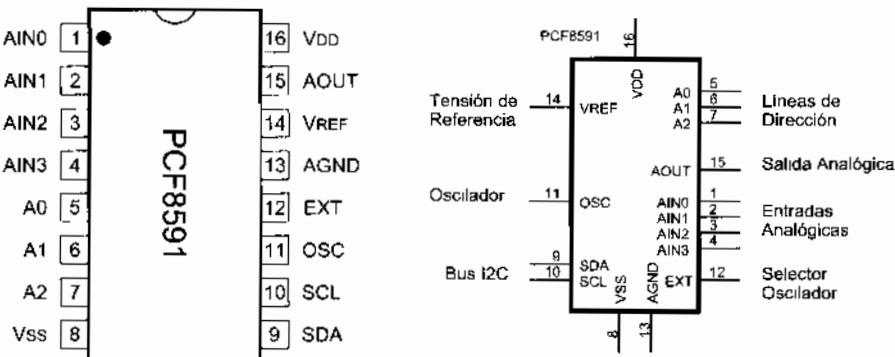


Figura 27-1 Patillaje del PCF8591

Este dispositivo es fabricado por *Philips Semiconductors* ([www.semiconductors.philips.com](http://www.semiconductors.philips.com)) en encapsulado de 16 pines con las funciones representadas en la figura 27-1.

- AIN0 ... AIN3. Entradas analógicas.
- SCL. Línea de reloj del bus I2C.
- SDA. Línea de datos del bus I2C.
- A0, A1 y A2. Entradas de dirección. Permite conectar varios PCF8591 en el mismo circuito variando la dirección de cada uno de ellos, tal como se explicará más adelante.
- V<sub>DD</sub> y V<sub>SS</sub>. Alimentación. Entre 2,5 a 6 V, siendo 5 V su valor típico.
- OSC. Entrada o salida del oscilador.
- EXT. Selector de oscilador externo o interno.
- AGND. Masa analógica.
- V<sub>REF</sub>. Entrada de tensión de referencia.
- AOUT: Salida analógica, proporciona una corriente máxima de 20 mA.

Su diagrama interno se representa en la figura 27-2.

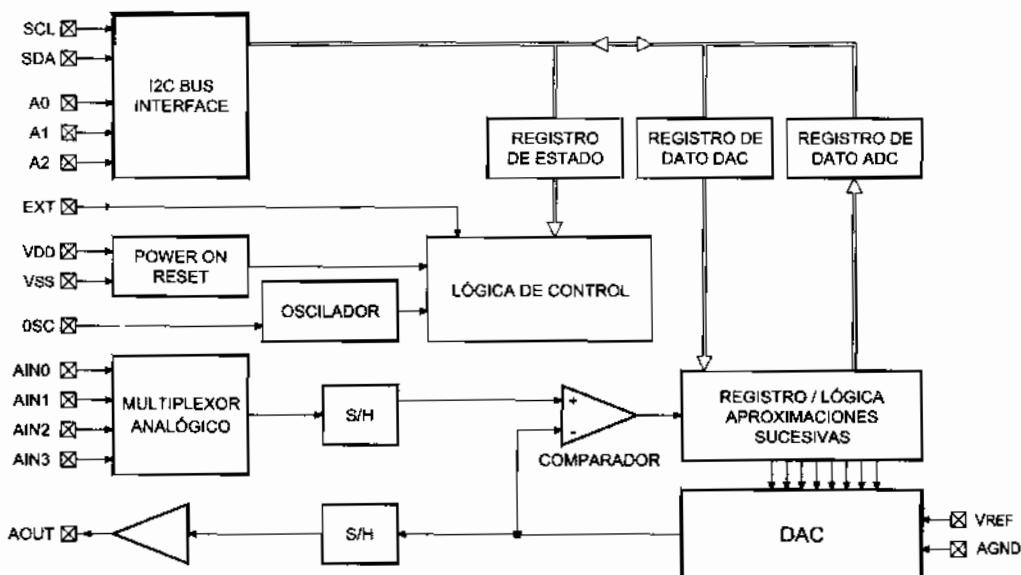


Figura 27-2 Diagrama interno del PCF8591

## 27.2 DIRECCIONAMIENTO COMO ESCLAVO

Como otros dispositivos compatibles con bus I2C, el PCF8591 se activa cuando recibe la dirección válida que se ilustra la figura 27-3. Esta dirección consta de una parte

fija y otra p  
A1 y A2.

Así p  
b'10011111  
hilos de di  
dispositivos,  
analógicas d

Se ha  
asignada al t  
líneas A0, A  
utilizarlos co

## 27.3 RE

El PC  
funcionamie  
figura 27-4.

- Los
- El bi
- auto
- El bi
- Los
- com
- El bi
- El bi

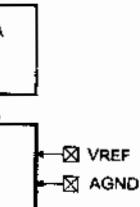
Esta p  
escritura sig

semiconductores  
las funciones

CF8591 en el  
o de ellos, tal  
típico.

a de 20 mA.

O DE  
OC



fija y otra programable. La parte programable la define el conexionado de los pines A0, A1 y A2.

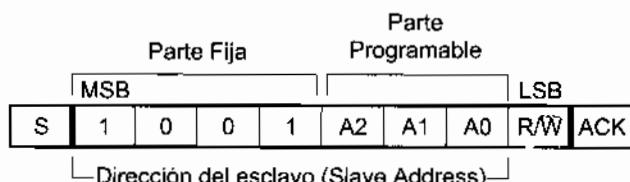


Figura 27-3 Dirección del PCF8591 como esclavo en un bus I2C

Así por ejemplo, si A2, A1 y A0 se conectan a la alimentación, la dirección será b'10011111' en lectura y b'10011110' en escritura. Esta programación mediante los tres hilos de direccionamiento hardware (A0, A1 y A2) permite conectar hasta ocho dispositivos, es decir, hasta 32 líneas de entradas analógicas de ADC y 8 líneas de salida analógicas de DAC.

Se hace la observación de que la parte fija de la dirección '1001' coincide con la asignada al termómetro DS1624. Por ello hay que tener precaución en la conexión de las líneas A0, A1 y A2 para que no coincida la dirección de ambos dispositivos en caso de utilizarlos conjuntamente.

### 27.3 REGISTRO DE CONTROL

El PCF8591 dispone del llamado **registro de control** encargado de configurar el funcionamiento del dispositivo. La palabra de control tiene la estructura descrita en la figura 27-4.

- Los bits 0 y 1 seleccionan una de las cuatro entradas analógicas para el ADC.
- El bit 2 es de autoincremento. Si está activado, el número de canal se incrementa automáticamente después de cada conversión del ADC.
- El bit 3 se pone a 0.
- Los bits 4 y 5 programan las entradas analógicas como simples o diferenciales, como muestra la figura.
- El bit 6 se emplea para habilitar la salida analógica cuando trabaja como DAC.
- El bit 7 no tiene utilidad alguna.

Esta palabra de control debe ser enviada por el microcontrolador maestro en modo escritura siguiendo la secuencia que ilustra la figura 27-5.

ctiva cuando  
de una parte

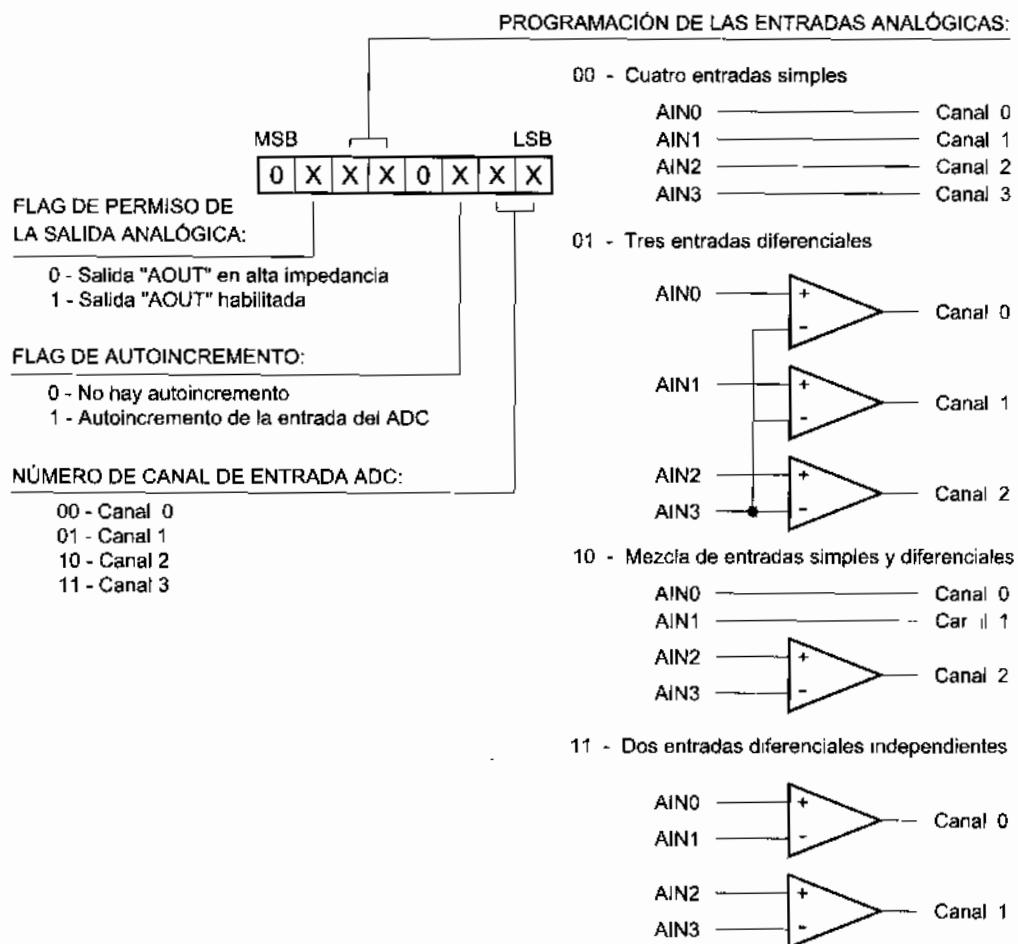


Figura 27-4 Byte de control

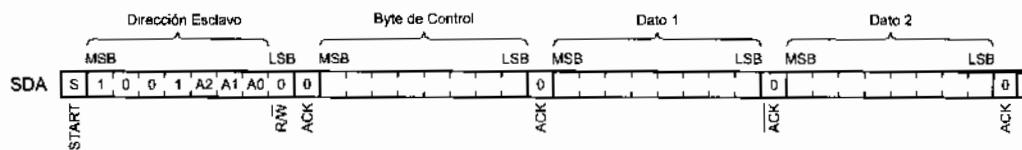
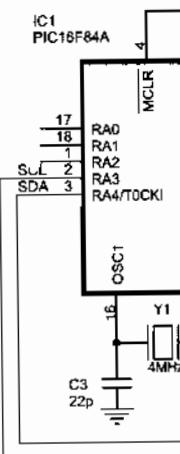


Figura 27-5 Envío del byte de control

## 27.4 EL PCF8591 COMO DAC

La figura 27-6 ilustra un ejemplo del PCF8591 trabajando como convertidor Digital - Analógico DAC (*Digital to Analog Converter*).



Figura

A destacar:

- La salida a corriente inversa a través del bus
- El bus I2C PCF8591 tienen el valor de
- La salida de registro de
- El valor de
- Tras un res
- El con
- El DA
- La sal
- La tensión la siguiente
- Los pines DAC.
- Selecciona
- Al configu
- escribir en la
- salida AOU escritura de
- Al estar co dispositivo

- SÍMBOLOS ANALÓGICOS:**
- Canal 0
  - Canal 1
  - Canal 2
  - Canal 3
  

**es**

    - Canal 0
    - Canal 1
    - Canal 2  

**res y diferenciales**

    - Canal 0
    - Canal 1
    - Canal 2  

**s independientes**

    - Canal 0
    - Canal 1  

**Dato 2**

```

    Dato 2
      |
      |----(LSB)
      |----(0)
      |----(ACK)
  
```

**o convertidor**

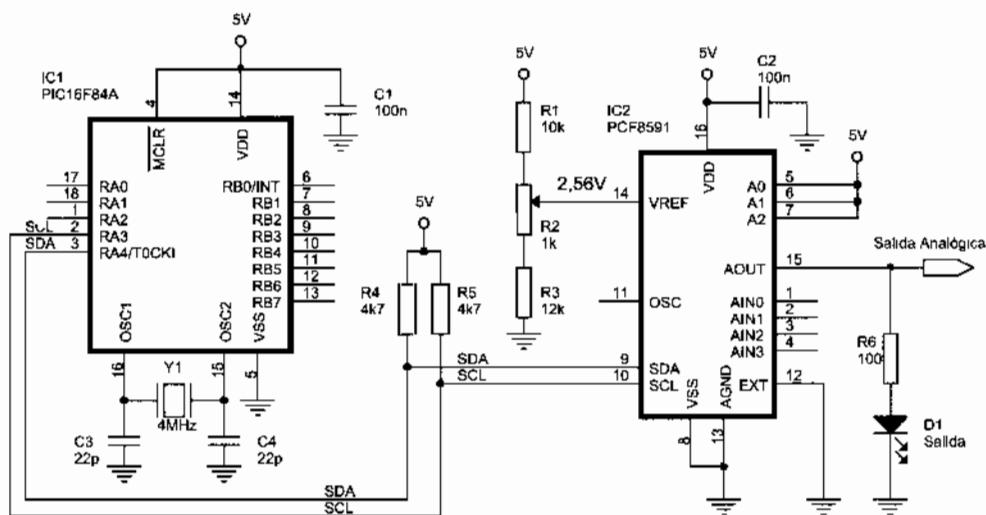


Figura 27-6 Ejemplo de configuración del PCF8591 como DAC

A destacar:

- La salida analógica se obtiene de la patilla AOUT, que puede proporcionar una corriente máxima de 20 mA. El valor digital de entrada a convertir se envía a través del bus I2C.
- El bus I2C se logra mediante las conexiones de los pines SCL y SDA del PCF8591 a líneas RA3 y RA4 del microcontrolador. Las resistencias de Pull-Up tienen el valor característico de 4k7.
- La salida analógica puede ser bloqueada mediante el bit de habilitación del registro de control (figura 27-4)
- El valor de la salida analógica se mantiene hasta que llega el dato siguiente.
- Tras un reset:
  - El contenido del registro de control es 00h, el DAC.
  - El DAC y el oscilador quedan inhabilitados para ahorrar energía.
  - La salida analógica queda en alta impedancia.
- La tensión aplicada al pin VREF es de 2,56 V, por el motivo que se explicará en la siguiente sección.
- Los pines AINx y OSC no se utilizan cuando el PCF8591 se configura como DAC.
- Selecciona el oscilador interno llevando el pin EXT a masa.
- Al configurarse el PCF8591 como DAC el microcontrolador maestro debe escribir en el dispositivo esclavo el dato digital a convertir, obteniéndose en su salida AOUT la tensión analógica. Por tanto, sólo es necesaria la dirección de escritura del PCF8591.
- Al estar conectados A0, A1 y A2 a V<sub>CC</sub> la dirección de esclavo en escritura del dispositivo es la b'1001(A2)(A1)(A0)0' = b'10011110'.

El valor digital de entrada a convertir se envía a través del bus I<sub>2</sub>C siguiendo el formato descrito en la figura 27-7.

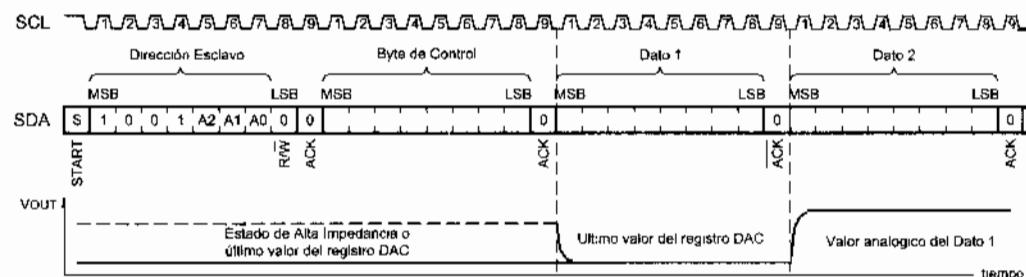


Figura 27-7 Secuencia de conversión cuando el PCF8591 actúa como DAC

- Primero el microcontrolador maestro envía la condición de Start.
- Luego envía la dirección del PCF8591 (*Slave Address*) en modo escritura que en binario es: 1001(A2)(A1)(A0)0, siendo (A2), (A1) y (A0) el estado lógico de las líneas de direccionamiento del PCF8591.
- A continuación se envía la palabra de control.
- Después se transmiten los datos digitales para convertir a valor analógico.
- El proceso termina cuando el microcontrolador maestro envía la condición de Stop.

## 27.5 RESOLUCIÓN DEL DAC

El PCF8591 convierte el dato digital de entrada al DAC a su correspondiente tensión analógica de salida según la siguiente ecuación:

$$V_{AOUT} = LSB \cdot \sum_{i=0}^7 Di \cdot 2^i$$

Siendo:

- **LSB**, la denominada **resolución** del DAC cuyo valor es controlado por la tensión en el pin V<sub>REF</sub> del PCF8591, según la siguiente ecuación, suponiendo que la masa analógica (pin AGND) está conectada a masa:

$$LSB = \frac{V_{REF}}{256}$$

- La expresión  $\sum_{i=0}^7 Di \cdot 2^i$  es el valor del dato digital de entrada que se desea convertir en analógico.

**EJEMPLO:** Deducir la tensión en la salida analógica AOUT del PCF8591 configurado como DAC del esquema de la figura 27-6, donde la tensión aplicada al pin

V<sub>REF</sub> es de 2,56 b'01001011'.

Solución: L

El dato de e

Por tanto el

V<sub>AOUT</sub>

La resolución ADC) ya que relaci

La resolución que puede experimentar digital. Es el valor llama "LSB" (Leas

En un DAC los casos como el ejemplo anterior tie

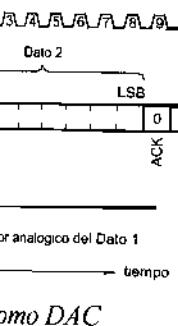
FS

Cuando la t mV que es un val VREF es de unos utilizar el divisor d 12 k apenas cargar tensión de referencia

La exactitud resolución del circ V se mantenga mu referencia necesaria (www.national.co

I2C siguiendo el

VREF es de 2,56 voltios, el pin AGND está conectado a masa y el dato de entrada es b'01001011'.



escritura que en  
do lógico de las

alógico.  
la condición de

correspondiente

trolado por la  
on, suponiendo

a que se desea

del PCF8591  
aplicada al pin

**Solución:** La resolución LSB del circuito es:

$$LSB = \frac{V_{REF}}{256} = \frac{2,56}{256} = 0,01V = 10mV$$

El dato de entrada es:  $D_i \cdot 2^i = b'01001011' = 75$  expresado en decimal.

Por tanto el valor de la tensión de salida por el pin AOUT del PCF8591 es igual a:

$$V_{AOUT} = LSB \cdot \sum_{i=0}^7 Di \cdot 2^i = 10 \cdot 75 = 750mV = 0,75V$$

La resolución LSB es el parámetro más importante de un DAC (y también de un ADC) ya que relaciona los valores analógicos y los digitales según la ecuación anterior.

La resolución de un convertidor DAC se define también como la menor variación que puede experimentar la salida analógica como resultado de un cambio en la entrada digital. Es el valor de la salida para un valor digital igual a b'00000001', por lo que se le llama "LSB" (*Least Significant Bit*) que es el valor de ponderación del bit de menor peso.

En un DAC es importante identificar el *Full Scale (FS)*, definido en la mayoría de los casos como el máximo valor que puede alcanzar la señal analógica. Así, el circuito del ejemplo anterior tiene un FS igual a:

$$FS = LSB \cdot 11111111_2 = 10 \cdot 255_{10} = 2550mV = 2.55V$$

Cuando la tensión en el pin VREF es de 2,56 V se consigue una resolución de 10 mV que es un valor muy cómodo con el que trabajar. La impedancia vista desde el pin VREF es de unos 100 k, por tanto para conseguir los 2,56 V en esta patilla se puede utilizar el divisor de tensión descrito en la figura 27-6, donde las resistencias fijas de 10 y 12 k apenas cargan al dispositivo, con la resistencia variable de 1 k es muy fácil ajustar la tensión de referencia a 2,56 V.

La exactitud en el proceso de conversión es determinada por el valor de la resolución del circuito. Por esto es muy importante que esta tensión de referencia de 2,56 V se mantenga muy estable. La figura 27-8 muestra otra forma de conseguir la tensión de referencia necesaria mediante el dispositivo LM336-2.5 V de Nacional Semiconductor ([www.national.com](http://www.national.com)), que proporciona una tensión de 2.5 V muy estable.

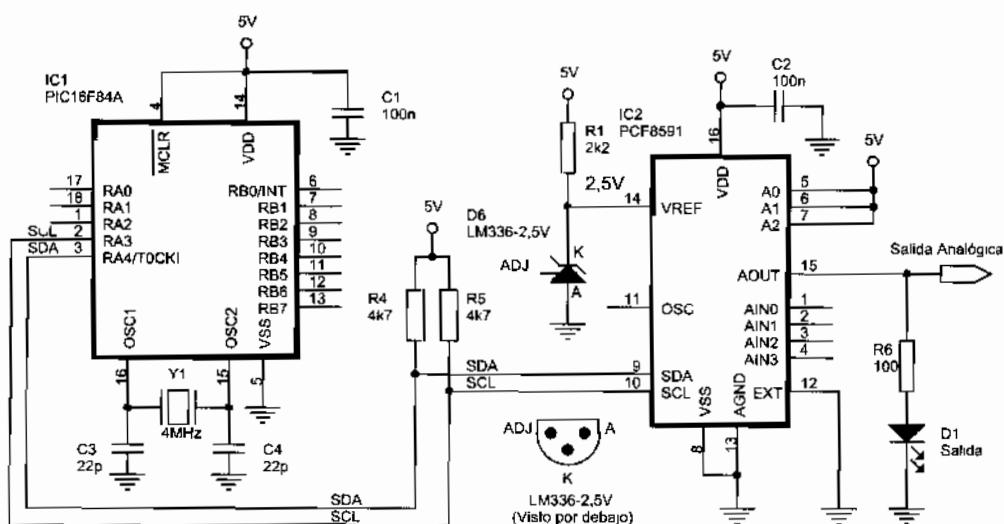


Figura 27-8 Tensión de referencia de 2,5 V mediante LM336-2.5V

## 27.6 EJEMPLOS DEL PCF8591 COMO DAC

Como aplicación de lo explicado vemos el siguiente programa que es un ejemplo del manejo del PCF8591 como DAC para el circuito de la figura 27-6 ó 27-8 aplicable a muchos proyectos.

```
;*****12C_DAC_01.asm*****
;
; En la salida analógica del PCF8591 está conectado un diodo LED en serie con una resistencia
; de unos 100 ohmios. Desde la posición de apagado, el LED comienza a encenderse
; paulatinamente hasta llegar a su luminosidad máxima, alcanzada la cual volverá a apagarse
; repitiendo el ciclo.
;
; Para ello, la tensión en la salida comienza en 1500 mV (1,5 V), que es el nivel de umbral de
; luminosidad del LED, asciende en pasos de 10 mV hasta alcanzar 2500 mV (2,5 V) para volver
; a caer a 1,5 V y repetir el ciclo.
;
; En cada paso estará unos 50 ms. El total de cada ciclo durará pues unos 5 segundos ya que
; de 1500 a 2500 hay 100 pasos de 10 mV, multiplicado por los 50 ms que dura cada paso
; resulta 5000 ms en total.
;
; El DAC va a utilizar un contador auxiliar que contiene el valor digital a convertir. Se
; incrementa cada vez que ejecuta la subrutina principal, desde 150 hasta 250 y repite el ciclo.
;
; Como el PCF8591 del esquema trabaja con una resolución de LSB=10mV, el valor del contador
; será 10 veces menor que la tensión analógica deseada a la salida expresada en milivoltios.
; Así por ejemplo, si (Contador)=147 el valor de la tensión de salida será igual a:
; VOUT = LSB x Digital = 10 x 147 = 1470 mV = 1,47 V.
;
```

; Por tanto, como (Contador) varía desde 1500 mV (1,5 V) a 2500 mV (2,5 V) con un aumento paulatino de 10 mV, la salida analógica varía entre 1,5 V y 2,5 V.

**ZONA DE DATOS \*\***

**CONFIG**  
**LIST**  
**INCLUDE** <

**CBLOCK** 0  
**ENDC**

**PCF8591\_DireccionEscritura**  
**ZONA DE CÓDIGOS**

<b>ORG</b>	0
<b>Inicio</b>	call Call
	call Call
	movlw PC
	call Call
	movlw b'0 I2C
	call Call
<b>Principal</b>	I2C
	call Call
	call Call
	call Call
	goto Pri

; Subrutina "IncrementaContador"

**CBLOCK**  
**Contador**  
**ENDC**

**ValorMinimo**  
**ValorMaximo**  
**SaltoIncremento**

<b>IncrementaContador</b>	Sal
	movlw Cont
	addwf ST
	btfsc Car
	goto Con
	movf Con
	sublw Va
	btfsc ST
	goto Fin
<b>CargaInicialContador</b>	Car
	movlw Va
	movwf Co

; Por tanto, como (Contador) varía de 150 a 250 en pasos de 1, la tensión analógica de salida varía desde 1500 mV (1,50 V) hasta 2500 mV (2,50 V) en pasos de 10 mV (0,01 V), produciendo un aumento paulatino de la tensión en la salida analógica y de la luminosidad del LED.

; ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <PI6F84A.INC>
```

```
CBLOCK 0x0C
ENDC
```

PCF8591\_DireccionEscritura EQU b'10011110'

; ZONA DE CÓDIGOS \*\*\*\*\*

ORG 0

Inicio

```
call CargaInicialContador
call I2C_EnviaStart ; Envía condición de Start.
movlw PCF8591_DireccionEscritura ; Apunta al dispositivo.
call I2C_EnviaByte
movlw b'01000000' ; Carga la palabra de control activando la
call I2C_EnviaByte ; salida analógica.
```

Principal

```
call IncrementaContador
call I2C_EnviaByte ; Convierte el dato digital de entrada a tensión
call Retardo_50ms ; analógica presente en el pin AOUT.
goto Principal
```

; Subrutina "IncrementaContador"

```
CBLOCK
Contador
ENDC
```

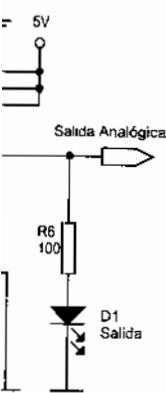
ValorMinimo EQU .150
ValorMaximo EQU .250
SaltoIncremento EQU .1

IncrementaContador

```
movlw SaltoIncremento ; Incrementa el valor deseado.
addwf Contador,F ; Si se desborda realiza la carga inicial.
btfsr STATUS,C ; ¿Ha llegado a su valor máximo?
goto CargaInicialContador ; (W) = ValorMaximo - (Contador)
movf Contador,W ; ¿C=0?, ¿(W) negativo?, ¿ValorMaximo<(Contador)?
sublw ValorMaximo ; No, resulta ValorMaximo>=(Contador) y sale.
btfsr STATUS,C
goto FinIncrementar
```

CargaInicialContador

```
movlw ValorMinimo ; Sí, entonces inicializa el registro.
movwf Contador
```



2.5V

es un ejemplo  
27-8 aplicable a

\*\*\*\*\*

ncia

e

de  
olver

o.

atador

```

FinIncrementar
    movf Contador,W           ; El resultado en (W).
    retum

INCLUDE <BUS_I2C.INC>
INCLUDE <RETARDOS.INC>
END

```

El circuito de la figura 27-9 es otro ejemplo de aplicación. Se trata de un generador de tensión de referencia ajustable mediante pulsador. Su programa de control se detalla a continuación.

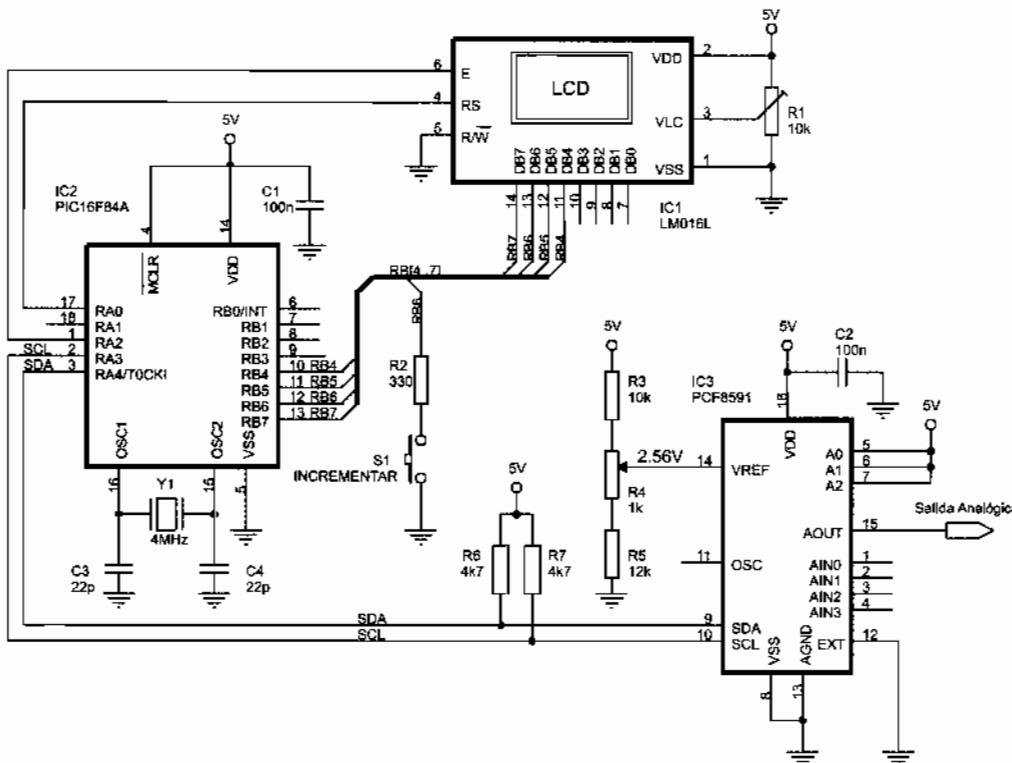


Figura 27-9 Otro ejemplo de configuración del PCF8591 como DAC

```
***** I2C_DAC_02.asm *****
```

; En la salida analógica del PCF8591 que trabaja como DAC se obtiene una tensión seleccionada  
; por un pulsador conectado a la línea RB6 del PIC. La tensión varía entre 0,50 y 2,50 V en  
; saltos de 0,25 V y se visualiza en el módulo LCD.

```
***** ZONA DE DATOS *****
```

```

_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST      P=16F84A

```

```

INCLUDE
CBLOCK
ENDC

PCF8591_Direccion
#define Incrementar
; ZONA DE CÓDIGOS
ORG
goto
ORG
goto

Mensajes
addwf
MensajeTension
DT "Tensión"
MensajeVoltois
DT "V."
Inicio
call
bsf
bsf
bcf
bcf
call
call
call
movlw
movwf

Principal
sleep
goto

; Subrutina "Servicio"
; Incrementa el registro
; Como el PCF8591
; será 10 veces menor
; Así por ejemplo, si
; VOUT = LSB x Di
; ServicioInterrupcion
call
btfs
goto
IncrementarTension
call
call
call

```

```

INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

PCF8591_DireccionEscritura EQU b'10011110'

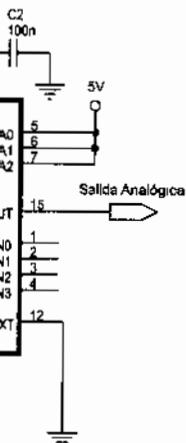
#define IncrementarPulsador PORTB,6
;
; ZONA DE CÓDIGOS ****
;

ORG 0
goto Inicio
ORG 4
goto ServicioInterrupcion

Mensajes
    addwf PCL,F
MensajeTension
    DT "Tension: ", 0x00
MensajeVoltios
    DT " V. ", 0x00
Inicio
    call LCD_Inicializa
    bcf STATUS.RP0
    bcf IncrementarPulsador; Se configura como entrada.
    bcf STATUS.RP0
    call CargaInicialContador
    call PCF8591_DAC
    call Visualiza
    movlw b'10001000'
    movwf INTCON
;
Principal
    sleep ; Pasa a modo de reposo.
    goto Principal

; Subrutina "ServicioInterrupcion"
;
; Incrementa el registro Contador cada vez que se presiona el pulsador "INCREMENTAR".
;
; Como el PCF8591 del esquema trabaja con una resolución de LSB=10mV el valor del (Contador)
; será 10 veces menor que la tensión analógica deseada a la salida expresada en milivoltios.
; Así por ejemplo, si (Contador)=147 el valor de la tensión de salida será igual a:
; VOUT = LSB x Digital = 10 x 147 = 1470 mV = 1,47 V.
;
ServicioInterrupcion
    call Retardo_20ms ; Espera a que se establezcan los niveles de tensión.
    btfsc IncrementarPulsador; Si es un rebote sale fuera.
    goto FinInterrupcion
IncrementarTensionDeseada
    call IncrementaContador ; Aumenta el valor del contador.
    call PCF8591_DAC ; Lo envía al DAC para su conversión.
    call Visualiza ; Visualiza mientras espera que deje
;
```

a de un generador  
control se detalla a



no DAC

cionada  
n

\*\*\*\*\*

```

call    Retardo_100ms      ; de pulsar durante este tiempo.
btfsf  IncrementarPulsador; Mientras permanezca pulsado
goto   IncrementarTensionDeseada ; incrementará el digito.
FindInterrupcion
bcf    INTCON,RBIF
retfie

```

```

; Subrutina "PCF8591_DAC"
;
; Escribe en el PCF8591 con el dato del registro W para su conversión a tensión analógica.

```

```

CBLOCK
PCF8591_Dato           ; Guarda el dato que tiene que enviar.
ENDC

```

```

PCF8591_DAC
movwf  PCF8591_Dato      ; Guarda el dato a enviar.
call    I2C_EnviaStart    ; Envía condición de Start.
movlw  PCF8591_DireccionEscritura ; Apunta al dispositivo.
call    I2C_EnviaByte
movlw  b'01000000'         ; Carga la palabra de control activando la
call    I2C_EnviaByte      ; salida analógica.
movf   PCF8591_Dato,W     ; Escribe el dato dos veces para que la
call    I2C_EnviaByte      ; conversión sea correcta tal como se indica en
movf   PCF8591_Dato,W     ; los cronogramas del fabricante.
call    I2C_EnviaByte
call    I2C_EnviaStop      ; Termina.
movf   PCF8591_Dato,W     ; En (W) se recupera de nuevo el dato de entrada.
return

```

```

; Subrutinas "IncrementaContador" y "CargaInicialContador"
;
```

```

CBLOCK
Contador
ENDC

```

```

ValorMinimo  EQU    d'50'        ; El valor mínimo de tensión será 0,5 V.
ValorMaximo  EQU    d'250'       ; El valor máximo de tensión será 2,5 V.
SaltoIncremento EQU    d'25'       ; El incremento se producirá en saltos de 0,25 V.
;
```

#### IncrementaContador

```

movlw  SaltoIncremento      ; Incrementa el valor deseado.
addwf  Contador,F          ; Si se desborda realiza la carga inicial.
btfsf  STATUS,C            ; ¿Ha llegado a su valor máximo?
                            ; (W) = ValorMaximo - (Contador).
                            ; ¿C=0?, ¿(W) negativo?, ¿ValorMaximo<(Contador)?
                            ; No, resulta ValorMaximo>=(Contador) y sale.
                            ; FinIncrementar

```

#### CargaInicialContador

```

movlw  ValorMinimo          ; Si, entonces inicializa el registro.
movwf  Contador

```

#### FinIncrementar

```

movf  C
return

```

; Subrutinas "Visualiza"

```

; Visualiza el valor que
; Hay que tener en cuenta
; el valor de entrada se
; Así por ejemplo, si (W)
; = 10 x 147 = 1470 mV
;
; En conclusión:
; - Las centenas del valor
; - Las decenas del valor
; - Las unidades del valor

```

```

CBLOCK
Auxiliar
ENDC

```

#### Visualiza

```

movwf
call
movlw
call
movf
call
movf
call
movlw
call
movf
call
movf
call
movlw
call
movf
call
movf
call
movlw
call
return
;
```

```

INCLUDE
INCLUDE
INCLUDE
INCLUDE
INCLUDE
INCLUDE
END

```

Este circuito  
escalas habría que  
lector a desarrollar

```
    movf Contador,W           ; En (W) el resultado.
    return
```

: Subrutinas "Visualiza"

; Visualiza el valor que se le introduce por el registro de trabajo W en formato de tensión.  
; Hay que tener en cuenta que el PCF8591 del esquema trabaja con una resolución de LSB=10mV,  
; el valor de entrada será 10 veces menor que la tensión real expresada en milivoltios.  
; Así por ejemplo, si (W)=147 el valor de la tensión será igual a: VOUT = LSB x Digital =  
;  $\approx 10 \times 147 = 1470 \text{ mV} = 1,47 \text{ V}$ , que es lo que se debe visualizar en la pantalla.

; En conclusión:

- Las centenas del valor de entrada corresponden a las unidades de voltio.
- Las decenas del valor de entrada corresponden a las décimas de voltio.
- Las unidades del valor de entrada corresponden a las centésimas de voltios.

```
CBLOCK
Auxiliar
ENDC
```

Visualiza

movwf Auxiliar	; Lo guarda.
call LCD_Linea1	; Se sitúa al principio de la primera línea.
movlw MensajeTension	; Visualiza la tensión deseada.
call LCD_Mensaje	
movf Auxiliar,W	; Recupera el dato a visualizar y lo
call BIN_a_BCD	; pasa a BCD.
movf BCD_Centenas,W	; Visualiza las centenas que corresponden a las
call LCD_Nibble	; unidades de voltios.
movlw ''	; Visualiza el punto decimal.
call LCD_Caracter	
movf BCD_Decenas,W	; Visualiza las decenas que corresponden a las
call LCD_Nibble	; décimas de voltios.
movf BCD_Unidades,W	; Visualiza las unidades que corresponden a las
call LCD_Nibble	; centésimas de voltios.
movlw MensajeVoltios	
call LCD_Mensaje	
return	

```
INCLUDE <BUS_I2C.INC>
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END
```

Este circuito proporciona tensiones entre 0 y 2,5 V. Para conseguir otros valores de escalas habría que añadirle a la salida circuitos atenuadores y amplificadores. Se anima al lector a desarrollarlo como proyecto.

## 27.7 EL PCF8591 COMO ADC

En la figura 27-10 se muestra un PCF8591 en configuración típica trabajando como convertidor Analógico - Digital ADC (*Analog to Digital Converter*). A destacar:

- La entrada analógica se aplica al pin AIN0. El resto de las entradas analógicas no se utiliza. La salida digital convertida se lee a través del bus I2C.
  - La tensión aplicada al pin VREF es de 2,56 V para conseguir una resolución de 10 mV. También se podría haber utilizado un LM336-2,5V.
  - Un generador interno proporciona la señal de reloj necesaria para el ciclo de conversión ADC, siempre que el pin EXT esté conectado a masa. En la patilla OSC se dispone de la frecuencia de oscilación.
  - El divisor de tensión R7/R8 permite obtener una tensión variable para probar el circuito.

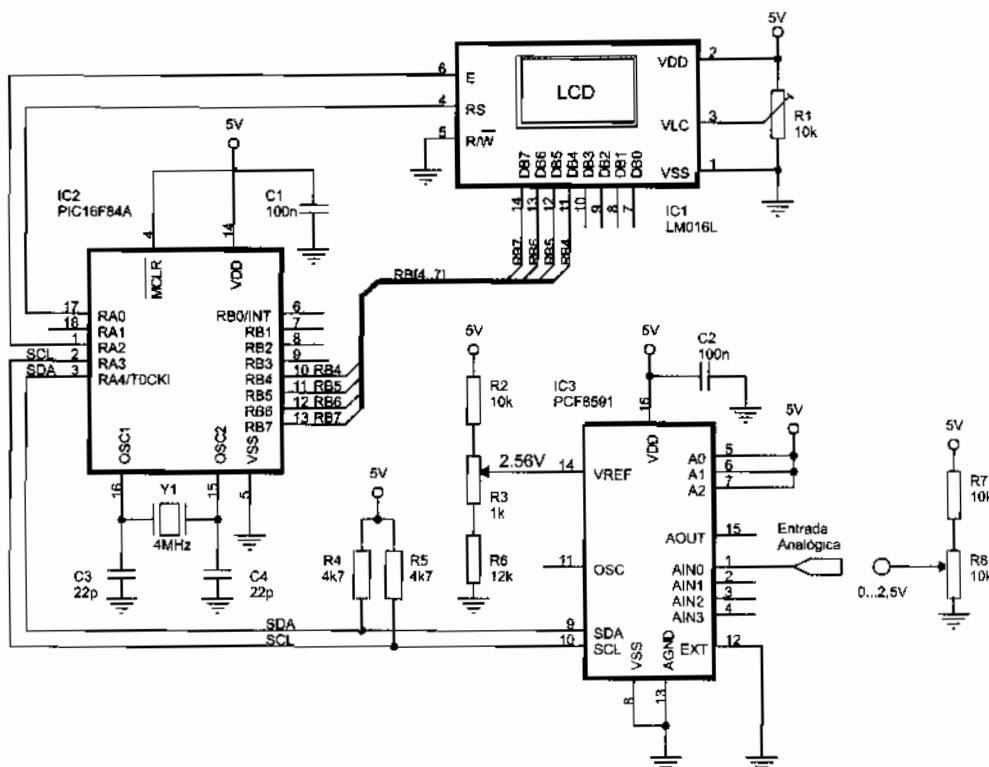


Figura 27-10 Ejemplo de configuración del PCF8591 como ADC

Antes de leer el ADC es necesario configurarlo enviando la palabra de control tal como se ha explicado anteriormente.

Cuando trabaja como ADC, el ciclo de conversión se inicia siempre tras el envío de una dirección de lectura válida y se inicia con el flanco descendente del pulso de

reconocimiento  
conversión anterior

Una vez ir canal seleccionado resultante se guarda activando el flag de

El primer  
anterior ciclo de  
normalmente será

La máxima  
trabajo del bus (2)

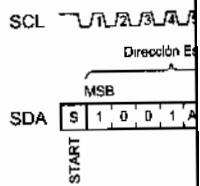


Figura 27-

Para el caso de la ecuación:

Siendo:

- **V<sub>AIN</sub>**, el que quiere conectar
  - **LSB**, la de los PCF conectados

## EJEMPL

ADC de la figura.

reconocimiento y, finalmente, se ejecuta mientras se transmite el resultado de la conversión anterior tal como se muestra en la figura 27-11.

Una vez iniciado el ciclo de conversión una muestra de la tensión de entrada del canal seleccionado se convierte al correspondiente código binario de 8 bits. La conversión resultante se guarda en el registro de datos del ADC en espera de ser transmitida. Si está activado el flag de autoincremento será seleccionado el siguiente canal.

El primer byte transmitido en un ciclo de lectura contiene el código resultante del anterior ciclo de conversión. Después de un reset, el primer byte leído será el 80h y normalmente será desecharado.

La máxima velocidad de conversión del ADC viene dada por la velocidad de trabajo del bus I2C.

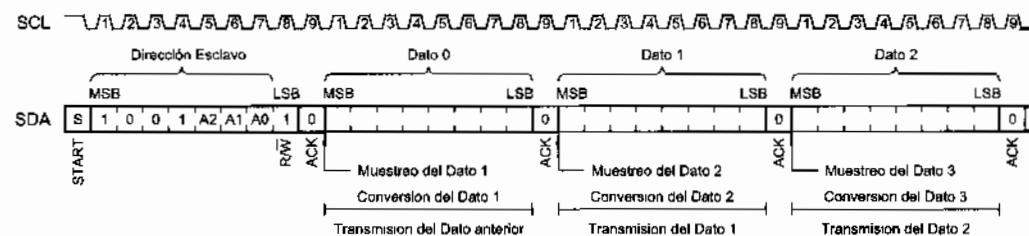


Figura 27-11 Secuencia de conversión cuando el PCF8591 actúa como ADC

Para el caso de entradas simples el valor digital de la salida vendrá dado por la ecuación:

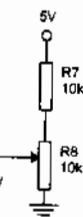
$$Digital = \frac{V_{AIN}}{LSB}$$

Siendo:

- $V_{AIN}$ , el valor de la tensión analógica aplicada a una de las entradas AIN que se quiere convertir.
- $LSB$ , la resolución del DAC cuyo valor viene dado por la tensión en el pin VREF del PCF8591 según la siguiente ecuación, suponiendo que el pin AGND está conectado a masa:

$$LSB = \frac{V_{REF}}{256}$$

**EJEMPLO:** Deducir el valor digital de salida del PCF8591 configurado como ADC de la figura 27-10, donde la tensión aplicada al pin VREF es de 2,56 V, el pin AGND está conectado a masa y la tensión de entrada aplicada al pin AIN0 es de 2,17 V.



control tal  
el envío de  
pulso de

**Solución:** La resolución LSB del circuito es:

$$LSB = \frac{V_{REF}}{256} = \frac{2,56}{256} = 0,01V = 10mV$$

El dato de entrada es:  $V_{AIN} = 2,17 V = 2170 mV$ .

Por tanto el valor del dato digital de salida ofrecido por el PCF8591 al microcontrolador será:

$$Digital = \frac{V_{AIN}}{LSB} = \frac{2170}{10} = 217_{10} = 11011001_2$$

La exactitud en el proceso de conversión está determinada por el valor de la resolución del circuito (LSB), por lo que es muy importante que esta tensión de referencia se mantenga muy estable.

## 27.8 EJEMPLO DEL PCF8591 COMO ADC

Una de las aplicaciones más interesantes en proyectos con ADC es la realización de un voltímetro. El siguiente programa es un ejemplo para el circuito de la figura 27-10.

```
***** I2C_ADC_01.asm *****
;
; El microcontrolador lee constantemente la entrada analógica AIN0 del PCF8591 y
; visualiza la tensión en la pantalla del módulo LCD.
;
; ZONA DE DATOS *****
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
ENDC

PCF8591_DireccionEscritura EQU b'10011110'
PCF8591_DireccionLectura EQU b'10011111'

;
; ZONA DE CÓDIGOS *****

ORG 0
Inicio
    call LCD_Inicializa
    call I2C_EnviaStart ; Va a configurar el PCF8591.
    movlw PCF8591_DireccionEscritura ; Apunta al dispositivo.
    call I2C_EnviaByte
    movlw b'00000000' ; Carga la palabra de control utilizando la
    call I2C_EnviaByte ; entrada AIN0 en modo simple.
    call I2C_EnviaStop ; Termina la configuración
;
    call I2C_EnviaStart ; Comienza a leer.
```

Principal	movlw
	call
	call
	call
	call
	goto

;	Subrutinas "Visualiz"
;	;
;	Visualiza el valor q
;	Hay que tener en cu
;	LSB=10mV, el val
;	milivoltios. Así por
;	VAIN = LSB x Dig
;	en la pantalla.
;	;
;	En conclusión:
;	- Las centenas del v
;	- Las decenas del v
;	- Las unidades del v

CBLOCK
Auxiliar
ENDC

Visualiza	movwf
	call
	movlw
	call
	movf
	call
	movw
	call
	movlw
	call
	movf
	call
	movwf
	call
	return

Mensajes	addwf
MensajeTension	DT "Tens
MensajeVoltios	DT " V.

INCLUDI
---------

```

movlw  PCF8591_DireccionLectura ; Apunta al dispositivo.
call   I2C_EnviaByte
call   I2C_LeeByte               ; La primera lectura es incorrecta y por lo tanto
Principal                   ; la desecha.
call   I2C_LeeByte               ; Lee la entrada analógica.
call   Visualiza                ; La visualiza.
goto  Principal

```

; Subrutinas "Visualiza"

; Visualiza el valor que se le introduce por el registro de trabajo W en formato de tensión.  
; Hay que tener en cuenta que el PCF8591 del esquema trabaja con una resolución de  
; LSB=10mV, el valor de entrada será 10 veces menor que la tensión real expresada en  
; milivoltios. Así por ejemplo, si (W)=147 el valor de la tensión será igual a:  
; VAIN = LSB x Digital = 10 x 147 = 1470 mV = 1,47 V, que es lo que se debe visualizar  
; en la pantalla.

; En conclusión:

; - Las centenas del valor de entrada corresponden a las unidades de voltio.  
; - Las decenas del valor de entrada corresponden a las décimas de voltio.  
; - Las unidades del valor de entrada corresponden a las centésimas de voltios.

#### CBLOCK

Auxiliar

ENDC

#### Visualiza

movwf Auxiliar	;	Lo guarda.
call LCD_Linea1	;	Se sitúa al principio de la primera línea.
movlw MensajeTension	;	Visualiza la tensión deseada.
call LCD_Mensaje		
movf Auxiliar,W	;	Recupera el dato a visualizar y lo
call BIN_a_BCD	;	pasa a BCD.
movf BCD_Centenas,W	;	Visualiza las centenas que corresponden a las
call LCD_Nibble	;	unidades de voltios.
movlw ','	;	Visualiza el punto decimal.
call LCD_Caracter	;	Visualiza las decenas que corresponden a las
movf BCD_Decenas,W	;	décimas de voltios.
call LCD_Nibble	;	Visualiza las unidades que corresponden a las
movf BCD_Unidades,W	;	centésimas de voltios.
call LCD_Nibble		
movlw MensajeVoltios		
call LCD_Mensaje		
return		

#### Mensajes

addwf PCL,F

MensajeTension

DT "Tension: ", 0x00

MensajeVoltios

DT " V. ", 0x00

INCLUDE <BUS\_I2C.INC>

```

INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

Para probar este circuito se puede utilizar la tensión variable que se obtiene del divisor de tensión formado por R7 y R8.

Este voltímetro puede medir una tensión entre 0 y 2,5 V. Para conseguir otros valores de escalas habría que añadirle a la entrada circuitos atenuadores y amplificadores. Se anima al lector a desarrollarlo como proyecto.

El fabricante fabrica una familia de dispositivos que utilizan una linea de señal de alta velocidad y una gran variedad de aplicaciones. Una de es el termómetro DS18B20.



## 28.1 SEN

El DS18B20 es un sensor de temperatura leí

06/06/2013

e obtiene del

nseguir otros  
mplificadores.

## CAPÍTULO 28

# BUS DE UNA LÍNEA

El fabricante americano *Dallas Semiconductors* ([www.dalsemi.com](http://www.dalsemi.com)) ha diseñado una familia de periféricos muy atractiva para su uso con microcontroladores porque sólo utiliza una línea para transferir datos denominada *1-Wire Bus* o **bus de una línea**. Hay una gran variedad de dispositivos compatibles con este protocolo, uno de más populares es el termómetro digital DS1820 (figura 18-1).

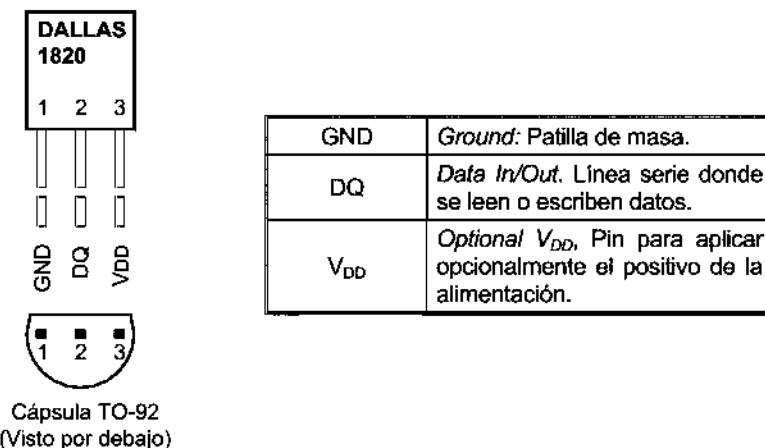


Figura 28-1 Patillaje del DS1820 para cápsula TO-92

### 28.1 SENSOR DE TEMPERATURA DS1820

El DS1820 es un sensor de temperatura que transmite el valor digital de la temperatura leída en el lugar donde se encuentre el dispositivo. Utiliza una única línea

serie que se comunica con otros dispositivos mediante el protocolo para bus de una línea diseñado por *Dallas Semiconductors*. Sus principales características son:

- El patillaje para cápsula TO-92 que se representa en la figura 28-1.
- La temperatura es leída como un valor digital de 2 bytes que incluye el signo.
- Medida de temperatura desde  $-55^{\circ}\text{C}$  hasta  $+125^{\circ}\text{C}$ .
- Requiere una única línea para la comunicación mediante el *1-Wire Bus*.
- Apenas requiere componentes externos.

Dispone de dos modelos que sólo se diferencian por el tiempo de conversión de la temperatura: 200 ms para el DS1820 y 750 ms para el DS18S20.

## 28.2 DIAGRAMA EN BLOQUES DEL DS1820

La figura 28-2 muestra el diagrama en bloque del DS1820 en el que se distinguen los siguientes componentes:

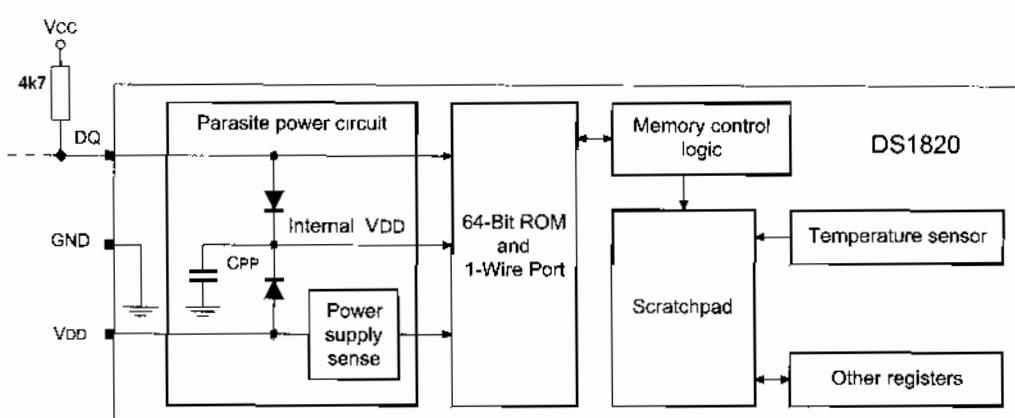


Figura 28-2 Diagrama de bloques del DS1820

- **64-Bit ROM.** Zona de memoria ROM donde se graba un código identificativo de 64 bits y que es único para cada chip. Cada dispositivo tiene un código asociado grabado en ROM que sirve para identificarlo en el caso de que varios de ellos se encuentren conectados a la misma línea de datos. Esta especial característica hace que no sea necesario conectar dispositivos adicionales ni aumentar el cableado para la selección de uno de estos dispositivos entre varios.
- **Parasite Power Circuit.** Una característica importante de estos dispositivos es la posibilidad que tienen de alimentarse sin necesidad de una fuente de alimentación externa. Para ello, aprovecha la potencia procedente de la línea DQ cuando se encuentra a nivel alto. Esta energía es almacenada en el

condensador se encuentra parásito el pi

- **1-Wire Port** patentado por diferentes dispositivos correcto funciona aprecia en la t
- **Scratchpad** M bytes con la continuación.

## 28.3 LECTURA

El DS1820 propone ejemplos de la tabla 28-

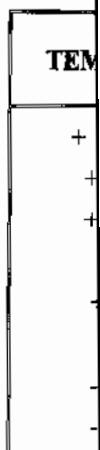


Tabla 28-1 Relación entre temperatura y datos

Se observa que:

- El formato es de
- El primer byte es
  - Las temperaturas "00000000"
  - Las temperaturas "11111111"
- El bit de menor significado para temp

de una línea  
el signo.  
as.

versión de la  
se distinguen

S1820  
ture sensor  
registers

identificativo  
e un código  
que varios  
sta especial  
cionales ni  
itivos entre

positivos es  
fuente de  
de la línea  
nada en el

condensador Cpp, que actúa como alimentación del chip cuando la línea DQ se encuentra a nivel bajo. Cuando el DS1820 utiliza el modo de alimentación parásito el pin 3 ( $V_{DD}$ ) debe conectarse a masa.

- *I-Wire Port (DQ)*. El DS1820 utiliza para comunicarse el bus de 1 línea patentado por *Dallas Semiconductors*, que permite la comunicación entre diferentes dispositivos utilizando una única línea. Esta línea necesita para su correcto funcionamiento una resistencia de Pull-Up de unos 4k7 como se aprecia en la figura 28-2.
- *Scratchpad Memory*. Esta memoria auxiliar contiene principalmente los dos bytes con la temperatura medida (*Temperature Sensor*) que se describen a continuación.

### 28.3 LECTURA DE LA TEMPERATURA

El DS1820 proporciona la lectura de temperatura según el formato mostrado en los ejemplos de la tabla 28-1.

TEMPERATURA	CÓDIGO DE SALIDA PARA EL DS1820
+ 125,0 °C	00000000 11111010
+ 25,5 °C	00000000 00110011
+ 25,0 °C	00000000 00110010
+ 0,5 °C	00000000 00000001
+ 0,0 °C	00000000 00000000
- 0,5 °C	11111111 11111111
- 25,0 °C	11111111 11001110
- 55,0 °C	11111111 10010010

Tabla 28-1 Relación entre la temperatura y el contenido de los registros para el DS1820

Se observa que:

- El formato es de dos bytes.
- El primer byte es de signo:
  - Las temperaturas positivas comienzan con el byte alto todo a ceros: "00000000 xxxxxxxx".
  - Las temperaturas negativas comienzan con el byte alto todo a unos: "11111111 xxxxxxxx".
- El bit de menor peso determina el decimal de la temperatura leída:
  - Para temperaturas "xx.0 °C", termina en cero: "xxxxxxxx0".

- Para temperaturas "xx.5 °C", termina en uno: "xxxxxx1".
- El valor de la temperatura está especificado en los 7 bits de mayor peso del byte bajo. Por ejemplo: +25°C = xxxxxxxx 0011001x = 25 en decimal. Por tanto, para obtener el valor de la temperatura habrá que desplazar un lugar hacia la derecha los bits del byte bajo.
- Las temperaturas vienen dadas en complemento a 2. Hay que negar las temperaturas negativas para obtener su valor absoluto.

## 28.4 BUS DE UNA LÍNEA

El bus de 1 línea (*1-Wire Bus System*) es un bus ideado por *Dallas Semiconductors* para transmitir información a través de una sola línea. Se basa en la And cableada. Normalmente se conecta:

- Un maestro que suele ser un microcontrolador que lleva el control del sistema.
- Uno o más esclavos que proporcionan información y son gobernados por el maestro. El DS1820 siempre es un esclavo.

El bus de una linea tiene, por definición, una simple línea de datos representada en la figura 28-3. Cada dispositivo maestro o esclavo se conecta al bus a través de una línea en drenador abierto. Esto permite a cada dispositivo conectado "ceder" el bus cuando no está transmitiendo datos y posibilita que el bus sea usado por otros dispositivos.

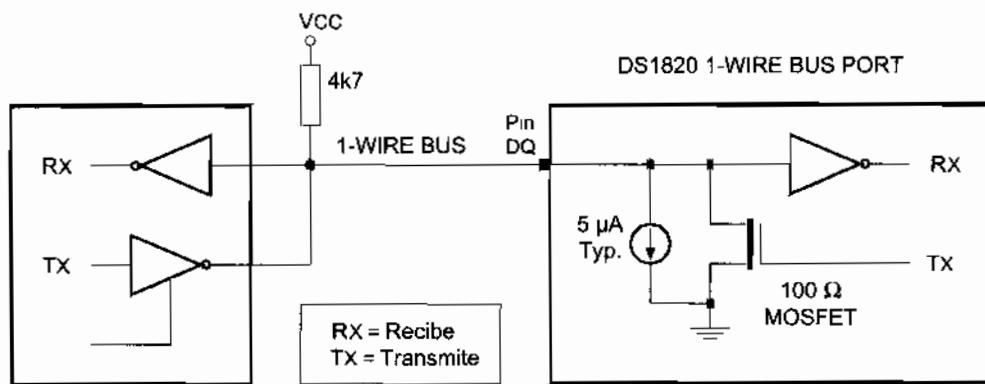


Figura 28-3 Configuración hardware del 1-Wire Bus

El bus de una línea requiere de una resistencia externa de Pull-Up de unos 4k7, de esta forma el estado de reposo del bus será el nivel alto.

Si el bus se mantiene a nivel bajo durante más de 480 µs, se produce un reset de todos los dispositivos esclavos conectados a él.

## 28.5 SEN

Los dispositivos de comunicación con Dallas Semiconductor

- Pulso de alta
- Pulso de baja
- Escritura
- Escritura
- Lectura
- Lectura

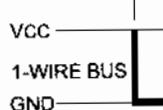
Todas estas señales se manejan en el microcontrolador.

Entre cada dispositivo conectado al bus, cual el bus debe tener una resistencia de Pull-Up de 4k7.

Todos los dispositivos tienen un bit de Significante Bit) en el bus.

## 28.6 INICIACIÓN

En el bus de una línea se define una secuencia de inicio que consiste en una pausa larga seguida de un pulso de presencia (Presence pulse).



## 28.5 SEÑALES DEL BUS DE UNA LÍNEA

Los dispositivos compatibles con *I-Wire Bus* utilizan un estricto protocolo de comunicación con el fin de asegurar la integridad de los datos. El protocolo ideado por *Dallas Semiconductors* define varias señales tipos:

- Pulso de *Reset*.
- Pulso de *Presence*.
- Escritura de nivel lógico “0”.
- Escritura de nivel lógico “1”.
- Lectura de nivel lógico “0”.
- Lectura de nivel lógico “1”.

Todas estas señales, con excepción del *Presence*, son inicializadas por el microcontrolador maestro mediante un flanco de bajada y un nivel bajo de al menos 1  $\mu$ s.

Entre cada señal diferente debe transcurrir un tiempo mínimo de 1  $\mu$ s durante el cual el bus debe mantenerse en alta impedancia, lo que supone un nivel alto debido a la resistencia de Pull-Up.

Todos los datos y comandos son transmitidos con el bit de menor peso LSB (*Least Significant Bit*) en primer lugar.

## 28.6 INICIALIZACIÓN: PULSOS RESET Y PRESENCE

En el bus de 1 línea toda comunicación comienza con una secuencia de iniciación que consiste en un pulso de *Reset* ejecutado por el microcontrolador, seguido por un pulso de *Presence* ejecutado por el dispositivo esclavo DS1820 (figura 28-4).

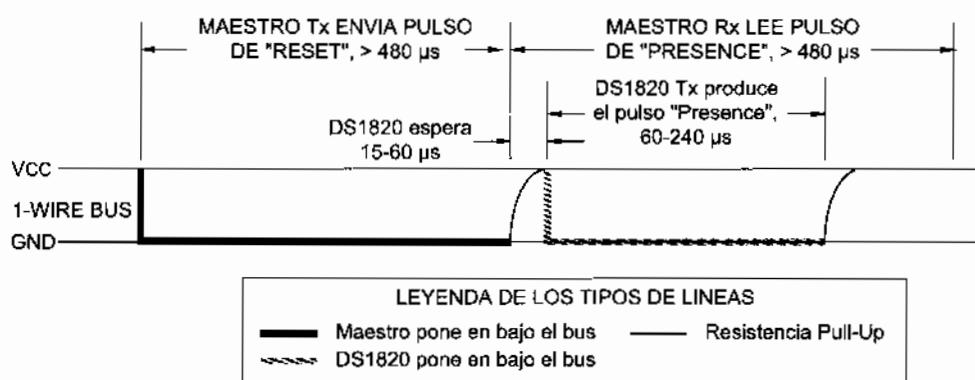


Figura 28-4 Pulsos de Reset y Presence

Cuando el DS1820 envía el pulso de *Presence* en respuesta al de *Reset* le indica al microcontrolador que está preparado para trabajar.

Durante la secuencia de iniciación, se produce el siguiente protocolo:

- 1º El microcontrolador maestro transmite un pulso de *Reset* que mantiene en bajo la línea durante al menos 480  $\mu$ s.
- 2º A continuación el maestro pasa a modo entrada y, por tanto, pone el bus a nivel alto gracias a la resistencia de Pull-Up.
- 3º Cuando el DS1820 detecta esto, espera de 15 a 60  $\mu$ s y entonces transmite el pulso *Presence*, que consiste en poner la línea a nivel bajo durante un tiempo de entre 60 a 240  $\mu$ s.
- 4º A continuación, el DS1820 deja el bus en alta impedancia y espera comandos u órdenes del microcontrolador.

## 28.7 ESCRITURA DE UN BIT SOBRE EL DS1820

Para escribir un bit sobre el DS1820, el microcontrolador maestro debe seguir el protocolo explicado en la figura 28-5, que consta de los siguientes pasos:

- 1º El maestro manda un pulso en bajo durante un mínimo de 1  $\mu$ s
- 2º El maestro manda un “0” o un “1”, según corresponda, durante un tiempo de entre 15 y 60  $\mu$ s.

Del análisis de la figura, caben destacar los siguientes puntos:

- La escritura de “0” ó “1” viene precedida siempre de un flanco de bajada y de un nivel bajo, entre 1 y 15  $\mu$ s, ejecutado por el microcontrolador maestro.
- El bit realmente se graba en el DS1820, pasados 15  $\mu$ s desde el flanco de bajada.
- El máximo tiempo que puede durar la escritura de un bit es de 60  $\mu$ s.

Entre cada escritura de bit debe transcurrir un tiempo mínimo de 1  $\mu$ s en el que el bus se debe mantener en alta impedancia, lo cual supone un nivel alto debido a la resistencia de Pull-Up.

## 28.8 LECTURA DE UN BIT PROCEDENTE DEL DS1820

Para leer un bit procedente del DS1820 el microcontrolador maestro debe seguir el protocolo de la figura 28-5, que consta de los siguientes pasos:

- 1º El maestro manda un pulso en bajo durante un mínimo de 1  $\mu$ s.
- 2º El maestro lee el bit procedente del DS1820 pasados 15  $\mu$ s desde el flanco de bajada.

Del análisis

- La lectura
- nivel bajo
- El bit real
- El máxi

Entre cada

el bus se debe m

resistencia de Pull

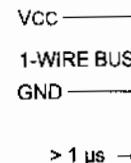
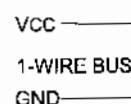


Figura 28-5 Diagrama de escritura de un bit.

## 28.9 LIBREASIS

Para una mejor comprensión pasamos a describir el protocolo para cualquier dispositivo de memoria fabricado por *Semiconductors*. Los

- "Bus1Linea\_Reset". Envía un impulso de *Reset* al dispositivo conectado en el bus de 1 línea y después espera recibir un impulso de *Presence*.
- "Bus1Linea\_EscribeByte". El microcontrolador maestro escribe un byte al esclavo conectado al bus de 1 línea, comenzando por el bit LSB.
- "Bus1Linea\_LeeByte". El microcontrolador maestro lee un byte del esclavo conectado al bus de 1 línea.

```
;***** Librería "BUS_1LIN.INC" *****
;
; Estas subrutinas permiten realizar las tareas básicas de control del bus de una soia linea
; según el protocolo del fabricante de semiconductores Dallas Semiconductors.
;
; ZONA DE DATOS *****

CBLOCK
Bus1Linea_Dato ; Guarda dato a transmitir o recibir por el bus.
Bus1Linea_ContadorBits ; Cuenta los bits a transmitir o a recibir.
ENDC

#DEFINE Bus1Linea PORTB,3 ; Línea donde se conecta el DS1820.

; Subrutina "Bus1Linea_AltalImpedancia"
;
; Configura la línea única de 1-Wire Bus como entrada. Lo pone en alta impedancia.

Bus1Linea_AltalImpedancia
bsf STATUS,RP0
bsf Bus1Linea ; La línea como entrada, por tanto, en alta
bcf STATUS,RP0 ; impedancia.
return

; Subrutina "Bus1Linea_Bajo"
;
; Mantiene el pin de datos del Bus de 1 línea en bajo.

Bus1Linea_Bajo
bsf STATUS,RP0
bcf Bus1Linea ; La línea como salida.
bcf STATUS,RP0
bcf Bus1Linea ; Pasa a bajo.
return

; Subrutina "Bus1Linea_Reset"
;
; Envía un impulso de Reset al dispositivo conectado en el bus de 1 línea y después
; espera recibir un impulso de Presence". Esto consiste en:
; 1º. El PIC manda un impulso en bajo durante unos 500 µs. (El protocolo permite entre 480
; y 960 µs).
; 2º. El PIC pasa a modo entrada, espera 20 µs. (El protocolo fija entre 15 y 60 µs).
; 3º. El PIC lee la linea de entrada y si es bajo está correcto (pulso de "Presence").
; Se interpreta como que el procedimiento está correctamente inicializado.
```

```
Bus1Linea_Res
call
call
call
call
call
call
Bus1Linea_Esp
bfsc
goto
Bus1Linea_Esp
bfss
goto
return
;
; Subrutina "Bus
; El maestro esc
; Consiste en en
; 1º El PIC
minim
- Si v
- Si v
perm
; 2º Para e
establ
;
; Entrada: En (V
Bus1Linea_Esc
movw
movi
movw
call
Bus1Linea_Env
call
;
; Pregunta por e
;
; rff
; bfss
; goto
Bus1Linea_Ear
call
goto
Bus1Linea_Ear
call
Bus1Linea_Ear
call
call
decf
got
return
```

ctado en el bus  
e un byte al  
te del esclavo

\*\*\*\*\*

bus.

180

```

Bus1Linea_Reset
    call    Bus1Linea_AltaImpedancia ; Empieza en alto para conseguir el flanco de
                                    ; bajada con la siguiente instrucción.
    call    Bus1Linea_Bajo          ; El bus de 1 línea en bajo durante 500 µs.
    call    Retardo_500micros
    call    Bus1Linea_AltaImpedancia ; Permanece en alta impedancia (nivel alto por
                                    ; las resistencias de Pull-Up) durante unos µs.
    call    Retardo_20micros
Bus1Linea_EsperaLeerBajo
    btfsc  Bus1Linea             ; Espera a que el dispositivo le envíe el pulso
    goto   Bus1Linea_EsperaLeerBajo ; en bajo de "Presence".
Bus1Linea_EsperaLeerAlto
    btfsf  Bus1Linea             ; Y ahora el dispositivo lo debe poner en alto.
    goto   Bus1Linea_EsperaLeerAlto
    return

; Subrutina "Bus1Linea_EscribeByte" -----
;
; El maestro escribe un byte al esclavo conectado al bus de 1 línea comenzando por el bit LSB.
; Consiste en enviar 8 bits, repitiendo 8 veces la siguiente secuencia:
; 1º    El PIC manda un impulso en bajo durante unos microsegundos. (El protocolo fija un
;           mínimo de 1 µs y sin exceder con lo que viene a continuación de 60 µs) A continuación:
;           - Si va a enviar un "0" permanece en bajo durante 50 µs.
;           - Si va a escribir un "1" se pone en alta impedancia durante 50 µs. (El protocolo
;             permite unos valores entre 15 y 60 µs para ambos casos).
; 2º    Para escribir otro bit se pone en alta impedancia durante unos µs. (El protocolo
;           establece un mínimo de 1 µs).
;
; Entrada: En (W) el dato a enviar.

Bus1Linea_EscribeByte
    movwf  Bus1Linea_Dato         ; Guarda el byte a transmitir.
    movlw   0x08                  ; 8 bits a transmitir.
    movwf  Bus1Linea_ContadorBits
    call    Bus1Linea_AltaImpedancia ; Empieza en alta impedancia.
Bus1Linea_EnviaBit
    call    Bus1Linea_Bajo         ; Flanco de bajada.
;
; Pregunta por el valor del bit a transmitir.
;
    rrf    Bus1Linea_Dato,F       ; Lleva el bit de menor peso LSB al Carry.
    btfsf  STATUS,C             ; ¿Es un "1" el bit a transmitir?.
    goto   Bus1Linea_EnviaCero   ; No, pues envía un "0".
Bus1Linea_EnviaUno
    call    Bus1Linea_AltaImpedancia ; Transmite un "1".
    goto   Bus1Linea_Espera50us
Bus1Linea_EnviaCero
    call    Bus1Linea_Bajo         ; Transmite un "0".
Bus1Linea_Espera50us
    call    Retardo_50micros
    call    Bus1Linea_AltaImpedancia ; Flanco de subida.
    decfsz Bus1Linea_ContadorBits,l ; Comprueba que es el último bit.
    goto   Bus1Linea_EnviaBit     ; Como no es el último bit repite la operación.
    return

```

```

; Subrutina "Bus1Linea_LeeByte"
;
; El microcontrolador maestro lee un byte del dispositivo esclavo conectado al bus de
; 1 linea. Consiste en leer 8 bits, repitiendo 8 veces la siguiente secuencia:
; 1º El PIC manda un pulso en bajo durante unos µs. (El protocolo fija un mínimo de 1 µs).
; 2º El PIC se pone en entrada (alta impedancia) durante otros 10 µs (con un total de 15 µs
; desde el flanco de bajada) para proceder a la lectura de la línea.
; 3º El PIC espera 50 µs para realizar la siguiente lectura (total unos 65 µs desde el
; flanco de bajada). El protocolo fija que las lecturas se harán cada 60 µs ó más.
;
; Salida: En (W) el dato leido.

```

**Bus1Linea\_LeeByte**

```

    movlw 0x08          ; 8 bits a recibir.
    movwf Bus1Linea_ContadorBits
    call Bus1Linea_AltalImpedancia ; Empieza en alta impedancia.
Bus1Linea_LeeBit
    call Bus1Linea_Bajo      ; Flanco de bajada.
    call Bus1Linea_AltalImpedancia
    call Retardo_10micros   ; Añadiendo las instrucciones anteriores, supone
                           ; un total de 15 µs desde el flanco de bajada.
    bcf STATUS,C           ; Ahora lee el pin. En principio supone que es 0.
    btfsc Bus1Linea         ; ¿Realmente es cero?
    bsf STATUS,C           ; No, pues cambia a uno.
    rrf Bus1Linea_Dato,l   ; Introduce el bit en el registro.
    call Retardo_50micros  ; Espera.
    call Bus1Linea_AltalImpedancia
    decfsz Bus1Linea_ContadorBits,l
    goto Bus1Linea_LeeBit   ; Comprueba que es el último bit.
    movf Bus1Linea_Dato,W   ; Si no es el último bit pasa a leer el siguiente.
    return

```

## 28.10 ÚNICO DS1820 CONECTADO AL BUS DE 1 LÍNEA

Cuando hay un único DS1820 esclavo conectado a un bus de 1 línea de *Dallas Semiconductors*, como en el termómetro digital ilustrado en la figura 28-6, el fabricante simplifica el control software de éste mediante el envío de los comandos especificados en el diagrama de flujos de la figura 28-7.

Este control se puede concretar en las siguientes subrutinas descritas en la librería DS1820.INC:

- "DS1820\_Inicializa". Inicializa el DS1820 para que comience la conversión de temperatura.
- "DS1820\_LeeTemperatura". Proporciona el valor de la temperatura en tres registros:
  - (DS1820\_Temperatura), valor absoluto de la temperatura.
  - (DS1820\_TemperaturaDecimal), parte decimal.
  - (DS1820\_TemperaturaSigno), informa si la temperatura es negativa.

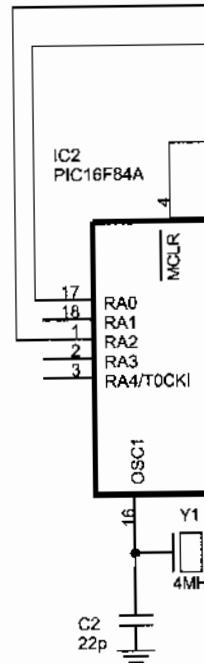


Figura 28-6 Ejemp...

```

*****
;
; Estas subrutinas permit...
; Este sensor transmite la...
; ZONA DE DATOS ***

```

```

CBLOCK
DS1820_Temp
DS1820_Temp
DS1820_Temp
ENDC

```

```

DS1820_Skip_ROM_C
DS1820_Read_Scratchp
DS1820_Convert_T_C
;
; Subrutina "DS1820_In...
;
; Inicializa el DS1820 pa...
;
DS1820_Inicializa
    call Bus1Linea_LeeByte

```

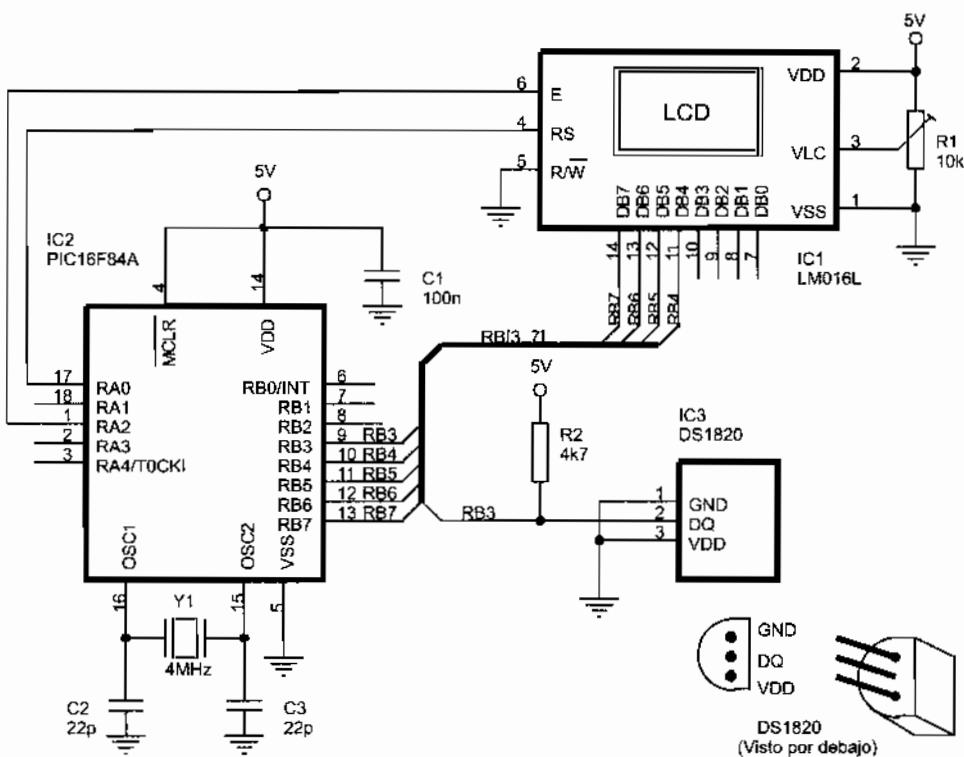


Figura 28-6 Ejemplo de conexión de un DS1820 a un PIC16F84A. Termómetro digital

```
;***** Librería "DS1820.INC" *****
;
; Estas subrutinas permiten realizar las tareas de manejo del sensor de temperatura DS1820.
; Este sensor transmite la información vía serie a través de un 1-Wire Bus.
;
; ZONA DE DATOS *****
CBLOCK
    DS1820_Temperatura          ; Primer byte leído del termómetro DS1820.
    DS1820_TemperaturaSigno     ; Segundo byte leído del termómetro DS1820.
    DS1820_TemperaturaDecimal   ; Parte decimal de la temperatura medida.
ENDC

DS1820_Skip_ROM_Command      EQU 0xCC ; Comandos del DS1820.
DS1820_Read_Scratchpad_Command EQU 0xBE
DS1820_Convert_T_Command      EQU 0x44

;
; Subrutina "DS1820_Inicializa"
;
; Inicializa el DS1820 para que comience la conversión de temperatura.
;
DS1820_Inicializa
    call Bus1Linea_Reset        ; Primero los pulsos de Reset y Presence.
```

```

movlw DS1820_Skip_ROM_Command ; Salta los comandos de ROM ya que hay
call Bus1Linea_EscribeByte ; conectado un único dispositivo al bus.
movlw DS1820_Convert_T_Command ; Para que comience la conversión de temperatura.
call Bus1Linea_EscribeByte
return

```

; Subrutina "DS1820\_LeeTemperatura"

; El DS1820 lee la temperatura en un formato ejemplarizado como sigue:

```

; +125,0 °C      00000000 11111010
; + 25,5 °C      00000000 00110011
; + 25,0 °C      00000000 00110010
; + 0,5 °C 00000000 00000001
; + 0,0 °C 00000000 00000000
; - 0,5 °C 11111111 11111111
; - 25,0 °C 11111111 11001110
; - 55,0 °C 11111111 10010010

```

; Se observa que:

- El formato es de dos bytes. El primer byte es el de signo.
- Las temperaturas positivas comienzan con 1 byte todo a cero: "00000000 xxxxxxxx".
- Las temperaturas negativas comienzan con 1 byte todo a uno: "11111111 xxxxxxxx".
- Las temperaturas "xx.0 °C" terminan en cero: "xxxxxxxx0".
- Las temperaturas "xx.5 °C" terminan en uno: "xxxxxxxx1".
- El valor de la temperatura está contenido en los 7 bits de mayor peso del byte bajo. Por ejemplo: +25°C = xxxxxxxx 0011001x = 25 en decimal. Por tanto, para obtener el valor de la temperatura habrá que desplazar un lugar hacia la derecha los bits del byte bajo.
- Las temperaturas negativas vienen dadas en complemento a 2. Por tanto para obtener su valor absoluto habrá que negarlas.

; Esta subrutina lee la temperatura proporcionando tres datos:

; Salida:

- En el registro DS1820\_Temperatura, el valor de la temperatura en valor absoluto.
- En el registro DS1820\_TemperaturaDecimal, la parte decimal del valor de la temperatura.
- En (DS1820\_TemperaturaSigno)=b'1111111' si la temperatura es negativa y  
(DS1820\_TemperaturaSigno)=b'0000000' si es positiva.

#### DS1820\_LeeTemperatura

```

call Bus1Linea_Reset ; Primero los pulsos de "Reset" y "Presence".
movlw DS1820_Skip_ROM_Command ; Salta los comandos de ROM ya que hay
call Bus1Linea_EscribeByte ; conectado un único dispositivo al bus.
movlw DS1820_Read_Scratchpad_Command ; Va a leer la primera posición de memoria,
call Bus1Linea_EscribeByte ; donde se localiza el valor de la temperatura.
call Bus1Linea_LeeByte ; Lee la temperatura en
movwf DS1820_Temperatura ; complemento a 2.
call Bus1Linea_LeeByte ; Ahora lee el segundo byte que indica el signo
movwf DS1820_TemperaturaSigno ; de la temperatura.
call Bus1Linea_Reset ; Y para terminar, resetea el dispositivo.

```

; Ahora deduce si la temperatura es positiva o negativa y halla su valor absoluto.

```

btss
goto
DS1820_Temperatu
movf
sublw
movwf
DS1820_FinLeeTer
bcf
trf
movlw
bfss
clrw
movwf
movf
return

```

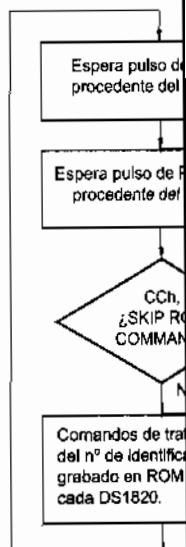


Figura 28-7 T...

## 28.11 TER...

Un proyecto

El funcio

programa de co

figuras 28-9 a 28

```

atura.

        btfss DS1820_TemperaturaSigno,7 ; Si cualquier bit es "1": temperatura negativa.
        goto DS1820_FinLeeTemperatura ; La temperatura es positiva y salta.

DS1820_TemperaturaNegativa
        movf DS1820_Temperatura,W      ; La niega para convertirla a su valor
        sublw 0x00                      ; absoluto. Para ello, simplemente le resta de
        movwf DS1820_Temperatura       ; cero.

DS1820_FinLeeTemperatura
        bcf STATUS,C                  ; Para no corromper luego el MSB.
        rrf DS1820_Temperatura,F      ; Obtiene el valor absoluto de la temperatura.
        movlw .5
        btfss STATUS,C                ; Si el bit LSB es "1", la temperatura
        clrw                           ; medida es "xx.5°C" sino "xx.0°C".
        movwf DS1820_TemperaturaDecimal
        movf DS1820_Temperatura,W      ; En (W) la parte entera del valor absoluto de la
        return                          ; temperatura.
    
```

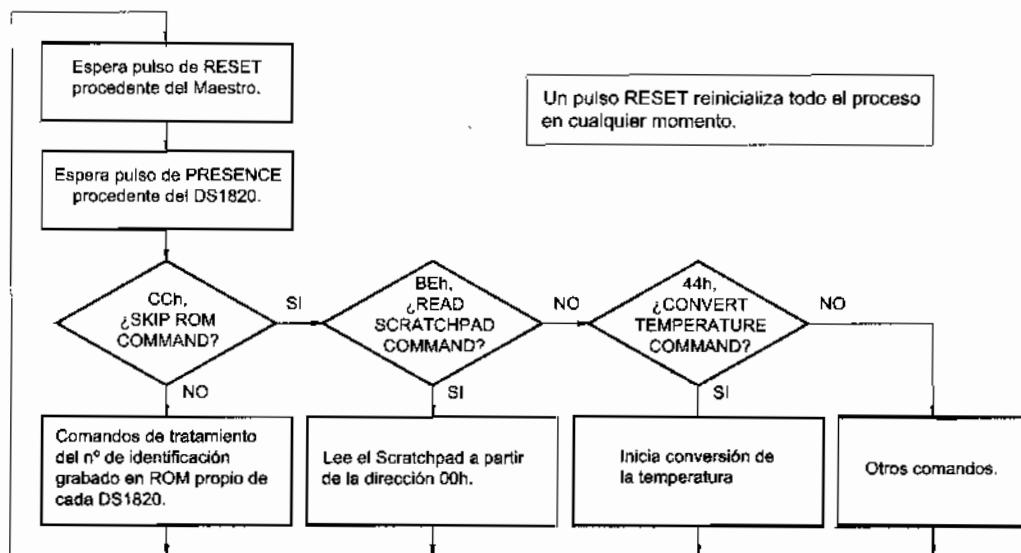


Figura 28-7 Tabla resumida del procedimiento de los comandos de control del DS1820

## 28.11 TERMOSTATO DIGITAL

Un proyecto clásico es el diseño de un termostato digital. Como aplicación práctica del DS1820 se va a proceder a ello según el esquema de la figura 28-8, donde a su salida hay conectado un módulo de potencia para alimentar una carga con los 230 V de la red eléctrica analizado en el capítulo 2.

El funcionamiento de este termostato digital se detalla en los comentarios del programa de control expuesto a continuación y en los diagramas descriptivos insertados, figuras 28-9 a 28-13.

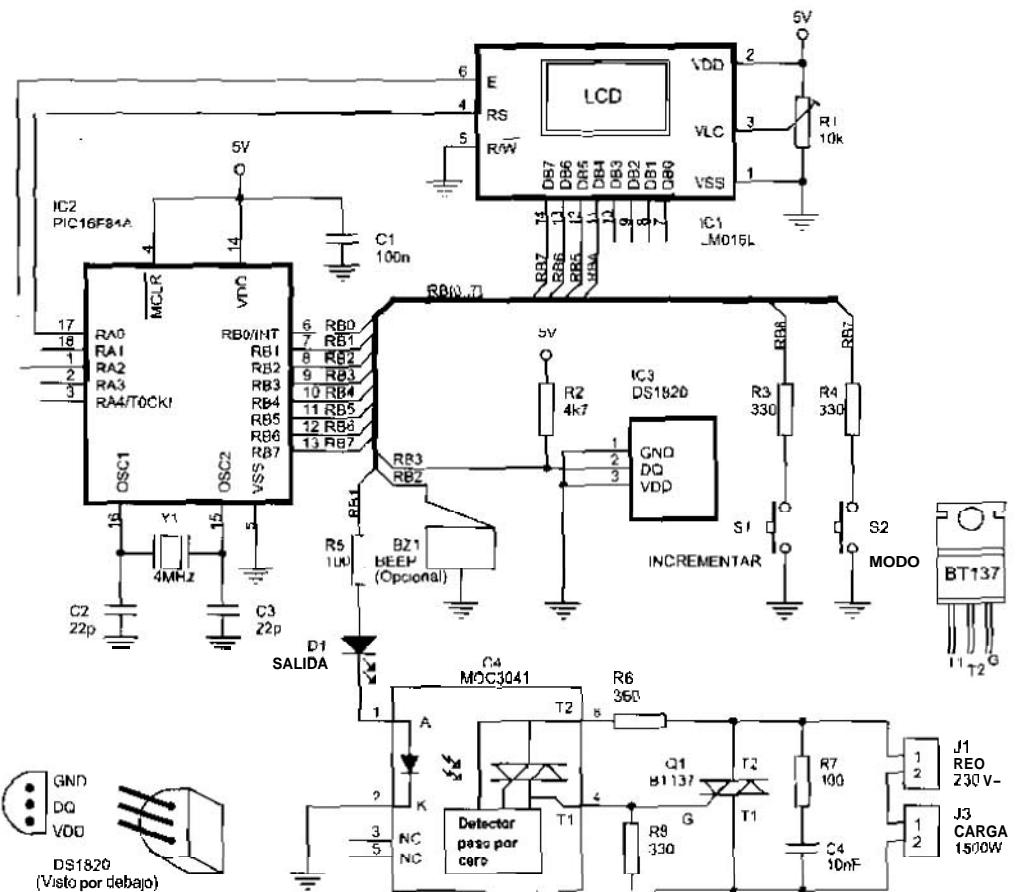


Figura 28-8 Esquema de un termostato digital de precisión

\*\*\*\*\* DS1820 Termostato.asm \*\*\*\*\*

Programa de control para un termómetro y termostato digital. Utiliza el sensor de temperatura DS1820 que transmite la información vía serie a través de un bus de una sola línea según un protocolo del fabricante de semiconductores Dallas Semiconductors.

El ajuste de la temperatura a la que comuta el termostato se logra mediante dos pulsadores: "MODO" e "INCREMENTAR", que se conectan a pines del Puerto B y cuyo funcionamiento se basa en interrupción por cambio en la línea del Puerto B.

Se maneja de la siguiente forma:

- En estado de reposo funciona sólo como termómetro. Aparece la temperatura en pantalla del módulo LCD. La salida del termostato está apagada
- Pulsa "MODO" y se ajusta la temperatura deseada mediante el pulsador "INCREMENTAR"
- Vuelve a pulsar "MODO". se activa el termostato. Si la temperatura medida es menor que la deseada enciende la carga, que puede ser un calefactor. Si la temperatura medida es mayor que la deseada, apaga la carga.
- Si se vuelve a pulsar "MODO", apaga la carga y pasa a funcionar sólo como termómetro.

; Así pues, en el ci mediante tres fla  
; A) Modo "r  
; B) Modo "r  
; C) Modo "r  
;

; El programa cons

; Al apagar el sist  
; para la próxima v

; ZONA DE DATO

CONF  
LIST  
INCLUDE

CBLOCK  
Tempera  
Registro  
FlagsMo  
ENDC

ORG  
DE

```
#DEFINE SalidaTemp
#DEFINE Zumbador
#DEFINE ModoPunto
#DEFINE Incrementar
#DEFINE F_Temperatura
#DEFINE F_Temperatura
#DEFINE F_Temperatura
```

TMR0\_Carga50ms
Carga2s

; ZONA DE CÓDIGO

ORG
goto
ORG
goto

Mensajes
addwf
MensajePublicitario
DT "ES

; Así pues, en el circuito se distinguen tres modos de funcionamiento que se identifican mediante tres flags:

- ; A) Modo "Termostato\_OFF", donde funciona como termómetro normal sin termostato. Se reconoce por el flag F\_Termostato\_OFF.
- ; B) Modo "Termostato\_Ajuste", donde se ajusta la temperatura deseada cuando funcione como termostato. Se reconoce por el flag F\_Termostato\_Ajuste.
- ; C) Modo "Termostato\_ON", donde funciona como termómetro normal con termostato. Se reconoce por el flag F\_Termostato\_ON.

; El programa consigue que esté activado uno solo de los flags anteriores.

; Al apagar el sistema debe conservar el valor de la temperatura deseada en el termostato para la próxima vez que se encienda.

; ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK0x0C
TemperaturaDeseada
Registro50ms
FlagsModos
ENDC

ORG 0x2100
DE .24

#DEFINE SalidaTermostato PORTB,1
#DEFINE Zumbador PORTB,2
#DEFINE ModoPulsador PORTB,7
#DEFINE IncrementarPulsador PORTB,6
#DEFINE F_Termostato_ON FlagsModos,2
#DEFINE F_Termostato_Ajuste FlagsModos,1
#DEFINE F_Termostato_OFF FlagsModos,0

TMR0_Carga50ms EQU -d'195'
Carga2s EQU d'40'

; Corresponde a la dirección 0 da la zona
; EEPROM de datos. Aquí se va a guardar el
; la temperatura deseada. En principio 24 °C.

; Guarda los incrementos cada 50 ms.
; Guarda los flags para establecer los
; modos de trabajo

; Carga controlada por el termostato.
; Aquí se conecta el zumbador.
; Los pulsadores se conectan a estos
; pines del puerto B.
; Flags utilizados en el ajuste de la
; temperatura del termostato.

; Para conseguir interrupción cada 50 ms.
; Leerá cada 2s = 40 x 50ms = 2000ms.
```

; ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG 0
goto Inicio
ORG 4
goto ServicioInterrupcion
```

#### Mensajes

```
addwf PCL,F
MensajePublicitario
DT "IES. ISAAC PERAL", 0x00
```

```

MensajeTermostato_ON
    DT "Termostato: ", 0x00
MensajeTermostato_Ajuste
    DT "Temper. deseada", 0x00
MensajeGradoCentigrado
    DT "°C ", 0x00

```

; En pantalla LCD: "°C "

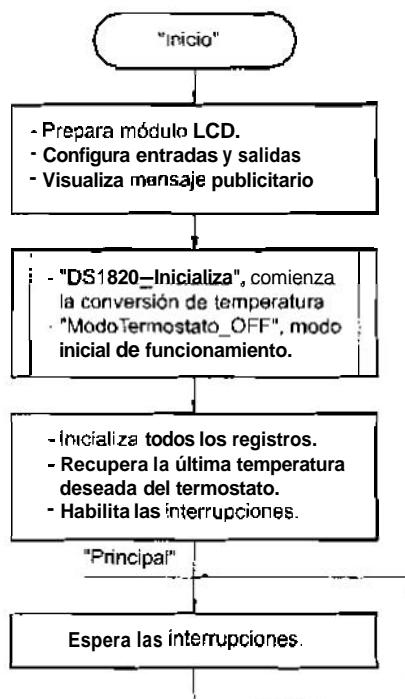


Figura 28-9 Diagrama de flujo principal del termostato digital

Inicio	call	LCD_Inicializa	
	bsf	STATUS,RP0	
	movlw	b'00000111'	; Prescaler de 256 para el TMR0 y habilita resistencia de Pull-Up del Puerto B.
	movwf	OPTION_REG	; Se configuran como entrada.
	bsf	ModoPulsador	
	bsf	IncrementarPulsador	
	bcf	SalidaTermostato	; Se configuran como salida.
	bcf	Zumbador	
	bcf	STATUS,RP0	
	call	LCD_Linea1	; Se sitúa al principio de la primera línea.
	movlw	MensajePublicitario	
	call	LCD_Mensaje	
	call	DS1820_Inicializa	; Comienza la conversión del termómetro y pone este modo de funcionamiento.
	call	ModoTermostato_OFF	; Carga el TMR0 en complemento a 2.
	movlw	TMR0_Carga50ms	
	movwf	TMR0	
	movlw	Carga2s	; Y el registro cuyo decremento contará los 2 s.
	movwf	Registro50ms	

Figura 2

; Subrutina "Ser"; Detecta qué ha ServicioInterrup

```

clrw      ; Lee la posición 0x00 de memoria EEPROM de datos
call     EEPROM_LeeDatos
movwf    TemperaturaDeseada
movlw    b'10101000'
movwf    INTCON
; donde se guarda la temperatura deseada de la última
; vez que se ajustó.
; Activa interrupción del TMR0 (TOIE), por cambio de
; líneas del Puerto B (RBIE) y la general (GIE)

```

; La sección "Principal" es mantenimiento. Sólo espera las interrupciones.  
; No se puede poner en modo de bajo consumo porque la instrucción "sleep" detiene el Timer 0.

Principal  
gato      Principal

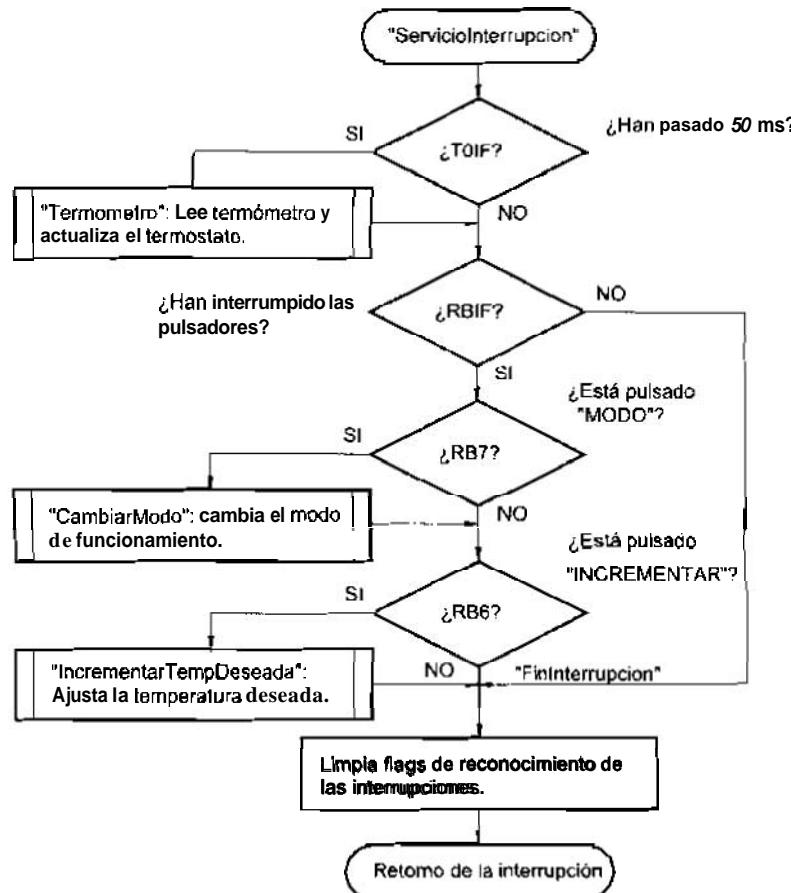


Figura 28-10 Diagrama de flujo de la subrutina de atención a las interrupciones

; Subrutina "ServicioInterrupcion"

;

; Detecta qué ha producido la interrupción y ejecuta la subrutina de atención correspondiente.

ServicioInterrupcion

```

btfc  INTCON,T0IF      ; Si es una interrupción producida por el Timer 0
call   Termometro       ; lee el termómetro y actualiza termostato.
btfs  INTCON,RBIF      ; Si es una interrupción RBI lee los pulsadores.
goto  FinInterrupcion
btfs  ModoPulsador     ; ¿Está presionado el pulsador de "AJUSTE"?
call   CambiarModo     ; Sí. Ajusta la temperatura deseada en el termostato.
btfs  IncrementarPulsador; ¿Pulsado "INCREMENTAR"?
call   IncrementarTempDeseada ; Si, pasa a incrementar la temperatura deseada.
FinInterrupcion
bcf   INTCON,RBIF      ; Limpia los flgs de reconocimiento.
bcf   INTCON,T0IF
retfie

```

```

; Subrutina "Termometro"
;
; Esta subrutina lee y
; debido a la petición
; temporización de 2
;
; También actúa sobr
Termometro
    movlw  ...
    movwf ...
    decfsz ...
    goto ...
    movlw  ...
    movwf ...
    call ...
    call ...
    call ...
    call ...
    return

```

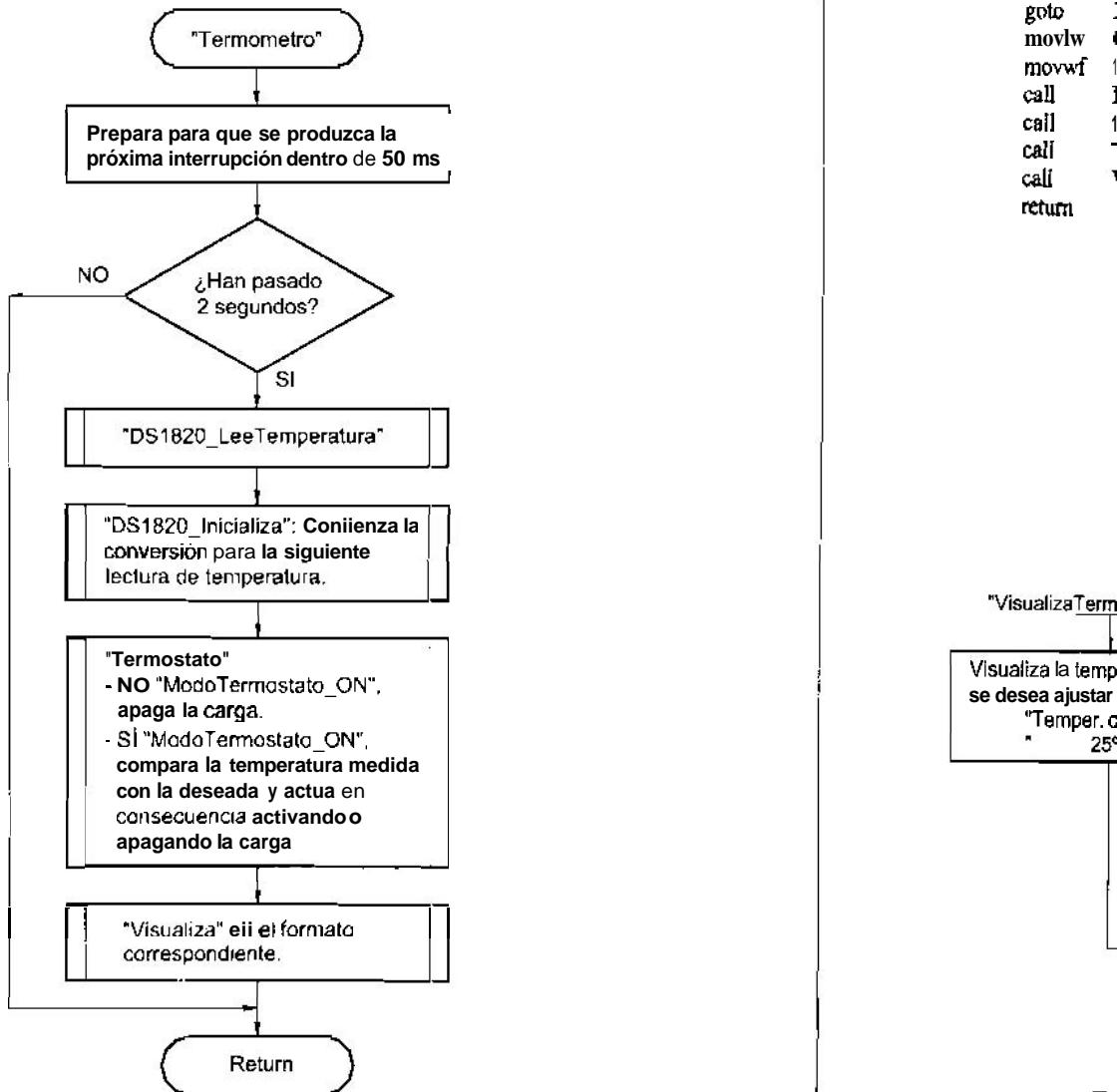


Figura 28-11 Diagrama de flujo de la subrutina Termometro

Fig

## • Subrutina "Termómetro"

; Esta subrutina lee y visualiza el termómetro cada 2 segundos aproximadamente. Se ejecuta debido a la petición de interrupción del Timer 0, cada 50 ms. Para conseguir una temporización de 2 s, habrá que repetir 40 veces el lazo de 50 ms ( $40 \times 50\text{ms} = 2000\text{ms} = 2\text{s}$ ).

; También actúa sobre la salida del termostato posicionándola adecuadamente.

## Termómetro

movlw	TMR0_Carga50ms	
.movwf	TMR0	; Recarga el TMRO.
.decfsz	Registro50ms,F	; Decrementa el contador.
goto	FinInterrupcion	; No han pasado 2 segundos, por tanto sale
movlw	Carga2s	; Repone este contador nuevamente.
.movwf	Registro50ms	
call	DS1820_LeeTemperatura	; Lee la temperatura.
call	DS1820_Inicializa	; Comienza conversión para la siguiente lectura.
call	Termostato	; Actúa sobre el termostato.
call	Visualiza	; Como esta subrutina se escribe a continuación
return		; se ahorra estas dos instrucciones y ahorra
		; también espacio en la pila.

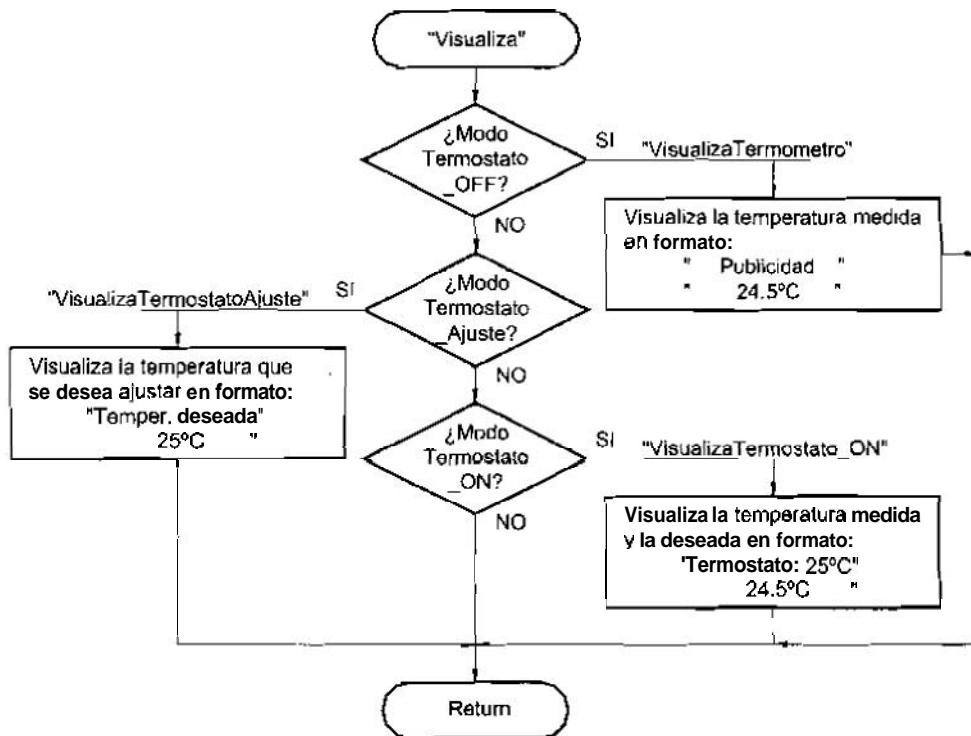


Figura 28-12 Diagrama de flujo de la subrutina Visualiza

; Subrutina "Visualiza"

; Visualiza el termómetro en tres formatos posibles:

; A) Con el termostato desactivado, modo "Termostato\_OFF". Por ejemplo:

"IES, Isaac Peral" (Primera linea)

" 24.5°C " (Segunda linea).

Donde en la primera linea se visualiza un mensaje publicitario y en la segunda linea la temperatura medida actual.

; B) Ajuste del termostato, modo "Termostato\_Ajuste". Por ejemplo:

"Temper. deseada" (Primera linea)

" 23°C " (Segunda linea).

Donde en la segunda linea visualiza la temperatura que se desea ajustar.

; C) Con el termostato activado, modo "Termostato\_ON". Por ejemplo:

"Termostato: 23°C" (Primera linea)

" 23.5°C " (Segunda linea).

Donde en la primera linea se visualiza la temperatura que se desea ajustar y en la segunda linea la temperatura medida actual.

Visualiza

```
btfsr F_Termostato_OFF
goto VisualizaTermometro
btfsr F_Termostato_Ajuste
goto VisualizaTermostato_Ajuste
btfsr F_Termostato_ON
goto VisualizaTermostato_ON
return
```

; "VisualizaTermostato\_ON"

; Visualiza el valor de la temperatura deseada en la primera linea y el valor de la temperatura medida en la segunda linea.

VisualizaTermostato\_ON

```
call LCD_Lineal
movlw MensajeTermostato_ON
call LCD_Mensaje
call VisualizaTemperaturaDeseada
call VisualizaTemperaturaMedida
return
```

; "VisualizaTermostatoAjuste" y "VisualizaTemperaturaDeseada"

; Visualiza en la pantalla el formato propio de este modo.

; Entradas: (TemperaturaDeseada) temperatura ajustada en la subrutina Incrementar.

VisualizaTermostato\_Ajuste

```
call LCD_Lineal
movlw MensajeTermostato_Ajuste ; Se sitúa al principio de la primera linea.
                                ; Visualiza mensaje en la primera linea.
call LCD_Mensaje
movlw .6                         ; Se coloca para centrar visualización en la
call LCD_PosicionLinea2          ; segunda linea.
```

VisualizaTemperaturaDeseada

```
movwf TemperaturaDeseada,W
```

```
call
call
movlw
call
return
```

; "VisualizaTermometro"

; En la primera linea

; temperatura medida

; Entradas: -

VisualizaTermometro

```
call
movlw
call
```

VisualizaTemperatura

```
movlw
call
btfs
goto
```

TemperaturaNegativa

```
movlw
call
```

TemperaturaPositiva

```
movf
call
call
movlw
call
movf
call
movlw
call
return
```

; Subrutina "Termostato\_Ajuste"

; Controla una carga o descarga de la resistencia de acuerdo con la temperatura deseada. Para evitar

; Así por ejemplo, si la temperatura deseada es menor que la temperatura actual, se enciende la resistencia.

; Si la temperatura deseada es mayor que la temperatura actual, se apaga la resistencia.

; Para temperaturas negativas se enciende la resistencia.

; Entradas: -

```

call    BIN_a_BCD           ; La pasa a BCD.
call    LCD_Byt              ; Visualiza, apagando los ceros no significativos.
movlw   MensajeGradoCentigrado ; En pantalla aparece "°C".
call    LCD_Mensaje
return

;"VisualizaTermometro" y ""VisualizaTemperaturaMedida"
;
;En la primera linea se visualiza un mensaje publicitario y en la segunda linea la
;temperatura medida
;
;Entradas: - (DS1820_Temperatura), temperatura medida en valor absoluto.
;- (DS1820_TemperaturaDecimal), parte decimal de la temperatura medida.
;- (DS1820_Signo), registro con el signo de la temperatura. Si es igual a
;b'00000000' la temperatura es positiva. Si es b'11111111' resulta que
;la temperatura es negativa.
;
VisualizaTermometro
call    LCD_Lineal           ; Se sitúa al principio de la primera linea.
movlw   MensajePublicitario
call    LCD_Mensaje

VisualizaTemperaturaMedida
movlw   .5                  ; Se coloca para centrar visualización en la
call    LCD_PosicionLinea2  ; segunda linea.
btfs   DS1820_TemperaturaSigno,7 ; Temperatura negativa?
goto   TemperaturaPositiva  ; No, es positiva.

TemperaturaNegativa:
movlw   '-'                ; Visualiza el signo "-" de temperatura negativa.
call    LCD_Caracter

TemperaturaPositiva
movf   DS1820_Temperatura,W
call    BIN_a_BCD           ; La pasa a BCD.
call    LCD_Byt              ; Visualiza apagando los ceros no significativos.
movlw   '.'
call    LCD_Caracter         ; Visualiza el punto decimal.
movf   DS1820_TemperaturaDecimal,W ; Visualiza la parte decimal.
call    LCD_Nibble
movlw   MensajeGradoCentigrado ; En pantalla LCD aparece "°C".
call    LCD_Mensaje
return

;Subrutina "Termostato"
;
;Controla una carga en función del valor de la temperatura medida respecto de la temperatura
;deseada. Para evitar inestabilidad en la salida, tendrá un pequeño ciclo de histéresis.
;Así por ejemplo, si la temperatura deseada es 24 °C, la carga se activará cuando la
;temperatura baje o sea igual a 23,5 °C y se apagará cuando la supere o sea igual a 25°C.
;Si la temperatura medida está entre esos márgenes (23,5 y 25°C), se queda en el estado
;anterior, tanto si está encendida como apagada.
;
;Para temperaturas negativas la salida se debe activar siempre.
;
;Entradas: - (DS1820_Temperatura), temperatura medida en valor absoluto.

```

- (*TemperaturaDeseada*), temperatura a partir de la cual se tomarán decisiones sobre la salida.
  - (*DS1820\_Signo*), registro con el signo de la temperatura medida. Si es cero la temperatura es positiva y todos sus bits son "1", es negativa

;Salida - Su funcionamiento:

  - Estando apagada, si la temperatura medida desciende por debajo de la temperatura deseada la salida se activará.
  - Estando encendida, si la temperatura medida supera la deseada la salida se apagará.
  - Si las temperaturas medidas y deseada son iguales se queda en estado anterior, tanto si está encendida como si está apagada.
  - Para temperaturas negativas la salida se debe activar siempre.

## Termostato

	<b>bffs</b>	F_Termostato_ON	; Si el termostato no está activado salta a
	<b>goto</b>	ApagaCarga	; apagar la carga.
	<b>btfcsc</b>	DS1820_TemperaturaSigno.7	; Con temperaturas negativas pasa a activar
	<b>goto</b>	EnciendeCarga	; la carga.
	<b>bffs</b>	SalidaTermostato	; Comprueba el estado actual de la salida para
	<b>goto</b>	SalidaEstabaApagada	; actuar en consecuencia.
<b>SalidaEstabaActivada</b>			; Pava a comprobar si tiene que apagar la carga.
	<b>movf</b>	DS1820_Temperatura,W	
	<b>subwf</b>	TemperaturaDeseada,W	; (W)=(TemperaturaDeseada)-(DS1820_Temperatura).
	<b>btfcsc</b>	STATUS,C	; ¿(TemperaturaDeseada)<(DS1820_Temperatura)?
	<b>goto</b>	FinTermostato	; Si, por tanto, lo deja encendido y sale.
	<b>call</b>	Pitido	; Pitido cada vez que commuta la carga.
<b>ApagaCarga</b>			
	<b>bef</b>	SalidaTermostato	; Apaga la salida y sale.
	<b>goto</b>	FinTermostato	
<b>SalidaEstabaApagada</b>			; Pasa a comprobar si tiene que encender la carga
	<b>movf</b>	TemperaturaDeseada,W	
	<b>subwf</b>	DS1820_Temperatura,W	; (W)=(DS1820_Temperatura)-(TemperaturaDeseada).
	<b>btfcsc</b>	STATUS,C	; ¿(DS1820_Temperatura)<(TemperaturaDeseada)?
	<b>gota</b>	FinTermostato	; Si, la deja apagada y sale.
<b>EnciendeCarga</b>			
	<b>call</b>	Pitido	; Pitido cada vez que activa la carga.
	<b>bef</b>	SalidaTermostato	

bsf  
FinTermostato  
Rattrap

Subrutinas "CambiarModo" y "ModoTermostato\_QEE"

; Subrutina de atención a la interrupción producida por el pulsador "MODO" que cambia el modo de funcionamiento. Cada vez que pulsa pasa por los modos "Termostato\_Ajuste", "Termostato\_ON", "Termostato\_OFF" y vuelta repetir.

; El ajuste de la temperatura deseada en el termostato se logra mediante dos pulsadores: "MODO" e "INCREMENTAR" conectados a pines del Puerto B.

; Al principio aparecerá sólo el termómetro y el termostato estará desactivado: modo "Termostato OFF"

; Para comprender el funcionamiento de esta subrutina, hay que saber que el registro FlagsModos  
; contiene 3 flags que permiten diferenciar cada uno de los modos de funcionamiento:

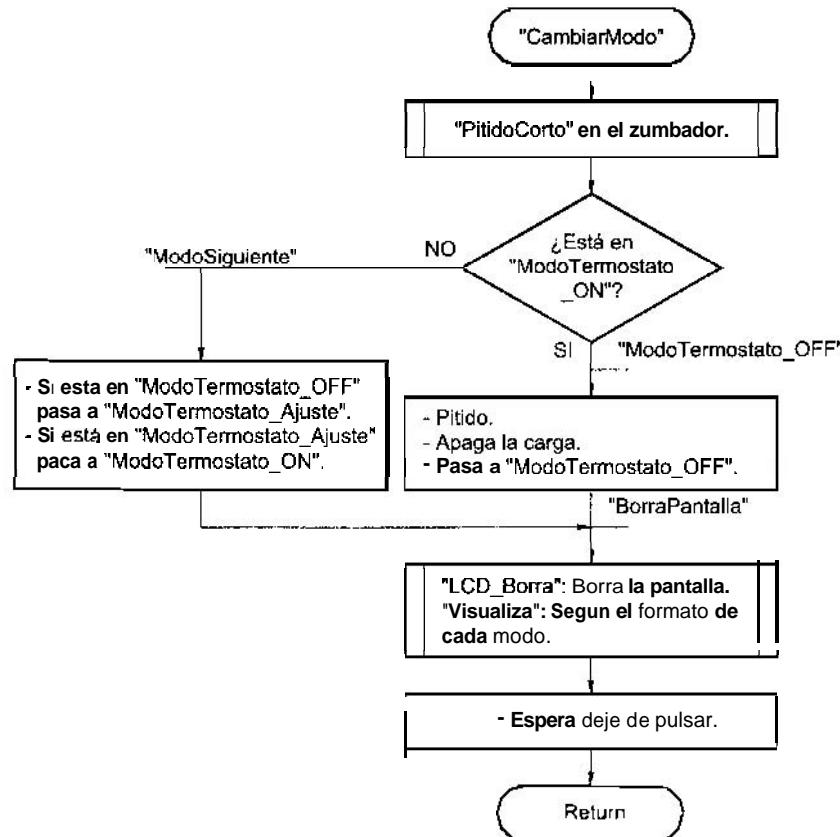


Figura 28-13 Diagrama de flujo de la subrutina CambiarModo

- ; A) Modo "Termostato \_OFF", donde funciona como termómetro normal sin termostato. Se reconoce por el flag F\_Termostato \_OFF, que es el bit 0 del registro FlagsModos.
  - ; B) Modo "Termostato \_Ajuste", donde se ajusta la temperatura deseada cuando funcione como termostato. Se reconoce por el flag F\_Termostato \_Ajuste, que es el bit 1 del registro FlagsModos.
  - ; C) Modo "Termostato \_ON", donde funciona como termómetro normal y, además, como termostato. Se reconoce por el flag F\_Termostato \_ON, que es el bit 2 del registro FlagsModos.
- ; Así pues, el contenido del registro (FlagsModos) identifica los siguientes modos de Funcionamiento:
- ; - (FlagsModos)=b'00000001'. Está en el modo "Termostato \_OFF".
  - ; - (FlagsModos)=b'00000010'. Está en el modo "Termostato \_Ajuste".
  - ; - (FlagsModos)=b'00000100'. Está en el modo "Termostato \_ON".
- ; Pueden darse dos casos:
- ; - Que pulse "AJUSTE" estando en el modo más alto, "Termostato-ON", (FlagsModos)=b'00000100'. En este caso debe pasar al modo inicial "Termostato \_OFF" poniendo (FlagsModos)=b'00000001'.
  - ; - Que pulse "AJUSTE" estado ya en cualquiera de los otros dos modos, m cuyo caso &be pasar al siguiente modo. Esto lo hace mediante un desplazamiento a izquierdas. Así, por

; ejemplo, si antes estaba en modo "Termostato\_OFF", (FlagsModos)=b'00000001', pasará a  
; (FlagsModos)=b'00000010' que identifica al modo "Termostato\_Ajuste".

**CambiarModo**

call	Retardo_20ms	; Espera a que se estabilicen niveles de tensión.
btfsc	ModoPulsador	; Si es un rebote, sale fuera.
goto	FinCambiarModo	
call	PitidoCorto	; Cada vez que pulsa se oye un pitido.
btfss	F_Termostato_ON	; Detecta si está en el último modo.
goto	ModoSiguiente	; Si no, pasa al modo siguiente.

**ModoTermostato\_OFF**

call	Pitido	; Pitido cada vez que comuta la carga.
bcf	SalidaTermostato	; Apaga la carga.
movlw	b'00000001'	; Actualiza el registro FlagsModos pasando al
movwf	FlagsModos	; modo inicial "Termostato_OFF".
goto	BorraPantalla	

**ModoSiguiente**

bcf	STATUS,C	; Desplaza un "1" a la izquierda del registro
rif	FlagsModos,F	; FlagsModos para ajustar secuencialmente
		; cada uno de los modos de funcionamiento.

**BorraPantalla**

call	LCD_Borra	; Borra la pantalla anterior.
------	-----------	-------------------------------

**FinCambiarModo**

call	Visualiza	
btfss	ModoPulsador	; Ahora espera a que deje de pulsar.
goto	FinCambiarModo	

return		
--------	--	--

## ; Subrutina "IncrementarTempDeseada"

; Subrutina de atención a la interrupción por cambio de la línea RB6 a la cual se ha conectado  
; el pulsador "INCREMENTAR". Estando en el modo "Termostato\_Ajustar" incrementa el valor de  
; la temperatura deseada entre unos valores máximo y mínimo.

; Al final debe guardar el valor de la temperatura deseada en memoria EEPROM de datos para  
; preservar su valor en caso que desaparezca la alimentación.

TemperaturaMinima EQU .20  
TemperaturaMaxima EQU .36

**IncrementarTempDeseada**

call	Retardo_20ms	; Espera a que se estabilicen niveles de tensión.
btfsc	IncrementarPulsador; Si es un rebote sale fuera.	
goto	FinIncrementar	
btfss	F_Termostato_Ajuste	; Si no está en modo "Termostato_Ajuste" sale
goto	FinIncrementar	; fuera.
call	PitidoCorto	; Pitido cada vez que pulsa.
incf	TemperaturaDeseada,F	; Incrementa el valor de la temperatura deseada.
movlw	TemperaturaMaxima	; ¿Ha llegado a la temperatura máxima de ajuste?
subwf	TemperaturaDeseada,W	; (W) = (TemperaturaDeseada) - TemperaturaMaxima.
btfss	STATUS,C	; (TemperaturaDeseada)>=TemperaturaMaxima?
goto	VisualizaIncremento; No, pasa a visualizarlo.	
movlw	TemperaturaMinima	; Sí, entonces inicializa el registro.
movwf	TemperaturaDeseada	

VisualizaIncremento	call	
	call	
	btfss	
	goto	
	clrw	
	mov	
	mov	
	call	
FinIncrementar	retur	
	;	
	Subrutina de	
	;	
	PitidoLargo	
	bsf	
	Pitido	bsf
	call	
	;	
	PitidoCorto	
	bsf	
	call	
	bcf	
	return	
	;	
	INCI	
	END	

Este n  
28-6, que res  
pulsadores, z  
estará trabaj  
necesidad de

## VisualizaIncremento

```

    call Visualiza          ; Visualiza mientras espera a que deje
    call Retardo_200ms     ; de pulsar.
    btfss IncrementarPulsador; Mientras permanezca pulsado,
    goto IncrementarTempDeseada ; incrementa el dígito.
    clrw
    movwf EEADR
    movf TemperaturaDeseada,W ; Salva el valor de la temperatura deseada en la
    call EEPROM_EscribeDato ; posición 00h de la EEPROM de datos. Se conserva
                           ; aunque se apague la alimentación.

FinIncrementar
return

```

## ; Subrutina de pitidos

## PitidoLargo

```

    bsf Zumbador
    call Retardo_500ms
Pitido  bsf Zumbador
        call Retardo_200ms
PitidoCorto
    bsf Zumbador
    call Retardo_20ms
    bcf Zumbador
    return

```

```

INCLUDE <BUS_1LIN.INC>
INCLUDE <DS1820.INC>
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
INCLUDE <EEPROM.INC>
END

```

; Subrutinas de control del bus de 1 línea.  
; Subrutinas de control del termómetro digital.

Este mismo programa puede servir para un termómetro digital como el de la figura 28-6, que resulta ser el mismo esquema del termostato, del cual se han eliminado todos los pulsadores, zumbador e interface con la carga de 230 V. En consecuencia, el programa estará trabajando siempre en el modo "Termostato\_OFF" como simple termómetro, sin necesidad de alterar nada del programa del termostato.

## CAPÍTULO 29

# MOTORES DE CORRIENTE CONTINUA

El conocimiento de los sistemas control de motores de corriente continua **c.c.** (o **DC**) de pequeña potencia es fundamental para cualquier aficionado que quiera realizar proyectos con microcontroladores, por lo que le dedicamos este capítulo.

El primer problema a considerar es la forma de alimentar el motor, ya que la corriente máxima que puede proporcionar cualquier línea de salida de un PIC16F84A está limitada a 25 mA como máximo. Esta corriente es demasiado pobre para alimentar un motor DC directamente. Por ello, se hace necesario la utilización de transistores que pueden ser configurados en diferentes disposiciones, siendo la más utilizada el Puente en H (figura 29-1).

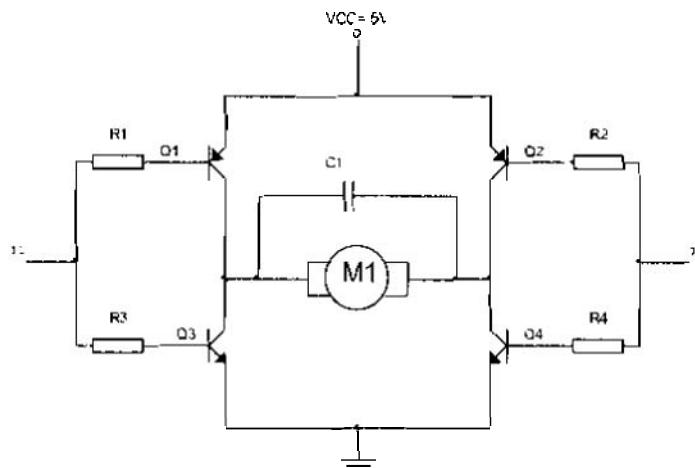


Figura 29-1 Puente en H con transistores

## 29.1 PUENTE EN H

Es conocido que el sentido de giro de un motor de corriente continua depende de la polaridad que se aplica a sus terminales, en consecuencia para cambiar el giro es necesario intercambiar los terminales del motor o bien cambiar la polaridad de la alimentación.

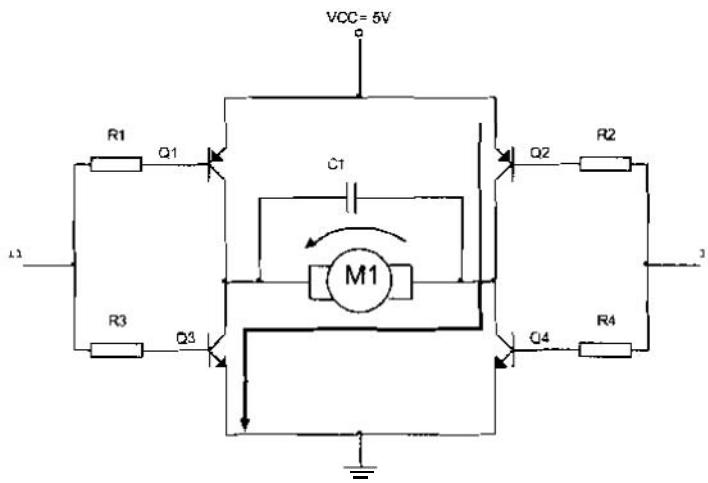


Figura 29-2 Funcionamiento con  $I_1$  a nivel alto e  $I_2$  a nivel bajo

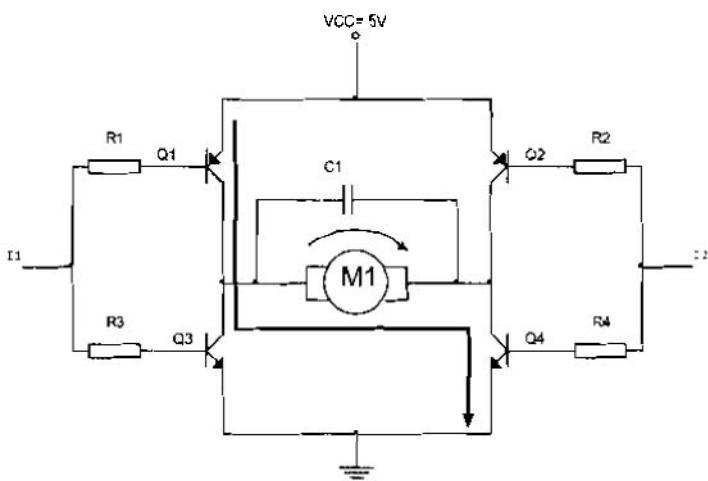


Figura 29-3 Funcionamiento con  $I_1$  a nivel alto e  $I_2$  a nivel bajo

La forma más sencilla de controlar un motor de corriente continua de baja potencia, en velocidad y sentido de giro, es mediante la commutación electrónica de unos circuitos realizados básicamente con transistores que reciben el nombre de Puente en H, como se describe en la figura 29-1.

Este ci  
comportan co  
Su funcionam

- Cuando el transistor Q1 se enciende, el transistor Q2 se apaga. El resultado es que el terminal de salida Z2 es alto. La polaridad de la alimentación es la misma que en la figura 29-1.
- Cuando el transistor Q2 se enciende, el transistor Q1 se apaga. El resultado es que el terminal de salida Z2 es bajo. La polaridad de la alimentación es la misma que en la figura 29-2.

El prob  
transistores y c  
problemas se p

## 29.2 DRIVERS

El L293  
de hasta 1 A p  
y cada pareja c  
de los mismos  
L293B y el enc

CHIP	
ENABLE 1	1
INPUT 1	2
OUTPUT1	3
GND	4
GND	5
OUTPUT2	6
INPUT 2	7
V <sub>S</sub>	8

nde de la  
l giro es  
ad de la

Este circuito está formado por cuatro transistores que trabajan en conmutación y se comportan como interruptores controlados por la señal que les llega a las entradas I1 e I2. Su funcionamiento es el siguiente:

- Cuando se activa la entrada I1 a nivel alto y la entrada I2 a nivel bajo los transistores Q3 y Q2 (NPN y PNP) entran en saturación simultáneamente, mientras que Q1 y Q4 están en corte por ser de signo contrario (PNP y NPN respectivamente). En estas condiciones el motor gira en un sentido, por ejemplo en el contrario a las agujas del reloj (figura 29-2).
- Cuando se invierten las señales de entrada, es decir I1 a nivel bajo e I2 a nivel alto, los transistores que se saturan son Q1 y Q4, mientras que los que entran en estado de corte son Q2 y Q3. Esto hace que el motor gire en sentido contrario (figura 29-3).

El problema de este tipo de circuitos es la caída de tensión real que hay en los transistores y que habrá que compensarla con la tensión de alimentación. Para evitar estos problemas se puede utilizar circuito integrado como el LM293B.

## 29.2 DRIVER L293B

El L293B es un driver de 4 canales capaz de proporcionar una corriente de salida de hasta 1 A por canal. Cada canal es controlado por señales de entrada compatibles TTL y cada pareja de canales dispone de una señal de habilitación que desconecta las salidas de los mismos. La figura 29-4 describe cada una de las patillas de los que dispone el L293B y el encapsulado de 16 pines.

Pin	Nombre	Descripción
1	Chip Enable 1	Habilitación de los canales 1 y 2
2	Input 1	Entrada del canal 1
3	Output 1	Salida del canal 1
4,5	GND	Masa alimentación
6	Output 2	Salida del canal 2
7	Input 2	Entrada del Canal 2
8	V <sub>s</sub>	Alimentación de las cargas
9	Chip Enable 2	Habilitación de los canales 3 y 4
10	Input 3	Entrada del canal 3
11	Output 3	Salida del canal 3
12,13	GND	Masa alimentación
14	Output 4	Salida del Canal 4
15	Input 4	Entrada del canal 4
16	V <sub>ss</sub>	Alimentación del chip

Figura 29-4 Patillaje del Driver L293B

ua de baja  
ica de unos  
iente en H,

Dispone de **una patilla** para la alimentación **de las cargas** que se están controlando ( $V_s$ ) de manera que dicha alimentación es independiente de la lógica de control.

La figura 29-5 representa el diagrama de bloques del L293B. La señal de control EN1 activa o desactiva la pareja de canales formada por los drivers 1 y 2. La señal EN2 controla la pareja de drivers 3 y 4. Las salidas OUTn se asocian con las correspondientes INn. La tabla de dicha figura detalla el funcionamiento para cada uno de los drivers.

$V_{EN1}$	$V_{INn}$	$V_{OUTn}$
H	H	H
H	L	L
L	H	Z
L	L	Z

Donde:

- H = Nivel alto "1"
- L = Nivel bajo "0"
- Z = Alta impedancia

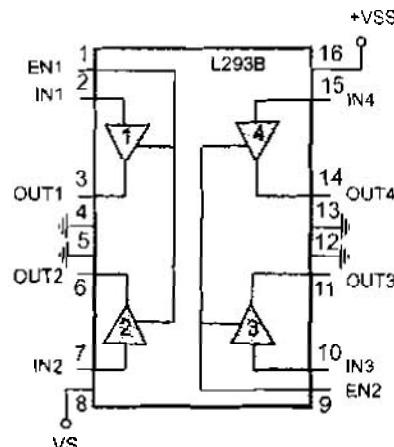


Figura 29-5 Diagrama de bloques del L293B y tabla de funcionamiento

La tabla 29-1 informa sobre los máximos valores admisibles.

Símbolo	Parámetro	Valor
$V_s$	Tensión de alimentación para las cargas	36 V
$V_{SS}$	Tensión de alimentación de la lógica	36 V
$V_i$	Tensión de entrada	7 V
$V_{inh}$	Tensión de habilitación	7 V
$I_{out}$	Intensidad de pico de salida	2 A
$P_{tot}$	Potencia total de dissipación	5 W

Tabla 29-1 Rangos absolutos del driver L293B

También se fabrica el modelo L293D, cuya principal diferencia respecto del L293B es que proporciona una corriente máxima de 600 mA.

Veamos seguidamente algunas formas de conectar los motores de corriente continua a este driver.

## 29.3 GIR

La figura continua que sigue

- El motor detiene
- El motor dañan

Figura

La Tabla

$V_{inh}$	A
H	H
H	L
L	X

Tab

Es indispensable para los motores tal como la contraelectromagnetismo y la comutación.

### 29.3 GIRO EN UN ÚNICO SENTIDO

La figura 29-6 muestra el modo de funcionamiento de dos motores de corriente continua que giran en un único sentido, suponiendo que la  $V_{inh} = 5 \text{ V}$ :

- El motor M1 se pone en marcha al poner a nivel bajo la entrada de control A y se detiene con un nivel alto de entrada.
- El motor M2 se pone en marcha al poner a nivel alto la entrada de control B y se detiene con un nivel bajo de entrada.

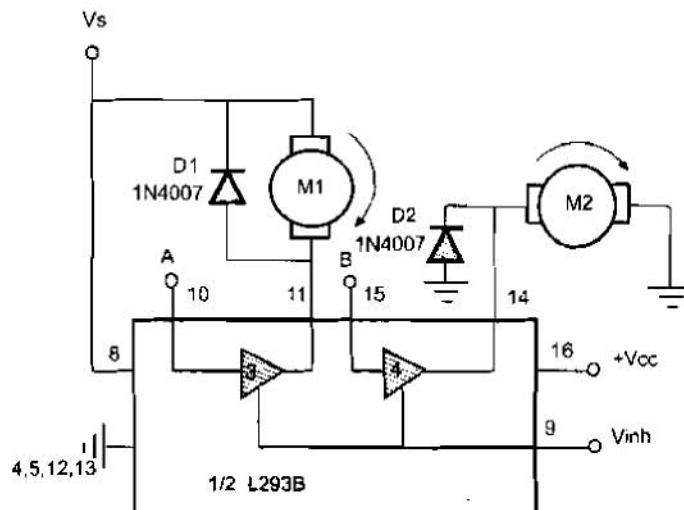


Figura 29-6 Conexión de dos motores DC. M1 activo por "0" y M2 por "1"

La Tabla 29-2 muestra el modo de funcionamiento del circuito.

$V_{inh}$	A	Motor M1	$V_{inh}$	B	Motor M2
H	H	Parada rápida del motor	H	H	Giro
H	L	Giro	H	L	Parada Rápida del motor
L	X	Motor desconectado, giro libre	L	X	Motor desconectado, giro libre

Tabla 29-2 Modo de funcionamiento del circuito de la figura 29-6

Es indispensable conectar los diodos D1 y D2 en paralelo con los devanados de los motores tal como muestra la figura, como protección frente a los picos de fuerza contraelectromotriz producidos por la carga inductiva de la bobina en el momento de la conmutación.

## 29.4 GIRO EN LOS DOS SENTIDOS

El circuito de la figura 29-7 permite controlar el **doble** sentido de giro del motor:

- Cuando la entrada C está a nivel bajo y la D a nivel alto, el motor gira en un sentido.
- Cambiando la entrada C a nivel alto y la D a nivel bajo se cambia el sentido de giro del motor al contrario.

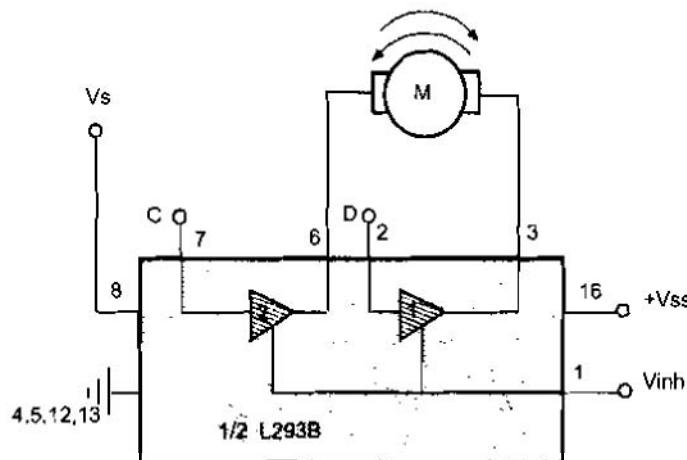


Figura 29-7 Circuito de control para el doble giro de un motor de c.c.

Los diodos de protección se pueden conectar según se muestra en la figura 29-8.

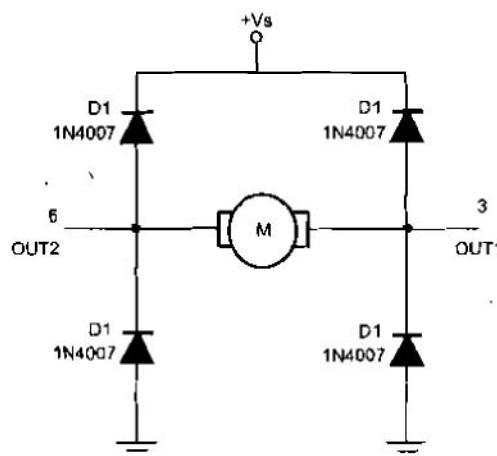
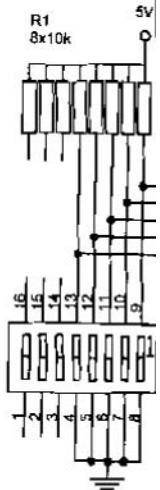


Figura 29-8 Conexión de diodos de protección al circuito de la figura 29-7

La Tabla 29-3 muestra el modo de funcionamiento del circuito.

## 29.5 CO

La figura muestra el circuito de control para el doble giro de un motor de c.c. El microcontrolador es la parte central del circuito. La alimentación es la de alimentación.



Figura

Para el motor DC de la figura 29-7. La marcha del motor depende de la posición de los giros. La tabla 29-3 muestra el modo de funcionamiento del circuito.

del motor:  
or gira en un  
. el sentido de



$V_{inh}$	C	D	Motor
H	L	L	Parada rápida del motor
H	H	H	Parada rápida del motor
H	L	H	Giro a la izquierda
H	H	L	Giro a la derecha
L	X	X	Motor desconectado, giro libre

Tabla 29-3 Modo de funcionamiento del circuito de la figura 29-7

## 29.5 CONEXIÓN DE MOTOR C.C. Y PIC16F84

La figura 29-9 muestra una conexión típica de un motor de corriente continua a un microcontrolador PIC16F84A a través de un driver L293B. La tensión aplicada al pin  $V_S$  es la de alimentación del motor, en este ejemplo se utiliza un motor de 12 V.

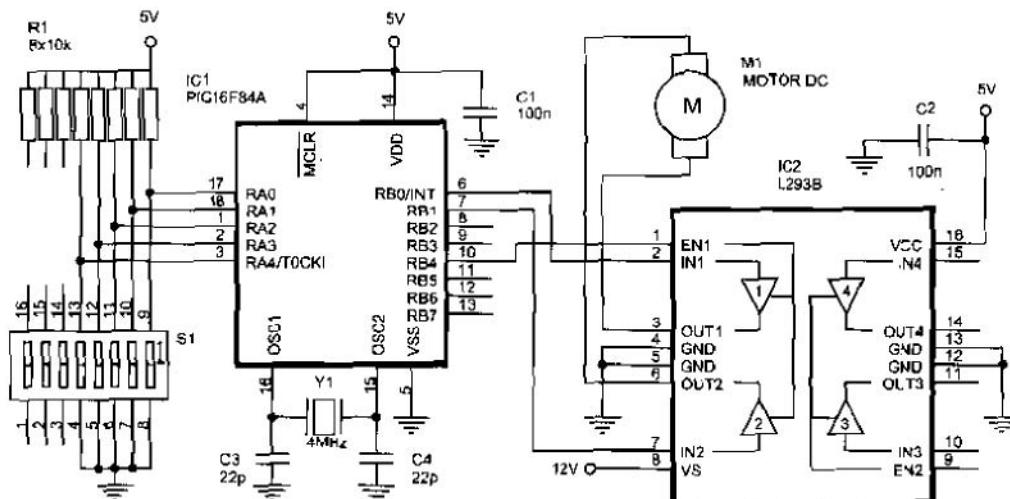


Figura 29-9 Control de motores c.c con el driver L293B y el PIC16F84A

Para comprobar su funcionamiento se puede cargar el programa MotorDC\_01.asm, que controla el driver del motor. La línea RA0 controla la puesta en marcha del motor, si RA0 es "0" el motor se pone en marcha y si RA0 es "1" se detiene. La posición del interruptor conectado a RA4 controla el sentido de giro que por supuesto depende de la polaridad de conexión del motor. Si se intercambian los terminales del motor los giros serán en sentido contrario. El organigrama de este programa se muestra en la figura 29-10.

ra 29-7

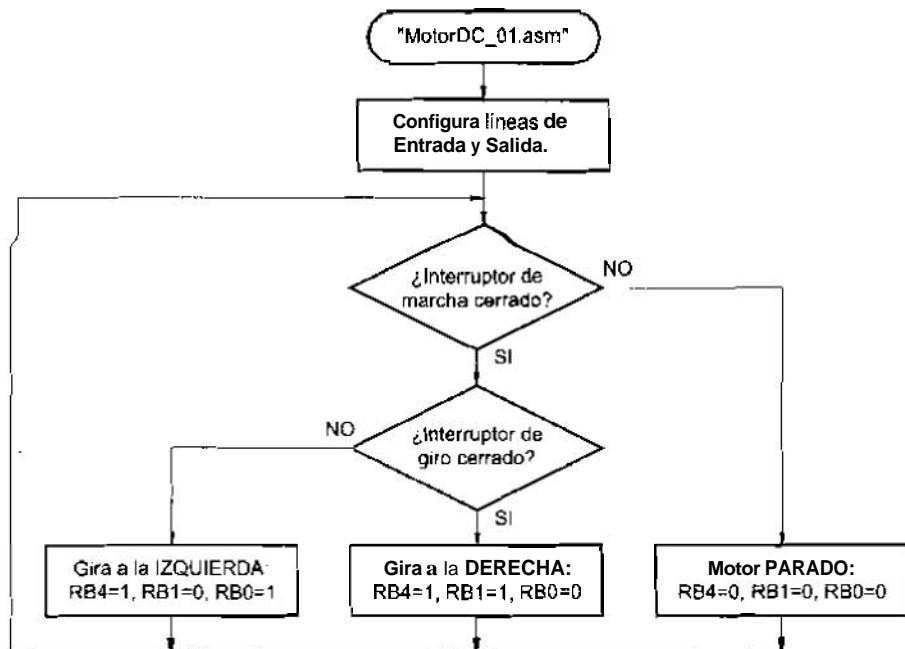


Figura 29-10 Organigrama del programa MotorDC\_01.asm

```

***** MotorDC_01.asm *****

; Programa de control para un motor de corriente continua en funcionamiento y sentido de
; giro. Con RA0=0, el motor se pone en marcha y su sentido de giro dependerá del valor
; que tenga RA4.

; ZONA DE DATOS *****

        CONFIG _CP OFF & _WDT OFF & _PWRTE ON & _XT OSC
        LIST P=16F84A
        INCLUDE <P16F84A.INC>

#define EntradaMarcha PORTA,0      ; Interruptor de puesta en marcha.
#define EntradaSentido PORTA,4      ; Interruptor de sentido de giro.

; ZONA DE CÓDIGOS *****

        ORG 0
Inicio
        bsf STATUS,RP0
        bsf EntradaMarcha          ; Configura las líneas de entrada.
        bsf EntradaSentido
        clrf PORTB
        bcf STATUS,RP0               ; Las líneas del Puerto B configuradas como salida.

Principal
        clrw
        btfs EntradaMarcha          ; Con esta combinación se detiene el motor.
        btfsc EntradaMarcha         ; Comprueba el estado del interruptor de funcionamiento.

```

```

goto    ActivaSalida
movlw  b'00010010'      ; Gira en un sentido.
btfsr  EntradaSentido   ; Comprueba el sentido de giro deseado.
movfw  b'00010001'      ; Gira en el otro sentido.

ActivaSalida
  movwf PORTB
  goto Principal

END

```

## 29.6 CONTROL DE VELOCIDAD

La velocidad de un motor de corriente continua depende del valor medio de la tensión aplicada en sus extremos.

El sistema más utilizado para controlar la velocidad de un motor DC de pequeña potencia es mediante la modulación por ancho de pulso PWM (*Pulse Width Modulation*) de una señal cuadrada TTL, como muestra la figura 29-11. Bajo el control PWM, el motor gira a una velocidad determinada por la media del nivel de la señal cuadrada.

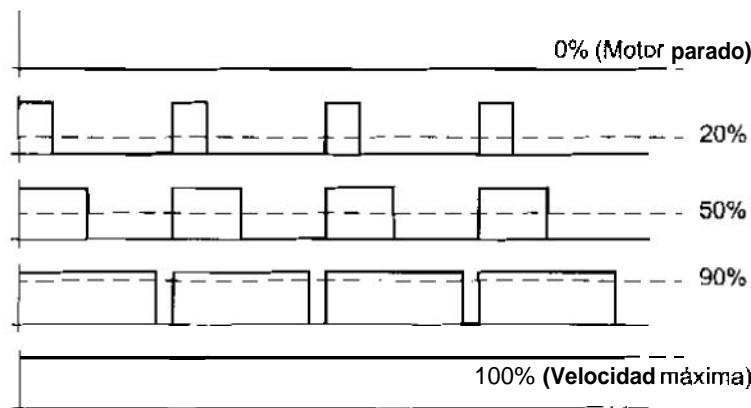


Figura 29-11 Control velocidad de un motor c.c. mediante PWM

La tensión continua media presentada al motor se controla manteniendo la frecuencia constante, y variando el tiempo que la señal permanece en alto, es decir, variando el ciclo de trabajo (*duty cycle*). Así, si el ciclo de trabajo es del 50% se suministra al motor una tensión media del 50%, con un ciclo de trabajo del 20% sólo una quinta parte de la tensión máxima es suministrada a la carga. Cambiar de un ciclo de trabajo del 50% a otro del 20% conllevará una disminución de la velocidad del motor.

La regulación PWM proporciona un eficaz método mediante la utilización de una simple señal digital de control. Si se utiliza una señal de estas características para atacar la entrada EN1 del montaje de la figura 29-9 se consigue que el valor medio de la señal de alimentación del motor varie, de tal manera que cuanto más tiempo esté la línea RB4 a

nivel alto más deprisa girará el motor. Lógicamente si la duración del impulso a nivel bajo es muy grande el motor se parará.

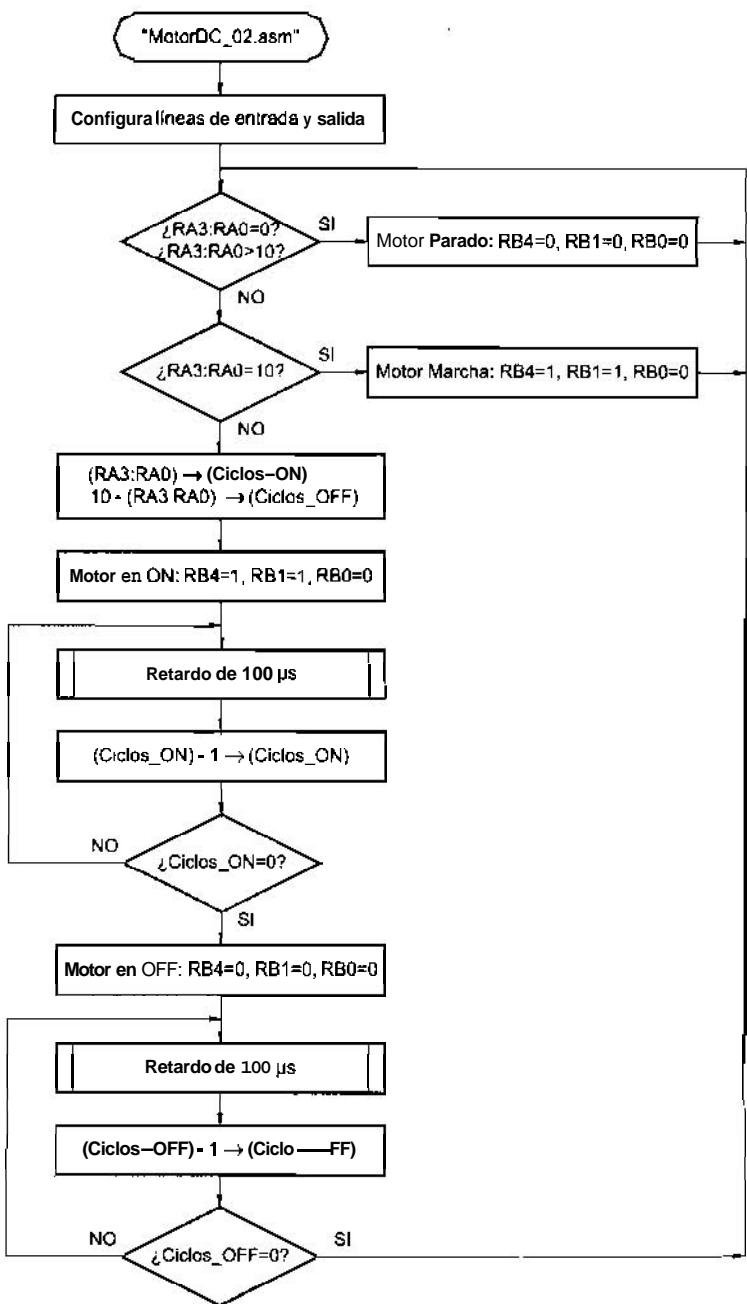


Figura 29-12 Organigrama del programa MotorDC\_02.asm

El programa MotorDC\_02.asm es un ejemplo de gobierno de la velocidad de un motor de c.c. mediante control PWM. Su funcionamiento se explica en los comentarios y en su diagrama de flujo de la figura 29-12.

\*\*\*\*\* MotorDC\_02.asm \*\*\*\*\*

; Programa de regulación de velocidad de un motor de corriente continua mediante la modulación de anchura de pulso (PWM). Por la línea de salida se genera una onda cuadrada de frecuencia constante a 100 Hz (periodo de 10 ms) y ciclo de trabajo variable desde 0% a 100%, dependiendo del valor de la entrada del Puerto A. Es decir, el tiempo en alto varía entre 0 ms (0%) y 10 ms (100%) de acuerdo con la siguiente tabla:

Entrada	DC (%)	(Ciclos_ON) SEMIPERIODO ALTO	(Ciclos_OFF) SEMIPERIODO BAJO
0	0 %	0 ms = 0 x 1 ms	10 ms = 10 x 1 ms
1	10 %	1 ms = 1 x 1 ms	9 ms = 9 x 1 ms
2	20 %	2 ms = 2 x 1 ms	8 ms = 8 x 1 ms
...	...	...	...
9	90 %	9 ms = 9 x 1 ms	1 ms = 1 x 1 ms
10	100 %	10 ms = 10 x 1 ms	0 ms = 0 x 1 ms
>10	0 %		

; ZONA DE DATOS \*\*\*\*\*

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
Ciclos_ON
Ciclos_OFF
GuardaEntrada
ENDC
```

```
MaximaEntrada EQU .10
```

; ZONA DE CÓDIGOS \*\*\*\*\*

```
ORG 0
Inicio
    bsf STATUS,RP0
    movlw b'00001111' ; RA3:RA0 como entradas.
    moywf TRISA
    clrf TRISB ; Las líneas del Puerto B se configuran como salidas.
    bcf STATUS,RP0
Principal
    movf PORTA,W ; Lee el puerto de entrada.
    andiw b'00001111'
    movwf GuardaEntrada ; Guarda el valor.
    btfsc STATUS,Z ; Si RA3:RA0=0 el motor se detiene.
    gote DC_CeroPorCiento
    sublw MaximaEntrada ; (W)=10-(PORTA)
```

			GuardaEn
			Timer0_C
			ENDC
			TMR0_Carga
			MaximaEntrada
Motor_ON			
	btfsc	STATUS,Z	
	goto	DC_100PorCiento	
	btffs	STATUS,C	; ¿C=1? ,¿(W) positivo?,¿(PORTA)<=10?
	goto	DC_CeroPorCiento	; Ha resultado (PORTA>10)
	movwf	Ciclos-OFF	; 10-(PORTA)-->(Ciclos_OFF).
	movf	GuardaEntrada,W	
	movwf	Ciclos-ON	; Carga RA3:RA0 en (Ciclos_ON).
	movlw	b'00010010'	
	movwf	PORTB	; Habilita los drivers y un sentido de giro.
	call	Retardo_1ms	
	decfsz	Ciclos_ON,F	; Si (Ciclos_ON)=0 salta a Motor-OFF.
	goto	Motor_ON+2	
			#DEFINE SalidaSe
Motor_OFF			#DEFINE SalidaSe
	clrf	PORTB	#DEFINE SalidaM
	call	Retardo_1ms	#DEFINE EntradaS
	decfsz	Ciclos_OFF,F	
	goto	Motor_OFF+1	; ZONA DE CÓDIGO
	goto	Fin	
			ORG
			goto
			ORG
			goto
DC_CeroPorCiento			
	clrf	PORTB	
	goto	Fin	; Inhabilita los drivers. Motor parado.
DC_100PorCiento			
	movlw	b'00010010'	
	movwf	PORTB	; Habilita los drivers y un sentido de giro.
Fin	goto	Principal	
			Inicio
			bsf
			movlw
			movwf
			movlw
			movwf

La generación de la onda cuadrada PWM también se puede realizar mediante interrupciones por desbordamiento del Timer 0, tal como se explicó en un ejercicio del capítulo 18. El siguiente programa es un ejemplo de ello.

\*\*\*\*\* MotorDC\_03.asm \*\*\*\*\*

, Programa de control de velocidad de un motor de corriente continua mediante la modulación de anchura de pulso (PWM) similar al MotorDC\_02.asm donde el control de tiempos se realiza mediante interrupciones por desbordamiento del Timer 0.

El sentido de giro del motor se decide en función del valor de la línea RA4.

; El control de las líneas de salida se realizará mediante direccionamiento por bit con las instrucciones "bsf" y "bcf".

## ZONA DE DATOS \*\*\*\*\*

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC  
LIST    P=16F84A  
INCLUDE <P16F84A.INC>
```

**CBLOCK 0x0C** : Ciclo de trabajo deseado.

```

GuardaEntrada
Timer0_ContadorA
ENDC

TMR0_Carga EQU -d'245' ; Valor obtenido experimentalmente con la ventana
                           ; Stopwatch para un tiempo de 1 ms.

MaximaEntrada EQU .10

#DEFINE SalidaSentido0 PORTB,0 ; Salidas que determinan el sentido de giro.
#DEFINE SalidaSentido1 PORTB,1
#DEFINE SalidaMarcha PORTB,4 ; Salida de puesta en marcha o paro del motor.
#DEFINE EntradaSentido PORTA,4 ; Interruptor de sentido de giro.

```

; ZONA DE CÓDigos \*\*\*\*\*

```

ORG 0
goto inicio
ORG .4
goto Timer0_Interrupcion

Inicio
  bsf STATUS,RP0
  bcf SalidaMarcha
  bcf SalidaSentido0
  bcf SalidaSentido1
  movlw b'00011111'
  movwf PORTA
  movlw b'00000001'
  movwf OPTION_REG
  bcf STATUS,RP0
  bcf SalidaMarcha ; Estas líneas se configuran como salida.

Principal
  btfsc EntradaSentido ; Puerto A configurado como entrada.
  goto OtroSentido
  bsf SalidaSentido0 ; Puerto A configurado como salida.
  bcf SalidaSentido1 ; Puerto A configurado como salida.
  goto TesteaVelocidad ; Gira en un sentido.

OtroSentido
  bcf SalidaSentido0 ; Al principio el motor parado.
  bcf SalidaSentido1 ; Comprueba el sentido de giro deseado.

TesteaVelocidad
  movf PORTA,W ; Gira en el sentido opuesto.
  andlw b'00001111'
  movwf GuardaEntrada ; Gira en el sentido opuesto.
  btfsc STATUS,Z ; Lee el puerto de entrada
  goto DC_CeroPorCiento ; Guarda el valor.
  sublw MaximaEntrada ; Lee el puerto de entrada
  btfsc STATUS,Z ; Guarda el valor.
  goto DC_100PorCiento ; Lee el puerto de entrada
  btfss STATUS,C ; Guarda el valor.
  goto DC_CeroPorCiento ; Lee el puerto de entrada
  movf GuardaEntrada,W ; Guarda el valor.
  movwf CicloTrabajo ; Lee el puerto de entrada
  movlw b'10100001' ; Lee el puerto de entrada

```

```

        movwf INTCON           ; Autoriza interrupción T0I y la general (GIE).
        goto Fin

DC_CeroPorCiento
        bcf SalidaMarcha      ; Pone la salida siempre en bajo.
        goto InhabilitaInterrupcion

DC_100PorCiento
        bsf SalidaMarcha      ; Pone la salida siempre en alto.
InhabilitaInterrupcion
        clrf INTCON            ; Inhabilita interrupciones.
Fin      goto Principal

```

#### Subrutina "Timer0-Interrupcion"

; Mantiene la salida en alto un tiempo igual a 1 ms x (CicloTrabajo)  
; y en bajo un tiempo igual a 1 ms x (10-CicloTrabajo).

```

CBLOCK
Guarda_W
Guarda_STATUS
ENDC

```

#### Timer0\_Interrupcion

```

        movwf Guarda_W          ; Guarda los valores de tenian W y STATUS en el
        swapf STATUS,W           ; programa principal.

        movwf Guarda_STATUS       ; Garantiza que trabaja en el Banco 0.

        bcf STATUS,RP0
        movlw TMR0_Carga
        movwf TMR0
        decfsz Timer0_ContadorA,F ; Decrementa el contador.

        goto Fin_Timer0_Interrupcion
        btfsc SalidaMarcha      ; Testea el anterior estado de la salida.

        goto EstabaAlto
EstabaBajo
        bsf SalidaMarcha          ; Estaba bajo y lo pasa a alto.
        movf CicloTrabajo,W       ; Repone el contador nuevamente con el tiempo en
        movwf Timer0_ContadorA     ; alto.

        goto Fin_Timer0_Interrupcion
EstabaAlto
        bcf SalidaMarcha          ; Estaba alto y lo pasa a bajo.
        movf CicloTrabajo,W       ; Repone el contador nuevamente con el tiempo
        sublw .10                  ; en bajo.

        movwf Timer0_ContadorA
        swapf Guarda_STATUS,W     ; Restaura registros W y STATUS.

        swapf STATUS
        swapf Guarda_W,F
        swapf Guarda_W,W
        bcf INTCON,RBIF
        bcf INTCON,T0IF
        retfie
END

```

No es r...  
conseguir un m...  
de un proyecto.

## 30.1 MOTORES

Los moto...  
dispositivos co...  
papel de una ir...  
disquetas de e...  
observa que en...  
por el eje del m...

Los mot...  
comente contin...  
hacerlo contin...  
microcontrolad...  
dentro de una ...  
control de pos...  
revoluciones pe...  
y carga mecán...  
por métodos ser...

Un moto...  
devanados. El ...  
Cada pulso pro...

Volumen  
0531

## CAPÍTULO 30

# MOTORES PASO A PASO

No es raro que un aficionado a la electrónica desguace una vieja impresora para conseguir un motor paso a paso y poder realizar alguna tarea de posicionamiento dentro de un proyecto. En este capítulo se explica cómo trabajar con estos útiles motores.

### 30.1 MOTORES PASO A PASO (PAP)

Los motores paso a paso o PAP (*Stepper Motor*) son muy utilizados en los dispositivos controlados por sistemas digitales. Por ejemplo los mecanismos que arrastran papel de una impresora, los que mueven el brazo de un robot o los que hacen girar las disqueteras de un ordenador dependen de motores PAP para su funcionamiento. Se observa que en estas situaciones se requiere un control preciso de la trayectoria a seguir por el eje del motor.

Los motores PAP proporcionan una considerable ventaja sobre los motores de corriente continua o DC. El eje de un motor PAP gira a intervalos regulares en lugar de hacerlo continuamente, como ocurre con los motores de continua. Bajo el control de un microcontrolador, los motores PAP pueden ser usados para posicionamientos precisos dentro de una amplia gama de aplicaciones, incluyendo robótica, automatización y control de posicionamiento. La velocidad de un motor de DC viene expresada en revoluciones por minuto (rpm) y es función de la tensión aplicada, corriente por el motor y carga mecánica del mismo. Un posicionamiento preciso de un motor DC no es posible por métodos sencillos.

Un motor PAP gira en función de una secuencia de pulsos aplicados a sus devanados. El eje del motor gira un determinado ángulo por cada impulso de entrada. Cada pulso provoca la rotación del rotor del motor en un incremento de ángulo preciso,

denominado **paso**, de ahí **el nombre** de motor "paso a paso". El resultado de este **movimiento**, fijo y repetible, es un **posicionamiento** preciso y fiable. Los **incrementos** de pasos de la rotación del rotor se traducen en **un alto grado** de control de posicionamiento.

Los incrementos de rotación o pasos se miden **en grados** y es el parámetro fundamental de **un motor PAP**. También se puede expresar en **números de pasos** por revolución de 360 grados. Un motor paso a paso puede girar **un número exacto** de grados en ambos sentidos.

Los motores PAP se comercializan dentro de una gran variedad de grados de rotación por paso, desde  $0,72^\circ$  a  $22,5^\circ$ , correspondientes a 500 y **16 pasos por revolución**, respectivamente (efectivamente,  $360^\circ/0,72^\circ = 500$  y  $360^\circ/22,5^\circ = 16$ ). El **motor PAP más comercializado es el de  $7,5^\circ$  por paso o 48 pasos por revolución**.

El **principal problema** que presentan los motores PAP es su limitada potencia. Sin embargo, este **problema está siendo resuelto** por los **nuevos diseños**, con los que se ha logrado potencias superiores a 1 CV.

**En este capítulo** se va a explicar **como** realizar el control de motores PAP mediante un microcontrolador PIC16F84, pero antes hay **que** examinar sus **principios de funcionamiento**.

## 30.2 PRINCIPIO DE FUNCIONAMIENTO

Aún basados en el mismo fenómeno que el principio de funcionamiento de los **motores de corriente continua**, los motores paso a paso son **mucho más sencillos**, si cabe, que cualquier otro tipo de motor eléctrico.

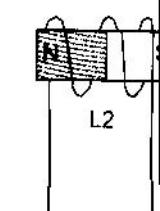
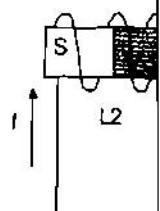
La figura 30-1 ilustra el modo de funcionamiento de un motor PAP. Suponemos que las bobinas L1 como L2 poseen un núcleo de hierro dulce capaz de imantarse cuando dichas bobinas sean recorridas por una corriente eléctrica, esto es **el denominado estator**. Por otra parte, el imán M puede girar libremente sobre el eje de sujeción central, este es el **rotor**.

Inicialmente, sin aplicar ninguna corriente a las bobinas (que también reciben el nombre de fases) y **con el imán M en una posición cualquiera**, el imán permanecerá en reposo si no se somete a una fuerza externa.

Si se hace circular corriente por ambas **fases**, como se muestra en la figura 30-1(a), se crearán dos **polos magnéticos NORTE** en la parte **interna**, bajo cuya influencia el imán M se desplazará hasta la posición indicada en dicha figura.

Si invertimos la polaridad de la corriente que circula por la bobina L1 se obtendrá la situación magnética indicada en la figura 30-1(b) y el imán M se verá desplazado hasta

la nueva posición de las agujas del reloj.



Invirtiendo la situación de las bobinas, el imán M se habrá desplazado 90 grados.

Si se invierten las corrientes en las bobinas, las posiciones de los polos magnéticos se intercambiarán.

la nueva posición de equilibrio. Es decir, ha girado 90 grados en sentido contrario a las agujas del reloj.

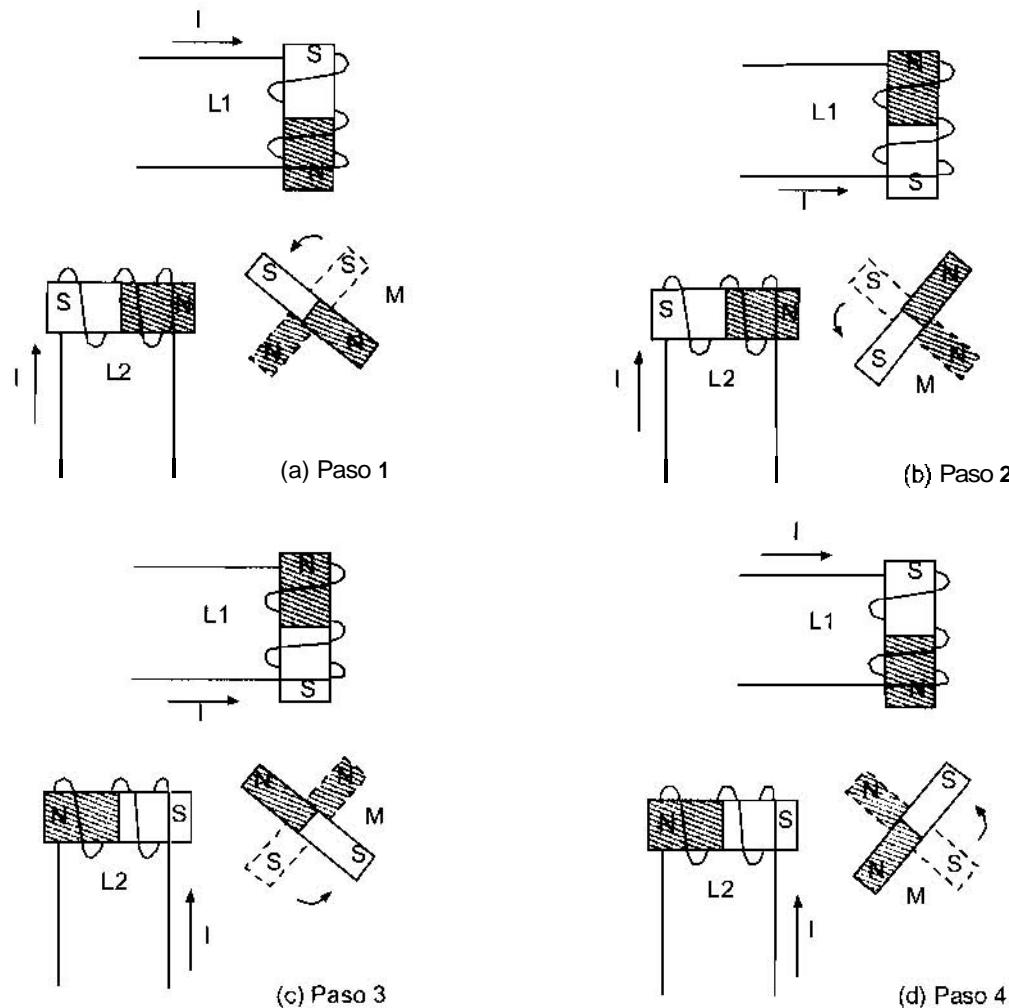


Figura 30-1 Principio de funcionamiento de un motor paso a paso

Invirtiendo ahora la polaridad de la corriente que atraviesa la bobina L2, se llega a la situación de la figura 30-1 (c) habiendo girado el imán M otros 90 grados. Si invertimos de nuevo el sentido de la corriente en la bobina L1 el imán M girara otros 90 grados y se habrá obtenido una revolución completa de dicho imán en cuatro pasos de 90 grados.

Si se mantiene la secuencia de excitación expuesta para L1 y L2 y dichas corrientes son aplicadas en forma de pulsos, el rotor avanzará pasos de 90 grados por cada pulso aplicado.

Analizando lo expuesto podemos decir que un motor paso a paso es un dispositivo electromecánico que convierte impulsos eléctricos en un movimiento rotacional constante y finito dependiendo de las características propias del motor.

El modelo de motor paso a paso que hemos analizado recibe el nombre de bipolar ya que para obtener la secuencia completa, se requiere disponer de corrientes de dos polaridades, presentando tal circunstancia un inconveniente importante a la hora de diseñar el circuito que controle el motor.

### 30.3 MOTORES PAP BIPOLARES

En este tipo de motores las bobinas del estator se conectan formando dos grupos, tal y como se muestra en la figura 30-2.

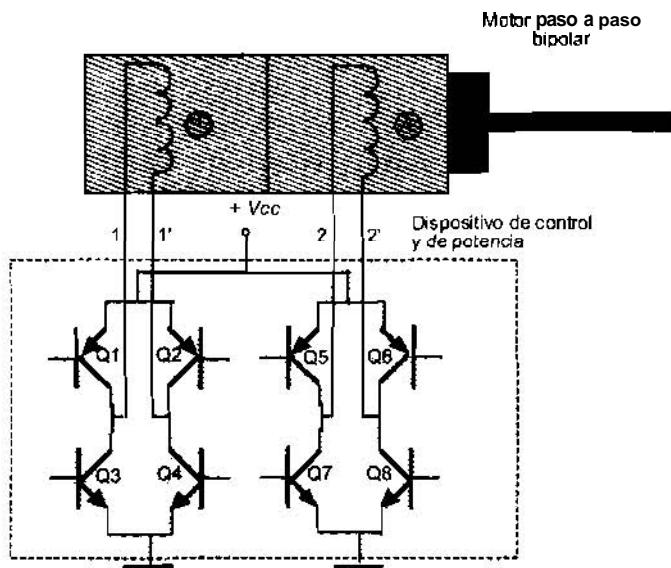


Figura 30-2 Circuito de Control de un motor bipolar

De este motor salen cuatro hilos conectados al circuito de control, que realiza la función de cuatro interruptores electrónicos dobles que permiten variar la polaridad de la alimentación de las bobinas. Con la activación y desactivación adecuada de dichos interruptores dobles, se puede obtener las secuencias correctas para que el motor gire en un sentido o en otro. Estos transistores configurados como puente en H pueden ser sustituidos por el driver L293B tal como se estudió en el capítulo anterior, la conexión de un motor PAP bipolar y el driver L293B se muestra en la figura 30-13. En la figura 30-3, puede analizarse como debe ser la secuencia de excitación para que el motor gire en un sentido.

Tiene dos modos de funcionamiento: *Full Step* y *Half Step*.

30.3.1 N

En el  
excitación.

Term

Term

Term

Term

Fig

La ta  
horario CW  
términos "he  
mira del obs  
forma de ind

Paso
1
2
3
4

Tabla 3

30.3.2 M

En el  
de excitaci

1 dispositivo  
al constante

e de bipolar  
antes de dos  
la hora de

dos grupos,

que realiza la  
polaridad de la  
ada de dichos  
motor gire en  
pueden ser  
a conexión de  
a figura 30-3,  
cor gire en un

### 30.3.1 Motor PAP bipolar en modo Full Step

En el modo **Full Step** el rotor del motor PAP avanza un paso por cada pulso de excitación.

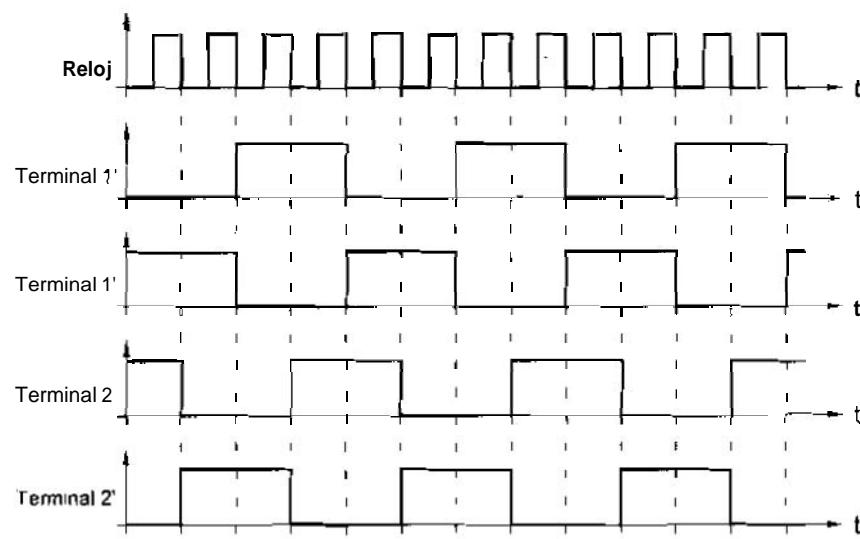


Figura 30-3 Secuencia de la señal de excitación del motor PAP bipolar

La tabla 30-1 muestra las secuencias para generar los movimientos en sentido horario CW (Clockwise) y antihorario CCW (Counterclockwise) de un motor bipolar. Los términos "horario" y "antihorario" se deben relativizar ya que dependen del punto de mira del observador y de la posición del motor, por tanto, solo se deben tomar como una forma de indicar que gira en un sentido o en el contrario.

Paso	Q2Q3	Q1Q4	Q6Q7	Q5Q8	FULL STEP ↓ CW ↑ CCW	L1	L1'	L2	L2'
1	ON	OFF	OFF	ON		-	+	+	-
2	ON	OFF	ON	OFF		-	+	-	+
3	OFF	ON	ON	OFF		+	-	-	+
4	OFF	ON	OFF	ON		+	-	+	-

Tabla 30-1 Secuencia de control de un motor PAP bipolar para modo Full Step

### 30.3.2 Motor PAP bipolar en modo Half Step

En el modo de medio paso (**Half Step**) el rotor avanza medio paso por cada pulso de excitación. Presenta como principal ventaja una mayor resolución de paso, ya que

disminuye el **avance angular** (la mitad que en el modo de paso completo). Para conseguir tal cometido, el modo de excitación consiste en hacerlo alternativamente sobre dos bobinas y sobre una sola de ellas, según se muestra en la tabla 30-2 para ambos sentidos de giro.

Según la figura 30-1 al excitar dos bobina consecutivas del estator simultáneamente, el rotor se alinea con la bisectriz de ambos campos magnéticos. Cuando desaparece la excitación de una de ellas, extinguéndose el campo magnético inducido por dicha bobina, el rotor queda bajo la acción del único Campo existente, dando lugar a un desplazamiento mitad.

Paso	Q1Q3	Q1Q4	Q6Q7	Q5Q8	
1	ON	OFF	OFF	ON	
2	ON	OFF	OFF	OFF	
3	ON	OFF	ON	OFF	
4	OFF	OFF	ON	OFF	
5	OFF	ON	ON	OFF	
6	OFF	ON	OFF	OFF	
7	OFF	ON	OFF	ON	
8	OFF	OFF	OFF	ON	

**HALF STEP**  
 ↓              ↑  
 CW            CCW

Tabla 30-2 Secuencia de control de un motor PAP bipolar para modo Half Step

En este c  
control del senti  
tabla 3-4 para m

### 30.4 MOTORES PAP UNIPOLARES

Una forma de paliar el inconveniente que supone la necesidad de dos polaridades de la corriente para generar la secuencia del motor, si este dispone de una toma media entre las bobinas, es realizar el montaje que se representa en la figura 30-4. Se obtiene un **motor PAP unipolar** de cuatro fases (o bobinas), donde la corriente circula por las bobinas en un único sentido.

En los motores PAP unipolares, todas las bobinas del estator están conectadas formando cuatro grupos. Estos, a su vez, se conectan dos a dos y se montan sobre dos estatores diferentes, tal como se aprecia en la figura 30-4. Del motor paso a paso salen dos grupos de tres cables, uno de los cuales es común a dos bobinados. Los seis terminales que parten del motor se conectan al circuito de control, el cual, se compone de cuatro conmutadores electrónicos que al ser activados o desactivados producen la alimentación de los cuatro grupos de bobinas. Generando una secuencia adecuada de funcionamiento de los transistores que trabajan como interruptores, se pueden producir saltos de un paso en el número y sentido deseado.

Tabla 30-3 S

Tabla 30-4 S

Su princip  
inicialmente se ap

conseguir  
sobre dos  
sentidos

el estator  
agnéticos.  
magnético  
nte, dando

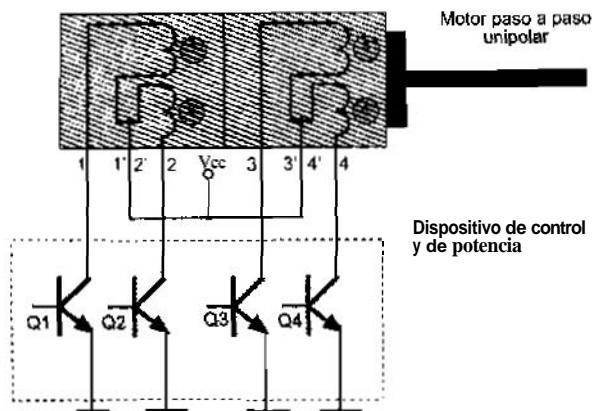


Figura 30-4 Control de motor PAP unipolar

En este caso la tabla de la secuencia que debe introducirse en las bobinas para el control del sentido de giro es la mostrada en la tabla 30-3 para modo Full Step y en la tabla 30-4 para modo Half Step.

Paso	Q1	Q2	Q3	Q4	
1	ON	OFF	OFF	ON	
2	ON	OFF	ON	OFF	
3	OFF	ON	ON	OFF	
4	OFF	ON	OFF	ON	

**FULL STEP**  
 ↓ ↑  
 CW CCW

Tabla 30-3 Secuencia de control de un motor PAP Unipolar para modo Full Step

Paso	Q1	Q2	Q3	Q4	
1	ON	OFF	OFF	ON	
2	ON	OFF	OFF	OFF	
3	ON	OFF	ON	OFF	
4	OFF	OFF	ON	OFF	
5	OFF	ON	ON	OFF	
6	OFF	ON	OFF	OFF	
7	OFF	ON	OFF	ON	
8	OFF	OFF	OFF	ON	

**HALF STEP**  
 ↓ ↑  
 CW CCW

Tabla 30-4 Secuencia de control de un motor PAP Unipolar para modo Half Step

Su principio de funcionamiento se representa gráficamente en la figura 30-5. Si inicialmente se aplica la corriente a las bobinas L1 y L3 cerrando los interruptores S1 y

S3, se generarán dos polos NORTE que atraerán al polo SUR del imán M hasta encontrar la posición de equilibrio entre ambos como puede verse en la figura 30-5(a). Si se abre posteriormente el interruptor S1 y se cierra S2, por la nueva distribución de polos magnéticos, M evoluciona hasta la situación representada en la figura 30-5(b)

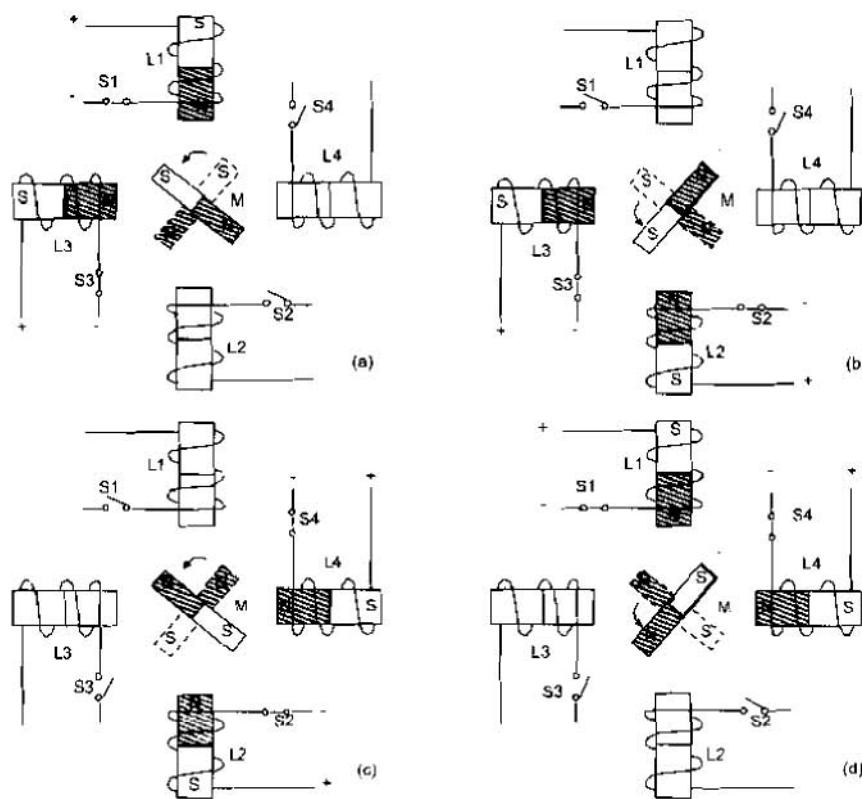


Figura 30-5 Principio básico de un motor.unipolar de cuatro fases

Siguiendo la secuencia representada en la figuras 30-5(c) y 30-5(d) de la misma forma se obtienen avances del rotor dc 90 grados habiendo conseguido, como en el motor bipolar de dos fases, hacer que el rotor avance pasos dc 90 grados por la acción de los impulsos eléctricos de excitación de cada una de las bobinas. El movimiento obtenido ha sido en sentido contrario al de las agujas del reloj. Ahora bien, si las secuencias de excitación se generan en orden inverso, el rotor girará en sentido contrario. Podemos deducir que el sentido de giro en los motores paso a paso es reversible en función de la secuencia de excitación y, por tanto, se puede hacer avanzar o retroceder al motor un número determinado de pasos según las necesidades.

La función de los interruptores sera ejecutada por los transistores NPN de la figura 30-4 que se incluyen dentro del driver L293B estudiado en el capítulo anterior, la conexión de un motor PAP unipolar y el driver L293B se muestra en la figura 30-14.

## 30.5 CON:

El modelo mayor atractivo angulares.

Una forma el número de bobinas y a pérdidas muy Hasta ahora para núcleos de 11 b micropolos magnéticos con motores de hasta resumida esta disp

### 30.5 CONSTITUCIÓN INTERNA DE UN MOTOR PAP

El modelo de motor paso a paso estudiado, salvo por su valor didáctico, no ofrece mayor atractivo desde el punto de vista práctico debido a la amplitud da sus avances angulares.

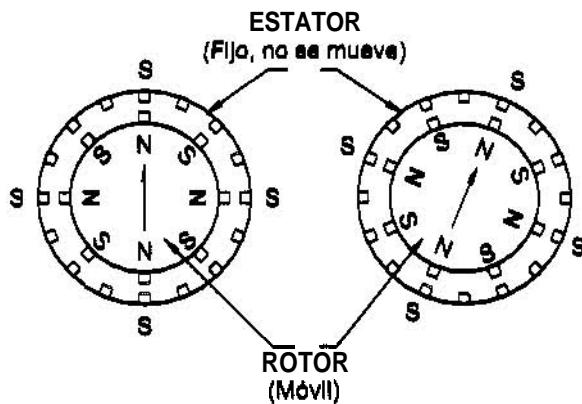


Figura 30-6 Aumento de resolución en un motor PAP

Una forma de conseguir motores PAP de paso más reducido consiste en aumentar el número de bobinas del estator, pero ello llevaría a un aumento del coste y del volumen y a pérdidas muy considerables en el rendimiento del motor, por lo que no es viable. Hasta ahora para conseguir la solución más idónea se recurre a la mecanización de los núcleos de las bobinas y el rotor en forma de hendiduras o dientes, creándose así micropolos magnéticos, tantos como dientes y estableciendo las situaciones de equilibrio magnéticos con avances angulares mucho menores. De esta forma es posible conseguir motores de hasta de 500 pasos e incluso más. En la figura 30-6 se muestra de forma resumida esta disposición para un motor PAP de 22,5°.

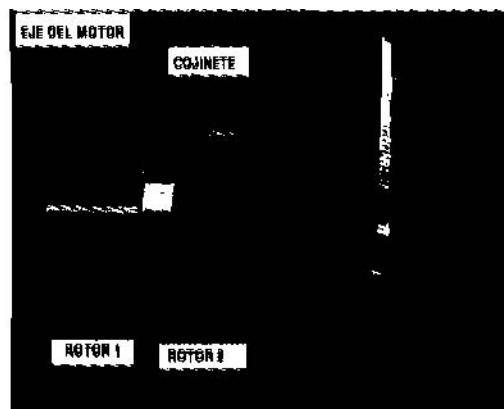


Figura 30-7 Rotor de un motor paso a paso

En las fotografías de las figuras 30-7 y 30-8 pueden apreciarse un estator y un rotor de un motor bipolar de 48 pasos desguazado.

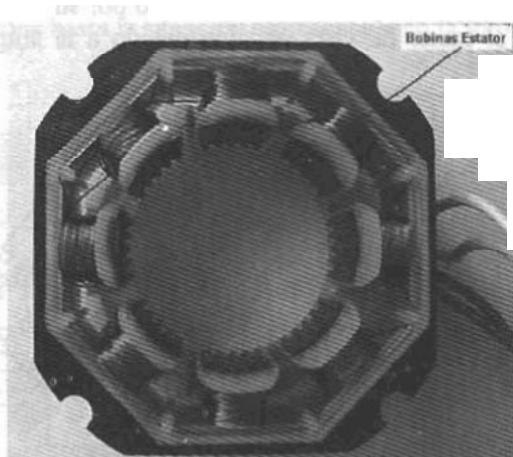


Figura 30-8 Estotor del motor paso a paso

## 30.6 DISPOSICIÓN DE LAS BOBINAS

La existencia de varios bobinados en el estotor de los motores paso a paso da lugar a varias formas de agrupar dichos bobinados para que sean alimentados adecuadamente. (figura 30-9)

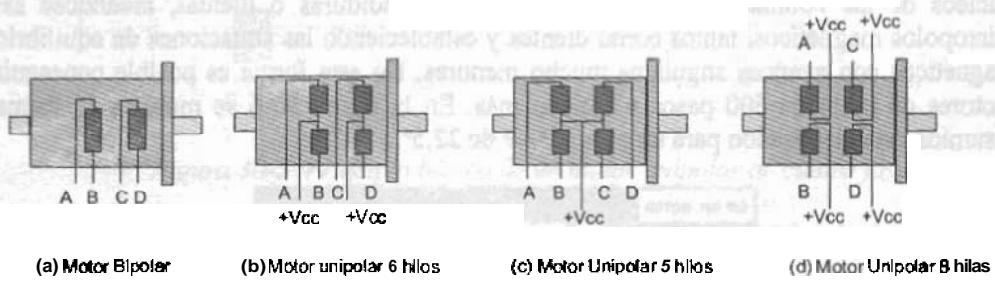


Figura 30-9 Disposición de las bobinas de motores paso a paso

En el caso de los motores paso a paso unipolares se pueden encontrar con cinco, seis u ocho terminales ya que además de los bobinados hay otros terminales que corresponden con las tomas intermedias de las bobinas, los cuales se conectan directamente a positivo de la fuente dc alimentación para su correcto funcionamiento. Las figuras 30-9(b), 30-9(c) y 30-9(d) describen como están conectados internamente los terminales de estos tipos de motores.

Hay que tener en cuenta que los motores PAP unipolares de seis u ocho hilos, pueden hacerse funcionar como motores paso a paso bipolares si no se utilizan las tomas

centrales, mient  
en el interior est

## 30.7 PAR

Desde el de algunas de las

Par din  
dinámica  
perder p  
estotor  
velocida  
generada  
bobinado

- Ángulo o  
motor po  
pasos est

| Gra

T

- Par de m  
régimen d  
mayor qu  
posición e  
Número d  
para reali

donde NP

- Frecuenc  
número d  
adecuadame

centrales, mientras que las de cinco hilos no podrán usarse jamás como bipolares porque en el interior están conectados los dos cables centrales.

### 30.7 PARÁMETRO DE LOS MOTORES PAP

Desde el punto de vista mecánico y eléctrico es conveniente conocer el significado de algunas de las principales características y parámetros que definen un motor PAP:

- **Par dinámico de trabajo (Working Torque).** Depende de sus características dinámicas y es el momento máximo que el motor es capaz de desarrollar sin perder paso, es decir, sin dejar de responder a algún impulso de excitación del estator y dependiendo de la carga. Hay que tener en cuenta que cuando la velocidad de giro del motor aumenta, se produce un aumento de la f.c.e.m. en él generada y, por tanto, una disminución de la corriente absorbida por los bobinados del estator. Como consecuencia de todo ello, disminuye el par motor.
- **Ángulo de paso (Step Angle  $\alpha$ ).** Se define como el avance angular producido en el motor por cada impulso de excitación. Se mide en grados, siendo el número de pasos estándar más empleados los mostrados en la tabla 30-5.

Grados por impulso de excitación	Nº de pasos por vuelta
0,72°	500
1,80°	200
3,75°	96
7,50°	48
15,00°	24

Tabla 30-5 Ángulos de paso más comunes en los motores PAP

- **Par de mantenimiento (Holding Torque).** Es el par requerido para desviar, en régimen de excitación, un paso el rotor cuando la posición anterior es estable. Es mayor que el par dinámico y actúa como freno para mantener el rotor en una posición estable dada.
- **Número de pasos por vuelta.** Es la cantidad de pasos que ha de efectuar el rotor para realizar una revolución completa. Evidentemente su ecuación es:

$$NP = \frac{360}{\alpha}$$

donde NP es el número de pasos y  $\alpha$  el ángulo de paso.

- **Frecuencia de paso máximo (Maximum pull-in/out).** Se define como el máximo número de pasos por segundo que puede ejecutar el motor funcionando adecuadamente.

## 30.8 CONTROL DE LOS MOTORES PASO A PASO

Para realizar el control de los motores paso a paso es necesario, como hemos visto, generar una secuencia determinada de impulsos. Además, estos impulsos deben ser capaces de entregar la corriente necesaria para que las bobinas del motor se exciten. La figura 30-10 muestra el diagrama de bloques general de un sistema de control para un motor PAP.

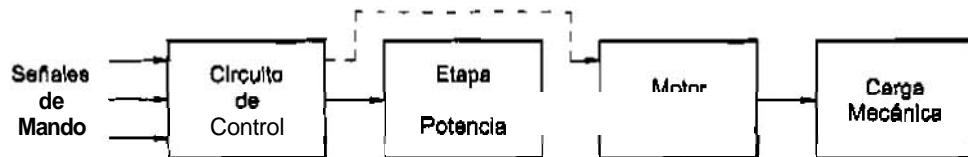


Figura 30-10 Diagrama de bloques de un sistema de control para motor paso a paso

Vamos a centrarnos en el control de los motores paso a paso utilizando el microcontrolador PIC16F84A. Como el microcontrolador no es capaz de generar la corriente suficiente para excitar las bobinas del motor PAP utilizaremos los drivers contenidos en el integrado L293B (figuras 30-13 y 30-14)

**EJEMPLO:** El circuito de control que gobierna un motor PAP de  $7,5^\circ$  por paso, produce el avance de 11 pasos del motor a una frecuencia de 200 Hz, ¿cuántos grados gira? si se desprecia la inercia y los rozamientos mecánicos, ¿cuánto tiempo tarda en realizar dicho movimiento?

**Solución:** Como el motor gira  $7,5^\circ$  por pulso, al terminar de aplicar los 11 pulsos, el eje del motor habrá girado:  $7,5 \times 11 = 82,5^\circ$ . El periodo de la señal es de 5 ms, por tanto, el motor tardará 55 ms en realizar este movimiento.

## 30.9 IDENTIFICACIÓN DE UN MOTOR PAP

Para la realización de los ejercicios que vamos a plantear, se van a utilizar dos motores paso a paso, uno bipolar y otro unipolar, recuperados de máquinas de desguace, como discos duros, impresoras o similar.

La primera dificultad es la identificación de las bobinas internas con los terminales del motor. Es conveniente tener en cuenta el número de hilos que dispone el motor PAP, para intentar identificar su estructura interna con alguno de los modelos descritos en la figura 30-9.

Generalmente se deduce por el color de los cables pero, en caso de duda, es conveniente medir la resistencia de la bobina con un ohmímetro. Así, por ejemplo, para un motor bipolar que tiene cuatro hilos debe utilizarse un polímetro en posición de medida

de resistencia hilos que miden de ohmios. Esto a la parej

En este A-B (1-1') o a cables al circuito sentido antihorario de la bobina C

Para lo figura 30-4, pr

Se mide cualquiera dist. bobinas y los central se mide medida de 150 mitad corresponde 2' y 3'- 4'.

SO

no hemos visto,  
usos deben ser  
se exciten. La  
control para un

Carga  
Mecánica

paso a paso

utilizando el  
de generar la  
nos los drivers

e 7,5° por paso,  
cuántos grados  
tiempo tarda en

ar los 11 pulsos,  
es de 5 ms, por

n a utilizar dos  
as de desguace,

n los terminales  
el motor PAB,  
descritos en la

so de duda, es  
ejemplo, para un  
ción de medida

de resistencias para detectar las dos bobinas independientes. Para ello, hay que buscar dos hilos que midan un valor cualquiera que no sea infinito, generalmente unas pocas decenas de ohmios. Estos dos hilos pertenecen a los terminales de una de las bobinas y los otros dos a la pareja opuesta.

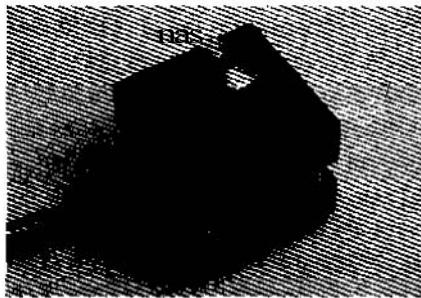


Figura 30-11 Motor paso a paso de 4 hilos

En este caso no es importante conocer el devanado que corresponde con la bobina A-B (1-1') o a la C-D (2-2'), ni identificar sus terminales, porque una vez conectados los cables al circuito de control si el motor gira en sentido horario y queremos que gire en sentido antihorario, sólo tendremos que cambiar las conexiones de la bobina A-B por los de la bobina C-D.

Para los motores de 6 hilos si queremos identificarlos con los del motor de la figura 30-4, procederemos de la siguiente manera:

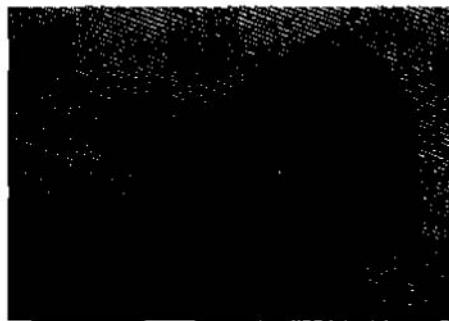


Figura 30-12 Fofó de un motor paso a paso de 6 hilos

Se mide con el ohmetro para buscar los tres hilos que entre sí midan un valor cualquiera distinto de infinito. Estos tres hilos pertenecerán a uno de los juegos de bobinas y los otros tres pertenecerán al otro juego de bobinas. Para averiguar el terminal central se mide la resistencia entre dos cables, obteniendo para nuestro caso concreto la medida de  $150 \Omega$ , midiendo los otros dos resulta  $300 \Omega$ , por lo tanto, el que tiene el valor mitad corresponde con la toma central de la bobina, numerados en la figura 304 con 1'-2' y 3'-4'.

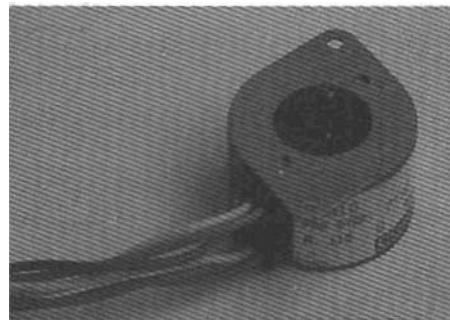
de resistencias para detectar las dos bobinas independientes. Para ello, hay que buscar **dos hilos que midan un valor** cualquiera que no sea infinito, generalmente unas pocas decenas de ohmios. Estos dos hitos pertenecen a los terminales de una de las bobinas y los otros dos a la pareja opuesta.



*Figura 30-11 Motor paso a paso de 4 hilos*

En este caso no es importante conocer el devanado que corresponde con la bobina A-B (1-1') o a la C-D (2-2'), ni identificar sus terminales, porque una vez conectados los cables al circuito de control si el motor gira en sentido horario y queremos que gire en sentido antihorario, sólo tendremos que cambiar las conexiones de la bobina A-B por los de la bobina C-D.

Para los motores de 6 hilos si queremos identificarlos con los del motor de la figura 30-4, procederemos de la siguiente manera:



*Figura 30-12 Foto de un motor paso a paso de 6 hilos*

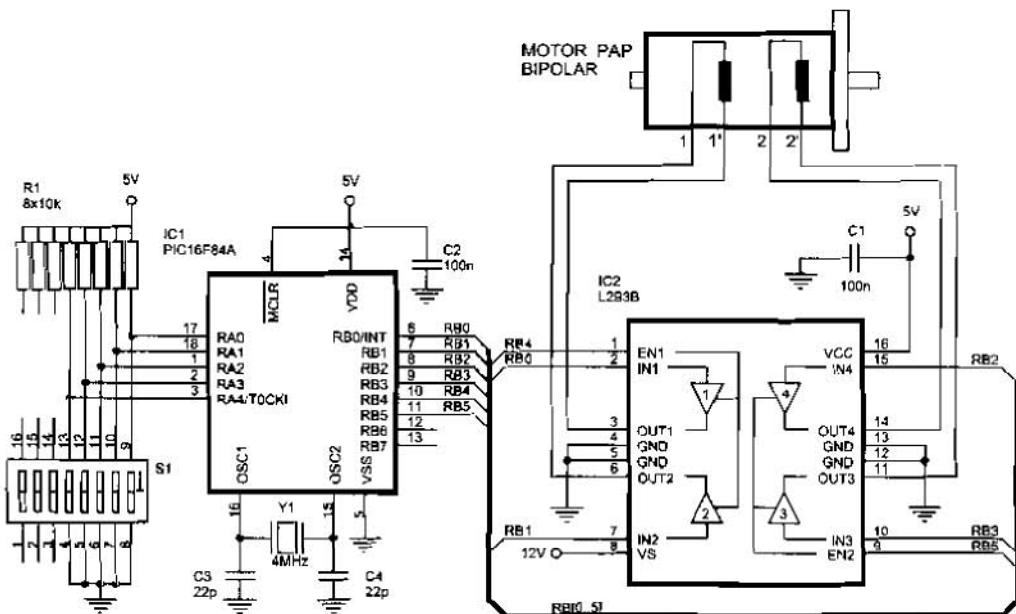
Se mide con el ohmetro para buscar los tres hilos que entre si midan un valor cualquiera distinto de infinita. Estos tres hilos pertenecerán a uno de los juegos de bobinas y los otros tres pertenecerán al otro juego de bobinas. Para averiguar el terminal central se mide la resistencia entre dos cables, obteniendo para nuestro caso concreto la medida de  $150 \Omega$ , midiendo los otros dos resulta  $300 \Omega$ , por lo tanto, el que tiene el valor mitad corresponde con la toma central de la bobina, numerados en la figura 304 con 1'-2' y 3'-4'.

Para identificar cual de los hilos corresponde a las bobinas 1, 2, 3 ó 4, procedemos de la siguiente forma, tendremos que alimentar el motor, el valor de la tensión de alimentación normalmente suele ir indicado por una etiqueta o serigrafiado en la carcasa. En caso contrario, deberemos de tener en cuenta que la mayoría de los motores paso a paso están construidos para trabajar a 4, 5, 6, 12 y 24 voltios. Probamos con 5 V, por precaución, conectando esta alimentación a la patilla central de las dos bobinas. Seguidamente se toma uno de los dos hilos, se numera con el número 1 y se conecta a masa y el otro hilo del mismo juego de bobinas se numerará con el número 2 y se deja sin conectar. A continuación tomamos uno de las terminales del otro juego de bobinas y se conecta a masa, si el motor gira un paso en sentido horario se numera el terminal con el número 3 y el otro terminal con el 4. Si por el contrario lo hace en sentido contrario a las agujas del reloj se numera con 4 y el último termina con el número 3.

Una vez identificados los terminales del motor bipolar y del unipolar, en los siguientes epígrafes veremos los circuitos que vamos a montar para cada uno de ellos.

### **30.10 CONEXIÓN MOTOR PAP BIPOLAR Y PIC16F84**

El montaje que vamos a realizar para el motor bipolar se muestra en la figura 30-13, en el que se ha realizado la conexión del motor PAP a través de los drivers del L293B.



*Figura 30-13 Conexión del motor PAP bipolar u PIC16F84A y driver L293B*

Las líactivación de activar las elas bobinas funcionamiento. Por ejemplo se u

Las lí  
dichos pines  
cerrados, res  
del motor se

30.11 CC

El modelo bipolar. En el sistema alimentación terminales 3 centrales de la red en este ejemplo

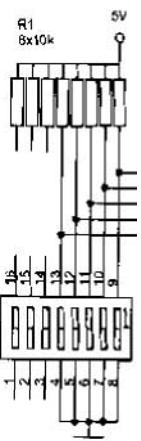


Figura 1

4, procedemos la tensión de en la carcasa. motores paso a s con 5 V, por ; dos bobinas. y se conecta a b2 y se deja sin le bobinas y se terminal con el contrario a las

anipolar, en los  
no de ellos.

**6F84**

en la figura 30-  
los drivers del

Las líneas RB0, RB1, RB2 y RB3 serán las encargadas de generar la secuencia de activación del motor paso a paso, mientras que RB4 y RB5 se ponen siempre a “1” para activar las entradas de habilitación de los drivers. Las salidas de los drivers se conectan a las bobinas del motor para conseguir la corriente suficiente que permita su funcionamiento. La tensión aplicada al pin  $V_S$  es la de alimentación del motor, e*ii* este ejemplo se utiliza un motor de 12 V.

**Las** líneas RA5:RA0 se han conectado a unos interruptores **que** pueden entregar a dichos pines un nivel alto “1” o un nivel bajo “0”, dependiendo de que **estén** abiertos o cerrados, **respectivamente**. Esto **permitirá** controlar las condiciones de funcionamiento del motor según el estado dc estos intrrruptorcs.

### **30.11 CONEXIÓN MOTOR PAP UNIPOLAR Y PIC16F84**

El montaje se muestra en la figura 30-14 y es muy similar al del motor paco a paso bipolar. En este caso se ha conectado la toma intermedia de los bobinados a la tensión de alimentación, el terminal 1 y 2 a las salidas de un par de drivers del L293B y los terminales 3 y 4 a las salidas del otro par de drivers. La tensión aplicada a la tomas centrales de los devanados del motor debe ser igual a su tensión nominal dc alimentación. en este ejemplo se utiliza un motor de 12 V.

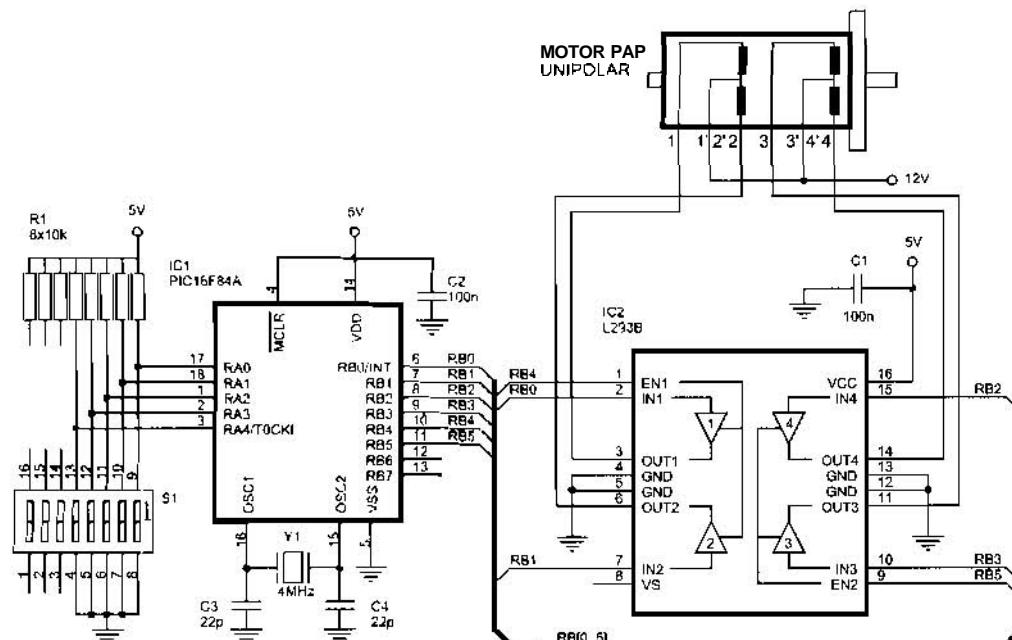


Figura 30-14 Conexión del motor PAP univolar a PIC16F84A y driver L293B

## 30.72 CONTROL DE MOTOR PAP EN MODO FULL STEP

Utilizando cualquiera de los montajes de las figuras 30-13 y 30-14, vamos a analizar el programa MotorPAP\_01.asm, que es un programa que hace girar el motor siempre que esté cerrado el interruptor conectado a la línea RA0 y dependiendo del estado de la línea RA4 girará en un sentido o en el contrario. El programa para los dos tipos de motores es el mismo y su organigrama es el que se muestra en la figura 30-15.

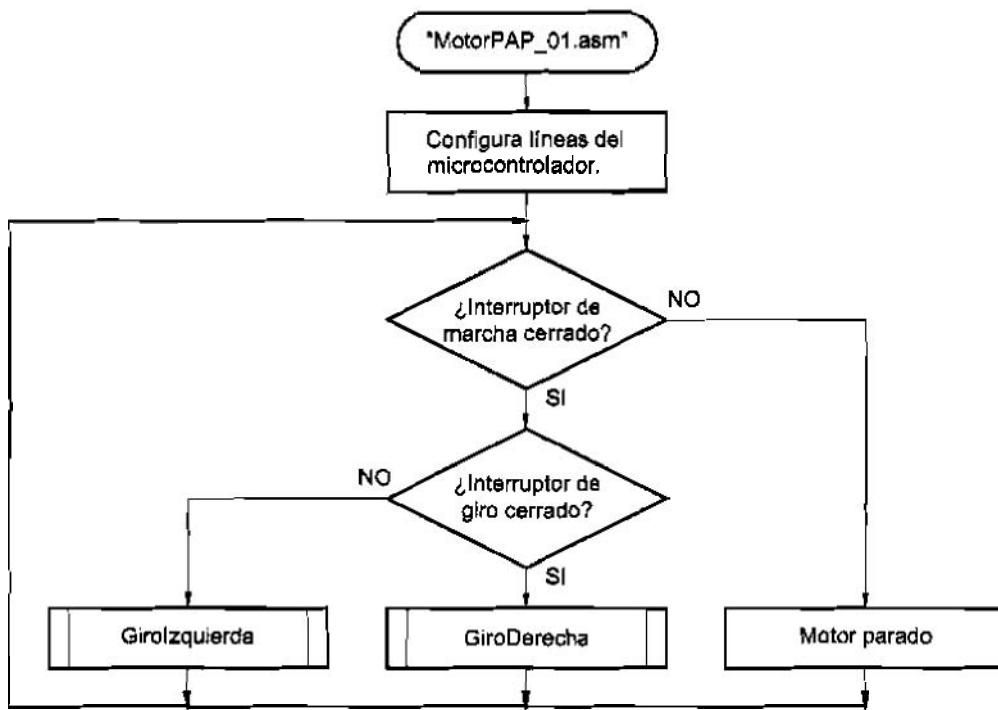


Figura 30-15 Organigrama del programa MotorPAP\_01.asm

\*\*\*\*\* MotorPAP\_01.asm \*\*\*\*\*

Programa de control para un motor paso a paso (PAP) en funcionamiento y sentido de giro. Con RA0 a "0" el motor se pone en marcha y su sentido de giro dependerá del valor que tenga RA4.

ZONA DE DATOS \*\*\*\*\*

```

CONFIG_CPD_OFF & WDT_OFF & PWRT_ON & XT_OSC
LIST      P=16F84A
INCLUDE <PI16F84A.INC>

CLOCK8MOSC
ENDC

```

#DEFINE Entrada  
#DEFINIE Entrada

; ZONA DE CÓDIGO

ORG

Inicio

baf  
baf  
baf

Principal

baf  
baf  
goto  
chr  
goto  
bfsc  
goto  
&  
m

Subrutina "Giroizquierda"

movlw  
call  
movlw  
call  
movlw  
call  
movlw  
call  
retlw

Subrutina "GiroDerecha"

movlw  
call  
movlw  
call  
movlw  
call  
movlw  
call  
retlw

Subrutina "ActivaSalida"

ActivaSalida

## STEP

14, vamos a girar el motor viendiendo del 4 para los dos pines 30-15.

```
#DEFINE EntradaMarcha    PORTA,0      ; Interruptor de puesta en marcha.
#DEFINE EntradaSentido    PORTA,4      ; Interruptor de sentido de giro.

; ZONA DE CÓDIGOS *****

        ORG 0
Inicio
        bcf STATUS,RPO
        bcf EntradaMarcha
        bcf EntradaSentido
        clrf PORTB
        bcf STATUS,RPO
; Estas líneas se configuran como entrada.

Principal
        brfse . . .
        clrf PORTB
        goto Fin ido
; No, para el motor, poniendo a cero la línea de habilitación.

Gira
        bafsc EntradaS
        goto A_Izquierda
        call GiroDerecha
        goto Fin
; Comprueba el sentido de giro deseado.

A_Izquierda
        call GiroIzquierda
        goto Principal
; Gira en un sentido.

Fin
; Gira en sentido contrario.

; Subrutina "GiroIzquierda"
GiroIzquierda
        moviw b'00110101'
        call ActivaSalida
        moviw b'00110110'
        call ActivaSalida
        moviw b'00111010'
        call ActivaSalida
        moviw b'00111001'
        call ActivaSalida
        return
; Primer paso.
; Lo envía a la salida donde está conectado el motor PAP.
; Segundo paso.

; Tercer paso.

; Cuarto y último paso

; Subrutina "GiroDerecha"
GiroDerecha
        moviw b'00111001'
        call ActivaSalida
        moviw b'00111010'
        call ActivaSalida
        moviw b'00110110'
        call ActivaSalida
        moviw b'00110101'
        call ActivaSalida
        return
; Primer paso.
; Segundo paso.

; Tercer paso.

; Último paso.

; Subrutina "ActivaSalida"
ActivaSalida
```

```

movwf PORTB
call Retardo_10ms      ; Temporización antes del siguiente paso.
return

INCLUDE <RETARDOS.INC>
END

```

Este control también se podía haber realizado mediante tablas de datos, de forma similar a los juegos de luces planteados y resueltos en el capítulo 12. Se anima al lector a modificar éste y los próximos ejercicios resolviéndolos mediante el procedimiento descrito en el ejercicio Retardo-08.asm.

### 30.13 REALIZACIÓN DE SECUENCIAS DE MOVIMIENTOS

En muchos proyectos es necesario realizar una secuencia de movimientos con los motores PAP. A continuación se muestra un ejemplo que también nos puede servir para comprobar correcto funcionamiento.

Si no se conoce el numero de pasos que tiene nuestro motor ahora puede ser un buen momento para saberlo. Así, si se hace la temporización mayor llamando por ejemplo a la subrutina Retardo\_50ms en lugar de Retardo\_10ms, dentro de la subrutina "ActivaSalida", podemos contar fácilmente cuántos pasos necesita para dar una vuelta completa. El motor unipolar que hemos utilizado nosotros necesita 48, por lo que cada impulso de excitación recorre 7,5°, que corresponde a uno de los valores comerciales recogidos en la Tabla 30-5.

El programa MotorPAP\_02.asm hace que el motor ejecute una vuelta completa en sentido horario y dos en sentido contrario a las agujas del reloj. Para ello, tiene en cuenta el ángulo de paso del motor o, lo que es igual, el número de pasos a recorrer para que el eje del motor ejecute una vuelta completa.

```

***** MotorPAP_02.asm *****
; El motor PAP realiza una vuelta en sentido y dos en sentido contrario utilizando el modo Full Step.
; ZONA DE DATOS *****

_CONFIG CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

CBLOCK 0x0C
Ciclos          ; Se decrementará cada ciclo de 4 pasos.
VueltaHorario
VueltaAntihorario
ENDC

NumeroCiclos EQU .12
; Un ciclo de 4 pasos son 30 grados para un PAP

```

; ZONA DE C

ORC

Inicio

bsf

cirlf

bcf

Principal

movf

movi

OtraVueltaDere

movi

movs

OtroCicloDerec

call

decfs

goto

decfs

goto

;

movi

movw

OtraVueltaIzqui

movh

movw

OtroCicloIzquier

call

decfsz

goto

decfsz

goto

goto

;

Subrutina "Gir

GiroIzquierda

movlw

call

movlw

call

movlw

call

movlw

call

return

;

Subrutina "Gir

GiroDerecha

movlw

call

; de 7,5° en modo Half-Step. Por tanto para  
; completar una vuelta de **3W**. se requieren 12  
; ciclos de 4 paso, cada uno.

; ZONA DE CÓDIGOS \*\*\*\*\*

	ORG	0	
Inicio	bsf	STATUS,RP0	
	clrf	WRTB	; Las líneas del Puerto B configuradas como salidas.
	bcf	STATUS,RP0	
Principal	movlw	0x02	
	movwf	VueltasHorario	; Dos vueltas en sentido horario.
OtraVueltaDerecha	movlw	NumeroCiclos	
	movwf	Ciclos	
OtroCicloDerecha	call	GiroDerecha	
	decfsz	Ciclos,F	
	goto	OtroCicloDerecha	
	decfsz	VueltasHorario,F	
	goto	OtraVueltaDerecha	
	movlw	0x01	
	movwf	VueltasAntihorario	; Una vuelta en sentido antihorario.
OtraVueltaIzquierda	movlw	NumeroCiclos	; Al ser una sola vuelta no haría falta el contador.
	movwf	Ciclos	; Pero se deja para que el lector pueda hacer las
			; pruebas que crea oportunas cambiando la carga
OtroCicloIzquierda	call	GiroIzquierda	; (VueltaAntihorario).
	decfsz	Ciclos,F	
	goto	OtroCicloIzquierda	
	decfsz	VueltasAntihorario,F	
	goto	OtraVueltaIzquierda	
	goto	Principal	

; Subrutina "GiroIzquierda"

GiroIzquierda		
movlw	b'00110101'	; Primer paso.
call	ActivaSalida	
movlw	b'00110110'	; Segundo paso.
call	ActivaSalida	
movlw	b'00111010'	; Tercer paso.
call	ActivaSalida	
movlw	b'00111001'	; Cuarto y último paso
call	ActivaSalida	
return		

; Subrutina "GiroDerecha"

GiroDerecha		
movlw	b'00111001'	
call	ActivaSalida	; Primer paso para el giro hacia la derecha.

os, de forma  
na al lector a  
ocedimiento

## IENTOS

entos con los  
e servir para

puede ser un  
llamando por  
la subrutina  
ir una vuelta  
lo que cada  
comerciales

completa en  
ene en cuenta  
er para que el

\*\*\*\*\*

Step.

\*\*\*\*\*

PAP

```

movlw b'00111010' ; Segundo paso.
call ActivaSalida
movlw b'00110110' ; Tercer paso.
call ActivaSalida
movlw b'00110101' ; Cuarto y ultimo paso.
call ActivaSalida
return

; Subrutina "ActivaSalida"
ActivaSalida
    movwf PORTB
    call Retardo_50ms ; Temporización antes del siguiente paso.
    return

INCLUDE <RETARDOS.INC>
END

```

### 30.14 CONTROL DE MOTOR PAP EN MODO HALF STEP

Una de las posibilidades que tienen los motores paso a paso es la de utilizarlos en el modo medio paso u *Half Step*. En este caso hay que utilizar una secuencia de 8 estados, como la que se muestra en las Tablas 30-2 y 30-4, como puede comprobarse en el programa MotorPAP\_03.asm, el proceso de programación es el mismo. Como ejemplo de aplicación se va a repetir el programa de la sección anterior pero en modo *Half Step*.

```
***** MotorPAP_03.asm *****

; El motor PAP realiza una vuelta en sentido y dos en sentido contrario utilizando medios
; pasos (modo Half Step) para obtener mas precisión.

; ZONA DE DATOS *****

    CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
    LIST    P=16F84A
    INCLUDE <P16F84A.INC>

    CBLOCK 0x0C
    Ciclos          ; Se decrementará cada ciclo de 8 pasos.
    VueltasHorario
    VueltasAntihorario
    ENDC

    NumeroCiclos    EQU    .12      ; Un ciclo de 8 pasos son 30 grados para un PAP
                                    ; de 7,5° en modo Half-Step. Para tanto para
                                    ; completar una vuelta de 360°, se requieren 12
                                    ; ciclos de 8 pasos cada uno.

; ZONA DE CÓDIGOS *****

    ORG    0
    Inicio
        bsf    STATUS, RP0
```

```

        clrf    PORTB
        bcf    STATUS,RP0      ; Las líneas del Puerto B configuradas como salidas.

Principal
        movlw   0x02           ; Dos vueltas en sentido horario.

OtraVueltaDerecha
        movlw   NumeroCiclos
        movwf   Ciclos

OtroCicloDerecha
        call    GiroDerecha
        decfz  Ciclos,F
        goto   OtroCicloDerecha
        decfz  VueltasHorario,F
        goto   OtraVueltaDerecha

;
        movlw   0x01           ; Una vuelta en sentido antihorario.

OtraVueltaIzquierda
        movlw   NumeroCiclos
        movwf   Ciclos          ; Al ser una sola vuelta no haría falta el contador.
                                ; Pero se deja para que el lector pueda hacer las
                                ; pruebas que crea oportunas cambiando la carga
                                ; (VueltaAntihorario).

OtroCicloIzquierda
        call    GiroIzquierda
        decfz  Ciclos,F
        goto   OtroCicloIzquierda
        decfsz VueltasAntihorario,F
        goto   OtraVueltaIzquierda
        goto   Principal

```

\*\*\*\*\*  
STEP  
utilizarlos en  
e 8 estados,  
barse en el  
no ejemplo  
'alf Step.

```

GiroIzquierda
        movlw   b'00110001'      ; Primer paso.
        call    ActivaSalida
        movlw   b'00110101'      ; Segundo paso.
        call    ActivaSalida
        movlw   b'00110100'      ; Tercer paso.
        call    ActivaSalida
        movlw   b'00110110'      ; Cuarto paso.
        call    ActivaSalida
        movlw   b'00110010'      ; Quinto paso.
        call    ActivaSalida
        movlw   b'00111010'      ; Sexto paso.
        call    ActivaSalida
        movlw   b'00111000'      ; Séptimo paso.
        call    ActivaSalida
        movlw   b'00111001'      ; Octavo y último paso.
        call    ActivaSalida
        return

```

\*\*\*\*\*  
AP  
12  
\*\*\*\*\*  
; Subrutina "GiroDerecha"

```

GiroDerecha
        movlw   b'00111001'      ; Primer paso para el giro hacia la derecha.
        call    ActivaSalida

```

```

movlw b'00111000' ; Segundo paso.
call ActivaSalida
movlw b'00111010' ; Tercer paso.
call ActivaSalida
movlw b'00110010' ; Cuarto paso.
call ActivaSalida
movlw b'00110110' ; Quinto paso.
call ActivaSalida
movlw b'00110100' ; Sexto paso.
call ActivaSalida
movlw b'00110101' ; Séptimo paso.
call ActivaSalida
movlw b'00110001' ; Octavo y ultimo paso.
call ActivaSalida
return

```

: Subrutina "ActivaSalida"

```

ActivaSalida
    movwf PORTB
    call Retardo_50ms ; Temporización antes del siguiente paso.
    return

```

```

INCLUDE <RETARDOS.INC>
END

```

## 30.15 CONTROL DE VELOCIDAD

La velocidad del motor queda determinada por la frecuencia a la que se ejecutan los pasos de la secuencia y la dirección de giro depende del sentido en que se aplique dicha secuencia. Esto proporciona un excelente control de velocidad y posición sin realimentación.

El programa MotorPAP\_04.asm describe un procedimiento para controlar la velocidad. El hardware utilizado será el de las figuras 30-13 y 30-14. La lectura de las cuatro líneas bajas del Puerto A determina el tiempo de retardo que hay entre la aplicación de una nueva combinación a los devanados del motor. Cuanto mayor sea este retardo más despacio girara el motor.

```
***** MotorPAP_04.asm *****
```

```

; Programa de control de velocidad de un motor PAP. La velocidad del motor estará gobernada
; por el valor de las cuatro líneas bajas del Puerto A. El sentido de giro de motor se decide
; en función del valor de la línea RA4.
;
```

```
***** ZONA DE DATOS *****
```

```

_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

```

movlw b'00111000'	;	Segundo paso.	CBLO
call ActivaSalida			Velocid
movlw b'00111010'	;	Tercer paso.	ENDC
call ActivaSalida			#DEFINE Entrad
movlw b'00110010'	;	Cuarto paso.	; ZONA DE CÓD
call ActivaSalida			
movlw b'00110110'	;	Quinto paso.	
call ActivaSalida			
movlw b'00110100'	;	Sexto paso.	ORG
call ActivaSalida			Inicio
movlw b'00110101'	;	Séptimo paso.	bsf
call ActivaSalida			movlw
movlw b'00110001'	;	Octavo y ultimo paso.	movwf
call ActivaSalida			clrf
return			bcf
Principal			
			movf
			andiw
			btfsc
			goto
			call
			movwf
			btfsc
			goto
			cali
			goto
A_Izquierda			A_Izquierda
			call
			goto
ParaMotor			ParaMotor
			clrf
Fin			goto
; Subrutina "Selecc			
; Alterando los val			
SeleccionaVelocida			
			addwf
			DT
			DT
; Subrutina "Giroza			
GiroIzquierda			
			movlw
			call
			movlw
			call
			movlw
			call
			movlw
			call
			return

```
CBLOCK 0x0C
Velocidad
ENDC
```

```
#DEFINE EntradaSentido PORTA,4 ; Interruptor de sentido de giro.
```

```
; ZONA DE CÓDIGOS *****
```

	ORG	0	
Inicio	bsf	STATUS.RP0	
	inovlw	b'0001111'	; El Puerto A se configura como entrada.
	clrfwf	PORTA	
	clrf	PORTB	; Las líneas del Puerto B configuradas como salidas.
Principal	bcf	STATUS.RP0	
	movf	PORTA,W	
	andiw	b'00001111'	; Lee el puerto de entrada
	btfc	STATUS.Z	; Se queda con los cuatro bits bajos.
		ParaMotor	
	call	SeleccionaVelocidad	
	movwf	Velocidad	
	btfc	EntradaSentido	
	goto	A_Izquierda	
	di	GiroDerecha	
	goto	Fin	
A-Izquierda	call	GiroIzquierda	
	goto	Fin	
ParaMotor	clrf	PORTB	
	Fin	goto Principal	; Para el motor, poniendo a cero la linea de habilitación.

; Subrutina "SeleccionaVelocidad"

; Alterando los valores de esta tabla se pueden conseguir diferentes retardos.

SeleccionaVelocidad

addwf	PCL,F	
DT	0, d'75', d'70', d'65', d'60', d'55', d'50', d'45', d'40'	
DT	d'35', d'30', d'25', d'20', d'15', d'10', d'5'	

; Subrutina "GiroIzquierda"

GiroIzquierda

movlw	b'00110101'	; Primer paso.
call	ActivaSalida	; Lo envía a la salida donde está conectado el motor PAP.
movlw	b'00110110'	; Segundo paso.
call	ActivaSalida	
movlw	b'00111010'	; Tercer paso.
call	ActivaSalida	
movlw	b'00111001'	; Cuarto y último paso
call	ActivaSalida	
return		

; Subrutina "GiroDerecha"

GiroDerecha

```
    movlw  b'00111001'      ; Primer paso.
    call   ActivaSalida
    movlw  b'00111010'      ; Segundo paso.
    call   ActivaSalida
    movlw  b'00110110'      ; Tercer paso.
    call   ActivaSalida
    movlw  b'00110101'      ; Último paso.
    call   ActivaSalida
    return
```

; Subrutina "ActivaSalida"

```
CBLOCK
Contador
ENDC
```

ActivaSalida

```
    movwf  PORTB
    movf   Velocidad,W
    movwf  Contador
; Y ahora el retardo en función del valor de
; de la variable Velocidad.
```

Retardo

```
    call   Retardo_1ms
    decfsz Contador,F
    goto  Retardo
    return
```

INCLUDE <RETARDOS.INC>

END

SE

Para la servomotores, (figura 31-1).  
de modelismo a timón dc un barco tamaño, bajo coste ideales para la co

### 31.1 SERVOMOTOR

Un servomotor es una ruedas dentadas que gira una pequeña tarjeta figura 31-2 muestra

## 31.2 FU

## CAPÍTULO 31

**SERVOMOTORES DE RADIOCONTROL**

Para la realización de microrobots experimentales es frecuente utilizar servomotores, que son pequeños dispositivos utilizados tradicionalmente en radiocontrol (figura 31-1). Popularmente reciben el nombre de "servos" y suelen usarse para el control de modelismo a distancia, actuando sobre el acelerador de un motor de combustión, en el timón de un barco o de un avión, en el control de dirección de un coche, etc. Su pequeño tamaño, bajo consumo, además de una buena robustez y notable precisión, los hacen ideales para la construcción de los microrobots.



Figura 31-1 Servomotor de radiocontrol

**31.1 SERVOMOTORES PARA MICROROBÓTICA**

Un servomotor está constituido por un pequeño motor de corriente continua, unas ruedas dentadas que trabajan como reductoras, lo que le da una potencia considerable, y una pequeña tarjeta de circuito impreso con la electrónica necesaria para su control. La figura 31-2 muestra el despiece de un servo

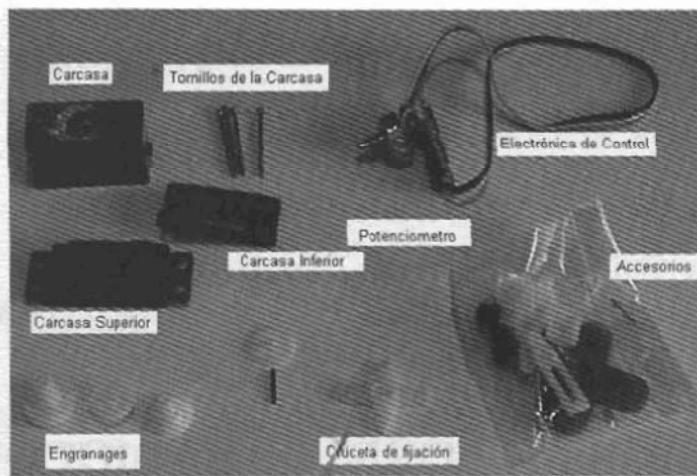


Figura 31-2 Despiece de un servomotor de radiocontrol

La figura 31-3 explica como están ensambladas estas piezas dentro del servomotor

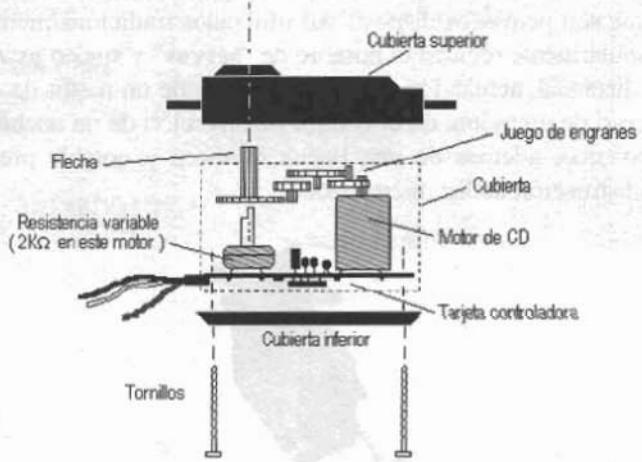


Figura 31-3 Ensamblaje de un servomotor

Debido a la reductora mecánica formada por las ruedas dentadas, un servo estándar como el Hitec HS-300 tiene un torque de 3Kg por cm y una velocidad constante y proporcional a la carga. Otro servo compatible con el anterior y muy utilizado es el Futaba S3003, [www.futaba-rc.com](http://www.futaba-rc.com).

Si el lector tiene dificultades para comprar estos servos en su proveedor habitual de componentes electrónicos puede adquirirlos en tiendas de modelismo donde vendan material para radiocontrol de aviones, coches, barcos, etc. También puede entrar en [www.modelimport.com](http://www.modelimport.com), que es la página Web de Model Import S.A, empresa con sede

en Torrejón de Ardoz, donde expone su catálogo, supuesto, también en Internet, siempre

### 31.2 FUNCIONAMIENTO

La tensión de alimentación es de 4.8 a 6.0 voltios. El control se realiza mediante una señal cuadrada de 100 Hz (Pulse Width Modulation). La señal de control se aplica al eje del motor (que es el eje de retroalimentación)

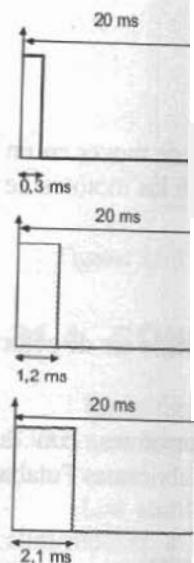


Figura 31-4 Tríplice de pulsos

La duración de los pulsos se indica en la figura 31-4. Cada pulso consta de un ancho de señal y un tiempo muerto. La duración total de cada pulso es constante y es de 20 ms. Los tiempos de los pulsos son:

en Torrejón de Ardoz (Madrid), distribuidora oficial para España de los servos Futaba, donde expone una completa relación de tiendas ordenadas por comunidades. Y, por supuesto, también puede adquirirlos en alguna de las múltiples tiendas que hay en Internet, siempre comprobando que trabaja con las necesarias garantías.

## 31.2 FUNCIONAMIENTO DEL SERVOMOTOR

La tensión de alimentación de los servos suele estar comprendida entre los 4 y 8 voltios. El control de un servo se limita a indicar en qué posición se debe situar, mediante una señal cuadrada TTL modulada en anchura de impulsos PWM (*Pulse Width Modulation*). La duración del nivel alto de la señal indica la posición donde queremos poner el eje del motor. El potenciómetro que el servomotor tiene unido solidariamente al eje del motor (ver figura 31-3) indica al circuito electrónico de control interno mediante una retroalimentación, si éste ha llegado a la posición deseada.

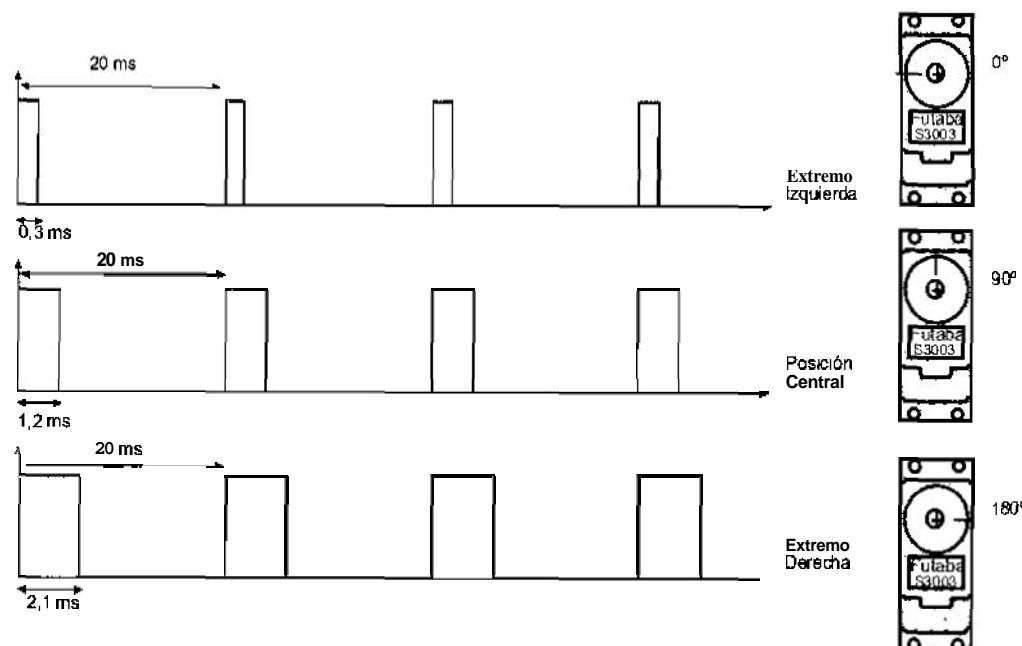


Figura 31-4 Tren de impulsos para control de un servo de radiocontrol Futaba S3003

La duración de los impulsos indica el ángulo de giro del motor, como muestra la figura 31-4. Cada servomotor tiene sus márgenes de operación, que se corresponden con el ancho del pulso máximo y mínimo que el servo cintiendo y que, en principio, mecánicamente no puede sobrepasar. Estos valores varían dependiendo del modelo de servomotor utilizado. Para el servomotor Futaba S3003 los valores de la señal a nivel alto están entre 0,3 y 2,1 ms, que dejarían al motor en ambos extremos de giro. El valor 1,2 ms indicaría la posición central, mientras que otros valores de anchura del pulso lo dejarían

en posiciones intermedias que son proporcionales a la anchura de los impulsos. Si se sobrepasan los límites de movimiento del servo, éste comenzará vibrar o a emitir un zumbido, denunciando un cambio en la anchura del pulso.

**El periodo entre pulso y pulso** no es critico. Se suelen emplear valores entre **10 ms** y **30 ms**, aunque lo habitual es utilizar **20 ms**, que implica una **frecuencia de 50 Hz**. Si el **intervalo entre pulso** y pulso es inferior al mínimo puede interferir con la temporización interna del servo causando un zumbido y la vibración del brazo de salida. Si es mayor que el máximo, entonces el servo pasará a estado dormido entre pulsos provocando que se **mueva a pequeños intervalos**.

Es importante destacar **que** para que un servo se mantenga en la misma posición, es necesario **enviarle** continuamente un pulso de una anchura constante. De este modo si existe **alguna fuerza que** le obligue a abandonar esta posición intentará **resistirse**. Si se **deja de enviar** pulsos, o el intervalo entre pulsos es mayor del máximo permitido, entonces el servomotor perderá **fuerza** y dejará de **intentar** mantener su posición, de modo que cualquier fuerza externa podría desplazarlo.

### 31.3 TERMINALES

Un servomotor es básicamente un motor eléctrico que sólo se puede mover en un ángulo de aproximadamente 180 grados (no dan vueltas completas como los motores de corriente continua). Los servomotores disponen de tres terminales:

- Positivo de alimentación unido al cable de color rojo.
  - Masa o negativo, que casi siempre es un cable de color negro.
  - Señal por donde se aplica la entrada de impulsos y cuyo cable suele ser de color blanco, amarillo o naranja.

La Tabla 31-1 muestra una relación de fabricantes de servomotores, con la descripción de cada uno de los terminales destacándose en negrilla los fabricantes **Futaba** y **Hitec** que son los más importantes.

Marca Servomotor de R/C	Positivo	Señal	Negativo
Futaba	Rojo	Blanco	Negro
JR	Rojo	Naranja	Marrón
Hitec	Rojo	Amarillo	Negro
Airtronics	Rojo	Naranja	Negro
Fleet	Rojo	Blanco	Negro
Kraft	Rojo	Naranja	Negro

Tabla 31-1 Identificación de los terminales de los servos, según diversos fabricantes

En la  
fabricantes.

Figura 31-

314 COM

En la fin

**Los ter**  
alimentación  
microcontrolad  
las dos fuentes  
se construye un  
**utilizar dos fu**  
energía a los  
presupuestopara

El siguiente ejemplo, el eje

n pulsos. Si se o a emitir un

es entre 10 ms de 50 Hz. Si el temporización a. Si es mayor provocando que

sma posición, e este modo si resistirse. Si se mo permitido. ción, de modo

e mover en un los motores de

le ser de color

tores, con la riantes Futaha

En la figura 31-5 pueden identificarse los conectores de algunos de estos fabricantes.

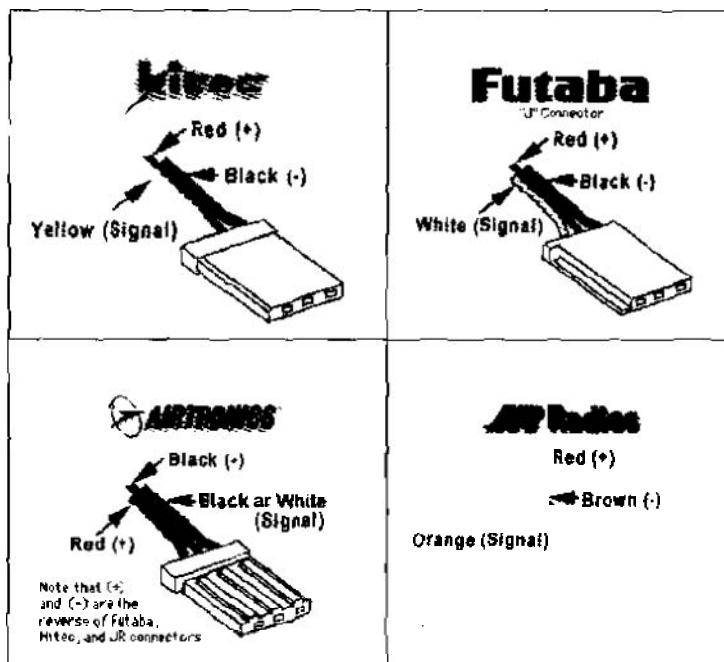


Figura 31-5 Conectores y cables usados por algunos fabricantes de servomotores

### 31.4 CONEXIÓN DE UN SERVOMOTOR A UN PIC16F84

En la figura 31-6 se muestra un ejemplo de conexión de un servomotor Futaba S3003 a un microcontrolador PIC16F84A.

Los terminales de alimentación del servomotor se conectan a una fuente de alimentación a 5V que puede ser la misma que se utilice para alimentar al microcontrolador. En caso de utilizar dos fuentes distintas debemos conectar las masas de las dos fuentes de alimentación, para que tengan la misma tensión de referencia. Cuando se construye un mecanismo o un microrobot con servomotores, es siempre recomendable utilizar dos fuentes de energía distintas, una para la "electrónica" y la otra para dar energía a los servomotores, pero desgraciadamente no siempre hay espacio ni presupuesto para ello.

El siguiente programa ayudará a comprender la técnica de programación. En este ejemplo, el eje del servomotor girará de 0 a 180° y de 180 a 0° indefinidamente.

fabricantes

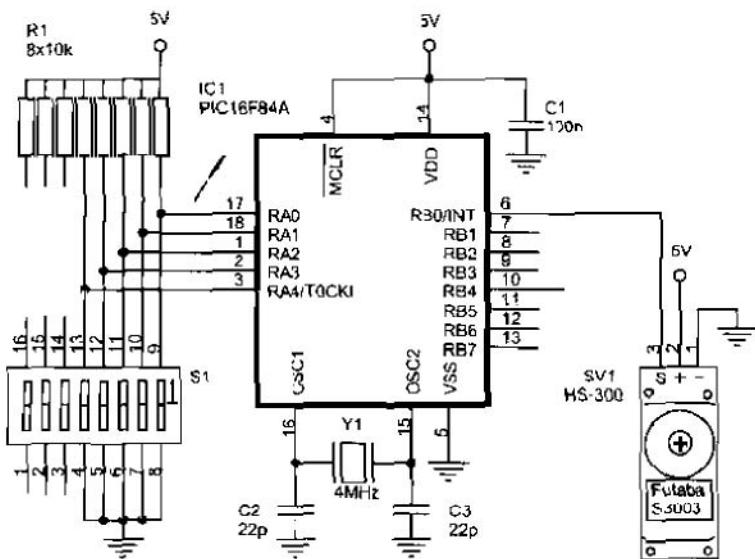


Figura 31-6 Conexión del PIC16F84A para control del servomotor

\*\*\*\*\* Servos\_01.asm \*\*\*\*\*

; Programa de control del posicionamiento de un servomotor Futaba S3003. Controla el ángulo mediante una señal cuadrada PWM de 20 ms de periodo que se aplica a su línea de control.  
; El ángulo es gobernado por el tiempo en alto de la señal cuadrada desde 0° (para 0,3 ms de tiempo en alto), hasta 180° (para un tiempo en alto de 2,1 ms).

; En este programa el servomotor se posiciona en 0°, 90°, 180°, vuelve a 90° y repite el ciclo.  
; Permanece en cada posición durante 1 s. El funcionamiento se explica en la siguiente tabla,  
; donde se ha tomado como tiempo patrón 100 µs (0,1 ms) conseguidos mediante interrupciones  
; por desbordamiento del Timer0.

FactorAlto	Tiempo Alto 0,1·FactorAlto	Tiempo Bajo 0,1·(200-FactorAlto)	Ángulo (Grados)
3	0,1 ms	19,7 ms	0°
12	1,2 ms	18,8 ms	90°
21	2,1 ms	17,9 ms	180°
12	1,2 ms	18,8 ms	90°

\*\*\*\*\* ZONA DE DATOS \*\*\*\*\*

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
Contador
FactorAlto
ENDC
```

; Factor por el que se va a multiplicar el tiempo  
; patrón de 100 µs para obtener el tiempo en atto.

TMR0\_Carga

#DEFINE Se

; ZONA DE C

OR

got

OR

got

bsf

bcf

mo

mo

bcf

mo

andi

call

mov

call

incf

goto

SeleccionaFac

addh

DT

; Subrutina "Ti

; Mantiene la a

; a 100µs x (20

CBI

Gua

Tim

ENI

Timer0\_Intern

mov

swaj

mov

bcf

mov

decf

goto

bfsc

```

TMR0_Carga EQU -d'90' ; Valor obtenido experimentalmente con la ventana
; Stopwatch para un tiempo de 100 µs.
#define Salida PORTB,0 ; Línea del Puerto B donde se conecta el servomotor.

; ZONA DE CÓDIGOS ****
ORG 0
goto Inicio
ORG .4
goto Timer0_Interrupcion
Inicio
bsf STATUS,RP0
bcf Salida ; Esta línea se configura como salida.
movlw b'00001000' ; TMRO sin prescaler.
movwf OPTION_REG
bcf STATUS,RP0
movlw b'10100000'
movwf INTCON ; Autoriza interrupción T0I y la general (GIE).
clr F Contador
Principal
movf Contador,W
andlw b'00000011'
call SeleccionaFactorAlto
movwf FactorAlto
call Retardo_1s
incf Contador,F
goto Principal

SeleccionaFactorAlto
addwf PCL,F
DT d'3', d'12', d'21', d'12' ; Tabla para el servo Futaba S3003

; Subrutina "Timer0_Interrupcion"
;
; Mantiene la salida en alto un tiempo igual a 100µs x (FactorAlto) y en bajo un tiempo igual
; a 100µs x (200-FactorAlto). El periodo de la señal cuadrada lo mantiene en 20 ms.

CBLOCK
Guarda_W
Guarda_STATUS
Timer0_ContadorA ; Contador auxiliar.
ENDC

Timer0_Interrupcion
movwf Guarda_W ; Guarda los valores de tenían W y STATUS en el
swapf STATUS,W ; programa principal.
movwf Guarda_STATUS ; Garantiza que trabaja en el Banco 0.
bcf STATUS,RP0
movlw TMR0_Carga
movwf TMR0
decfsz Timer0_ContadorA,F ; Decrementa el contador.
goto Fin_Timer0_Interrupcion ; Testea el anterior estado de la salida
btfs Salida

```

```

    goto    EstabaAlto
EstabaBajo
    bsf     Salida
    movwf  FactorAlto,W
    movwf  Timer0_ContadorA
    goto   Fin_Timer0_Interrupcion
    ; Estaba bajo y lo pasa a alto
    ; Repone el contador nuevamente con el tiempo en
    ; alto.
    ; A partir de una e
EstabaAlto
    bcf     Salida
    movwf  FactorAlto,W
    sublw  .200
    movwf  Timer0_ContadorA
    ; Estaba alto y lo pasa a bajo.
    ; Repone el contador nuevamente con el tiempo
    ; en bajo.
    ; El periodo será de  $100\mu s \cdot 200 = 20000\mu s = 20ms$ .
    ; ZONA DE DATOS
    ; CON LIST INCLUI
Fin_Timer0_Interrupcion
    Guarda_STATUS,W
    movwf STATUS
    swapf Guarda_W,F
    swapf Guarda_W,W
    bcf   INTCON,RBIF
    bcf   INTCON,T0IF
    reffie
    ; Restaura registros W y STATUS.

INCLUDE <RETARDOS.INC>
END
    ; TMRO_Carga
    ; La próxima constante
    ; AltoCeroGrados
    ; TiempoPatron
    ; FactorMinimo
    ; #DEFINE Salida
    ; ZONA DE CÓDIGO
    ; ORG
    ; goto
    ; ORG
    ; goto
    ; Inicio
    ; bsf
    ; bcf
    ; movlw
    ; movwf
    ; movlw
    ; movwf
    ; bcf
    ; movlw
    ; movwf
    ; movlw
    ; movwf
    ; Principal
    ; movf
    ; andlw
    ; addlw
    ; movwf

```

A la hora de elaborar el programa de control, hay que tener en cuenta que las especificaciones de estos servomotores dc pequeño costo no suelen ser muy estrictas. De hecho si cambia un servo por otro de la misma marca y modelo no es raro que tenga que reajustar el centrado e incluso los recorridos. Luego no debe extrañarse si necesitará reajustar ligeramente las constantes de estos programas, para su caso particular.

En el siguiente programa ejemplo el ángulo del servomotor es controlado por el valor de entrada fijado por los interruptores conectados al Puerto A con una resolución de 10°. Así si la entrada vale 0 se posiciona en 0°, si vale 1 en 10°; si vale 2 en 20°; ..., y si vale 18 en 180°.

\*\*\*\*\* Servos\_02.asm \*\*\*\*\*

; Programa de control del posicionamiento de un servomotor Futaba S3003. Controla el ángulo mediante una señal cuadrada PWM de 20 ms de periodo que se aplica a su línea de control.  
; El ángulo es gobernado por el tiempo en alto de la señal cuadrada desde 0° (para 0,3 ms  
; de tiempo en alto) hasta 180° (para un tiempo en alto de 2,1 ms)

; En este programa las líneas del Puerto A controlan el ángulo de posicionamiento con una resolución de 10° según los valores que se indican en la siguiente tabla, tornando como tiempo patrón 100 µs (0,1 ms) conseguidos mediante interrupciones por desbordamiento del Timer 0.

Entrada RA4:RA0	FactorAlto (3+Entrada)	Tiempo Alto 0,1xFactorAlto	Tiempo Bajo 0,1x(200-FactorAlto)	Ángulo (Grados)
0	3	0,3 ms	19,7 ms	0°
1	4	0,4 ms	19,6 ms	10°
2	5	0,5 ms	19,5 ms	20°

; 3	6	0,6 ms	19,4 ms	30°
; ...	...	...	...	...
; 17	20	2,0 ms	18,0 ms	170°
; 18	21	2,1 ms	17,9 ms	180°
;				

; A partir de una entrada superior a 18 el servo vibrará.

#### ZONA DE DATOS \*\*\*\*\*

```
_CONFIG_C_P_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

CBLOCK 0x0C			
FactorAlto			;
ENDC			Factor por el que se va a multiplicar el tiempo ;patrón de 100 µs para obtener el tiempo en alto.
TMR0_Carga EQU -d'90'			;
			Vaor obtenido experimentalmente con la ventana ; Stopwatch para un tiempo de 100 µs.

; La próxima constante hay que variarla según el tipo de Servomotor utilizado.

AltoCeroGrados EQU d'300'		;
TiempoPatron EQU d'100'		Timeo en alto para 0°. Para el Futaba S3003. 300 µs. ; 100 µs conseguido mediante interrupciones.
FactorMinimo EQU AltoCeroGrados/TiempoPatron		
#DEFINE Salida PORTB,0		;
		Línea del Puerto B donde se conecta el servomotor.

#### ZONA DE CÓDIGOS \*\*\*\*\*

ORG 0			
goto inicio			
ORG .4			
goto Timer0_Interrupcion			
<b>Inicio</b>			
bsf STATUS,RP0			
bcf Salida			;
movlw b'00011111'			Esta línea se configura como salida.
movwf PORTA			;
movlw b'00001000'			Puerto A configurado como entrada.
movwf OPTION_REG			
bcf STATUS,RP0			
movlw TMR0_Carga			
movwf TMR0			;
movlw b'10100000'			TMRO sin prescaler.
movwf INTCON			
<b>Principal</b>			
PORTA,W			;
andlw b'00011111'			Lee el puerto de entrada
addlw FactorMinimo			;
movwf FactorAlto			Se queda con los bits válidos.
			;
			Para conseguir el tiempo mínimo correspondiente a 0°.
			;
			Valor entregado a la subrutina de

```

    goto Principal ; atención a la interrupción.

; Subrutina "Timer0_Interrupcion"
; Mantiene la salida en alto un tiempo igual a 100µs x (FactorAlto) y en bajo un tiempo igual
; a 100µs x (200-FactorAlto). El periodo de la señal cuadrada lo mantiene en 20 ms.

    CBLOCK
    Guarda_W
    Guarda_STATUS
    Timer0_ContadorA : Contador auxiliar.
    ENDC

Timer0_Interrupcion
    movwf Guarda_W ; Guarda los valores de tenían W y STATUS en el
    swapf STATUS,W ; programa principal.
    movwf Guarda_STATUS
    bcf STATUS,RP0 ; Garantiza que trabaja en el Banco 0.
    movlw TMRO_Carga
    movwf TMRO
    decfsz Timer0_ContadorA,F ; Decrementa el contador.
    goto Fin_Timer0_Interrupcion
    btfsc Salida ; Testea el anterior estado de la salida.
    goto EstabaAlto

EstabaBajo
    bsf Salida ; Estaba bajo y lo pasa a alto.
    movf FactorAlto,W ; Repone el contador nuevamente con el tiempo en
    movwf Timer0_ContadorA ; alto.

EstabaAlto
    bcf Salida ; Estaba alto y lo pasa a bajo.
    movf FactorAlto,W ; Repone el contador nuevamente con el tiempo
    sublw .200 ; en bajo.
    movwf Timer0_ContadorA ; El periodo será de 100µs·200=20000µs=20ms.

Fin_Timer0_Interrupcion
    swapf Guarda_STATUS,W ; Restaura registros W y STATUS.
    movwf STATUS
    swapf Guarda_W,F
    swapf Guarda_W,W
    bcf INTCON,RBIF
    bcf INTCON,T0IF
    retfie

END

```

## 32.1 SE

El pres  
sensores y  
especialmente  
Los sensores e

- LDR,
- CNY
- colores
- OPB7
- H21A
- GP2E
- IS471
- Bump
- SRF0

Pero ar  
mediante inve

## 32.2 INV

Alguno  
dicha señal an  
la figura 32-1.

*(Notas Capítulo 32)*

## CAPÍTULO 32

# SENSORES PARA MICROROBÓTICA

## 32.1 SENSORES PARA MICROROBÓTICA

El **presente** capítulo es una base de datos o guía de referencia sobre algunos sensores y de cómo conectarlos al microcontrolador para realizar aplicaciones especialmente dirigidas a los microrobots pero adaptable a muchos proyectos industriales. Los sensores que se van a estudiar son:

- LDR, detector pasivo de luz. Se utiliza para detectar ausencia o presencia de luz.
- CNY70, sensor óptico reflexivo por infrarrojo con salida a transistor. Diferencia colores blanco y negro.
- OPB703/4/5, sensor óptico reflexivo por infrarrojo.
- H21A1, sensor óptico de barrera. Indicado para medir velocidad de motores.
- GP2Dxx, sensores infrarrojos para medición de distancia.
- IS471F, censor detector de proximidad de obstáculos por infrarrojos.
- Bumper, final de carrera mecánico para detección de obstáculos
- SRF04, sensor de obstáculos y medidor de distancias por ultrasonido.

Pero **antes** es necesario conocer una forma de acondicionar señales no digitales mediante inversores Trigger Schmitt.

## 32.2 INVERSOR TRIGGER SCHMITT 40106

Algunos sensores no proporcionan señales digitales puras y es necesario conformar dicha señal antes de aplicarla al microcontrolador, como en el ejemplo que se muestra en la figura 32-1.

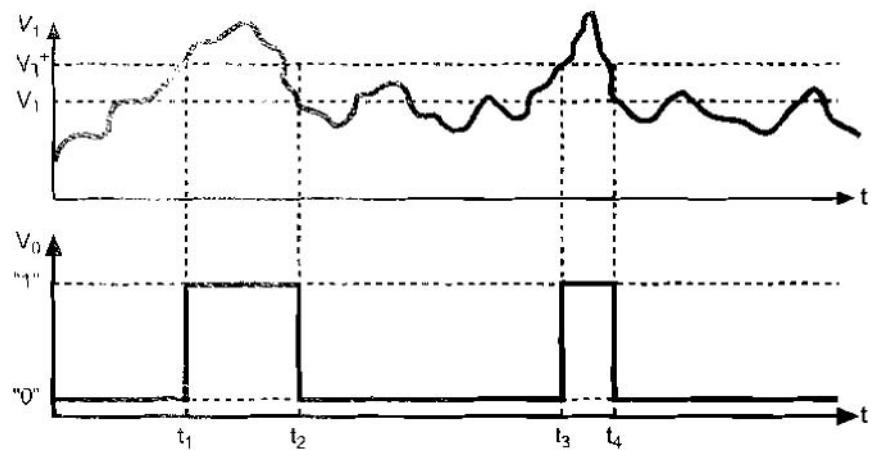


Figura 32-1. Señales de entrada y salida de un circuito Trigger Schmitt

Una forma sencilla de conformar un señal en digital es mediante puertas **Trigger Schmitt**, como **Iris** que tiene el **circuito integrado 40106**. Este dispositivo contiene 6 inversores Trigger Schmitt encapsulados según se indica en la figura 32-2.

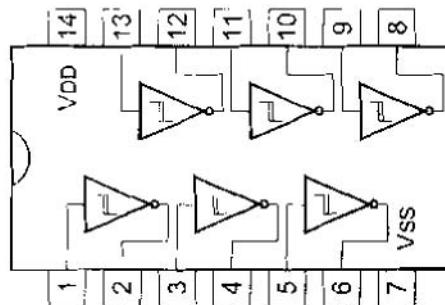


Figura 32-2. 40106. Seis inversores Trigger Schmitt

Estos dispositivos tienen una característica de transferencia como la que se muestra en la figura 32-3. En esta curva se aprecia que si la tensión de entrada aumenta desde 0 V hasta un nivel alto la transición se produce siguiendo la curva A y comuta para el valor  $V_1^+$ , denominado **umbral superior**. Por el contrario, si la entrada está a un nivel alto y disminuye hasta 0V, la transición se produce siguiendo la curva B cuando se alcanza el denominado **umbral inferior**  $V_1^-$ . Los valores de  $V_T^+$  y de  $V_T^-$  para el caso del 40106 dependen de la tensión de alimentación y pueden tomar los valores de la tabla 32-1,

En la curva de transferencia del inversor Schmitt es importante observar que la transición de salida de Alto → Bajo es distinta que la de Bajo → Alto. A este fenómeno se le conoce como **Histeresis**.

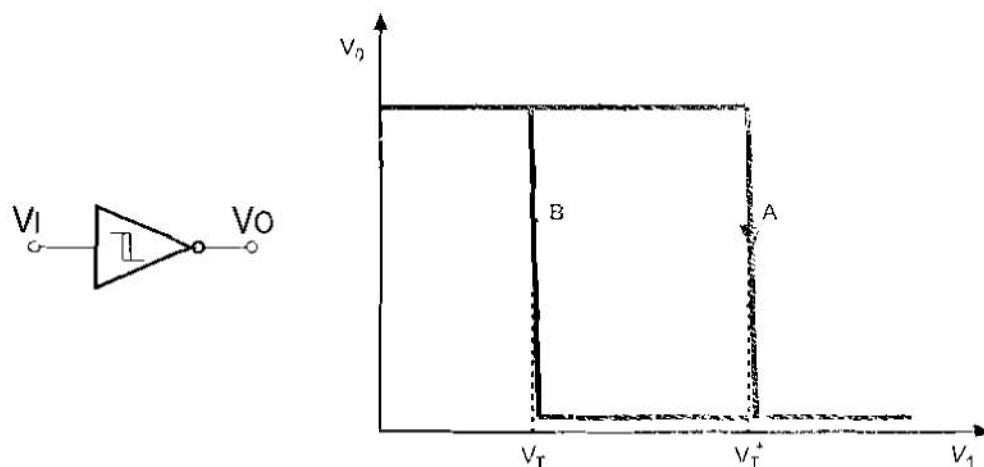


Figura 32-3 Curva de transferencia de un inversor. Trigger Schmitt

**Trigger tiene 6**  
**El inversor Trigger Schmitt, y todos.** los dispositivos con este tipo de funcionamiento, **utiliza el símbolo de la figura 32-3 para indicar que pueden responder de una manera fiable ante señales que cambian con lentitud.** Esta simbología se basa en añadir el dibujo de la gráfica de histéresis en la entrada correspondiente.

PARÁMETRO	$V_{DD}$	MÍNIMO	TÍPICO	MÁXIMO
$V_T^+$	5,0	2,2	2,9	3,6
	10	4,6	5,9	7,1
	15	6,8	8,8	10,8
$V_T^-$	5,0	0,9	1,9	2,8
	10	2,5	3,9	5,2
	15	4,0	5,8	7,4

Tabla 32-1 Valores de  $V_T^+$  y  $V_T^-$  para el 40106 (todos los valores en voltios)

Estos circuitos **son** de gran utilidad cuando se desea controlar un circuito digital con señales que no lo son o señales digitales con una señal de ruido sumada. En la figura 32-1 **se muestra** como actúa un circuito **no** inversor frente a una señal que no es puramente digital. Cuando la señal  $V_I$  alcanza el valor  $V_T^+$  la salida  $V_O$  bascula a un nivel alto y no **vuelve** a tomar un nivel bajo hasta que la entrada no llegue a  $V_T^-$ . En la figura 32-1 se ha **considerado** un no inversor para una más fácil explicación. Para un inversor como el 40106, la señal de salida hubiera estado invertida respecto de la figura.

En conclusión, un dispositivo Trigger Schmitt produce transiciones de salida limpias y rápidas, aunque la entrada no lo sea. El microcontrolador PIC16F84A posee la línea RA4 con entrada Trigger Schmitt (figura 5-7) que se puede utilizar para este fin sin necesidad de intercalar un 40106.

### 32.3 LDR

Las resistencias dependientes de la luz, LDR (*Light Dependent Resistor*) o fotorresistencias, son dispositivos que varían su resistencia en función de la luz que incide sobre su superficie. Cuanto mayor sea la intensidad luminosa que incide sobre ella menor será la resistencia entre extremos de la misma. Para su fabricación se utilizan materiales fotosensibles. Su aspecto físico y simbología más común se muestra en la figura 32-4.

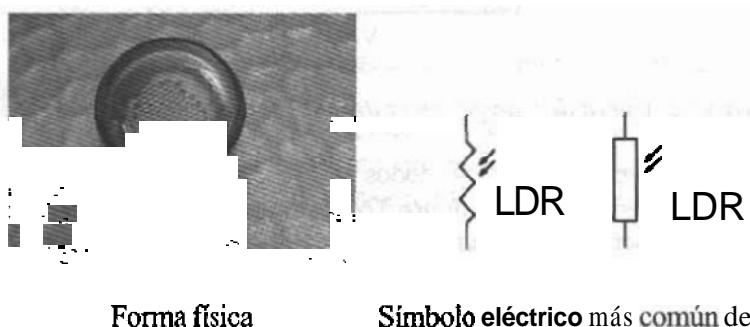


Figura 32-4 LDR

Su valor nominal se especifica sin que incida la luz externa. Así, por ejemplo, una LDR de valor nominal  $50\text{ k}\Omega$ , como la de la figura, tendrá dicho valor si se tapa de manera que no incida la luz sobre su superficie, si se la acerca a una bombilla de  $60\text{ W}$  puede bajar hasta unos  $30\text{ }\Omega$ .

Las principales aplicaciones de estos componentes son controles de iluminación, control de circuitos con relés, en alarmas, etc.

La forma más sencilla de conectar estos sensores de luz a un microcontrolador es realizando un divisor de tensión con la LDR y un potenciómetro, que permite ajustar el nivel de luz a detectar. A la salida del divisor de tensión se le coloca una puerta Trigger Schmitt para conformar la señal, también puede hacerse en una entrada de este tipo como la RA4 del PIC16F84A tal muestra la figura 32-5.

El programa Sensor\_LDR\_01.asm cuenta el número de veces que es cortado un haz de luz que incide sobre la LDR.

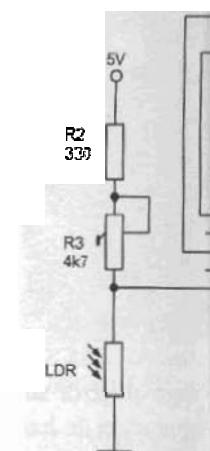


Figura 32-5 Circuito de conexión de un LDR

\*\*\*\*\*  
; Una LDR se conecta a la RA4 del PIC16F84A. Una vez que se oscurece la LDR, la señal de salida de la RA4 se invierte. Por lo tanto, la señal de salida del módulo LCD se invierte cada vez que se oscurece la LDR. (hasta 99% de oscurecimiento máximo).

; ZONA DE DATOS

CONFIG  
LIST  
INCLUDE  
CBLOCK0  
ENDC

; ZONA DE CÓDIGO

ORG 0000H  
Inicia  
call L1  
bsf S0  
movlw b0  
movwf C0  
bcf S1  
clrf T0

; La sección "Principia"

es de salida  
84A posee la  
ra este fin sin

(Resistor) o  
uz que incide  
ore ella menor  
an materiales  
ura 32-4.

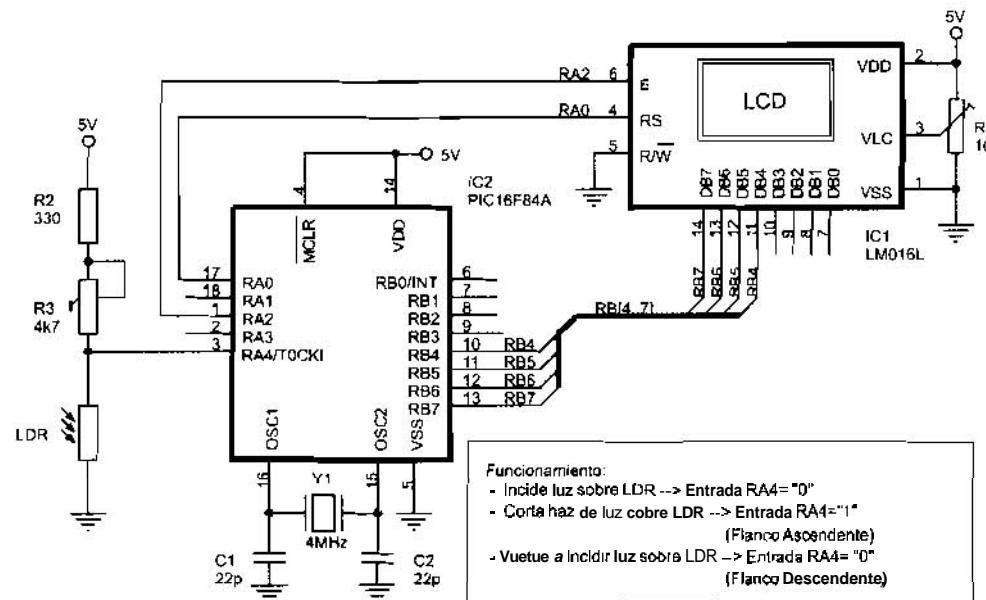


Figura 32-5 Conexión de una LDR a la entrada Trigger Schmitt RA4 de un PIC16F84A

```
***** Sensor_LDR_01.asm *****
;
; Una LDR se conecta a la entrada Trigger Schmitt RA4/T0CKI aplicando impulsos al Timer 0 cada
; vez que se oscurece al interponerse un objeto entre la fuente de luz y la LDR. En la pantalla
; del módulo LCD se visualiza el número de veces que se interrumpe el haz de luz en dos dígitos
; (hasta 99 máximo).
;
; ZONA DE DATOS *****
```

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK0x0C
ENDC
```

```
; ZONA DE CÓDIGOS *****
```

ORG	0	
Inicio	call LCD_Inicializa	
	bsf STATUS,RP0	; Acceso al Banco 1.
	movlw b'00f01000'	; TMR0 como contador por flanco ascendente de
	movwf OPTION_REG	; RA4/T0CKI. Prescaler asignado al Watchdog.
	bcf STATUS,RP0	; Acceso al Banco 0.
	clrf TMR0	; Inicializa contador.

; La sección "Principal" es de mantenimiento. Sólo se dedica a visualizar el Timer 0, cuya

; cuenta se incrementa con los flancos ascendente procedente de la entrada Trigger Schmitt : RA4/T0CKI donde se ha conectado la LDR.

Principal.

```
call    LCD_Lineal      ; Se pone al principio de la línea 1.
movf   TMR0,W           ; Lee el Timer 0.
call    BIN_a_BCD        ; Se debe visualizar en BCD.
call    LCD_Byt          ; Visualiza, apagando las decenas en caso de que sean 0.
goto   Principal
```

```
INCLUDE <RETARDOS.INC>
INCLUDE <BIN_BCD.INC>
INCLUDE <LCD_4BIT.INC>
END
```

Este programa solo permite contar de forma correcta hasta 99. Se deja al lector su mejora para poder aumentar la cuenta y utilizarlo. **por ejemplo, en la computación de los objetos que están en una cinta transportadora, personas que pasan por una puerta, etc.** Este mismo programa se puede utilizar para otros **censores** que se describen **más** adelante, por ejemplo para contar el número de objetos que pasan por delante de un sensor de ultrasonidos.

El circuito de la figura 32-6 junto con el programa Sensor\_LDR\_02.asm constituye otra aplicación **típica**. Se trata de un interruptor crepuscular, que es un circuito para encender una lámpara cuando llega la noche y apagarla con los primeros rayos del sol.

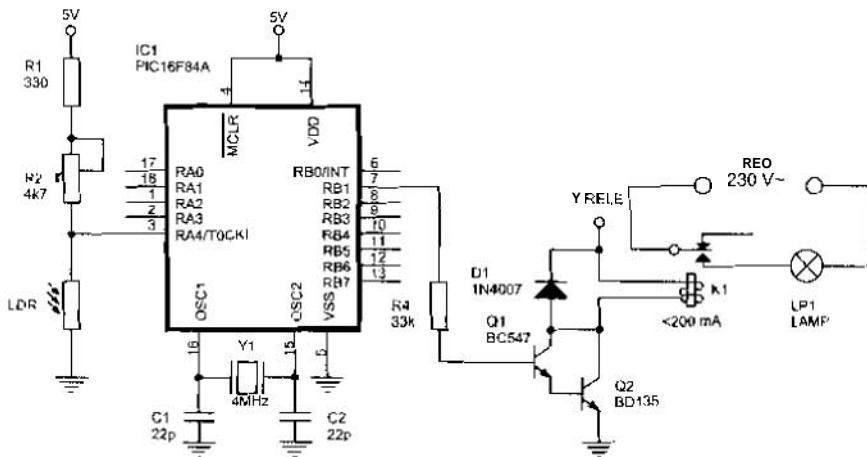


Figura 32-6 Interruptor crepuscular

\*\*\*\*\* Sensor\_LDR\_02.asm \*\*\*\*\*

; Programa de un interruptor crepuscular: una lámpara se encenderá mientras sea ; de noche. Una LDR detectará la luz ambiente (sin que le llegue la luz de la lámpara que ; pretende controlar) y estará conectada a la entrada Trigger Schmitt RA4.

; Cuando la LDR d ; - LDR iluminada ; - LDR en oscurida

; ZONA DE DATO

```
_CONF
LIST
INCLUDE
CBLOCK
ENDC
```

```
#DEFINE Lampara
#DEFINE LDR
```

; ZONA DE CÓDIGO

ORG

```
Inicio
bsf
bsf
bcf
bcf
bcf
```

```
Principal
btfs
goto
```

```
EnciendeLampara
call
btfs
goto
bsf
goto
```

```
ApagaLampara
call
btfc
goto
bcf
call
goto
```

```
Fin
INCLUDE
END
```

## 32.4 FOTOC

Este tipo de ; LED, y un receptor define el tipo ya ;

- ; Cuando la LDR detecte oscuridad el sistema activará una lámpara:
- ; - LDR iluminada → Entrada PIC = "0" → Lámpara apagada.
- ; - LDR en oscuridad → Entrada PIC = "1" → Lámpara encendida.
- ;
- : ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
ENDC
```

- ```
#DEFINE Lampara PORTB,1 ; Línea donde se conecta la salida.
#DEFINE LDR PORTA,4 ; Entrada Trigger Schmitt del PIC donde se conecta
; la LDR.
```

: ZONA DE CÓDIGOS \*\*\*\*\*

|                 | ORG     | 0              |                                                         |
|-----------------|---------|----------------|---------------------------------------------------------|
| Inicio          |         |                |                                                         |
|                 | bsf     | STATUS.RP0     | ; Acceso al Banco 1.                                    |
|                 | bsf     | LDR            | ; Configurada como entrada.                             |
|                 | bcf     | Lampara        | ; Configurada como salida.                              |
|                 | bcf     | STATUS.RP0     | ; Acceso al Banco 0.                                    |
|                 | bcf     | Lampara        | ; En principio lámpara apagada.                         |
| Principal       |         |                |                                                         |
|                 | btfss   | LDR            | ; ¿Entrada=1? , ¿LDR en oscuridad?                      |
|                 | goto    | ApagaLampara   | ; No , LDR iluminada por el sol. Apaga la lámpara.      |
| EnciendeLampara |         |                |                                                         |
|                 | call    | Retad - 20s    | ; Espera este tiempo para confirmar la oscuridad.       |
|                 | btfss   | W R            | ; ¿Entrada=1? , ¿LDR sigue en oscuridad?                |
|                 |         | Fin            | ; No , sale fuera.                                      |
|                 | bsf     | Lampara        | ; SI, enciende la lámpara.                              |
|                 | goto    | Eln            |                                                         |
| ApagaLampara    |         |                |                                                         |
|                 | call    | Retado-20s     | ; Espera este tiempo para confirmar la luz del sol.     |
|                 | btfsc   | LDR            | ; ¿Entrada=0? , ¿LDR sigue iluminada por luz del sol?   |
|                 |         | Fm             | ; No , sale fuera.                                      |
|                 | bcf     | Lampara        | ; Si, apaga lámpara.                                    |
| Fin             |         |                |                                                         |
|                 | call    | Retardo_20s    | ; Permanece en el estado anterior al menos este tiempo. |
|                 | goto    | Principal      |                                                         |
|                 |         |                |                                                         |
|                 | INCLUDE | <RETARDOS.INC> |                                                         |
|                 | END     |                |                                                         |

### 32.4 FOTOLENSORES ACTIVOS

Este tipo de sensores consta de un emisor de luz, que normalmente es un diodo LED, y un receptor, que suele ser un fotodiodo o un fototransistor, la situación de ambos define el tipo ya sea reflexión o barrera como se estudiará a continuación.

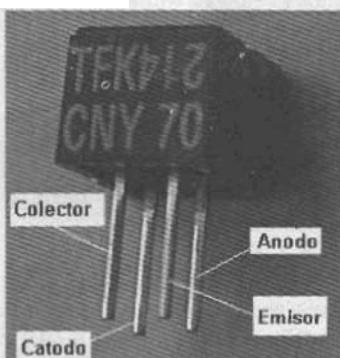
Su aplicación, entre otras, suele ser la detección de la presencia de objetos, medida de distancias muy cortas, lectura de códigos de barras, etc. En microrobótica suele utilizarse para detectar una marca o seguir una línea (normalmente negra sobre fondo blanco).

Funcionamiento:  
 - Detecta Blanco  
 - Detecta Negro

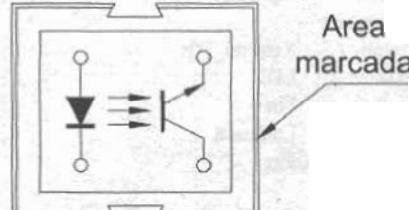
Los problemas más comunes que suelen darse con este tipo de sensores son que la reflexión depende las características del material y del color; en principio los colores más claros reflejan el haz de luz infrarroja más que los oscuros. La luz ambiente es una importante fuente de moho a tener en cuenta.

### 32.4.1 Sensor óptico CNY70

El CNY70 es un sensor óptico reflexivo con salida a transistor (figura 32-7) fabricado por *Vishay Telefunken Semiconductors* ([www.vishay.com](http://www.vishay.com)). Tiene una construcción compacta donde el emisor de luz y el receptor se colocan en la misma dirección para detectar la presencia de un objeto por medio del empleo de la reflexión del haz de luz infrarroja IR (*Infrared*) sobre el objeto. La longitud de onda de trabajo es 950 nm. El emisor es un diodo LED infrarrojo y el detector consiste en un fototransistor. La distancia del objeto reflectante debe estar entre los 5 y 10 mm de distancia. La corriente directa del diodo  $I_F = 50 \text{ mA}$  y la intensidad del colector es de  $I_C = 50 \text{ mA}$ .



Aspecto y patillaje



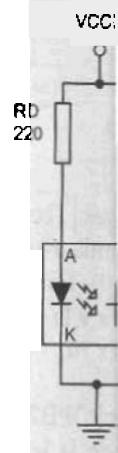
Vista desde arriba

Diagrama interno

Figura 32-7 Sensor óptico reflexivo con salida a transistor CNY70

Para conectar estos dispositivos hay que polarizarlos, ésa es la función de las resistencias del circuito de la figura 32-8, donde se muestran las dos posibles formas de conexión, según se quiera la salida alto para color blanca o negra.

El inversor Trigger Schmitt 40106 se intercala para conformar las tensiones a valores lógicos. Hay que tener en cuenta que los valores de transición de la puerta son  $V_T^+ = 2,9 \text{ V}$  y  $V_T^- = 1,9 \text{ V}$  para una tensión de alimentación de 5V y no podemos variarlos.



El circuito  
Utiliza un ampli  
ha conectado en  
potenciómetro qui

is, medida  
tica suele  
bre fondo

son que la  
colores más  
ante es una

Figura 32-7)  
Tiene una  
la misma  
flexión del  
abajo es 950  
transistor. La  
a corriente

ea  
cada

Y70  
ción de las  
s formas de

tensiones a  
erta son  $V_T^+$   
ariarlos.

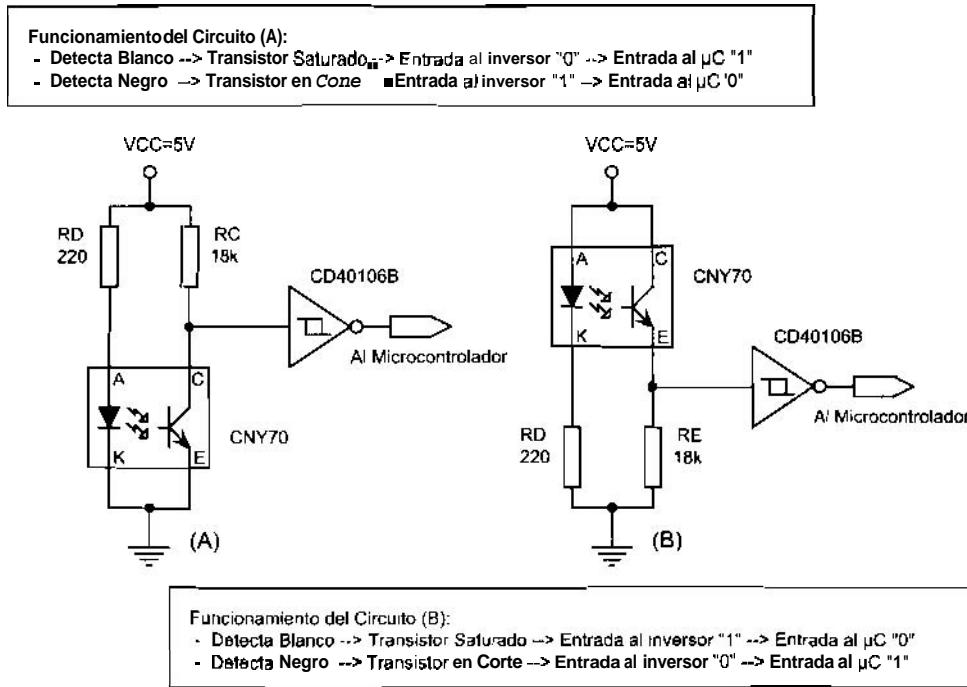


Figura 32-8 Circuitos típicos de conexión del CNY70

El circuito de la figura 32-9 permite ajustar la tensión de disparo del CNY70. Utiliza un amplificador operacional, que trabaja como comparador de tensión, al que se le ha conectado en la entrada no inversora el sensor y a la entrada inversora un potenciómetro que trae como divisor de tensión.

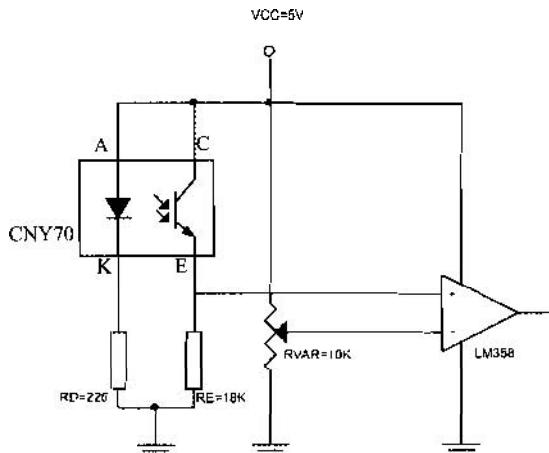


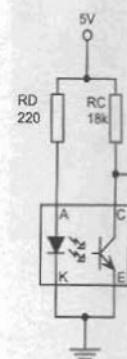
Figura 32-9 Conexión del CNY70 u través de un comparador

Su funcionamiento es el siguiente: si la tensión en la entrada no inversora es mayor que la tensión presente en la entrada inversa, la salida del amplificador operacional toma el valor "1" y si, por el contrario, la tensión en la no inversora es menor que la existente en la entrada inversa, la salida toma un nivel '0'. Es recomendable que la resistencia variable del circuito sea un potenciómetro multivuelta, ya que de esta forma es más sencillo el ajuste.

### 32.4.2 Sensores ópticos OPB703/4/5

Los sensores OPB703, OPB704 y OPB705 de la empresa Optek Technology ([www.optekinc.com](http://www.optekinc.com)) son también del tipo reflexivo y utilizan como emisor un diodo LED emisor de infrarrojos y como receptor un fototransistor (figura 32-10). En este caso están montados sobre unas lentes convergentes alojadas en la carcasa negra y la forma del sensor permite que el haz refleje en una superficie más concreta que el CNY70.

De los tres modelos aquí tratados, el OPB703 no dispone de lente; el OPB704 tiene una lente azul de polisulfuro y el OPB705 permite un desplazamiento de la lente para corregir el offset. En estos dispositivos la lente de polisulfuro elimina gran parte de las interferencias producidas por la luz ambiente.



### 32.4.3 Ejemplo

Para el OPB703, el OPB704 y los OPB705, el circuito con superficie blanca hará que en la pantalla LCD aparezca:

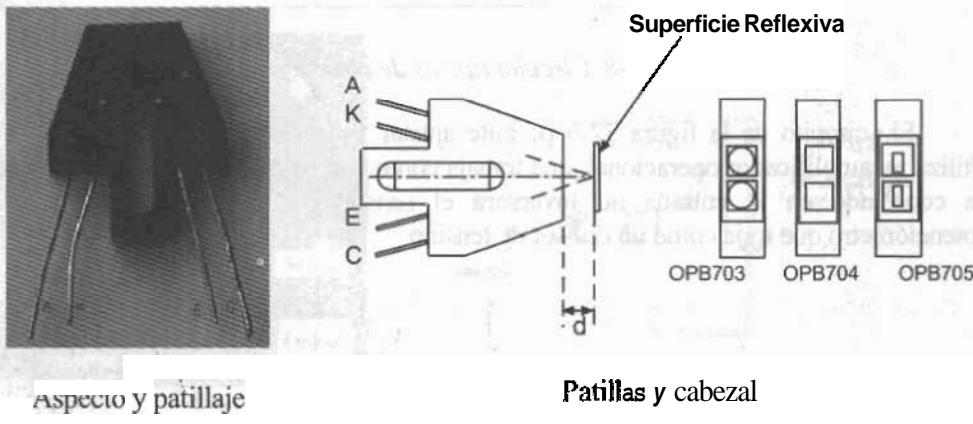


Figura 32-10 Sensores ópticos reflexivos OPB703/4/5

La distancia del objeto reflectante debe estar entre los 4 mm y 10 mm de distancia, La corriente directa máxima del diodo  $I_F = 40 \text{ mA}$  y la intensidad del colector máxima es de  $I_C = 30 \text{ mA}$ .

Los circuitos de aplicación serán los que se han mostrado en las figuras 32-8 y 32-9 pero habrá que modificar las resistencias de polarización teniendo en cuenta la corriente máxima que soportan estos dispositivos.

En pantalla LCD según la config:  
- Color Blanco  
- Color Negro

ZONA DE DATOS

COM  
LIST  
INCL

CBLC  
ENDC

#DEFINE Sense

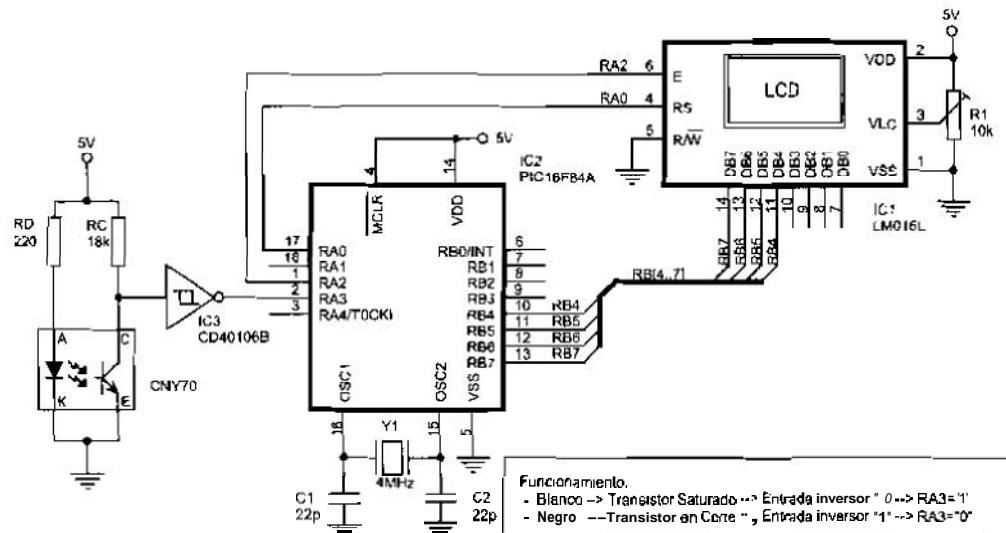


Figura 32-11 Detector de Blanco/Negro

### 32.4.3 Ejemplo de aplicación

Para comprobar el funcionamiento de los dispositivos que hemos analizado CNY70 y OP703/04/05 se puede cargar el programa Sensor\_CNY70\_01.asm y montar el circuito con el sensor de la figura 32-11. Cuando el sensor se encuentra frente a una superficie blanca el haz infrarrojo refleja y en la entrada RA3 aparece un nivel alto que hará que en el LCD visualice el mensaje "Color BLANCO". Si el haz infrarrojo se enfrenta a una superficie de color negro o no encuentra superficie, el haz no refleja y en el LCD aparece el mensaje "Color NEGRO".

\*\*\*\*\* Sensor CNY70\_01.asm \*\*\*\*\*

- En pantalla LCD se visualiza el color "Blanco" o "Negro" que está detectando el sensor CNY70, según la configuración del esquema correspondiente. Si:
  - Color Blanco -> transistor saturado -> entrada al inversor "0" -> RA3 = "1".
  - Color Negro -> transistor en corte -> entrada al inversor "1" -> RA3 = "0".

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC  
LIST    P=16F84A  
INCLUDE <P16F84A.INC>
```

CBLOCK 0x0C  
ENDC

```
#DEFINE Sensor PORTA,3
```

: Líneas donde se conecta el sensor.

; ZONA DE CÓDIGOS \*\*\*\*\*

```

ORG 0
Inicio
    call LCD_Inicializa
    movlw MensajeColor
    call LCD_Mensaje
    bsf STATUS,RP0
    bsf Sensor ; Linea del sensor se configura como entrada.
    bcf STATUS,RP0

Principal
    call LCD_Linea2
    movlw MensajeNegro ; En principio considera que es negro.
    btfsc Sensor ; Lee el sensor.
    movlw MensajeBlanco ; No, es blanco.
    call LCD_Mensaje ; Visualiza el resultado.
    goto Principal

; "Mensajes"
;
Mensajes
    addwf PCL,F
MensajeColor
    DT " Color ",0x00
MensajeBlanco
    DT " BLANCO",0x00
MensajeNegro
    DT " Negro ",0x00

INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
END

```

### 32.4.4 Sensor óptico de barrera H21A1

El representante más popular de este tipo de sensores es el H21A1 (figura 32-12) fabricado entre otros por *Isocom Components* ([www.isocom.com](http://www.isocom.com)) y *Fairchild Semiconductors* ([www.fairchildsemi.com](http://www.fairchildsemi.com)).

Estos sensores también tienen como emisor un diodo de infrarrojos y como receptor un fototransistor. En este caso el emisor y el receptor están enfrentados a una distancia de 3 mm y entre ellos existe un espacio para que un objeto pueda introducirse y romper la barrera infrarroja.

El circuito de aplicación para este tipo de dispositivos es similar al de la figura 32-8. Los valores de las resistencias de polarización deben limitar la corriente por el diodo emisor  $I_F$  a 60 mA y por el colector del transistor a una corriente  $I_C$  no superior a 20 mA. El fabricante facilita los valores de prueba indicados en la tabla 32-2.

Aspecto

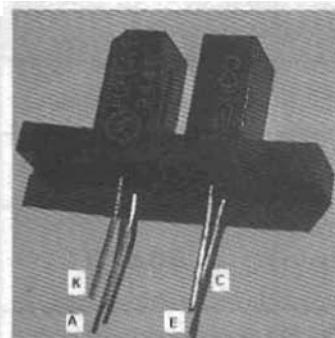
Con  
cole  
Tran  
satur  
Ten  
satu

Para c  
los progra  
el sensor ó  
Sensor\_LDR  
el haz infrarr

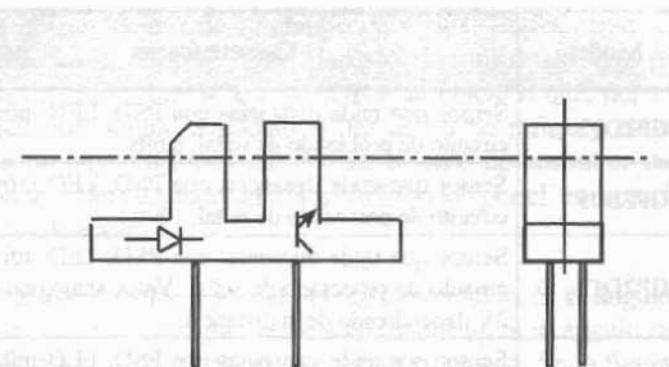
Una ap  
se acopla al e  
que pasan po  
luz en la unid

32.5 SE

Los IR  
alta sensibili  
Corporation  
muestran en



Aspecto y patillaje



Posición del emisor y del receptor en el dispositivo

Figura 31-12 Sensor óptico de barrera H21A1

| Corriente de colector<br>Transistor saturado | Condiciones de test                          | Símbolo       | Valor Mínimo |
|----------------------------------------------|----------------------------------------------|---------------|--------------|
| $I_F = 5\text{mA}$ , $V_{CE} = 5\text{V}$    | $I_{C(ON)}$                                  | $I_{C(ON)}$   | 0,15 mA      |
| $I_F = 20\text{mA}$ , $V_{CE} = 5\text{V}$   | $I_{C(ON)}$                                  | $I_{C(ON)}$   | 1,0 mA       |
| $I_F = 30\text{mA}$ , $V_{CE} = 5\text{V}$   | $I_{C(ON)}$                                  | $I_{C(ON)}$   | 1,9 mA       |
| Tensión de saturación                        | $I_F = 30\text{ mA}$ , $I_C = 1,8\text{ mA}$ | $V_{CE(SAT)}$ | 0,40V        |

Tabla 32-2 Valores de test para distintos valores de  $I_F$ 

Para comprobar el funcionamiento de estos sensores se pueden utilizar muchos de los programas propuestos en temas anteriores sustituyendo pulsadores por el circuito con el sensor óptico. Por ejemplo las programas `Int_int_01.asm`, `Int_int_02.asm` o `Sensor_LDR_01.asm`, que presentarán en un LCD el número de veces que se interrumpe el haz infrarrojo del sensor.

Una aplicación típica de estos sensores es medir la velocidad de un motor. Para ello se acopla al eje del motor un encoder, que es una lámina circular con una serie de ranuras que pasan por el centro del sensor y cortan el haz de luz, el número de cortes del haz de luz en la unidad de tiempo será proporcionada la velocidad del motor.

## 32.5 SENSORES INFRARROJOS GP2DXX

Los IR Sharp GP2DXX son una familia de sensores de infrarrojos compactos de alta sensibilidad para la detección y medida de distancia. En la página Web de Sharp Corporation (<http://sharp-world.com>) encontramos algunos de los modelos que se muestran en la tabla 32-3.

| Modelo   | Características                                                                                                                                                           | Rango de distancia de medida en mm          |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|
| GP2D02   | Sensor que mide distancias con PSD, LED infrarrojo y circuito de procesado de señal, 8 bits.                                                                              | 100 a 800                                   |
| GP2D05   | Sensor que mide distancias con PSD, LED infrarrojo y circuito de procesado de señal, 1 bits.                                                                              | 100 a 800                                   |
| GP2D12   | Sensor que mide distancias con PSD, LED infrarrojo y circuito de procesado de señal. Valor analógico entre 0-3V dependiendo de la distancia.                              | 100 a 800                                   |
| GP2D120  | Sensor que mide distancias con PSD, LED infrarrojo y circuito de procesado de señal. Valor analógico entre 0-3V dependiendo de la distancia.                              | 40 a 300                                    |
| GP2D15   | Sensor que mide distancias con PSD (Detector Sensible a la Posición), LED infrarrojo y circuito de procesado de señal, valor digital (0 o 1) ajustado de fábrica a 24 cm. | 100 a 800                                   |
| GP2D150A | Sensor que mide distancias con PSD, LED infrarrojo y circuito de procesado de señal. Valor digital (0 ó 1).                                                               | 30 a 300 (Detección distancia típica 15 cm) |

Tabla 32-3 Sensores de la serie GP2Dxx

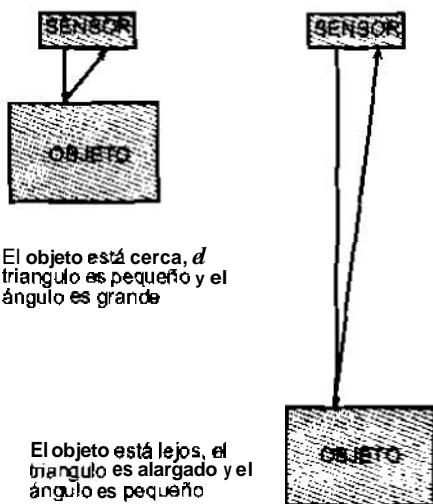


Figura 32-13 Concepto de medida por triangulación

### 32.5.1 Principio de funcionamiento

Estos dispositivos emplean el método de triangulación, utilizando un pequeño Sensor Detector de Posición lineal (*PSD, Position Sensitive Detector*) para determinar la

distancia o la modo de función transmite a través contrario, no lectura que se obtáculo el ha punto de reflexión

La información es grande, entero pequeño, quiere 32-13 podemos

Seguidamente

### 32.5.2 GP

El Sharp mediante una sonda (figura 32-14). Es variable que incluye

Figura

El sensor microcontrolador la entrada de control alto no ha detectado alcance al que se

distan<sup>c</sup>a o la presencia de los objetos dentro de su campo de visi<sup>o</sup>n. Básicamente su modo de funcionamiento consiste en la emisi<sup>o</sup>n de un pulso de luz infrarroja, que se transmite a tráves de su campo de visi<sup>o</sup>n, que se refleja contra un objeto o que, por el contrario, no lo hace. Si no encuentra ningún obstáculo el haz de luz no refleja y en la lectura que se hace indica que no hay ningún obstáculo. En el caso de encontrar un obstáculo el haz de luz infrarroja se refleja creando un triángulo formado por el emisor, el punto de reflexión (obstáculo) y el detector.

La informaci<sup>o</sup>n de la distancia se extrae midiendo el ángulo recibido. Si el ángulo es grande, entonces el objeto est<sup>a</sup> cerca, el triángulo es pequeño y ancho. Si el ángulo es pequeño, quiere decir que el objeto est<sup>a</sup> lejos, el triángulo es largo y delgado. En la figura 32-13 podemos ver una representaci<sup>o</sup>n de estos conceptos.

Seguidamente describiremos el modo de manejo de algunos de estos dispositivos.

### 32.5.2 GP2D05

El Sharp GP2D05 es un sensor capaz de medir distancias por infrarrojos e indica mediante una salida lógica (0 ó 1) si hay algún objeto dentro de un alcance preestablecido (figura 32-14). El rango se ajusta entre 10 y 80 cm con la ayuda de una resistencia variable que incluye el propio dispositivo, que es fácil de regular.



Figura 32-14 Vista de un GP2D05, (cortesia de [www.superrobotica.com](http://www.superrobotica.com))

El sensor utiliza sólo una línea de entrada y otra de salida para comunicarse con el microcontrolador. Su utilización es tan sencilla como mandar un impulso a nivel bajo en la entrada de control ( $V_i$ ), esperar entre 28 y 56 ms y leer el estado de  $V_o$ , si esta a nivel alto no ha detectado obstáculo, si está a nivel bajo ha detectado un obstáculo dentro del alcance al que se ha ajustado.

## 32.5 GP2D15

El Sharp GP2D15 es un sensor medidor de distancias por infrarrojos que indica mediante una salida digital si hay un objeto a menos de  $24 \pm 3$  cm (figura 32-15). La detección se hace de forma continua, esto significa que no es necesario ningún tipo de circuito de control ni temporización externa. Basta con aplicar tensión para que la medida esté disponible cada 50 ms. El sensor utiliza una única línea de salida para comunicarse con el microcontrolador.

De características similares a este dispositivo, Sharp ha sacado recientemente al mercado el GP2Y0D21YK.

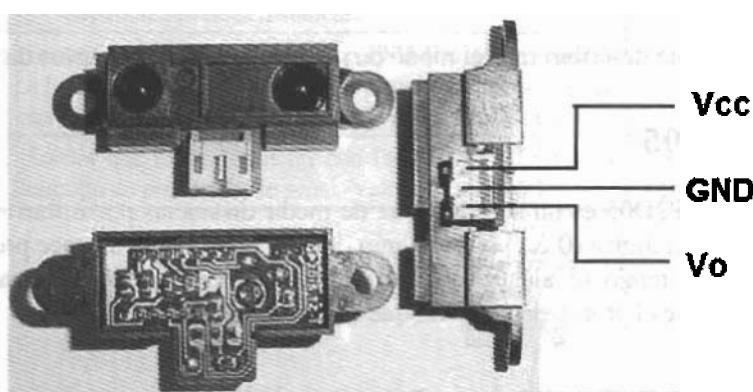


Figura 32-15 Vistas de un GP2D15 o GP2D12

### 32.5.4 GP2D12

El Sharp GP2D12 es un sensor que mide distancias por infrarrojos. El dispositivo indica mediante una salida analógica la distancia medida al objeto sobre el que refleja el haz de luz. La tensión de salida varía de forma no lineal cuando se detecta un objeto en una distancia entre 10 y 80 cm, tal y como se aprecia en la curva de la figura 32-16. La salida está disponible de forma continua y su valor es actualizado cada 32 ms. Normalmente se conecta esta salida a la entrada de un convertidor analógico digital, el cual convierte la distancia en un número binario que es utilizado por el microcontrolador. La salida también puede ser usada directamente en un circuito analógico. El sensor utiliza sólo una línea de salida para comunicarse con el microcontrolador. Su margen de medida es de 10 a 80 cm.

De características similares a este dispositivo, Sharp ha sacado recientemente al mercado el GP2Y0A21YK.

32.6 R

32.6.1 D

El SI  
infrarroja n  
negro esta c

Este  
un preampl  
paso banda.

Se co  
la portadora

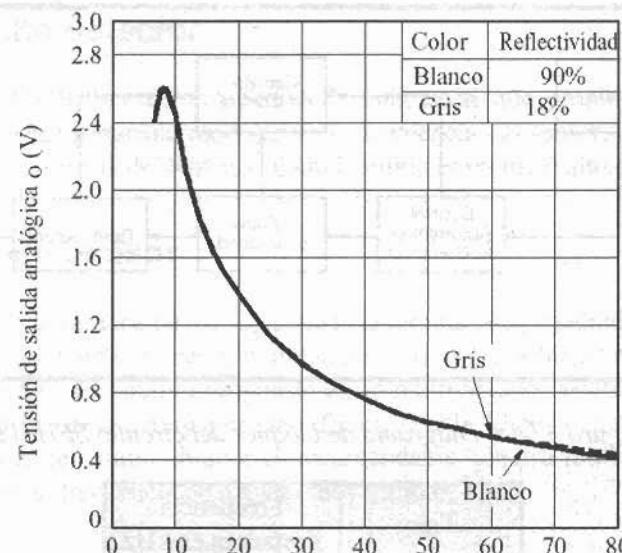


Figura 32-16 Función de transferencia del GP2D12

## 32.6 RECEPTOR PARA CONTROL REMOTO SFH5110

### 32.6.1 Descripción

El SFH5110 es un receptor de IR (*Infrared*) utilizado para detectar un haz de luz infrarroja modulada en sistemas de telemando (figura 32-17). Su encapsulado en color negro está diseñado con un filtro de corte para la luz del día.

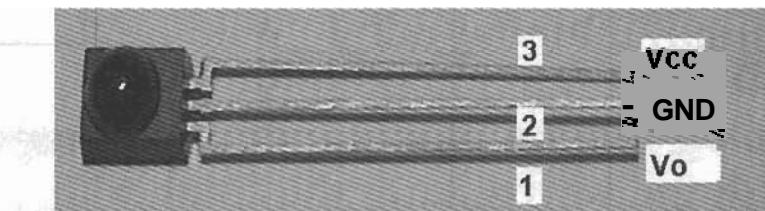


Figura 32-17 Vista y patillaje del SFH5110

Este circuito integrado incluye un fotodiodo sensible al haz infrarrojo de 940 nm, un preamplificador, un control de ganancia automática, un demodulador y varios filtros paso banda, como puede apreciarse en el diagrama de bloques de la figura 32-18.

Se comercializan distintos modelos de este sensor dependiendo de la frecuencia de la portadora a utilizar, como puede comprobarse en la tabla 32-4.

### 32.6.2 Circuito

La figura muestra el diagrama de bloques del circuito SFH5110. El sensor PIN recibe una señal de la patilla 3 y la envía al controlador. El controlador también recibe una señal de la patilla 1 y controla el circuito de control. El circuito de control envía una señal a la patilla 2, que es la salida. La salida es una señal digital que se conecta a un transistor Darlington para proporcionar una señal de alta potencia.

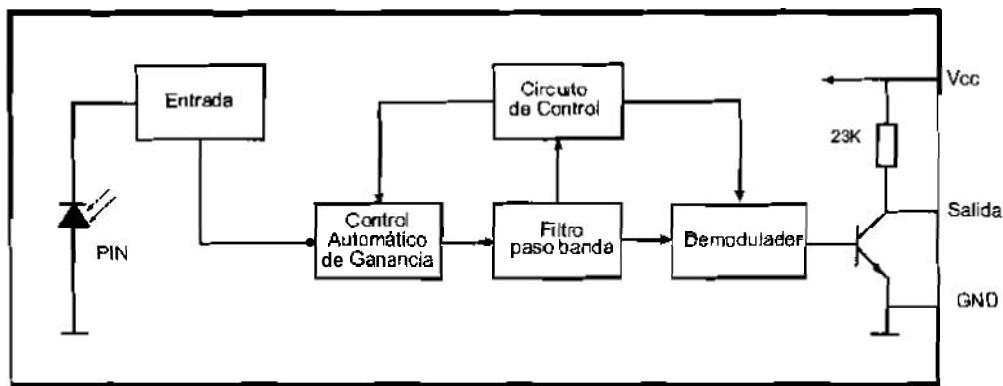
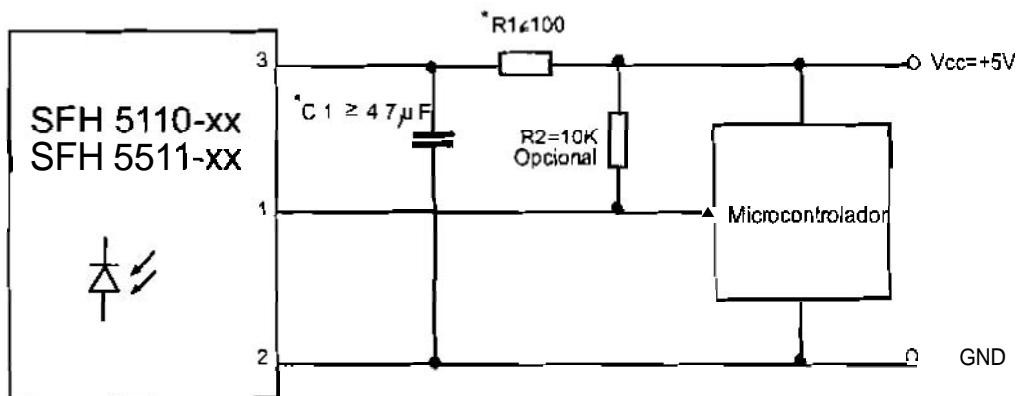


Figura 32-18 Diagrama de bloques del circuito SFH5110

| Tipo        | Frecuencia Portadora en kHz |
|-------------|-----------------------------|
| SFH 5110-30 | 30                          |
| SFH 5110-33 | 33                          |
| SFH 5110-36 | 36                          |
| SFH 5110-38 | 38                          |
| SFH 5110-40 | 40                          |

Tabla 32-4 Frecuencia de portadora de los distintos modelos de sensores

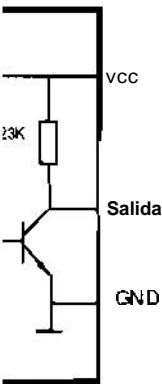


(\*) R1 y C1 sólo son necesarias para eliminar las perturbaciones de la fuente de alimentación

Figura 32-19 Circuito receptor de haz infrarrojo modulado

### 32.7 SENSORES

El IS47 es capaz de detectar una señal de infrarrojos emitida por un sensor de infrarrojos.



### 32.6.2 Circuito detector

La figura 32-19 muestra el circuito de aplicación que proponemos. Cuando se detecta una haz de luz infrarroja modulada a la frecuencia del sensor, pone un nivel bajo en la patilla  $V_o$ , en ausencia del haz modulado la salida es un nivel alto.

### 32.6.3 Circuito emisor

El circuito emisor debe de ser capaz de generar una onda cuadrada a ser posible con un ciclo de trabajo del 50%, es decir, la mitad del periodo la señal estará a nivel bajo y la otra mitad a nivel alto. El circuito propuesto es un multivibrador astable realizado con un circuito integrado 555 de uso muy corriente (figura 32-20). El potenciómetro con el que se ajusta la frecuencia del multivibrador es recomendable que sea multivuelta, para poder realizar el ajuste de la frecuencia de manera más cómoda.

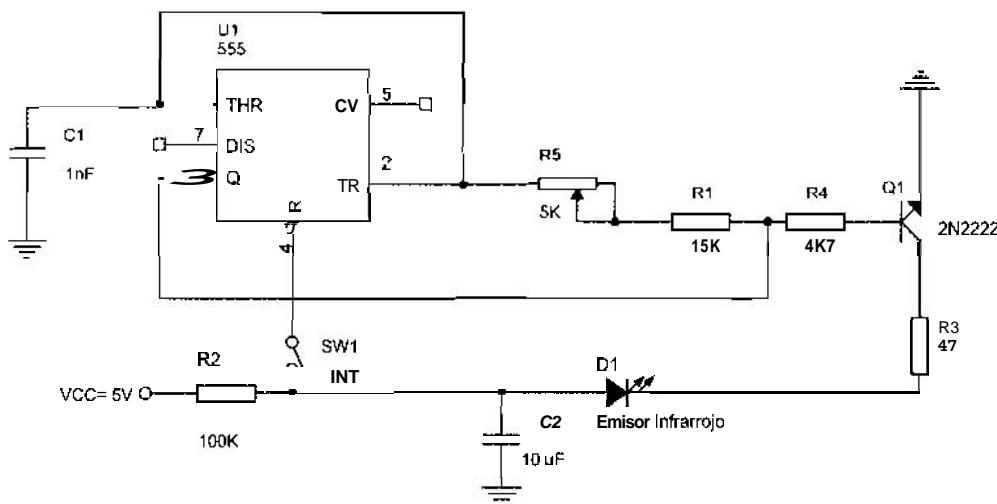


Figura 32-20 Circuito emisor de haz infrarrojo modulado 36-40 kHz

Si se disminuye el valor de la resistencia R3 hasta hacer pasar por un diodo emisor de infrarrojos SFH 4510/SFH 4515 una corriente de 500 mA, se puede llegar a recibir la señal a una distancia de 30 m. El transistor 2N2222 se puede sustituir por un par Darlington BD135 y BC547 utilizado en otros esquemas (por ejemplo en el 32-6).

## 32.7 SENSOR DE PROXIMIDAD IS471F

El IS471F fabricado por Sharp Corporation es un detector de obstáculos infrarrojo que es capaz de detectar cuando se refleja sobre un objeto el haz que emite un fotodiodo

emisor de infrarrojos que se le conecta tal y como se muestra en el circuito de la figura 32-23.

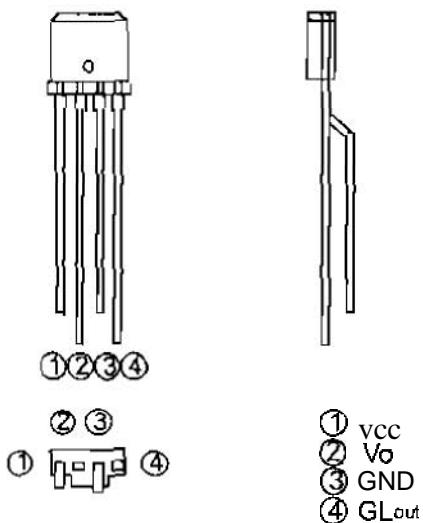


Figura 32-21 Patillaje del IS471F

Estos dispositivos son inmunes a las perturbaciones de luces externas debido al sistema de modulación de luz que llevan incorporado. El propio circuito incluye el driver de pulsos y sincronismo, como puede verse en el diagrama de bloques de la figura 32-22. El circuito detector y emisor tiene un amplio margen de tensión de alimentación que va desde 4,5 a 16V.

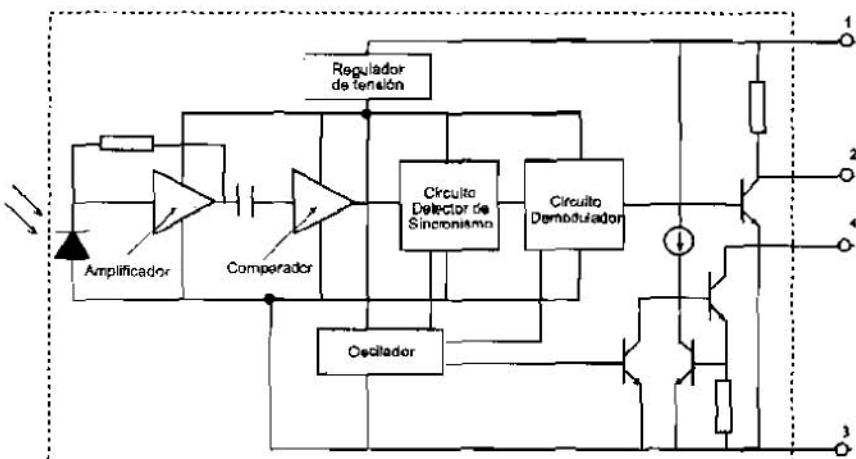
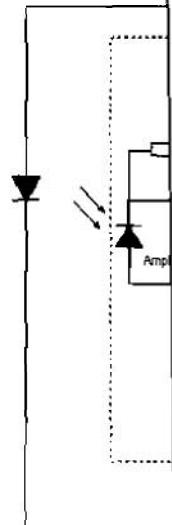


Figura 32-22 Diagrama de bloques del IS471F

El circuito que se pueden contacto físico objeto sobre el patilla de salida próximo. Si se salida se obtien



En este conectar un con

## 32.8 BUN

Los dete uno o varios mecánicos com control de nive robatica conven

Los bum conmutador qu metal que hace que dispone de longitudes de lá

gura 32-

El circuito práctico de aplicación que proponemos es el de la figura 32-23, con el que se pueden detectar objetos situados a una distancia máxima de 70 mm sin que haya contacto físico con los mismos, la distancia puede disminuir dependiendo del color del objeto sobre el que reflecta el haz infrarrojo. Cuando se alimenta el circuito a 5V, en la patilla de salida (V<sub>O</sub>) aparece un nivel alto con una tensión de 5V si no hay ningún objeto próximo. Si se coloca un objeto en el radio de acción del dispositivo, en la patilla de salida se obtienen 0 V y por lo tanto un nivel bajo.

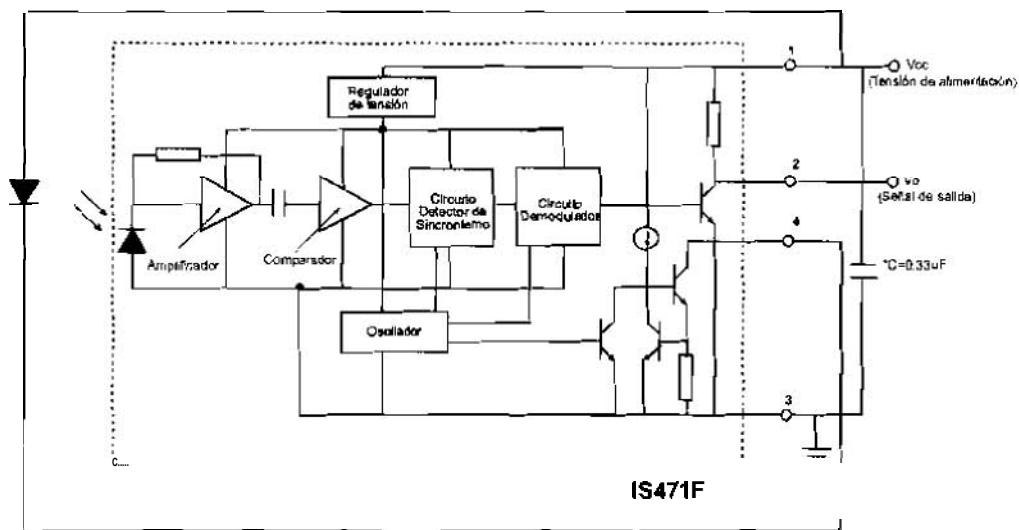


Figura 32-23 Circuito básico de aplicación del IS471F

En este circuito para estabilizar la tensión de alimentación es recomendable conectar un condensador de filtro  $\geq 0.33\mu F$ , entre V<sub>CC</sub> y GND cerca del dispositivo.

## 32.8 BUMPERS

Los detectores de obstáculos mecánicos son unos dispositivos que abren o cierran uno o varios circuitos eléctricos. En el mercado existen innumerables detectores mecánicos como pueden ser pulsadores, interruptores: sensores de presión, boyas para control de nivel de agua, etc. Nosotros trataremos de los finales de carrera que en la robótica convencional y microrobótica reciben el nombre de **bumpers** (figura 3-23).

Los bumpers como puede verse en el diagrama de su representación eléctrica, es un commutador que cambia de posición al realizar la presión necesaria sobre la lámina de metal que hace de brazo de una palanca de primer orden que a su vez activa un pulsador que dispone de un muelle de recuperación. Se comercializan bumpers con diferentes longitudes de lámina según la aplicación a utilizar.

## 32.9 DETECCIÓN DE OBSTACULOS

Los ultrasonidos son sensores de proximidad que emiten un pulso de frecuencia acústica de alta intensidad y detectan la señal reflejada por un objeto. Los detectores de proximidad utilizan un dispositivo de detección que activa el sensor cuando detecta algún objeto, a través de la retroalimentación de la señal reflejada. Los detectores tienen un tiempo que tardan en activarse y desactivarse, dependiendo del tipo de sensor.

EL SRF04 es un sensor de proximidad de tipo ultrasonido fabricado por la empresa Sharp Corporation Ltd. (www.robot-electronics.co.uk). El sensor SRF04 mide la distancia entre el sensor y un objeto con una precisión de ± 3 mm y una velocidad de respuesta de 50 ms. El sensor SRF04 se encuentra en un caso compacto y tiene un bajo consumo y gran alcance.

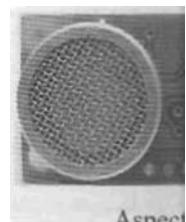
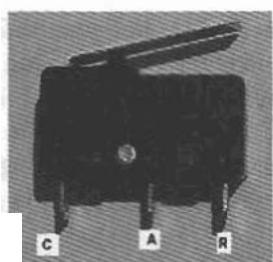


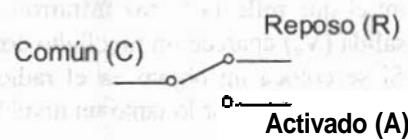
Figura 32-26 Módulo SRF04

Este sensor fué diseñado para ser usado en robots y vehículos autónomos. Se utiliza para medir la distancia entre el sensor y un objeto. La anchura del pulso de salida indica la distancia medida.

El sensor SRF04 emite pulsos de ultrasonido que viajan a la velocidad de la luz. Los pulsos reflejados por los objetos cercanos son captados por el sensor y convertidos en señales digitales que se incorporan al microcontrolador para emitir órdenes de control.



Final de carrera



Representación de un final de carrera

Figura 32-24 Final de carrera de tipo Bumper

Al activar el brazo de la palanca, el terminal común C realiza un contacto eléctrico con el terminal activado A. Cuando no se activa la palanca el terminal C está en contacto con el terminal de reposo R.

En temas anteriores se ha utilizado siempre sistemas antirrebote software para los pulsadores. Un sistema antirrebote hardware se muestra en la figura 32-25, se trata de un flip flop RS que presenta en las salidas Q1 y Q2 unas señales complementarias. En el estado de reposo la salida Q1 tiene un nivel alto y la salida Q2 un nivel bajo y al activar el final de carrera la salida Q1 pasará a tener un nivel bajo mientras que la salida Q2 pasará a nivel alto.

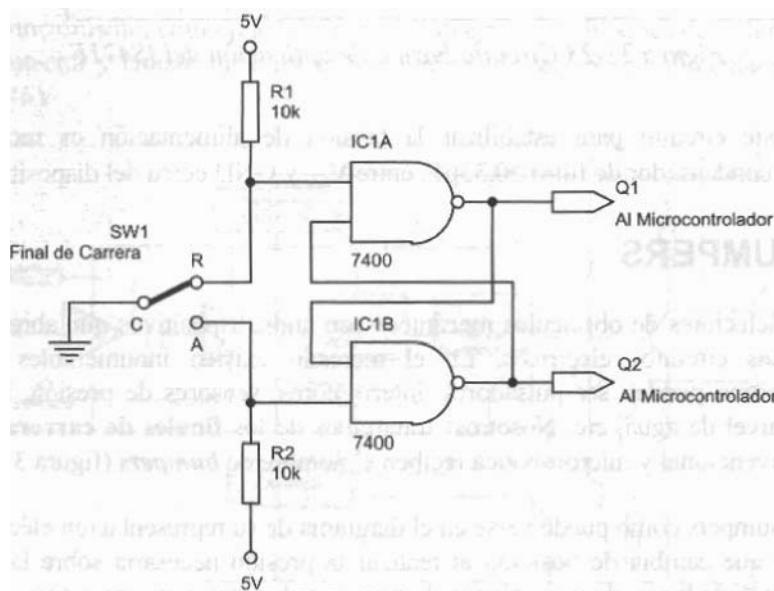


Figura 32-25 Conexión del final de carrera a través de un circuito antirrebote

## 32.9 DETECTOR POR ULTRASONIDO SRF04

Los ultrasonidos son vibraciones del aire de la misma naturaleza que el sonido, pero de frecuencia superior a los 20 kHz por lo que no son audibles por los seres humanos. Los detectores de obstáculos por ultrasonidos emiten pulsos de ultrasonido mediante un dispositivo transmisor, cuando las ondas ultrasónicas se reflejan sobre algún objeto, a través de una cápsula sensible se captan los pulsos reflejados. El tiempo que tardan en volver los pulsos reflejados es proporcional a la distancia del objeto sobre el que se reflejan.

EL SRF04 es un módulo de sensores por ultrasonidos, desarrollado por *Devantech Ltd* ([www.robot-electronics.co.uk](http://www.robot-electronics.co.uk)) y comercializado en España por Intplus S.L ([www.superrobotica.com](http://www.superrobotica.com)), que es capaz de detectar objetos y calcular la distancia a la que se encuentra en un rango de 3 a 300 cm. De muy pequeño tamaño, el SRF04 destaca por su bajo consumo y gran precisión. Su aspecto físico y conexiones se muestran en la figura 32-26.

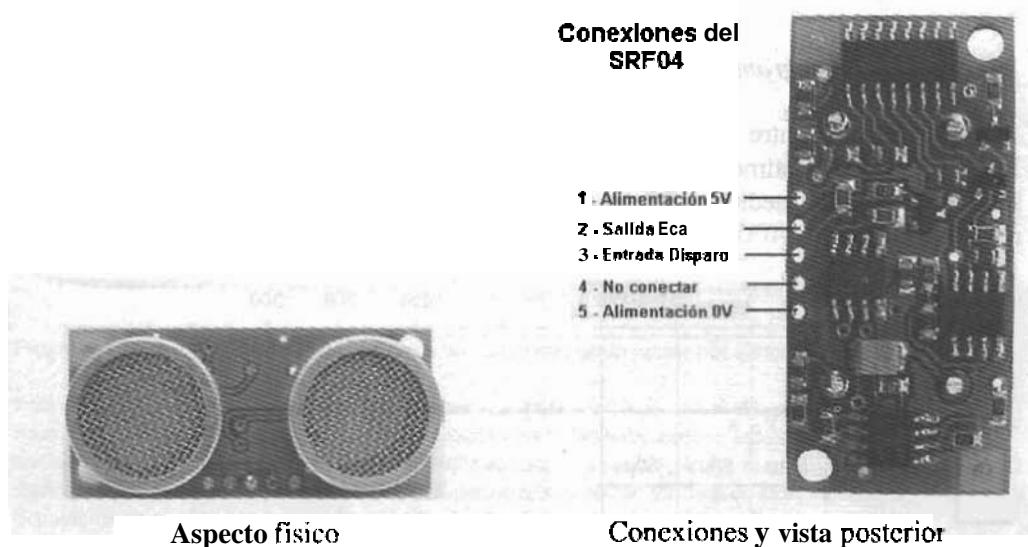


Figura 32-26. Módulo de ultrasonidos SRF04, (cortesía de [www.superrobotica.com](http://www.superrobotica.com))

Este sensor funciona por ultrasonidos y contiene toda la electrónica encargada de hacer la medición. Su uso es tan sencillo como enviar el pulso dc arranque y medir la anchura del pulso de retorno tal como se muestra en la figura 32-27.

El sensor SRF04 funciona emitiendo impulsos de ultrasonidos. Los impulsos emitidos viajan a la velocidad del sonido hasta alcanzar un objeto, entonces el sonido es reflejado y captado de nuevo por el receptor de ultrasonidos. Lo que hace el controlador incorporado es emitir una ráfaga de impulsos, a continuación empieza a contar el tiempo

que tarda en llegar el eco. Este tiempo se traduce en un pulso de eco de anchura proporcional a la distancia a la que se encuentra el objeto.

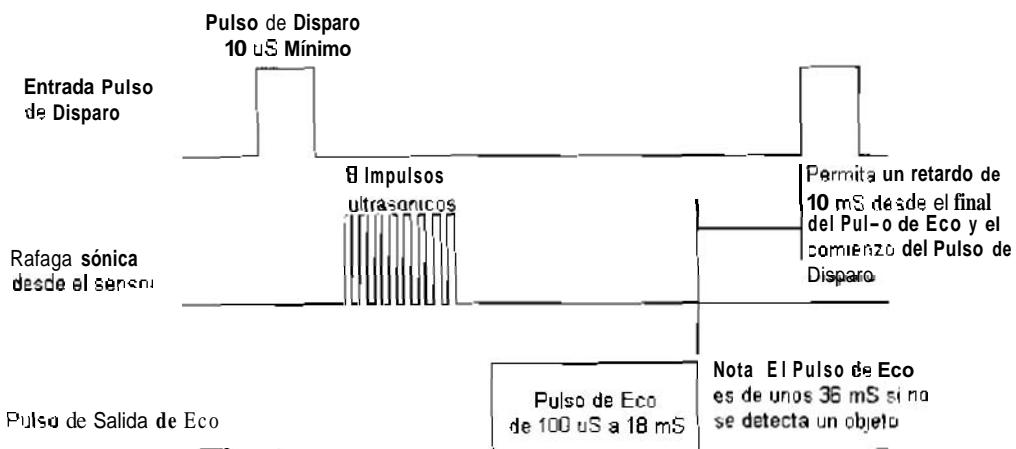


Figura 32-27 Diagramas de tiempos del SRF04, (cortesía de [www.superrobotica.com](http://www.superrobotica.com))

La relación entre la distancia medida y el ancho del pulso se muestra en la figura 32-28. Por cada centímetro de distancia la anchura del pulso se incrementa en 60  $\mu$ s, siendo el rango de medida válido entre 180  $\mu$ s y 18 m, correspondientes a 3 y 300 cm respectivamente.

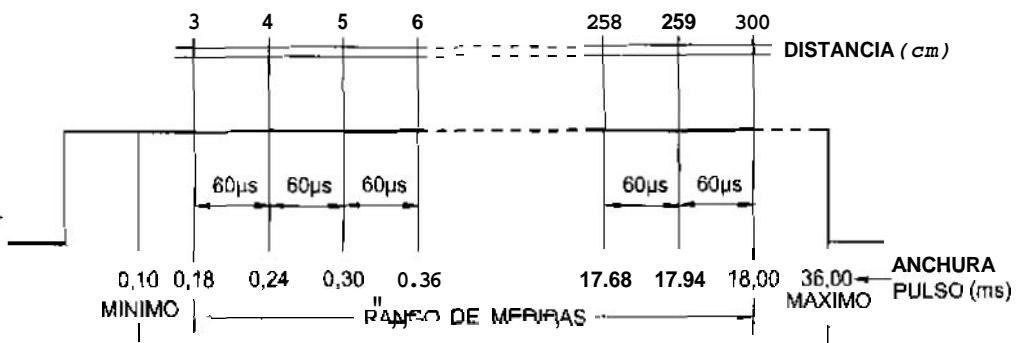
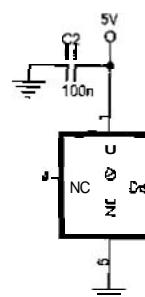


Figura 32-28 Relación entre la distancia medida y la anchura de pulso

Una aplicación típica puede ser un medidor de distancias, como el de la figura 32-29, donde en el display se visualiza la distancia desde el sensor a un objeto.

El programa Sensor\_Ultrasonido\_01.asm, puede utilizarse para comprobar el funcionamiento del sensor. La técnica de programación consiste en mandar un impulso a nivel alto de disparo con una duración mínima de 10  $\mu$ s al pin 3 del SRF04 (entrada

disparo). Des-  
longitud, que



\*\*\*\*\*  
; Programa para u  
;  
; Para el control de  
; linea RA3 que se  
; sensor se ponga t  
; interrupciones po  
; Seguidamente se  
; centímetros.  
;  
; Por cada centíme  
; En este programa  
;  
; ZONA DE DATO

\_CON  
LIST  
INCLUI

CBLOC  
Distanci  
ENDC

anchura  
disparo). Después se lee el pulso que sale por la patilla de ECO pin 2 y se mide su longitud, que es proporcional al eco recibido.

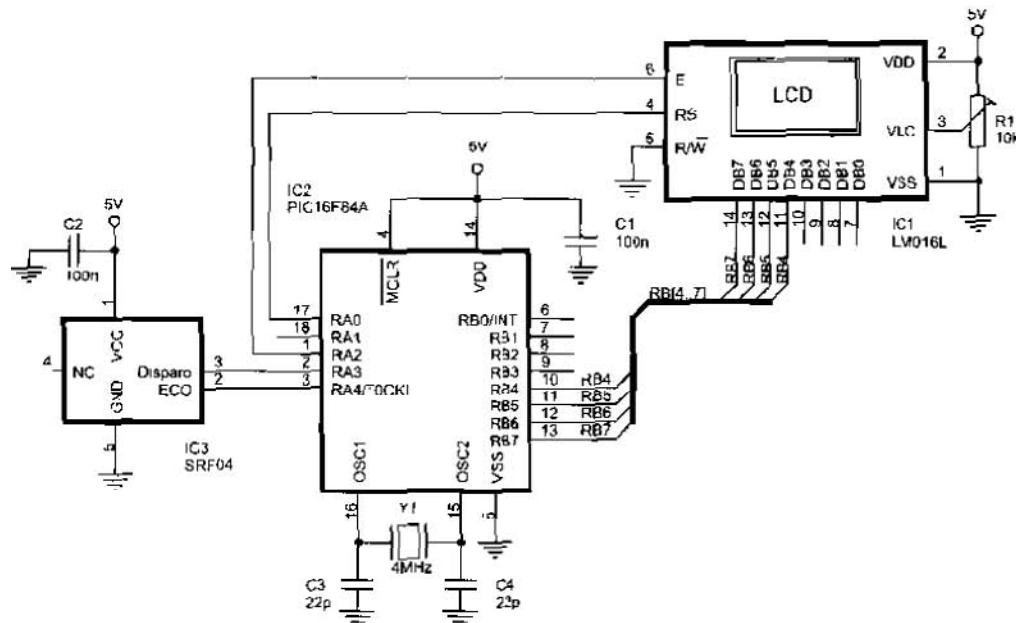


Figura 32-29 Medidor de distancias con SRF04

\*\*\*\*\* Sensor\_Ultrasonido\_01.asm \*\*\*\*\*

; Programa para un medidor de distancias hasta un objeto utilizando sensor por ultrasonido SRF04.

; Para el control del sensor en primer lugar se genera un pulso de 10 µs a nivel alto por la  
; línea RA3 que se conecta a la entrada de disparo del sensor. Seguidamente se espera a que en el  
; sensor se ponga un nivel alto en la salida ECO que se conecta a la línea RA4 y se utilizan las  
; interrupciones por desbordamiento del Timer 0 para medir el tiempo que está en alto el pulso.  
; Seguidamente se visualiza en el módulo LCD el valor de la distancia hasta al objeto expresada en  
; centímetros.

; Por cada centímetro de distancia al objeto el SRF04 aumenta 60 µs la anchura del pulso.  
; En este programa la distancia mínima es 3 cm y la máxima 250 cm.

; ZONA DE DATOS \*\*\*\*\*

```
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
CBLOCK 0x0C
Distancia
ENDC
```

; Se expresará en centímetros.

figura 32-

probar el  
impulso a  
(entrada

```

#define Disparo PORTA,3      ; Disparo para iniciar la medida.
#define Eco      PORTA,4      ; Pulso cuya anchura hay que medir.

MinimaDistancia EQU .3
MaximaDistancia EQU .250
TMR0_Carga60micros EQU -d'27' ; Valor obtenido experimentalmente con la
                                ; ventana Stopwatch para una interrupción del
; Timer0 cada 60 µs. Si no mide correctamente por las tolerancias de los componentes habrá
; que hacer un ajuste fino de este valor, comprobándolo sobre las condiciones reales.

; ZONA DE CÓDIGOS ****
ORG 0
goto Inicio
ORG 4
goto ServicioInterrupcion

Mensajes
    addwf PCL,F
MensajeDistancia
    DT " Distancia: ",0x00
MensajeCentimetro
    DF "cm", 0x00
MensajeDistanciaMenor
    DT "Dist. MENOR de:",0x00
MensajeDistanciaMayor
    DT "Dist. MAYOR de:",0x00
Inicio
    call LCD_Inicializa
    bsf STATUS,RPO
    bcf Disparo
    bsf Eco
    movlw b'00000000'
    movwf OPTION_REG
    bcf STATUS,RPO
    bcf Disparo
; Prescaler de 2 para el TMR0.

Principal
    clrf Distancia
    bsf Disparo
    call Retardo_20micros
    bcf Disparo
; Inicializa línea de disparo en bajo.

Espera_Eco_1
    btfss Eco
    goto Espera_Eco_1
    movlw TMR0_Carga60micros
    movwf TMR0
    movlw b'10100000'
    movwf INTCON
; Si ECO=0, espera el flanco de subida de la señal
; de salida del censor.
; Ya se ha producido el flanco de subida
; Carga el Timer0.
; Autoriza interrupción del TMR0 (TOIE).

Espera_Eco_0
    btfsc Eco
    goto Espera_Eco_0
    clrf INTCON
    call Visualiza
; Espera flanco de bajada de la señal de la salida
; del SRF04.
; Se ha producido el flanco de bajada. Prohibe interrup.
; Visualiza la distancia.

```

call  
 fin  
 gote  
 ; Subrutina "S"  
 ; Se ejecuta de  
 ; de la anchura  
 ; valor de la di  
 ; ServicioIntern  
 mov  
 mov  
 mov  
 addr  
 mov  
 bfrs  
 mov  
 bcf  
 retfi  
 ; Subrutina "V"  
 ; Visualiza la d  
 ; visualizar un  
 ; Y cuando sea  
 ; Si la distanci  
 ; Visualiza  
 call  
 mov  
 subv  
 btfss  
 goto  
 mov  
 subf  
 btfsc  
 goto  
 ; DistanciaMayo  
 mov  
 mov  
 mov  
 goto  
 ; DistanciaMenor  
 mov  
 mov  
 mov  
 goto  
 ; DistanciaFiable  
 mov  
 VisualizaDista

Fin      call      **Reíardo-2s** ; Espera un tiempo hasta la próxima medida.  
           goto      Principal

• Subrutina "ServicioInterrupcion"

; Se ejecuta debido a la petición de interrupción del Timer 0 cada 60  $\mu$ s que es el incremento  
   ; de la anchura de pulso por centímetro de distancia medida. La variable "Distancia" contiene el  
   ; valor de la distancia expresada en centímetros.

**ServicioInterrupcion**

|        |                    |                                                      |
|--------|--------------------|------------------------------------------------------|
| movlw  | TMR0_Carga60micros | ; Carga el Timer 0.                                  |
| movwf  | TMR0               |                                                      |
| movlw  | .1                 | ; Se utiliza instrucción "addwf", en lugar de "incf" |
| addwf  | Distancia,F        | ; para posicionar flag de Carry.                     |
| movlw  | MaximaDistancia    | ; En caso de desbordamiento carga su máximo valor.   |
| btfsc  | STATUS,C           |                                                      |
| movwf  | Distancia          |                                                      |
| bcf    | INTCON,T0IF        |                                                      |
| retfie |                    |                                                      |

• Subrutina "Visualiza"

; Visualiza la distancia expresada en centímetros. Se hace de manera que cuando haya que  
   ; visualizar un número mayor de 99 las decenas siempre se visualicen aunque sean cero.  
   ; Y cuando sea menor de 99 las decenas no se visualicen si es cero.

; Si la distancia es menor de 3 cm o mayor de 250 cm aparece un mensaje de error.

**Visualiza**

|       |                 |                                                             |
|-------|-----------------|-------------------------------------------------------------|
| call  | LCD_Borra       | ; Borra la pantalla anterior.                               |
| movlw | MinimaDistancia | ; Va a comprobar si es menor del mínimo admisible.          |
| subwf | Distancia,W     | ; $(W) = (Distancia) - MinimaDistancia$                     |
| btfss | STATUS,C        | ; ¿C=1? , ¿(W) positivo?. $(Distancia) >= MinimaDistancia?$ |
| goto  | DistanciaMenor  | ; No ha resultado menor, y salta al mensaje de error.       |
| movvf | Distancia,W     | ; Va a comprobar si es mayor del máximo admisible.          |
| sublw | MaximaDistancia | ; $(W) = MaximaDistancia - (Distancia)$                     |
| btfsc | STATUS,C        | ; ¿C=0? , ¿(W) negativo?. $MaximaDistancia < (Distancia)?$  |
| goto  | DistanciaFiable | ; No, la medida de la distancia entra dentro del rango.     |

**DistanciaMayor**

|       |                       |                                       |
|-------|-----------------------|---------------------------------------|
| movlw | MaximaDistancia       | ; La distancia es mayor que el máximo |
| movwf | Distancia             |                                       |
| movlw | MensajeDistanciaMayor |                                       |
| goto  | VisualizaDistancia    |                                       |

**DistanciaMenor**

|       |                       |                                            |
|-------|-----------------------|--------------------------------------------|
| movlw | MinimaDistancia       | ; La distancia es menor del mínimo fiable. |
| movwf | Distancia             |                                            |
| movlw | MensajeDistanciaMenor |                                            |
| goto  | VisualizaDistancia    |                                            |

**DistanciaFiable**

|                    |                  |  |
|--------------------|------------------|--|
| movlw              | MensajeDistancia |  |
| VisualizaDistancia |                  |  |

```

call    LCD_Mensaje
movlw  .S
call    LCD_PosicionLinea2 ; Centra la mitad de la distancia en la segunda linea
movf   Distanzia,W          ; de la pantalla.
call    BIN_a_BCD           ; Lo pasa a BCD.
movf   BCD_Centenas,W       ; Primero las centenas.
btfs  STATUS,Z              ; Si son cero no visualiza las centenas.
goto   VisualizaCentenas
movf   Distanzia,W          ; Vuelve a recuperar este valor.
call    BIN_a_BCD           ; Lo pasa a BCD.
call    LCD_Byte             ; Visualiza las decenas y unidades.
goto   Visualiza_cm

VisualizaCentenas
call    LCD_Nibble           ; Visualiza las centenas.
movf   Distanzia,W          ; Vuelve a recuperar este valor.
call    BIN_a_BCD           ; Lo pasa a BCD.
call    LCD_ByteCompleto     ; Visualiza las decenas (aunque sea cero) y
                            ; unidades.

Visualiza_cm
movlw  MensajeCentimetro
call    LCD_Mensaje
return

```

```

INCLUDE <RETARDOS.INC>
INCLUDE <LCD_4BIT.INC>
INCLUDE <LCD_MENS.INC>
INCLUDE <BIN_BCD.INC>
END

```

**Un sensor con mejor resolución es el SRF08.** Se trata de un medidor de distancias por ultrasonidos para robots con conexión I2C. Tiene un alcance de 6 m y un consumo de 35 mA activo y de sólo 3mA en reposo. Gracias a su bus I2C se pueden conectar hasta 16 unidades con sólo dos líneas de entrada y salida, permitiendo montar un completo sonar perimetral en cualquier robot únicamente con dos pines libres. Como valor añadido, incorpora un sensor fotoeléctrico que indica el nivel de iluminación a través también del bus I2C. Su precio es aproximadamente el doble que el del SRF04.

C

La mba  
hace poco tie  
electrónica. H  
construir un  
investigación  
para realizar ta

Para un  
sobre una ap  
experimental,  
conocimientos  
"Trasto", el le  
como único lí

### 33.1 INT

Al plan  
que hace la e  
clasificación  
niveles, cada  
microrobot. E

- Nivel  
potenc  
único  
capac

## CAPÍTULO 33

# CONSTRUCCIÓN DE UN MICROROBOT

La robótica es una de las aplicaciones más apasionantes de la electrónica. Hasta hace poco tiempo había que ser todo un experto para poder adentrarse en esa rama de la electrónica. Hoy en día, gracias al imparable avance de la microelectrónica, no es difícil construir un microrobot, denominado también **microbot**, que es un pequeño robot de investigación que normalmente se controla con un microcontrolador y que está diseñado para realizar tareas concretas.

Para un sólido aprendizaje, se van a desarrollar las explicaciones de este capítulo sobre una aplicación real. Vamos a detallar la construcción de un microrobot experimental, a partir del cual, el lector pueda desarrollar toda sus habilidades manuales y conocimientos de mecánica y electrónica. A este microrobot experimental le llamaremos "Trasto", el lector puede introducir todas las mejoras que sea capaz de idear, poniendo como único límite su imaginación (y presupuesto).

### 33.1 INTRODUCCIÓN A LA MICROBÓTICA

Al plantear la construcción de un microbot es interesante conocer la clasificación que hace la empresa Microbótica, una de las pioneras en este campo en España. Esta clasificación está basada en la Torre de Bot o **TorreBot** (figura 33-11, que tiene seis niveles, cada uno de los cuales diferencia un paso en el diseño y construcción del microrobot. Estos niveles son:

- **Nivel físico.** Comprende la estructura física, las unidades motoras, y las etapas de potencia. Es posible encontrar desde sistemas sumamente sencillos basados en un único motor hasta estructuras sumamente complejas que buscan emular las capacidades mecánicas de algunos insectos.

- **Nivel de reacción.** Está formado por el conjunto de sensores y los sistemas básicos para su manejo. Estos sensores cubren un amplio margen de posibilidades, así podemos encontrar desde simples bumpers (finales de carrera), hasta microcámaras digitales con sistemas de reconocimiento. Un microbot que haya superado en cuanto a su construcción tanto el nivel físico como el de reacción, se denomina **microbot reactivo**. Estas unidades trabajan cumpliendo la premisa, "acción-reacción". En este caso los sensores son los propios controladores de las unidades motoras, sin ningún tipo de control intermedio.

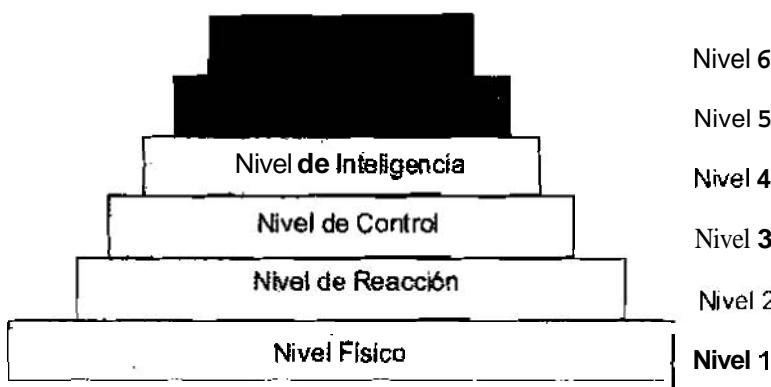


Figura 33-1 Representación de la Torrebot

- **Nivel de control.** Incluye los circuitos más básicos que relacionan las salidas de los sensores con las restantes unidades. Partiendo de una simple lógica digital y llegando hasta potentes microcontroladores buscan dotar al microbot de la capacidad para procesar la información obtenida por los sensores así como actuar de una manera controlada sobre las unidades motoras.
- **Nivel de inteligencia.** Abarca la planificación a largo plazo. En este nivel se introducen los objetivos del microbot que tienen relativa independencia de los sensores. Este es el nivel más alto de inteligencia que puede alcanzar un microbot como una unidad individual.
- **Nivel de comunidad.** Se trata de la puesta en funcionamiento de más de un microbot dentro de un mismo entorno de forma simultánea y sin que ninguno de ellos tenga conocimientos explícitos de la existencia de otros en su mismo entorno. A estos recintos se los denomina granjas. Los centros de investigación utilizan las granjas como entornos de observación de los microbot. Dichos establecimientos pueden contar con sistemas sofisticados que permitan a un operario monitorizar el comportamiento de la comunidad así como alterar las condiciones externas del sistema (agregar obstáculos, cambiar la temperatura, etc.)

- Niveles de comunión como cooperación popular.

En este

## 33.2 NIVEL DE INTELIGENCIA

Dentro de un motor para factores como que queremos mercado.

### 33.2.1 MÓDULOS

Dentro son los que se revoluciones apropiados para un sistema de

### 33.2.2 MÓDULOS

En los o la posibilidad de reductores, que microbot con algunos de

### 33.2.3 SISTEMAS

En micr radiocontrol. Se estudiado en e

Estos s construcción de trasladar objetos de multitud de ad

- **Nivel de cooperación.** Comprende los sistemas donde a partir de un nivel de comunidad se planifican o programan los microbot para que tengan conocimiento de la existencia de otros, de manera que posean la capacidad de cooperar para el buen desarrollo de una tarea. Dentro de este grupo estarían los populares equipos de fútbol constituidos por microrobots.

En este capítulo vamos a construir un robot que llegará hasta el nivel de control.

## 33.2 NIVEL FÍSICO. MOTORES

Dentro del nivel físico comenzaremos a hablar de los motores. A la hora de elegir un motor para aplicaciones de micróbotica, debemos tener en cuenta que existen varios factores como son la velocidad, el par, el frenado, la inercia y el modo de control. Si lo que queremos es utilizar un motor de corriente continua existen varias posibilidades en el mercado.

### 33.2.1 Motores de corriente continua de pequeña potencia

Dentro de la gran variedad de tipos existentes en el mercado los más económicos son los que se utilizan en algunos juguetes. Tienen el inconveniente de que su número de revoluciones por minuto es muy elevado y su par es pequeño, lo que no los hace muy apropiados para la construcción de un microbot si no se utilizan reductoras adicionales o un sistema de regulación electrónico.

### 33.2.2 Motores de corriente continua con reductoras

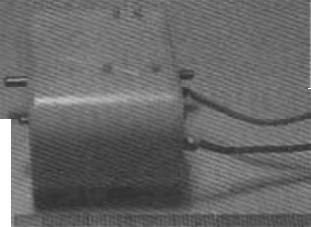
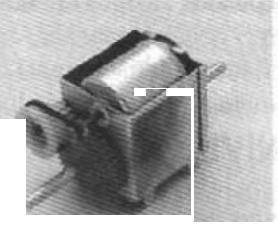
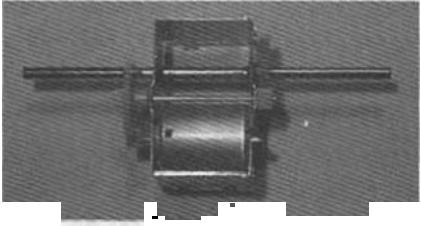
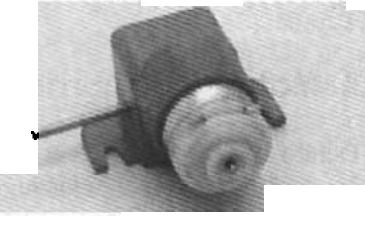
En los juguetes como *Mecano* y *Lego* podemos encontrar motores con reductoras o la posibilidad de construirlos. También podemos encontrar en el mercado motores con reductores, que además de disminuir la velocidad le dan más par, lo que permite mover el microbot con su estructura y batería que proporcionalmente pesa mucho. Las fotografías de algunos de estos motores se muestran en las figuras de la tabla 33-1.

### 33.2.3 Servomotores

En micróbotica se suelen utilizar los mismos servomotores que en modelismo y radiocontrol. Se trata de unos motores con un circuito electrónico, como los que se han estudiado en el capítulo 31.

Estos servomotores cumplen las características que los hacen idóneos para la construcción de nuestro microbot, como un buen par de salida, potencia suficiente para trasladar objetos o una batería, baja inercia, capacidad de mover 3,5 Kg x cm, incluyen multitud de accesorios para poder fijar las ruedas del microbot y son fáciles de fijar a una

estructura plana al ir dentro de una carcasa de plástico rectangular con soportes para fijar los tornillos. De hecho la opción que hemos elegido para la construcción de nuestro microbot "Trasto" ha sido utilizar estos servomotores como elementos motrices.

|                                                                                                 |                                                                                                  |
|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
|                |                |
| <b>Motor reductor dc relación 194:1 con doble eje φ 2,4x6 mm, tensión de trabajo 1,5V-12V</b>   | <b>Motor con reducción de 17:1 con doble eje de φ 4x20 mm, tensión de trabajo 1,5-12 V</b>       |
|                |                |
| <b>Motor reductor de relación 23:1 con doble eje φ 4x40 mm, tensión de trabajo 1,5 V a 12 V</b> | <b>Motor reductor de relación 10:1 con doble eje de φ 2x20 mm, tensión de trabajo 1,5V-4,5 V</b> |

*Tabla 33-1 Tipos de motores de c.c. comerciales con reductoras*

Para poder utilizarlo en nuestra aplicación, un servomotor debe ser "trucado" para que el eje del motor pueda girar los 360 ° ya que normalmente giran entre hasta 180° ó 270°, dependiendo de los fabricantes.

### 33.2.4 Modificación de un servomotor

El "trucar" los servomotores los hará inservibles para su uso en las aplicaciones típicas de radiocontrol, ya que se convertirá en un motor de corriente continua con una caja reductora, pero desde luego para nuestro fin no tiene ningún problema.

Seguidamente se muestran los pasos a seguir para convertir los servomotores en motores de corriente continua con una caja reductora. La mayor parte de los servomotores son similares, nosotros vamos a modificar el modelo HS-300B de la marca Hitec que es muy similar al Futaba S3003.

Los servomotores giran como los que están en restricción mecánicos que circuito eléctrico podemos por lo tanto, motor de DC con

Quitar el tornillo rueda tractora que es estriada sacarla despues

Sacar los cu posterior. Al circuito eléctrico está metido a hay que extraer del potenciómetro engranajes que que hemos abierto

En esta figura de la etapa de reducir la velocidad y par

tes para fijar  
de nuestro  
es.

de trabajo

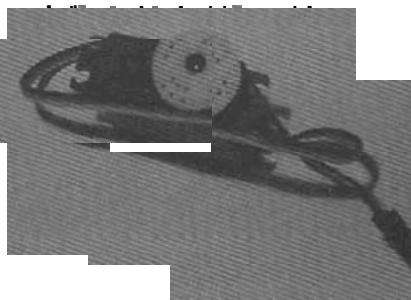
con doble  
bajo 1,5V-

ucado" para  
asta 180" ó

plcaciones  
ua con una

motores en  
te dc los  
e la marca

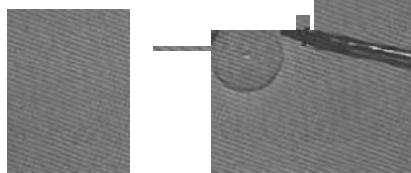
Los servomotores de origen son capaces de girar como mucho entre  $180^\circ$  y  $270^\circ$ , como es requerido en las aplicaciones para las que están pensados inicialmente. Esta restricción viene impuesta por unos topes mecánicos que limitan el giro a  $180^\circ$  y un circuito electrónico. Si eliminamos las dos cosas podemos conseguir el giro de  $360^\circ$  y, por lo tanto, que se comporte como un motor de DC con caja reductora



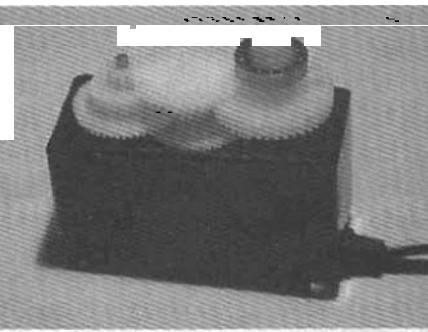
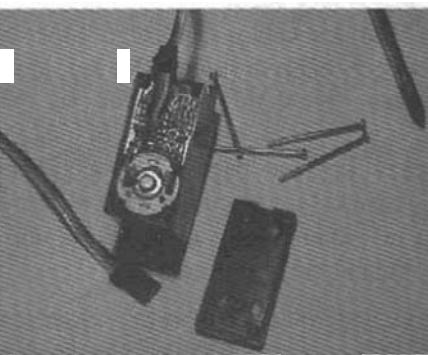
Quitar el tornillo que sujeta el soporte de la rueda tractora unida solidariamente al eje, que es estriado, por lo que habrá que sacarla después a presión.



Sacar los cuatro tornillos de la tapa posterior. Al levantarla, se puede ver un circuito electrónico que en nuestro caso está metido a presión, para poder quitarlo hay que extraer el tornillo que sujeta el eje del potenciómetro por la parte de los engranajes que están en la cara opuesta a la que hemos abierto.



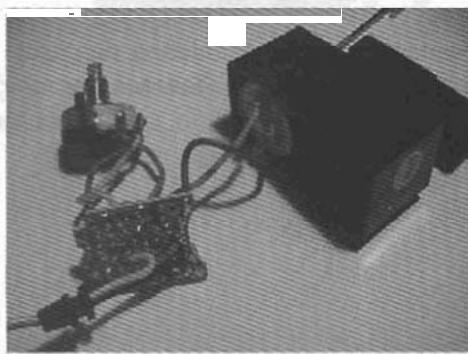
En esta figura se aprecian los engranajes de la etapa reductora cuya misión es reducir la velocidad del motor y dar mayor potencia y par de arranque al sistema



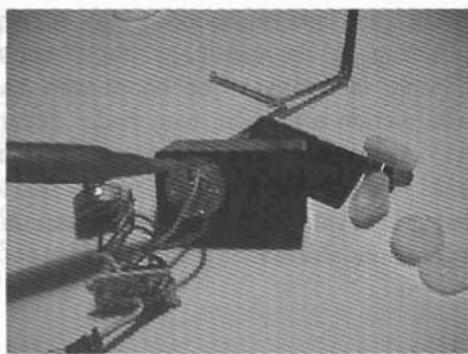
Desmontar las ruedas dentadas, teniendo mucho cuidado de no perder ninguna de ellas, prestar atención al pequeño eje que hay entre las ruedas intermedias, en algunos modelos de servomotores es móvil, en nuestro caso está fijado a la carcasa. Con ayuda de unos alicates de punta plana quitar ahora la tuerca que sujeta el potenciómetro.



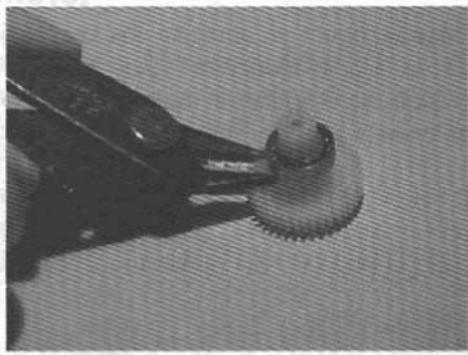
Proceder a desmontar la placa del circuito impreso y el potenciómetro ayudándose con un destornillador para hacer un poco de palanca.



Desoldar los cables que están conectados al motor para desprenderlo del circuito impreso. Hacer lo mismo con los cables que conectan el exterior a la placa del circuito impreso para poder reutilizarlos. Seguidamente conectar el rojo al terminal con el punto rojo y el negro al otro, el tercer cable no se utiliza.



Ahora eliminar el limitador mecánico, que consiste en una pestaña de la rueda dentada, para ello utilizar unos alicates de corte tal y como se muestra en la figura. Usar una lima pequeña si hay que eliminar algunos restos de la pestaña. Tener mucho cuidado para no romper la rueda porque el servomotor se volvería inservible.



Volver a montar la caja reductor no confundir, no forzar ni de ninguna manera que la tapa superior. Nuestro caso el eje de las ruedas está en la parte

Atornillar nur aconsejable los cables del interior para el caso de tirar

Tabla 33-2 Sec

En todo caso la rueda dentada ocurre muy frecuentemente que la rueda, se puede convenientemente separado en tiem

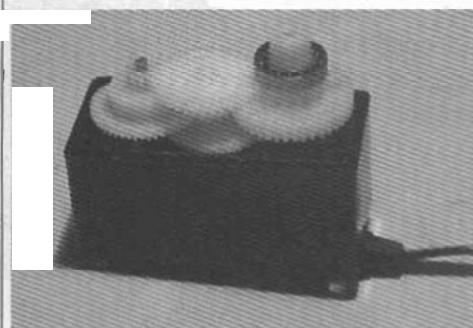
Pues bien deberemos hacer

### 33.2.5 Fijación

Para fijar la próxima apartad hacerlas nosotros

Si se utilizan han mostrado la carcasa redonda y el tubo de las in

**Volver a montar las ruedas dentadas de la caja reductora, fijándose en la figura para no confundirse, y tener mucho cuidado de no forzar ninguno de los engranajes, de manera que no puedan deteriorarse. La tapa superior deberá entrar sin forzarla.** en nuestro caso hay que tener cuidado con el eje de las ruedas superior e inferior que está en la propia carcasa.



Atornillar nuevamente la tapa inferior, es aconsejable hacer antes un nudo en los cables del motor y dejar el nudo en el interior para que proteja las soldaduras en el caso de tirar del cable.

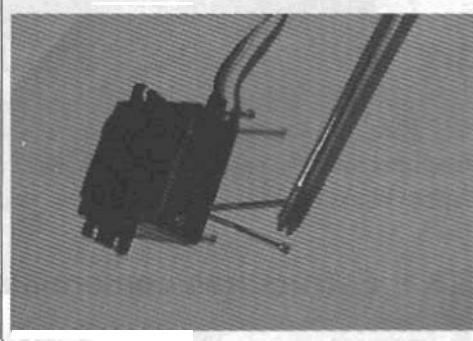


Tabla 33-2 Secuencia para transformar un servomotor en motor DC con caja reductora

En todo este proceso, el momento más delicado es la eliminación de la pestaña de la rueda dentada ya que si no se hace con sumo cuidado la rueda se puede partir, caso que ocurre muy frecuentemente. Si aún con todos los avisos que se le estamos dando, rompe la rueda, se puede intentar pegar con un pegamento de contacto. En este punto, es conveniente informar de que estas ruedas dentadas también se pueden comprar por separado en tiendas de modelismo y radiocontrol.

Pues bien, ya tenemos uno de los motores preparados para nuestro microbot, deberemos hacer lo mismo con el otro para tener la pareja necesaria.

### 33.2.5 Fijación del motor a la estructura

Para fijar el motor a la estructura de nuestro microbot, que describiremos en el próximo apartado, tan sólo tendremos que conseguir unas escuadras con unos taladros, o hacerlas nosotros mismos con un trozo de aluminio y la ayuda de un tornillo de banco.

Si se utiliza cualquiera de los otros tipos de motores de corriente continua que se han mostrado la fijación al chasis puede ser más o menos compleja. Si el motor tiene una carcasa redonda, que es lo normal, se puede utilizar una grapa de las utilizadas para fijar el tubo de las instalaciones eléctricas de superficie, tal como se muestra en la figura 33-2.



Figura 33-2 Fijación de un motor a una estructura plana

### 33.3 NIVEL FÍSICO. ESTRUCTURA

Para la construcción de nuestro microbot podemos utilizar muchos tipos de estructuras, que dependerán de la función que queramos realizar, no es lo mismo diseñar un robot bipedo que un rastreador a un hexápedo.

#### 33.3.1 Estructuras comerciales

Unas de las estructuras más utilizadas es la de los juegos educacionales de construcción tipo *Lego*, *Mecano* o *Eitech*, interesantes por su flexibilidad. Para un diseño un poco más profesional se pueden utilizar las estructuras de los *Fischer Technik* que fueron diseñadas originariamente para aplicaciones técnicas tanto estáticas como de estructuras mecánicas con movimiento (figura 33-3).

#### 33.3.2 Es

Para el  
microbot expe  
por ejemplo  
incluso puec

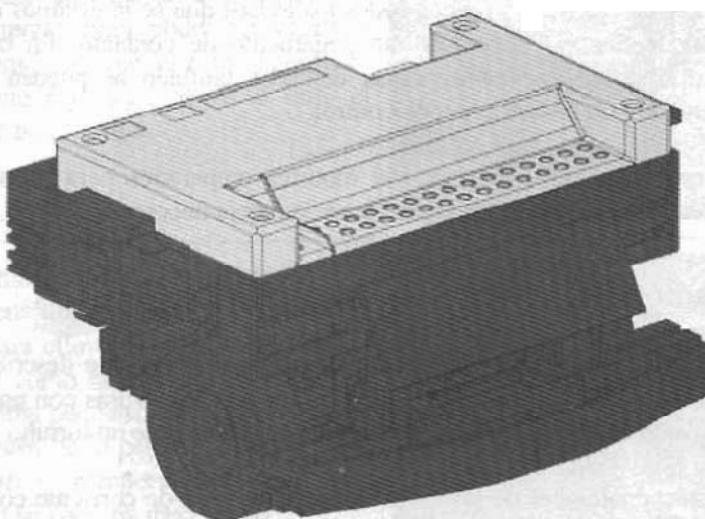


Figura 33-3 Estructura Fischer Technik, (cortesía de Métodos y Sistemas S.L.)

Un ejemplo de estructura realizada con un *Mecano* es el caso del microrobot Pivot-1 de la empresa *Microsystems Engineering*. En la foto que se muestra en la Figura 334, donde podemos ver cómo sus diseñadores han integrado hábilmente la placa de circuito impreso de control en la estructura mecánica y justamente debajo de él se encuentra alojada una batería de plomo de 12V y 0,8 mA·h.

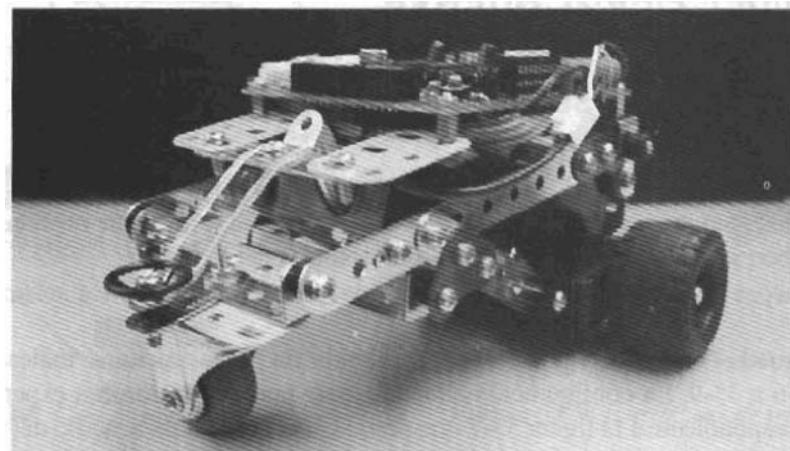


Figura 33-4 *Microbot Pivot-1 modificado*

### 33.3.2 Estructura del microrobot experimental "Trasto"

Para el caso de nuestro microrobot, que no debe salvar obstáculos y que sea un microrobot experimental de bajo coste, podemos utilizar una estructura más sencilla, como por ejemplo un trozo de metacrilato, poliestireno, PVC, placa de circuito impreso o incluso puede servirnos la caja de plástico de un *Compact Discs*.

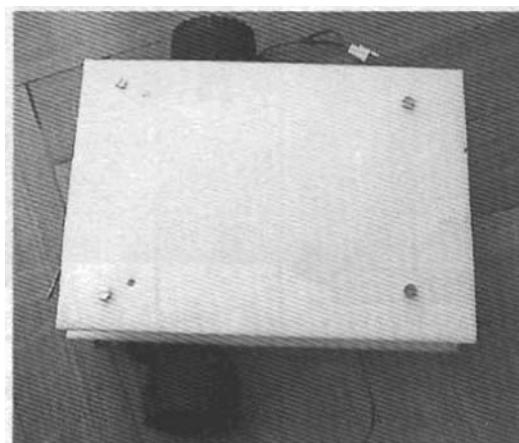


Figura 33-5 *Estructura del microrobot "Trasto"*

En nuestro caso vamos a utilizar dos placas de poliestireno blanco de 180 x 13,5 mm, pero pueden utilizarse otras medidas. Se utilizan dos placas para poder poner las baterías en el piso inferior y el circuito de control de los motores y sensores en el superior (figura 33-5).

DIRECCIÓN  
Ruedas de

## 33.4 NIVEL FÍSICO. RUEDAS

### 33.4.1 Estructuras según la colocación de las ruedas

Los microbots utilizan dos tipos de ruedas:

- Ruedas motrices o de tracción, que están conectadas al motor mediante un eje y deben ser capaces de adaptarse a los obstáculos del terreno.
- Ruedas "locas", que deben ser capaces de rodar y pivotar sobre sí mismas.

Las ruedas se pueden colocar según alguna de las estructuras indicadas en las figuras 33-6 a 33-9. La configuración adoptada para nuestro microrobot experimental ha sido la correspondiente a la figura 33-7 que permite un control más sencillo del sistema.

Figura 33-

**ESTRUCTURA DE COCHE**  
Tracción y dirección en  
das ruedas

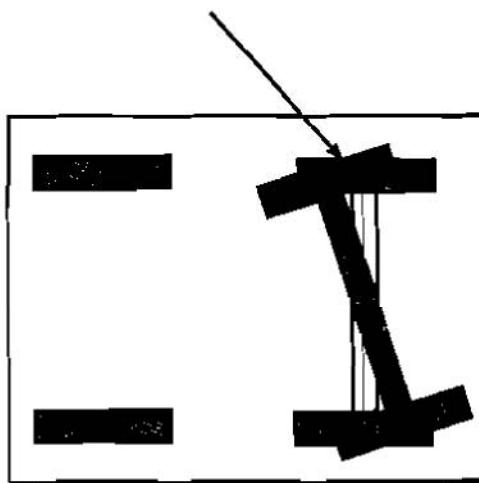


Figura 33-6 Estructura tipo coche

**ESTRUCTURA DE  
NUESTRO MICROBOT**

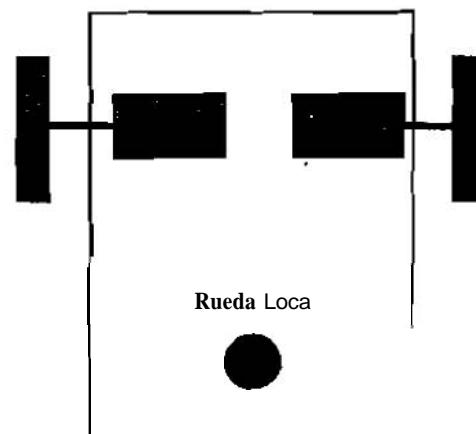


Figura 33-7 Para microrobot Trasto

### 33.4.2 Ru

Las rue  
movimiento lo  
contrario es po

Las sol  
optado por la  
gracias a una  
cualquier ferre

180 x 13,5  
er poner las  
el superior

ante un eje y

mas.

rcadas en las  
perimental ha  
el sistema.

ot Trasto

**DIRECCIÓN DIFERENCIAL**  
Ruedas de tracción

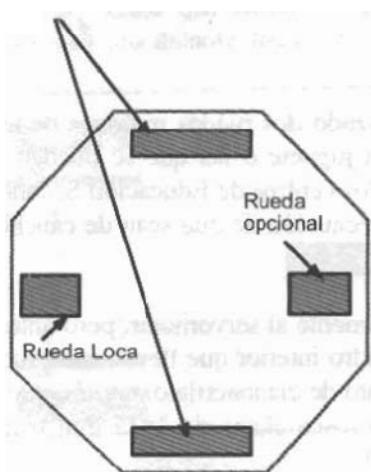


Figura 33-8 Con dirección diferencial

**ESTRUCTURA DE TRICICLO**  
Tracción y dirección independientes

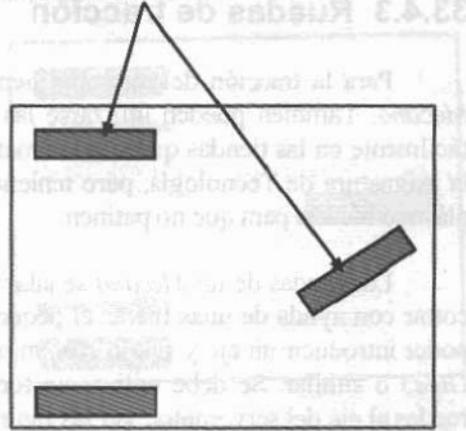


Figura 33-9 Estructura de triciclo

### 33.4.2 Ruedas "locas"

Las ruedas "locas" deben ser capaces de dar y pivotar sobre sí mismas con un movimiento lo más suave posible para no dificultar la rotación del microrobot, de lo contrario es posible que se bloquee y patine.

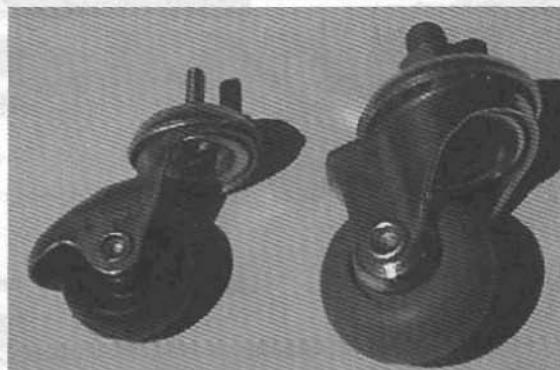


Figura 33-10 Ruedas "locas" con rodamientos

Las soluciones para este tipo de ruedas pueden ser muchas. Nosotros hemos optado por las de la figura 33-10, que son ruedas que giran libremente sobre su eje gracias a una pequeña plataforma con rodamientos, las podemos encontrar fácilmente en cualquier ferretería, además hay un gran surtido de ellas en lo referente a tamaños. Otra

opción podría ser utilizar la bola de un *roll-on* de desodorante, a la que se le adapta un eje acabado en un terminal para fijarla a la estructura.

33.5

### 33.4.3 Ruedas de tracción

Para la tracción del microbot hemos utilizado dos ruedas motrices de un juguete *Mecano*. También pueden utilizarse las de otro juguete o las que se pueden encontrar fácilmente en las tiendas que venden material a los centros de Educación Secundaria para la asignatura de Tecnología, pero teniendo la precaución de que sean de caucho o de un plástico blando para que no patinen.

Las ruedas de un *Mecano* se adaptan fácilmente al servomotor, pero antes hay que cortar con ayuda de unas tijeras el pequeño cilindro interior que llevan estas ruedas, para poder introducir un eje y fijarlo con un pegamento de cianoacrilato, tipo *Loctite de Super Glue3* o similar. Se debe utilizar un tornillo de rosca chapa de 3x25 mm para fijar las ruedas al eje del servomotor, ver las figuras 33-11.

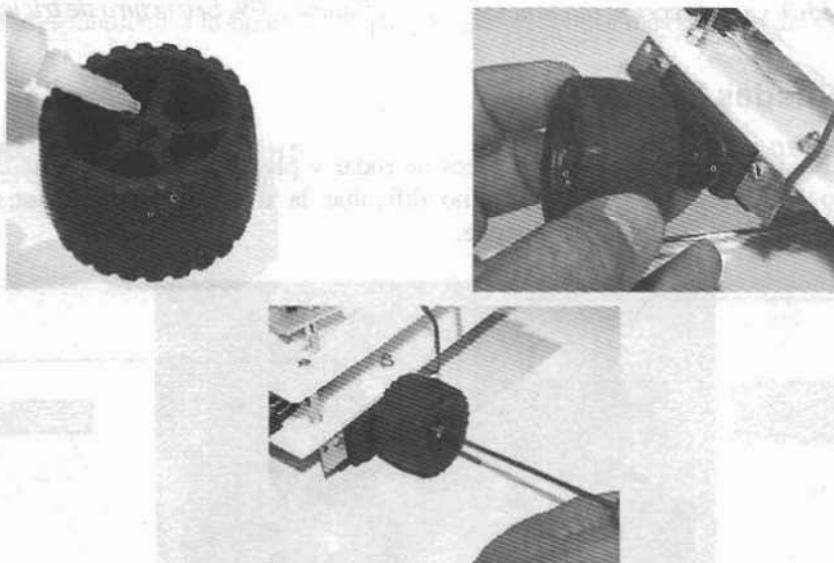


Figura 33-11 Secuencia de fijación de una rueda al servomotor

Las mejores ruedas pueden comprarse en tiendas de modelismo, que son ruedas de aire y neopreno, ideales para esta aplicación, pero suelen ser bastante más caras.

Este tipo de ruedas tienen una gran adherencia al suelo y una gran duración, pero suelen ser más caras que las ruedas de plástico. Las ruedas de plástico suelen ser más económicas y duraderas, pero suelen tener una menor adherencia al suelo y una menor duración.

adapta un eje

en un juguete  
en encontrar  
cundaria para  
icho o de un

ntes hay que  
ruedas, para  
tite de Super  
para fijar las

### 33.5 NIVEL FÍSICO. MOVILIDAD

La estructura que hemos elegido para nuestro microbot nos permitirá realizar movimientos hacia delante, hacia atrás, giro a la derecha, a la izquierda y sobre si mismo.

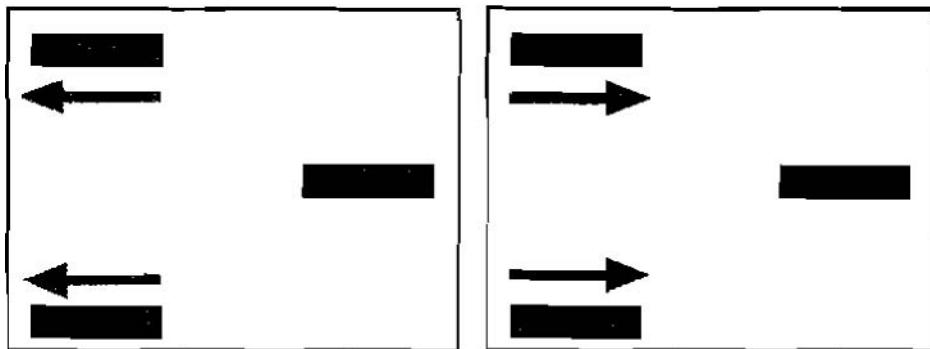


Figura 33-12 *Movimiento hacia delante* Figura 33-13 *Movimiento hacia atrás*

En la figura 33-12 se muestra como se realiza un movimiento hacia delante. Se hacen girar los dos motores en la misma dirección hacia delante, esto provoca un movimiento rectilíneo, suponiendo que los dos motores sean exactamente iguales.

La figura 33-13 representa la forma de realizar el movimiento hacia **atrás**. Se hacen girar los **dos** motores en la misma dirección hacia atrás, esto provoca un movimiento rectilíneo, suponiendo que los dos motores sean exactamente iguales.

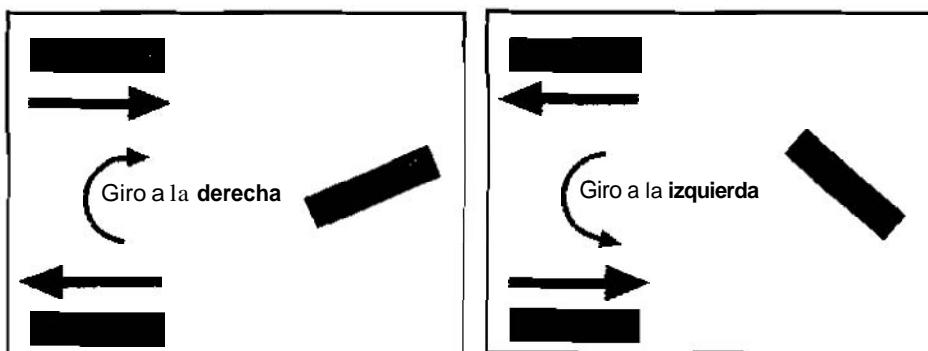
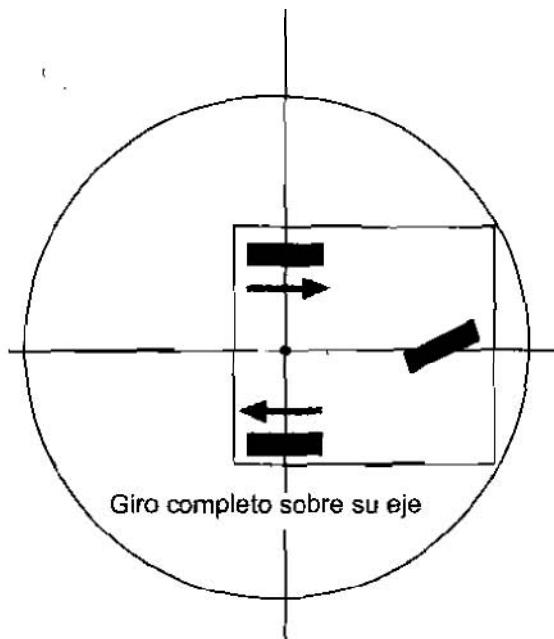


Figura 33-14 *Giro en sentido horario*

Figura 33-15 *Giro a la izquierda*

Por su parte la figura 33-14 muestra la forma de realizar un movimiento de giro a la derecha. Se hace girar el motor izquierdo hacia adelante y el motor de la derecha hacia atrás. Esto provoca un movimiento de giro a la derecha de la estructura.

La figura 33-15 indica cómo realizar un movimiento de giro hacia la izquierda. Se hace girar el motor izquierda hacia atrás y el motor de la derecha hacia delante, esto provoca un movimiento de giro a la izquierda de la estructura.



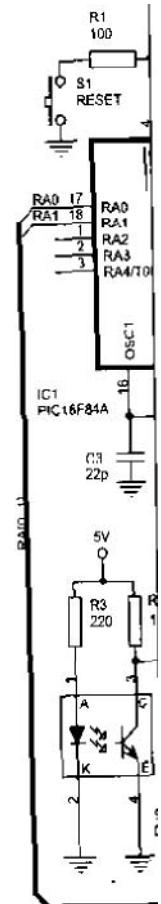
**Figura 33- 16** Movimiento de giro sobre su propio eje

El movimiento de giro completo sobre su propio eje abarca una superficie muy grande que no hace la estructura muy adecuada para moverse en recintos muy pequeños (figura 33-16) como podría ser el caso de movimientos en pruebas de laberintos.

### **33.6 NIVEL DE REACCIÓN**

Como deciamos al principio de este capítulo, este nivel esta formado por los sistemas electrónicos y sensoriales básicos para su control. Vamos a realizar **seguidamente** un sistema de control para construir un microbot reactivo, gobernado por el microcontrolador PIC16F84A, que sea capaz de seguir una línea negra sobre un fondo blanco. A este tipo de microbots se los denomina **rastreadores** y probablemente son los más sencillos para introducirse en el mundo de la microróbótica.

La figura 33-17 muestra el circuito eléctrico donde se aprecia que los sensores utilizados son infrarrojos reflexivos del tipo CNY70, descritos en el capítulo anterior. Para controlar los motores utilizamos el driver L293B explicado en el capítulo 29.



Para fi  
adquisición e

Para  
microcontrol  
la ventaja d

quierda. Se elante, esto

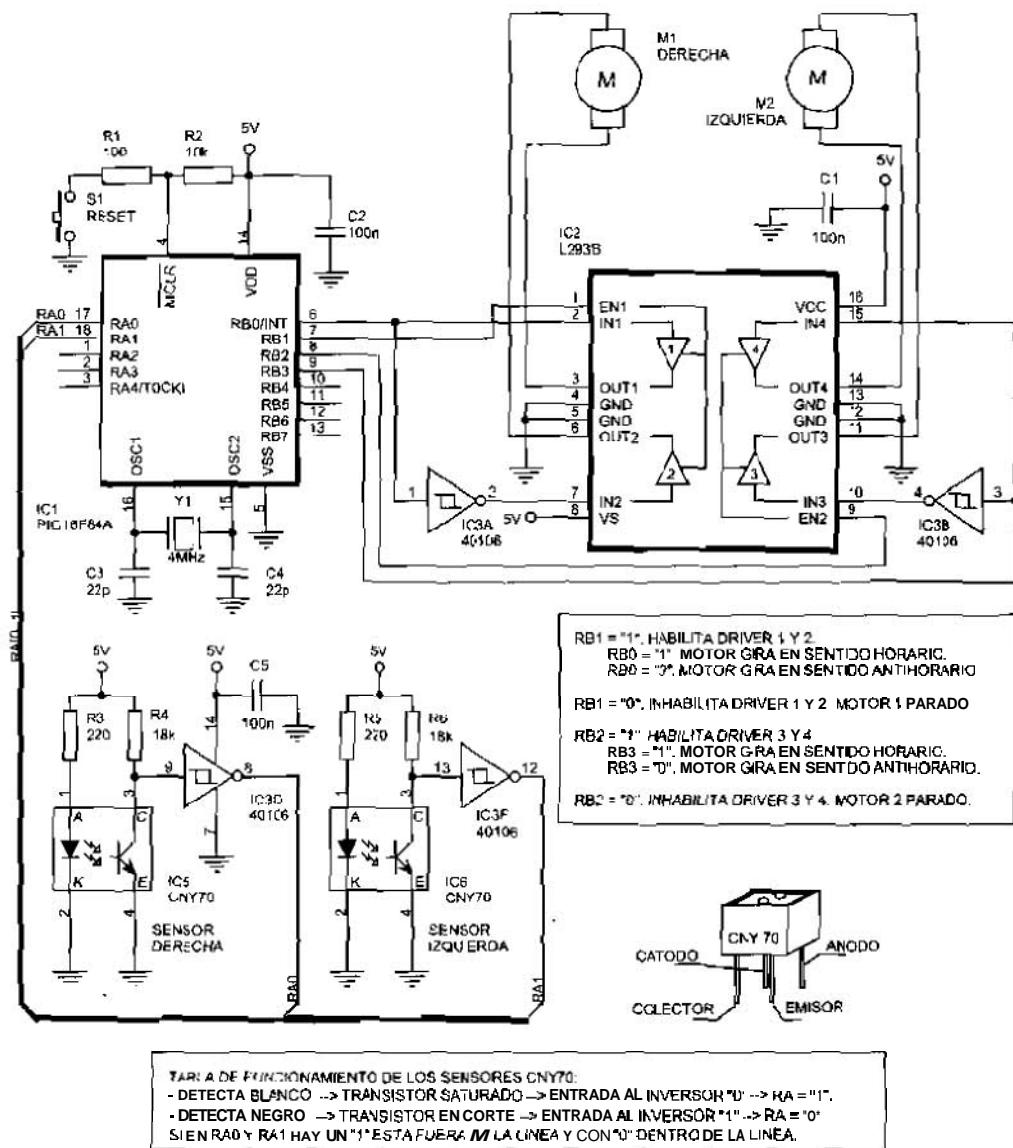
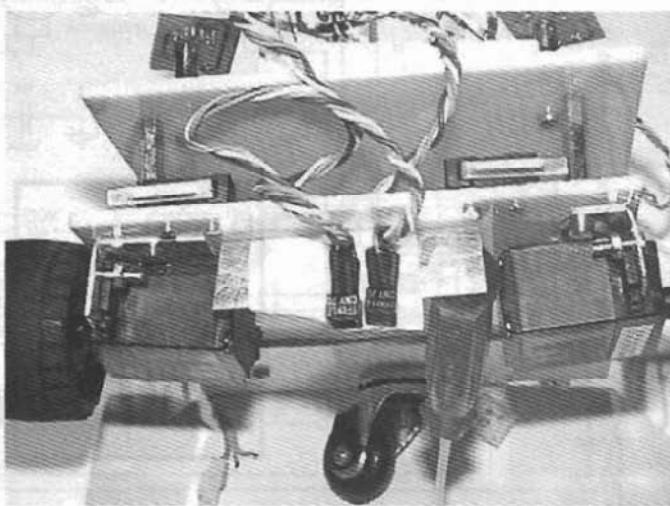


Figura 33-17 Circuito eléctrico del Microbot "TRASTO"

Para fijar los sensores hemos utilizado una cinta adhesiva por las dos caras. de fácil adquisición en una ferretería. En la Figura 33-18 se aprecia la forma de fijarlos.

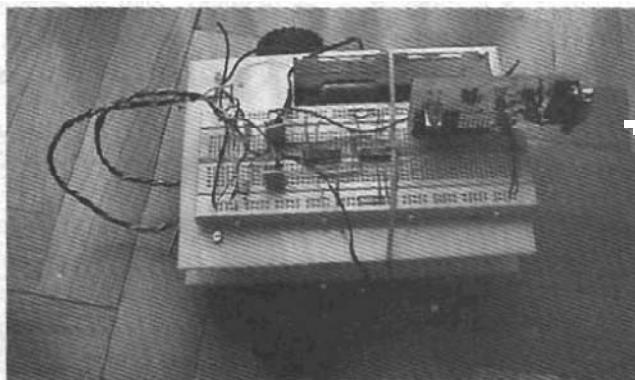
Para poder conformar las señales de los sensores CNY70 a la entrada del microcontrolador hemos utilizado puertas inversoras Trigger Schmitt, que además, tienen la ventaja de que en el mismo chip 40106 nos encontramos con seis inversores. El

funcionamiento y la descripción de estos dispositivos se explicaron en el capítulo anterior. Como puede apreciarse en el circuito de la figura 33-17:



*Figura 33-18 Fijación de los sensores CNY70 a la estructura del microbot Trasto*

- Cuando un sensor detecta el fondo blanco, a la entrada de la línea del PORTA al que está conectado le llega un "1".
- Cuando un sensor está sobre la línea negra, a la entrada de la línea del PORTA al que está conectado le llega un "0".



*Figura 33-19 Vista lateral del microbot Trasto*

Al L293B le hemos conectado los dos motores que necesita el microbot:

- El motor derecho se encuentra conectado a los drivers 1 y 2 que están controlados por las líneas RB0 y RB1 del microcontrolador.

• El  
co

El m

| Salida | RBI |
|--------|-----|
| 0      | 0   |
| 1      | 1   |
| 1      | 1   |

| Salidas | RB2 |
|---------|-----|
| 0       | 0   |
| 1       | 1   |
| 1       | 1   |

En casa  
que hacer es ir  
que presenta el

### 33.7 NIV

#### 33.7.1 Est

Antes d  
microbot, ya s  
función. De est

En princi  
hora de diseñar

- Primer  
que el i  
los cu  
correspo  
encluent

- El motor izquierdo se encuentra conectado a los drivers 3 y 4, que a su vez están controlados por las líneas KB2 y RB3 del microcontrolador.

El modo de funcionamiento de los motores se muestra en las tablas 33-3 y 33-4.

| Salidas PIC16F84A |     | Entradas Driver |     |     | Salidas Driver |      | Motor Derecho             |
|-------------------|-----|-----------------|-----|-----|----------------|------|---------------------------|
| RB1               | RB0 | EN1             | IN1 | IN2 | OUT1           | OUT2 |                           |
| 0                 | 0   | 0               | 0   | 1   | 0              | 0    | No gira                   |
| 1                 | 1   | 1               | 1   | 0   | 1              | 0    | Gira en sentido avance    |
| 1                 | 0   | 1               | 0   | 1   | 0              | 1    | Gira en sentido retroceso |

Tabla 33-3 Control del motor derecho

| Salidas PIC16F84A |     | Entradas Driver |     |     | Salidas Driver |      | Motor Izquierdo           |
|-------------------|-----|-----------------|-----|-----|----------------|------|---------------------------|
| RB2               | RB3 | EN2             | IN4 | IN3 | OUT4           | OUT3 |                           |
| 0                 | 0   | 0               | 0   | 1   | 0              | 0    | No gira                   |
| 1                 | 1   | 1               | 1   | 0   | 1              | 0    | Gira en sentido avance    |
| 1                 | 0   | 1               | 0   | 1   | 0              | 1    | Gira en sentido retroceso |

Tabla 33-4 Control del motor izquierdo

En caso que al montarlo el motor le gire en sentido contrario, lo único que tiene que hacer es invertir sus conexiones. La fotografía de la figura 33-19 muestra el aspecto que presenta el microbot "Trasto" con el circuito implementado.

## 33.7 NIVEL DE CONTROL

### 33.7.1 Estrategia a seguir para un microbot rastreador

Antes de realizar el programa debemos fijar la estrategia que debe seguir el microbot, ya sea para que se comporte como un robot rastreador o para cualquier otra función. De esta manera podremos fijar el algoritmo de control.

En principio parece razonable pensar en cualquiera de las siguientes estrategias a la hora de diseñar el algoritmo de funcionamiento:

- **Primer Algoritmo.** Dependiendo de la posición de los sensores podemos hacer que el microbot tome las decisiones mostradas en la figura 33-20. En este dibujo los cuadrados de la derecha e Izquierda indican lectura del sensor correspondiente: si es blanco indica que detecta fondo blanco y si es negro que encuentra encima de la línea negra.



Figura 33-20 Decisiones a tomar según el primer algoritmo

Analizando este algoritmo detenidamente se puede comprobar que no es lo suficientemente bueno, puesto que el seguimiento de la linea depende de la imprecisión del camino seguido por el microbot, es decir, depende de la anchura de la pista. Esto puede ocasionar retrasos en el recorrido, cabeceos no deseados o incluso que llegue a perderse.

- **Segundo Algoritmo.** En este caso, dependiendo de la posición donde se encuentra el microbot sobre la línea, decidimos seguir uno de los **bordes**, en nuestro caso el **borde** derecho, es decir, la **detección de negro-blanco** respectivamente por los sensores colocados a la derecha y a la **izquierda** tal y como se muestra en la figura 33-21.



Figura 33-21 Decisiones a tomar según el segundo algoritmo

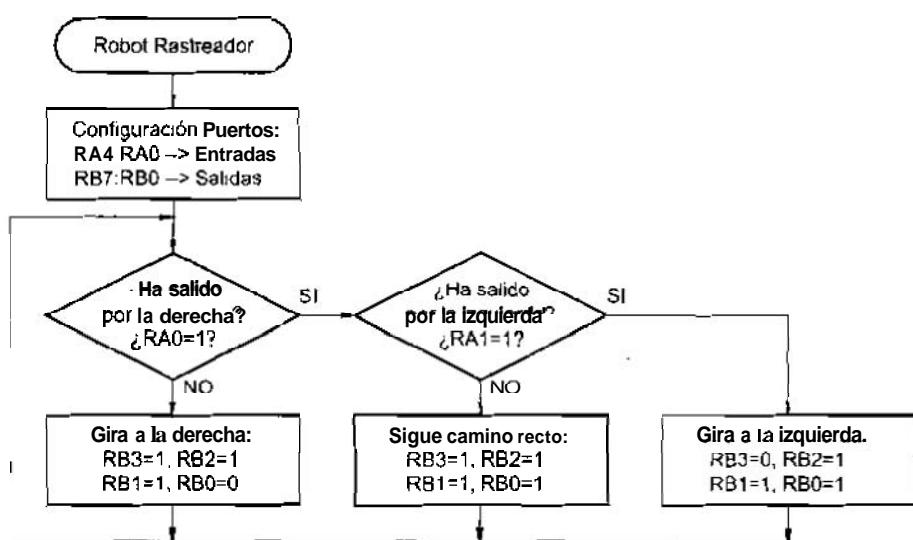


Figura 33-22 Diagrama de flujo del programa del microrobot rastreador

### 33.7.2 Programa del rastreador

El programa que realiza el segundo algoritmo del rastreador se muestra a continuación, es fácil deducir su funcionamiento si hemos seguido los razonamientos y se entiende el organigrama de la figura 33-22.

```
***** Robot_Rastreador_01.asm *****
;
; Programa de control para el microbot TRASTO, el cual se desplaza siguiendo una línea negra
; marcada sobre fondo blanco a modo de pista.
; Los sensores ópticos de reflexión CNY70 están situados en la parte delantera inferior del
; microbot: El sensor de la derecha está conectado a RA0 y el sensor de la izquierda a RA1.
;
; El programa adopta la estrategia de seguir la línea por el borde derecho:
; - Si detecta que está en el borde derecho: sensor izquierdo sobre negro y derecho sobre
;   blanco sigue en hacia delante.
; - Si el sensor de la derecha detecta línea negra gira hacia la derecha buscando el borde,
;   independientemente de como esté el sensor de la izquierda.
; - Si el microbot tiene los dos sensores fuera de la línea, se le hace girar a la izquierda
;   hasta que vuelva a encontrarla.
;
; La señal de los sensores CNY70 se aplican a las entradas del microcontrolador a través de un
; inversor 40106 de manera tal, que para color:
; - Color Blanco --> transistor saturado --> entrada al inversor "0" --> RAx = "1"
;   (No está encima de la línea negra, se ha salido de la pista)
; - Color Negro --> transistor en corte --> entrada al inversor "1" --> RAx = "0".
;   (Está encima de la línea negra, está dentro de la pista)
;
; ZONA DE DATOS *****
;
CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>

#DEFINE SensorDerecha PORTA,0 ; Sensor óptico Derecho.
#DEFINE SensorIzquierda PORTA,1 ; Sensor óptico Izquierdo.

; ZONA DE CÓDIGOS *****
;
ORG 0
Inicio
    bsf STATUS,RP0 ; Selecciona Banco 1 de registros.
    bsf SensorDerecha ; Estas líneas se configuran como entrada.
    bsf SensorIzquierda
    clrf PORTB ; Las líneas del Puerto B se configuran como salidas.
    bcf STATUS,RP0 ; Selecciona Banco 0 de registros.
Principal
    movlw b'00001110' ; Para girar a la derecha.
    bffs SensorDerecha ; ¿Ha salido por la derecha?, ¿detecta blanco?
    goto ActivaSalida ; No, el detector derecho está encima de la línea
                       ; negra, gira a la derecha.
    movlw b'00000111' ; Para girar a la izquierda.
    btfs SensorIzquierda ; ¿Ha salido también por la izquierda?
```

```

    movlw b'000001111'          ; No, está en el borde derecho. Sigue recto.
ActivaSalida
    movwf PORTB
    goto Principal
    END

```

### 33.7.3 Estrategia a seguir para un robot detector de baliza

Nuestro microbot "Trasto" en este caso deberá detectar una señal infrarroja modulada a 38 kHz que emite un circuito como el de la figura 32-20 (capítulo 32) y dirigirse hacia ella.

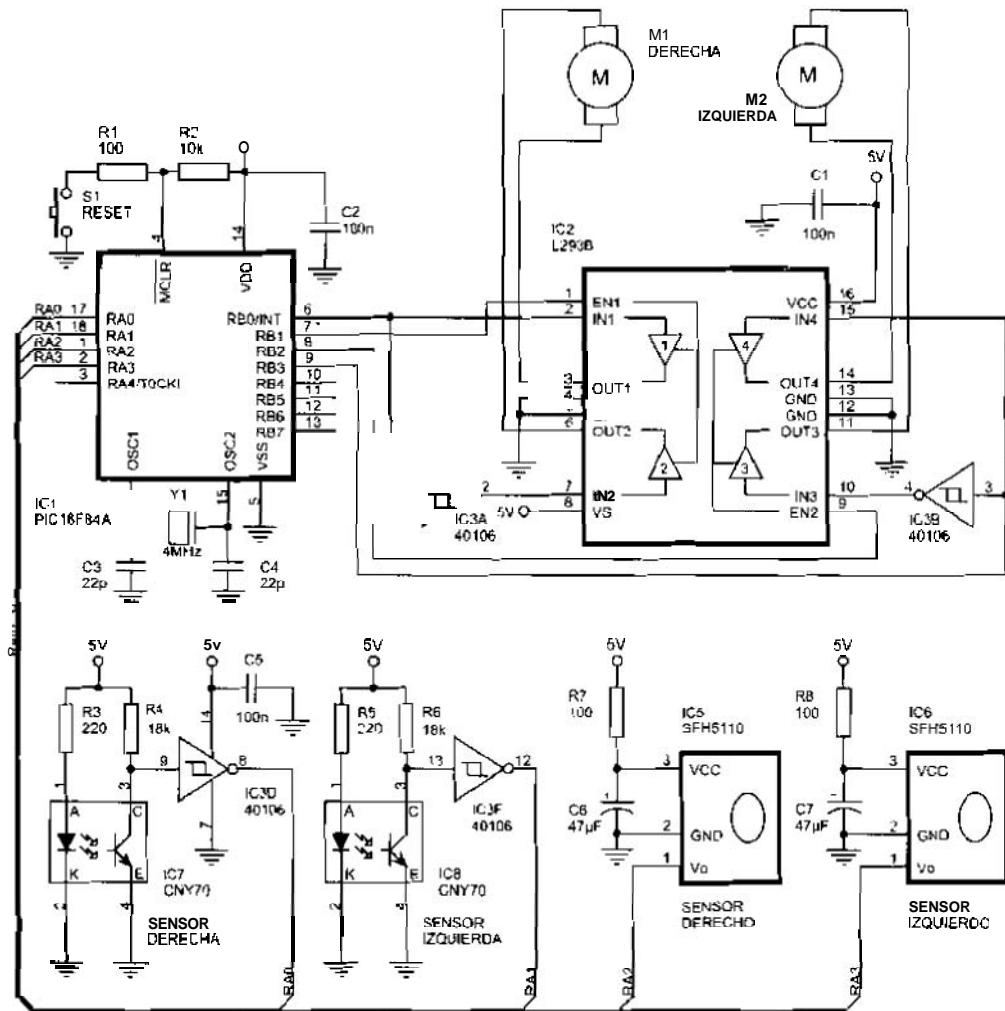
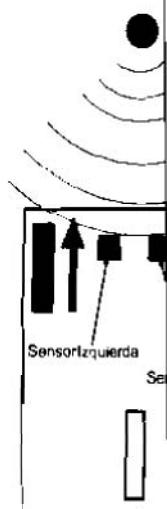


Figura 33-73 Esquema del microbot Trasto con los sensores de infrarrojos

Para el tipo SFH5110 figura 32-19. El esquema de

La estr señal de la ba ruedas deberá SensorIzquier la izquierda h girará en sent SensorIzquier SensorDerech izquierda en SensorIzquier 33-24 aclara lo



### 33.7.4 Proyecto

Si se h fácilmente el pr

; Programa de cont ; una señal infrarroj

Para conseguirlo añadiremos en el frontal de nuestro microbut dos **sensores** del tipo SFH5110-38 montados como detector de señal infrarroja, según se indicaba en la figura 32-19, y que denominaremos SensorDerecho y SensorIzquierdo, respectivamente. El esquema completo es el que se muestra en la figura 33-23.

La estrategia que vamos a seguir es la siguiente. Si los **dos sensores detectan** la señal de la baliza el microrobot deberá desplazarse en línea recta y por lo tanto, las dos ruedas deberán de girar en sentido de avance. En caso contrario se comprueba si sólo el SensorIzquierdo detecta la señal infrarroja, en caso afirmativo el microbot deberá girar a la izquierda hasta que el SensorDerecha detecte la señal, por lo que la rueda derecha girará en sentido de avance y la izquierda en sentido de retroceso. Por el contrario, si SensorIzquierdo no detecta el haz infrarrojo modulado se comprueba si lo hace el SensorDerecho, en caso afirmativo el robot gira a la derecha, haciendo girar la rueda izquierda en sentido de avance y la derecha en sentido de retroceso, hasta que SensorIzquierdo detecte la señal infrarroja. Se repite el ciclo de forma continua. La figura 33-24 aclara los movimientos que seguirá el microbot.

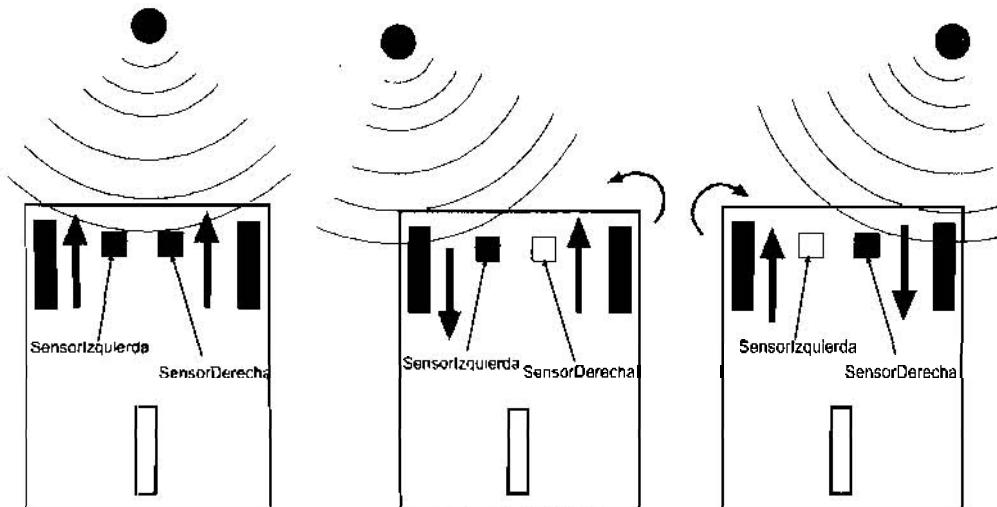


Figura 22-24 Estrategia a seguir para llegar a la baliza

### 33.7.4 Programa de robot detector de baliza

Si se han seguido los razonamientos del apartado anterior se comprenderá fácilmente el programa Robot\_Baliza\_01.asm.

```
;***** Robot_Baliza_01.asm *****
; Programa de control para Microbot TRASTO el cual detecta una baliza que genera
; una señal infrarroja modulada a 38 kHz.
```

; Los sensores ópticos SFH5110 están situados en la parte frontal del microbot:  
; El sensor de la derecha está conectado a RA2 y el sensor de la izquierda a RA3.

; Cuando el sensor SFH5110 detecta luz infrarroja modulada, proporciona un nivel bajo en su  
; línea de salida.

; El programa adopta la estrategia siguiente:

- Si no se detecta la baliza por ningún sensor el microbot gira siempre a la derecha.
- Si los dos sensores detectan portadora el microbot avanza hacia adelante.
- Si se detecta portadora en el sensor de la izquierda y no en el de la derecha el microbot gira a la izquierda hasta que los dos sensores detecten la baliza.
- Si se detecta portadora en el sensor de la derecha y no en el de la izquierda el microbot gira a la derecha hasta que los dos sensores detecten la baliza.

; ZONA DE DATOS \*\*\*\*\*

```
_CONFIG _CP_OFF & _WDT_OFF & _PWRTE ON & _XT_OSC
LIST P=16F84A
INCLUDE <P16F84A.INC>
```

```
#DEFINE SensorDerecha PORTA,2 ; Sensor Derecho.
#DEFINE SensorIzquierda PORTA,3 ; Sensor Izquierdo.
```

; ZONA DE CÓDIGOS \*\*\*\*\*

|               | ORG   | 0               |                                                       |
|---------------|-------|-----------------|-------------------------------------------------------|
| Inicio        | bsf   | STATUS,RP0      | ; Selecciona Banco 1 de registros.                    |
|               | bsf   | SensorDerecha   | ; Estas líneas se configuran como entrada.            |
|               | bsf   | SensorIzquierda |                                                       |
|               | clrf  | PORTB           | ; Las líneas del Puerto B se configuran como salidas. |
|               | bcf   | STATUS,RP0      | ; Selecciona Banco 0 de registros.                    |
| Principal     | btfsc | SensorDerecha   | ; ¿Ha detectado señal por la derecha?                 |
|               | goto  | Ver-Izquierda   | ; No recibe por la derecha.                           |
|               | btfsc | SensorIzquierda | ; Si, ¿también señal por la izquierda?                |
|               | goto  | GiroDerecha     | ; No, solo señal por la derecha, gira a derecha.      |
|               | movlw | b'00000111'     | ; Sí, recibe por los dos sensores. Sigue recto.       |
|               | goto  | ActivaSalida    |                                                       |
| Ver-Izquierda | btfsc | SensorIzquierda | ; Por la derecha no recibe. ¿Y por la izquierda?      |
|               | goto  | GiroDerecha     | ; Tampoco, ni por la derecha ni por la izquierda.     |
| GiroIzquierda | movlw | b'00000111'     | ; Gira a la izquierda.                                |
|               | goto  | ActivaSalida    |                                                       |
| GiroDerecha   | movlw | b'00000110'     | ; Gira a la derecha.                                  |
| ActivaSalida  | movwf | PORTB           |                                                       |
|               | goto  | Principal       |                                                       |
| END           |       |                 |                                                       |

## CARACT

- La arqu
- El sct d
- Todas l
- Velocid
- Frecuen
- (PIC16
- Memor
- Memor
- Memor
- Instruc
- Los da
- Dispon
- La pila
- Dispon
- deshab

## APÉNDICE A

# CARACTERÍSTICAS TÉCNICAS DEL PIC16F84A

### CARACTERÍSTICAS DE LA CPU RISC

- La arquitectura de la CPU es del tipo Harvard.
- El set de instrucciones tiene 35 instrucciones de una sola palabra.
- Todas las instrucciones duran un ciclo máquina, excepto las de salto que duran dos.
- Velocidad de operación:
  - DC 20 MHz, para la frecuencia del reloj de entrada.
  - DC – 200 ns, para la duración del ciclo máquina.
- Frecuencia máxima de funcionamiento de 4 MHz (PIC16F84A-04) o 20 MHz (PIC16F84A-20).
- Memoria de programa tipo Flash de 1024 posiciones.
- Memoria RAM de datos de 68 bytes.
- Memoria EEPROM de datos de 64 bytes.
- Instrucciones con una longitud de 14 bits.
- Los datos tienen una longitud de 1 byte (8 bits)
- Dispone de 15 registros de funciones especiales.
- La pila tiene 8 niveles de profundidad.
- Dispone de cuatro fuentes de interrupción, las cuales pueden ser habilitadas o deshabilitadas independientemente por software:
  - Externa por el pin RB0/INT.
  - Por desbordamiento del Timer 0.
  - Por cambio en las líneas PORTB <7:4>
  - Por finalización de escritura de la memoria EEPROM de datos.

## ENCAPSULADO DIL-18

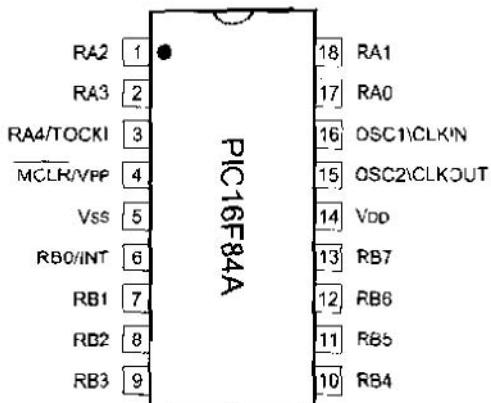


Figura A-1 Microcontrolador PIC16F84A

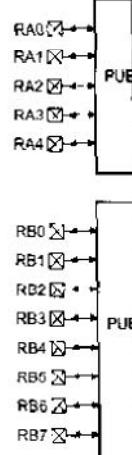
## CARACTERÍSTICAS DE LOS PERIFÉRICOS

- Dispone de 13 líneas de entrada/salida con control individual de dirección.
- Alta capacidad de corriente por terminal. Proporciona suficiente corriente para gobernar un LED:
  - Consumo 25 mA por pin cuando está a nivel bajo.
  - Proporciona 20 mA por pin cuando está a nivel alto.
- Dispone de un Temporizador/Contador de 8 bits (TMRO) con división de frecuencia programable.

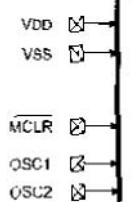
## CARACTERÍSTICAS ESPECIALES DEL MICROCONTROLADOR

- La memoria Flash de programas admite hasta 1.000 ciclos de borrado y escritura.
- La memoria EEPROM de datos admite hasta 1.000.000 de ciclos de borrado y escritura.
- Garantiza una retención de datos para la memoria EEPROM de datos superior a los 40 años.
- Se puede programar en el circuito vía serie mediante dos pines, ICSP (*In Circuit Serial Programming*)
- Power-On Reset (PON), Power-Up Timer (PWRT), Oscillator Start-Up Timer (OST)
- Dispone de un temporizador Watchdog (WDT) con su propio oscilador RC para un funcionamiento fiable.
- Protección de código de programa mediante la activación de un bit de protección.
- Modo de bajo consumo SLEEP.
- Tipo de oscilador seleccionable.

ARQUITEC



64 POSICIONES  
DE MEMORIA DE  
DATOS  
EEPROM



CARACTEI

- Tensión (excep)
- Tensión
- Tensión

## ARQUITECTURA INTERNA

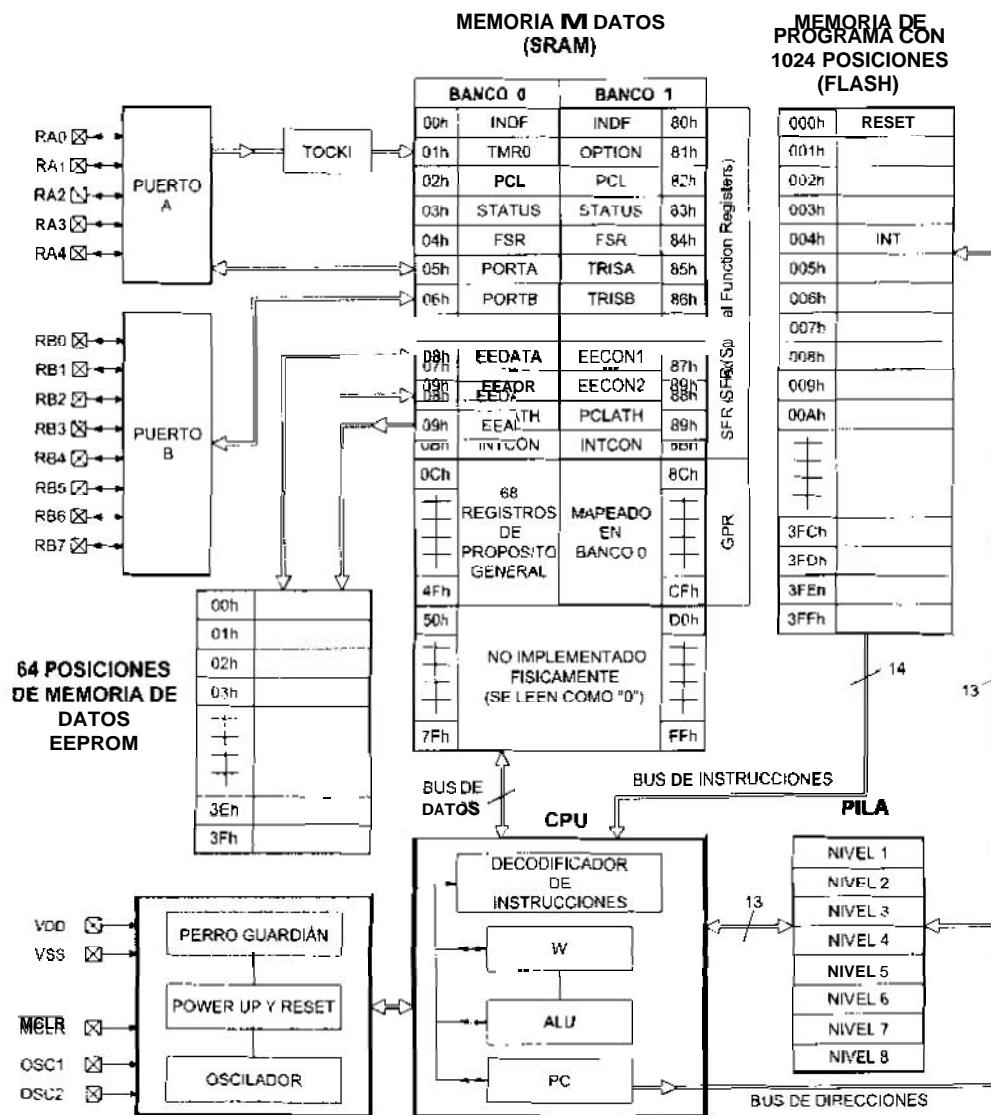


Figura A-2 Arquitectura interna del PIC16F84A

## CARACTERÍSTICAS ELÉCTRICAS MÁXIMAS ADMISIBLES

- Tensión de cualquier pin respecto de V<sub>SS</sub> (excepto V<sub>DD</sub>, MCLR y RA4) ..... -0,3 V a (V<sub>D</sub>, +0,3 V)
- Tensión en V<sub>DD</sub> respecto de V<sub>SS</sub> ..... -0,3 a +7,5V
- Tensión en MCLR respecto de V<sub>SS</sub> ..... -0,3 a +14V

- Tensión en RA4 respecto de V<sub>SS</sub> ..... -0,3 a +8,5V
- Potencia de dissipación total ..... 800 mW
- Máxima corriente por el pin V<sub>SS</sub> ..... 150 mA
- Máxima corriente por el pin V<sub>DD</sub> ..... 100 mA
- Máxima corriente de salida en bajo por cualquier pin I/O ..... 25 mA
- Máxima corriente de salida en alto por cualquier pin I/O ..... 20 mA
- Máxima corriente de salida en bajo por el conjunto del Puerto A ..... 80 mA
- Máxima corriente de salida en alto por el conjunto del Puerto A ..... 50 mA
- Maxima corriente de salida en bajo por el conjunto del Puerto B ..... 150 mA
- Máxima corriente dc de salida en alto por el conjunto del Puerto A ..... 100 mA

## CARACTERÍSTICAS ELÉCTRICAS PIC16F84A

| Símbolo                            | Característica                                                                                                     | Min                                                   | Típ            | Max                                                               | Unid        | Condiciones                                                                                                                          |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|----------------|-------------------------------------------------------------------|-------------|--------------------------------------------------------------------------------------------------------------------------------------|
| V <sub>DD</sub>                    | Tensión de alimentación                                                                                            | 4,0                                                   | ---            | 5,5                                                               | V           | Configuraciones XT,RC y LP.                                                                                                          |
| I <sub>DD</sub>                    | Corriente de alimentación                                                                                          |                                                       | 1,8<br>3       | 4,5<br>10                                                         | mA<br>mA    | Configuración RC y XT:<br>FOSC=4MHz, V <sub>DD</sub> =5,5V<br>FOSC=4MHz, V <sub>DD</sub> =5,5V<br>(Durante programación de la Flash) |
| I <sub>PP</sub>                    | Consumo en standby                                                                                                 | ---                                                   | 1,0            | 14                                                                | μA          | Watchdog inhabilitado.                                                                                                               |
| V <sub>H</sub>                     | Tensión de entrada en bajo<br>Puertos I/O:<br>- con buffer TTL.<br>- con buffer Trigger Schmitt<br>MCLR, RA4/TOCK1 | V <sub>SS</sub><br>V <sub>SS</sub><br>V <sub>SS</sub> | --<br>--<br>-- | 0,8<br>0,2 V <sub>DD</sub><br>0,2 V <sub>DD</sub>                 | V<br>V<br>V | 4,5V ≤ V <sub>DD</sub> ≤ 5,5V                                                                                                        |
| V <sub>HI</sub>                    | Tensión de entrada en alto<br>Puertos I/O:<br>- con buffer TTL<br>- con buffer Trigger Schmitt<br>MCLR, RA4/TOCK1  | 2,0<br>0,8 V <sub>DD</sub><br>0,8 V <sub>DD</sub>     | --<br>--<br>-- | 0,8 V <sub>DD</sub><br>0,8 V <sub>DD</sub><br>0,8 V <sub>DD</sub> | V<br>V<br>V | 4,5V ≤ V <sub>DD</sub> ≤ 5,5V                                                                                                        |
| I <sub>H</sub>                     | Corriente de entrada en bajo<br>Puertos I/O<br>MCLR, RA4/TOCK1                                                     | --<br>--                                              | --<br>--       | ±1<br>±5                                                          | μA<br>μA    | V <sub>DD</sub> ≤ V <sub>HH</sub> ≤ V <sub>DD</sub>                                                                                  |
| V <sub>OL</sub>                    | Tensión de salida en bajo<br>Puertos I/O                                                                           | --                                                    | --             | 0,6                                                               | V           | I <sub>OL</sub> = 8,5 mA, V <sub>DD</sub> = 4,5 V                                                                                    |
| V <sub>OH</sub>                    | Tensión de salida en alto<br>Puertos I/O                                                                           | V <sub>DD</sub> -0,7                                  | --             | --                                                                | V           | I <sub>OH</sub> = -3 mA, V <sub>DD</sub> = 4,5 V                                                                                     |
| V <sub>ON</sub>                    | Tensión drenador abierto alto<br>Pin RA4                                                                           | --                                                    | --             | 8,5                                                               | V           |                                                                                                                                      |
| E <sub>U</sub><br>T <sub>ULW</sub> | Memoria EEPROM de datos:<br>Duración<br>Tiempo de ciclo escritura                                                  | 1M<br>--                                              | 10M<br>4       | —<br>8                                                            | L/W<br>ms   | 25°C a 5 V                                                                                                                           |
| E <sub>P</sub><br>T <sub>PLW</sub> | Memoria Flash de programa<br>Duración<br>Tiempo de ciclo escritura                                                 | 100<br>--                                             | 1000<br>4      | —<br>8                                                            | E/W<br>ms   |                                                                                                                                      |

El PIC16F84A tiene 35 instrucciones. El código de operación es similar a las instrucciones de la Tabla B-1.

La nomenclatura:

- En las instrucciones:
  - Siempre se incluye el nombre de la instrucción.
  - Siempre se incluye el nombre de la operación.
- En las instrucciones de control del bit de dirección, siempre el bit de dirección.
- En las instrucciones de control literal que incluyen:

Las 35 instrucciones RISC, que no sólo incluyen las instrucciones cálculo.

- Las instrucciones de control de tiempo de ejecución.
- Las instrucciones de control de salto, condicional.
- Las instrucciones para utilizar como memoria.

0,3 a +8,5V  
.....800 mW  
.....150 mA  
.....100 mA  
.....25 mA  
.....20 mA  
.....80 mA  
.....50 mA  
.....150 mA  
.....100 mA

## APÉNDICE B

# REPERTORIO DE INSTRUCCIONES

El PIC16F84 está compuesto por una CPU de tipo RISC con un juego de 35 instrucciones. En los 14 bits que forman las instrucciones máquina del PIC se incluyen el código de operación propiamente dicho y los operandos, en caso de que haya. Las 35 instrucciones de que consta este microcontrolador son las que se muestran clasificadas en la Tabla B-1.

La nomenclatura que utilizan estas instrucciones es la siguiente:

- En las instrucciones la letra f representa **un registro** y d representa **el destino**:
  - Si d es "0" el resultado de la operación se sitúa en el registro de trabajo W.
  - Si d es "1" el resultado se sitúa en el mismo registro f.
- En las instrucciones que manejan bits, b representa en binario la posición (0-7) del bit deseado dentro del byte, f representa el byte o registro. El bit 0 es siempre el bit de menor peso.
- En las instrucciones con literales y de control k representa una **constante o literal** que según los casos puede ser de 8 u 11 bits.

Las 35 instrucciones del PIC16F84 cumplen las características de un procesador RISC, que no sólo supone tener un juego de instrucciones reducido sino que, además, sus instrucciones cumplen las siguientes características:

- **Las instrucciones son simples y rápidas.** Todas las instrucciones tienen un tiempo de ejecución de un ciclo máquina (4 ciclos de reloj) a excepción de las de salto, que tienen un tiempo de ejecución de 2 ciclos máquina.
- **Las instrucciones son ortogonales.** Casi todas las instrucciones pueden utilizar cualquier operando.

- La longitud de las instrucciones y los datos es constante. Todas las instrucciones tienen una longitud de 14 bits y los datos una longitud de 1 byte.

## DESCRIPCIÓN DE LOS CÓDIGOS UTILIZADOS

|                                     |                                                                                                                                                                                         |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>f</b>                            | Dirección del registro (de 0x00 hasta 0x7F)                                                                                                                                             |
| <b>W</b>                            | Registro de trabajo.                                                                                                                                                                    |
| <b>b</b>                            | Posición de un bit dentro de un registro de Y bits                                                                                                                                      |
| <b>k</b>                            | Literal, dato constante o etiqueta                                                                                                                                                      |
| <b>x</b>                            | Valor indeterminado que puede ser "0" o "1"                                                                                                                                             |
| <b>d</b>                            | Destino seleccionado: si <b>d</b> = 0, el resultado se guarda en <b>W</b> , si <b>d</b> = 1, el resultado de la operación se guarda en el registro <b>f</b> . El valor por defecto es - |
| <b>TOS</b>                          | Parte superior de la pila o <i>Stack</i>                                                                                                                                                |
| <b>PC</b>                           | Contador de programa ( <i>Program Counter</i> )                                                                                                                                         |
| <b>TO</b>                           | Bit de <i>Timer Out</i>                                                                                                                                                                 |
| <b>PD</b>                           | Bit de <i>Power Down</i>                                                                                                                                                                |
| <b>[]</b>                           | Opcional                                                                                                                                                                                |
| <b>()</b>                           | Contenido                                                                                                                                                                               |
| <b>→</b>                            | Sentido de la transferencia                                                                                                                                                             |
| <b>Campo del bit de un registro</b> |                                                                                                                                                                                         |

|       |     |
|-------|-----|
| addlw | k   |
| addwf | f,d |
| decf  | f,d |
| incf  | f,d |
| sublw | k   |
| subwf | f,d |
| andlw | k   |
| andwf | f,d |
| comf  | f,d |
| iorlw | k   |
| iorwf | f,d |
| rlf   | f,d |
| rff   | f,d |
| swapf | f,d |
| xorlw | k   |
| xorwf | f,d |

|        |     |
|--------|-----|
| btfsc  | f,b |
| btfss  | f,b |
| decfsz | f,d |
| incfsz | f,d |
| goto   | k   |

|        |   |
|--------|---|
| call   | k |
| retfie |   |
| retlw  | k |
| return |   |

|        |  |
|--------|--|
| clrwdt |  |
| nop    |  |
| sleep  |  |

## SET DE INSTRUCCIONES DEL PIC16F84

| NEMÓNICO               | DESCRIPCIÓN                       | CÓDIGO DE OPERACIÓN | FLAGS AFECTADOS |
|------------------------|-----------------------------------|---------------------|-----------------|
| Instrucciones de CARGA |                                   |                     |                 |
| clrf f                 | 00 → (f)                          | 00 0001 1fff ffff   | Z               |
| clrw                   | 00 → (W)                          | 00 0001 0xxx xxxx   | Z               |
| movf f,d               | (f) → (destino)                   | 00 1000 dfff ffff   | Z               |
| movlw k                | k 3 (W)                           | 11 00xx kkkk kkkk   | Ninguno         |
| movwf f                | (W) → (f)                         | 00 0000 1fff ffff   | Ninguno         |
| Instrucciones de BIT   |                                   |                     |                 |
| bcf f,b                | Pone a 0 el bit 'b' del reg. 'f'. | 01 00bb bfff ffff   |                 |
| bsf f,b                | Pone a 1 el bit 'b' del reg. 'f'. | 01 01bb bfff ffff   | Ninguno         |

Todas las  
d de 1 byte.

i d = 1, el  
defecto es

FLAGS  
PECTADOS

Z  
Z  
Z

Ninguno  
Ninguno

Ninguno  
Ninguno

| Instrucciones ARITMÉTICAS             |                                                     |                   |          |  |
|---------------------------------------|-----------------------------------------------------|-------------------|----------|--|
| addlw k                               | (W) + k → (W)                                       | 11 111x kkkk kkkk | C, DC, Z |  |
| addwf f,d                             | (W) + (f) → (destino)                               | 00 0111 dfff ffff | C, DC, Z |  |
| decf f,d                              | (f) - 1 → (destino)                                 | 00 0011 dfff ffff | Z        |  |
| incf f,d                              | (f) + 1 → (destino)                                 | 00 1010 dfff ffff | Z        |  |
| sublw k                               | k - (W) → W                                         | 11 110x kkkk kkkk | C, DC, Z |  |
| subwf f,d                             | (f) - (W) → (destino)                               | 00 0010 dfff ffff | C, DC, Z |  |
| Instrucciones LÓGICAS                 |                                                     |                   |          |  |
| andlw k                               | (W) AND k → (W)                                     | 11 1001 kkkk kkkk | Z        |  |
| andwf f,d                             | (W) AND (f) → (destino)                             | 00 0101 dfff ffff | Z        |  |
| comf f,d                              | (f) → (destino)                                     | 00 1001 dfff ffff | Z        |  |
| iorlw k                               | (W) OR k → (W)                                      | 11 1000 kkkk kkkk | Z        |  |
| iorwf f,d                             | (W) OR (f) → (destino)                              | 00 0100 dfff ffff | Z        |  |
| rlf f,d                               | Rota (f) a izquierda a través del Carry → (destino) | 00 1101 dfff ffff | C        |  |
| rrf f,d                               | Rota (f) a derecha a través del Carry → (destino)   | 00 1100 dfff ffff | C        |  |
| swapf f,d                             | Intercambia los nibbles de f → (destino)            | 00 1110 dfff ffff | Ninguno  |  |
| xorlw k                               | (W) XOR k → (W)                                     | 11 1010 kkkk kkkk | Z        |  |
| xorwf f,d                             | (W) XOR (f) → (destino)                             | 00 0110 dfff ffff | Z        |  |
| Instrucciones de SALTO                |                                                     |                   |          |  |
| btfsc f,b                             | Salta si el bit 'b' del 'f' es 0                    | 01 10bb bfff ffff | Ninguno  |  |
| btfss f,b                             | Salta si el bit 'b' del 'f' es 1                    | 01 11bb bfff ffff | Ninguno  |  |
| deefsz f,d                            | (f)-1 → destino y salta si es 0                     | 00 1011 dfff ffff | Ninguno  |  |
| incfsz f,d                            | (f)+1 → destino y salta si es 0                     | 00 1111 dfff ffff | Ninguno  |  |
| goto k                                | Salta a la dirección 'k'                            | 10 1kkk kkkk kkkk | Ninguno  |  |
| Instrucciones de manejo de SUBRUTINAS |                                                     |                   |          |  |
| call k                                | Llamada a subrutina                                 | 10 0kkk kkkk kkkk | Ninguno  |  |
| retfie                                | Retorno de una interrupción                         | 00 0000 0000 1001 | Ninguno  |  |
| retlw k                               | Retorno con un literal en W                         | 11 01xx kkkk kkkk | Ninguno  |  |
| return                                | Retorno de una subrutina                            | 00 0000 0000 1000 | Ninguno  |  |
| Instrucciones ESPECIALES              |                                                     |                   |          |  |
| clwdt                                 | Borra Timer del Watchdog                            | 00 0000 0110 0100 | /TO, /PD |  |
| nop                                   | No operación                                        | 00 0000 0xx0 0000 | Ninguno  |  |
| sleep                                 | Entra en modo bajo consumo                          | 00 0000 0110 0011 | /TO, /PD |  |

Tabla B-1 Repertorio de instrucciones del PIC16F84

**addlw****Suma el literal k con W**

Sintaxis: addlw k

Operandos:  $0 \leq k \leq 255$ Operación:  $(W) + k \rightarrow (W)$ 

(Add Literal to W)

Flags afectados: C, DC, Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 11 | 111x | kkkk | kkkk |
|----|------|------|------|

Descripción: Suma el contenido del registro W al literal 'k' y almacna el resultado en W. Si se produce acarreo el flag C se pone a "1".

Ejemplo 1: addlw 0x15 ;  $(W) + 0x15 \rightarrow (W)$ Antes instrucción:  $(W) = 0x10$ , y C = ?Después instrucción:  $(W) = 0x10 + 0x15 = 0x25$  y C = 0. $(W) = b'0001\ 0000' + b'0001\ 0101' = b'0010\ 0101'$ Ejemplo:  
Antes instrucción:  
Después instrucción:**andwf**Sintaxis:  
Operandos:Operación:  
Flags afectados:

Código de OP :

Descripción:

**addwf****Suma W con el registro f**

Sintaxis: addwf f,d

Operandos:  $0 \leq k \leq 127$  $d \in [0,1]$ Operación:  $(W) + (f) \rightarrow (\text{destino})$ 

(Add W and f)

Flags afectados: C, DC, Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0111 | ffff | ffff |
|----|------|------|------|

Descripción: Suma el contenido del registro W al contenido del registro 'f' y almacna el resultado en W si d = 0, y en el registro f si d = 1. Si se produce acarreo el flag C se pone a "1".

Ejemplo: addwf Registro,0 ;  $(\text{Registro}) + (W) \rightarrow (W)$ Antes instrucción:  $(W) = 0x17$ ,  $(\text{Registro}) = 0xC2$ , y C = ?Después instrucción:  $(W) = 0xD9$ ,  $(\text{Registro}) = 0xC2$ , y C = 0.Ejemplo:  
Antes instrucción:  
Después instrucción:**bcf**Sintaxis:  
Operandos:Operación:  
Flags afectados:  
Código de OP :

Descripción:

**andlw****W AND Literal k**

Sintaxis: andlw k

Operandos:  $0 \leq k \leq 255$ Operación:  $(W) \text{ AND } (k) \rightarrow (W)$ 

(AND Literal with W)

Flags afectados: Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 11 | 1001 | kkkk | kkkk |
|----|------|------|------|

Descripción: Efectúa la operación AND lógica entre el contenido del registro W y el literal 'k', el resultado se almacna en W.

Ejemplo:  
Antes instrucción:  
Después instrucción:**bsf**Sintaxis:  
Operandos:

Operación:

Ejemplo: andlw b'01011111'; (W) AND b'01011111' → (W).  
 Antes instrucción: (W) = b'10100011' y Z=?  
 Despues instrucción: (W) = b'00000011' y Z=0.

**andwf****W AND f**

Sintaxis: andwf f,d  
 Operandos:  $0 \leq f \leq 127$   
 $d \in [0,1]$   
 Operación: (W) AND (f) → (destino)

Flags afectados: Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0101 | dfdf | ffff |
|----|------|------|------|

Descripción: Efectúa la operación AND lógica entre el contenido del registro W y el contenido del registro 'f' y almacena el resultado en W si d = 0, y en 'f' si d = 1.

Ejemplo: andwf Registro,1 ; (W) AND (Registro) → (Registro)  
 Antes instrucción: (W) = b'00101111', (Registro) = b'11000010' y Z=?  
 Despues instrucción: (W) = b'00101111', (Registro) = b'00000010' y Z=0.

**bcf****Borra un bit de f**

Sintaxis: bcf f,b  
 Operandos:  $0 \leq f \leq 127$   
 $0 \leq b \leq 7$   
 Operación: 0 → (f<b>)

Flags afectados: Ninguno

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 01 | 00bb | bfff | ffff |
|----|------|------|------|

Descripción: Pone a cero el bit número 'b' del registro 'f'.

Ejemplo: bcf FlagReg,7 ; 0 → (FlagReg,7)  
 Antes instrucción: (FlagReg) = b'11000111'.  
 Despues instrucción: (FlagReg) = b'01000111'.

**bsf****Activa un bit de f**

Sintaxis: bsf f,b  
 Operandos:  $0 \leq f \leq 127$   
 $0 \leq b \leq 7$   
 Operación: 1 → (f<b>)

Flags afectados: Ninguno

Código de OP: **01 | 0lbh | bfff | fff**

Descripción: Pone a uno el bit número 'b' del registro 'f'.

Ejemplo: **bsf FlagReg,7 ; 1 → (FlagReg,7)**

Antes Instrucción: **(FlagReg) = b'01000111'**.

Después Instrucción: **(FlagReg) = b'11000111'**.

**Ejemplo:**

**A**  
**F**  
**V**

**Antes instrucción**  
**Después instrucción**

## btfsc

### Test de bit de f y salta si es cero

Sintaxis: **btfsc f,b**

Operandos: **0 ≤ f ≤ 127**

**0 ≤ b ≤ 7**

Operación: **Salta si (f<b>)=0**

*(Bit Test f, Skip if Clear)*

Flags afectados: Ninguno

Código de OP: **01 | 10bb | bfff | ffff**

Descripción: Si el bit número b del registro f es cero la instrucción que sigue a ésta se ignora y se trata como un "nop". En este caso, y sólo en este caso, la instrucción *btfsc* precisa dos ciclos para ejecutarse.

Ejemplo:

|        |              |                 |                                                       |
|--------|--------------|-----------------|-------------------------------------------------------|
| Aquí   | <b>btfsc</b> | <b>Flag,1</b>   | <b>; Si el bit 1 del registro Flag es cero salta.</b> |
| Falso  | <b>goto</b>  | <b>ProcesoX</b> | <b>; Ha sido uno.</b>                                 |
| Verdad | <b>...</b>   | <b>...</b>      | <b>; Ha sido cero.</b>                                |
|        |              | <b>...</b>      |                                                       |

Antes instrucción: **(PC) = Dirección de "Aquí".**

Después instrucción: **Si el bit Flag <1> = 0 (PC) = Dirección de "Verdad".**

**Si el bit Flag <1> = 1 (PC) = Dirección de "Falso".**

## call

Sintaxis:

Operandos:

Operación:

Flags afectados:

Código de OP:

Descripción:

**Ejemplo:**

**Antes instrucción**

**Después instrucción**

## btfss

### Test de bit de f y salta si es uno

Sintaxis: **btfss f,b**

Operandos: **0 ≤ f ≤ 127**

**0 ≤ b ≤ 7**

Operación: **Salta si (f<b>)=1**

*(Bit Test f, Skip if Set)*

Flags afectados: Ninguno

Código de OP: **01 | 11bb | bfff | ffff**

Descripción: Si el bit número b del registro f es uno la instrucción que sigue a esta se ignora y se trata como un "nop". En este caso, y sólo en este caso, la instrucción *btfss* precisa dos ciclos para ejecutarse.

## clrf

Sintaxis:

Operandos:

Operación:

Flags afectados:

Código de OP:

Descripción:

**Ejemplo:**

**Antes instrucción**

**Después instrucción**

**Ejemplo:**

|        |      |          |                                                 |
|--------|------|----------|-------------------------------------------------|
| Aqui   | btfs | Flag, l  | ; Si el bit l del registro "Flag" es uno salta. |
| Falso  | goto | ProcesoX | ; Ha sido cero.                                 |
| Verdad | ...  |          | ; Ha sido uno.                                  |

**Antes instrucción:** (PC) = Dirección de "Aqui".

**Después instrucción:** Si el bit Flag <1> = 0, (PC) = Dirección de "Falso".

Si el bit Flag <1> = 1, (PC) = Dirección de "Verdad".

es cero

**call****Llamada a Subrutina**

Sintaxis: call k

Operandos:  $0 \leq k \leq 2047$

Operación:  $(PC)+1 \rightarrow TOS$  *(Call Subroutine)*  
 $k \rightarrow (PC<10:0>),$   
 $(PCLATH<4:3>) \rightarrow (PC<12:11>)$

Flags afectados: Ninguno

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 10 | 0kkk | kkkk | kkkk |
|----|------|------|------|

Descripción: Salvaguarda la dirección de vuelta en la Pila y después llama a la subrutina situada en la dirección cargada en el PC. Tarda dos ciclos máquina en ejecutarse.

Ejemplo: Aquí call Allí

Antes instrucción: (PC)= Dirección de "Aquí".

Después instrucción: (PC) = Dirección de "Allí".

(TOS) = Dirección de "Aquí" + 1.

**clrf****Borra f**

Sintaxis: clrf f

Operandos:  $0 \leq f \leq 137$

Operación:  $00h \rightarrow (f)$  *(Clear f)*  
 $1 \rightarrow Z$

Flags afectados: Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0001 | 1fff | ffff |
|----|------|------|------|

Descripción: Se borra el contenido del registro 'f' y el flag Z se activa poniéndose a "1".

Ejemplo: clrf FlagReg ; 0 → (FlagReg).

Antes instrucción: (FlagReg) = ? y Z = ?

Después instrucción: (FlagReg) = 0x00 y Z = 1.

kip if Clear)

que a ésta se  
este caso, la

salta.

es uno

Skip if Set)

je a ésta se  
ste caso, la

**clrw****Borra el registro W**

Flags afe  
Código d  
Descripc

Sintaxis: clrw  
 Operandos: Ninguno  
 Operación:  $00h \rightarrow (W)$   
 $1 \rightarrow Z$  *(Clear W)*

Flags afectados: Z  
 Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0001 | 0xxx | xxxx |
|----|------|------|------|

  
 Descripción: El registro de trabajo W se carga con 00h. El flag Z se pone a “1”.

Ejemplo: clrw ;  $0 \rightarrow (W)$

Antes instrucción:  $(W) = ?$  y  $Z = ?$

Después instrucción:  $(W) = 0x00$  y  $Z = 1$ .

Ejemplo:  
Antes ins  
Despues

**decf**

Sintaxis:  
Operand

Operació  
Flags afe  
Código d  
Descripc

Ejemplo:  
Antes ins  
Despues

**decfs**

Sintaxis:  
Operande

Operació  
Flags afe  
Código d  
Descripc

**clrwdt****Borra el Timer del Watchdog**

Sintaxis: clrwdt  
 Operandos: Ninguno  
 Operación:  $00h \rightarrow WDT$   
 $0 \rightarrow \text{WDT prescaler}$   
 $1 \rightarrow \overline{T0}$   
 $1 \rightarrow \overline{PD}$  *(Clear Watchdog Timer)*

Flags afectados: TO , PD

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0000 | 0110 | 0100 |
|----|------|------|------|

Descripción: Se borra el Timer del Watchdog (WDT). Los bits  $\overline{T0}$  y  $\overline{PD}$  del registro de estado se ponen a “1”.

Ejemplo: clrwdt

Antes instrucción:  $(\text{Timer WDT}) = ?$

Después instrucción:  $(\text{Timer WDT}) = 0x00$

bit de estado  $\overline{T0} = 1$

bit de estado  $\overline{PD} = 1$

**comf****Complementa f**

Ejemplo:

Sintaxis: comf f  
 Operandos:  $0 \leq f \leq 127$   
 $d \in [0,1]$   
 Operación:  $(\overline{f}) \rightarrow (\text{destino})$  *(Complement f)*

Flags afectados: Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 1001 | dfff | ffff |
|----|------|------|------|

Descripción: Hace el complemento del contenido del registro 'f' bit a bit. El resultado se almacena en el registro 'f' si d=1 y en el registro W si d=0, en este caso 'f' no varía.

Ejemplo: comf Reg1,0 ; (/Reg) → (W)

Antes instrucción: (Reg1)=b'00010011', (W)=? y Z=?

Después instrucción: (Reg1)=b'00010011', (W)=b'11101100' (invertido unos y ceros) y Z=0.

**decf****Decrementa f**

Sintaxis: decf f

Operandos:  $0 \leq f \leq 127$  $d \in [0,1]$ Operación:  $(f-1) \rightarrow (\text{destino})$ 

(Decrement f)

Flags afectados: Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0011 | dfff | ffff |
|----|------|------|------|

Descripción: Se decrementa el contenido del registro 'f' en una unidad. El resultado se almacena en f si d=1 y en W si d=0, en este caso 'f' no varía.

Ejemplo: decf Contador,1 ; (Contador)-1 → (Contador).

Antes instrucción: (Contador)=0x01 y Z=?

Después instrucción: (Contador)=0x00 y Z=1.

**decfsz****Decrementa f y salta si el resultado es 0**

Sintaxis: decfsz f,d

Operandos:  $0 \leq f \leq 127$  $d \in [0,1]$ Operación:  $(f-1) \rightarrow (\text{destino}); \text{salta si el resultado es "0"}$ . (Decrement f, Skip if 0)

Flags afectados: Ninguno

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 1011 | dfff | ffff |
|----|------|------|------|

Descripción: Decrementa el contenido del registro 'f' en una unidad. el resultado se almacena en 'f' si d=1 y en W si d=0, en este caso, 'f' no varia. Si el resultado del decremento es cero se ignora la siguiente instrucción y en ese caso la instrucción tiene una duración de dos ciclos.

Ejemplo:

|        |        |            |
|--------|--------|------------|
| Aqui   | decfsz | Contador,1 |
|        | goto   | NoEsCero   |
| EsCero |        | ...        |

**incfsz**

**Antes instrucción:** (PC)=Dirección de "Aqui".  
**Después instrucción:** (Contador)= (Contador) - 1 y además:  
 - Si (Contador)=0, (PC)=Dirección de "EsCero".  
 - Si (Contador)≠0, (PC)=Dirección de "Aqui"+1.

Sintaxis:  
 Operandos:

Operación:  
 Flags afectados:  
 Código de OP:  
 Descripción:

**goto****Salto incondicional**

Sintaxis: goto k  
 Operandos:  $0 \leq k \leq 2047$   
 Operación:  $k \rightarrow (PC<10:0>)$  (*Unconditional Branch*)  
 $(PCLATH<4:3>) \rightarrow (PC<12:11>)$

Flags afectados: Ninguno  
 Código de OP: [ 10 | 1kkk | kkkk | kkkk ]  
 Descripción: Salto incondicional. Carga bits 0 al 10 de la constante **k** en el (PC) y los bits 3 y 4 del registro PCLATH en los 11 y 12 del PC. Esta instrucción se ejecuta en dos ciclos máquina.

Ejemplo: goto Principal ; Principal → (PC)  
 Antes instrucción: (PC)=?  
 Despues instrucción: (PC)=Dirección apuntada por la etiqueta "Principal".

Ejemplo:

Antes instrucción:  
 Despues instrucción:

**incf****Incrementa f**

Sintaxis: incf f  
 Operandos:  $0 \leq f \leq 127$   
 $d \in [0,1]$   
 Operación:  $(f)+1 \rightarrow (\text{destino})$  (*Increment f*)  
 Flags afectados: Z  
 Código de OP: [ 00 | 1010 | dfff | Ffff ]  
 Descripción: El contenido del registro 'f' se incrementa en una unidad, si  $d=1$  el resultado se almacena en 'f', si  $d=0$  el resultado se almacena en W, en este caso el resultado de 'f' no vana.

Ejemplo1: incf Contador,1 ;(Contador)+1 → (Contador).  
 Antes instrucción: (Contador)=0xFF y Z=?  
 Despds instrucción: (Contador)=0x00 y Z=1.

**iorlw**

Sintaxis:  
 Operandos:  
 Operación:  
 Flags afectados:  
 Código de OP:  
 Descripción:

Ejemplo:  
 Antes instrucción:  
 Despues instrucción:

**incfsz****Incrementa f y salta si el resultado es 0**

**Sintaxis:** incfsz f,d

**Operandos:**  $0 \leq f \leq 127$

$d \in [0,1]$

**Operación:**  $(f) + 1 \rightarrow (\text{destino})$ ; salta si el resultado es 0. *(Increment f, Skip if 0)*

**Flags afectados:** Ninguno

**Código de OP :**

|    |      |      |      |
|----|------|------|------|
| 00 | 1111 | dfff | ffff |
|----|------|------|------|

**Descripción:** Incrementa el contenido del registro 'f' en una unidad. El resultado se almacena en 'f' si  $d=1$  y en W si  $d=0$ , en este caso 'f' no varía. Si el resultado del incremento es cero se ignora la siguiente instrucción y, en ese caso, la instrucción tiene una duración de dos ciclos máquina.

Ejemplo:

|          |        |            |
|----------|--------|------------|
| Aqui     | incfsz | Contador,l |
| goto     | Bucle  |            |
| Continua | ....   |            |
|          | ....   |            |

**Antes instrucción:** (PC)= Dirección de "Aqui".

**Después instrucción:** (Contador)= (Contador) + 1 y además:

- Si Contador = 0 (256), (PC) = Dirección de "Continua".
- Si Contador  $\neq 0$ , (PC) = Dirección de "Aqui"+1.

**iorlw****OR entre W y el literal k**

**Sintaxis:** iorlw k

**Operandos:**  $0 \leq k \leq 255$

**Operación:** (W) OR (k)  $\rightarrow$  (W)

*(Inclusive OR Literal with W)*

**Flags afectados:** Z

**Código de OP :**

|    |      |      |      |
|----|------|------|------|
| 11 | 1000 | kkkk | kkkk |
|----|------|------|------|

**Descripción:** Efectúa la operación lógica OR entre el contenido del registro W y el literal 'k'. El resultado se almacena en el registro W .

Ejemplo: iorlw b'00110101'; (W) OR b'00110101'  $\rightarrow$  (W)

Antes instrucción: (W) = b'10011010' y Z = ?

Después instrucción: (W) = b'10111111' y Z = 0.

**iorwf****OR entre W y f**

Sintaxis: iorwf f,d

Operandos:  $0 \leq f \leq 127$  $d \in [0,1]$ Operación: (W) OR (f) **Z** (destino)

(Inclusive OR W with f)

Flags afectados: **Z**

Código de OP : [ 00 | 0100 | dfff | ffff ]

Descripción: Efectúa la operación lógica OR entre el contenido del registro W y el contenido del registro 'f'. Almacena el resultado en 'f' si d=1 y en W si d=0.

Ejemplo: iorwf Resultado,0 ;(W) OR (Resultado)  $\rightarrow$  (W)

Antes instrucción: (Resultado) = b'00010011', (W) = b'10010001' y Z = ?

Después instrucción: (Resultado) = b'00010011', (W) = b'10010011' y Z = 0.

Ejemplo:  
Antes ins  
Después i**movw**

Sintaxis:

Operando:

Operació:

Flags afec

Código di

Descripci

Ejemplo:

Antes ins

Después i

**movlw****Mover literal a W**

Sintaxis: movlw k

Operandos:  $0 \leq k \leq 255$ Operación: k  $\rightarrow$  (W)

(Move Literal to W)

Flags afectados: Ninguno

Código de OP : [ 11 | 00xx | kkkk | kkkk ]

Descripción: El registro W se carga con el valor de 8 bits del literal k.

Ejemplo: movlw 0x5A ; 5Ah  $\rightarrow$  (W)

Antes instrucción: (W) = ?

Después instrucción: (W) = 0x5A

**nop**

Sintaxis:

Operando:

Operaciór:

Flags afec

Código de

Descripció

Ejemplo:

**movf****Mover f**

Sintaxis: movf f,d

Operandos:  $0 \leq f \leq 127$  $d \in [0,1]$ Operación: (f)  $\rightarrow$  (destino)

(Move f)

Flags afectados: **Z**

Código de OP : [ 00 | 1000 | dfff | ffff ]

Descripción: El contenido del registro 'f' se carga en el registro destino dependiendo del valor de 'd'. Si d=0 el destino es el registro W, si d=1 el destino es el propio registro 'f'. Esta instrucción permite verificar dicho registro, ya que el flag Z queda afectado.

Sintaxis:  
Operando.  
OperacionFlags afec  
Código de  
Descnpcii

entre W y f*(Move W to f)**(Move W to f)**(Move W to f)*literal a W*(Move Literal to W)**(Move f)**(Move f)*

**Ejemplo:** movf PORTA,0 ;(PORTA) → (W)  
**Antes instrucción:** (PORTA)=0x1A, (W)=? y Z=?  
**Después instrucción:** (PORTA)=0x1A, (W)=0x1A y Z=0

**movwf****Mover W a f**

**Sintaxis:** movwf f  
**Operandos:** 0 ≤ f ≤ 127  
**Operación:** (W) → (f)  
**Flags afectados:** Ninguno

*(Move W to f)*Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0000 | 1fff | ffff |
|----|------|------|------|

Descripción: Mueve el contenido del registro W al registro 'f'.

**Ejemplo:** movwf PORTB ;(W) → (PORTB)**Antes instrucción:** (PORTB)=? y (W)=0x4F.**Después instrucción:** (PORTB)=0x4F y (W)=0x4F.**nop****No Operación**

**Sintaxis:** nop  
**Operandos:** Ninguno  
**Operación:** No operar  
**Flags afectados:** Ninguno

*(No Operation)*Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0000 | 0xx0 | 0000 |
|----|------|------|------|

Descripción: No realiza operación alguna. Consumo un ciclo de instrucción sin hacer nada.

**Ejemplo:** nop**retfie****Retorno de Interrupción**

**Sintaxis:** retfie  
**Operandos:** Ninguno  
**Operación:** TOS → (PC)  
**I → GIE**

*(Return from Interrupt)***Flags afectados:** NingunoCódigo de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0000 | 0000 | 1001 |
|----|------|------|------|

**Descripción:** Carga el PC con el valor que se encuentra en la parte alta de la pila, asegurando así la vuelta de la interrupción. Pone a 1 el bit GIE con el fin de autorizar de nuevo que se tengan en cuenta las interrupciones. Tarda dos ciclos máquina.

**Ejemplo:** `retfie` ; Retorna de la interrupción.  
**Antes instrucción:**  $(PC) = \dots$  y  $GIE = 0$   
**Después instrucción:**  $(PC) = (TOS)$  y  $GIE = 1$

retlw

### Retorno con un literal en W

**Sintaxis:** rctlw  
**Oprandos:**  $0 \leq k \leq 255$   
**Operación:**  $k \rightarrow (W)$   
 $TOS \rightarrow (PC)$  *(Return with Literal in W)*

Flags afectados: Ninguno

Código de OP : 11 01xx kkkk kkkk

(Return with Literal in W)

Descripción: Carga el registro W con el literal 'k' y después carga el PC con el valor que se encuentra en la parte superior de la pila, efectuando así un retorno de subrutina. Tarda dos ciclos máquina.

Ejemplo: call Tabla ; W contiene el valor de offset de tabla

**Tabla** : Comienza la tabla

**addwf PCL,l ;(PC) = (PC) + (W)**

retlw k0

retlw k1

retlw k2

retención: Fin de tabla

**Antes instrucción:** (W) = 0x02

Después instrucción: **(W) = Toma el valor de k2.**

return

## Retorno de subrutina

Flags afectados: Ninguno

Código de OP : **00 0000 0000 1000**  
Descripción: Carga el PC con el valor que se encuentra en la parte superior de la pila.

Ejemplo: `return` ; Retorna de la subrutina

Ejemplo:  
Antes instrucción:

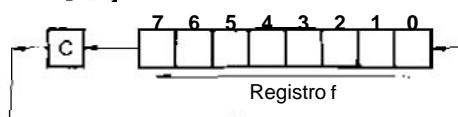
Después instrucción: (BC) = 0?

**rlf****Rota a la izquierda con el Carry**

Sintaxis: rlf f,d

Operandos:  $0 \leq f \leq 127$  $d \in [0,1]$ 

Operación:



(Rotate Left through Carry)

Flags afectados: C

Código de OP : 00 1101 dfff ffff |

Descripción: Rotación de un bit a la **izquierda** del contenido **del registro 'f'** pasando por el bit **de acarreo C**. Si  $d=1$  el **resultado** se almacena en '**f**', si  $d=0$  el resultado se almacena en **W**.

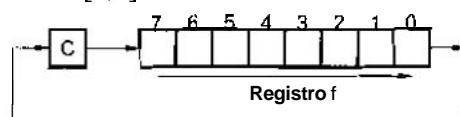
Ejemplo: rlf Reg1,0

Antes instrucción:  $(Reg1) = b'1110\ 0110'$ ,  $W = ?$  y  $C = 0$ .Después instrucción:  $(Reg1) = b'1110\ 0110'$ ,  $(W) = b'1100\ 1100'$  y  $C = 1$ .**rrf****Rota a la derecha con el Carry**

Sintaxis: rrf f,d

Operandos:  $0 \leq f \leq 127$  $d \in [0,1]$ 

Operación:



(Rotate Right through Carry)

Flags afectarlos: C

Código de OP : 00 1100 dfff ffff |

Descripción: Rotación de un bit a la **derecha** del contenido **del registro 'f'** pasando por el bit **de acarreo C**. Si  $d=1$  el **resultado** se almacena en '**f**', si  $d=0$  el resultado se almacena en **W**.

Ejemplo: rrf Reg1,0

Antes instrucción:  $(Reg1) = b'1110\ 0110'$ ,  $(W) = ?$  y  $C = 0$ .Después instrucción:  $(Reg1) = b'1110\ 0110'$ ,  $(W) = b'0111\ 0011'$  y  $C = 0$ .

**sleep****Pasa a Standby o modo de bajo consumo**

Sintaxis: sleep

Operandos: Ninguno

Operación:  $00h \rightarrow \text{WDT}$   
 $0 \rightarrow \text{WDT prescaler}$  $1 \rightarrow \overline{TO}$  $0 \rightarrow \overline{PB}$ Flags afectados:  $\overline{TO}, \overline{PD}$ Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 00 | 0000 | 0110 | 0011 |
|----|------|------|------|

Descripción: Pone el circuito en modo *Sleep* (bajo consumo) con parada del oscilador y Timer 0. (Ver capítulo 16 para más detalles). Se puede salir de este estado por:

- Activación del pin /MCLR para provocar un **reset**
- Desbordamiento del **Watchdog** si quedó operativo en el modo reposo.
- Generación de una interrupción que no sea TMRO, ya que Csta se desactiva con la instrucción *sleep*.

Ejemplo: sleep ; Pasa a "standby" o modo de bajo consumo.

**sublw****Resta el literal k a W**

Sintaxis: sublw k

Operandos:  $0 \leq k \leq 255$ Operación:  $k - (W) \rightarrow (W)$ 

(Subtract W from Literal)

Flags afectados: C, DC, Z

Código de OP : 

|    |      |      |      |
|----|------|------|------|
| 11 | 110x | kkkk | kkkk |
|----|------|------|------|

Descripción: Resta por el método de complemento a 2 al literal 'k' el contenido del registro W. Almacena el resultado en W.

Ejemplo 1: sublw 0x03 ; 03h - (W) → (W)

Antes instrucción: (W) = 0x01, C = ? y Z = ?

Después instrucción: (W) = 0x02, C = 1 (el resultado es positivo) y Z = 0.

Ejemplo 2: sublw 0x02 ; 02h - (W) → (W)

Antes instrucción: (W) = 0x02, C = ? y Z = ?

Después instrucción: (W) = 0x00, C = 1 y Z = 1 (el resultado es cero).

Ejemplo 3: sublw 0x02 ; 02h - (W) → (W)

Antes instrucción: (W) = 0x03, (+3 en decimal), C = ? y Z = ?

Después instrucción: (W) = 0xFF, (-1 en decimal), C = 0 (resultado negativo) y Z = 0.

**subwf**

Sintaxis: subwf

Operandos: 0 :

d e

(f)

: C,

Código de OP : C

Descripción: Re

me

en

Ejemplo 1:

Antes instrucción:

Después instrucción:

Ejemplo 2:

Antes instrucción:

Después instrucción:

Ejemplo 3:

Antes instrucción:

Después instrucción:

**swapf**

Sintaxis: swapf

Operandos: 0 :

d e

(f &lt;

(f &lt;

Flags afectados: Ni

Código de OP : 0

Descripción: Lo

de

W,

Ejemplo1:

Antes instrucción:

Después instrucción:

**subwf****Resta el registro f a W**Sintaxis: **subwf f,d**Operando:  $0 \leq f \leq 127$  $d \in [0,1]$ Operación:  $(f) - (W) \rightarrow (\text{destino})$ 

(Subtract W from f)

Flags afectados: C, DC, Z

Código de OP : | 00 | 0010 | dfff | ffff |

Descripción: Resta por el método de complemento a 2 el contenido del registro 'f' menos el contenido del registro W. Almacena el resultado en W si d=0 y en 'f' si d=1.

Ejemplo 1: **subwf Reg1,I ;(Reg1)-(W) → (Reg1)**Antes instrucción:  $(\text{Reg1}) = 0x03, (W) = 0x02, C = ? \text{ y } Z = ?$ Después instrucción:  $(\text{Reg1}) = 0x01, (W) = 0x02, C = 1 \text{ (positivo) y } Z = 0.$ Ejemplo 2: **subwf Reg1,I ;(Reg1)-(W) → (Reg1)**Antes instrucción:  $(\text{Reg1}) = 0x02, (W) = 0x02, C = ? \text{ y } Z = ?$ Después instrucción:  $(\text{Reg1}) = 0x00, (W) = 0x02, C = 1 \text{ y } Z = 1 \text{ (resultado c.m.).}$ Ejemplo 3: **subwf Reg1,I ;(Reg1)-(W) → (Reg1)**Antes instrucción:  $(\text{Reg1}) = 0x01, (+1 decimal), (W) = 0x02, C = ? \text{ y } Z = ?$ Después instrucción:  $(\text{Reg1}) = 0xFF, (-1 decimal), (W) = 0x022, C = 0 \text{ (neg.) y } Z = 0.$ **swapf****Intercambia nibbles de f**Sintaxis: **swapf f,d**Operando:  $0 \leq f \leq 127$  $d \in [0,1]$ Operación:  $(f<3:0>) \rightarrow (d<7:4>)$ 

(Swap Nibbles in f)

 $(f<7:4>) \rightarrow (d<3:0>)$ 

Flags afectados: Ninguno

Código de OP : | 00 | 1110 | dfff | ffff |

Descripción: Los cuatro bits de más peso del registro 'f' se intercambian con los 4 bits de menos peso del mismo registro. Si d=0 el resultado se almacena en W, si d=1 el resultado se almacena en 'f'.

Ejemplo1: **swapf Reg1,0**Antes instrucción:  $(\text{Reg1}) = 0xA5 \text{ y } (W) = ?$ Después instrucción:  $(\text{Reg1}) = 0xA5 \text{ y } (W) = 0x5A.$ y  $Z = 0.$

**xorlw****OR-Exclusiva del literal k con W****Sintaxis:** xorlw k**Operandos:**  $0 \leq k \leq 355$ **Operación:** (W) XOR k  $\rightarrow$  (W) *(Exclusive OR Literal with W)***Flags afectados:** Z**Código de OP:** 11 1010 kkkk kkkk**Descripción:** Realiza la función OR-Exclusiva entre el contenido del registro W y la constante 'k' dc 8 bits. El resultado se almacena en W.**Ejemplo:** xorlw b'10101111'; (W) XOR b'10101111'  $\rightarrow$  (W)**Antes instrucción:** (W) = b'10110101' y Z = ?**Después instrucción:** (W) = b'00011010' y Z = 0**xorwf****OR-Exclusiva de W con el registro f****Sintaxis:** xorwf f,d**Operandos:**  $0 \leq f \leq 127$  $d \in [0,1]$ **Operación:** (W) XOR (f)  $\rightarrow$  (destino) *(Exclusive OR W with f)***Flags afectados:** Z**Código de OP:** 11 0110 dfff ffff**Descripción:** Realiza la función OR-Exclusiva entre el contenido del registro W y el contenido del registro 'f'. Almacena el resultado en 'f' si d=1 y en W si d=0.**Ejemplo:** xorwf Reg,I ; (W) XOR (Reg)  $\rightarrow$  (Reg)**Antes instrucción:** (Reg) = b'10101111', (W) = b'10110101' y Z = ?**Después instrucción:** (Reg) = b'00011010', (W) = b'10110101' y Z = 0.

al k con W*Literal with W)*

registro W y la

**APÉNDICE C**

I' → (W)

**CONSTANTES Y OPERADORES**l registro f

| TIPO                             | SINTAXIS                                                                     | EJEMPLO                                                            |
|----------------------------------|------------------------------------------------------------------------------|--------------------------------------------------------------------|
| Decimal                          | D'<cantidad>'<br>d'<cantidad>'<br>. <cantidad>                               | movlw D'109'<br>movlw d'109'<br>movlw .109                         |
| Hexadecimal                      | H'<cantidad>'<br>h'<cantidad>'<br>0x<cantidad><br><cantidad>H<br><cantidad>h | movlw H'6D'<br>movlw h'6D'<br>movlw 0x6D<br>movlw 6DH<br>movlw 6Dh |
| Octal                            | O'<cantidad>'<br>o'<cantidad>'                                               | movlw O'155'<br>movlw o'155'                                       |
| Binario                          | B'<cantidad>'<br>b'<cantidad>'                                               | movlw B'01101101'<br>movlw b'01101101'                             |
| ASCII                            | A'<carácter>'<br>a'<carácter>'<br>'<carácter>'                               | movlw A'M'<br>movlw a'M'<br>movlw 'M'                              |
| "String" o Cadena de Caracteres. | "<string>"                                                                   | DT "Estudia DPE"                                                   |

Tabla C-1 Formato de las constantes

La tabla C-1 representa la forma de especificar el sistema de numeración o códigos alfanuméricos, para el ensamblador MPASM, con un ejemplo por caso. Hay que tener en cuenta:

- Las constantes hexadecimales que comiencen por una letra (A-F) deben ir precedidas de un cero para que no sean confundidas con una etiqueta. Ejemplo: `movlw 0FAh`.
- Las constantes pueden ser opcionalmente precedidas por un signo "+" (valores positivos) o "-" (valores negativos). Si no se antepone nada se asume que el valor es positivo.
- **Cuando los operandos son caracteres ASCII**, deben estar encerrados entre apóstrofes o comillas simples. Ejemplo: `movlw 'G'`.

Para facilitar la tarea de programación el ensamblador MPASM permite utilizar múltiples operadores aritméticos detallados en la ayuda del MPLAB y en la guía "MPASM. USER'S GUIDE" que se puede obtener gratuitamente en la página Web del fabricante [www.microchip.com](http://www.microchip.com). En la tabla C-2 se detallan los principales:

| OPERADOR | EJEMPLO                        |
|----------|--------------------------------|
| \$       | Retorna el valor del PC        |
| (        | Paréntesis izquierdo           |
| )        | Paréntesis derecho             |
| !        | Operador NOT (completo lógico) |
| -        | Negación (complemento a 2)     |
| ~        | Complemento                    |
| high     | Retorna byte alto              |
| low      | Retorna byte bajo              |
| *        | Multiplica                     |
| /        | Divide                         |
| %        | Módulo                         |
| +        | Suma                           |
| -        | Resta                          |
| >=       | Mayor o igual                  |
| >        | Mayor que                      |
| <        | Menor que                      |
| <=       | Menor o igual                  |
| ==       | Igual a                        |
| !=       | No igual a                     |
| =        | Hacer igual a                  |

Tabla C-2 Principales operadores aritméticos del ensamblador MPASM

3 codigos  
2 tener e si

deben ir  
Ejemplo:

(valores  
ie que el  
los entre

te utilizar  
la guía  
Web del

## APÉNDICE D

# PRINCIPALES DIRECTIVAS DEL ENSAMBLADOR MPASM

Las directivas del ensamblador no son instrucciones del microcontrolador, sino que son unas herramientas del programa ensamblador, que hacen que al programador le resulte más sencilla la programación.

En este apéndice presentamos en la tabla D-1 todas las directivas que tiene el ensamblador MPASM. Todas estas directivas se explican en detalle en el manual "*MPASM. USER'S GUIDE*" que se puede bajar de la web del fabricante [www.microchip.com](http://www.microchip.com) y en la ayuda del MPASM dentro del entorno MPLAB.

| DIRECTIVA | DESCRIPCIÓN                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------|
| BADRAM    | Especifica las posiciones del RAM inválidas                                                        |
| BANKSEL   | Genera el código que selecciona el banco de memoria de memoria RAM para direccionamiento indirecto |
| BANKSEL_  | Genera el código que selecciona el código de memoria RAM                                           |
| CBLOCK    | Define un bloque de constantes                                                                     |
| CODE      | Empieza la sección del código ejecutable                                                           |
| _CONFIG   | Especifica los bits de configuración                                                               |
| CONSTANT  | Declara los símbolos de las constantes                                                             |
| DATA      | Crea datos numéricos y de texto                                                                    |
| DB        | Declara datos de un byte                                                                           |
| DE        | Define los datos de EEPROM                                                                         |
| #DEFINE   | Define una etiqueta de sustitución de texto                                                        |
| DT        | Define tabla                                                                                       |
| DW        | Declara datos de un word                                                                           |
| ELSE      | Empieza el bloque alternativo de un IF                                                             |

|            |                                                           |
|------------|-----------------------------------------------------------|
| END        | Fin de bloque de programa                                 |
| ENDC       | Acaba un bloque constante                                 |
| ENDIF      | Fin del bloque de instrucciones condicionales             |
| ENDM       | Fin de la definición de una Macro                         |
| ENDW       | Fin de un bucle de While                                  |
| EQU        | Define una constante para el ensamblador                  |
| ERROR      | Manda un mensaje de error                                 |
| ERRORLEVEL | Sitúa el nivel del error                                  |
| EXITM      | Salida de una Macro                                       |
| EXPAND     | Expande una lista de Macro                                |
| EXTERN     | Declara una etiqueta externa                              |
| FILL       | Llena la memoria                                          |
| GLOBAL     | Exporta una etiqueta definida                             |
| IDATA      | Comienza una sección de identificadores (ID)              |
| IDLOCS     | Especifica dónde están colocados los Identificadores (ID) |
| IF         | Empieza un bloque de código condicional                   |
| IFDEF      | Ejecutar si el símbolo ha sido definido                   |
| IFNDEF     | Ejecutar si el símbolo no ha sido definido                |
| INCLUDE    | Incluye ficheros fuente adicionales                       |
| LIST       | Opciones listado                                          |
| LOCAL      | Declara una Macro variable como local                     |
| MACRO      | Declara la definición del Macro                           |
| MAXRAM     | Especifica la dirección del RAM máxima                    |
| MESSG      | Crea mensajes definidos por el usuario                    |
| NOEXPAND   | Termina la expansión del Macro                            |
| NOLIST     | Termina el listado                                        |
| ORG        | Pone el origen del programa                               |
| PAGE       | Inserta el número de página del listado                   |
| PAGESEL    | Genera el código de selección de la página de ROM         |
| PROCESSOR  | El tipo del procesador utilizado                          |
| RADIX      | Especifica sistema de numeración predefinido              |
| RES        | Reserva memoria                                           |
| SET        | Define variable de ensamblador                            |
| SPACE      | Inserta líneas en blanco                                  |
| SUBTITLE   | Especifica el subtítulo del programa                      |
| TITLE      | Especifica el título del programa                         |
| UDATA      | Empieza la sección de datos no inicializados              |
| UDATA OVR  | Empieza la sección de datos no inicializados superpuestos |
| UDATA SHR  | Empieza la sección de datos no inicializados compartidos  |
| #UNDEFINE  | Anula una etiqueta de sustitución                         |
| VARIABLE   | Declara un símbolo como variable                          |
| WHILE      | Realiza el bucle mientras la condición es verdadera       |

Tabla U-1 Directivas del ensamblador MPASM

A continuación se expone un resumen de las principales.

CBLO

Sintaxis:

Descripción:

Estas

&lt;label&gt; s

propósito

finaliza cu

&lt;ex

Si no se a

variable fi

ningún &lt;e

Si &lt;

de &lt;incre

Pue

Ejemplo:

CBL

nom

nomp

END

En

es asignar

lista qued

utilización

CL

C

D

U

EI

**CBLOCK****Define un bloque de constantes**

**Sintaxis:** C'RLOCK [<expr>]  
 <label>[:<increment>][,<label>[:<increment>]]  
 ENDC

**Descripción:** *(Define a Block of Constants)*

Esta directiva se explica en el **capítulo 9**. Define una lista de constantes. A cada <label> se le asigna un valor inmediatamente superior que a la anterior <label>. El propósito de esta directiva es asignar direcciones a muchas etiquetas. La lista de etiquetas finaliza cuando se encuentra la directiva ENDC.

<expr> indica el valor de arranque para el primer nombre del bloque de etiquetas. Si no se asigna en la expresión, la primera variable recibirá un valor superior al de la variable final del CBLOCK anterior. Si el primer CBLOCK en el archivo fuente no tiene ningún <expr> los valores asignados empiezan con el cero.

Si <increment> se especifica, entonces a la próxima <label> se le asigna el valor de <increment> superior a la anterior <label>.

Pueden darse múltiples nombres en una línea, separada por las comas.

**Ejemplo:**

```
CBLOCK 0x20          ; A la primera variable se le asigna el valor 20
nombre_1,nombre_2      ; nombre_2, se le asigna el valor 21.
nombre_3,nombre_4      ; nombre_4 se le asigna el valor 23.
ENDC
```

En la mayoría de las aplicaciones el propósito de las directivas CHLOCK y ENDC es asignar direcciones (generalmente de memoria RAM de datos) a muchas etiquetas. La lista queda encerrada entre las directivas CBLOCK y ENDC. Un ejemplo típico de utilización:

|             |                                                                 |
|-------------|-----------------------------------------------------------------|
| CBLOCK 0x0C | ; Las variables se posicionan a partir de esta posición de RAM. |
| Centenas    | , La variable Centenas ocupa la posición 0x0C de RAM.           |
| Decenas     | , La variable Decenas ocupa la posición 0x0D de RAM.            |
| Unidades    | , La variable Unidades ocupa la posición 0x0E de RAM.           |
| ENDC        |                                                                 |

CONFIG

## **Define la palabra de configuración**

**Sintaxis:** `_CONFIG <expr>`

**Descripción:** *(Set Processor Configuration Bits)*

Directiva para la definición de los bit de la palabra de configuración del microcontrolador con el valor descrito en *<expr>*.

## Ejemplo:

~~\_CONFIG \_CP\_OFF & \_WDT\_OFF & \_PWRTE\_ON & \_XT\_OSC~~

Esta directiva indica la configuración elegida para el proceso de grabación del microcontrolador. En este caso:

- No hay protección de código (`_CP_OFF`).
  - No se habilita el Watchdog (`_WDI_OFF`).
  - Se habilita el reset mediante Power-Up Timer (`_PWRTE_ON`).
  - Se utiliza el oscilador por cristal de cuarzo. ( `XT OSC`).

Es importante resaltar que “\_CONFIG” se inicia con dos subrayados (guiones bajos), no con uno (este error es muy frecuente en los diseñadores noveles).

DE

## Define datos en la EEPROM

Syntaxis: [ <label> ] DE <expr>[ <expr> ... <expr> ]

**Descripción:** *(Declare EEPROM Data Bytes)*

Reserva palabras de memoria de 8 bits en la memoria EEPROM de datos. Cada expresión reserva un valor de 8 bits. Cada carácter de una expresión se guarda en una posición separada.

Ejemplo:

**ORG** 0x2100 ;Corresponde a la dirección 0 de la zona EEPROM  
;de datos.  
**DE** "Programa EEPROM 04, Versión 2.5, 15-08-2003" .0x00

coincid  
en la fig

EE POLY

#DEM

## Sintaxis

E  
Europe

E  
método

### Ejemplos

#DEFINE

## Ejempl

Como en este caso el origen de la directiva está en la dirección 2100h, que coincide con el inicio de la EEPROM de usuario, se obtiene el resultado que se muestra en la figura D-1, que corresponde con la ventana de la memoria EEPROM de usuario.

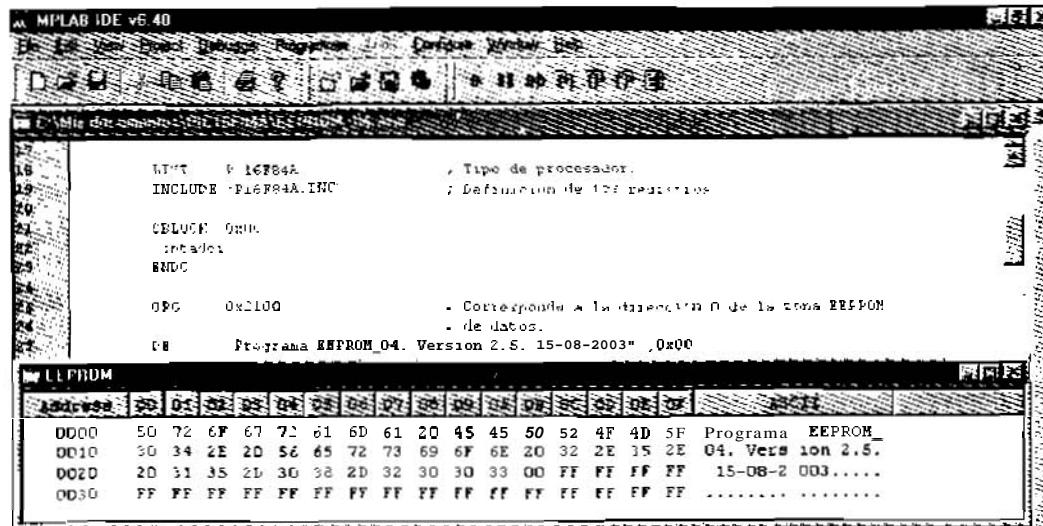


Figura D-1 Memoria EEPROM de usuario después de la directiva DE

## #DEFINE

Define una etiqueta de sustitución de texto

Sintaxis: #DEFINE <name> [<string>]

Descripción: (Define a Text Substitution Label)

Esta directiva define una cadena de sustitución de texto. Dondequiero que <name> se encuentre en el ensamblador se sustituirá por <string>.

Esta directiva emula el ANSI C<sup>1</sup> estándar como `#define`. Definir símbolos con este método no está disponible para ser usado por el MPLAB.

### Ejemplo 1:

```
#DEFINE LED PORTB,4 ; El LED se conecta en esta linea.  
...  
bsf LED ; Enciende el LED.
```

### Ejemplo 2:

```
#DEFINE Banco0 bcf STATUS,RP0 ; Acceso al Banco 0.  
#DEFINE Banco1 bsf STATUS,RP0 ; Acceso al Banco 1.
```

**Inicio**

```

Banco0
bcf    LED      ; Configura esta linea como salida
Banco1
bsf    LED      ; Enciende el diodo LED

```

**DT****Define tabla**

**Sintaxis:** [] DT <expr> [, <expr>, ...., <expr>]

**Descripción:**

(*Define Table*)

Esta directiva genera durante la fase de ensamblado tantas instrucciones *retlw* como caracteres o octetos la acompañen. Se explica en capítulo 11.

**Ejemplo:**

```
DT "mensaje".0x10,.15
```

Esta directiva genera los códigos de operación de las siguientes instrucciones:

```

rctlw 0x6D ; ('m' en ASCII)
retlw 0x65 ; ('e' en ASCII)
retlw 0x6E ; ('n' en ASCII)
rctlw 0x73 ; ('s' en ASCII)
retlw 0x61 ; ('a' en ASCII)
retlw 0x6A ; ('j' en ASCII)
retlw 0x65 ; ('e' en ASCII)
retlw 0x10
retlw 0x0F ; (15 en decimal)

```

**ELSE****Empieza bloque alternativo de un IF**

**Sintaxis:** ELSE

**Descripción:**

(*Begin Alternative Assembly Block to IF*)

Usada junto con la directiva IF para proporcionar un camino alternativo de ensamblado si al evaluar la condición es falsa. ELSE puede usarse dentro de un bloque de programa o en una macro.

**Ejemplo:** Ver directiva IF.

**END****Fin de bloque de programa****Sintaxis:** END**Descripción:** *(End Program Block)*

Esta directiva indica el final del programa y es obligatoria. Si se detecta el fin de fichero y no se ha encontrado la directiva END se produce error. Todas las líneas posteriores a la línea en la que se encuentra esta directiva se ignoran y no se ensamblan.

**Ejemplo:**

|        |     |            |                                   |
|--------|-----|------------|-----------------------------------|
| Inicio | bsf | STATUS,RP0 | ; Comienza el programa ejecutable |
|        | ... | ...        |                                   |
|        | END |            | ; Fin del programa                |

**ENDC****Fin de un bloque de constantes****Sintaxis:** ENDC**Descripción:** *(End Constant Block)*

ENDC se escribe al final de una lista de constantes CBLOCK. Debe escribirse para indicar el fin de la lista.

**Ejemplo:** Mirar la directiva CBLOCK.**ENDIF****Fin de un bloque de ensamblado condicional****Sintaxis:** ENDIF**Descripción:** *(End Conditional Assembly Block)*

Esta directiva marca el extremo de un bloque condicional de ensamblado. ENDIF puede usarse dentro de un bloque de programa o en una macro.

**Ejemplo:** Mirar la directiva IF.

**ENDM****Fin de la definición de una macro**

**Sintaxis:** ENDM

**Descripción:** *(End a Macro Definition)*

Termina una definición del macro comenzada con la directiva MACRO. Las macros se explican en el capítulo 16.

**Ejemplo:** Mirar la directiva MACKO.

**EQU****Define una constante para el ensamblador**

**Sintaxis:** <label> EQU <expr>

**Descripción:** *(Define an Assembler Constant)*

Esta directiva permite asignar el valor de <expr> a un identificador <label>. Su valor puede ser el resultado de una expresión compuesta por otros identificadores tan compleja como se desee.

Generalmente, el identificador es un nombre que describe el valor de manera más significativa para el programador. Suele utilizarse para definir constantes y direcciones de memoria. Así, es más fácil recordar "ValorCarga" que recordar el valor 147 o, en el caso de una dirección de memoria, PORTA que 0x05.

**Ejemplo:**

ValorCarga EQU d'147 ; Asigna el valor numérico de 147 a la etiqueta "ValorCarga".

**ERROR****Emite un mensaje de error**

**Sintaxis:** ERROR "<text\_string>"

**Descripción:** *(Issue an Error Message)*

Esta directiva genera un mensaje de error idéntico a cualquier error del ensamblador MPASM. Si el proceso de ensamblado ejecuta esta directiva aparece la clásica pantalla de error. El texto del mensaje debe ir entrecomillado y puede tener hasta 80 caracteres.

**Ejemplo:**

**IF**

**Sintaxis:**

**Descripción:**

Co  
verdadera  
siguientes

La  
de la lógi  
cero se  
expresión  
falsa (*false*)

**Ejemplo:**

**Ejemplo:**

En  
memoria

**macro****Ejemplo:**

```
ChequeoError      MACRO Argumento1
    IF Argumento1 >= $5          ; Si el argumento está fuera de rango
        ERROR "error_checking-01 el argumento está fuera de rango"
    ENDIF
ENDM
```

Otro ejemplo se **describe** en la directiva IF.

**IF****Comienza un bloque de código condicional**

**Sintaxis:** IF <expr>  
 ....  
 ENDIF

**Descripción:** (*Begin Conditionally Assembled Code Block*)

Comienzo de ejecución de un bloque condicional de ensamblado. Si <expr> es verdadera el código inmediato al IF se ensamblara. En caso contrario, las instrucciones siguientes se saltan hasta encontrar una directiva ELSE o una directiva ENDIF.

La evaluación de una expresión que sea cero se considera desde el punto de vista de la lógica falsa. La evaluación de una expresión que resulte cualquier valor distinto de cero se considera como verdadera. La directiva IF opera con el valor lógico de una expresión: una expresión verdadera (*true*) garantiza devolver un valor distinto de cero, y falsa (*false*) el valor cero.

**Ejemplo 1:**

```
JF      Version == 100           ; Comprueba la versión actual
      movlw  0x0A
      movwf  io_1
ELSE
      rmovlw 0x01
      movwf  io_1
ENDIF
```

**Ejemplo 2:**

En el siguiente ejemplo si la etiqueta “FinTabla” se localiza en una dirección de memoria de programa mayor de 0xFF el ensamblador emitirá un mensaje de error.

```
.....
FinTabla
IF (FinTabla > 0xFF)
    ERROR "¡CUIDADO!: La tabla ha superado el tamaño de la página de los"
    MESSG "primeros 256 bytes de memoria ROM. NO funcionará correctamente."
ENDIF
```

**INCLUDE****Incluye ficheros fuentes adicionales****MACR**

**Sintaxis:** INCLUDE <>include\_file>> o  
INCLUDE “<include\_file>”

**Sintaxis:**

**Descripción:**

(*Include Additional Source File*)

El archivo especificado se lee como código fuente. El efecto es igual que si el texto entero del archivo *include* se pusiera en el lugar donde se ha escrito la directiva INCLUDE. Se permiten seis niveles de anidamiento.

El <include\_file> puede escribirse entre comillas (“ ”) o entre los símbolos de "mayor que" y "menor que" (< >). Si se especifica totalmente el camino del fichero include solo se buscara en ese camino. Si no se indica camino el orden de la búsqueda es: el directorio activo actual, en segundo lugar el directorio del archivo fuente y por último en el directorio ejecutable de MPASM. Se explica en el capítulo 10.

**Ejemplo:**

**Ejemplo:**

|                              |                                                 |
|------------------------------|-------------------------------------------------|
| INCLUDE <P16F84A.INC>        | ; Define el archivo donde están definidos todos |
|                              | ; los registros del PIC16F84A.                  |
| INCLUDE “P16F84A.INC”        | ; También se puede definir de esta forma.       |
| INCLUDE “c:\sys\sysdefs.inc” | ; Define “sysdefs” con su trayectoria.          |
| INCLUDE <regs.h>             | ; Define “regs.h” sin trayectoria.              |

**LIST****Opciones de listado**

Donde se cumplirá con lo establecido en el apartado anterior.

**Sintaxis:** LIST [<list\_option>, ..., [<list\_option>]]

Cada uno de los argumentos que se inserten tienen que ser una opción de lista.

**Descripción:**

(*Listing Options*)

La directiva LIST tiene efecto sobre al proceso de ensamblado y sobre el formato del fichero listable de salida según una larga lista de opciones que se detallan en el manual "MPASM. USER'S GUIDE" y en la ayuda del MPASM. De todas la más importante es la que indica el procesador utilizado según detalla el siguiente ejemplo.

Si se cumple con lo establecido en el apartado anterior.

**Ejemplo:**

LIST P=P16F84A ; El PIC16F84A como procesador utilizado.

Este comando finalizará el listado.

Si se cumple con lo establecido en el apartado anterior.

Este comando finalizará el listado.

Este comando finalizará el listado.

adicionales

al Source File)

al que si el texto  
rito la directiva

os simbolos de  
mimo del fichero  
e la búsqueda es:  
nte y por ultimo

os todos  
na.

s de listado

string Options)  
sobre el formato  
e detallan en el  
e todas las mas  
e ejeniplo.

ado.

## MACRO

### Declara la definición de Macro

Sintaxis: <label> MACRO [<arg>, ..., <arg>]

**Descripción:** *(Declare Macro Definition)*

Una macro define un conjunto de instrucciones a las que se les asigna un nombre. Posteriormente, el programa fuente del usuario puede incluir el nombre de una macro y todas las instrucciones que la componen quedan insertadas en el momento de realizar el ensamblado formando parte del programa. Se explica en detalle en el capítulo 16.

**Ejemplo:**

Una estructura macro puede ser la siguiente:

|             |               |
|-------------|---------------|
| NombreMacro | MACRO         |
|             | Instrucción_1 |
|             | Instrucción_2 |
|             | .....         |
|             | .....         |
|             | Instrucción_n |
|             | ENDM          |

Donde “NombreMacro” indica el nombre de la macro que posteriormente se empleará en el programa fuente para incluir todas las instrucciones que estén definidas bajo este nombre. Las directivas MACRO y ENDM forman el cuerpo dentro del cual están incluidas todas las instrucciones deseadas.

Cada vez que se emplea “NombreMacro” en el programa fuente, implica que se inserten todas las instrucciones que conlleve dicho nombre en el programa.

## MESSG

### Crea mensajes definidos por el usuario

Sintaxis: MESSG "<message\_text>"

**Descripción:** *(Create User Defined Message)*

Esta directiva permite crear mensajes definidos por el usuario que aparecen al finalizar el proceso de ensamblado y en el fichero listable \*.list. Puede tener hasta 80 caracteres.

**Ejemplo:**

```
MensajeMacro      MACRO
MESSG "mssg_macro-001 llamada sin el argumento"
ENDM
```

Hay otros ejemplos descritos en la directiva IF.

**ORG****Origen de las instrucciones del programa**

Sintaxis:      [**<label>**] ORG <expr>

**Descripción:**

*(Set Program Origin)*

El **origen** del programa 'comienza' la dirección **indicada** en la <expr> de la directiva. Si se especifica una <label> se le da el valor de la <expr>. Si no hay ningún ORG especificado la generación del código comienza en la dirección cero.

**Ejemplo 1:**

|      |                     |                                      |
|------|---------------------|--------------------------------------|
| ORG  | 0                   | ; Dirección de comienzo del programa |
| goto | Inicio              |                                      |
| ORG  | 4                   | ; Vector de interrupción.            |
| goto | SevicioInterrupcion |                                      |

**Ejemplo 2:**

|       |     |            |                      |
|-------|-----|------------|----------------------|
| Int_1 | ORG | 0x20       | ; Vector SO va aquí. |
| Int_2 | ORG | Int_1+0x10 | ; Vector 30 va aquí. |

rograma

am Origin)

<expr> de la  
o hay ningún

## APÉNDICE E

## REGISTROS ESPECIALES

| BANCO 0 |                                                     | BANCO 1 |     | SFR (5 especiales)<br>GPR |
|---------|-----------------------------------------------------|---------|-----|---------------------------|
|         | INDF                                                | INDF    | 80h |                           |
|         | TMR0                                                | OPTION  | 81h |                           |
|         | PCL                                                 | PCL     | 82h |                           |
|         | STATUS                                              | STATUS  | 83h |                           |
| 04h     | FSR                                                 | FSR     | 84h |                           |
|         | PORATA                                              | TRISA   | 85h |                           |
| 06h     | PORTB                                               | TRISB   | 86h |                           |
| 07h     | —                                                   | ---     | B7h |                           |
| 08h     | EEDATA                                              | EECON1  | 88h |                           |
| 09h     | EEADRH                                              | EECON2  | 89h |                           |
| 0Ah     | PCLATH                                              | PCLATH  | 8Ah |                           |
| 0Bh     | INTCON                                              | INTCON  | 8Bh |                           |
| 0Ch     | 68 REGISTROS DE PROPOSITO GENERAL                   |         | 8Ch |                           |
| 4Fh     | MAPEADO EN BANCO 0                                  |         | CFh |                           |
| 50h     | NO IMPLEMENTADO FISICAMENTE.<br>(SE LEEN COMO "0"). |         | D0h |                           |
| 7Fh     |                                                     |         | FFh |                           |

Figura E-1 Memoria RAM de datos del PIC16F84

| Dirección | Nombre | bit 7                                                                          | bit 6  | bit 5 | bit 4                                            | bit 3 | bit 2 | bit 1 | bit 0   |  |  |  |  |  |
|-----------|--------|--------------------------------------------------------------------------------|--------|-------|--------------------------------------------------|-------|-------|-------|---------|--|--|--|--|--|
| BANCO 0   |        |                                                                                |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 00h       | INDF   | Registro utilizado en el direccionamiento indirecto (no es un registro físico) |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 01h       | TMR0   | Timer / Contador de 8 bit                                                      |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 02h       | PCL    | Registro con los 8 bits más bajos del Contador de Programa                     |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 03h       | STATUS | IRP                                                                            | RP1    | RP0   | /TO                                              | /PD   | Z     | DC    | C       |  |  |  |  |  |
| 04h       | FSR    | Registro utilizado como puntero en el direccionamiento indirecto               |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 05h       | PORTA  | --                                                                             | --     | --    | RA4/T0CK1                                        | RA3   | RA2   | RA1   | RA0     |  |  |  |  |  |
| 06h       | PORTB  | RB7                                                                            | RB6    | RB5   | RB4                                              | RB3   | RB2   | RB1   | RB0/INT |  |  |  |  |  |
| 07h       |        | Posición no implementada, se lee como 0                                        |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 08h       | EEDATA | Registro de datos EEPROM                                                       |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 09h       | EEADRR | Registro de direcciones EEPROM                                                 |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 0Ah       | PCLATH | --                                                                             | --     | --    | Buffer escrito con los 5 bit más altos del PC    |       |       |       |         |  |  |  |  |  |
| 0Bh       | INTCON | GIE                                                                            | EEIE   | TOIE  | INTE                                             | RBIE  | TOIF  | INTF  | RBIF    |  |  |  |  |  |
| BANCO 1   |        |                                                                                |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 80h       | INDF   | Registro utilizado en el direccionamiento indirecto (no es un registro físico) |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 81h       | OPTION | /RBU                                                                           | INTEDG | TOSC  | TOSE                                             | PSA   | PS2   | PS1   | PS0     |  |  |  |  |  |
| 82h       | PCL    | Registro con los 8 bit más bajos del Contador de Programa                      |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 83h       | STATUS | IRP                                                                            | RP1    | RP0   | /TO                                              | /PD   | Z     | DC    | C       |  |  |  |  |  |
| 84h       | FSR    | Registro utilizado como puntero en el direccionamiento indirecto               |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 85h       | TRISA  | --                                                                             | --     | --    | Reg. de configuración de las líneas del Puerto A |       |       |       |         |  |  |  |  |  |
| 86h       | TRISB  | Registro de configuración de las líneas del Puerto B                           |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 87h       |        | Posición no implementada, se lee como 0                                        |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 88h       | EECON1 | --                                                                             | --     | --    | EEIF                                             | WRERR | WREN  | WR    | RD      |  |  |  |  |  |
| 89h       | EECON2 | Reg. de control para grabar en la EEPROM de datos (no es un registro físico)   |        |       |                                                  |       |       |       |         |  |  |  |  |  |
| 8Ah       | PCLATH | --                                                                             | --     | --    | Buffer escrito con los 5 bit más altos del PC    |       |       |       |         |  |  |  |  |  |
| 8Bh       | INTCON | GIE                                                                            | EEIE   | TOIE  | INTE                                             | RBIE  | TOIF  | INTF  | RBIF    |  |  |  |  |  |

Tabla E-1 Registros del SFR. (Special Function Registers)

|                        |       |         |
|------------------------|-------|---------|
| 2                      | bit 1 | bit 0   |
| <hr/>                  |       |         |
| un registro físico)    |       |         |
| <hr/>                  |       |         |
| a                      |       |         |
|                        | DC    | C       |
| directo                |       |         |
| 2                      | RA1   | RA0     |
| 2                      | RB1   | RB0/INT |
| <hr/>                  |       |         |
| <hr/>                  |       |         |
| s altos del PC         |       |         |
| F                      | INTF  | RBIFF   |
| <hr/>                  |       |         |
| un registro físico)    |       |         |
| 2                      | PS1   | PS0     |
| a                      |       |         |
| .                      | DC    | C       |
| directo                |       |         |
| s líneas del Puerto A  |       |         |
| <hr/>                  |       |         |
| EN                     | WR    | RD      |
| es un registro físico) |       |         |
| s altos del PC         |       |         |
| IF                     | INTF  | RBIFF   |
| ters)                  |       |         |

## INDF

00h Banco 0, 80h Banco 1

Registro para el direccionamiento indirecto de datos. Éste no es un registro disponible físicamente. Utiliza el contenido del FSR para seleccionar indirectamente la memoria de datos o RAM del usuario; la instrucción determinará lo que se debe hacer con el registro señalado. El direccionamiento indirecto se explica en el capítulo 16.

## TMR0

01h Banco 0

(Timer 0). Temporizador/contador de 8 bits. Se incrementará con una señal externa aplicada al pin RA4/TOCKI o de acuerdo a una señal interna proveniente del reloj del microcontrolador. Se explica ampliamente en capítulos 15 y 18. Al conectar la alimentación su contenido es desconocido, (TMR0) = b'xxxxxxxx'.

## PCL

02h Banco 0, 82h Banco 1

(Program Counter Low byte). Byte bajo del contador de programa, figura E-3. El PIC16F84 dispone de un contador de programa de 13 bits. Sus bits de menor peso corresponden a los 8 bits del registro PCL, implementado en la posición de memoria RAM 02h (y duplicado en la posición 82h del Banco 1) por lo que pueden ser leídos o escritos directamente. Los cinco bits de mayor peso del PC corresponden con los del registro PC11 que no pueden ser leídos ni escritos directamente (figura E-2). Al conectar la alimentación se inicializa a (PCL) = b'00000000' y (PC11) = b'00000'.

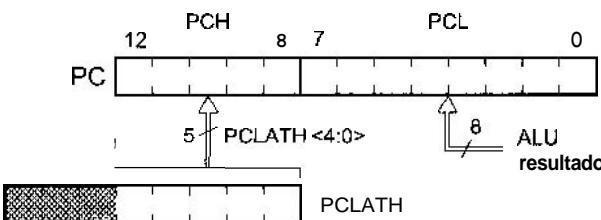


Figura E2 Composición del PC en instrucciones con PCL como destino

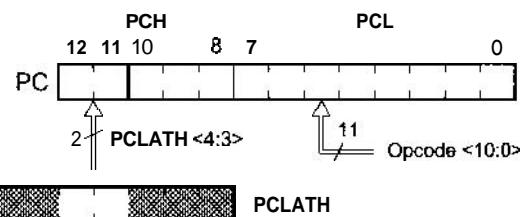


Figura E3 Composición del PC en instrucciones "call" y "goto"

## STATUS

## 03h Banco 0, 83h Banco 1

El registro de estado o *STATUS* indica el estado de la Última operación aritmética o lógica realizada, la causa de reset y los bits de selección de banco para la memoria de datos. A los bits del registro de estado se les suele denominar flags o bandera. Al conectar la alimentación su contenido es (*STATUS*) = b'00011xxx'.

| IRP   | RP1   | RP0   | /TO   | /PD   | Z     | DC    | C     |
|-------|-------|-------|-------|-------|-------|-------|-------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Tabla E-2 Registro de estado o *STATUS*

- **C (Carry bit).** Flag de acarreo en el octavo bit. En instrucciones de suma aritméticas se activa cuando se presenta acarreo desde el bit más significativo del resultado, lo que indica que el resultado ha desbordado la capacidad del registro sobre el que trabaja, es decir, el resultado de la operación ha superado el valor 1111111<sub>2</sub>, (255<sub>10</sub>), que es el máximo valor que se puede representar con 8 bits. En el capítulo 8 se explica en detalle su funcionamiento:
  - C = 0. En la suma significa que no ha habido acarreo y en la resta que el resultado ha sido negativo.
  - C = 1. En la suma significa que ha habido acarreo y en la resta que el resultado ha sido positivo.
- **DC (Digit Carry).** Flag dc acarreo en el 4º bit de menos peso. En operaciones aritméticas se activa cuando hay un acarreo entre el bit 3 y 4, es decir, cuando hay acarreo entre los nibbles de menor y de mayor peso.
- **Z (Zero).** Flag de cero. Se activa a "1" cuando el resultado de una operación aritmética o lógica es cero.
  - Z = 0. El resultado de la ultima operación ha sido distinto de cero.
  - Z = 1. El resultado de la última operación ha sido cero.
- **/PD (Power Down).** Flag de bajo consumo. Es un bit de sólo lectura, no puede ser escrito por el usuario. Sirve para detectar el modo de bajo consumo.
  - /PD = 0. Al ejecutar la instrucción *sleep* y entrar en reposo.
  - /PD = 1. Tras conectar la alimentación V<sub>DD</sub> o al ejecutar *clrwdt*.
- **/TO (Timer Out).** Flag indicador de fin temporización del Watchdog. Es un bit de sólo lectura, no puede ser escrito por el usuario. Se activa en "0" cuando el circuito de vigilancia Watclidog finaliza la temporización. Sirve para detectar si una condición de reset fue producida por el *Watchdog Timer*.
  - /TO = 0. Al desbordar el temporizador del Watchdog.
  - /TO = 1. Tras conectar V<sub>DD</sub> (funcionamiento normal) o al ejecutar las instrucciones *clrwdt* o *sleep*.
- **RP0 (Register Bank Select bit).** Selección del banco para el direccionamiento directo. Señala el banco de memoria de datos seleccionado.
  - RP0 = 0. Selecciona el Banco 0.
  - RP1 = 1. Selecciona el Banco 1.

**Banco 1**

aritmética o  
memoria de  
oanderas. Al

|       |
|-------|
| C     |
| Bit 0 |

nes de suma  
nificativo del  
l del registro  
rado el valor  
u con 8 bits.

resta que el

resta que el

operaciones  
ccir, cuando

na operación

ero.

ra, no puede  
o.

it.

g. Es un bit  
"0" cuando cl  
a detectar si

ejecutar las

ccionamiento

- **RP1** (*Register Bank Select bit*). No utilizado en el PIC16F84.
- **IRP1**. No utilizado en el PIC16F84.

**FSR****04h Banco 0, 84h Banco 1**

Selector de registros para direccionamiento indirecto. En asociación con el registro INDF se utiliza para seleccionar indirectamente los otros registros disponibles. El direccionamiento indirecto se explica en capítulo 16. Al conectar la alimentación su contenido es desconocido, (FSR) = b'xxxxxxxx'.

**PORTA****05h Banco 0**

Puerto de entrada/salida de 5 bits (pines RA0 a RA4). El puerto A puede leerse o escribirse como si se tratara de un registro cualquiera. El registro que controla el sentido (entrada o salida) de sus pines se llama TRISA y está localizado en la dirección 85h del Banco 1. Su pin RA4/TOCKI también puede servir de entrada al Timer 0. Al conectar la alimentación queda configurado como entrada.

**PORTB****06h Banco 0**

Puerto de entrada/salida de 8 bits (pines RB0 a RB7). El puerto B puede leerse o escribirse como si se tratara de un registro cualquiera. El registro que controla el sentido (entrada o salida) de sus pines se llama TRISB y está localizado en la dirección 86h del Banco 1. Algunos de sus pines tienen funciones alternas en la generación de interrupciones (ver capítulo 17). Al conectar la alimentación queda configurado como entrada.

**EEDATA****08h Banco 0**

(*EEPROM Data Register*). Contiene los bytes que se van a escribir o que se han leído de la EEPROM de datos. Al conectar la alimentación su contenido es desconocido, (EEDATA) = b'xxxxxxxx'.

**EEADR****09h Banco 0**

(*EEPROM Address Register*). Contiene la dirección de la EEPROM de datos a la que acceder para leer o escribir. Al conectar la alimentación su contenido es desconocido, (EEADR) = b'xxxxxxxx'.

**PCLATH****0Ah Banco 0**

(*PC Latch High*). Registro que permite acceder de forma indirecta a la parte alta del contador de programas en algunas instrucciones, tal como se describe en la figura E-3. Al conectar la alimentación se resetea (PCLATH) = b'---00000'.

**INTCON****0Bh Banca 0. 8Bh Banco 1**

(*Interrupts Control Register*). Registro para el control de las interrupciones. Es el encargado del manejo de las interrupciones. Contiene los 8 bits que se muestran en la tabla E-2, de los cuales unos actúan como flags señaladores del estado de la interrupción y otros como bit de permiso o autorización para que se pueda producir la interrupción. Al conectar la alimentación su contenido es (INTCON) = b'0000000x'.

| GIE   | EEIE  | T0IE  | INTE  | RBIE  | T0IF  | INTF  | RBIF  |
|-------|-------|-------|-------|-------|-------|-------|-------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Tabla E-3 Registro de control de las interrupciones INTCON

- **RBIF (RB port change Interrupt Flag)**. Flag de estado de la interrupción RBI. Indica que se ha producido una interrupción por cambio de estado de cualquiera de las líneas RR4 a RB7.
  - RBIF = 0. Ninguna de las entradas RR7 a RB4 ha cambiado de estado
  - RBIF = 1. Cualquiera de las líneas RB7 a RB4 del Puerto B ha cambiado. (Debe borrarse por software).
- **INTF (External Interrupt Flag bit)**. Flag de estado de la interrupción externa INT. Indica que se ha producido una interrupción a través del pin RB0/INT.
  - INTF = 0. No hay interrupción externa por el pin RB0/INT.
  - INTF = 1. Ha ocurrido una interrupción externa por la línea RB0/INT. (Debe borrarse por software).
- **T0IF (TMRO Overflow Interrupt Flag bit)**. Flag de estado de la interrupción producida por el TMRO. Indica que se ha producido una interrupción por desbordamiento del Timer 0, es decir, que ha pasado de b'1111111' (FFh) a b'00000000' (00h).
  - T0IF = 0. El TMRO no se ha desbordado
  - T0IF = 1. El TMRO se ha desbordado. (Debe borrarse por software).
- **RBIE (RB Port Change Interrupt Enable)**. Habilitación de la interrupción RBI. Flag que autoriza la interrupción por cambio de estado de las líneas RB7:RB4 del Puerto B.
  - RBIE = 0. Interrupción RBI deshabilitada.
  - RBIE = 1. Interrupción RBI habilitada.

- **INTE (External INT Enable bit)**. Habilitación de la interrupción externa INT. Flag que autoriza la interrupción externa a través del pin RB0/INT.
  - INTE = 0. Interrupción INT deshabilitada.
  - INTE = 1. Interrupción INT habilitada.
- **TOIE (TMRO Interrupt Enable bit)**. Habilitación de la interrupción TOI. Flag que autoriza la interrupción por desbordamiento del Timer 0.
  - TOIE = 0. Interrupción TOI deshabilitada.
  - TOIE = 1. Interrupción TOI habilitada.
- **EEIE (EEPROM Write Complete Interrupt Enable)**. Habilitación de la interrupción EEI. Flag que autoriza la interrupción por escritura completada de un byte en la EEPROM de datos del PIC (el flag EEIF se encuentra en el registro EECON1).
  - EEIE = 0. Interrupción EEI deshabilitada.
  - EEIE = 1. Interrupción EEI habilitada.
- **GIE (Global Interrupt Enable)**. Flag de habilitación global del permiso de interrupción. Se borra automáticamente cuando se reconoce una interrupción para evitar que se produzca ninguna otra mientras se está atendiendo a la primera. Al retornar de la interrupción con una instrucción *retfie*, el bit GIE se vuelve a activar poniéndose a "1".
  - GIE = 0. No autoriza interrupción de ningún tipo.
  - GIE = 1. Autoriza cualquier tipo de interrupción. Se pone a "1" automáticamente con la instrucción *retfie*.

## OPTION

81h Banco 1

Registro de configuración múltiple, aunque su misión principal es gobernar el comportamiento del TMRO. Algunos microcontroladores PIC tienen una instrucción denominada también *option*, por ello, el fabricante *Microchip* recomienda darle otro nombre a este registro. Así en el fichero de definición de etiquetas P16F84A.INC se le nombra como OPTION\_REG. Al conectar la alimentación todos sus bits se ponen a "1", (OPTION\_REG) = b'11111111'.

| /RBPU | INTEDG | T0CS  | T0SE  | PSA   | PS2   | PS1   | PS0   |
|-------|--------|-------|-------|-------|-------|-------|-------|
| Bit 7 | Bit 6  | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Tabla E-3 Registro OPTION\_REG

- **PS2:PS0 (Prescaler Rate Select bits)**. Bits para seleccionar los valores del Prescaler o rango con el que actúa el divisor de frecuencia, según la tabla E-5.

| PS2 PS1 PS0 | Divisor del TMR0 | Divisor 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           |

Tabla E-5 Selección del rango del divisor de frecuencia

- **PSA** (*Prescaler Assignment bit*). Asignación del divisor de frecuencia. El Prescaler es compartido entre el TMR0 y el WDT: su asignación es mutuamente excluyente ya que solamente a uno de ellos se puede aplicar el divisor de frecuencia a la vez.
  - PSA = 0. El divisor de frecuencia se asigna al TMRO.
  - PSA = 1. El divisor de frecuencia se asigna al Watchdog.
- **TOSE** (*TMR0 Source Edge Select bit*). Selecciona flanco de la señal al TMRO.
  - TOSE = 0. TMRO se incrementa en cada flanco ascendente de la señal aplicada al pin RA4/T0CKI.
  - TOSE = 1. TMRO se incrementa en cada flanco descendente de la señal aplicada al pin RA4/T0CKI.
- **TOCS** (*TMR0 Clock Source Select bit*). Selecciona la fuente de señal del TMRO.
  - TOCS = 0. Pulso de reloj interno Fosc/4 (TMRO como temporizador).
  - TOCS = 1. Pulso introducido a través del pin RA4/T0CKI (TMRO como contador).
- **INTEDG** (*Interrupt Edge Select bit*). Selector de flanco de la interrupción INT.
  - INTEDG = 0. Interrupción por flanco descendente del pin RB0/INT.
  - INTEDG = 1. Interrupción por flanco ascendente del pin RB0/INT.
- **/RBPU**, (*Resistor Port B Pull-Up Enable bit*). Habilitación de las resistencias de Pull-Up del Puerto B.
  - /RBPU = 0. Habilita las resistencias de Pull-Up del Puerto R.
  - /RBPU = 1. Deshabilita las resistencias de Pull-Up del Puerto B.

## TRISA

85h Banco 1

Registro de configuración de las líneas del Puerto A. Es el registro de control para el Puerto A. Un "0" en el bit correspondiente al pin lo configura como salida, mientras que un "1" lo hace como entrada. Al igual que el Puerto A, solo dispone de 5 bits. Al conectar la alimentación todos sus bits se ponen a "1", (TRISA)= b'---11111'.

TRIS

el Puer  
que un  
"1", (T

EEC

de dato

E  
Bit 7

**TRISB****86h Banco 1**

Registro de configuración de las líneas del Puerto B. Es el registro de control para el Puerto A. Un "0" en el bit correspondiente al pin lo configura como salida, mientras que un "1" lo hace como entrada. Al conectar la alimentación todos sus bits se ponen a "1", (TRISB) = b'11111111'.

**ECONI****88h Banco 1**

(*EEPROM Control Register 1*). Registro para el control de la memoria EEPROM de datos. Al conectar la alimentación su contenido es (ECONI) = b'---0x000'.

|       |       |       | EEIF  | WRERR | WREN  | WR    | RD    |
|-------|-------|-------|-------|-------|-------|-------|-------|
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Tabla E-6 Registro de control de la EEPROM de datos INTCON

- **RD** (*Read Control Bit*). Bit de control de lectura en la EEPROM. Al ponerlo en "1" se inicia la lectura de un byte en la EEPROM de datos. Este bit se limpia (se pone a "0") por hardware automáticamente al finalizar la lectura de la posición EEPROM.
  - RD = 0. No inicia la lectura de la EEPROM o la misma ha terminado.
  - RD = 1. Inicia la lectura de la EEPROM. Se borra por hardware.
- **WR** (*Write Control Bit*). Bit de control de escritura en la EEPROM. Al ponerlo en "1" se inicia una escritura de un byte en la EEPROM de datos. Este bit se limpia (se pone en "0") por hardware automáticamente una vez la escritura de la EEPROM ha terminado.
  - WR = 0. No inicia la escritura de la EEPROM o la misma ha terminado.
  - WR = 1. Inicia la escritura de la EEPROM. Se borra por hardware.
- **WREN** (*EEPROM Write Enable bit*). Permiso de escritura en la EEPROM.
  - WREN = 0. Prohibe la escritura de la EEPROM
  - WREN = 1. Permite la escritura de la EEPROM.
- **WRERR** (*EEPROM Write Error Flag Bit*). Flag de error en la escritura. Se posiciona a "1" cuando la operación de escritura termina prematuramente debido a cualquier condición de reset.
  - WRERR = 0. La operación de escritura se ha completado correctamente.
  - WRERR = 1. La operación de escritura ha terminado prematuramente.
- **EEIF** (*EEPROM Write Operation Interrupt Flag Bit*). Flag de estado de interrupción por finalización de escritura en EEPROM. Señala el final con éxito de la operación de escritura de un byte en la EEPROM.

- EEIF = 0. La operación de escritura de la EEPROM no ha terminado o no comenzó.
- EEIF = 1. La operación de escritura de la EEPROM ha terminado. Debe borrarse por software.
- Bits 5, 6 y 7 (*Unimplemented*). No implementados físicamente. Se leen "0".

## EECON2

89h Banco 1

(*EEPROM Control Register 2*). Este registro no está implementado físicamente, por lo que es imposible leerlo (si se intenta leer, todos sus bits se leen como ceros). Se emplea como 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.

## REGISTRO DE CONFIGURACIÓN

El PIC16F84 dispone de una palabra de configuración dc 14 bits que se escribe durante el proceso de grabación del microcontrolador y que no se puede modificar durante la ejecución de un programa. Dichos bits ocupan la posición reservada de memoria de programa 2007h.

| Bit 13 | ... | Bit 5 | CP    | PWRTE | WDTE  | FOSC1 | FOSC0 |
|--------|-----|-------|-------|-------|-------|-------|-------|
|        |     |       | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |

Tabla E-7 Registro de configuración (Configuration Word)

- **FOSC<1:0>** (*Flag Oscillator Selection*). Selección del tipo de oscilador:
  - FOSC = 00. Oscilador de bajo consumo LP (32 kHz - 200 kHz)
  - FOSC = 01. Oscilador estándar XT ( 100 kHz- 4MHz)
  - FOSC = 10. Oscilador de alta velocidad HS ( 4 MHz - 20 MHz)
  - FOSC = 11. Oscilador de bajo coste RC.
- **WDTE** (*Watchdog Enable*). Bit de habilitación del *Watchdog*.
  - WDTE = 0. Watchdog deshabilitado.
  - WDTE = 1. Watchdog habilitado.
- **PWRTE** (*Power-up Timer Enable*). Activación del temporizador *Power-Up*.
  - PWRTE = 0. Temporizador *Power-Up* deshabilitado.
  - PWRTE = 1. Temporizador *Power-Up* habilitado.
- **CP** (*Code Protection bit*) Bit de protección de código.
  - CP = 0. Toda la memoria de programa está protegida contra lecturas indeseables.
  - CP = 1. La memoria de programa se puede leer. No está protegida.

do o no

o. Debe

"0".

nco 1amente,  
ros). Se  
PROM,  
lo.escribe  
odificar  
vada deSCO  
it 0

jp.

ecciones

## APÉNDICE F

## GRABADOR T20-SE

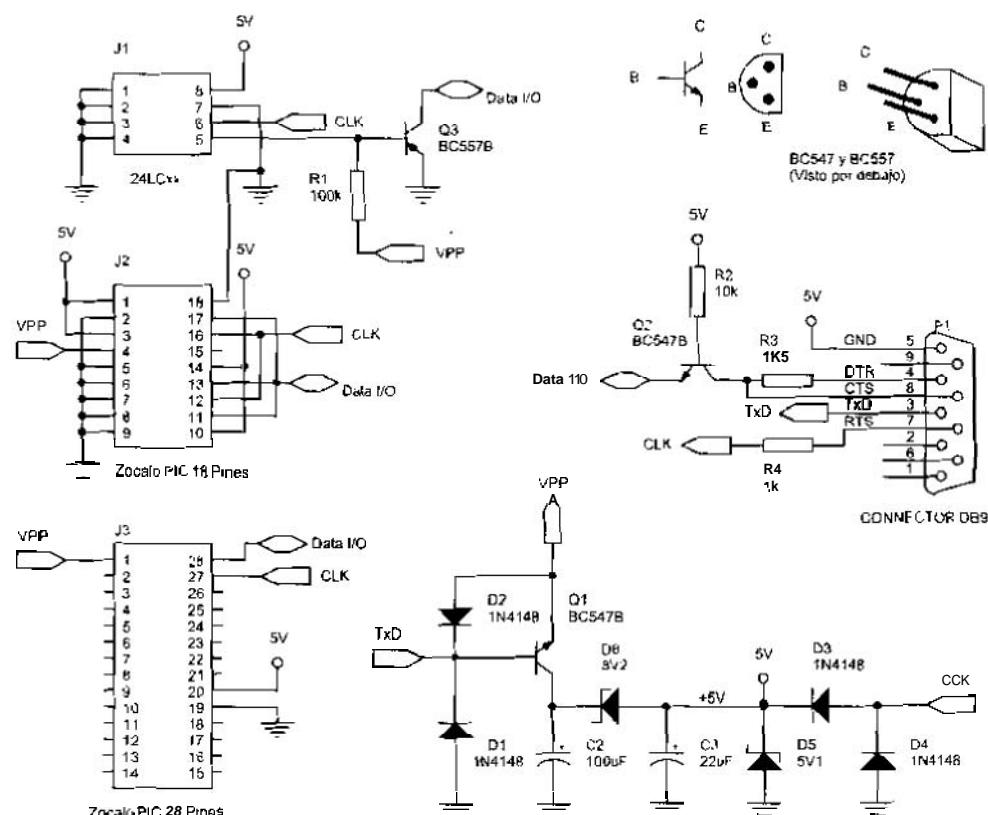
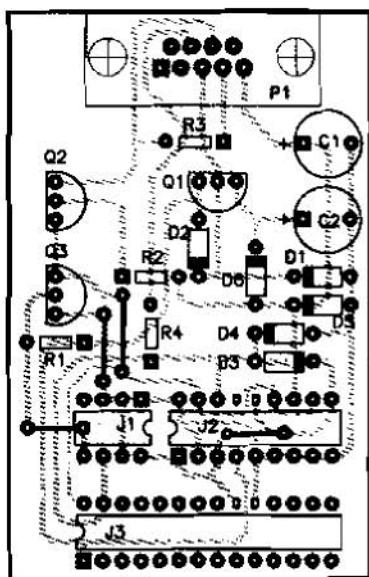
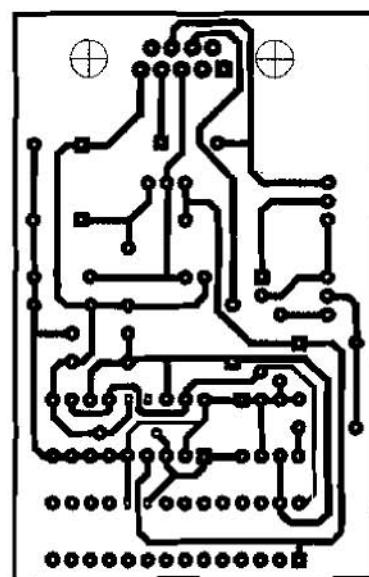


Figura F-1 Esquema eléctrico



Cara de componentes



Cara de pistas

Figura F-2 Placa de Circuito Impreso

**LISTADO DE COMPONENTES:**

|        |                              |
|--------|------------------------------|
| C1     | 22 $\mu$ F / 16V             |
| C2     | 100 $\mu$ F / 16V            |
| D1,D2  |                              |
| D3,D4  | 1N4148                       |
| D5     | Zénerde 5V1 $\frac{1}{2}$ W  |
| D6     | Zéner de 8V2 $\frac{1}{2}$ W |
| J1     | Zócalo 8 pines               |
| J2     | Zócalo 18 pines              |
| J3     | Zócalo 28 pines              |
| P1     | Conector DB9 hembra          |
| Q1, Q2 | BC547                        |
| Q3     | BC557                        |
| R1     | 100 k                        |
| R2     | 10 k                         |
| R3     | 1k5                          |
| R4     | 1 k                          |

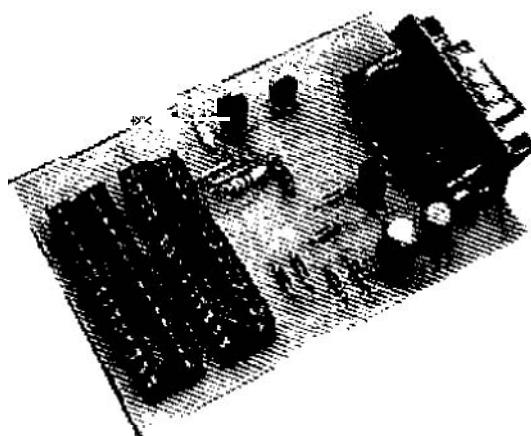


Figura F-3 Grabador TE20-SE

(Código  
es un s  
que uti  
número  
para d  
ordenadvalores  
extendi

**APÉNDICE G****CÓDIGO ASCII**

**ASCII** es el acrónimo de *American Standard Code for Information Interchange* (Código Normalizado Americano para el Intercambio de Información). El código ASCII es un sistema de representación utilizado en los sistemas digitales (incluido ordenadores) que utiliza un esquema de codificación que asigna valores numéricos a las letras, números, signos de puntuación y a otros caracteres. Al normalizar los valores utilizados para dichos caracteres se logra que, en sistemas digitales, microcontroladores, ordenadores y programas informáticos intercambien información.

Dado su origen en el sistema binario, el código ASCII está compuesto por 256 valores que están divididos a la mitad en el conjunto estandarizado y el conjunto extendido de 128 valores cada uno.

- **Código ASCII estándar.** El conjunto ASCII básico, o estándar, utiliza 7 bits para cada código, lo que da como resultado 128 códigos de caracteres desde 0 hasta 127 (00h hasta 7Fh hexadecimal). Son los mismos en todos los sistemas operativos y lenguajes de programación. Normalmente estos códigos se representan con 8 bits, poniendo el octavo bit o **MSB** a cero.
- **Código ASCII extendido.** Son los caracteres del 128 hasta el 255 (80h al FFh hexadecimal) y no hay un acuerdo respecto a ellos. Se asignan a conjuntos de caracteres que varían según los fabricantes y programadores de software. Estos códigos no son intercambiables entre los diferentes programas y sistemas digitales como los caracteres ASCII estándar.

En el conjunto de caracteres ASCII estándar se diferencian dos **grupos**:

- **Códigos de control.** Los primeros 32 valores (del 0 al 31) están asignados a los códigos de control de comunicaciones y de impresora (caracteres no imprimibles,

como retroceso, retorno de carro y tabulación) empleados para controlar la forma en que la información es transferida desde un ordenador a otro o desde un ordenador a una impresora.

- **Códigos alfanuméricos.** Los 96 códigos restantes (del 32 al 127) son caracteres ASCII normales, se asignan a **letras mayúsculas y minúsculas del alfabeto latino**, a los signos de puntuación corrientes, a los dígitos del 0 al 9 y a otros que se detalla en la tabla G-1.

| Códigos de Control     |     |     |      | Carácteres Alfanuméricos |     |         |     |     |      |     |     |      |  |  |  |
|------------------------|-----|-----|------|--------------------------|-----|---------|-----|-----|------|-----|-----|------|--|--|--|
| Nombre                 | Dec | Hex | Car. | Dec                      | Hex | Car.    | Dec | Hex | Car. | Dec | Hex | Car. |  |  |  |
| Nulo                   | 0   | 00  | NUL  | 32                       | 20  | Espacio | 64  | 40  | @    | 96  | 60  | `    |  |  |  |
| Inicio de cabecera     | 1   | 01  | SOH  | 33                       | 21  | !       | 65  | 41  | A    | 97  | 61  | a    |  |  |  |
| Inicio de texto        | 2   | 02  | STX  | 34                       | 22  | "       | 66  | 42  | B    | 98  | 62  | b    |  |  |  |
| Fin de texto           | 3   | 03  | ETX  | 35                       | 23  | #       | 67  | 43  | C    | 99  | 63  | c    |  |  |  |
| Fin de transmisión     | 4   | 04  | EOT  | 36                       | 24  | \$      | 68  | 44  | D    | 100 | 64  | d    |  |  |  |
| Enquiry                | 5   | 05  | ENQ  | 37                       | 25  | %       | 69  | 45  | E    | 101 | 65  | e    |  |  |  |
| Acknowledge            | 6   | 06  | ACK  | 38                       | 26  | &       | 70  | 46  | F    | 102 | 66  | f    |  |  |  |
| Campanilla (beep)      | 7   | 07  | BEL  | 39                       | 27  | '       | 71  | 47  | G    | 103 | 57  | g    |  |  |  |
| Backspace              | 8   | 08  | BS   | 40                       | 28  | (       | 72  | 48  | H    | 104 | 68  | h    |  |  |  |
| Tabulador horizontal   | 9   | 09  | HT   | 41                       | 29  | )       | 73  | 49  | I    | 105 | 69  | i    |  |  |  |
| Salto de linea         | 10  | 0A  | LF   | 42                       | 2A  | *       | 74  | 4A  | J    | 106 | 6A  | j    |  |  |  |
| Tabulador vertical     | 11  | 0B  | VT   | 43                       | 2B  | +       | 75  | 4B  | K    | 107 | 6B  | k    |  |  |  |
| Salto de pagina        | 12  | 0C  | FF   | 44                       | 2C  | ,       | 76  | 4C  | L    | 108 | 6C  | l    |  |  |  |
| Retorno de carro       | 13  | 0D  | CR   | 45                       | 2D  | -       | 77  | 4D  | M    | 109 | 6D  | m    |  |  |  |
| Shift fuera            | 14  | 0E  | SO   | 46                       | 2E  | .       | 78  | 4E  | N    | 110 | 6E  | n    |  |  |  |
| Shift dentro           | 15  | 0F  | SI   | 47                       | 2F  | /       | 79  | 4F  | O    | 111 | 6F  | o    |  |  |  |
| Escape línea de datos  | 16  | 10  | DLE  | 48                       | 30  | \       | 80  | 50  | P    | 112 | 70  | p    |  |  |  |
| Control dispositivo 1  | 17  | 11  | DC1  | 49                       | 31  | 1       | 81  | 51  | Q    | 113 | 71  | q    |  |  |  |
| Control dispositivo 2  | 18  | 12  | DC2  | 50                       | 32  | 2       | 82  | 52  | R    | 114 | 72  | r    |  |  |  |
| Control dispositivo 3  | 19  | 13  | DC3  | 51                       | 33  | 3       | 83  | 53  | S    | 115 | 73  | s    |  |  |  |
| Control dispositivo 4  | 20  | 14  | DC4  | 52                       | 34  | 4       | 84  | 54  | T    | 116 | 74  | t    |  |  |  |
| neg acknowledge        | 21  | 15  | NAK  | 53                       | 35  | 5       | 85  | 55  | U    | 117 | 75  | u    |  |  |  |
| Sincronismo            | 22  | 16  | SYN  | 54                       | 36  | 6       | 86  | 56  | V    | 118 | 76  | v    |  |  |  |
| Fin bloque transmitido | 23  | 17  | ETB  | 55                       | 37  | 7       | 87  | 57  | W    | 119 | 77  | w    |  |  |  |
| Cancelar               | 24  | 18  | CAN  | 56                       | 38  | 8       | 88  | 58  | X    | 120 | 78  | x    |  |  |  |
| Fin medio              | 25  | 19  | EM   | 57                       | 39  | 9       | 89  | 59  | Y    | 121 | 79  | y    |  |  |  |
| Sustituto              | 26  | 1A  | SUB  | 58                       | 3A  | :       | 90  | 5A  | Z    | 122 | 7A  | z    |  |  |  |
| Escape                 | 27  | 1B  | ESC  | 59                       | 3B  | ;       | 91  | 5B  | [    | 123 | 7B  | {    |  |  |  |
| Separador archivos     | 28  | 1C  | FS   | 60                       | 3C  | <       | 92  | 5C  | \    | 124 | 7C  |      |  |  |  |
| Separador grupos       | 29  | 1D  | GS   | 61                       | 3D  | =       | 93  | 5D  | ]    | 125 | 7D  | }    |  |  |  |
| Separador registros    | 30  | 1E  | RS   | 62                       | 3E  | >       | 94  | 5E  | ^    | 126 | 7E  | ~    |  |  |  |
| Separador unidades     | 31  | 1F  | US   | 63                       | 3F  | ?       | 95  | 5F  | _    | 127 | 7F  | DEL  |  |  |  |

Tabla G-1 Código ASCII estándar

## APÉNDICE H

# DIRECCIONES DE INTERNET

**EDITORIAL Ra-Ma:** [www.ra-ma.es](http://www.ra-ma.es) o [www.ra-ma.com](http://www.ra-ma.com)

### FABRICANTES Y DISTRIBUIDORES

- [www.microchip.com](http://www.microchip.com) *Microchip Technology Inc.*, fabricante de microcontroladores PIC.
- [www.dalsemi.com](http://www.dalsemi.com) *Dallas Semiconductor-MAXIM*, fabricante de componentes electrónicos utilizados en el libro.
- [www.semiconductors.philips.com](http://www.semiconductors.philips.com) *Philips Semiconductors*, fabricante de componentes utilizados en el libro.
- [www.national.com](http://www.national.com) *National Semiconductor*.
- [www.fairchildsemi.com](http://www.fairchildsemi.com) *Fairchild Semiconductor*.
- [http://sharp-world.com](http://http://sharp-world.com) *Sharp Corporation*.
- [www.futaba-rc.com](http://www.futaba-rc.com) *Futaba*, fabricante servomotores.
- [www.vishay.com](http://www.vishay.com) *Vishay Semiconductors*, fabricante del CNY70 y otros componentes electrónicos.
- [www.optekinc.com](http://www.optekinc.com) *Optek Technology*, fabricante de sensores ópticos y otros.
- [www.isocom.com](http://www.isocom.com) *Isocom Components*, fabricante de optoacopladores.
- [www.robot-electronic1.co.uk](http://www.robot-electronic1.co.uk) *Devantech Ltd*, componentes para microrobótica.
- [www.htsoft.com](http://www.htsoft.com) *Hi-Tech Software*, propietarios de los derechos sobre el compilador PICC.
- [www.ccsinfo.com](http://www.ccsinfo.com) *CCS Inc*, propietarios del compilador PCW.
- [www.melabs.com](http://www.melabs.com) *MicroEngineering Labs*, propietarios del compilador PICBasic Pro.

- [www.sagitron.es](http://www.sagitron.es) Distribuidor de microcontroladores PIC. También **vende** todo tipo de componentes electrónicos.
- [www.modelimport.com](http://www.modelimport.com) Distribuidor de servomotores para **España**. Tiene una útil relación de tiendas que venden productos de modelismo.
- [www.jdm.homepage.dk/newpic.htm](http://www.jdm.homepage.dk/newpic.htm) ProgramadoresJDM.
- [www.ic-prog.com](http://www.ic-prog.com) Software IC-Prog.

### PÁGINAS SOBRE MICROCONTROLADORES EN ESPAÑOL

- [www.terra.es/personal/fremiro](http://www.terra.es/personal/fremiro)
- [www.odisea2010.com](http://www.odisea2010.com)
- [www.todopic.com.ar](http://www.todopic.com.ar)
- [www.conkct.com](http://www.conkct.com)

### FOROS SOBRE MICROCONTROLADORES EN ESPAÑOL

Las páginas Web de la sección anterior también tienen buenos foros sobre microcontroladores PIC. Otros foros:

- <http://ar.groups.yahoo.com/group/PicListLatina>
- <http://boards1.melodysoft.com/app?ID=creatronica.microcontroladores>
- Foros de Google:
  - [es.ciencia.electronica](http://es.ciencia.electronica)
  - [es.ciencia.electronica.micros](http://es.ciencia.electronica.micros)

### VENTA DE COMPONENTES ELECTRÓNICOS

- [www.amidata.es](http://www.amidata.es)
- [www.farnell.com](http://www.farnell.com)
- [www.superrobotica.com](http://www.superrobotica.com)
- [www.telkron.es](http://www.telkron.es)

### AUTORES DE ESTE LIBRO

- [www.terra.es/personal/fremiro](http://www.terra.es/personal/fremiro)
- [www.epalacios.com](http://www.epalacios.com)
- [www.ljlopez.com](http://www.ljlopez.com)

### OTRAS DE ESPECIAL INTERÉS

- [www.comunidadelectronicos.com](http://www.comunidadelectronicos.com) Mucha información útil sobre temas de electrónica.
- <http://personal.redesth.es/castillo> Más información útil.
- [www.piclist.com](http://www.piclist.com) Con mucha información sobre microcontroladores PIC y múltiples enlaces (en inglés).

vende todo

. Tiene una

## APÉNDICE I

### CONTENIDO DEL CD-ROM

foros sobre

es

**El CD-ROM que se incluye en la presente publicación** contiene todos los programas necesarios para realizar los proyectos desarrollados en este texto. Hay además otros ficheros de ayuda o complementarios.

Las diversos subdirectorios o ficheros que contiene son:

MPLAB IDE  
Contiene el archivo MPLAR v.6.40.EXE que es programa de instalación de MPLAB IDE versión 6.40.0.0 y algunos archivos en formato PDF con información sobre el ensamblador MPASM, MPLAB y otros.



Soluciones  
Programas

Los programas resueltos de todos los ejercicios y proyectos.

IC-Prog  
Ficheros para el IC-Prog que es el software de grabación de microcontroladores desarrollado por Bonny Gijzen y recomendado en el libro.

TE20-SE

Ficheros para la realización de la placa de circuito impreso del grabador TE20-SE.

PDF

información técnica sobre los componentes utilizados en el libro.

e temas de

ores PIC y

## ÍNDICE ALFABÉTICO

---

### I

I-Wire Bus, 441  
24LC256, 345  
40106, 515  
4N25, 11

### A

ACK, 335  
Addwf PCL.F, 134, 137, 159  
Algoritmo, 127  
ALU, 43  
AND lógica, 105  
Animate, 92  
Antirrebote, 182, 536  
Arquitectura interna, 37, 47  
ASCII, 613

### B

Bajo consumo, 108, 237  
Banco, 41, 69  
Baudio, 307  
BCD, 131, 147  
Bit ACK, 335  
Bits de configuración, 28, 610  
Bloquear un circuito, 221  
Bonoloto, 251  
Breakpoint, 93  
BT137-400, 18  
Bucles, 125  
Bumper, 536  
Bus de 1 línea, 441  
Bus serie I2C, 331  
Buses, 47

### C

C, 44, 102, 604  
Calendario digital, 371, 379  
Características técnicas, 565  
Carry, 44, 102, 103, 106, 604  
Cerradura electrónica, 299  
Ciclo máquina, 50, 169, 171  
CISC, 51  
CNY70, 522  
Código de operación, 59  
Código fuente, 59, 99  
Código máquina, 57, 70  
Comentarios, 59, 61  
Compilador, 111  
Complementar W, 109  
Configurar líneas, 69  
Constantes numéricas, 63, 587  
Contador de programa, 39, 42, 134, 145, 147, 258, 603  
Convertidor Analógico-Digital, 423  
Convertidor Digital-Analógico, 423  
CP, 46, 610  
CPU, 47  
Cronómetro MPLAB, 171

### D

DC, 44, 102, 604  
DDRAM, 189  
Decrementar, 104, 121  
    un registro, 109  
    W, 110  
Depurar, 117  
Detector de paso por cero, 17  
Diagrama de bloques, 37  
Diagrama de flujo, 127  
Diodo LED, 9

Direccionamiento, 241

Directiva

- CBLOCK, 130, 151, 591
  - CONFIG, 73, 237, 592
  - DE, 218, 592
  - DEFINE, 130, 593
  - DT, 159, 594
  - ELSE, 594
  - END, 71, 149, 151, 595
  - ENDC, 130, 595
  - ENDIF, 595
  - ENDM, 244, 596
  - EQU, 72, 596
  - ERROR, 161, 596
  - IF, 161, 597
  - INCLUDE, 73, 149, 152, 598
  - LIST, 73, 598
  - MACRO, 244, 599
  - MESSG, 160, 599
  - ORG, 72, 600
- Directivas, 71, 589  
Disassembly, 88  
Display de 7 segmentos, 12, 162, 397  
Dividir, 111  
Divisor de Frecuencia, 225  
Dólar, 68  
DS1307, 371  
DS1624, 359, 405  
DS1820, 441

**E**

- Editar, 116  
EEADR, 214, 605  
EECON1, 214, 609  
EECON2, 214, 610  
EEDATA, 214, 217, 605  
EEIE, 258, 607  
EEIF, 215, 609  
EEPROM de datos, 213  
EEPROM serie, 344  
ELA232 Standard, 305  
Emulador, 112  
Emulador ICE 2000, 113  
Ensamblado, 112  
Ensamblar, 85, 116  
Estimulos, 95  
Etiquetas, 59, 60, 100  
Expansor de bus I2C, 409

**F**

- Fairchild Semiconductors, 527  
Fibonacci, 140  
Fichero  
de errores, 59  
fuente, 58  
hexadecimal, 59, 87, 96  
listable, 59, 98  
Formato de las constantes, 63, 587  
FOSC, 45, 610

Fototriac, 16

- Frecuencímetro digital, 234  
FSR, 241, 605  
Full Scale (FS), 429  
Futaba S3003, 50 /

**G**

- GIE, 258, 607  
GP2Dxx, 527  
GPR, 41  
Grabación, 25, 96, 117  
Grabador, 21

**H**

- H21A1, 526  
Halt, 92  
Harvard, 49  
Herramientas, 111  
Hex, 59, 87, 96  
Histércsis, 516  
Hitec HS-300, 507  
HyperTerminal, 314

**I**

- I2C, 332  
IC-Prog, 24, 46, 97, 114, 117, 236, 355  
Área de código, 27  
Área de configuración, 27  
Driver NT/2000/XP, 34  
ID, 46  
Identificadores, 46  
Indexado 134, 241  
Incrementar, 105, 121  
    un registro, 109  
    W, 110  
INDF, 241, 603  
Instrucción  
    addlw, 102, 573  
    addwf, 103, 573  
    andlw, 105, 573  
    andwf, 105, 574  
    bcf, 68, 574  
    bsf, 68, 574  
    btfsz, 120, 575  
    btfsz, 120, 575  
    call, 142, 143, 145, 147, 256, 576  
    clrf, 67, 576  
    clrw, 66, 67, 577  
    clrwdt, 236, 577  
    comf, 106, 577  
    decf, 104, 578  
    decfsz, 121, 578  
    goto, 68, 579  
    incf, 105, 579  
    incfsz, 121, 580  
    iorlw, 105, 580  
    iorwf, 106, 580  
    movf, 67, 581

Instrucc

- mov
- mov
- nop
- retfii
- retlv
- retu
- rlf
- mf
- sleep
- subl
- subv
- swap
- xori
- xorw

Instrucc

- algu
- aritm
- de bi
- de ca
- de re
- de sa
- de su
- esped
- lógic
- repen
- set, 6

INTCON

INTE, 23

INTEDG

Interrup

Aver

EEI, 2

Fases

INT, 2

RBI, 2

Regis

TOL, 2

Interrupt

INTF, 25

Invertir b

IRP1, 605

JDM, 23,

Instrucció

L293B, 4

Lazos, 12

Lazos ani

LCD, 187

LDR, 518

Lenguaje

Alto n

Basic,

C, 112

ensam

máqui

- Instrucción**
- movlw, 67, 581
  - movwf, 67, 582
  - nop, 171, 582
  - retfie, 258, 260, 582
  - retlw, 157, 159, 583
  - return, 142, 143, 146, 147, 583
  - rif, 106, 584
  - rrf, 106, 584
  - sleep, 108, 238, 584
  - sublw, 103, 585
  - subwf, 104, 586
  - swapp, 107, 586
  - xorlw, 107, 587
  - xorwf, 107, 587
- InSTRUCCIONES**
- algunas útiles, 109
  - aritméticas, 64
  - de bit, 64, 68
  - de carga, 64, 66
  - de resta, 103
  - de salto, 64, 119
  - de suma, 102
  - especiales, 64
  - lógicas, 64, 105
  - repertorio, 64, 572
  - set, 64, 572
- INTCON, 224, 226, 257, 258, 77, 606
- INTE, 259, 607
- INTEDG, 260, 608
- Interrupción, 255**
- Averiguar causas, 264
  - EEI, 257
  - Fases, 264
  - INT, 256, 260
  - RBI, 257, 267
  - Registros alterados, 262
  - TOI, 257, 77
- Interruptores, 11**
- INTF, 259, 606
- Invertir bits, 110
- IRP1, 605
- J**
- JDM, 23, 27, 97, 114
- Instrucción**
- L**
- L293B, 469
- Lazos, 125
- Lazos anidados, 174
- LCD, 187
- LDR, 518
- Lenguaje**
- Alto nivel, 112
  - Basic, 112
  - C, 112
  - ensamblador, 58
  - máquina, 57
- Librería**
- BIN\_BCD.INC, 151
  - BUS\_1LIN.INC, 447
  - BUS\_I2C.INC, 340
  - DISPLAY\_7S.INC, 164
  - DS1307.INC, 377
  - DS1624.INC, 364
  - DS1820.INC, 450
  - EPPROM.INC, 216
  - LCD\_4BIT.INC, 194, 196
  - LCD\_MENS.INC, 204
  - M24LC256.INC, 351, 355
  - MACROS.INC, 246
  - P16F84A.INC, 73
  - PCF8574.INC, 412
  - PCF8574\_TECLADO.INC, 417
  - RETARDOS.INC, 176
  - RS232.INC, 312
  - RS232\_MEN.INC, 319
  - TECLADO.INC, 293
- Librerías, 149**
- LIFO, 145
- LM016L, 188
- LST, 59, 98
- M**
- Macros, 243
- Máscara, 110
- MAX232, 309
- MCLR, 7, 44
- Memoria, 1, 37**
- de programa, 37, 99
  - EPPROM de datos, 38, 213
  - Organización, 37
  - RAM de datos, 38, 41, 242
  - ROM Flash, 39
- Memoria 24LC256, 344
- Microbot, 543
- Microbótica, 543
- Microcontrolador, 1, 47, 48
- Microprocesador, 47
- Microrobot, 543
- MOC3041, 16
- Monitorización, 322
- Motor**
- DC, 467
  - de corriente continua, 467, 545
  - PAP, 482
  - PAP Bipolar, 485
  - PAP Unipolar, 487
  - Paso a paso, 482
- MPASM, 59, 71, 78, 112, 116, 589
- MPLAB, 59, 76, 115, 116
- MPLAB SIM, 78, 112, 116
- Multiplexación de displays, 400
- Multiplicar, 111

- N**
- Negar W, 110
  - Nemónicos, 58
  - Normas de estilo, 62
- O**
- OPB703/4/5, 524
  - Operaciones
    - aritméticas, 102
    - lógicas, 105
  - Operadores aritméticos, 64, 588
  - Operando, 59, 61
  - OPTION, 226, 259, 607
  - OPTION\_REG, 226
  - Optoacoplador, 11
  - OR lógica, 105
  - OR-Exclusiva, 107
  - Organización de la memoria, 37
  - Ortogonal, 52
  - OSC1/CLKIN, 4
  - OSC2/CLKOUT, 4, 5
  - Oscilador, 4, 28, 45, 96, 610
- P**
- P16F84A.INC, 61, 62, 73
  - PC, 39, 42, 136, 258, 603
  - PCF8574, 409
  - PCF8591, 423
  - PCH, 39, 42, 136, 258, 603
  - PCL, 39, 42, 136, 258, 603
  - PCLATH, 137, 606
  - PCW, 112
  - PD, 238, 604
  - PIC, 1
  - PIC16C84, 41
  - PICBasic Pro, 112
  - PICC, 112
  - PICDEM 4, 114
  - PICSTART PLUS, 22
  - Pila, 145
  - Pin Stimulus, 96
  - Pipeline, 50
  - Polling, 253
  - PORTA, 42, 53, 605
  - PORTB, 42, 56, 605
  - Power-On Reset POR, 7
  - Prescaler, 225, 236
  - Programa, 1
  - Programa de control, 21
  - Programa ensamblador, 58, 112
  - Programación estructurada, 155
  - Programador, 21
  - Protoboard, 8
  - Proyectos, 115
  - PS2:PS0, 226, 607
  - PSA, 227, 608
  - Puente en H, 468
  - Puerto serie RS-232, 305
- Puertos, 3, 42, 53, 69, 605**
- Pull-Up, 249, 291, 333, 417, 446**
- Pulsador, 11, 181**
- Puntero, 241**
- Punto de paro, 93**
- PWM, 475**
- PWRTE, 45, 610**
- Q**
- Quintela, 211
- R**
- RBIE, 259, 606
  - RBIF, 56, 259, 606
  - RBPU, 250, 608
  - RD, 215, 609
  - Rebotes, 181
  - Recursos especiales, 235
  - Registro
    - Configuración, 45, 610
    - de estado, 43, 604
    - de trabajo W, 43
  - Registros
    - Funciones especiales, 41, 42
    - Propósito general, 41
    - Tras un reset, 44
  - Relé, 14
  - Reloj, 4, 371
  - Reloj digital, 94, 288, 379
  - Reset, 7, 44, 93
  - Reset PWRT, 7
  - Resolución, 428
  - Retardo, 172
  - RISC, 51
  - Rotación, 106
  - RP0, 41, 44, 69, 604
  - RP1, 605
  - RS-232, 305
  - Run, 93
  - Run to Cursor, 94
- S**
- SAA1064, 397
  - Salto
    - Condicional, 119
    - Incondicional, 68
    - Indexado, 134, 159
  - Schmitt, 516
  - SCL, 332
  - SDA, 332
  - Segmentado, 50
  - Sensores, 515
  - Servomotores, 506, 545
  - Servos, 506
  - SFH5110, 532
  - SFR, 41, 42
  - Simulación, 92

Simulador, 112, 116  
 Simulate Trace, 94  
 SISC, 51  
 Sistema de desarrollo, 114  
 Sleep, 44, 238  
 SRF04, 538  
 Stack, 145, 154  
 Standby, 237  
 Start, 335  
 STAIUS, 43, 238, 604  
 Step Into, 92, 154  
 Step Over, 154  
 Stop, 335  
 Stopwatch, 171, 232  
 Subrutina, 128, 141  
     anidada, 143  
     de retardo, 172  
     interrupción, 255  
     simulación, 154

## T

T0CKI, 224  
 T0CS, 224, 227, 608  
 T0IE, 259, 607  
 T0IF, 224, 226, 259, 606  
 T0SE, 224, 227, 608  
 Tabla de datos, 157  
 Tabla ROM, 159  
 Tablas de verdad, 157  
 TF20-SE, 23, 115, 117, 355  
 Teclado hexadecimal, 290, 417  
 Teclado matricial, 290  
 Temporizaciones  
     largas, 81  
     exactas, 79  
 Temporizador, 224  
 Temporizador digital, 83

Termómetro digital, 367, 404, 450  
 Termostato digital, 453  
 Tiempo de retardo, 172  
 Timer 0, 223  
 TMR0, 223, 603  
 TO, 239, 604  
 Triac, 17, 18  
 Trigger Schmitt, 516  
 TRISA, 42, 53, 69, 86, 608  
 TRISB, 42, 69, 86, 609

## U

ULN2003, 15

## V

Vector de interrupción, 258  
 Ventanas, 88  
 Visualización dinámica, 400  
 Visualizar mensajes, 204, 207  
 Von Neumann, 48

## W

W, 43  
 Watchdog, 45, 225, 235, 610  
 WDT, 45, 225, 235, 610  
 WDTE, 45, 610  
 WR, 215, 609  
 WREN, 215, 609  
 WRERR, 215, 609

## Z

Z, 44, 102, 604  
 Zumbador piezoelectrónico, 19

**CM27/E1/04**

Esta edición se terminó de imprimir en agosto de 2004. Publicada por ALFAOMEGA GRUPO EDITOR, S.A. de C.V. Apartado Postal 73-267, 03311, México, D.F. La impresión se realizó en TALLERES GRÁFICOS DEL D.F., Puente Moralillo No. 49, Col. Puente Colorado, C.P. 01730, México, D.F.