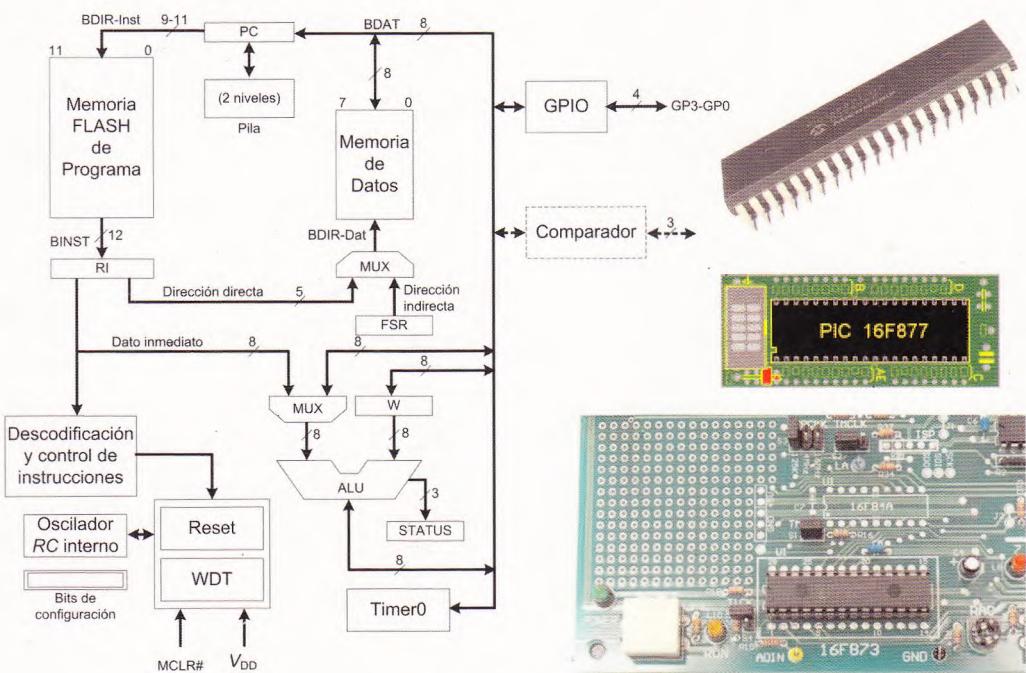


MICROCONTROLADORES

FUNDAMENTOS Y APLICACIONES CON PIC

Fernando E. Valdés Pérez
Ramón Pallás Areny



Alfaomega



marcombo

Fernando E. Valdés Pérez
Ramon Pallàs Areny

MICROCONTROLADORES: FUNDAMENTOS Y APLICACIONES CON PIC

Fernando E. Valdés Pérez

Universidad de Oriente

Cuba

Ramon Pallàs Areny

Universidad Politécnica de Cataluña

España

MICROCONTROLADORES: FUNDAMENTOS Y APLICACIONES CON PIC



Título

Microcontroladores: Fundamentos y aplicaciones con PIC

Autores

© Fernando E. Valdés Pérez
© Ramon Pallàs Areny

Editores

© MARCOMBO, EDICIONES TÉCNICAS 2007

MARCOMBO, S.A.

Gran Via de les Corts Catalanes 594

08007 Barcelona (España)

© ALFAOMEGA GRUPO EDITOR, S.A. 2007

C/ Pitágoras 1139

Colonia del Valle - 03100

México D.F. (Méjico)

ISBN (por MARCOMBO): 84-267-1414-5

ISBN-13 (por MARCOMBO): 978-84-267-1414-5

ISBN (por ALFAOMEGA GRUPO EDITOR): 970-15-1149-2

ISBN-13 (por ALFAOMEGA GRUPO EDITOR): 978-970-15-1149-7

Composición y pre impresión: Carles Parcerisas Civit (3Q Editorial)

D.L.: BI-3396-06

Impreso en Gráficas Díaz Tudurí

Índice general

Prólogo	9
1 Introducción a los microcontroladores	11
1.1 Micropresesadores y microcontroladores: caracterización.....	11
1.2 Componentes de un microcontrolador	14
1.2.1 <i>El perro guardián</i>	17
1.2.2 <i>Reset</i>	18
1.2.3 <i>Bajo consumo</i>	19
1.2.4 <i>Protección del programa frente a copias</i>	21
1.3 Arquitecturas von Neumann y Harvard	21
1.4 Arquitecturas CISC y RISC.....	24
1.5 Microcontroladores, micropresesadores y sus fabricantes	24
2 Los microcontroladores PIC	29
2.1 Características generales de los microcontroladores PIC	29
2.1.1 <i>La Unidad Aritmética y Lógica y el registro W en los microcontroladores PIC</i>	30
2.1.2 <i>Ciclos de máquina y ejecución de instrucciones</i>	31
2.1.3 <i>Segmentado (pipeline) en la ejecución de instrucciones</i>	33
2.1.4 <i>Osciladores</i>	34
2.1.5 <i>Bits de configuración</i>	36
2.1.6 <i>Fuentes de reset</i>	38
2.1.7 <i>Modo de bajo consumo</i>	42
2.1.8 <i>Perro guardián</i>	43
2.2 Familias de microcontroladores PIC.....	45
2.2.1 <i>Microcontroladores de gama baja</i>	45
2.2.2 <i>Microcontroladores de gama media</i>	47
2.2.3 <i>Microcontroladores de gama alta</i>	51
3 La memoria en los microcontroladores	55
3.1 Conceptos básicos	55
3.1.1 <i>Organización lógica de la memoria</i>	56
3.1.2 <i>Tipos de memorias</i>	58
3.2 La memoria en los microcontroladores PIC de gama media.....	61
3.2.1 <i>La memoria de programa</i>	62
3.2.1.1 <i>Direccionamiento de la memoria de programa</i>	62
3.2.1.2 <i>Lectura y escritura de la memoria de programa</i>	64

3.2.2	<i>La memoria RAM de datos</i>	68
3.2.2.1	<i>Direccionamiento de la memoria de datos</i>	70
3.2.2.2	<i>Registros de funciones especiales</i>	72
3.2.3	<i>La memoria EEPROM de datos</i>	75
4	Repertorio de instrucciones y programación en lenguaje ensamblador	79
4.1	Conceptos básicos	79
4.1.1	<i>Código de máquina y lenguaje ensamblador</i>	79
4.1.2	<i>Estructura de las instrucciones</i>	82
4.1.3	<i>Modos de direccionar los datos</i>	84
4.1.4	<i>La pila</i>	85
4.2	Repertorio de instrucciones de los PIC de gama media.....	88
4.2.1	<i>Instrucciones de transferencia de datos</i>	91
4.2.2	<i>Instrucciones aritméticas y lógicas</i>	92
4.2.3	<i>Instrucciones de transferencia de control</i>	95
4.2.3.1	<i>Saltos incondicionales, llamadas o subrutinos y retornos</i>	95
4.2.3.2	<i>Saltos condicionados</i>	99
4.2.4	<i>Instrucciones que operan con bits</i>	102
4.2.5	<i>Otras instrucciones</i>	103
4.3	Elementos del lenguaje ensamblador (para el ensamblador MPASM de Microchip).....	103
4.3.1	<i>Introducción</i>	103
4.3.2	<i>Expresiones, operaciones y operadores</i>	109
4.3.2.1	<i>Operadores aritméticos</i>	110
4.3.2.2	<i>Operadores lógicos y de relación</i>	111
4.3.2.3	<i>Operadores lógicos que operan directamente con bits</i>	112
4.3.2.4	<i>Operadores de asignación</i>	112
4.3.2.5	<i>Operadores de dirección</i>	114
4.3.3	<i>Directivas</i>	114
4.3.3.1	<i>Directivas de uso general</i>	116
4.3.3.2	<i>Directivas utilizadas en la codificación relocable</i>	121
4.3.4	<i>Macroinstrucciones</i>	127
4.3.5	<i>Organización de un programa en lenguaje ensamblador</i>	130
4.4	Recursos disponibles para programar en el lenguaje ensamblador de los microcontroladores PIC	136
4.4.1	<i>El ensamblador MPASM</i>	137
4.4.1.1	<i>Generación de código absoluto</i>	137
4.4.1.2	<i>Generación de código relocable</i>	138
4.4.1.3	<i>Archivos involucrados en el ensamblaje</i>	139
4.4.2	<i>El enlazador MPLINK</i>	142
4.4.3	<i>El gestor de bibliotecas MPLIB</i>	145
5	La entrada y salida en paralelo.	149
5.1	Conceptos básicos sobre entradas y salidas digitales	149
5.1.1	<i>Métodos de transferencia de datos</i>	150
5.1.2	<i>Técnicas de entrada y salida</i>	153

5.2 Los puertos paralelos en los PIC de clase media	155
5.2.1 <i>El puerto A</i>	158
5.2.2 <i>El puerto B.....</i>	159
5.2.3 <i>El puerto C.....</i>	161
5.2.4 <i>Los puertos D, E, F y G</i>	161
5.2.5 <i>El Puerto Paralelo Esclavo.....</i>	162
5.3 Conexión y tratamiento a periféricos comunes	164
5.3.1 <i>Interruptores y diodos LED.....</i>	164
5.3.2 <i>Teclados matriciales</i>	169
5.3.3 <i>Visualizadores numéricos de 7 segmentos</i>	177
5.3.4 <i>Visualizadores alfanuméricos de cristal líquido.....</i>	180
6 Los temporizadores	189
6.1 Los temporizadores en los microcontroladores PIC	189
6.1.1 <i>El módulo Timer0</i>	190
6.1.2 <i>El módulo Timer1</i>	195
6.1.3 <i>El módulo Timer2</i>	199
6.2 El módulo CCP	202
6.2.1 <i>Modo de captura</i>	204
6.2.2 <i>Modo comparador.....</i>	208
6.2.3 <i>Modo PWM</i>	210
7 Las interrupciones	217
7.1 Conceptos básicos sobre las interrupciones	217
7.1.1 <i>Las solicitudes de interrupción y recursos asociados.....</i>	217
7.1.2 <i>Atención a las solicitudes de interrupción</i>	219
7.1.3 <i>Interrupciones fijas y vectorizadas</i>	221
7.2 Las interrupciones en los microcontroladores PIC	224
7.2.1 <i>Fuentes de interrupción y registros asociados</i>	224
7.2.2 <i>Estructura del subprograma de atención a una interrupción</i>	230
7.3 Ejemplos de uso de las interrupciones.....	234
7.3.1 <i>Reloj de tiempo real.....</i>	234
7.3.2 <i>Sincronización de eventos al reloj de tiempo real</i>	239
7.3.3 <i>Protección contra fallos de hardware</i>	243
8 La entrada y salida en serie	247
8.1 Conceptos básicos sobre entradas y salidas en serie	247
8.1.1 <i>Introducción a la transmisión de datos en serie.....</i>	247
8.1.2 <i>Comunicación asincrónica</i>	249
8.1.3 <i>Comunicación sincrónica</i>	249
8.1.4 <i>Conexión entre equipos: interfaz RS-232C</i>	250
8.1.5 <i>El bus I²C</i>	252

8.2 El puerto serie USART en los microcontroladores PIC.....	258
8.2.1 Descripción general.....	258
8.2.2 Funcionamiento en modo asincrónico.....	259
8.2.3 Funcionamiento en modo sincrónico.....	262
8.2.4 Velocidad de la comunicación.....	263
8.3 El puerto serie SSP en los microcontroladores PIC	265
8.3.1 Interfaz SPI.....	265
8.3.2 Interfaz PC	270
9 Las entradas y salidas analógicas. Adquisición y distribución de señales.	275
9.1 Funciones y estructura de un sistema de adquisición y distribución de señales	275
9.1.1 Funciones básicas en los sistemas de medida y control.....	275
9.1.2 Margen o rango dinámico	278
9.1.3 Ancho de banda	280
9.1.4 Muestreo de señales.....	282
9.1.5 Arquitecturas para la adquisición de señales. Sistemas de alto y bajo nivel	283
9.2 La etapa frontal para la adquisición de señales	284
9.2.1 Atenuadores.....	285
9.2.2 Amplificadores	289
9.2.3 Filtros y protecciones de entrada.....	293
9.2.4 Multiplexores analógicos	296
9.2.5 Filtros anti-alias.....	298
9.2.6 Amplificador de muestreo y retención.....	300
9.2.7 Convertidores A/D	302
9.3 El módulo de conversión A/D de 10 bits en los microcontroladores PIC.....	305
9.3.1 Arquitectura del módulo de conversión A/D	305
9.3.2 Tiempos de una conversión A/D.....	309
9.3.3 Programación del módulo de conversión A/D.....	312
9.4 Calibración	315
9.5 Interfaces directas entre sensor y microcontrolador	316
9.6 La etapa de salida para salidas analógicas	320
9.6.1 Convertidores D/A	320
9.6.2 Desmultiplexado analógico	321
9.6.3 Métodos de extrapolación	321
9.6.4 Salidas PWM	322
9.6.5 Protecciones de salida	324
Bibliografía	327
Anexo. Siglas y acrónimos utilizados en el libro.	329
Índice alfabético	335

Prólogo

Los microcontroladores están presentes en muchos de los productos electrónicos que empleamos en nuestra vida cotidiana. Su enseñanza es un reto debido a la variedad de modelos existentes en el mercado y a la gran cantidad de aplicaciones posibles. Sin embargo, a pesar de su diversidad, hay unidad en los principios de funcionamiento y en las arquitecturas de muchos microcontroladores. Este libro aprovecha esa unidad presente en la diversidad para explicar los fundamentos del diseño y la programación de los microcontroladores.

El objetivo del libro es enseñar la arquitectura y la programación de los microcontroladores en general, tomando como ejemplos los microcontroladores PIC de Microchip. La documentación que ofrecen los fabricantes es tan abundante que su mero acopio ocuparía varios volúmenes. En este libro se han seleccionado los temas de forma fundamentada, buscando el rigor en las descripciones y la claridad en la exposición de los conceptos. Se han incluido figuras que complementan el texto de forma sustancial, evitando fotografías u otro material gráfico que aumenta el número de páginas y aporta información menos útil, y por lo demás disponible en los sitios web de los fabricantes. Para ayudar a identificar los conceptos, los términos nuevos están en letra cursiva la primera vez que aparecen en el texto. Donde ha parecido oportuno, se ha incluido entre paréntesis y en letra cursiva, el término equivalente en inglés.

Cada tema es tratado con un enfoque que va de lo general a lo particular. Primero se explican las cuestiones propias del tema que son comunes a la mayoría de los microcontroladores, y seguidamente se particulariza para los microcontroladores PIC. Las explicaciones se ilustran con ejemplos prácticos. En el sitio web del libro, en www.marcombo.com, el lector interesado podrá encontrar otros ejemplos resueltos y problemas de interés para estudiantes y profesores.

El libro está organizado en nueve capítulos. En el capítulo 1 se estudian las características generales y los recursos comúnmente disponibles en los microcontroladores. El capítulo 2 ofrece una visión panorámica de los microcontroladores PIC, en particular los de la gama media. En el capítulo 3 se estudia la organización y estructura de la memoria en los microcontroladores

en general y en los PIC de gama media en particular. El capítulo 4 trata la programación en el lenguaje ensamblador de los microcontroladores PIC de gama media. La programación en lenguaje ensamblador es la mejor forma de programar los proyectos con microcontroladores que son pequeños, por lo que es esencial minimizar la cantidad de memoria utilizada, y donde las tareas que debe realizar el microcontrolador son pocas y los algoritmos son relativamente simples. La programación de algoritmos complejos es mejor hacerla con lenguajes de alto nivel, y ello exige disponer del correspondiente compilador (que no es gratuito). Los capítulos 5, 6, 7 y 8 explican cómo los microcontroladores pueden adquirir, procesar y generar señales digitales. Se estudian las técnicas disponibles para el tratamiento de la entrada y salida digital de datos en paralelo, la conexión y tratamiento de periféricos de amplio uso en sistemas con microcontroladores, los recursos disponibles para trabajar con la variable tiempo, las fuentes, recursos y tratamiento dado a las interrupciones, y la transmisión de información en serie, sus formatos, parámetros e interfaces utilizadas en los microcontroladores PIC. En el capítulo 9 se estudia cómo se pueden adquirir y generar señales analógicas empleando módulos o dispositivos integrados en el propio microcontrolador, o en circuitos periféricos, y se dan los criterios básicos para diseñar el sistema.

El libro incluye un amplio índice alfabético de términos técnicos utilizados, incluyendo los términos en inglés, y, en un anexo, una relación de las siglas y acrónimos empleados en el texto. Al final, se dan referencias bibliográficas para quienes deseen profundizar en los temas tratados en el libro.

El libro está dirigido especialmente a estudiantes y a profesionales de la electrónica, pero también resultará útil a los lectores interesados en conocer el fascinante mundo de los microcontroladores, en particular de los PIC, y utilizarlos de forma eficiente en un sinfín de aplicaciones.

*Fernando E. Valdés Pérez
Ramon Pallàs Areny*

1 Introducción a los microcontroladores

En este capítulo se estudian las características generales comunes a muchos microcontroladores. Primeramente se define lo que se entiende por microcontrolador y se establece su diferencia con el microprocesador. Seguidamente se exponen los recursos comúnmente disponibles en un microcontrolador, haciendo énfasis en aquellos recursos propios de estos dispositivos y que no se encuentran en los microprocesadores en general. Se presenta también las arquitecturas von Neumann y Harvard empleadas en el diseño de los microcontroladores, así como las arquitecturas CISC y RISC, diferenciadas fundamentalmente por la forma de concebir el repertorio de instrucciones del microcontrolador. Finalmente se enumeran los principales microcontroladores existentes y sus fabricantes.

1.1 Microprocesadores y microcontroladores: caracterización

La figura 1.1 muestra el esquema general básico de un microcomputador. Se compone de tres bloques fundamentales: la CPU (*Central Processing Unit*), la memoria, y la entrada y salida. Los bloques se conectan entre sí mediante grupos de líneas eléctricas denominados *buses*. Los buses pueden ser de direcciones (si transportan direcciones de memoria o de entrada y salida), de datos (si transportan datos o instrucciones) o de control (si transportan señales de control diversas).

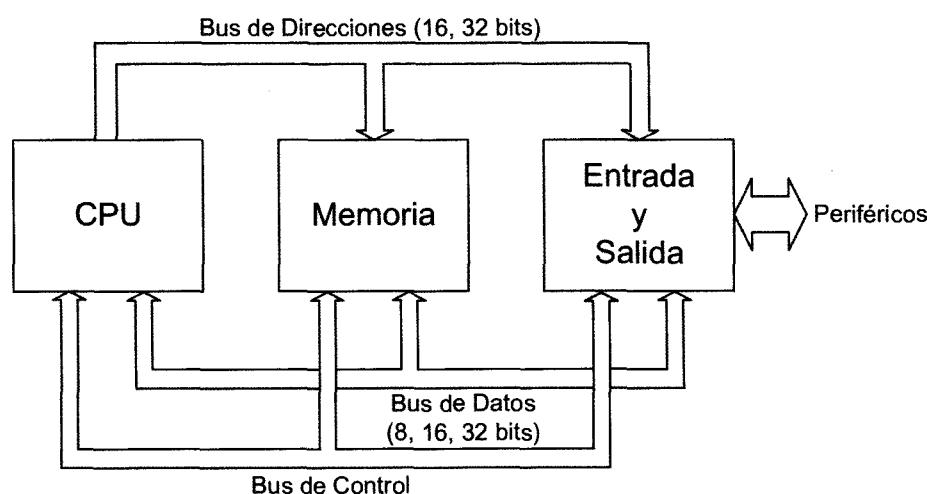


Figura 1.1 Esquema básico general de un microcomputador. La CPU es el microprocesador.

La CPU es el “cerebro” del microcomputador y actúa bajo el control del programa almacenado en la memoria. La CPU se ocupa básicamente de traer las instrucciones del programa desde la memoria, interpretarlas y hacer que se ejecuten. La CPU también incluye los circuitos para realizar operaciones aritméticas y lógicas elementales con los datos binarios, en la denominada Unidad Aritmética y Lógica (*ALU: Arithmetic and Logic Unit*).

En un microcomputador, la CPU no es otra cosa que el *microprocesador*, el circuito integrado capaz de realizar las funciones antes mencionadas. Un *microcontrolador* es un microcomputador realizado en un circuito integrado (*chip*). Históricamente, los microcontroladores aparecieron con posterioridad a los microprocesadores y han tenido evoluciones independientes.

Los microprocesadores se han desarrollado fundamentalmente orientados al mercado de los ordenadores personales y las estaciones de trabajo, donde se requiere una elevada potencia de cálculo, el manejo de gran cantidad de memoria y una gran velocidad de procesamiento. Un parámetro importante en los microprocesadores es el tamaño de sus registros internos (8, 16, 32 ó 64 bits), que determina la cantidad de bits que pueden procesar simultáneamente.

Los microcontroladores se han desarrollado para cubrir las más diversas aplicaciones. Se usan en automoción, en equipos de comunicaciones y de telefonía, en instrumentos electrónicos, en equipos médicos e industriales de todo tipo, en electrodomésticos, en juguetes, etc.

Los microcontroladores están concebidos fundamentalmente para ser utilizados en aplicaciones puntuales, es decir, aplicaciones donde el microcontrolador debe realizar un pequeño número de tareas, al menor costo posible. En estas aplicaciones, el microcontrolador ejecuta un programa almacenado permanentemente en su memoria, el cual trabaja con algunos datos almacenados temporalmente e interactúa con el exterior a través de las líneas de entrada y salida de que dispone. El microcontrolador es parte de la aplicación: es un controlador incrustado o embebido en la aplicación (*embedded controller*). En aplicaciones de cierta envergadura se utilizan varios microcontroladores, cada uno de los cuales se ocupa de un pequeño grupo de tareas.

Hay varias características que son deseables en un microcontrolador:

1. Recursos de entrada y salida. Más que en la capacidad de cálculo del microcontrolador, muchas veces se requiere hacer énfasis en los recursos de entrada y de salida del dispositivo, tales como el manejo individual de líneas de entrada y salida, el manejo de interrupciones, señales analógicas, etc.

2. Espacio optimizado. Se trata de tener en el menor espacio posible, y a un coste razonable, los elementos esenciales para desarrollar una aplicación. Dado que el número de terminales que puede tener un circuito integrado viene limitado por las dimensiones de su encapsulado, el espacio se puede optimizar haciendo que unos mismos terminales realicen funciones diferentes.
3. El microcontrolador idóneo para una aplicación. Se procura que el diseñador disponga del microcontrolador hecho a la medida de su aplicación. Por esto los fabricantes ofrecen familias de microcontroladores, compuestas por miembros que ejecutan el mismo repertorio de instrucciones pero que difieren en sus componentes de hardware (más o menos memoria, más o menos dispositivos de entrada y salida, etc.), permitiendo así que el diseñador de aplicaciones pueda elegir el microcontrolador idóneo para cada aplicación.
4. Seguridad en el funcionamiento del microcontrolador. Una medida de seguridad elemental es garantizar que el programa que esté ejecutando el microcontrolador sea el que corresponde, es decir, que si el microcontrolador se “ pierde”, esto pueda ser rápidamente advertido y se tome alguna acción para corregir la situación. Un componente común en los microcontroladores y que contribuye a una operación segura es el perro guardián (WDT: *Watchdog Timer*), dispositivo que no existe en los ordenadores personales.
5. Bajo consumo. Dado que hay muchas aplicaciones donde se desea utilizar baterías como fuente de alimentación, es altamente deseable que el microcontrolador consuma poca energía. También interesa que el microcontrolador consuma muy poco cuando no está realizando ninguna acción. Por ejemplo, si está a la espera de que se pulse una tecla, el microcontrolador debería consumir muy poco durante la espera; para ello conviene paralizar total o parcialmente al microcontrolador, poniéndolo a “ dormir” hasta que ocurra la acción esperada.
6. Protección de los programas frente a copias. Se trata de proteger la información almacenada en la memoria, es decir, el programa de la aplicación, contra lecturas furtivas de la memoria del microcontrolador. Los microcontroladores disponen de mecanismos que les protegen de estas acciones.

1.2 Componentes de un microcontrolador

Un microcontrolador combina los recursos fundamentales disponibles en un microcomputador, es decir, la unidad central de procesamiento (CPU), la memoria y los recursos de entrada y salida, en un único circuito integrado. La figura 1.2 muestra el diagrama de bloques general de un microcontrolador.

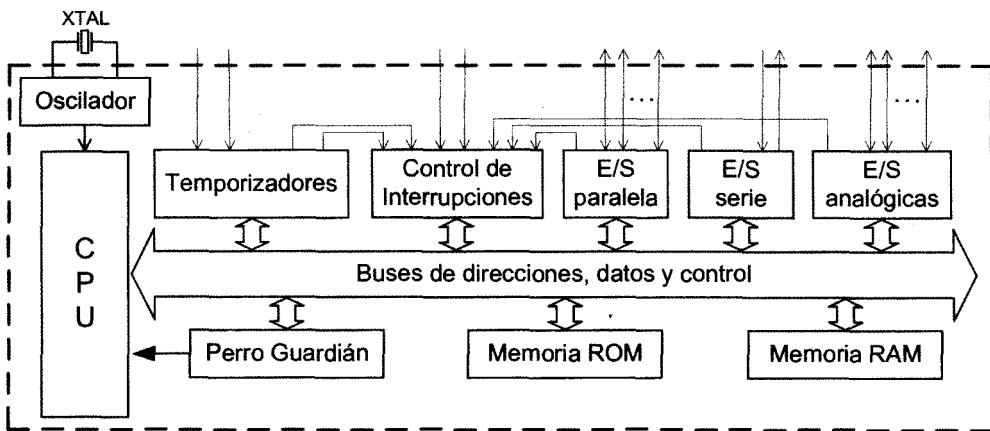


Figura 1.2 Esquema de bloques general de un microcontrolador.

Los microcontroladores disponen de un oscilador que genera los pulsos que sincronizan todas las operaciones internas. El oscilador puede ser del tipo *RC*, aunque generalmente se prefiere que esté controlado por un cristal de cuarzo (XTAL) debido a su gran estabilidad de frecuencia. La velocidad de ejecución de las instrucciones del programa está en relación directa con la frecuencia del oscilador del microcontrolador.

Igual que en un microcomputador, la CPU es el “cerebro” del microcontrolador. Esta unidad trae las instrucciones del programa, una a una, desde la memoria donde están almacenadas, las interpreta (descodifica) y hace que se ejecuten. En la CPU se incluyen los circuitos de la ALU para realizar operaciones aritméticas y lógicas elementales con los datos binarios.

La CPU de un microcontrolador dispone de diferentes registros, algunos de propósito general y otros para propósitos específicos. Entre estos últimos están el *Registro de Instrucción*, el *Acumulador*, el *Registro de Estado*, el *Contador de Programa*, el *Registro de Direcciones de Datos* y el *Puntero de la Pila*.

El Registro de Instrucción (RI) almacena la instrucción que está siendo ejecutada por la CPU. El RI es invisible para el programador.

El Acumulador (ACC: *Accumulator*) es el registro asociado a las operaciones aritméticas y lógicas que se pueden realizar en la ALU. En cualquier operación, uno de los datos debe estar en el ACC y el resultado se obtiene en el ACC. El ACC no existe en los microcontroladores PIC, que tienen en cambio el registro W (*Working Register*), con características muy parecidas a las del ACC.

El Registro de Estado (STATUS) agrupa los bits indicadores de las características del resultado de las operaciones aritméticas y lógicas realizadas en la ALU. Entre estos indicadores están el signo del resultado (si es positivo o negativo), si el resultado es cero, si hay acarreo o préstamo, el tipo de paridad (par o impar) del resultado, etc.

El Contador de Programa (PC: *Program Counter*) es el registro de la CPU donde se almacenan direcciones de instrucciones. Cada vez que la CPU busca una instrucción en la memoria, el PC se incrementa, apuntando así a la siguiente instrucción. En un instante de tiempo dado, el PC contiene la dirección de la instrucción que será ejecutada a continuación. Las instrucciones de transferencia de control modifican el valor del PC.

El Registro de Direcciones de Datos (RDD) almacena direcciones de datos situados en la memoria. Este registro es indispensable para el direccionamiento indirecto de datos en la memoria. El RDD toma diferentes nombres según el microcontrolador. En los PIC, el RDD es el registro FSR (*File Select Register*).

El Puntero de la Pila (SP: *Stack Pointer*) es el registro que almacena direcciones de datos en la pila. En el capítulo 4 se estudian con detalle la pila y el registro SP. Los microcontroladores PIC carecen de registro SP.

La memoria del microcontrolador es el lugar donde son almacenadas las instrucciones del programa y los datos que manipula. En un microcontrolador siempre hay dos tipos de memoria: la memoria RAM (*Random Access Memory*) y la memoria ROM (*Read Only Memory*). La memoria RAM es una memoria de lectura y escritura, que además es volátil, es decir, pierde la información almacenada cuando falta la energía que alimenta la memoria. La memoria ROM es una memoria de solo lectura y no volátil. Las diferentes tecnologías para realizar las memorias de solo lectura (ROM, EPROM, EEPROM, OTP, FLASH) se estudian en el capítulo 3. Tanto la memoria RAM como las memorias ROM son de acceso aleatorio, pero la costumbre ha dejado el nombre de RAM para las memorias de lectura y escritura. El término “acceso aleatorio” se refiere a que el tiempo necesario para localizar un dato no depende del lugar de la memoria donde esté almacenado. En las memo-

rias de acceso secuencial, en cambio, cuando más alejado esté un dato de la posición a la que se ha accedido por última vez, más se tarda en localizarlo.

La memoria ROM se usa para almacenar permanentemente el programa que debe ejecutar el microcontrolador. En la memoria RAM se almacenan temporalmente los datos con los que trabaja el programa. Un número creciente de microcontroladores dispone de alguna memoria no volátil de tipo EEPROM para almacenar datos fijos o que sólo sean cambiados esporádicamente.

La cantidad de memoria ROM disponible es normalmente muy superior a la cantidad de memoria RAM. Esto obedece a dos razones: la primera es que la gran mayoría de las aplicaciones requieren programas que manejan pocos datos; entonces basta una memoria RAM en la que se pueda almacenar algunas decenas de datos. La segunda razón es que la memoria RAM ocupa mucho más espacio en el circuito integrado que la memoria ROM, de modo que es mucho más costosa que ésta.

La entrada y salida es particularmente importante en los microcontroladores, pues a través de ella el microcontrolador interacciona con el exterior. Forman parte de la entrada y salida los puertos paralelo y serie, los temporizadores y la gestión de las interrupciones. El microcontrolador puede incluir también entradas y salidas analógicas asociadas a convertidores A/D y D/A. Tienen particular importancia los recursos que garantizan un funcionamiento seguro del microcontrolador, como el denominado perro guardián.

Los puertos paralelos se organizan en grupos de hasta 8 líneas de entradas y salidas digitales. Normalmente es posible manipular individualmente las líneas de los puertos paralelos. Los puertos serie pueden ser de varios tipos, según la norma de comunicación que implementen: RS-232C, I²C, USB, Ethernet, etc.

Un requisito general para que un microcontrolador se pueda utilizar en un gran número de aplicaciones es que tenga muchos recursos de entrada y salida. Este requisito está relacionado con el número de terminales del circuito integrado. A la vez se desea tener esos recursos en un circuito integrado lo más pequeño posible. Si cada bloque de entrada y salida tuviera terminales exclusivos en el microcontrolador, posiblemente no alcanzarían los terminales o habría que fabricar microcontroladores con muy pocos recursos de entrada y salida, en contra del interés general. La solución para este problema es que un número importante de terminales del circuito integrado sean compartidos por unidades de entrada y salida diferentes. Es decir, un mismo terminal puede estar conectado internamente a más de un bloque de entrada

y salida. Por ejemplo, las líneas de entrada y salida de un puerto serie pueden ser parte de las líneas de un puerto paralelo, etc.

1.2.1 El perro guardián

El perro guardián (WDT: *Watchdog Timer*) es un recurso disponible en muchos microcontroladores. La figura 1.3 muestra el esquema básico de un perro guardián. Consta de un oscilador y un contador binario de N bits. El oscilador puede ser el oscilador principal del microcontrolador, aunque se prefiere un oscilador independiente. La salida de la última etapa del contador va conectada al circuito de *reset* del microcontrolador. El conteo no se puede detener de ninguna forma, pero el contador se puede borrar (poner a 0) desde el programa.

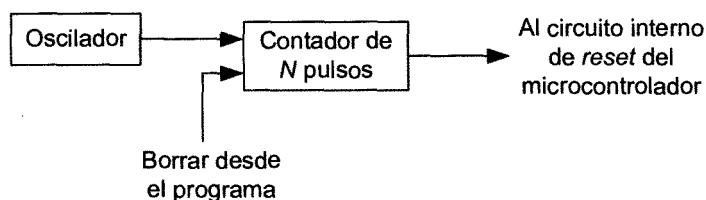


Figura 1.3 Esquema básico de un perro guardián. La salida del perro guardián va a algún punto del circuito interno que tiene el microcontrolador para realizar la acción de *reset*.

El funcionamiento del perro guardián es como sigue. El oscilador envía sus pulsos periódica y permanentemente a la entrada de reloj del contador. Si el contador llega a contar los N pulsos, se desborda, su salida se activa y produce el *reset* del microcontrolador.

El objetivo del programador es evitar el desbordamiento del perro guardián. Dado que una vez iniciado el conteo, el perro guardián no se puede detener, la única forma de evitar su desbordamiento, y con ello el *reset* del microcontrolador, es poner a 0 el contador del perro guardián desde el programa, y hacerlo a intervalos de tiempo más cortos que el tiempo que se tarda en contar los N pulsos. Para conseguirlo, el programador debe distribuir a lo largo del programa las instrucciones que borran el perro guardián. Si el programa se ejecuta correctamente, el perro guardián nunca se desborda pues antes de hacerlo ha sido borrado oportunamente desde el programa. En cambio, si el microcontrolador se pierde y el programa deja de ser ejecutado en la secuencia correcta, el perro guardián no es borrado a tiempo, se desborda y produce el *reset* del microcontrolador, con lo que es posible retomar el control y reconducir el programa por el camino correcto.

El perro guardián es un elemento muy importante en un microcontrolador, pues garantiza la seguridad de su funcionamiento, porque cualquier fallo es detectado a tiempo y se pueden tomar las medidas necesarias para evitar situaciones que podrían ser catastróficas.

1.2.2 Reset

El *reset* es una acción con la cual se “inicia” el trabajo de los microprocesadores y microcontroladores. Esta acción se ejecuta cuando se aplica una señal —denominada de *reset*— a un terminal, designado también como *reset*. El efecto práctico de la señal es poner el contador de programa (PC) en un valor predeterminado (por ejemplo, $PC = 0$), haciendo así que el microprocesador o microcontrolador comience a ejecutar las instrucciones que están a partir de esa posición de memoria apuntada por el PC.

En un microcomputador, la señal de *reset* se genera manualmente al pulsar un botón (*reset manual*) o cuando se pone en marcha el sistema (*reset por encendido*). La figura 1.4 muestra un circuito utilizado comúnmente para excitar el terminal de *reset* de un microprocesador o microcontrolador, y que facilita el *reset* manual y el *reset* por encendido. En el esquema mostrado, V_{RESET} es la tensión aplicada en el terminal RESET y V_{UMBRAL} es la tensión umbral de dicho terminal. Si $V_{RESET} < V_{UMBRAL}$, el microcontrolador interpreta que en el terminal hay un valor lógico “0” ($RESET = 0$); si $V_{RESET} > V_{UMBRAL}$, se interpreta que en el terminal hay un valor lógico “1” ($RESET = 1$). Tal como se indica en la figura, la acción de *reset* tiene lugar cuando el terminal toma el valor lógico “0”. El circuito formado por la resistencia R y el condensador C tiene una constante de tiempo $\tau = RC$.

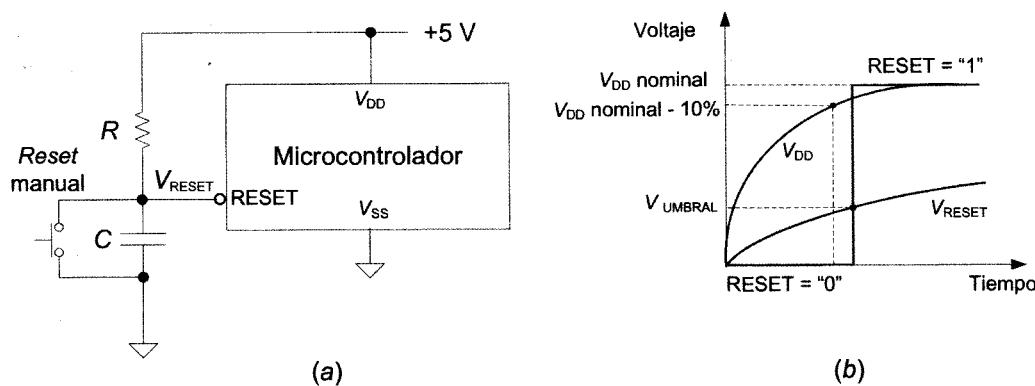


Figura 1.4 Reset manual y reset por encendido. (a) Circuito de reset típico en un microcontrolador. (b) Comportamiento temporal de las tensiones.

Cuando se pone en marcha el sistema, la tensión de alimentación de 5 V va cargando C a través de R . Si τ es suficientemente grande, V_{RESET} es inferior a V_{UMBRAL} durante el tiempo que la tensión de alimentación tarda en alcanzar el valor adecuado para que el microcontrolador trabaje de forma confiable.

En un microcontrolador hay otras posibles fuentes de *reset* como son el *reset por fallo de alimentación* (*power-glitch reset*, *brown-out reset*) y el *reset por desbordamiento del perro guardián*.

El *reset* por fallo de alimentación se produce cuando la tensión de alimentación cae momentáneamente por debajo de un cierto valor umbral, de modo que C se descarga parcialmente y se alcanza la condición $V_{\text{RESET}} < V_{\text{UMBRAL}}$.

El *reset* por desbordamiento del perro guardián se produce cuando por alguna razón no se ha refrescado (borrado) a tiempo el perro guardián. Generalmente ello significa que el microcontrolador se ha “perdido” en la ejecución del programa, es decir, se ha salido de la secuencia correcta.

Es muy importante que el microcontrolador efectúe una acción de *reset* cuando se producen fallos de este tipo. Ello garantiza que en situaciones como éstas, en las que el microcontrolador perdería inexorablemente la secuencia correcta de instrucciones del programa, no salte a una dirección aleatoria sino que vaya a una dirección determinada (la del vector de *reset*) donde se pueden efectuar acciones que contrarresten el fallo que se ha producido.

Aunque en un microcomputador general puede que no sea necesario, en un microcontrolador es muy importante y conveniente que pueda reaccionar con una acción de *reset* ante los posibles fallos antes mencionados y además que deje constancia de cuál fue la causa del *reset*.

En algunos microcontroladores (como los PIC), se señala la causa del *reset* mediante algunos bits de un registro del microcontrolador. Entonces, en el programa se puede indagar sobre la causa del *reset* y realizar las acciones correspondientes.

1.2.3 Bajo consumo

Un parámetro importante en los microcontroladores es su consumo de corriente. Hay un gran número de aplicaciones (por ejemplo, la telefonía móvil) que requieren que el dispositivo sea alimentado mediante baterías. Un bajo consumo de corriente conlleva una mayor duración de la batería.

El consumo de corriente en un circuito integrado depende fundamentalmente de tres factores: la tecnología de fabricación, la frecuencia del oscilador y la tensión de alimentación. En la fabricación de los microcontroladores se

usa tecnología CMOS precisamente por el bajo nivel de corriente que necesita para funcionar. En condiciones estáticas (frecuencia cero), las puertas CMOS prácticamente no consumen corriente. Cuando la puerta permanece estática en uno de los estados lógicos "0" ó "1", la corriente que circula por ella es en teoría igual a cero y en la práctica es muy pequeña y se debe a la corriente de fuga de los transistores. El consumo de corriente significativo tiene lugar sólo durante la conmutación entre los niveles lógicos. Por lo tanto, dado que al aumentar la frecuencia del oscilador aumenta la frecuencia de las conmutaciones, aumenta también el consumo de corriente del dispositivo.

Muchas veces la tarea que debe realizar el microcontrolador consiste fundamentalmente en esperar la ocurrencia de un evento externo, como la pulsación de una tecla, una interrupción, etc. y entonces realizar una determinada tarea y volver a esperar la ocurrencia de un nuevo evento externo. Mientras dura la espera conviene paralizar total o parcialmente al microcontrolador para disminuir así su consumo de corriente.

Una forma práctica de paralizar al microcontrolador es detener el oscilador principal y hacer que los diferentes bloques del dispositivo permanezcan en un estado estático, a la espera de que ocurra la acción externa que saque al microcontrolador de ese "letargo". Cuando el microcontrolador se ha paralizado total o parcialmente para disminuir su consumo de corriente, se dice que ha entrado a un *modo de consumo reducido o de bajo consumo* (*power down, idle, sleep mode*). La forma de entrar en el modo de consumo reducido varía según el microcontrolador. En algunos se modifica algún bit de uno de sus registros, que se dedica a este propósito; otros tienen una instrucción especial para este fin dentro de su repertorio. Normalmente el microcontrolador sale del estado de bajo consumo cuando hay una interrupción externa o un *reset*.

Ejemplo 1.1

En los microcontroladores de la familia del 8051, hay dos modos de bajo consumo denominados (en inglés) *idle* y *power down*. La entrada a cualquiera de estos modos se realiza al poner a 1 determinados bits del registro PCON (Power Control) del microcontrolador.

En el modo *idle*, se paraliza la CPU pero continúan funcionando el oscilador principal y los restantes bloques del microcontrolador. Se sale de este modo por cualquier interrupción interna o externa o por *reset*.

En el modo *power down*, el oscilador, y con él todo el microcontrolador, se paraliza. Se sale de este modo solamente por *reset*.

1.2.4 Protección del programa frente a copias

La protección de la información almacenada permanentemente en la memoria del microcontrolador es un aspecto muy importante que tienen en cuenta los fabricantes de microcontroladores. En general, cuando se desarrolla una aplicación, se desea impedir que el programa pueda ser copiado o extraído de la memoria del dispositivo, una vez que el dispositivo ha sido programado.

Los microcontroladores disponen de recursos para proteger, en general de forma opcional, el programa almacenado en la memoria. En algunos microcontroladores, como los PIC, se puede configurar el dispositivo para que la memoria de programa no se pueda leer una vez haya sido programada. En otros microcontroladores, particularmente aquellos que tienen una arquitectura de memoria abierta, es decir, que admiten la adición de memoria de programa externa al dispositivo, y donde por consiguiente el programa puede ser visible y copiado, lo que se hace es encriptar la información que se intercambia con la memoria externa, pudiéndose hacer esto con diferentes niveles de seguridad. Este es el caso, por ejemplo, de los microcontroladores de la familia del 8051.

Ejemplo 1.2

Protección del programa almacenado en la memoria del microcontrolador 8051. Los microcontroladores de la familia del 8051 poseen una arquitectura de memoria abierta, en el sentido de que es posible añadir memoria externa al microcontrolador. Se pueden usar dos niveles de protección:

Nivel 1. La información almacenada en la memoria se encripta con una cadena de encriptación formada por entre 16 y 64 bytes que se programan, mediante una operación NOR exclusivo (XNOR) entre la cadena y los contenidos que se desea almacenar en la memoria. Cuando la CPU lee el contenido de la memoria, se hace una operación XNOR entre el byte direccionado y uno de los bytes de encriptación, y esto recupera el valor original desencriptado. Si no se conoce la cadena de encriptación, resulta prácticamente imposible conocer la información real almacenada en la memoria interna o externa.

Nivel 2. Se impide el acceso total o parcial a la memoria de programa interna, mediante la programación de algunos bits de seguridad disponibles en un registro del dispositivo.

1.3 Arquitecturas von Neumann y Harvard

En la memoria de un ordenador, un microcomputador o un microcontrolador, se almacenan instrucciones y datos. Las instrucciones deben pasar secuencialmente a la CPU para su descodificación y ejecución, en tanto que algunos datos en memoria son leídos por la CPU y otros son escritos en la memoria desde la CPU. Puede intuirse que la organización de la memoria

y su comunicación con la CPU son dos aspectos que influyen en el nivel de prestaciones del ordenador.

Las arquitecturas von Neumann y Harvard son modelos generales del *hardware* de los ordenadores que representan dos soluciones diferentes al problema de la conexión de la CPU con la memoria y a la organización de la memoria como almacén de instrucciones y datos.

La arquitectura von Neumann toma el nombre del matemático John von Neumann que propuso la idea de un ordenador con el programa almacenado (*stored-program computer*). J. von Neumann trabajó en el equipo de diseñadores de la computadora ENIAC (*Electronic Numerical Integrator and Calculator*) diseñada en la Universidad de Pennsylvania durante la Segunda Guerra Mundial.

El término arquitectura Harvard se debe al nombre del lugar donde Howard Aiken diseñó los ordenadores Mark I, II, III y IV. Estos ordenadores fueron los primeros en utilizar memorias separadas para instrucciones y datos, una concepción diferente al ordenador de programa almacenado.

La figura 1.5 muestra estos dos modelos de ordenadores. La arquitectura von Neumann utiliza una memoria única para instrucciones y datos. Esto significa que con un mismo bus de direcciones se localizan (direccinan) instrucciones y datos y que por un único bus de datos transitan tanto instrucciones como datos. La misma señal de control que emite la CPU para leer un dato, sirve para leer una instrucción. No hay señales de control diferentes para datos e instrucciones. Debe quedar claro que aunque se use memoria ROM para almacenar el programa y RAM para los datos, para la CPU no hay tal distinción, sino que ROM y RAM forman un conjunto único (una memoria de lectura y escritura) para el cual la CPU emite señales de control, de dirección y de datos:

La arquitectura Harvard utiliza memorias separadas para instrucciones y datos. En este caso la memoria de programa (que almacena instrucciones) tiene su bus de direcciones (de instrucciones), su propio bus de datos (más bien es un bus de instrucciones) y su bus de control. Por otra parte, la memoria de datos tiene sus propios buses de direcciones, datos y control, independientes de los buses de la memoria de programa. La memoria de programa es sólo de lectura, mientras que en la de datos se puede leer y escribir.

La arquitectura von Neumann requiere menos líneas que la Harvard para conectar la CPU con la memoria, lo cual significa una conexión más simple entre ambas. Pero con esta arquitectura es imposible manipular si-

multáneamente datos e instrucciones, debido a la estructura de buses únicos, algo que sí es posible en la arquitectura Harvard, que tiene buses separados. Esto confiere a la arquitectura Harvard la ventaja de una mayor velocidad de ejecución de los programas.

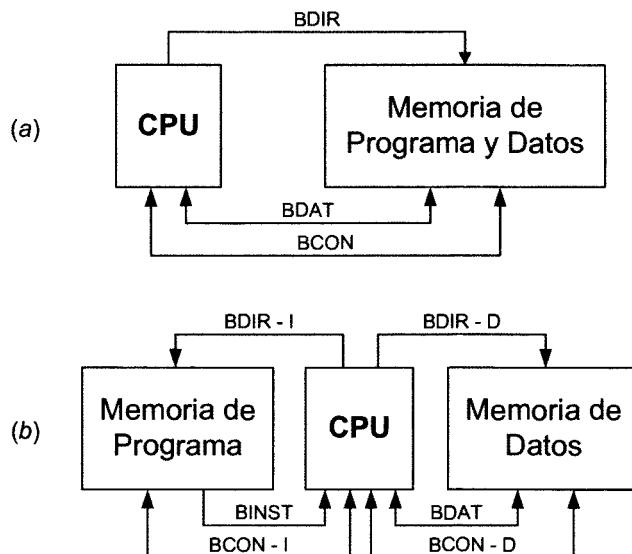


Figura 1.5 Arquitecturas (a) von Neumann y (b) Harvard. La arquitectura von Neumann utiliza una memoria única que se conecta a la CPU mediante los buses de direcciones (BDIR), datos (BDAT) y control (BCON). La arquitectura Harvard utiliza memorias separadas para instrucciones y datos, las cuales se conectan a la CPU mediante los buses de direcciones de instrucciones (BDIR-I) y de direcciones de datos (BDIR-D), los buses de instrucciones (BINST) y de datos (BDAT) y los buses de control de instrucciones (BCON-I) y de datos (BCON-D).

En los microcomputadores, la CPU es un circuito integrado: el microprocesador. Es obvio que la arquitectura von Neumann requiere menos terminales en el microprocesador que la arquitectura Harvard. Esta fue una razón decisiva para que desde sus inicios los microcomputadores basados en un microprocesador se hayan diseñado utilizando casi exclusivamente la arquitectura von Neumann. En los microcontroladores la situación es diferente. Al estar todos los componentes del sistema dentro del circuito integrado, desaparece la necesidad de minimizar el número de terminales de la CPU, de modo que en ellos ha predominado la arquitectura Harvard. Los microcontroladores PIC son un ejemplo de sistemas con arquitectura Harvard.

1.4 Arquitecturas CISC y RISC

CISC (*Complex Instruction Set Computer*) y RISC (*Reduced Instruction Set Computer*) son dos modelos generales de ordenadores, desde el punto de vista de la concepción de su repertorio de instrucciones, lo cual repercute directamente sobre la arquitectura de la CPU. Un ordenador CISC tiene un repertorio de instrucciones complejo y un ordenador RISC tiene un repertorio de instrucciones reducido.

Al aparecer los microprocesadores y los microcontroladores, la tendencia inicial fue proveerlos de un repertorio de instrucciones lo más potente posible, de modo que el modelo predominante fue el CISC. La complejidad de las instrucciones fue en aumento; en un mismo repertorio había instrucciones que hacían operaciones muy simples, como por ejemplo mover un dato desde la memoria al acumulador, junto a otras que efectuaban operaciones tan complejas como mover una cadena de datos de un lugar a otro en la memoria. Las instrucciones tenían diferente longitud y los modos de direccionamiento se hicieron cada vez más elaborados. Este aumento en la complejidad de las instrucciones se reflejó, por supuesto, en la complejidad del hardware de la CPU, en el que se hacía necesario dedicar un gran espacio del circuito integrado a la descodificación y ejecución de las instrucciones.

En la arquitectura RISC, la CPU dispone de un repertorio corto de instrucciones sencillas. Cada instrucción puede realizar una operación muy simple, como mover un dato entre la CPU y la memoria, pero a alta velocidad. Se puede lograr que todas las instrucciones tengan la misma longitud. Hay pocos modos de direccionamiento de los datos y son aplicables a todas las celdas de la memoria de datos. La complejidad de la CPU disminuye, de modo que es fácil aumentar la frecuencia del oscilador de la CPU y con ello la velocidad de las instrucciones. Como tienen menos transistores, son más baratas de diseñar y producir. Desde mediados del octavo decenio del siglo XX, ésta ha sido la tendencia predominante en el diseño de microprocesadores y microcontroladores. Los microcontroladores PIC son un ejemplo de dispositivos con arquitectura RISC.

1.5 Microcontroladores, microprocesadores y sus fabricantes

Los microcontroladores de un mismo tipo forman una familia, que se caracteriza, en general, por tener la misma CPU y ejecutar el mismo repertorio de instrucciones. Es lo que se conoce como “núcleo” (*core*) del microcontrolador. Los miembros de una familia de microcontroladores tienen el mismo núcleo, pero se diferencian en la entrada y salida y en la memoria. Por

ejemplo, todos los microcontroladores de la familia del 8051 (MCS51) tienen una CPU semejante, con un grupo de registros comunes a todos los miembros de la familia, y ejecutan el mismo repertorio de instrucciones. En cambio, los distintos miembros de la familia tienen puertos de entrada y salida y memoria diferentes, en cantidad y tipo.

Los microprocesadores y microcontroladores se fabrican como circuitos integrados independientes (*stand-alone devices*), que contienen exclusivamente al microcontrolador o microprocesador. Pero también el núcleo del procesador puede estar embebido (*embedded-processor core*) en un circuito integrado de alta escala de integración, cuya función es, en general, configurable por el usuario. Tal es el caso de los denominados Dispositivos Lógicos Programables (PLD: *Programmable Logic Devices*), entre los que están los FPGA (*Field Programmable Gate Array*). Los PLD en general y los FPGA en particular, son circuitos de alta escala de integración que disponen de un conjunto de elementos, cuya interconexión es programable por el usuario. Uno de estos elementos puede ser el núcleo de un microcontrolador o un microprocesador, cuya conexión a cierta cantidad de memoria y a dispositivos de entrada y salida disponibles en el PLD, es programable por el usuario. Así se puede configurar un microcontrolador “a la medida” de la necesidad de la aplicación, con la ventaja de ser compatible con un microprocesador o un microcontrolador “estándar”, como el 8051 o un PIC, pues tiene su mismo núcleo.

Hay un número elevado de compañías que fabrican microcontroladores y microprocesadores en alguna de las formas mencionadas anteriormente. A continuación se presenta una relación de fabricantes de microcontroladores y microprocesadores (o de dispositivos que utilizan la arquitectura de un determinado microcontrolador o microprocesador), con el correspondiente comentario acerca de los dispositivos que fabrica.

- Actel. FPGA con núcleos 8051 y ARM7.
- Advanced Micro Devices (AMD). Microprocesadores compatibles con xx86.
- Altera. FPGA con núcleos Nios II.
- Analog Devices. Arquitecturas para procesamiento digital de señales basadas en núcleos 8052, ARM7 y otros procesadores.
- Applied Micro Circuits Corp. (AMCC). Arquitecturas basadas en el microprocesador PowerPC.
- ARC International. Arquitecturas basadas en los procesadores ARC 600, ARC 700, etc.

- ARM. Arquitecturas basadas en los núcleos de los procesadores ARM7, ARM9, ARM10, etc.
- Atmel. Arquitecturas basadas en Marc 4, AVR, 8051, ARM7, ARM9, ARM11, PowerPC y SPARC.
- Broadcom. Procesadores para redes de datos y comunicaciones con arquitectura MIPS.
- Cambridge Consultants. Arquitecturas basadas en los núcleos de los procesadores XAP1, XAP2 y XAP3.
- Cavium Networks. Arquitecturas basadas en MIPS.
- Cirrus Logic. Arquitecturas basadas en ARM.
- Cradle Technologies. Procesadores digitales de señales: CT3400 y CT3600.
- Cyan Technology. Microcontrolador eCOG1k.
- Cybernetic Micro Systems. ASICs con microcontrolador P-51.
- Cypress Microsystems. Dispositivos con arquitectura PSoC (*Programmable System-on-Chip*).
- Dallas Semiconductor. Microcontroladores compatibles con 8051.
- EM Microelectronics. Microcontroladores EM6812 de muy bajo consumo.
- Freescale Semiconductor (procede de Motorola). Microcontroladores de las familias 68HC05, 68HC08, 68HC11, 68HC12 y 68HC16. Procesadores digitales de señales (DSP: *Digital Signal Processor*). Procesadores ColdFire y PowerQuicc con núcleo PowerPC.
- Fujitsu Microelectronics America. Microcontroladores FR80, MB9140x, F2MC-8FX, etc.
- Goal Semiconductor. Arquitecturas basadas en 8051.
- Hyperstone. Procesadores digitales de señales E1-32XSR/XSRU, HyNet32S, etc.
- Holtek Semiconductor. Microcontrolador HT8.
- Infineon Technologies (antes Siemens). Microcontroladores C500, C800, C166, TriCore, etc.
- Infrant Technologies. Microcontroladores para redes de datos.

- Integrated Device Technology (IDT). Procesadores para comunicaciones de datos basados en arquitectura MIPS.
- Intel. Microcontroladores de las familias MCS51, MCS151, MCS251, MCS96, MCS296, etc. Microprocesadores xx86, IXP4xx, etc.
- Microchip Technology. Microcontroladores PIC (PICmicro) y controladores digitales de señales dsPIC.
- MIPS Technologies. Procesadores MIPS (*Microprocessor without Interlocked Pipeline Stages*).
- National Semiconductor. Microcontroladores COP8, CR16 y microprocesadores NS32000.
- NEC Electronics America. Microcontroladores 78K0, V850 y otros.
- NetSilicon. Procesadores basados en núcleos ARM7 y ARM9.
- NXP (antes Philips Semiconductors). Microcontroladores con núcleo 8051, ARM7 y ARM9.
- Oki Semiconductor. Microcontroladores con núcleo ARM.
- PMC-Sierra. Procesadores basados en MIPS.
- Rabbit Semiconductor. Procesadores Rabbit 2000 y 3000.
- Renesas Tech. Corp. (antes Hitachi). Microcontroladores R8, H8 y otros.
- Sharp Microelectronics. Microcontroladores BlueStreak con núcleo ARM7 y ARM9.
- Silicon Laboratories. Microcontroladores con núcleo 8051.
- Silicon Storage Technology. Microcontroladores con núcleo 8051.
- STMicroelectronics. Microcontroladores con núcleos 8051 y ARM7.
- Texas Instruments (TI). Procesadores digitales de señales TMS370 y TMS470, microcontroladores MSP430.
- Toshiba America Electronic Components. Microcontroladores CISC y RISC.
- UbiCom. Microcontroladores SX, IP2000 e IP3000.
- Xemics. Microcontroladores con núcleo CoolRISC.
- Xilinx. FPGA con núcleos PowerPC.
- ZiLOG. Microcontroladores de 8 bits con arquitecturas Z8 y Z80.

2 Los microcontroladores PIC

Este capítulo da una visión panorámica de los microcontroladores PIC. Primeramente se estudian las características generales de su arquitectura, que son comunes a la mayoría de los miembros de las diferentes familias PIC. Para ello se analizan varios elementos de la arquitectura, basada en el uso del registro de trabajo W. En particular, se abordan los siguientes tópicos: la ejecución segmentada de las instrucciones, los tipos de osciladores disponibles, los bits de configuración de los dispositivos, el *reset* y sus fuentes, el modo de bajo consumo y el perro guardián. Finalmente se exponen las características de las diferentes familias de microcontroladores PIC existentes en el mercado en el momento de escribir este libro.

2.1 Características generales de los microcontroladores PIC

La arquitectura de los PIC responde al esquema de bloques de la figura 1.2. Todos están basados en la arquitectura Harvard, con memorias de programa y de datos separadas. Como en la mayoría de los microcontroladores, la memoria de programa es mucho mayor que la de datos. La memoria de programa está organizada en palabras de 12, 14 ó 16 bits mientras que la memoria de datos está compuesta por registros de 8 bits. El acceso a los diversos dispositivos de entrada y salida se realiza a través de algunos registros de la memoria de datos, denominados registros de funciones especiales (SFR: *Special Function Registers*). Muchos microcontroladores PIC cuentan con una cierta cantidad de memoria EEPROM para el almacenamiento no volátil de datos.

Por otra parte, todos los PIC son microcontroladores RISC que cuentan con un pequeño número de instrucciones: entre 33 y 77. Todas las instrucciones son del mismo tamaño: una palabra de 12, 14 ó 16 bits. Desde el punto de vista del programador, el modelo general de los microcontroladores PIC consta de un registro de trabajo (registro W) y los registros de la memoria de datos. Para las operaciones aritméticas y lógicas, uno de los operandos debe estar en el registro W y el resultado se obtiene en W o en cualquier registro de la memoria de datos. Las transferencias de datos se realizan entre algún registro de la memoria de datos y el registro W, aunque en los PIC de la gama alta se permiten transferencias directas entre dos registros de la memoria de datos, sin necesidad de pasar por el registro W. Se dispone de instrucciones para acceder a cualquier bit de cualquier registro de la memoria de datos.

Todos los microcontroladores PIC aplican la técnica del segmentado (*pipeline*) en la ejecución de las instrucciones, en dos etapas, de modo que las ins-

trucciones se ejecutan en un único ciclo de instrucción, equivalente a cuatro pulsos del oscilador principal del microcontrolador, excepto las instrucciones de transferencia de control que toman dos ciclos de instrucción.

Otra característica común a los microcontroladores PIC es la forma en que está implementada la pila. La pila no forma parte de la memoria de datos sino que ocupa un espacio independiente y tiene además una profundidad limitada, según el modelo de PIC. En estos microcontroladores no hay un puntero de pila (registro SP: *Stack Pointer*), tan común en la mayoría de los microprocesadores y en muchos microcontroladores.

Los microcontroladores PIC cuentan con una amplia gama de dispositivos de entrada y salida. Disponen de puertos paralelos de 8 bits, temporizadores, puertos serie sincrónicos y asincrónicos, convertidores A/D de aproximaciones sucesivas de 8, 10 ó 12 bits, convertidores D/A, moduladores de ancho de pulso (PWM: *Pulse Width Modulation*), etc. Excepto en los PIC de gama baja, que no disponen de un sistema de interrupciones, los dispositivos de entrada y salida generan solicitudes de interrupción al microcontrolador, que se pueden enmascarar individualmente.

Todos los microcontroladores PIC cuentan con un temporizador que trabaja como perro guardián y tienen un cierto número de bits para configurar el dispositivo, a los que se accede al programar el microcontrolador. Mediante alguno de los bits de configuración, se puede proteger la memoria de programa frente a copias no autorizadas.

Muchos microcontroladores PIC pueden ser programados en el propio circuito de la aplicación (ICSP: *In-Circuit Serial Programming*), utilizando un pequeño número de líneas.

2.1.1 La Unidad Aritmética y Lógica y el registro W en los microcontroladores PIC

Uno de los componentes fundamentales de la CPU de un microcontrolador es la unidad aritmética y lógica (ALU: *Arithmetic and Logic Unit*). Tal como indica su nombre, la ALU realiza las operaciones aritméticas y lógicas previstas en el repertorio de instrucciones del microcontrolador. La ALU tiene asociado un registro que almacena temporalmente uno de los datos que intervienen en la operación de la ALU y eventualmente el resultado de la operación realizada. También se asocian a la ALU algunos bits que indican determinadas características del resultado de la operación (si el resultado es cero, si se ha producido acarreo o préstamo, si el resultado es positivo o negativo, etc.). Estos bits indicadores usualmente forman parte del llamado registro de estado (STATUS).

En muchos microprocesadores y microcontroladores, el registro asociado a la ALU recibe el nombre de Acumulador (ACC: *Accumulator*). En los microcontroladores PIC se denomina Registro de Trabajo (W: *Working Register*) y hace funciones semejantes al Acumulador de los microprocesadores y microcontroladores tradicionales, pero su posición respecto a la ALU es distinta a la que tiene el ACC, según muestra la figura 2.1, y por lo tanto los registros ACC y W no se comportan de forma exactamente igual.

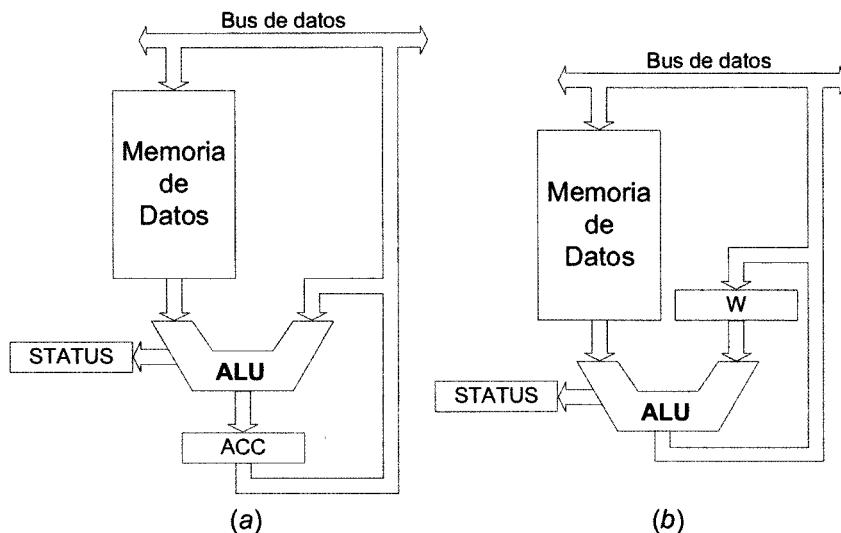


Figura 2.1 La unidad aritmética y lógica (ALU) y su relación con el registro de trabajo y la memoria de datos. (a) Estructura tradicional empleada en muchos microprocesadores. (b) Estructura empleada en los microcontroladores PIC. La diferencia fundamental entre ambas estructuras es la ubicación del registro de trabajo, que en las arquitecturas tradicionales recibe el nombre de Acumulador (ACC) y en los PIC es el registro W.

En las arquitecturas tradicionales, el ACC está a la salida de la ALU, de modo que el resultado de cualquier operación aritmética o lógica siempre es depositado en el ACC. En los PIC, en cambio, el resultado de una operación aritmética o lógica puede depositarse en W o puede llevarse directamente a cualquier registro de la memoria de datos, y esto les proporciona gran flexibilidad y potencia.

2.1.2 Ciclos de máquina y ejecución de instrucciones

Como todos los microcontroladores, los PIC tienen un oscilador principal que dicta la cadencia de sus operaciones internas. Los pulsos generados por este oscilador (OSC1) son divididos internamente para generar cuatro señales denominadas Q1, Q2, Q3 y Q4, que sincronizan todo el trabajo inter-

no del microcontrolador. Cada cuatro pulsos del oscilador principal se tiene un ciclo de máquina (CM). La figura 2.2 muestra la relación entre el oscilador principal, las señales internas y la duración de los CM.

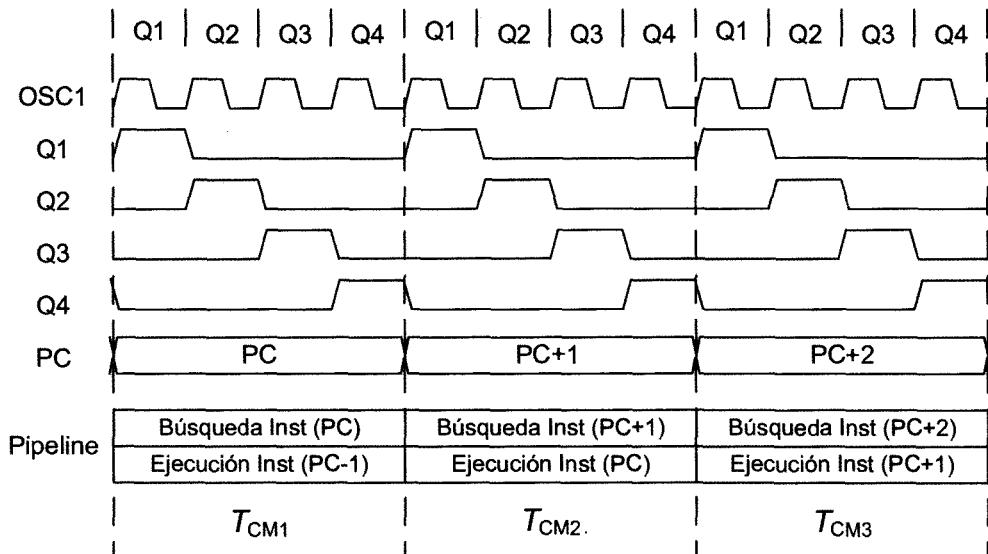


Figura 2.2 Señales de reloj en los microcontroladores PIC. OSC1 es la señal del oscilador principal, de la cual se derivan las señales internas Q1, Q2, Q3 y Q4, que sincronizan la búsqueda, descodificación y ejecución de las instrucciones. T_{CM} es el tiempo que dura un ciclo de máquina, y equivale a cuatro pulsos del oscilador principal.

Durante el tiempo Q1 de cada ciclo de máquina, el Contador de Programa (PC) se incrementa, apuntando hacia la instrucción que debe ser buscada, lo cual ocurre durante el tiempo Q4. Paralelamente, durante todo el CM (desde Q1 a Q4) se está ejecutando la instrucción anterior.

La ejecución de una instrucción cualquiera se realiza en tres fases: *búsqueda, descodificación y ejecución* propiamente dicha. En la fase de búsqueda (*fetch*), el microcontrolador lee la instrucción que está en la memoria de programa y la lleva a la CPU. En la fase de descodificación, la CPU determina cuál es la operación indicada en la instrucción. En la fase de ejecución, se ejecuta la operación prevista por la instrucción.

La ejecución de una instrucción se realiza en dos ciclos de máquina. En el primer CM se busca la instrucción en la memoria de programa y en el segundo CM se descodifica y ejecuta la instrucción. En realidad, debido al mecanismo de segmentado (*pipeline*) (apartado 2.1.3), ese segundo CM se

solapa, en el tiempo, con el primer CM de la siguiente instrucción, según se puede apreciar en la figura 2.2, resultando que, en promedio, se ejecute una instrucción por ciclo de máquina.

2.1.3 Segmentado (pipeline) en la ejecución de instrucciones

El segmentado o *pipeline* es una técnica mediante la cual se consigue que dos o más instrucciones se solapen durante su ejecución. Esto introduce un cierto nivel de paralelismo en la ejecución de las instrucciones y reduce el tiempo promedio de su ejecución. El segmentado es transparente al programador.

El principio en que se basa el segmentado es similar al de una línea de producción en cadena, como las utilizadas para producir automóviles, televisores y muchos otros artículos de consumo. La producción transita por una serie de etapas, en cada una de las cuales se realiza alguna operación que contribuye a obtener el producto final. En una línea de producción en cadena de n etapas, hay simultáneamente n productos en proceso de fabricación. Los productos permanecen un cierto tiempo T_e en cada una de las etapas. Cada producto transita por las n etapas de la línea, lo cual ocupa un tiempo $n \times T_e$. Como cada T_e segundos sale un producto diferente, el tiempo *promedio* de fabricación de un artículo es T_e .

En el caso de un *pipeline* de instrucciones que tenga n etapas, cada instrucción permanece un tiempo T_{CM} en cada etapa y el tiempo que le toma transitar por todas las etapas es $n \times T_{CM}$ segundos. Cada T_{CM} segundos “sale” una instrucción del *pipeline*, por lo que el tiempo promedio de ejecución de una instrucción cualquiera será de T_{CM} segundos. T_{CM} es el tiempo que dura un ciclo de máquina. Dado que hay instrucciones, como las de transferencia de control, que requieren ciclos de máquina adicionales en el *pipeline*, en la práctica el tiempo promedio de ejecución de una instrucción resulta algo mayor que T_{CM} .

En los microcontroladores PIC, las instrucciones se ejecutan a través de un *pipeline* de dos etapas, según muestra la figura 2.3. Cada instrucción se ejecuta en dos etapas: la primera etapa corresponde a la búsqueda, para lo cual se requiere el tiempo correspondiente a un CM; en la segunda etapa se descodifica y ejecuta la instrucción, lo cual toma otro ciclo de máquina, mientras se busca la instrucción siguiente. Es decir, que en cada ciclo de máquina se busca una instrucción y a la vez se ejecuta la instrucción anterior. Entonces, con cada nuevo CM se ejecuta completamente una nueva instrucción, excep-

to cuando haya instrucciones de transferencia de control, que requieren dos ciclos de máquina, como ilustra el ejemplo 2.1.

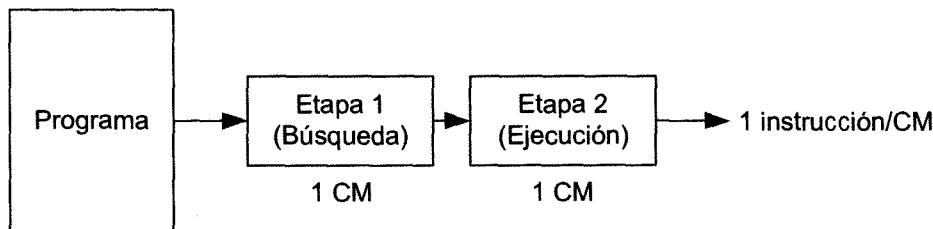


Figura 2.3 Pipeline de dos etapas. En la primera etapa se busca la instrucción, que se ejecuta en la segunda etapa. Cada etapa dura un ciclo de máquina (CM). En el pipeline hay simultáneamente dos instrucciones en etapas diferentes. Cada instrucción permanece en el pipeline durante dos CM, pero como promedio, sale del pipeline una instrucción por cada CM.

Ejemplo 2.1

La figura 2.4 muestra la operación de un *pipeline* de dos etapas, al ejecutar un segmento de programa que incluye una instrucción de transferencia de control.

Durante el primer ciclo de máquina (CM), se busca la instrucción I1, mientras se ejecuta la anterior instrucción (I0, no mostrada). En el segundo CM se busca la instrucción I2 mientras se ejecuta I1. En el tercer CM se busca la instrucción de transferencia de control I3 mientras se ejecuta I2. En el cuarto CM se busca la instrucción I4, mientras se ejecuta la instrucción I3. La ejecución de I3, que es una instrucción de llamada a una subrutina que comienza en la instrucción I10, consiste en poner el contador de programa en la pila y colocar en el PC la dirección de la instrucción I10, que será buscada en el siguiente CM. En el quinto CM se busca la instrucción I10, pero la I4, que ya está en el *pipeline*, no debe ser ejecutada y por lo tanto se saca virtualmente del *pipeline*, ejecutando en su lugar una instrucción de "nop" (no operación). En el sexto CM se busca la instrucción I11 de la subrutina (no mostrada en la figura) y se ejecuta I10. La instrucción de transferencia de control toma, pues, dos CM.

2.1.4 Osciladores

Los microcontroladores PIC disponen de las siguientes opciones para el oscilador principal: oscilador de cristal (de cuarzo), oscilador RC y oscilador externo. Algunos dispositivos disponen de un oscilador RC interno de unos 4 MHz. Al aumentar la frecuencia del oscilador principal, se acorta la duración de los ciclos de máquina y con ello el tiempo de ejecución de las instrucciones, pero aumenta el consumo de energía.

El tipo de oscilador a utilizar se selecciona mediante los bits de configuración del PIC. En estos bits se especifican varias configuraciones de osciladores (LP, XT, HS) de cristal de cuarzo o resonador cerámico. La opción LP se selecciona en aplicaciones de muy baja potencia, en el rango de frecuencias de

32 kHz a 200 kHz, aproximadamente; la opción XT se ajusta a las aplicaciones a frecuencias medias, de 100 kHz a 4 MHz; y la opción HS es apropiada en aplicaciones de alta frecuencia, entre 8 MHz y 20 MHz. La figura 2.5 muestra una configuración típica del oscilador de cristal.

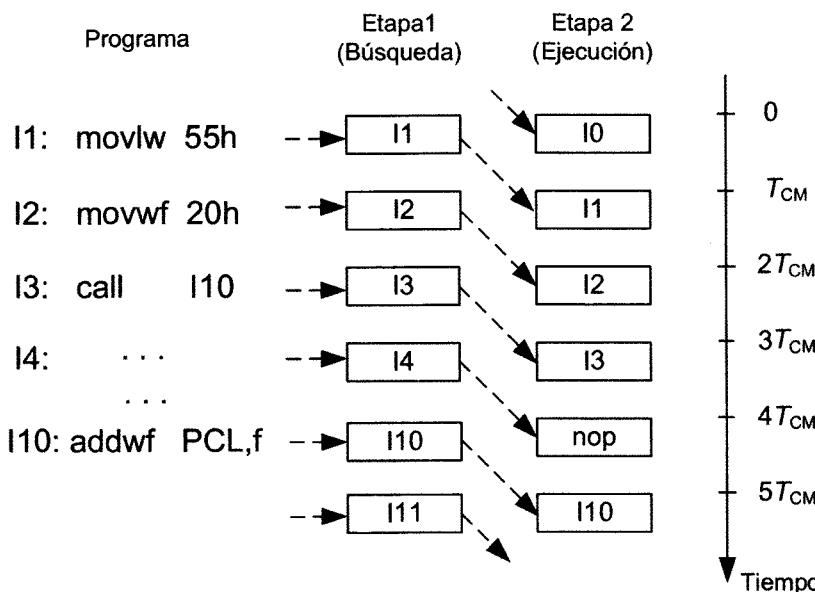


Figura 2.4 Ejemplo de flujo de instrucciones a través de un pipeline de 2 etapas. La instrucción I3 se ejecuta en dos ciclos de máquina (CM), lo cual ocurre con todas las instrucciones de transferencia de control. El resto de las instrucciones se ejecutan en un CM. T_{CM} es el tiempo que dura un CM.

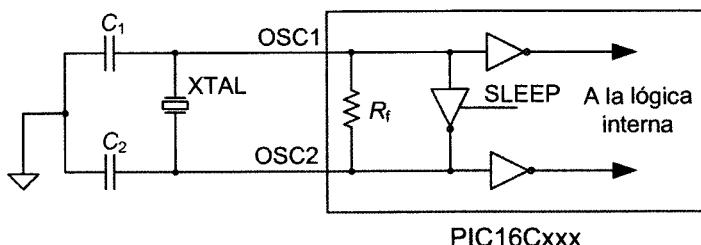


Figura 2.5 Oscilador de cristal. $C_1, C_2 = 15 \text{ pF a } 68 \text{ pF}$ para un cristal (XTAL) de 4 MHz.

El oscilador RC es una opción de bajo coste para el oscilador principal del microcontrolador, apropiada cuando la precisión y la estabilidad en el valor de la frecuencia no son esenciales. La figura 2.6 muestra esta configuración en un PIC de gama media. El fabricante no ofrece fórmulas para determinar la frecuencia de oscilación en función de los valores de R_{EXT} y C_{EXT} . En

su lugar se ofrecen curvas para cada dispositivo que relacionan la frecuencia con V_{DD} , R_{EXT} y C_{EXT} , a una temperatura determinada (25 °C).

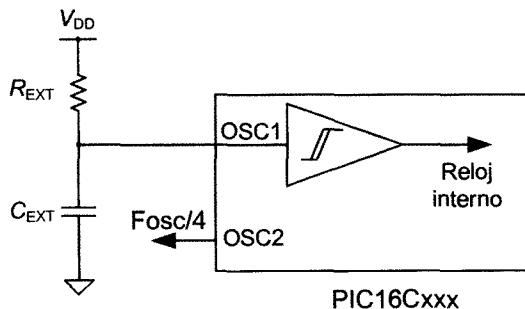


Figura 2.6 Oscilador RC. $R_{EXT} = 3 \text{ k}\Omega$ a $100 \text{ k}\Omega$; $C_{EXT} > 20 \text{ pF}$.

La tercera opción es utilizar un oscilador externo como fuente de reloj del microcontrolador. La figura 2.7 muestra esta configuración en un PIC de gama media.

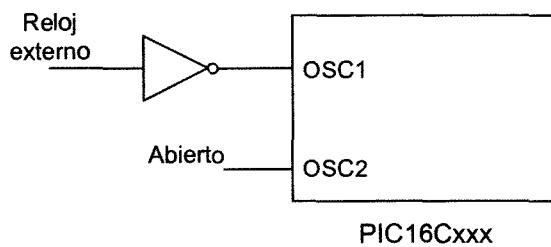


Figura 2.7 Uso de un oscilador externo como reloj en un PIC.

2.1.5 Bits de configuración

Todos los PIC disponen de un cierto número de bits para configurar el microcontrolador. Estos bits de configuración están disponibles en memoria no volátil (EEPROM) cuando se programa el dispositivo, pero no son accesibles durante el funcionamiento normal del microcontrolador. Es decir, una vez estos bits han tomado algún valor durante la programación del PIC, ya no pueden ser modificados por el programa de la aplicación.

Los bits de configuración permiten que el usuario programe ciertas características del dispositivo para adaptarlo mejor a las necesidades de la aplicación. Cuando se pone en marcha el dispositivo, el estado de estos bits determina la forma en que operará el microcontrolador. Aunque estos bits normalmente forman parte de la memoria de programa, ocupan una dirección que no es accesible durante el funcionamiento normal del microcontrolador. Sólo se puede acceder a ellos durante la programación del microcontrolador.

Las características que se programan en los bits de configuración son las siguientes:

- El tipo de oscilador.
- La habilitación o no del perro guardián.
- La protección de la memoria de programa.
- La protección de la memoria EEPROM de datos, si existe en el dispositivo.
- Las características del *reset* y la alimentación del dispositivo.

Según el dispositivo concreto, puede que alguna de estas características no sea programable. A continuación se presentan varios ejemplos que ilustran cómo se presentan esas características en los bits de configuración.

Ejemplo 2.2

Los bits de configuración en los microcontroladores de gama media se agrupan en una “palabra” de configuración que ocupa la dirección 2007h de la memoria de programa. Esta dirección sólo se alcanza durante la programación de los dispositivos, pero no durante la ejecución normal del programa de la aplicación. La figura 2.8 muestra, a modo de ejemplo, los bits de configuración en el microcontrolador PIC16F873.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP1	CP0	DEBUG	-	WRT	CPD	LVP	BODEN	CP1	CP0	PWRTE#	WDTE	FOSC1	FOSC0

CP1, CP0: Protección de la memoria FLASH de programa:

- 11 - memoria sin protección
- 10 - se protegen sólo las últimas 256 celdas (F00h a FFFh)
- 01 - se protege sólo la página 1 (800h a FFFh)
- 00 - toda la memoria se protege (000h a FFFh)

DEBUG: Modo de depuración en circuito:

- 1 - inhabilitado (RB6 y RB7 son terminales de propósito general)
- 0 - habilitado (RB6 y RB7 se usan para la depuración)

WRT: Escritura en la memoria FLASH vía EECON:

- 1 - habilitada
- 0 - inhabilitada

CPD: Protección de la memoria EEPROM de datos:

- 1 - sin protección
- 0 - protegida

LVP: Habilitación de la programación serie con bajo voltaje:

- 1 - habilitada
- 0 - inhabilitada

BODEN: Reset por fallo de alimentación:

- 1 - habilitada
- 0 - inhabilitada

PWRTE#: Habilitación del temporizador de arranque de encendido (PWRT):

- 1 - inhabilitado
- 0 - habilitado

WDTE: Habilitación del perro guardián (WDT):

- 1 - habilitado
- 0 - inhabilitado

FOSC1, FOS0: Selección del tipo de oscilador:

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

Figura 2.8 Bits de configuración del microcontrolador PIC16F873.

2.1.6 Fuentes de reset

En general, el *reset* de un microcontrolador hace que el dispositivo vaya a un estado conocido. Mientras se encuentra transitoriamente en estado de *reset*, el dispositivo está virtualmente detenido, es decir, no se ejecuta ninguna instrucción del programa. Cuando sale del estado de *reset*, el dispositivo va a un estado conocido. En particular, en los microcontroladores PIC, la salida del estado de *reset* hace que el contador de programa se ponga en el valor 0, de modo que la instrucción que se ejecuta después de un *reset* es la que está en esa dirección de la memoria de programa, independientemente de cualquier situación anterior al *reset*.

El *reset* en los microcontroladores PIC se puede producir por diferentes causas, denominadas fuentes de *reset*. El número de las fuentes de *reset* puede variar según el tipo de microcontrolador, pero las siguientes son comunes a casi todos los microcontroladores PIC:

1. *Reset* externo.
2. *Reset* por encendido (puesta en marcha).
3. *Reset* por desbordamiento del perro guardián.
4. *Reset* por fallo de la alimentación.

La figura 2.9 muestra el esquema lógico del circuito interno que produce el *reset* de los microcontroladores PIC y las posibles fuentes de *reset* enunciadas anteriormente. El circuito debe garantizar que, en el momento de abandonar el estado de *reset*, el microcontrolador se encuentre en condiciones estables que permitan su correcto funcionamiento. En particular, el circuito debe asegurar que la salida del estado de *reset* se produzca sólo si la fuente de alimentación ha alcanzado un valor estable y suficientemente alto. Es decir, el microcontrolador no debe salir del estado de *reset* mientras la tensión de alimentación no sobrepase cierto umbral. De ello se encarga el bloque asociado al terminal V_{DD} en la figura 2.9, que actúa durante el *reset* por encendido.

El circuito interno de *reset* también debe garantizar que el microcontrolador abandone el estado de *reset* sólo si el oscilador principal está en régimen estable. Esta es la función del bloque asociado al terminal OSC1 en la figura 2.9. Desde el momento en que arranca el oscilador principal del microcontrolador hasta que ese oscilador alcanza su operación estable, transcurre un cierto tiempo. Mientras la amplitud y frecuencia del oscilador no sean estables, el microcontrolador debe permanecer en el estado de *reset*. El oscilador principal arranca cuando se pone en marcha el microcontrolador, cuando éste

sale del modo de bajo consumo, o cuando hay un fallo en la alimentación del microcontrolador. Estas situaciones corresponden, respectivamente, al *reset* por encendido, al *reset* (externo o por desbordamiento del perro guardián) mientras el microcontrolador está en modo de bajo consumo, y al *reset* por fallo de la alimentación.

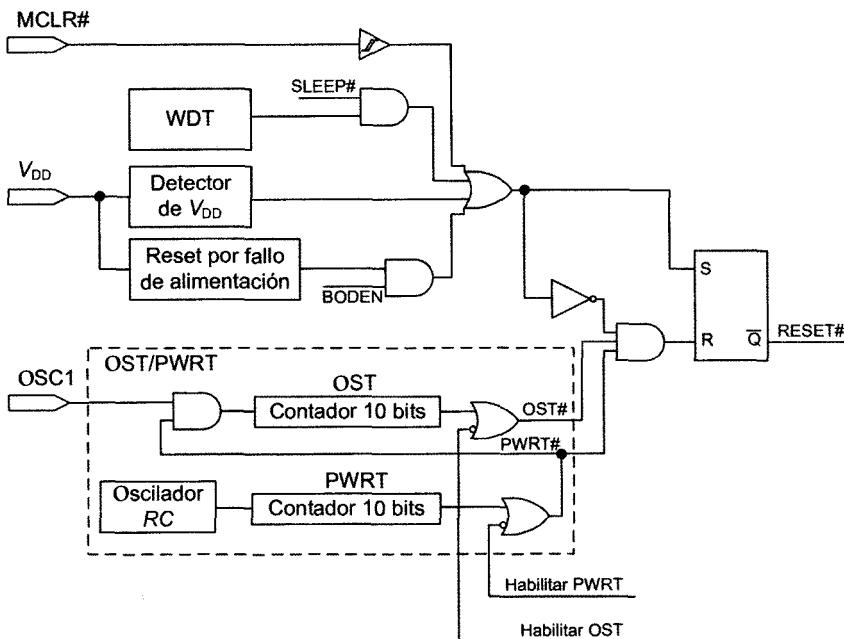


Figura 2.9 Diagrama de bloques simplificado del circuito interno que controla el *reset* en un microcontrolador PIC, que muestra las principales fuentes de *reset*. La operación de algunos de estos bloques se puede programar mediante los bits de configuración (BODEN, PWRTE, etc.).

El bloque OST/PWRT de la figura 2.9 contiene dos temporizadores: el Temporizador de Arranque del Oscilador (OST: *Oscillator Start-up Timer*) y el Temporizador de Arranque de Encendido (PWRT: *Power-up Timer*). Estos circuitos trabajan de la siguiente forma. Un oscilador RC interno, independiente del oscilador principal, garantiza un retardo de unos 72 ms para la señal PWRT, contados a partir del momento en que se alimenta el oscilador. El temporizador OST proporciona un retardo adicional de 1024 pulsos del oscilador principal, tiempo suficiente para que el oscilador alcance condiciones de operación estables. El temporizador OST comienza a trabajar sólo cuando el temporizador PWRT se ha desbordado.

La figura 2.10 muestra la secuencia en que se producen las señales asociadas al bloque OST/PWRT en dos situaciones diferentes: durante un *reset* por encendido (los terminales MCLR# y V_{DD} están unidos) y en un *reset* manual.

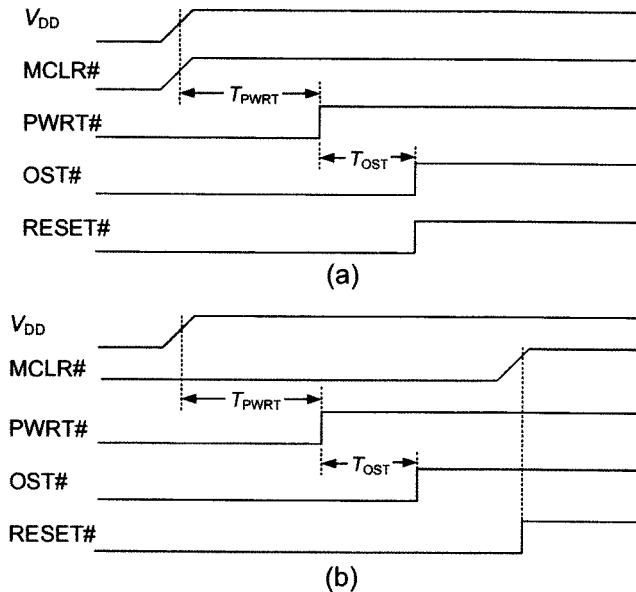


Figura 2.10 Diagramas de tiempo que ilustran la secuencia en que se producen las señales asociadas al bloque OST/PWRT. (a) Reset por encendido (el terminal $MCLR\#$ está conectado a V_{DD}). (b) Reset manual.

El *reset* externo ocurre cuando se pone a 0 el terminal $MCLR\#$. El terminal $MCLR\#$ debe estar a 1 durante el funcionamiento normal del microcontrolador. El *reset* externo puede ocurrir durante el funcionamiento normal del microcontrolador o mientras el microcontrolador está en modo de bajo consumo (modo SLEEP). En ambos casos, cuando el microcontrolador sale del estado de *reset* ejecuta la instrucción situada en la dirección de la memoria de programa. A este terminal se puede conectar un interruptor para producir un *reset* manual, utilizando un esquema como el de la figura 1.4a.

El *reset* por encendido (POR: Power-on Reset) ocurre si se conecta el terminal $MCLR\#$ al terminal V_{DD} de alimentación del microcontrolador, utilizando uno de los esquemas de la figura 2.11. El microcontrolador detecta la aparición del tensión de alimentación V_{DD} (flanco de subida en V_{DD}), provocando con ello un *reset* que garantiza el correcto inicio del trabajo del microcontrolador. De esta forma no es necesario colocar circuitos externos para conseguir el *reset*, basta con unir los terminales $MCLR\#$ y V_{DD} directamente o a través de una resistencia (figura 2.11a).

Si la fuente de alimentación tiene un tiempo de establecimiento elevado, hay que garantizar que la tensión en el terminal $MCLR\#$ permanezca por debajo del umbral hasta que V_{DD} alcance un valor adecuado. En estos casos se utiliza el circuito de la figura 2.11b.

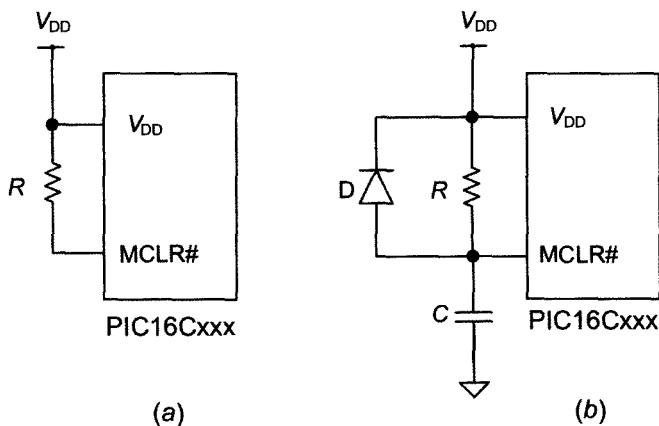


Figura 2.11 (a) Circuito para garantizar el *reset* por encendido: los terminales MCLR# y V_{DD} se pueden unir directamente o mediante una resistencia. (b) Circuito para el *reset* por encendido si la fuente de alimentación tiene un tiempo de establecimiento elevado.

El *reset* por desbordamiento del perro guardián ocurre cuando se desborda el temporizador denominado perro guardián (WDT: Watchdog Timer). El desbordamiento se produce si, desde el programa que está ejecutando el microcontrolador, no se borra a tiempo el resultado del conteo que realiza este temporizador. El *reset* por desbordamiento del perro de guardia puede ocurrir durante el funcionamiento normal del microcontrolador. Si el WDT se desborda mientras el microcontrolador está en el modo de bajo consumo, el microcontrolador sale de dicho modo, pero no se produce un *reset*. En el apartado 2.1.8 se estudia con más detalle el funcionamiento del WDT.

El *reset* por fallo de la alimentación (BOR: Brown-out Reset) ocurre cuando hay una disminución brusca y transitoria de la tensión de alimentación. El microcontrolador incluye un circuito que produce un *reset* en estas condiciones y mantiene dicho estado mientras la tensión de alimentación V_{DD} esté por debajo de un cierto umbral (figura 2.9). Cuando V_{DD} se recupera, el estado de *reset* se mantiene durante un tiempo adicional de unos 72 ms, proporcionado por el temporizador PWRT. De esta forma se garantiza que el oscilador principal y V_{DD} estén en sus valores nominales cuando se salga del estado de *reset*. La figura 2.12 muestra varias situaciones de *reset* por fallo de la alimentación.

Las características de algunas fuentes de *reset* se pueden programar mediante la palabra de configuración del dispositivo. Un detalle importante es que el programador puede determinar, si fuese necesario, la fuente del *reset* que ha actuado, consultando determinados bits de algunos registros de funciones especiales, tales como los registros STATUS y PCON.

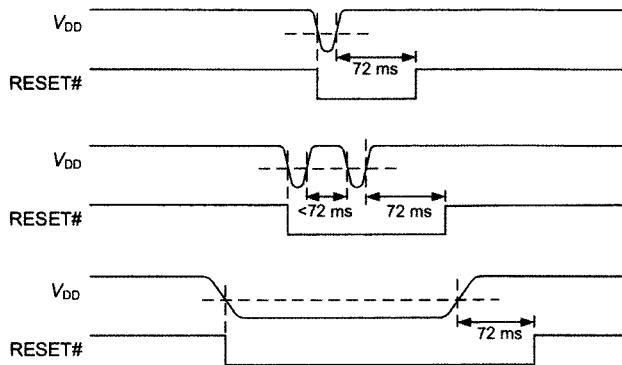


Figura 2.12 Comportamiento del reset por fallo de la alimentación en varias situaciones. El temporizador PWRT garantiza que V_{DD} esté en su valor nominal cuando se salga del estado de reset.

PCON							
7	6	5	4	3	2	1	0
-	-	-	-	-	-	POR#	BOR#

POR#: Indicador de reset por encendido (POR).

1 - no ha ocurrido reset por encendido.

0 - ha ocurrido un reset por encendido

BOR#: Indicador de reset por fallo de alimentación (BOR)

1 - no ha ocurrido un reset por fallo de alimentación

0 - ha ocurrido un reset por fallo de alimentación

Figura 2.13 El registro de funciones especiales PCON (Power Control) en el PIC16F873 tiene dos bits que permiten identificar el origen del reset que ha actuado.

Ejemplo 2.3

La figura 2.13 muestra el registro PCON de un microcontrolador de gama media: el PIC16F873. Se puede ver que tiene dos bits con los que se puede identificar si el reset ha tenido lugar por encendido o por fallo de alimentación. Según haya ocurrido un reset por encendido (POR) o por fallo de la alimentación (BOR), el bit correspondiente se activa (se pone a 0). Los bits POR# y BOR# deben ser puestos a 1 por programa después de que haya ocurrido el reset.

Por otra parte, en la figura 3.14 se muestra el registro STATUS, que dispone del bit TO# que se activa (se pone a 0) cuando se desborda el perro guardián.

2.1.7 Modo de bajo consumo

En el modo o estado de bajo consumo o de reposo (modo *sleep*), el microcontrolador suspende casi todas sus funciones, incluso el oscilador principal, que deja de funcionar. En esas condiciones, el microcontrolador consume muy poca corriente de la fuente de alimentación: menos de 1 μ A en algunos modelos.

En el modo de bajo consumo se entra con la instrucción sleep. Mientras el microcontrolador permanece en este modo, los valores almacenados en los registros de la memoria de datos no se alteran. Dado que cuando se ejecuta la instrucción sleep, la instrucción que le sigue ya ha entrado al *pipeline* de la CPU, cuando el microcontrolador despierta, se ejecuta esa instrucción. Por ello se recomienda colocar una instrucción nop (no operación) a continuación de sleep. La instrucción sleep también borra (pone en 0) el contador del perro guardián.

Se sale o se “despierta” del modo de bajo consumo cuando ocurre alguno de estos tres eventos:

- Un *reset*.
- Un desbordamiento del perro guardián (si está habilitado).
- Una interrupción externa o procedente de alguno de los módulos periféricos internos.

Si se despierta por *reset*, el microcontrolador va directo a ejecutar la instrucción que está en la dirección 0 de la memoria de programa.

Si se despierta por desbordamiento del perro guardián, continúa la ejecución del programa con la instrucción que sigue a la instrucción sleep.

Si se ha despertado por interrupción y el sistema de interrupción está habilitado, es decir, el bit INTCON<7> es 1 (bit 7 del registro de funciones especiales INTCON, denominado GIE: *Global Interrupt Enable bit*), se ejecuta la instrucción que sigue a la instrucción sleep y se salta a la dirección 4 de la memoria de programa en busca de la rutina de atención a la interrupción.

Si ocurre una interrupción mientras el microcontrolador está en el modo de bajo consumo y las interrupciones no están habilitadas (bit GIE = 0), el microcontrolador despierta, ejecuta la instrucción que sigue a la instrucción sleep y continúa la secuencia de instrucciones del programa, pero no salta a la dirección 4 de la memoria de programa.

2.1.8 Perro guardián

El perro guardián (WDT: *Watchdog Timer*) está realizado mediante un oscilador independiente del oscilador principal del microcontrolador, de modo que funciona incluso durante el modo de bajo consumo, y un contador de los pulsos que produce ese oscilador independiente. Si el contador se desborda mientras el microcontrolador está operando normalmente, es decir, no en modo de bajo consumo, se genera un *reset* al microcontrolador. Si el desbordamiento ocurre mientras el microcontrolador está en el modo de bajo consumo, el microcontrolador despierta y ejecuta la instrucción que está a continuación de la instrucción sleep.

La figura 2.14 muestra el diagrama de bloques del perro guardián en un PIC de gama media. El perro guardián se puede habilitar mediante el bit WDTE de la palabra de configuración. Una vez habilitado, no se puede inhabilitar por software durante el funcionamiento normal del microcontrolador.

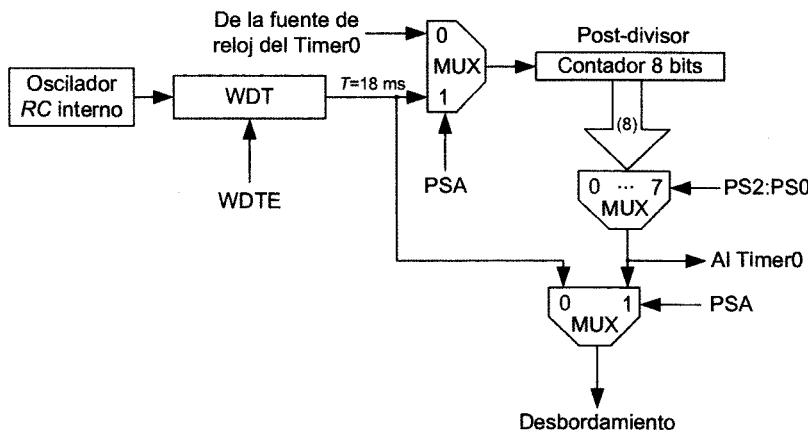


Figura 2.14 Diagrama de bloques de los circuitos asociados al perro guardián (WDT) en los PIC de gama media. PSA y PS2:PS0 son bits del registro de funciones especiales OPTION. WDTE es el bit de configuración del microcontrolador que habilita la operación del perro guardián.

El desbordamiento del perro guardián ocurre cada 18 ms, aproximadamente. Para evitarlo hay que poner a 0 el contador del perro guardián antes de que transcurra ese tiempo. Esto se hace con la instrucción `clrwdt`. Estos 18 ms se pueden ampliar hasta 2,3 s mediante la asignación de un contador adicional (post-divisor) al perro guardián. La tabla 2.1 muestra los tiempos que se pueden conseguir con diferentes asignaciones en el post-divisor. La asignación y programación del post-divisor se realiza con los bits PSA, PS0, PS1 y PS2 del registro OPTION. El bit PSA debe ser 1 para que el post-divisor sea asignado al perro guardián (figura 2.14).

PS2:PS0	Factor de división del post-divisor	Tiempo aproximado de desbordamiento del perro guardián (ms)
000	1:1	18
001	1:2	36
010	1:4	72
011	1:8	144
100	1:16	288
101	1:32	576
110	1:64	1152
111	1:128	2304

Tabla 2.1

Factores de división del post-divisor y los correspondientes tiempos aproximados de desbordamiento del perro guardián. PS2, PS1 y PS0 son bits del registro de funciones especiales OPTION.

2.2 Familias de microcontroladores PIC

Los microcontroladores PIC se pueden clasificar, atendiendo al tamaño de sus instrucciones, en tres grandes grupos o gamas:

- Gama baja: microcontroladores con instrucciones de 12 bits.
- Gama media: microcontroladores con instrucciones de 14 bits.
- Gama alta: microcontroladores con instrucciones de 16 bits.

Los microcontroladores PIC también se agrupan en cinco grandes familias: PIC10, PIC12, PIC16, PIC17 y PIC18. Los PIC10 son, básicamente, microcontroladores de 6 terminales. La familia de los PIC12 agrupa a los microcontroladores disponibles en encapsulado de 8 terminales. Algunas de estas cinco familias tienen numerosas subfamilias, como sucede con los PIC16. Además, algunas de estas familias incluyen dispositivos de más de una gama, como los PIC16 y PIC12, que tienen dispositivos de gama baja y media. Los PIC17 y PIC18 son de gama alta. El criterio empleado para clasificar un PIC dentro de una familia es, pues, un tanto complejo. La tabla 2.2 es una ayuda para ubicar los PIC.

Tabla 2.2 Tabla resumen de la relación familia – gama en los microcontroladores PIC.

Familia	Gama			Rasgo distintivo
	Baja	Media	Alta	
PIC10	x			6 terminales
PIC12X5	x			8 terminales
PIC12 (excepto PIC12X5)		x		8 terminales
PIC16X5	x			-
PIC16 (excepto PIC16X5)		x		-
PIC17			x	-
PIC18			x	Gama alta mejorada

2.2.1 Microcontroladores de gama baja

Los microcontroladores PIC de gama baja disponen de un repertorio de 33 instrucciones de 12 bits cada una. La memoria de programa tiene una capacidad de hasta 2k (2048) palabras de 12 bits y está organizada en páginas de 512 palabras cada una. La memoria de datos está formada por registros de 8 bits y se organiza en bancos de hasta 32 registros cada uno.

Los PIC de gama baja tienen una pila (*stack*) de dos niveles, para guardar direcciones de la memoria de programa. No tienen interrupciones. Su entrada y salida tiene un pequeño número de dispositivos, que comprende hasta tres puertos de entrada y salida de hasta 8 bits cada uno, un temporizador y un comparador (según el modelo de PIC).

Los microcontroladores PIC de gama baja son de tres familias:

- Los PIC16X5xx
- Los PIC12X5xx
- Los PIC10

Los PIC16X5x constituyen la principal familia de los PIC de gama baja. Estos microcontroladores se presentan con memoria de programa EPROM, OTP o FLASH, según el modelo, su consumo en condiciones normales es menor de 2 mA a 5 V, y en el modo de bajo consumo es menor de 3 μ A a 3 V. Su encapsulado es de 18, 20 ó 28 terminales. La figura 2.15 muestra su arquitectura interna.

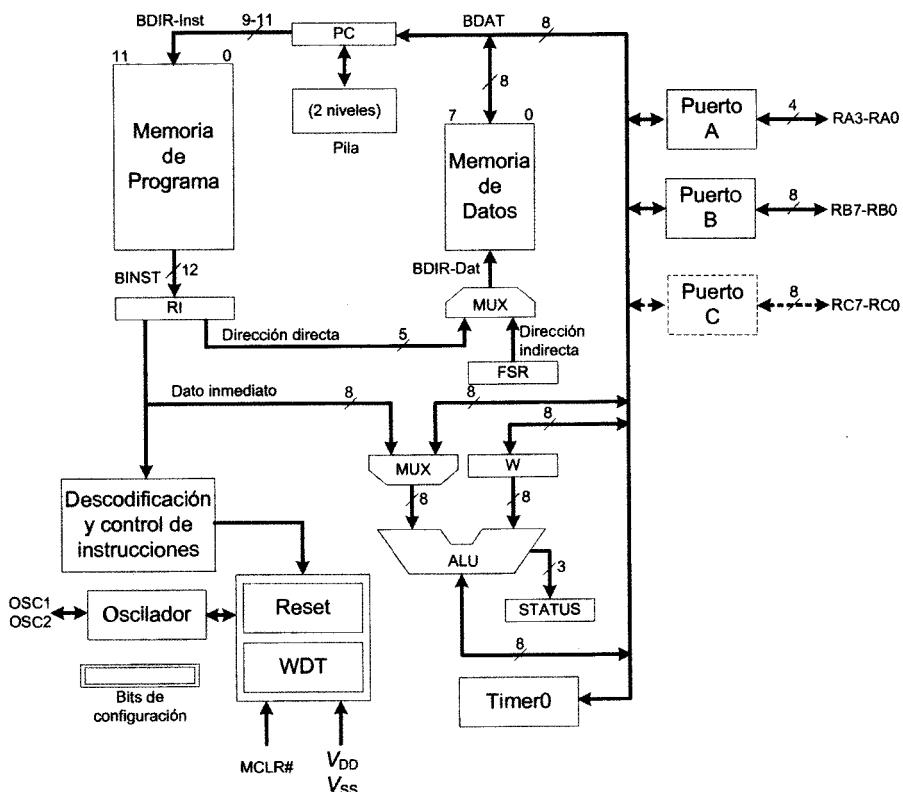


Figura 2.15 Arquitectura interna de la familia PIC16X5x

La familia PIC12X5x se caracteriza por su encapsulado de 8 terminales. Dado el pequeño número de terminales disponibles, los recursos de entrada y salida se reducen a un puerto paralelo de 6 bits, un temporizador y convertidor A/D, según el modelo. La memoria de programa es OTP o FLASH, también según el modelo. Algunos modelos tienen memoria EEPROM de da-

tos. El consumo de corriente es menor de 2 mA a 5 V, y menos de 2 μ A a 3 V en el modo de bajo consumo. La figura 2.16 muestra su arquitectura interna.

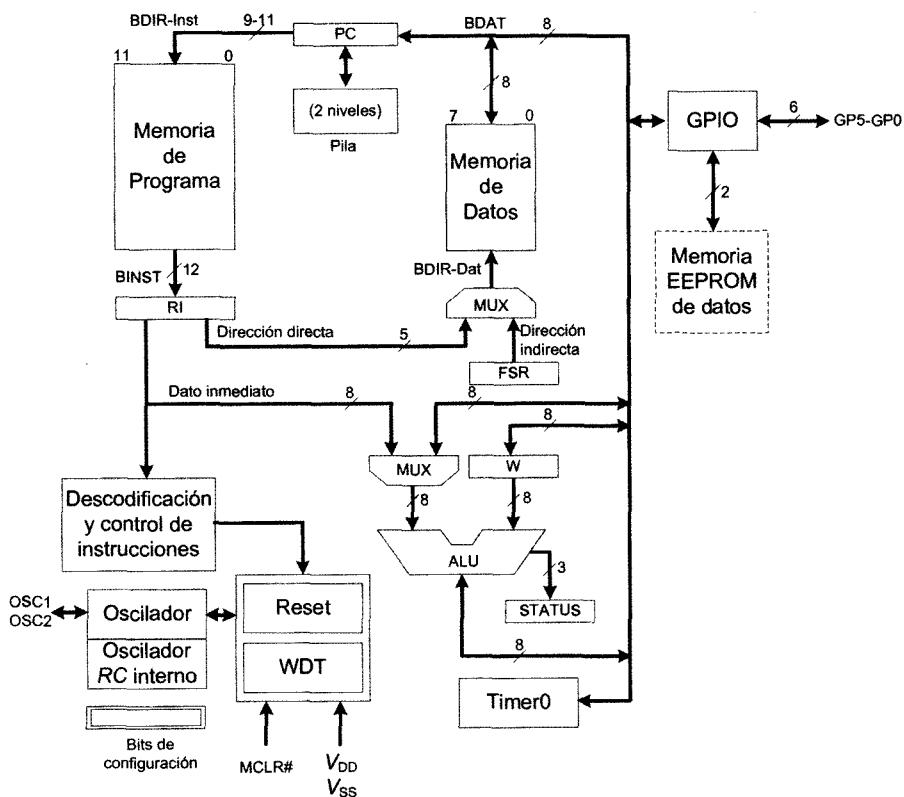


Figura 2.16 Arquitectura interna de la familia PIC12X5xx.

La familia PIC10Fxxx se distingue por su pequeño tamaño, pues se presenta en encapsulados de 8 ó 6 terminales. Todos estos PIC tienen memoria de programa de tipo FLASH, aunque no disponen de memoria EEPROM de datos. Sus recursos de entrada y salida se limitan a un puerto paralelo de 4 bits, un temporizador y un comparador (según el modelo). En condiciones normales de operación, el consumo es menor de 350 μ A a 2 V, y en el modo de bajo consumo es menor de 100 nA a 2 V. La figura 2.17 muestra su arquitectura interna.

2.2.2 Microcontroladores de gama media

La figura 2.18 muestra la arquitectura general de los PIC de gama media. Estos dispositivos tienen un repertorio de 35 instrucciones de 14 bits cada una. La memoria de programa puede llegar a las 8k (8192) palabras de 14 bits y se organiza en páginas de 2k (2048) palabras cada una. La memoria de datos

está formada por registros de 8 bits y está organizada en bancos de 120 registros cada uno, con un máximo de cuatro bancos. En general, los PIC de gama media poseen algo de memoria EEPROM de datos. Todos tienen una pila de 8 niveles, donde se almacenan direcciones de la memoria de programa.

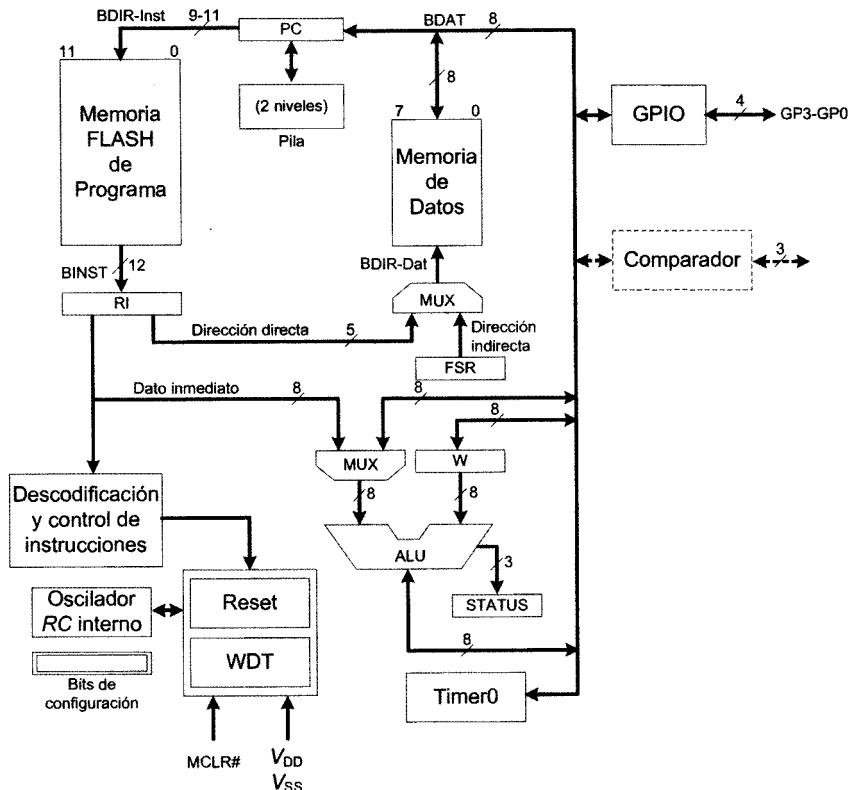


Figura 2.17 Arquitectura interna de los PIC10Fxxx, microcontroladores de gama baja con encapsulado de 6 terminales y memoria de programa FLASH.

Estos PIC poseen un sistema de interrupciones fijas para atender interrupciones internas y una interrupción externa. Las interrupciones internas provienen de sus dispositivos de entrada y salida. En general, cada bloque de entrada y salida puede generar una solicitud de interrupción a la CPU. Todos los PIC de gama media tienen un terminal para recibir las solicitudes de interrupción proveniente de algún dispositivo externo.

Los microcontroladores PIC de la gama media tienen una amplia variedad de dispositivos de entrada y salida. Cuentan con varios puertos paralelos (puertos A, B, C, etc.) para la comunicación paralela con dispositivos externos, aunque cada puerto tiene sus especificidades. Disponen también de hasta tres temporizadores, dos módulos de captura, comparación y modulación

de ancho de pulso (PWM), denominados módulos CCP, varios tipos de puertos serie para la comunicación serie asincrónica y sincrónica, un convertidor A/D de 10 bits asociado a un multiplexor con varias entradas analógicas, etc. La figura 2.19 muestra la arquitectura del PIC16F873, que se usa ampliamente en los ejemplos de este libro.

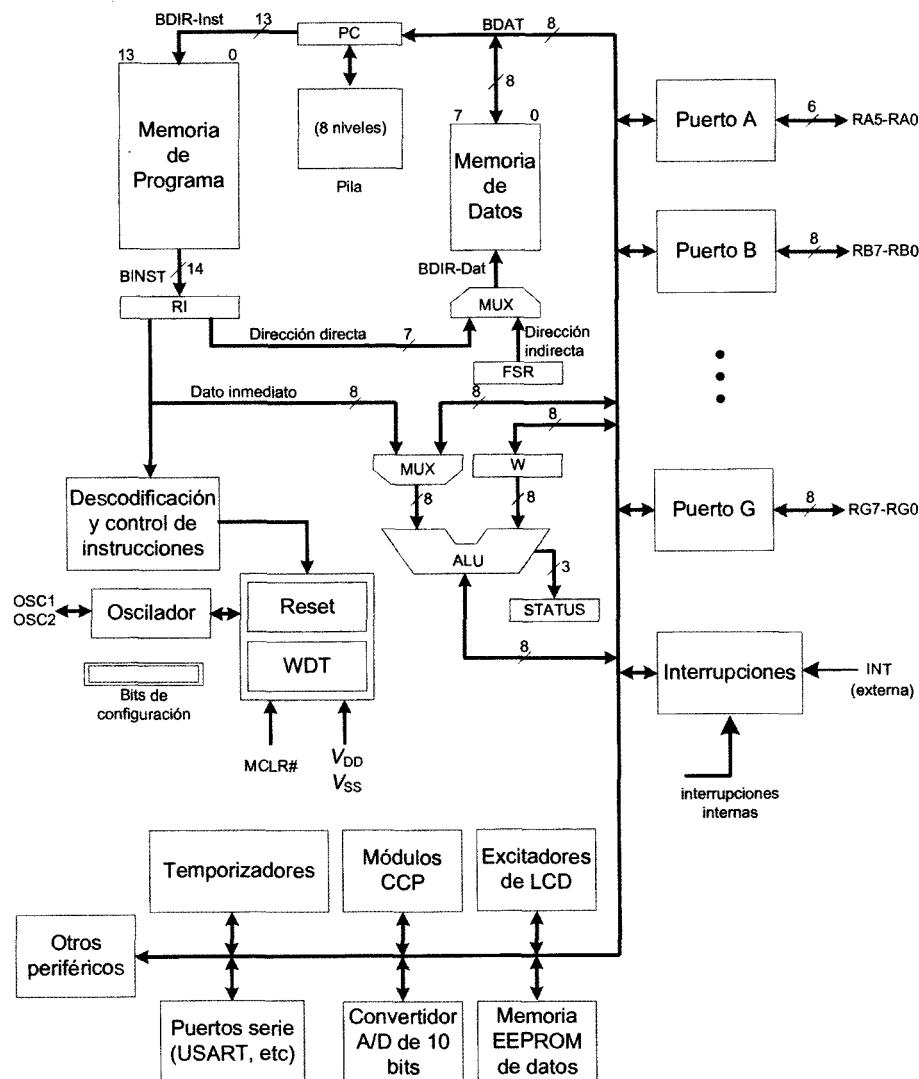


Figura 2.18 Arquitectura interna de los microcontroladores PIC de gama media.

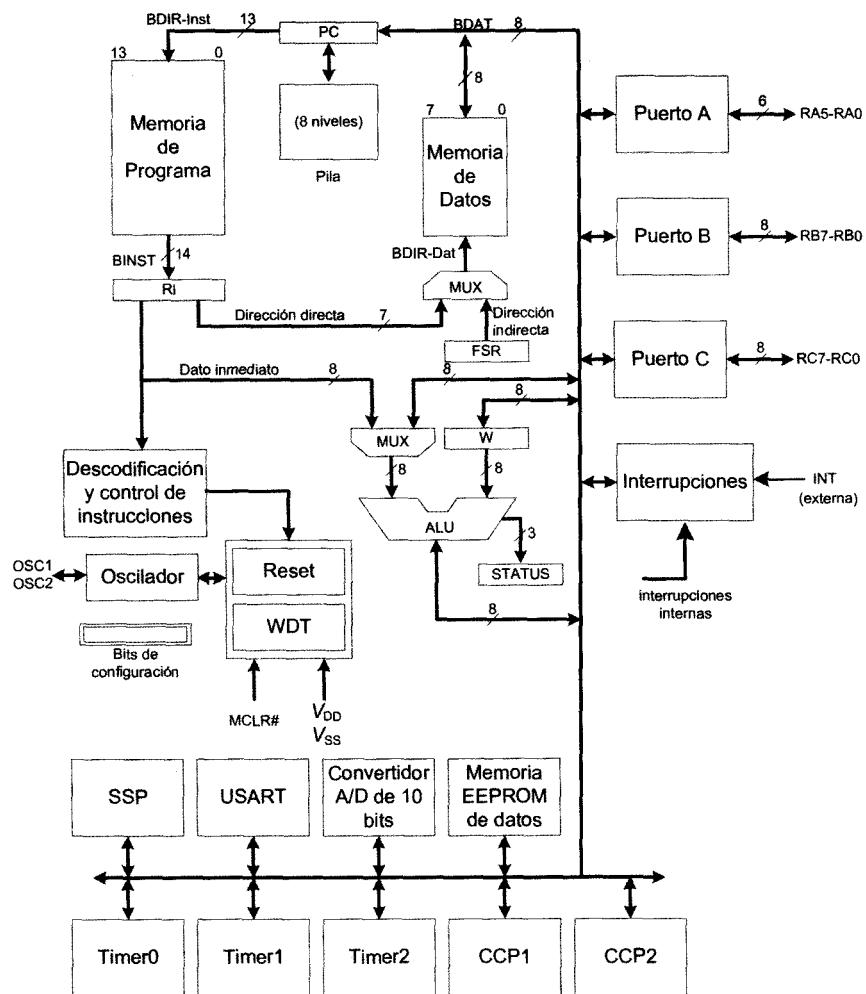


Figura 2.19 Arquitectura interna del PIC16F873, utilizado como ejemplo de microcontrolador PIC de gama media.

Los PIC de gama media comprenden las siguientes familias:

- Los PIC16, excepto los PIC16X5xx que son de gama baja.
- Los PIC12X6xx, con encapsulado de 8 terminales.

Los PIC de gama media con encapsulado de 8 terminales se caracterizan porque pueden funcionar con tensiones pequeñas (2 V) y con un consumo de corriente de sólo unos 100 µA en funcionamiento normal, y 1 nA a 2 V en el modo de bajo consumo. La figura 2.20 muestra la arquitectura interna de un PIC de gama media con encapsulado de 8 terminales.

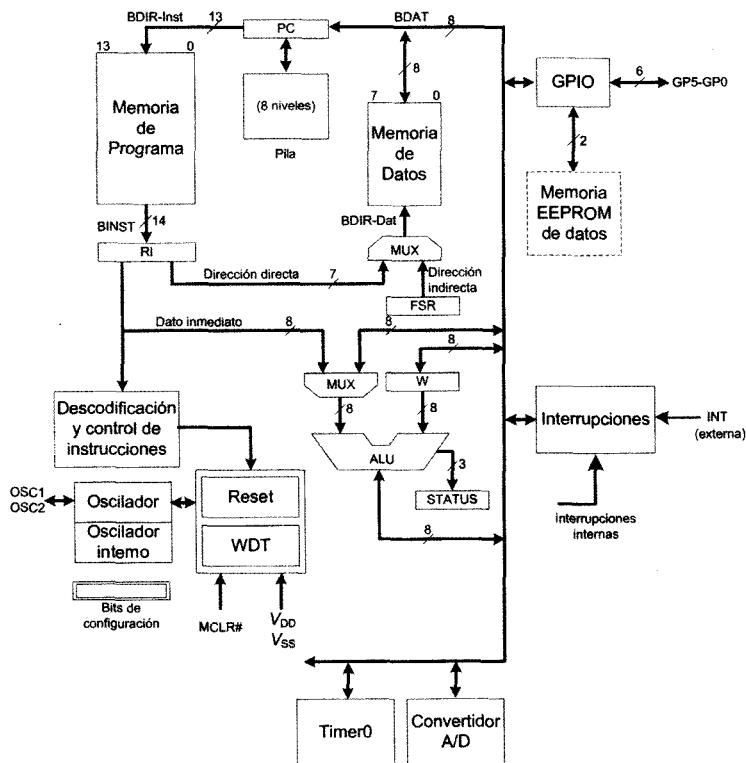


Figura 2.20 Arquitectura interna de los microcontroladores PIC12CE67x, que son de gama media con encapsulado de 8 terminales.

2.2.3 Microcontroladores de gama alta

Los microcontroladores de gama alta se distinguen por sus instrucciones de 16 bits, mayor profundidad en la pila y un sistema de interrupciones más elaborado que incluye, además de las interrupciones internas de los dispositivos integrados en el microcontrolador, varias entradas para interrupciones externas. Algunos PIC de la gama alta tienen una arquitectura abierta, que admite la ampliación de las memorias de programa y de datos. Finalmente, el número de dispositivos de entrada y salida es bastante más amplio que el de los PIC de gama media.

Los PIC de gama alta están disponibles en dos familias, que comprenden:

- Los PIC17
- Los PIC18

Los PIC17 tienen un repertorio de 58 instrucciones de 16 bits cada una. La memoria de programa puede ser de hasta 64k (65 536) palabras de 16 bits cada una y la memoria de datos puede llegar a tener hasta 1k (1024) registros

de 8 bits. La memoria de programa de los PIC17 es EPROM, ROM u OTP. La pila tiene 16 niveles de profundidad. Su sistema de interrupciones incluye el tratamiento de prioridades.

Una característica interesante de los PIC17 es su arquitectura abierta. Estos microcontroladores pueden trabajar en cuatro modos diferentes: como microcontrolador, microcontrolador protegido, microcontrolador ampliado y microprocesador. En los modos microcontrolador y microcontrolador protegido sólo se tiene acceso a la memoria de programa interna del microcontrolador. En los modos microcontrolador ampliado y microprocesador, es posible acceder a ampliaciones externas de la memoria de programa.

Estos microcontroladores disponen de una amplia gama de dispositivos de entrada y salida: puertos paralelos, puerto serie, temporizadores, convertidor A/D, etc. La figura 2.21 muestra la arquitectura interna de los PIC17.

Los microcontroladores PIC18 constituyen una numerosa familia de microcontroladores, que en su gran mayoría tienen memoria de programa de tipo FLASH. Tienen un repertorio de 77 instrucciones de 16 bits. La memoria de programa puede ser de hasta 2 MB (2^{20} bytes o 2^{10} palabras de 16 bits), y la memoria de datos puede llegar a los 4k (4096) registros de 8 bits cada uno. Algunos miembros de la familia PIC18 admiten una expansión externa de la memoria de programa. Poseen una pila de 31 niveles de profundidad, así como un sistema de interrupción muy elaborado, con interrupciones internas provenientes de los dispositivos de entrada y salida integrados en el microcontrolador, y tres interrupciones externas. Poseen un gran número y variedad de dispositivos de entrada y salida integrados.

Varios dispositivos PIC18 están diseñados para trabajar con tensiones bajas (2,0 V a 3,6 V) y con corrientes inferiores a 2 mA. La figura 2.22 muestra la arquitectura de la familia PIC18.

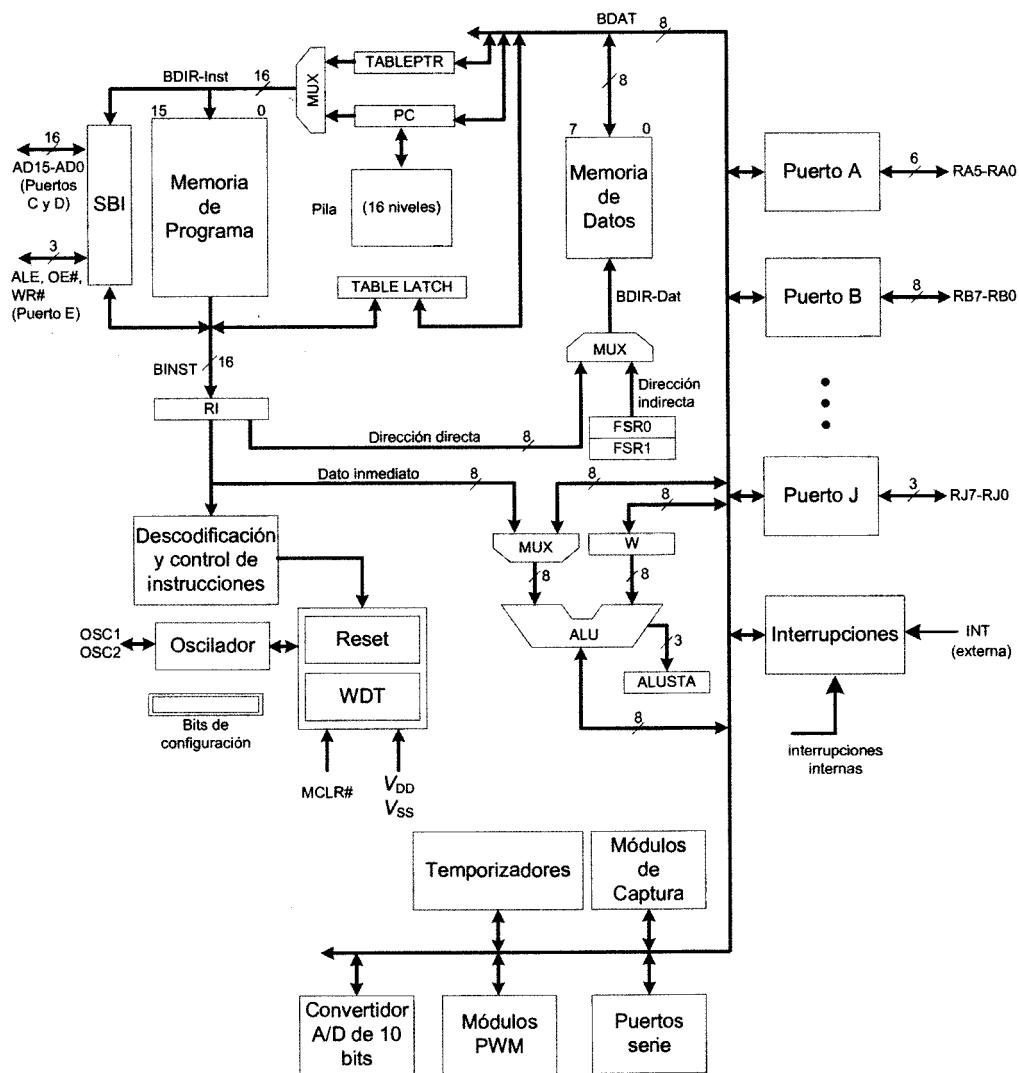


Figura 2.21 Arquitectura interna de los PIC17. Destaca la posibilidad de ampliar externamente la memoria de programa.

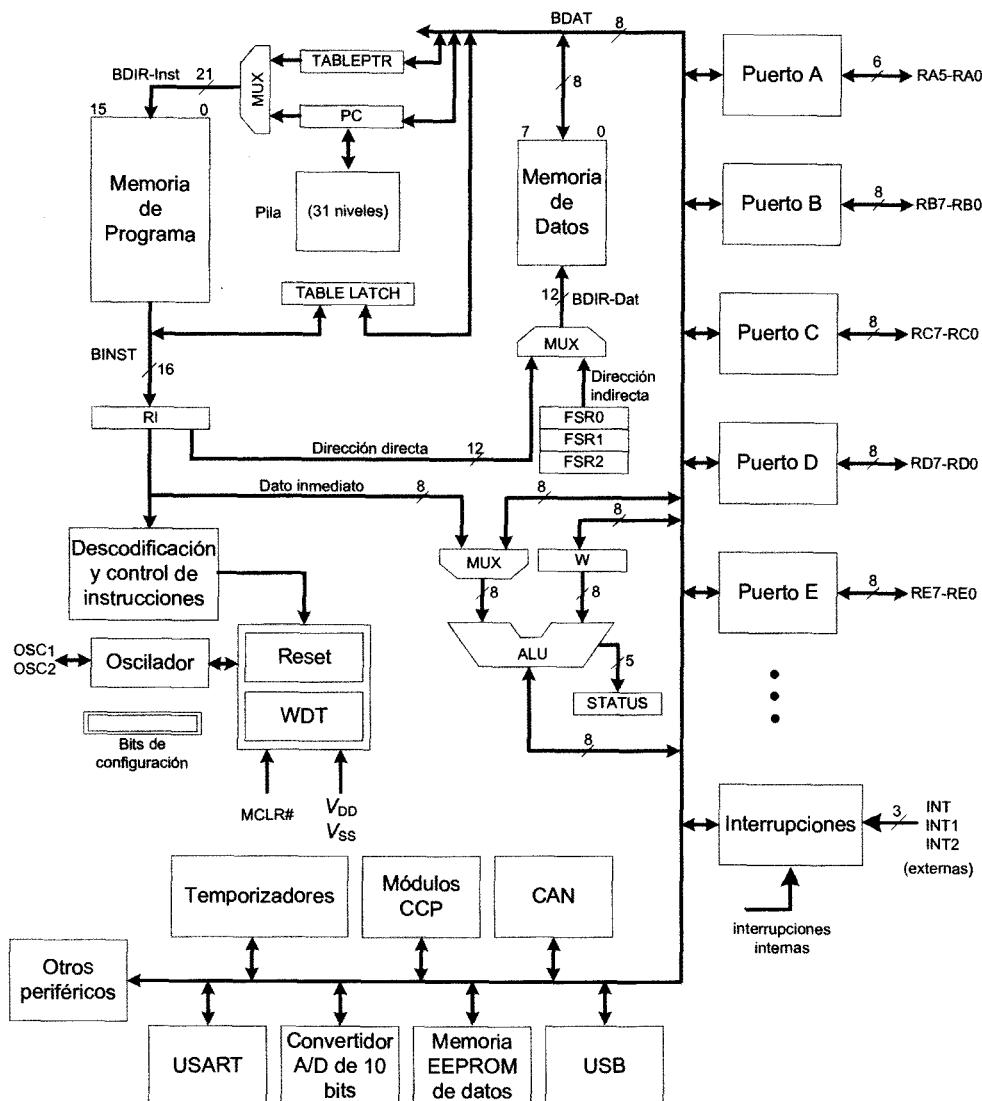


Figura 2.22 Arquitectura interna de los PIC18. Estos microcontroladores superan notablemente las prestaciones de los PIC de gama media.

3 La memoria en los microcontroladores

En este capítulo se estudia la organización y estructura de la memoria en los microcontroladores en general y en los PIC de gama media en particular. Primero se definen algunos conceptos básicos tales como palabra, dirección, capacidad de la memoria y unidades en que se mide. A continuación se estudian dos formas de organizar la memoria en los microcontroladores: lineal y por páginas. Se exponen luego las tecnologías empleadas en la fabricación de la memoria. Finalmente se describe la organización y características de la memoria en los PIC de gama media.

3.1 Conceptos básicos

La memoria de un microcontrolador es el lugar donde están almacenados el programa que se ejecuta y los datos o variables utilizados por el programa. La memoria es un conjunto de celdas o localizaciones que se identifican por su dirección. En cada celda se almacena una palabra.

Una *palabra* es la unidad lógica de información almacenada en una celda de la memoria. Se utilizan palabras de 1, 8, 12, 14 ó 16 bits de longitud. Una palabra de 8 bits es un *octeto* o *byte* (B) (figura 3.1).

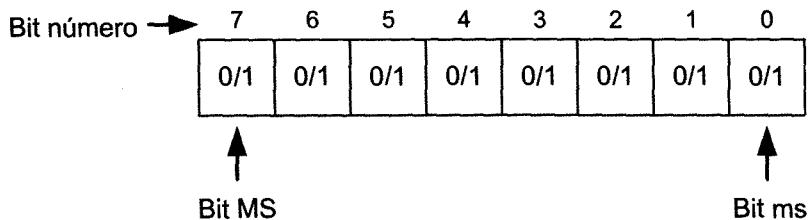


Figura 3.1 Una palabra de 8 bits u octeto o byte. Se indican las posiciones de los bits más significativo (MS) y menos significativo (ms) (o LS, least significant) en la palabra.

La cantidad de celdas de la memoria define su *capacidad*. La capacidad o tamaño de la memoria se mide en palabras, ya sean bits, bytes o palabras de 12, 14 o más bits. El Sistema Internacional (SI) de unidades establece prefijos para indicar factores de multiplicación que son potencias de 10, como, por ejemplo, kilo, mega, giga y tera. Sin embargo, en computación y electrónica es común usar estos mismos prefijos para indicar factores de multiplicación que son potencias de 2. Para evitar confusiones, la Comisión Electrotécnica Internacional (CEI) introdujo en 1998 otros prefijos para indicar factores de multiplicación que son potencias de 2. En la tabla 3.1 se dan algunos de los

prefijos indicadores de potencias de 10 y de 2. Sin embargo, dado que es una costumbre muy extendida utilizar los prefijos que indican potencias de 10 con el significado de ser potencias de 2, en este libro se sigue esa costumbre. Es decir, utilizaremos por ejemplo 1 kB para indicar una capacidad de memoria de 1024 bytes, en lugar de escribir 1 KiB, que sería lo correcto según la CEI. La diferencia es importante sólo cuando se trata de grandes cantidades de memoria.

Tabla 3.1 Algunos de los prefijos, símbolos y factores de multiplicación definidos en el Sistema Internacional de Unidades y por la Comisión Electrotécnica Internacional.

Prefijo	Símbolo	Factor	Prefijo	Símbolo	Factor
kilo	k	10^3	kibi	Ki	2^{10}
mega	M	10^6	mebi	Mi	2^{20}
giga	G	10^9	gibi	Gi	2^{30}
tera	T	10^{12}	tebi	Ti	2^{40}

La dirección de una celda es el ente que identifica la celda en la memoria. La forma más simple de identificar las celdas es asignar a cada una un número entero consecutivo. Entonces, la dirección es el número binario que denota la posición de la celda en la memoria (figura 3.2). Si D es la dirección de una celda cualquiera, los valores de las direcciones en una memoria de N celdas son:

$$D = 0, 1, 2, \dots, (N - 1) \quad (3.1)$$

Lógicamente, la cantidad n de bits necesarios para formar la dirección de una celda es función directa de la capacidad de la memoria y está dada por

$$N = 2^n \quad (3.2)$$

Por ejemplo, en una memoria de $N = 1$ kB se necesitan $n = 10$ bits para formar la dirección de una celda cualquiera, pues $2^{10} = 1024$, que es la cantidad de celdas que componen la memoria. Las direcciones posibles son $D = 0, 1, 2, \dots, 1023$. Expresando estos números en el sistema de numeración hexadecimal, tal como es costumbre, se tiene $D = 0, 1, 2, \dots, 3FFh$.

3.1.1 Organización lógica de la memoria

La memoria de los microcontroladores se organiza normalmente como un todo único (organización lineal) o por bloques llamados páginas. En la organización lineal, las direcciones de las celdas son números binarios consecutivos. En este caso, cada celda se identifica por su dirección lineal (D), formada por un número binario único. Las direcciones lineales tienen la forma dada por la expresión (3.1).

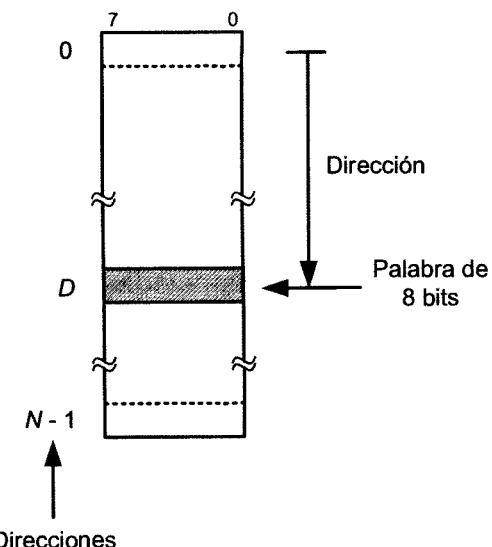


Figura 3.2 Direcciones en una memoria de N celdas de 8 bits cada una.

Una página es una porción de memoria de tamaño fijo. Las páginas están una a continuación de la otra sin solaparse. Cada página se puede identificar por un número consecutivo denominado número de página. Dentro de una página cualquiera, las celdas se identifican por su posición respecto al comienzo de la página, llamada *desplazamiento*.

En una memoria organizada en páginas (figura 3.3), la dirección de una celda se compone de dos elementos: el número de la página (*npag*) y el desplazamiento (*desp*). El conjunto de estos dos elementos constituye la dirección lógica (*Dlog*) de la celda en el esquema paginado de la memoria. La dirección lógica de una celda se representa entonces así:

$$Dlog = npag : desp \quad (3.3)$$

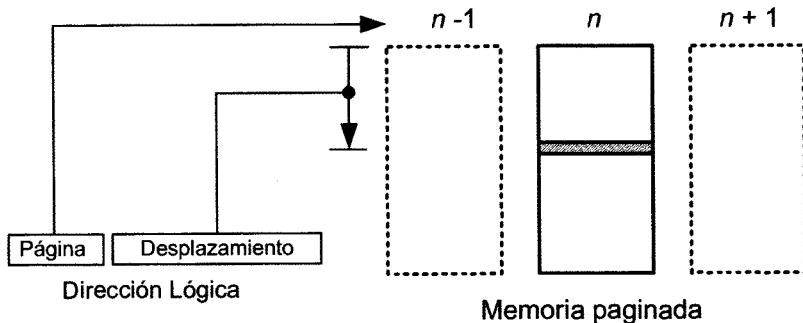


Figura 3.3 Organización de la memoria en páginas. Se muestran los dos elementos que conforman la dirección lógica de una celda de memoria: el número de la página y el desplazamiento.

La dirección lineal se puede obtener de la dirección lógica. Mediante un sencillo razonamiento se puede ver que la dirección lineal de una celda se obtiene multiplicando el número de la página por su tamaño ($tpag$) y sumando a este resultado el desplazamiento que tiene la celda dentro de la página. Es decir:

$$D = npag \times tpag + desp \quad (3.4)$$

Por ejemplo, en una memoria organizada en páginas de 256 B (100h bytes), una celda que se encuentre en la página 2 con desplazamiento A5h (figura 3.4), tiene una dirección lineal $D = 2 \times 100h + A5h = 2A5h$.

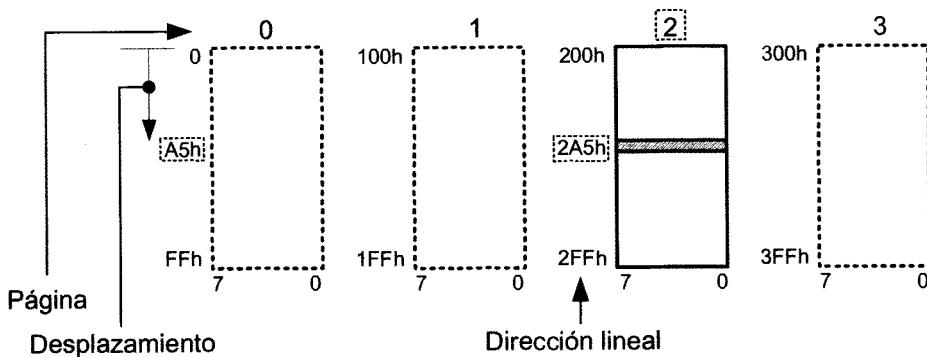


Figura 3.4 Paginado de una memoria de 1 kB en páginas de 256 B. Obsérvese la relación entre la dirección lineal y la lógica. La celda que se encuentra en la página 2 con desplazamiento A5h, tiene dirección 2A5 en un esquema de organización lineal.

Obsérvese en (3.4) que si el tamaño de las páginas es una potencia de 2, es decir, si $tpag = 2^k$, entonces k es el número de bits necesarios para representar cualquier desplazamiento dentro de la página; es decir, $desp$ es un número binario de k bits. En este caso, si se opera en el sistema de numeración binaria, el producto $npag \times tpag$ agrega k ceros a la derecha del número $npag$. Al realizar la suma indicada en (3.4), el valor $desp$ ocupa el lugar de esos ceros, de modo que la dirección lineal resultante tiene en sus bits más significativos el valor de $npag$ y en los menos significativos, el valor de $desp$, (figura 3.5). (Si se empleara el sistema de numeración decimal, sucedería lo mismo si el tamaño de las páginas fuera una potencia de 10).

3.1.2 Tipos de memorias

La memoria de programa de los microcontroladores es no volátil y básicamente de sólo lectura. En su fabricación se utilizan varias tecnologías: ROM (*Read-Only Memory*), EPROM (*Erasable Programmable ROM*), OTP (*One Time Programmable*) y FLASH.

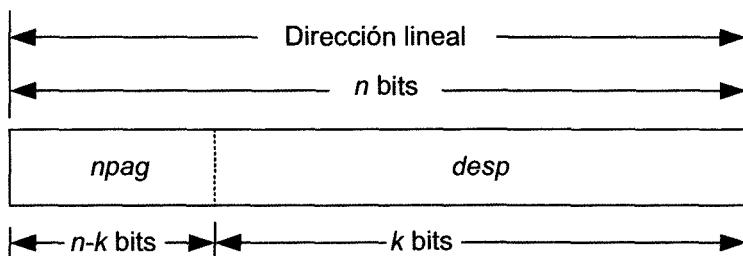


Figura 3.5 Formación de la dirección lineal a partir de los componentes de la dirección lógica en un sistema paginado, si el tamaño de las páginas es una potencia de 2. El desplazamiento (*desp*) ocupa los k bits menos significativos y el número de la página (*npag*) ocupa los $n-k$ bits más significativos de la dirección lineal de n bits.

La memoria de datos, en cambio, es fundamentalmente de lectura y escritura, y no hace falta que la información permanezca en ella al interrumpir el suministro de energía al microcontrolador; es decir, la memoria es volátil. Se utilizan memorias RAM (*Random Access Memory*) estáticas. Muchos microcontroladores utilizan cierta cantidad adicional de memoria (externa) de lectura y escritura, no volátil, como parte de la memoria de datos. Se trata de memoria EEPROM (*Electrically Erasable Programmable Read-Only Memory*), cuyo objetivo es permitir el almacenamiento de datos fijos o poco variables.

RAM

La memoria RAM es una memoria de lectura y escritura. Hay dos variantes: la estática y la dinámica. En la memoria RAM estática la información almacenada permanece estable indefinidamente mientras no se suprima la tensión de alimentación. Esas son las diferencias de las memorias RAM dinámicas, que requieren un refrescamiento periódico de la información almacenada. Las RAM dinámicas se usan profusamente en los ordenadores personales, pero no en los microcontroladores.

ROM

En los microcontroladores que utilizan memoria ROM, la información se graba durante el proceso de fabricación del dispositivo y no puede ser alterada ulteriormente. Por ello, la información que se desea grabar en la memoria, que puede ser el programa y algunos datos, debe haber sido comprobada y depurada minuciosamente antes de encargar la fabricación del microcontrolador. La memoria ROM es la más barata de todas las opciones, siempre

que el volumen de dispositivos fabricados sea suficientemente alto para que el proceso sea rentable.

En los microcontroladores PIC, las letras 'CR' en el nombre del dispositivo indican que la memoria de programa es ROM. Ejemplos: PIC16CR65, PIC16CR72.

EPROM y OTP

Estas memorias son muy similares pero se diferencian en su encapsulado. Los dispositivos con memoria EPROM tienen una ventana de cristal de cuarzo para poderlos borrar masivamente con luz ultravioleta. Los dispositivos OTP usan la misma memoria interna que las EPROM, pero sin la ventana de cristal, por lo que una vez programados no se puede borrar ni alterar su información.

En los microcontroladores PIC, la letra 'C' en el nombre del dispositivo indica que la memoria es OTP o EPROM. El PIC16C74B/JW es un dispositivo con memoria EPROM, porque el sufijo JW indica que el encapsulado dispone de una ventana de cuarzo para el borrado. Los dispositivos PIC16C72A/P y PIC16C74B/SO son OTP con encapsulado plástico DIP y SOIC para montaje en superficie, respectivamente.

EEPROM

La memoria EEPROM es una memoria no volátil de lectura y escritura. La escritura de la memoria se realiza por medios eléctricos. Las celdas pueden ser escritas individualmente sin una operación previa de borrado. La memoria se puede reprogramar un número finito, aunque muy grande, de veces (del orden de 10^6).

FLASH

En las memorias FLASH se pueden leer y escribir celdas individualmente, aunque, en general, para escribir en una celda hay que borrar primero su información. El borrado de estas memorias se realiza por bloques de celdas de memoria, no celda a celda. Esto las diferencia de las memorias EEPROM. Borrar es poner todos los bits del bloque de celdas en 0. La operación de escritura consiste en poner selectivamente a 1 los bits deseados. Con una operación de escritura se puede poner a 1 un bit que está en 0, pero para poner en 0 un bit que está en 1 hay que borrar el bloque de celdas correspondiente. Entonces, a menudo, una operación de escritura de información en una celda de memoria flash se convierte en una operación de lectura - borrado - escri-

tura del bloque de celdas donde se desea escribir la información. Todas las operaciones de borrado, lectura y escritura se realizan con la tensión de funcionamiento normal del dispositivo, sin necesidad de una tensión mayor. La memoria se puede borrar o escribir un número finito, aunque muy grande, de veces (del orden de 10^5). Por ejemplo, en un PIC16F873A, la memoria flash de programa se puede escribir unas 100 000 veces.

El uso de memoria FLASH en los microcontroladores PIC se indica con la letra 'F' en el nombre del dispositivo. Ejemplo: PIC16F873.

3.2 La memoria en los microcontroladores PIC de gama media

La memoria de los microcontroladores PIC se organiza, según la arquitectura Harvard, en dos espacios independientes: uno para la memoria de programa y otro para la memoria de datos.

La memoria de programa es básicamente de sólo lectura y puede ser ROM, OTP, EPROM o FLASH. En ella se almacenan las instrucciones del programa que ejecuta el microcontrolador. En algunos modelos de PIC, la memoria de programa se puede leer desde el programa que se está ejecutando, por lo que en ellos es posible almacenar datos fijos en dicha memoria. En algunos modelos de PIC con memoria FLASH, se puede incluso escribir datos en la memoria de programa.

La memoria de datos está realizada sobre memoria RAM estática, de modo que es una memoria volátil de lectura y escritura, aunque en algunos modelos de PIC puede existir, adicionalmente en un espacio separado, cierta cantidad de memoria EEPROM, denominada memoria EEPROM de datos. En esta EEPROM se pueden almacenar datos fijos o que varían poco. La tabla 3.2 muestra, a modo de ejemplo, la cantidad de memoria disponible en algunos modelos de microcontroladores PIC de gama media.

Tabla 3.2 Tamaño de la memoria de programa y de datos en algunos microcontroladores PIC de gama media.

PIC	Memoria de Programa (en palabras de 14 bits) FLASH	Memoria de Datos (en bytes)	
		RAM	EEPROM
PIC18F83	512	36	64
PIC16F84/84A	1024	68	64
PIC16F873/874	4096	192	128
PIC16F876/877	8192	368	256

3.2.1 La memoria de programa.

La memoria de programa se organiza en páginas. En los PIC de gama media puede haber hasta 4 páginas de 2k palabras cada una. Por tanto, la capacidad de la memoria puede llegar a ser 8k palabras. Las palabras de estos PIC son de 14 bits.

La figura 3.6 ilustra cómo está organizada esta memoria. En la página 0, la dirección 0h está reservada al *reset*, mientras que la dirección 4h se reserva para el vector de interrupciones. En esas direcciones se colocan las instrucciones de salto a los programas correspondientes.

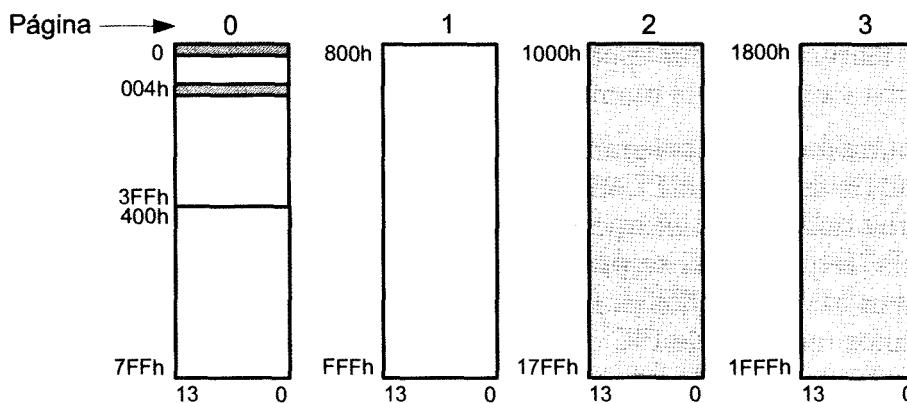


Figura 3.6 Pagenado de la memoria de programa en los microcontroladores PIC de gama media. Puede haber hasta 4 páginas de 2k palabras cada una, para un total de 8k palabras en la memoria. Las palabras son de 14 bits. En el PIC16F873 la parte sombreada no está realizada, pues este microcontrolador dispone de 4k celdas distribuidas en las páginas 0 y 1. En el PIC16F84 sólo existen las primeras 1024 celdas de memoria (direcciones 0 a 3FFh de la página 0).

3.2.1.1 Direcciónamiento de la memoria de programa

El contador de programa (PC: *Program Counter*) es el registro del microcontrolador cuya función es direccionar la memoria de programa. El PC almacena direcciones de instrucciones; más precisamente, en el PC está la dirección de la siguiente instrucción que hay que ejecutar. En otras palabras, el PC “apunta” a la instrucción del programa que se ejecutará a continuación de la instrucción actual.

En los microcontroladores de gama media, el PC es de 13 bits, de modo que puede direccionar el espacio de 8k palabras de la memoria de programa. Como la memoria está paginada, los bits 12 y 11 del PC dan el número de la

página, mientras que los bits del 10 al 0 dan la dirección dentro de la página o desplazamiento.

El PC se relaciona con dos registros de funciones especiales situados en la memoria de datos: PCLATH (*PC Latch High*) y PCL (*PC Low*). Los 8 bits menos significativos del PC constituyen el registro PCL, el cual puede ser leído y escrito por el programador. Los 5 bits más significativos del PC o PCH no se pueden leer, pero se pueden modificar a través del registro PCLATH. En particular, el número de la página se carga desde los bits 4 y 3 de PCLATH ($\text{PCLATH}^{<4:3>}$) (figura 3.7). Así pues, en todo momento es posible conocer el contenido del byte de menor peso del PC al leer el registro PCL, pero no se puede conocer el valor de los 5 bits más significativos del PC, porque PCLATH nunca se actualiza desde el PC.

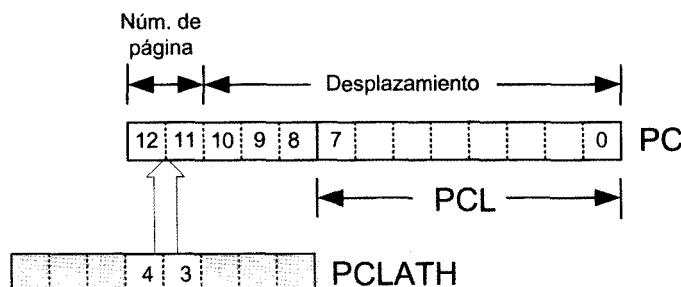


Figura 3.7 Contador de programa (PC) y componentes de una dirección en la memoria de programa: número de página (2 bits) y desplazamiento (11 bits). El número de página se carga desde los bits $\text{PCLATH}^{<4:3>}$. Los 8 bits menos significativos del PC constituyen el registro PCL. PCLATH y PCL son registros de funciones especiales situados en la memoria de datos del microcontrolador.

Durante la ejecución de un programa, el PC se incrementa en 1 con cada instrucción que se ejecute, excepto con aquellas que modifican el contenido del PC.

Las instrucciones de salto (goto), llamadas a subrutinas (call) y retornos (return, retfie y retlw), y aquellas instrucciones cuyo destino es el registro PCL, modifican el contenido del PC. En estos casos, la relación entre el PC y los registros PCLATH y PCL es la siguiente:

- Si se ejecuta una instrucción cuyo destino es el registro PCL (salto indirecto a la dirección apuntada por PCLATH y PCL), los 8 bits menos significativos del PC se cargan con el resultado de la instrucción, mientras que los 5 bits más significativos del PC se cargan con los 5 bits menos significativos del registro PCLATH (figura 3.8a).

- Si se ejecuta una instrucción goto o call, los 11 bits de menor peso del PC son aportados por la instrucción, mientras que los 2 bits más significativos del PC, que reportan el número de la página a la que se salta, se cargan desde los bits 4 y 3 del registro PCLATH (figura 3.8b).
- Si se ejecuta una instrucción de retorno desde una subrutina, los 13 bits del PC se cargan desde el tope de la pila. No interviene PCLATH.

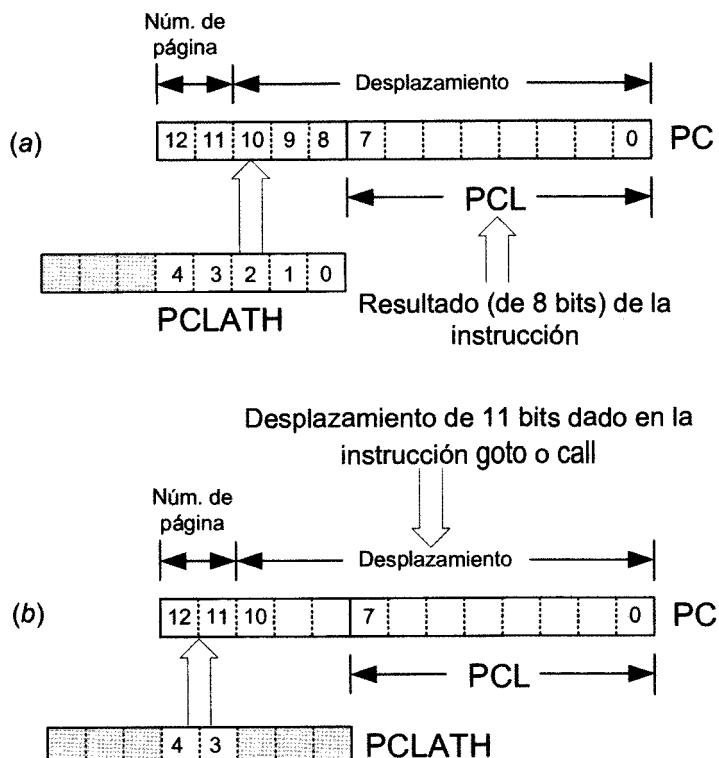


Figura 3.8 Contador de programa (PC) y registros de funciones especiales PCLATH y PCL. Se muestra lo que ocurre cuando (a) se ejecuta una instrucción cuyo destino es PCL o (b) se ejecuta una instrucción goto o call.

Si el programa ocupa más de una página, entonces hay que prestar especial atención a los saltos entre páginas y llamadas a subrutinas que estén en una página diferente a la actual, actualizando convenientemente los bits PCLATH<4:3> antes de realizar el salto.

3.2.1.2 Lectura y escritura de la memoria de programa

En el repertorio de instrucciones de los microcontroladores PIC de gama media, no existen instrucciones para leer la memoria de programa. Esto hace

que, en principio, esta memoria sea para almacenar solamente instrucciones y no datos fijos. Sin embargo, puede resultar conveniente y útil tener la posibilidad de leer datos almacenados permanentemente en esta memoria, en forma de tablas, textos en caracteres ASCII, etc. Algunos PIC dan esta posibilidad al permitir la lectura de la memoria de programa en forma indirecta, es decir, a través de algunos registros de funciones especiales de la memoria de datos. Más aún, algunos microcontroladores con memoria de programa FLASH permiten también escribir en esta memoria, usando igualmente algunos registros de funciones especiales. Por ejemplo, en los microcontroladores PIC16C781 y PIC16C782 es posible leer la memoria de programa. En la familia PIC16F87x, cuya memoria de programa es de tipo FLASH, es posible leer y escribir en la memoria de programa.

En la tabla 3.3 se muestran los registros de funciones especiales utilizados en la lectura y lectura/escritura de la memoria de programa. En los registros PMDATH:PMDATA o EEDATH: EEDATA se lee y/o escribe el dato de 14 bits; en PMADRH:PMADR o EEADRH:EEADR se ponen los 13 bits de la dirección del dato y los registros PMCON1 o EECON1 y EECON2 son los registros de control.

Tabla 3.3 *Registros de funciones especiales (SFR) utilizados en la lectura o lectura/escritura de la memoria de programa, en aquellos PIC que tienen implementada esta posibilidad.*

SFR utilizados en los dispositivos que permiten solamente la lectura de la memoria de programa		SFR utilizados en los dispositivos que permiten la lectura y escritura de la memoria de programa	
PMADRH	Dirección (13 bits): PMADRH:PMADR	EEADRH	Dirección (13 bits): EEADRH:EEADR
PMDATH	Dato (14 bits): PMDATH:PMDATA	EEDATH	Dato (14 bits): EEDATH:EEDATA
PMDATA		EEDATA	
PMCON1	Control	EECON1	Control
		EECON2	

El procedimiento para leer la memoria de programa es como sigue:

1. Colocar la dirección de la celda de memoria en los registros de funciones especiales PMADRH:PMADR (o EEADRH:EEADR, según el tipo de dispositivo). En los PIC que permiten la lectura/escritura, hay que poner el bit EEPGD del registro EECON1 a 1, para indicar que se accederá a la memoria de programa y no a la EEPROM de datos.
2. Poner el bit RD del registro PCON1 (o EECON1) a 1. Con esto se inicia la lectura de la celda de la memoria de programa. La lectura tiene lugar en dos ciclos de instrucción, durante los cuales, aunque el PC se incrementa en 2, no se buscan instrucciones. Por lo tanto, en el programa hay que po-

ner dos instrucciones a continuación de la que pone el bit RD a 1, aunque no se vayan a ejecutar. Se recomienda utilizar dos instrucciones de no operación (nop).

3. Cuando se completa la operación de lectura, el bit RD se pone automáticamente a 0. Además, el bit EEIF del registro PIR2 se pone a 1, indicando el fin de la operación de lectura.
4. El dato contenido en la celda leída puede tomarse de los registros PMDATA:TH:PMDATA (o EEDATH:EEDATA).

Ejemplo 3.1

A continuación se muestra el listado de un segmento de programa que ilustra la lectura de la memoria de programa de un microcontrolador PIC16F673, en el cual se puede leer y escribir la memoria de programa:

```

bsf      STATUS, RP1          ; Seleccionar el banco 2.
bcf      STATUS, RP0
movf    DIR_H, W             ; Escribir la dirección en
movwf   EEADRH               ; EEADRH:EEADR.
movf    DIR_L, W             ;
movwf   EEADR
bsf      STATUS, RP0          ; Seleccionar el banco 3.
bsf      EECON1, EEPGD        ; Seleccionar memoria de programa.
bsf      EECON1, RD           ; Comenzar la lectura.
nop
nop
bcf      STATUS, RP0          ; Seleccionar el banco 2 pues
                                ; el dato está listo en EEDATH:EEDATA.
movf    EEDATH, W             ; Leer el dato.
movwf   DATO_H
movf    EEDATA, W
movwf   DATO_L

```

El programa toma la dirección de la celda de memoria de programa en los registros DIR_H:DIR_L y la coloca en EEADRH:EEADR, usando como trampolín el registro W. La lectura se activa poniendo el bit RD de EECON1 a 1. La lectura se realiza en dos ciclos de instrucción; por eso se incluyen las dos instrucciones nop. Finalmente, el dato leído se toma de los registros de funciones especiales EEDATH:EEDATA y se coloca en los registros DATO_H:DATO_L.

Obsérvese que antes de acceder a los registros de funciones especiales hay que seleccionar el banco correcto mediante los bits RP1 y RP0 del SFR STATUS.

En dispositivos de sólo lectura, hay que utilizar los registros de funciones especiales PM y eliminar la instrucción bsf EECON1, EEPGD.

Para escribir en la memoria de programa de tipo FLASH, el procedimiento es el siguiente:

1. Colocar la dirección de la celda de memoria en los registros de funciones especiales EEADRH:EEADR y el dato que se desea escribir en EEDATH: EEDATA.
2. Poner el bit EEPGD del registro EECON1 a 1, para indicar que se accederá a la memoria de programa y no a la EEPROM de datos.
3. Poner el bit WREN del SFR EECON1 a 1 para habilitar la escritura de la memoria de programa.
4. Inhabilitar todas las interrupciones.
5. Escribir 55h en EECON2.
6. Escribir AAh en EECON2.
7. Poner el bit WR de ECON1 a 1. Con esto comienza la escritura, que tiene lugar en los siguientes dos ciclos de instrucción.
8. Colocar en el programa dos instrucciones de no operación (nop).
9. Poner el bit WREN del SFR EECON1 a 0 para inhabilitar la escritura de la memoria FLASH. Esto evita escrituras accidentales no deseadas de la memoria de programa.
10. Al finalizar la escritura, el bit WR de EECON1 se pone automáticamente a 0 y el bit EIF del SFR PIR2 se pone a 1, indicando que ha finalizado la escritura.
11. Habilitar las interrupciones.

Los pasos del 5 al 7 constituyen una secuencia de cinco instrucciones que deben ser ejecutadas sin interrupción. El paso 10 es una medida de seguridad que evita escrituras accidentales no deseadas en la memoria de programa.

Ejemplo 3.2

A continuación se muestra el listado de un segmento de programa que ilustra el procedimiento recomendado para escribir en la memoria FLASH de programa de un microcontrolador PIC16F873.

```

bsf      STATUS, RP1           ; Seleccionar el banco 2.
bcf      STATUS, RP0
movf    DIR_H, W              ; Escribir la dirección en
movwf   EEADRH                ; EEADRH:EEADR.
movf    DIR_L, W
movwf   EEADR

```

movf	DATO_H, W	; Escribir el dato en
movwf	EEDATH	; EEDATH:EEDATA.
movf	DATO_L, W	
movwf	EEDATA	
bsf	STATUS, RP0	; Seleccionar el banco 3.
bsf	EECON1, EEPGD	; Seleccionar memoria de programa y
bsf	EECON1, WREN	; Habilitar la escritura de la flash.
bcf	INTCON, GIE	; Inhabilitar todas las interrupciones.
movlw	55h	; Secuencia requerida.
movwf	EECON2	
movlw	AAh	; Secuencia requerida.
movwf	EECON2	
bsf	EECON1, WR	; Comenzar escritura en la flash.
nop		; Secuencia requerida mientras
nop		; se escribe en la flash.
bcf	EECON1, WREN	; Inhabilitar escritura de la memoria.
bsf	INTCON, GIE	; Habilitar las interrupciones (opcional).

Obsérvese que, antes de acceder a los registros de funciones especiales, hay que seleccionar el banco correcto mediante los bits RP1 y RP0 del SFR STATUS. Las interrupciones se habilitan o inhabilitan con el bit GIE del SFR INTCON.

3.2.2 La memoria RAM de datos

La memoria RAM de datos (o simplemente la memoria de datos) de los microcontroladores PIC está organizada en palabras de 8 bits. Del mismo modo que la memoria de programa, la memoria de datos también está paginada, aunque a las páginas de la memoria de datos se las llama *bancos*. Cada banco (*bank*) contiene hasta 128 localizaciones de memoria o registros, cuyas direcciones van desde 00h a 7Fh dentro de cada banco.

Todos los microcontroladores de gama media tienen al menos 2 bancos de registros, es decir, hasta 256 registros con las direcciones absolutas 00h a FFh. Algunos microcontroladores pueden tener hasta 4 bancos de registros, con un total de 512 posibles registros, con las direcciones absolutas 000h a 1FFh.

Los registros de la memoria de datos se clasifican en registros de funciones especiales (SFR: *Special Function Registers*) y registros de propósito general (GPR: *General Purpose Registers*). Los SFR son los registros a través de los cuales se controla el microcontrolador y se accede a sus diferentes periféricos, se programan sus funciones, etc. (apartado 3.2.2.2). Los GPR constituyen la memoria de datos propiamente dicha, disponible para el libre uso del usuario en sus programas.

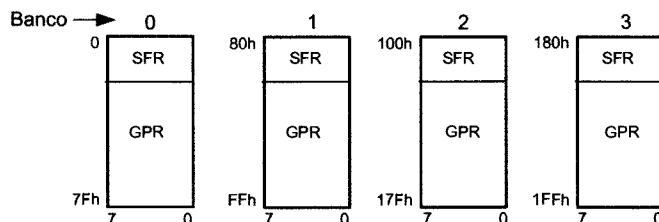


Figura 3.9 Paginado de la memoria de datos en los microcontroladores PIC de gama media. Puede haber hasta 4 bancos de 128 registros cada uno, con un total de 512 registros. En cada banco hay registros de funciones especiales (SFR) y de propósito general (GPR). Cada registro es de 8 bits.

La cantidad de SFR y GPR varía según el modelo de microcontrolador PIC. Como ejemplo, en las figuras 3.10 y 3.11 se muestran los registros de la memoria de datos de los microcontroladores PIC16F84 y PIC16F873, que cuentan con 14 FSR y 68 GPR el primero, y 50 SFR y 192 GPR el segundo. Obsérvese que para facilitar la programación, se puede acceder a muchos registros de funciones especiales desde cualquier banco de memoria.

	Banco 0	Banco 1	
00h	INDF(*)	80h	INDF(*)
01h	TMR0	81h	OPTION
02h	PCL	82h	PCL
03h	STATUS	83h	STATUS
04h	FSR	84h	FSR
05h	PORATA	85h	TRISA
06h	PORTB	86h	TRISB
07h		87h	
08h	EEDATA	88h	EECON1
09h	EEADR	89h	EECON2(*)
0Ah	PCLATH	8Ah	PCLATH
0Bh	INTCON	8Bh	INTCON
0Ch		8Ch	GPR mapeados en el banco 0
4Fh	68 GPR	CFh	
50h		D0h	
7Fh		FFh	

Figura 3.10 Registros de la memoria de datos del microcontrolador PIC16F84. Está organizada en dos bancos, con 68 registros de propósito general (GPR) a los que se puede acceder indistintamente desde cualquiera de los dos bancos y 14 registros de funciones especiales (SFR). Algunos SFR como STATUS, PCLATH, PCL, FSR e INTCON son accesibles desde cualquier banco. La primera celda de cada banco (INDF) se usa en el direccionamiento indirecto de datos y no es un registro real del microcontrolador.

■ Celdas no implementadas, se leen en 0.

(*) No es un registro físico

Banco 0	Banco 1	Banco 2	Banco 3
00h INDF(*)	80h INDF(*)	100h INDF(*)	180h INDF(*)
01h TMRO	81h OPTION	101h TMRO	181h OPTION
02h PCL	82h PCL	102h PCL	182h PCL
03h STATUS	83h STATUS	103h STATUS	183h STATUS
04h FSR	84h FSR	04h FSR	184h FSR
05h PORTA	85h TRISA	105h PORTB	185h TRISB
06h PORTB	86h TRISB	106h PORTC	186h TRISC
07h PORTC	87h TRISC	107h PORTD(*)	187h TRISD(*)
08h PORTD(*)	88h TRISD(*)	108h PORTE(*)	188h TRISE(*)
09h PORTE(*)	89h TRISE(*)	109h PCLATH	189h PCLATH
0Ah PCLATLH	8Ah PCLATLH	10Ah INTCON	18Ah INTCON
0Bh INTCON	8Bh INTCON	10Bh EEDATA	18Bh EECON1
0Ch PIR1	8Ch PIE1	10Ch EEADR	18Ch EECON2
0Dh PIR2	8Dh PIE2	10Dh EEDATH	18Dh ()
0Eh TMR1L	8Eh PCON	10Eh EEADRH	18Fh ()
0Fh TMR1H	8Fh	10Fh 190h	190h
10h T1CON	90h		
11h TMR2	91h SSPCON2		
12h T2CON	92h PR2		
13h SSPBUF	93h SSPADD		
14h SSPCON	94h SSPSTAT		
15h CCPR1L	95h		
16h CCPR1H	96h		
17h CCP1CON	97h		
18h RCSTA	98h TXSTA		
19h TXREG	99h SPBRG		
1Ah RCREG	9Ah		
1Bh CCPR2L	9Bh		
1Ch CCPR2H	9Ch		
1Dh CCP2CON	9Dh		
1Eh ADRESH	9Eh ADRESL		
1Fh ADCON0	9Fh ADCON1	11Fh 19Fh	
20h	A0h	120h 1A0h	
7Fh 96 GPR	FFh 96 GPR	17Fh GPR mapeados en Banco 0	1FFh GPR mapeados en Banco 1

■ Celdas no implementadas, se leen en 0.

(*) No es un registro físico

(!) No implementado en el PIC16F873

(?) Reservado

Figura 3.11 Registros de la memoria de datos en el PIC16F873. Está organizada en 4 bancos, con 192 registros de propósito general y 50 registros de funciones especiales.

3.2.2.1 Direccionamiento de la memoria de datos.

Para formar la dirección de un dato almacenado en la memoria de datos se necesitan 9 bits de dirección. Como la memoria está paginada, los bits 8 y 7 de la dirección dan el número del banco, y los bits 6 al 0 dan la dirección dentro del banco (desplazamiento), según muestra la figura 3.12. Los dos bits que identifican el banco los aporta el registro de funciones especiales STATUS, mientras que el desplazamiento puede estar en la instrucción (direcccionamiento directo) o en el registro de funciones especiales FSR (direcccionamiento indirecto).

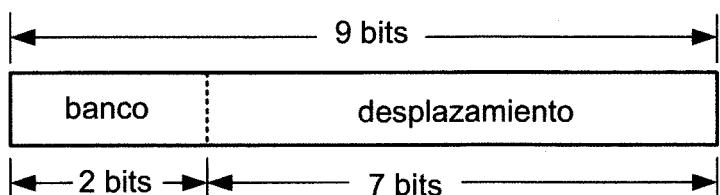


Figura 3.12 Componentes de una dirección en la memoria de datos. Se requieren 9 bits: los 2 bits más significativos dan el número del banco y los 7 bits de menor peso constituyen el desplazamiento o dirección dentro del banco.

A todos los registros de la memoria de datos se accede utilizando direccionamiento directo o indirecto (figura 3.13). Si se utiliza direccionamiento directo, el banco con el que se está trabajando se selecciona con RP1 y RP0, que son los bits 6 y 5 del registro de funciones especiales STATUS, respectivamente. En la instrucción se maneja el desplazamiento de 7 bits, es decir, una dirección desde 00h a 7Fh. Si el microcontrolador dispone de sólo 2 bancos, entonces basta con manipular el bit RP0, pues en este caso se ignora el bit RP1.

Si se usa direccionamiento indirecto, los 8 bits menos significativos de la dirección se toman del registro de funciones especiales FSR (*File Select Register*) y el noveno bit es IRP, es decir, el bit 7 del registro STATUS. En este caso el desplazamiento de 8 bits está en el registro FSR. En los dispositivos que tienen sólo 2 bancos, el bit IRP debe mantenerse a 0. En los dispositivos con 4 bancos de registros, IRP = 0 selecciona los bancos 0 y 1, mientras que IRP = 1 selecciona los bancos 2 y 3. A efectos del direccionamiento indirecto, se puede considerar que la memoria de datos está paginada en dos bancos de 256 direcciones cada uno, en lugar de los cuatro bancos de 128 direcciones utilizados en el direccionamiento directo.

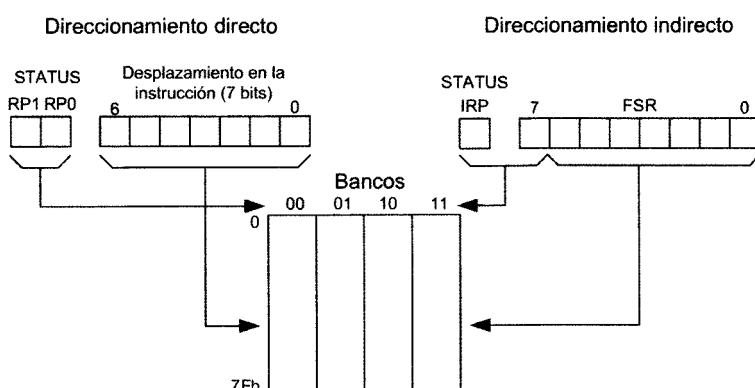


Figura 3.13 Direccionamiento de la memoria de datos. Se puede acceder a todos los registros de la memoria mediante direccionamiento directo o indirecto.

3.2.2.2 Registros de funciones especiales

Los registros de funciones especiales (SFR: *Special Function Registers*) son registros situados en la memoria de datos que tienen funciones específicas informativas o de control. Estas funciones pueden estar asociadas al funcionamiento propio del microcontrolador o a sus periféricos. Los SFR cuya función está asociada al núcleo del microcontrolador son más o menos iguales en todos los microcontroladores; en cambio, los SFR relacionados con el trabajo de los periféricos dependen mucho del dispositivo concreto de que se trate.

La tabla 3.4 muestra los nombres de los SFR asociados a diferentes funciones y periféricos de microcontroladores PIC de gama media. Los registros STATUS, PCLATH, PCL, FSR, OPTION, INTCON, PORTA, PORTB, TRISA, TRISB Y TMR0 están presentes en la mayoría de ellos. En aquellos dispositivos en los que no está implementado un determinado periférico, no existe el SFR correspondiente.

Tabla 3.4 *Registros de funciones especiales asociados a funciones del microcontrolador o de sus periféricos.*

Función o dispositivo	SFR
Selección del banco de memoria. Indicadores relacionados con las operaciones aritméticas y lógicas. Desbordamiento del perro guardián. Indicador de bajo consumo	STATUS
Valor del pre-divisor. Flancos de los pulsos de reloj. Flanco de la solicitud de interrupción externa. Pull-up interno del puerto B	OPTION
Indicadores de error de paridad en memoria. Tipo de reset. Bajo consumo	PCON
Contador de programa	PCLATH, PCL
Direccionamiento indirecto	FSR
Interrupciones	INTCON PIR1, PIE1 PIR2, PIE2
Puertos paralelos	PORTA, TRISA PORTB, TRISB PORTC, TRISC PORTD, TRISD PORTE, TRISE
Timer0	TMR0, OPTION, INTCON
Timer1	TMR1H, TMR1L, T1CON, PIR1

Timer2	TMR2, PR2, T2CON, PIR1
Módulos CCPx ($x = 1, 2, 3$)	CCPRxH, CCPRxL, CCPxCON
Puerto serie USART o SCI	TXREG, TXSTA, RCREG, RCSTA
Puerto serie sincrónico SSP	SSPSTAT, SSPCON, SSPBUF, SSPADD
Convertidor A/D	ADRESH, ADRESL, ADCON0, ADCON1
Memoria EEPROM de datos y FLASH de programa	EEADRH, EEADR, EEDATH, EEDATA, EECON1, EECON2

El registro STATUS

El registro STATUS contiene básicamente los bits de estado asociados a las operaciones aritméticas y los bits de selección del banco de memoria de datos que se va a utilizar. Contiene además dos bits indicadores del estado del perro guardián y el modo de bajo consumo. A este registro de funciones especiales se puede acceder desde cualquier banco de memoria. La presencia del registro STATUS en todos los bancos se debe a que contiene los bits con los que se selecciona el banco de la memoria de datos, por lo que debe estar accesible desde cualquier banco.

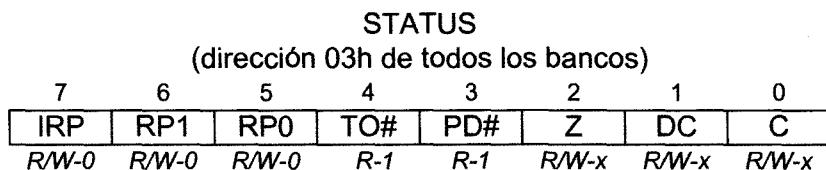


Figura 3.14 Registro STATUS. A este registro se puede acceder en la dirección 03h de cualquier banco de la memoria de datos. Se muestran los valores que toman los bits del registro después de un reset, y si se pueden leer y escribir (R/W) o si son de sólo lectura (R).

La figura 3.14 muestra los bits que componen el registro STATUS. Su significado es el siguiente:

IRP. Selección del banco de memoria de datos en direccionamiento indirecto. Con este bit se selecciona el banco de memoria de datos en el direccionamiento indirecto. 0: bancos 0 y 1, 1: bancos 2 y 3.

RP1, RP0. Selección del banco de memoria de datos en direccionamiento directo. Con estos bits se selecciona el banco de memoria de datos en el direccionamiento directo. 00: banco 0, 01: banco 1, 10: banco 2 y 11: banco 3.

TO#. Indicador de desbordamiento del perro guardián (WDT: *Watchdog Timer*). Este bit se pone a 0 cuando se desborda el WDT. Se pone a 1 con un *reset* por encendido y con las instrucciones `clrwdt` y `sleep`.

PD#. Indicador de modo de bajo consumo (*power-down*). Este bit se pone a 0 cuando el microcontrolador entra en modo de bajo consumo con la instrucción `sleep`. Se pone a 1 con el encendido y con la instrucción `clrwdt`.

Z. Indicador de cero. Se pone a 1 si el resultado de una operación aritmética o lógica es cero; en caso contrario se pone a 0.

DC. Indicador de acarreo auxiliar. Se pone a 1 cuando hay un acarreo del bit 3 al 4 en una operación aritmética de suma binaria y a 0 si no hay acarreo. En una operación de resta, se pone a 0 si hay préstamo del bit 4 al 3 y a 1 si no lo hay.

C. Indicador de acarreo. Se pone a 1 cuando hay un acarreo del bit 7 al 8 en una operación aritmética de suma binaria y a 0 si no hay acarreo. En una operación de resta, se pone a 0 si hay préstamo del bit 8 al 7 y a 1 si no lo hay.

Si el registro STATUS es el destino de una instrucción, el valor resultante en STATUS puede no coincidir con el valor que se supone será escrito por la instrucción. Hay que tener en cuenta que los bits TO# y PD# son de sólo lectura y por tanto no son modificables por escritura. Los bits Z, DC y C, tomarán el valor que corresponda según la lógica de la instrucción y no el valor que se supone será escrito por la instrucción.

Por ejemplo, la instrucción `clrf STATUS` no pone 00h en STATUS, sino que dejará TO# y PD# sin modificar; igualmente quedarán sin modificar los bits DC y C, y el bit Z irá a 1 siguiendo la lógica de la instrucción. Es decir, el valor resultante en STATUS será 000uu1uu, donde *u* significa que ese bit no se altera.

Si se quiere modificar bits del registro STATUS, se recomienda utilizar instrucciones que no alteren los bits Z, DC o C, como son las instrucciones `bcf`, `bsf`, `swapf` y `mowwf`.

El registro OPTION

En el registro OPTION (también denominado OPTION_REG) (figura 3.15) están los bits para controlar funciones relacionadas con la habilitación o no del *pull-up* interno del puerto B, la selección del flanco con que se reconocerá la interrupción externa, la fuente de pulsos del Timer0, la selección del valor del pre-divisor del Timer0 y del perro guardián (WDT).

OPTION (dirección 01h de los bancos 1 y 3)							
7	6	5	4	3	2	1	0
RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1

Figura 3.15 Registro OPTION. Se muestran los valores que toman los bits del registro después de un reset. Todos los bits se pueden leer y escribir (R/W).

RBPU#. Pull-up interno del puerto B. Un 1 en este bit inhabilita el pull-up interno del puerto B. Con un 0 se habilita la puesta individual de los pull-up para cada bit del puerto B.

INTEDG. Flanco de la interrupción externa. Con un 1, la interrupción externa se produce con el flanco de subida, mientras que con un 0 se produce con el flanco de caída.

T0CS. Fuente de reloj del Timer0. Con este bit se selecciona la fuente de los pulsos de reloj del Timer0. Un 1 selecciona como fuente los pulsos que lleguen por el terminal T0CKI y un 0 selecciona como fuente el reloj interno (dividido por 4).

T0SE. Flanco del reloj del Timer0. Un 1 hace que el Timer0 se incremente con los flancos de caída de los pulsos de reloj y un 0 hace que se incremente con los flancos de subida.

PSA. Asignación del pre-divisor. Un 1 asigna el pre-divisor al WDT y un 0 lo asigna al Timer0.

PS2, PS1, PS0. Valor del pre-divisor:

PS2	PS1	PS0	Pre-divisor asignado al Timer0, divide entre:	Pre-divisor asignado al WDT, divide entre:
0	0	0	2	1
0	0	1	4	2
0	1	0	8	4
0	1	1	16	8
1	0	0	32	16
1	0	1	64	32
1	1	0	128	64
1	1	1	256	128

3.2.3 La memoria EEPROM de datos.

Muchos de los microcontroladores PIC de gama media que utilizan memoria de programa FLASH, ofrecen también hasta un máximo de 256 bytes de memoria no volátil de datos o memoria EEPROM de datos. Esta memoria se encuentra en un espacio separado de la memoria RAM de datos y se acce-

de a ella a través de algunos registros de funciones especiales, los mismos que se utilizan para leer y escribir en la memoria FLASH de programa. La tabla 3.5 muestra estos registros y su función en el manejo de la memoria EEPROM de datos.

Tabla 3.5 *Registros de funciones especiales utilizados en el manejo de la memoria EEPROM de datos.*

Registro	Función
EEADR	Contiene la dirección del dato (8 bits)
EEDATA	Contiene el dato (8 bits)
EECON1	
EECON2	Control

El procedimiento para leer un dato almacenado en la memoria EEPROM de datos es el siguiente:

1. Colocar la dirección de la celda de memoria en EEADR.
2. Poner el bit EEPGD del registro EECON1 a 0, para indicar que se accederá a la memoria EEPROM de datos y no a la memoria de programa.
3. Poner el bit RD del registro EECON1 a 1. Con esto se inicia la lectura. El dato estará disponible en el registro EEDATA en el siguiente ciclo de instrucción.
4. Al completarse la operación de lectura, el bit RD se pone automáticamente a 0. Además, el bit EEIF del registro PIR2 se pone a 1, indicando el fin de la operación de lectura.

Ejemplo 3.3

A continuación se muestra el listado de un segmento de programa que ilustra el procedimiento recomendado para leer un dato en la memoria EEPROM de datos de un microcontrolador PIC16F873

```

bsf      STATUS, RP1          ; Seleccionar el banco 2.
bcf      STATUS, RP0          ;
movf    DIREC, W             ; La dirección de la EEPROM está en DIREC
movwf   EEADR                ; y se coloca en EEADR.
bsf      STATUS, RP0          ; Seleccionar el banco 3.
bcf      EECON1, EEPGD         ; Seleccionar memoria de datos y
bsf      EECON1, RD            ; comenzar la lectura.
bcf      STATUS, RP0          ; Seleccionar banco 1 pues el dato está listo
movf    EEDATA, W             ; en EEDATA. Moverlo a W.

```

El programa toma la dirección de la celda de la memoria EEPROM de datos en el registro DIREC (que ha

movwf	EEDATA	; EEDATA.
bsf	STATUS, RP0	; Seleccionar el banco 3.
bcf	EECON1, EEPGD	; Seleccionar memoria EEPROM de datos y
bcf	EECON1, WREN	; habilitar la escritura de la flash.
bcf	INTCON, GIE	; Inhabilitar todas las interrupciones.
movlw	55h	; Secuencia requerida.
movwf	EECON2	
movlw	AAh	; Secuencia requerida.
movwf	EECON2	
bsf	EECON1, WR	; Comenzar escritura en la flash de programa.
bcf	EECON1, WREN	; Inhabilitar escritura en la EEPROM de datos.
esperar:		
btfsC	EECON1, WR	; ¿WR = 0 ? sí – continuar,
goto	esperar	; no – esperar.
bsf	INTCON, GIE	; Habilitar las interrupciones (opcional).

Antes de acceder a los registros de funciones especiales hay que seleccionar el banco correcto mediante los bits RP1 y RP0 del SFR STATUS. Las interrupciones se habilitan o inhabilitan con el bit GIE del SFR INTCON.

4 Repertorio de instrucciones y programación en lenguaje ensamblador

Este capítulo trata sobre el diseño de programas en el lenguaje ensamblador de los microcontroladores PIC. Primero se exponen algunos conceptos básicos tales como lenguaje de máquina, lenguaje ensamblador, programa fuente, programa objeto, etc. A continuación se estudia el repertorio de instrucciones de los microcontroladores PIC de gama media y se explican los elementos de la programación en el lenguaje ensamblador, utilizando numerosos ejemplos. Finalmente se describe el funcionamiento de algunos recursos disponibles para desarrollar programas en lenguaje ensamblador usando un ordenador personal.

4.1 Conceptos básicos

4.1.1 Código de máquina y lenguaje ensamblador

Cualquier microprocesador o microcontrolador ejecuta las instrucciones de un programa en su *lenguaje de máquina*. El lenguaje o código de máquina es el constituido por los códigos binarios de las instrucciones que puede ejecutar el microcontrolador; es, por tanto, un lenguaje binario, de “unos” y “ceros”. En el lenguaje de máquina, cada instrucción de un programa está formada por un grupo de dígitos binarios. Por ejemplo, todas las instrucciones del lenguaje de máquina de los microcontroladores PIC de gama media tienen 14 bits de longitud. Por lo tanto, un programa en lenguaje de máquina para estos microcontroladores está formado por palabras de 14 bits cada una.

Obviamente, elaborar programas directamente en el lenguaje de máquina es difícil. Para reducir el nivel de dificultad de la programación a este “bajo nivel”, se crearon los *lenguajes ensambladores*, en los cuales las instrucciones que en el lenguaje de máquina se representan por grupos de bits, son representadas por símbolos mnemotécnicos.

Ejemplo 4.1

La instrucción “poner 0 en el registro W” en el lenguaje de máquina se representa por la palabra

00 0001 0xxx xxxx

donde x puede ser indistintamente 0 ó 1.

En lenguaje ensamblador, esta operación se representa con el mnemotécnico “**clr w**” (**clear W register**), que es mucho más fácil de recordar que el número binario del lenguaje de máquina.

Ejemplo 4.2

La instrucción “poner el valor K en el registro W ”, siendo K un número binario de 8 bits, se representa en el lenguaje de máquina por la palabra binaria:

11 00xx kkkk kkkk

donde k son los dígitos binarios del dato y x puede ser indistintamente 0 ó 1.

En lenguaje ensamblador, la operación “poner un dato de 8 bits (dado en la propia instrucción) en el registro W ” se representa con el mnemotécnico “**movlw**” (*move literal to W*). La instrucción completa es entonces:

movlw K

que es mucho más fácil de manejar que la instrucción en lenguaje de máquina.

Los lenguajes ensambladores, como los lenguajes de máquina, son muy particulares de cada microprocesador o microcontrolador. Cada tipo de microcontrolador tiene su propio lenguaje ensamblador. Los PIC de gama media tienen un lenguaje ensamblador compuesto por 35 instrucciones.

Un programa escrito en lenguaje ensamblador no se puede ejecutar directamente en el microcontrolador; es necesario “traducirlo” al lenguaje de máquina. Este proceso se denomina *ensamblaje* y lo realiza un programa denominado *ensamblador*, aunque pueden participar, además, otros programas, según se explica a continuación. El programa original escrito en lenguaje ensamblador se denomina *programa fuente* y el resultado del ensamblaje es el *programa objeto*. El proceso se realiza normalmente en un ordenador personal.

La obtención del código de máquina a partir del programa fuente escrito en lenguaje ensamblador se puede realizar de diferentes formas. En principio, la operación del ensamblador depende de la forma en que ha sido escrito el programa fuente. En particular, es determinante si en el programa fuente se precisan o no las direcciones reales donde se deben ubicar las instrucciones y los datos. Si en el programa fuente se precisan las direcciones reales de instrucciones y datos, el ensamblador genera *código absoluto* a partir del programa fuente; en caso contrario, genera *código relativo o relocable*. La figura 4.1 ilustra ambas modalidades del proceso de ensamblaje.

Si en el programa fuente está toda la información necesaria para la traducción al lenguaje de máquina, es decir, si están definidas las direcciones reales de las instrucciones del programa y las variables utilizadas y las direcciones que ellas van a ocupar en la memoria de datos, entonces el ensamblador puede generar directamente el programa objeto en código de máquina, pues dispone de toda la información necesaria. En este caso, el ensamblador

realiza una codificación absoluta del programa fuente, produciendo directamente el programa en lenguaje de máquina.

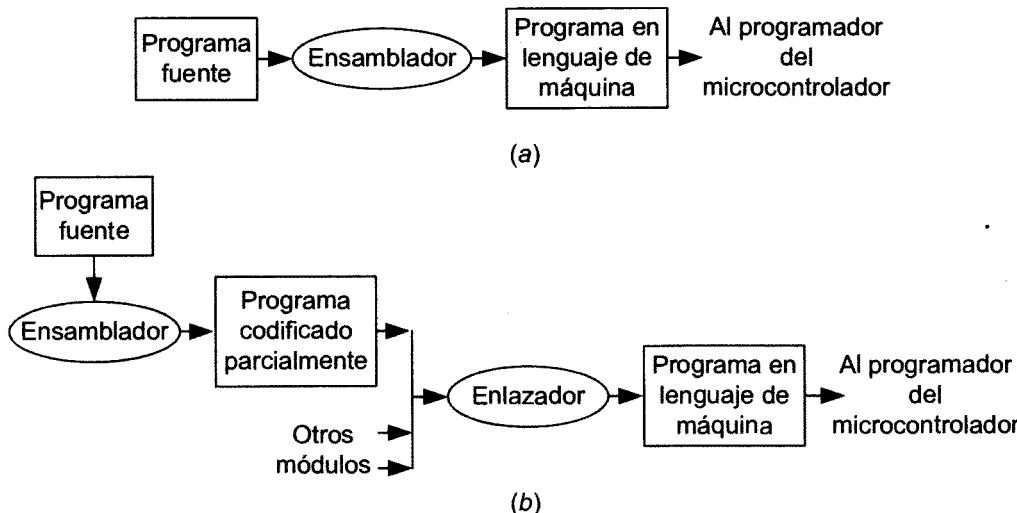


Figura 4.1 Proceso de traducción al lenguaje de máquina de un programa escrito en lenguaje ensamblador. En (a) el ensamblador genera directamente el programa objeto en lenguaje de máquina. En (b) el ensamblador realiza una codificación intermedia del programa fuente y es el enlazador el que genera el programa objeto en lenguaje de máquina a partir de los diferentes módulos.

La escritura de programas para su codificación absoluta por el ensamblador resulta relativamente simple cuando se trabaja en proyectos cortos y poco complejos. Pero si los programas son extensos y complejos, esta forma de programar no resulta ni flexible ni práctica.

En trabajos de cierta envergadura, es conveniente trabajar por módulos, lo que significa desarrollar los programas de la aplicación en módulos separados que luego se unen para obtener el programa objeto. Al escribir cada módulo en lenguaje ensamblador, puede ser conveniente no precisar las direcciones reales o absolutas de las instrucciones y datos. Entonces, el ensamblador sólo puede realizar una codificación parcial e incompleta de cada módulo fuente, pues falta la información de las direcciones; en este caso se dice que el ensamblador genera código relocable. El programa que se ocupa de enlazar los módulos entregados por el ensamblador, poner las direcciones reales de cada instrucción o dato y generar finalmente el programa objeto en código de máquina, se denomina *enlazador* (*linker*).

Esta forma de organizar la programación, mucho más flexible y potente que la primera, exige que el programa fuente en lenguaje ensamblador sea

escrito de forma diferente a como se escribiría si se quisiera generar código absoluto. Cuando se escribe el programa fuente en lenguaje ensamblador para codificación absoluta, la traducción al lenguaje de máquina requiere sólo el programa ensamblador. Si el programa se escribe sin precisar direcciones reales, el ensamblador producirá una codificación incompleta y el enlazador completará esta tarea poniendo las direcciones reales de datos e instrucciones. En este caso, la traducción al lenguaje de máquina se realiza en dos pasos y requiere el ensamblador y el enlazador.

La organización modular de la programación y el uso de un enlazador en el proceso de obtención del programa objeto en lenguaje de máquina proporcionan ventajas adicionales. Una de ellas es que se pueden enlazar módulos escritos originalmente en lenguajes de programación diferentes (ensamblador, C, etc.). Algunos módulos pueden constituir una biblioteca de programas. Por ejemplo, las subrutinas para operaciones aritméticas en coma flotante, para un tipo de microcontrolador, se pueden agrupar en un módulo. Al enlazar con otros módulos, el enlazador toma de la biblioteca sólo las subrutinas solicitadas y las incorpora en el programa objeto. Todas estas facilidades confieren a la organización modular de la programación una potencia extraordinaria.

4.1.2 Estructura de las instrucciones

En principio, las instrucciones de cualquier microprocesador o microcontrolador tienen dos componentes: el *código de operación* y los *operando*s. En el código de operación está codificada la orden para que el dispositivo ejecute la acción indicada por la instrucción. Los operandos son los datos necesarios para realizar esa acción y, en general, pueden representar direcciones o datos.

En los microcontroladores PIC de gama media, un operando pueden ser:

- Una dirección (de 7 bits) de la memoria de datos.
- Una dirección (de 11 bits) de la memoria de programa.
- Un dato de 8 bits.
- La dirección (de 3 bits) de un bit de un registro cualquiera de la memoria de datos.
- La indicación (con 1 bit) de cuál es el destino del resultado de la instrucción: si es el registro W o un registro de la memoria de datos.

En general, en los microprocesadores y microcontroladores hay instrucciones que no necesitan operandos y otras que trabajan con más de un operando. En los microcontroladores PIC de gama media hay instrucciones sin

operando e instrucciones con uno o dos operandos. En las instrucciones que tienen uno o dos operandos, las situaciones posibles son (figura 4.2):

- En las instrucciones que realizan operaciones con los registros de la memoria de datos, hay dos operandos: uno es la dirección de 7 bits del registro y el otro es el bit que indica cuál es el destino del resultado de la operación: el registro W o el registro cuya dirección se da en la instrucción.
- Las instrucciones que incluyen un dato de 8 bits en la propia instrucción, tienen como único operando ese dato.
- Las instrucciones que incluyen en la propia instrucción una dirección de 11 bits de la memoria de programa, tienen como único operando esa dirección.
- En las instrucciones que operan con bits de los registros de la memoria de datos, hay dos operandos: uno es la dirección del bit dentro del registro, lo cual requiere 3 bits, y el otro es la dirección de 7 bits del registro de la memoria de datos.

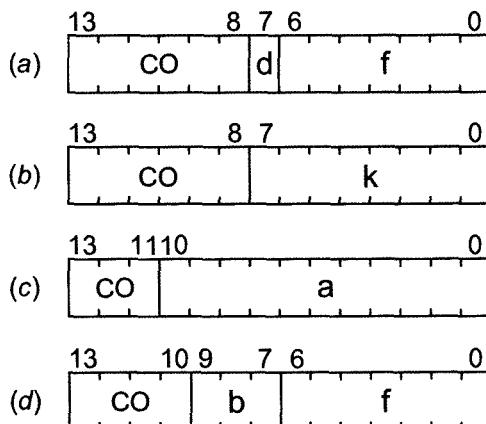


Figura 4.2 Formatos de las instrucciones en los microcontroladores PIC de gama media. Las instrucciones tienen un código de operación de 3 a 6 bits de longitud y hasta 2 operandos. (a) Formato de las instrucciones que operan con registros de la memoria de datos. (b) Instrucciones que operan con datos inmediatos de 8 bits. (c) Instrucciones que contienen direcciones de la memoria de programa. (d) Instrucciones que operan sobre un bit determinado de un registro de la memoria de datos. CO: código de operación, f: dirección de 7 bits del registro de la memoria de datos, k: dato de 8 bits, a: dirección de 11 bits, b: número del bit (0 a 7), d: destino del resultado de la instrucción: si d = 0, el destino es W; si d = 1, el destino es f.

En los PIC de gama media, todas las instrucciones tienen 14 bits, que es el tamaño de las celdas de la memoria de programa, de modo que cada ins-

trucción ocupa una única celda de memoria. Esta característica es propia de los microcontroladores y microprocesadores RISC.

4.1.3 Modos de direccionar los datos

Los modos de direccionar los datos se refieren a las distintas formas en que, desde una instrucción, se puede hacer referencia a un dato. Básicamente hay dos ubicaciones para un dato: la propia instrucción o la memoria de datos. En este último caso, la instrucción hace referencia al dato a través de su dirección.

En general, los microprocesadores y microcontroladores construyen la dirección de un dato de formas diversas, relacionadas con su arquitectura. En su forma más simple —que es la utilizada por los microcontroladores PIC de gama media—, la dirección de un dato puede estar en la instrucción o en un registro del dispositivo. Al registro que almacena direcciones de datos se le llama, genéricamente, “registro de direcciones de datos” (RDD). Por ello hay tres formas básicas de hacer referencia a un dato: con el dato en la instrucción, con la dirección del dato en la instrucción o con la dirección del dato en el registro RDD. Esto da lugar a tres modos de direccionar los datos: *direccionamiento inmediato*, *direccionamiento directo* y *direccionamiento indirecto*.

El direccionamiento es inmediato cuando el dato forma parte de la instrucción. El operando de la instrucción es el propio dato.

El direccionamiento es directo cuando la dirección del dato se da en la instrucción. El operando de la instrucción es la dirección del dato.

El direccionamiento es indirecto cuando la instrucción toma la dirección del dato en RDD. El operando de la instrucción es la dirección del RDD.

En los microcontroladores PIC de gama media, el registro de funciones especiales FSR (*File Select Register*) actúa como RDD. En estos microcontroladores se puede acceder a todos los registros de la memoria de datos utilizando direccionamiento directo o indirecto. Al utilizar los modos de direccionamiento directo e indirecto, hay que tener en cuenta que la memoria de datos se encuentra organizada en páginas o bancos de registros. Por ello, como paso previo, hay que seleccionar el banco en el que se encuentra el registro que se quiere direccionar. La selección del banco se realiza mediante los bits IRP, RP1 y RP0 del registro de funciones especiales STATUS (figura 3.13). En el direccionamiento directo, el banco de registros se selecciona con los bits RP1 y RP0. En el direccionamiento indirecto, el banco se selecciona con el bit IRP y el bit más significativo del registro de funciones especiales FSR, donde debe estar la dirección del registro que se quiere direccionar (apartado 3.2.2.1).

y figura 3.13). La lectura o escritura del dato en el registro apuntado por FSR se realiza al leer o escribir el dato en el falso registro INDF.

Los siguientes ejemplos ilustran cómo direccionar datos usando direccionamiento directo o indirecto, en un PIC de gama media.

Ejemplo 4.3

En un PIC16F84, colocar el valor 0x35 en el registro W.

Solución:

```
movlw 0x35
```

El modo de direccionamiento utilizado es inmediato pues el dato está en la propia instrucción.

Ejemplo 4.4

En un microcontrolador PIC16F873, colocar el valor 0x35 en el registro 0x20 del banco 1. Mostrar la solución al utilizar direccionamiento directo e indirecto.

Solución utilizando direccionamiento directo. En este caso se selecciona el banco 1 y se escribe el dato en el registro mediante la instrucción que tiene como operando la dirección (0x20) del registro:

bcf	STATUS, RP1	; Seleccionar el banco 1.
bsf	STATUS, RP0	
movlw	0x35	; Colocar el valor 0x35 en W y
movwf	0x20	; copiarlo en el registro con dirección 0x20 ; en el banco seleccionado.

Solución utilizando direccionamiento indirecto. Se selecciona el banco 1 al hacer IRP = 0 y colocar en FSR la dirección del registro (0x20) con el bit más significativo de FSR en 1, es decir, 0xA0. El dato 0x35 se coloca en el registro apuntado por FSR cuando se ejecuta la instrucción que escribe en el falso registro INDF:

bcf	STATUS, IRP	; Seleccionar los bancos 0 y 1.
movlw	0xA0	; Colocar la dirección del registro en W y
movwf	FSR	; copiarla en el registro FSR.
movlw	0x35	; Colocar el valor 0x35 en W y
movwf	INDF	; copiarlo en el registro apuntado por FSR, ; lo cual se indica mediante la escritura ; en el falso registro INDF.

4.1.4 La pila.

La *pila* es un tipo de estructura de datos con organización LIFO (*Last In First Out*): lo último que entra en la pila es lo primero que sale de ella.

La pila tiene una *base* y un *tope*. En la base de la pila está el elemento más antiguo, mientras que en el tope está el elemento guardado más recien-

temente. Está claro que cuando hay más de un elemento en la pila, todas las operaciones de almacenamiento o extracción se hacen sobre su tope. La profundidad de la pila es el tamaño que tiene la pila en un momento dado.

La pila de muchos microprocesadores y microcontroladores se ubica en su memoria de datos. Entonces el crecimiento de la pila es prácticamente ilimitado, pues se realiza en una zona de memoria RAM. Estos dispositivos tienen un registro para direccionar la pila: el denominado puntero de la pila o registro SP (*Stack Pointer*). El SP contiene siempre la dirección del tope de la pila. Las operaciones de almacenamiento o extracción de datos de la pila modifican el contenido del registro SP. Por ejemplo, al guardar un dato en la pila, el valor del SP se puede incrementar (o decrementar); al extraer el dato guardado, el SP se decremente (o incremente). La figura 4.3 ilustra la estructura general de la pila en un microprocesador o microcontrolador.

La pila se usa para almacenar direcciones de instrucciones y, en concreto, para “recordar” la dirección de retorno al programa principal desde una subrutina. Cuando se llama a una subrutina mediante una instrucción call o similar, el valor del contador de programa (PC), que no es otro que la dirección a la que se debe retornar al terminar la ejecución de la subrutina, se guarda en la pila. Al finalizar la subrutina con una instrucción return u otra similar, ésta toma el valor situado en el tope de la pila, que no es otro que la dirección colocada por la instrucción de llamada, y lo devuelve al PC, con lo que se produce, en efecto, el salto de retorno al programa desde el cual se llamó a la subrutina, exactamente en el punto correspondiente a la instrucción que sigue al call.

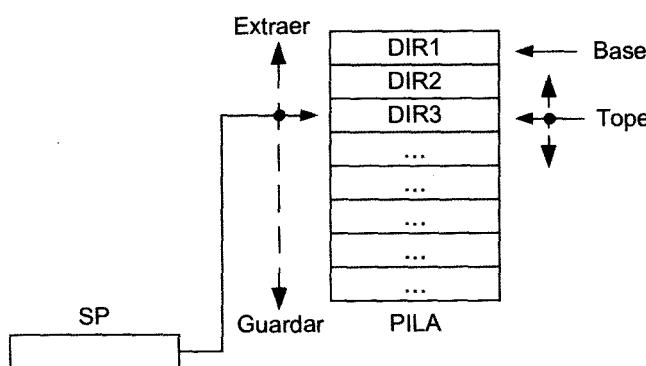


Figura 4.3 Estructura general de la pila en un microprocesador o microcontrolador. La pila se encuentra en una zona de la memoria RAM. El puntero de la pila (registro SP), apunta siempre al tope de la pila. El tope de la pila es variable pues la pila crece

cuando se guarda algún dato y disminuye cuando se extrae un dato. DIR1, DIR2 y DIR3 son direcciones almacenadas en la pila.

Este mecanismo de llamada-retorno funciona muy bien gracias a la organización LIFO de la pila. Además de ser muy útil en el manejo de subrutinas e interrupciones, la organización LIFO de la pila hace posible el anidamiento de subrutinas, es decir, la llamada a una subrutina desde otra subrutina (figura 4.4).

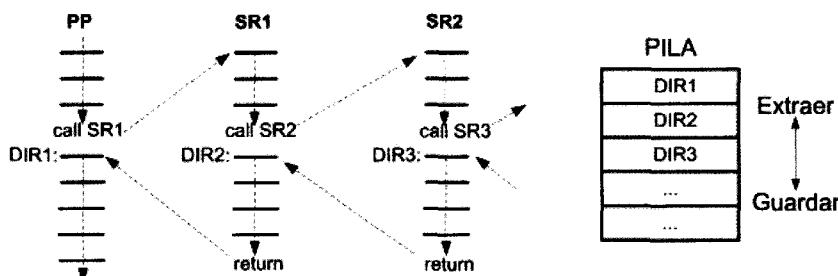


Figura 4.4 La pila y el anidamiento de subrutinas. Desde el programa principal (PP) se llama a la subrutina SR1, desde ésta a la subrutina SR2 y así sucesivamente. Con cada instrucción call, se guarda en la pila la dirección de retorno. Cada dirección de retorno (DIR1, DIR2, DIR3, ...) es el valor que tiene el contador de programa (PC) cuando se ejecuta la instrucción call correspondiente. La instrucción return con la que termina cada subrutina, extrae de la pila la dirección de retorno y la coloca en el PC. La última dirección almacenada en la pila es la primera en ser extraída, lo cual se acomoda perfectamente a la organización LIFO de la pila.

En los microcontroladores PIC de gama media, la pila tiene unas características muy particulares:

1. La pila se encuentra en un espacio de memoria separado de los espacios de memoria de programa y datos.
2. No existe el registro SP.
3. En la pila sólo se pueden guardar direcciones.
4. El tamaño de la pila es limitado: se pueden almacenar hasta 8 direcciones.

Teniendo en cuenta estas características, la pila en los microcontroladores PIC de gama media puede representarse por un conjunto de 8 registros de 13 bits cada uno, a los que se tiene acceso con una organización LIFO y en los que se almacenan direcciones de la memoria de programa.

Las instrucciones que manipulan la pila son call, return, retfie y retlw. También se guarda en la pila una dirección cuando se produce una interrupción.

Dado el tamaño de la pila, se pueden anidar hasta 8 subrutinas; es decir, desde el programa principal se puede llamar a una primera subrutina y desde ésta a una segunda subrutina, y desde ésta a una tercera, y así sucesivamente hasta completar un total de 8 llamadas. Esta profundidad de anidamiento es más que suficiente para la mayoría de las aplicaciones. No obstante, es responsabilidad del programador no excederse en el número de llamadas anidadas. Estos microcontroladores no tienen ningún indicador de desbordamiento de la pila.

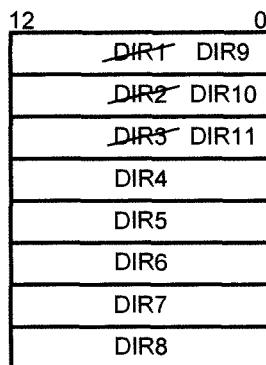


Figura 4.5 La pila en los microcontroladores PIC de gama media está formada por 8 registros o celdas de 13 bits, de modo que se desborda si se anidan más de 8 subrutinas.

4.2 Repertorio de instrucciones de los PIC de gama media

El modelo de programación de los microcontroladores PIC de gama media consta del registro de trabajo W (*Working Register*), que hace la función del tradicional acumulador de los microprocesadores, y de los registros de la memoria de datos, ya sean de funciones especiales (SFR: *Special Function Registers*) o de propósito general (GPR: *General Purpose Registers*) (figura 4.6).

Desde el punto de vista del programador, las características más relevantes del repertorio de instrucciones de los microcontroladores PIC de gama media son:

1. Todas las instrucciones son del mismo tamaño, es decir, 14 bits.
2. La mayoría de las instrucciones son ejecutadas en un sólo ciclo de instrucción que dura cuatro períodos del oscilador principal del microcontrolador. Sólo las instrucciones de salto toman dos ciclos, si se produce el

salto. También duran dos ciclos aquellas instrucciones que modifican el registro PCL.

3. Cualquier registro del microcontrolador puede ser fuente o destino en operaciones de transferencia de datos, aritméticas o lógicas.
4. Se puede acceder individualmente a cualquier bit de cualquier registro de la memoria de datos del microcontrolador.
5. No es posible hacer transferencias de memoria a memoria en una única instrucción; hay que usar el registro W como puente.
6. No hay instrucciones para guardar o extraer datos en la pila (como PUSH y POP, instrucciones que son comunes en muchos microprocesadores). La pila sólo almacena direcciones de instrucciones y con la pila operan sólo las instrucciones de llamadas a subrutinas o retorno desde ellas.

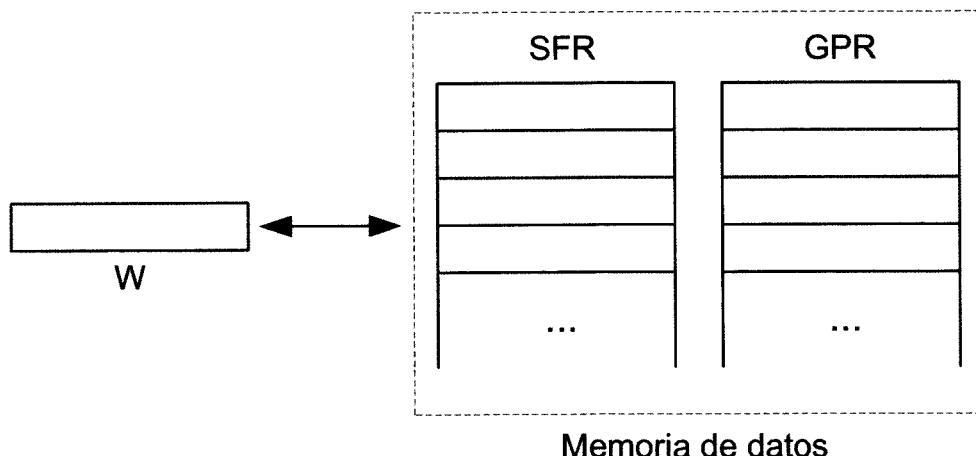


Figura 4.6 *Modelo de programación de los microcontroladores PIC de gama media. Por un lado está el registro de trabajo W y por otro lado están los registros de la memoria de datos: registros de funciones especiales (SFR) y registros de propósito general (GPR).*

La tabla 4.1 resume las instrucciones de los microcontroladores PIC de gama media, que se estudian a continuación. Para su mejor comprensión y estudio, atendiendo a la función que realizan, las instrucciones se han clasificado en:

- Instrucciones de transferencia de datos.
- Instrucciones aritméticas y lógicas.
- Instrucciones de transferencia de control.
- Instrucciones de operaciones con bits.
- Otras.

Tabla 4.1 Repertorio de instrucciones de los microcontroladores PIC de gama media. W: registro de trabajo, f: registro de la memoria de datos, k: constante de 8 bits, a: constante de 11 bits, b: bit, d: destino; si d = 0, el destino es W; si d = 1, el destino es f. C, DC, Z, TO# y PD3 son bits específicos del registro STATUS.

Mnemotécnico	Significado	Afecta	Ciclos
1. Transferencia de datos.			
movf	f, d f => d	Z	1
movwf	f W => f	-	1
movlw	k k => W	-	1
clrf	f 0 => f	Z	1
clrw	0 => W	Z	1
swapf	f, d f _L ↔ f _H => d	-	
2. Aritméticas y lógicas.			
addwf	f, d f + W => d	C, DC, Z	1
addlw	k k + W => W	C, DC, Z	1
subwf	f, d f - W => d	C, DC, Z	1
sublw	k k - W => W	C, DC, Z	1
incf	f, d f + 1 => d	Z	1
decf	f, d f - 1 => d	Z	1
andwf	f, d f and W => d	Z	1
andlw	k k and W	Z	1
iorwf	f, d f or W	Z	1
iorlw	k k or W	Z	1
xorwf	f, d f xor W	Z	1
xorlw	k k xor W	Z	1
rif	f, d rotar f izquierda a través de C => d	C	1
rrf	f, d rotar f derecha a través de C	C	1
comf	f, d #f => d	Z	1
3. Transferencia de control.			
goto	a saltar a la dirección a	-	2
btfsc	f, b salta si f = 0	-	1(2)
btfss	f, b salta si f = 1	-	1(2)
incfsz	f, d f + 1 => d, salta si 0	-	1(2)
decfsz	f, d f - 1 => d, salta si 0	-	1(2)
call	a llamar subrutina en dirección a	-	2
return	retornar de subrutina	-	2

retfie		retornar de interrupción	-	2
retlw	k	retornar de subrutina con k en W	-	2
4. Operaciones con bits.				
bcf	f, b	0 => f	-	1
bsf	f, b	1 => f	-	1
5. Otras.				
nop		no operación	-	1
clrwdt		0 => WDT	TO#, PD#	1
sleep		ir a modo bajo consumo	TO#, PD#	1

4.2.1 Instrucciones de transferencia de datos.

Las instrucciones de transferencia de datos se muestran en la tabla 4.2. Para estas instrucciones, cualquier registro del microcontrolador puede ser fuente o destino, pero no es posible hacer transferencias de memoria a memoria en una única instrucción; hay que usar el registro W como puente. Si el registro especificado es INDF, entonces se está utilizando direccionamiento indirecto y la operación se realiza con el registro apuntado por el registro de funciones especiales FSR.

Tabla 4.2 Instrucciones de transferencia de datos. W: registro de trabajo, f: registro de la memoria de datos, k: dato de 8 bits, b: bit, d: destino. Si d = 0, el destino es W; si d = 1, el destino es f. Z es un bit específico del registro STATUS.

Mnemotécnico		Significado	Afecta	Ciclos
movf	f, d	f => d	Z	1
movwf	f	W => f	-	1
movlw	k	k => W	-	1
clrf	f	0 => f	Z	1
clrw		0 => W	Z	1
swapf	f, d	f_L ↔ f_H => d	-	1

Las instrucciones movf, clrf y clrw afectan al indicador o bandera del cero (bit Z del registro STATUS), mientras que movwf, movlw y swapf no afectan a ninguna bandera. Todas las instrucciones de este grupo se ejecutan en un ciclo de instrucción.

Obsérvese que movwf f copia el contenido del registro de trabajo W en el registro f de la memoria de datos, sin alterar ningún indicador o bandera. En cambio, la instrucción movf f, 0 realiza la operación inversa, es decir, copia el contenido del registro f en el registro de trabajo, sin modificar el registro f,

pero afectando a la bandera del cero; `movf f, 1` simplemente copia el registro f en sí mismo, pero afecta a la bandera Z. Se puede usar para determinar si el valor en f es cero o no.

Al programar en lenguaje ensamblador, el parámetro d, que denota cuál es el destino de la operación de transferencia, puede ser indicado en la instrucción de varias formas, todas ellas igualmente aceptadas por el ensamblador. La instrucción `movf f, d`, por ejemplo, se puede escribir de varias formas. Supongamos que el registro con el que se va a operar es uno de propósito general que ha sido identificado como el registro X. Entonces, si d = 0, las formas `movf X, 0` y `movf X, W` son equivalentes. Si d = 1, se puede escribir indistintamente `movf X, 1` o `movf X, f`.

Ejemplo 4.5

Se dispone de dos registros de propósito general ubicados en un mismo banco de la memoria de datos, que han sido designados como REG1 y REG2. Diseñar un programa que intercambie sus contenidos.

Para resolver este problema se necesita un tercer registro —que denominaremos TEMP— para utilizarlo como almacén temporal en los movimientos de datos, pues no basta con el registro de trabajo W. Si REG1, REG2 y TEMP están en el mismo banco de memoria, un posible programa es el siguiente:

```
movf    REG1, W
movwf   TEMP
movf    REG2, W
movwf   REG1
movf    TEMP, W
movf    REG2
```

4.2.2 Instrucciones aritméticas y lógicas.

Las instrucciones aritméticas y lógicas se muestran en la tabla 4.3. Este grupo de instrucciones incluye, dentro de las operaciones aritméticas, la suma y la resta, y, como casos particulares de estas operaciones, el incremento y decrecimiento. Las operaciones lógicas son la negación lógica o complemento, la suma lógica (*or*), el producto lógico (*and*), la semisuma u or exclusivo (*xor*), rotaciones de 1 bit a la derecha o a la izquierda y el intercambio de cuartetos (grupos de 4 bits contiguos).

Tabla 4.3 Instrucciones aritméticas y lógicas. W: registro de trabajo, f: registro de la memoria de datos, k: dato de 8 bits, b: bit, d: destino. Si d = 0, el destino es W; si d = 1, el destino es f. C, DC y Z son bits específicos del registro STATUS.

Mnemotécnico	Significado	Afecta	Ciclos
addwf	f, d $f + W \Rightarrow d$	C, DC, Z	1
addlw	k $k + W \Rightarrow W$	C, DC, Z	1
subwf	f, d $f - W \Rightarrow d$	C, DC, Z	1
sublw	k $k - W \Rightarrow W$	C, DC, Z	1
incf	f, d $f + 1 \Rightarrow d$	Z	1
decf	f, d $f - 1 \Rightarrow d$	Z	1
andwf	f, d f and W $\Rightarrow d$	Z	1
<u>andlw</u>	k k and W $\Rightarrow W$	Z	1
<u>iorwf</u>	f, d f or W $\Rightarrow d$	Z	1
iorlw	k k or W $\Rightarrow W$	Z	1
xorwf	f, d f xor W $\Rightarrow d$	Z	1
xorlw	k k xor W $\Rightarrow W$	Z	1
rif	f, d rotar f izquierda a través de C $\Rightarrow d$	C	1
rif	f, d rotar f derecha a través de C $\Rightarrow d$	C	1
comf	f, d $\#f \Rightarrow d$	Z	1
swapf	f, d $f_L \leftrightarrow f_H \Rightarrow d$	-	1

Las instrucciones aritméticas afectan a las banderas C, DC y Z del registro STATUS; todas las instrucciones lógicas afectan a la bandera Z, excepto las de rotación que afectan a la bandera C, y la de intercambio de cuartetos, que no afecta a ninguna bandera.

En las operaciones aritméticas y lógicas con dos operandos, uno debe estar en el registro W, mientras que el otro puede estar en W o en cualquier otro registro de la memoria de datos. El resultado de la operación puede colocarse en W o en cualquier otro registro.

Al codificar la instrucción, el destino se indica mediante el parámetro binario d: si d = 0, el destino es W; si d = 1, el destino es el registro especificado en la instrucción.

En todas las instrucciones que trabajan con un registro de la memoria de datos, se admite direccionamiento directo o indirecto. Si el registro especificado es INDF, entonces se está utilizando direccionamiento indirecto y la operación indicada por la instrucción se realiza con el registro apuntado por el registro de funciones especiales FSR.

A diferencia de lo que ocurre con instrucciones semejantes en otros microprocesadores o microcontroladores, el bit de acarreo (bit C del registro STATUS) no interviene directamente en la operación indicada por la instrucción aritmética, aunque sí resulta afectado por el resultado de la instrucción. Es decir, no existe la instrucción que, por ejemplo, sume dos registros teniendo en cuenta el valor del acarreo (operación $f + W + C$), la cual es una operación común en la suma de números enteros de varios bytes. El ejemplo 4.7 ilustra como proceder en este caso.

Las instrucciones de rotación `rif f, d` y `rff f, d`, rotan el contenido del registro indicado en la instrucción, un bit a la izquierda o a la derecha respectivamente. En la rotación interviene el acarreo, es decir, el bit C del registro STATUS, que a todos los efectos funciona como extensión del registro f y ocupa la posición de un supuesto bit 8 de ese registro. El resultado se deposita en W sin modificar f (si d = 0), o se deposita en f (si d = 1), modificando su valor anterior.

Ejemplo 4.6

Algunas operaciones aritméticas y lógicas con el registro de trabajo W.

Incrementar W:

`addlw 1`

Decrementar W:

`addlw 0xff`

Negación lógica (complemento 1) de W:

`iorlw 0xff`

Complemento 2 de W:

`xorlw 0xff`
`addlw 1`

Poner varios bits de W a 0, por ejemplo, los bits 3, 2, 1 y 0:

`andlw 0xf0`

Poner varios bits de W a 1, por ejemplo, los bits 3, 2, 1 y 0:

`iorlw 0x0f`

Ejemplo 4.7

Al sumar o restar números enteros de más de un byte, hay que tener en cuenta el acarreo producido en el paso anterior. Uno de los bytes puede estar en el registro de trabajo W, el otro en un registro cualquiera que denominaremos REG y el acarreo del paso anterior es el bit C del registro STATUS. Si se desea tener el resultado en W, la operación que hay que realizar es $REG + W + C \Rightarrow W$.

El siguiente segmento de programa ilustra cómo realizar esta suma.

btfsc	STATUS, C	; ¿C = 0? Sí – saltar sin incrementar W.
addlw	1	; No – incrementar W.
addwf	REG, W	; W + REG => W.

4.2.3 Instrucciones de transferencia de control.

Las instrucciones de transferencia de control se muestran en la tabla 4.4, e incluyen los saltos incondicionales y los condicionados al estado de un bit de un registro, y las llamadas a subrutinas y retornos.

Tabla 4.4 Instrucciones de transferencia de control. W: registro de trabajo, f: registro de la memoria de datos, k: dato de 8 bits, a: dirección de 11 bits, b: bit, d: destino. Si d = 0, el destino es W; si d = 1, el destino es f.

Mnemotécnico	Significado	Afecta	Ciclos
goto	a	saltar a la dirección a	- 2
btfsc	f, b	salta si f = 0	- 1(2)
btfss	f, b	salta si f = 1	- 1(2)
incfsz	f, d	f + 1 => d, salta si 0	- 1(2)
decfsz	f, d	f - 1 => d, salta si 0	- 1(2)
call	a	llamar subrutina en dirección a	2
return		retornar de subrutina	- 2
retfie		retornar de interrupción	- 2
retlw	k	retornar de subrutina con k en W	- 2

4.2.3.1 Saltos incondicionales, llamadas a subrutinas y retornos.

La instrucción goto a produce un salto incondicional a la instrucción situada en la dirección a. En otras palabras, esta instrucción carga el valor a en el contador de programa (PC) del microcontrolador.

La instrucción call a produce una llamada a la subrutina o subprograma situado en la dirección a. Esta instrucción guarda en la pila el valor del PC y a continuación coloca el valor a en el PC, produciendo con ello el salto a la subrutina.

Las instrucciones goto y call operan con el esquema de memoria de programa organizada en páginas. El operando a de estas instrucciones es un número de 11 bits que representa la dirección dentro de una página. Los dos bits restantes del PC, es decir, los bits PC<12:11> se toman de los bits 4 y 3 del registro PCLATH (apartado 3.2.1.1 y figura 3.7). En principio, si no se modifica previamente el registro PCLATH, las instrucciones goto y call ejecutan saltos dentro de la misma página de memoria de programa en la que se encuentran ellas; por ello, el operando de estas instrucciones es un número de 11 bits.

Para ejecutar saltos entre páginas o realizar llamadas a subrutinas que estén en una página diferente a aquella en la que se encuentran estas instrucciones, hay que poner el número de la página a la que se quiere saltar en los bits 4 y 3 del registro PCLATH y entonces realizar el salto o la llamada. Mediante el operador HIGH del lenguaje ensamblador, este proceso es muy simple, según ilustran los ejemplos siguientes.

Ejemplo 4.8

El segmento de programa siguiente ilustra cómo efectuar el salto a una dirección que esté en una página diferente de la actual.

Prog:

```
movlw  HIGH Prog10
movwf  PCLATH
goto   Prog10
```

Prog10:

```
;
; Prog10 puede ser cualquier dirección de la memoria de programa.
;
```

HIGH es un operador del ensamblador que hace que los bits <15:8> de la dirección representada por la etiqueta Prog10 se tomen como el dato de la instrucción movlw, de modo que el número de la página de destino es colocado en los bits PCLATH<4:3>. Cuando se ejecuta la instrucción goto, los bits PCLATH<4:3> son cargados en los bits 12 y 11 del PC.

Ejemplo 4.9

El segmento de programa siguiente ilustra cómo efectuar la llamada a una subrutina que está en una página diferente a la página desde la que se hace la llamada.

```
movlw  HIGH Subrutina ; Los bits <15:8> de la dirección donde comienza la
movwf  PCLATH          ; subrutina se cargan en PCLATH.
call    Subrutina       ; Se hace una llamada a la subrutina.
;
;
```

Subrutina:

```
; Aquí comienza la subrutina, que puede estar en cualquier dirección
; de la memoria de programa.
;
```

El número de la página de destino es cargado en los bits PCLATH<4:3>. Cuando se ejecuta la instrucción call, los bits PCLATH<4:3> son cargados en los bits 12 y 11 del PC.

La instrucción `goto` produce un salto incondicional *directo*, pues la dirección de la meta del salto se da en la propia instrucción. Este no es el único tipo de salto incondicional posible en los microcontroladores PIC de gama media. Cualquier instrucción que modifique el registro de funciones especiales PCL produce un salto incondicional, en este caso, *indirecto*, pues la dirección de la meta del salto está en un registro y no en la propia instrucción. Como el registro PCL es de 8 bits, el salto se produce en un entorno de 256 direcciones. Para producir saltos más allá de este entorno, hay que cargar apropiadamente el registro PCLATH. El siguiente ejemplo ilustra un salto indirecto con estas características.

Ejemplo 4.10

Se quiere programar un salto incondicional a la dirección denotada por la etiqueta Prog20, pero sin usar la instrucción goto. Los operadores HIGH y LOW del ensamblador facilitan notablemente la programación. El siguiente segmento de programa ilustra el procedimiento:

movlw	HIGH Prog20
movwf	PCLATH
movlw	LOW Prog20
movwf	PCL

Prog20:

; Prog20 puede ser cualquier dirección de la memoria de programa.
;

HIGH y LOW son operadores del ensamblador. HIGH hace que los bits <15:8> de la dirección representada por la etiqueta Prog20 se tomen como el operando de la instrucción movlw, de modo que la parte alta de la dirección que es meta del salto se coloca en el registro W y a continuación en PCLATH. LOW hace que los bits <7:0> de la dirección representada por la etiqueta Prog20 se tomen como el operando de la instrucción movlw, con lo que la parte baja de la dirección que es meta del salto se coloca en el registro W y a continuación en PCL.

Al modificarse PCL, los contenidos de PCLATH y PCL pasan al contador de programa, con lo que se produce el salto a Prog20.

La instrucción `movwf PCL` se ejecuta en dos ciclos de máquina porque modifica el valor del PC.

También es posible realizar un salto relativo al valor del PCL. En PCL se puede tener una dirección base, a la que se le puede añadir un valor mediante la instrucción addwf PCL, F, formando así la dirección de salto. El siguiente ejemplo ilustra el uso de saltos relativos en el manejo de tablas almacenadas en la memoria de programa.

Ejemplo 4.11

La subrutina Tabla contiene una tabla de caracteres ASCII. En un registro de propósito general de la memoria de datos, que se ha identificado como INDICE, está la posición que ocupa un carácter ASCII cualquiera dentro de la tabla, relativa al inicio. Se quiere retornar en W el carácter ASCII apuntado por INDICE. El segmento de programa siguiente ilustra la solución, que utiliza un salto indirecto mediante PCL.

```

;
; Programa principal:
;
movlw  HIGH Tabla      ; Los bits <15:8> de la dirección donde comienza la
movwf  PCLATH          ; tabla se cargan en PCLATH.
movf   INDICE, W        ; El registro INDICE apunta hacia el interior de la tabla.
call   Tabla            ; Se hace una llamada a la subrutina Tabla.
                        ; En W retorna el valor apuntado por INDICE.

;
; Subrutina Tabla. Puede estar en cualquier página de la memoria de programa, siempre que su
; extensión no exceda las 256 palabras y quede comprendida completamente en un ámbito de 256
; direcciones, es decir, que cualquiera de sus instrucciones pueda ser ubicada con solo mover el
; valor de PCL sin alterar el valor de la parte alta del PC.
; Entradas: en W la posición del carácter ASCII.
; Salidas: en W el carácter ASCII solicitado.
;
Tabla:
addwf  PCL, f
retlw  'E'
retlw  'j'
retlw  'e'
retlw  'm'
retlw  'p'
retlw  'l'
retlw  'o'

```

En primer lugar, el ejemplo muestra la forma correcta de llamar una subrutina ubicada en una página diferente a la actual, lo cual se realiza cargando convenientemente los bits 4 y 3 de PCLATH con el número de la página. El operador HIGH hace que los bits <15:8> de la dirección de la subrutina se tomen como el dato de la instrucción movlw, que los coloca en el registro W; a continuación W se copia en PCLATH, de modo que el número de la página de destino es colocado en los bits PCLATH<4:3>. Al ejecutar la instrucción call, el valor actual del PC se guarda en la pila y los bits PCLATH<4:3> son cargados en los bits 12 y 11 del PC, con lo que se salta a la subrutina.

Una vez en la subrutina, la instrucción addwf PCL, f adiciona a PCL el valor que trae W, guardando el resultado en el propio PCL, con lo que se produce un salto a la instrucción retlw que tiene como operando el código ASCII solicitado. Por ejemplo, si W = 4, se retorna en W el código ASCII del carácter 'p'.

La subrutina Tabla puede estar en cualquier página de la memoria de programa, siempre que su extensión no exceda las 256 palabras y quede comprendida completamente en un ámbito de 256 direcciones, es decir, que cualquiera de sus instrucciones pueda ser ubicada con solo mover el valor de PCL.

La instrucción addwf PCL, f modifica el valor del PC, y por ello se ejecuta en dos ciclos de instrucción. La instrucción retlw también dura 2 ciclos de instrucción. Por lo tanto, la ejecución de la subrutina Tabla dura 4 ciclos de instrucción, es decir, 16 ciclos del oscilador principal del microcontrolador.

Las instrucciones return, retfie y retlw k se colocan dentro de una subrutina para retornar al programa desde donde fue llamada la subrutina.

4.2.3.2 Saltos condicionados

Las instrucciones de salto condicionado son btfsc f, b (*bit test file and skip if clear*) y btfss f, b (*bit test file and skip if set*). En estas instrucciones, el salto se efectúa si se cumple una condición: que el valor del bit b, del registro f sea 0 ó 1, respectivamente. El salto que producen es muy corto: si se cumple la condición, se “brinca”, es decir, no se ejecuta la instrucción siguiente; si la condición no se cumple, no se brinca, es decir, se ejecuta la instrucción siguiente.

Para esclarecer el modo de funcionamiento de estas instrucciones, analicemos el siguiente bloque de instrucciones:

btfsc f, b

instrucción 1

instrucción 2

La condición para el salto es que f sea 0. Si se cumple esta condición, se salta directamente a la instrucción 2; si no se cumple, se ejecuta la instrucción 1. La instrucción btfss funciona de modo similar. El diagrama de bloques de la figura 4.7 ilustra la semántica de estas instrucciones.

Ejemplo 4.12

Incrementar el registro de trabajo W si la bandera del acarreo es 1.

El siguiente segmento de programa ilustra la solución del problema:

btfsc STATUS, C
addlw 1

Sigue: ...

La bandera de acarreo es el bit C del registro STATUS. Si el acarreo es 1, no se cumple la condición requerida para el salto y por lo tanto se ejecuta la instrucción addlw 1, que incrementa en 1 el valor de W. Si el acarreo es 0, se cumple la condición de salto y se ejecuta directamente la instrucción marcada con la etiqueta Sigue, sin incrementar W.

Las instrucciones btfsc y btfss son muy útiles pues con ellas se pueden programar decisiones según el estado de cualquier bit de cualquier registro de la memoria de datos del microcontrolador, ya sea un registro de funciones

especiales o uno de propósito general. Combinando estas instrucciones con la instrucción de salto incondicional goto, es posible tomar múltiples decisiones. El ejemplo siguiente ilustra un caso.

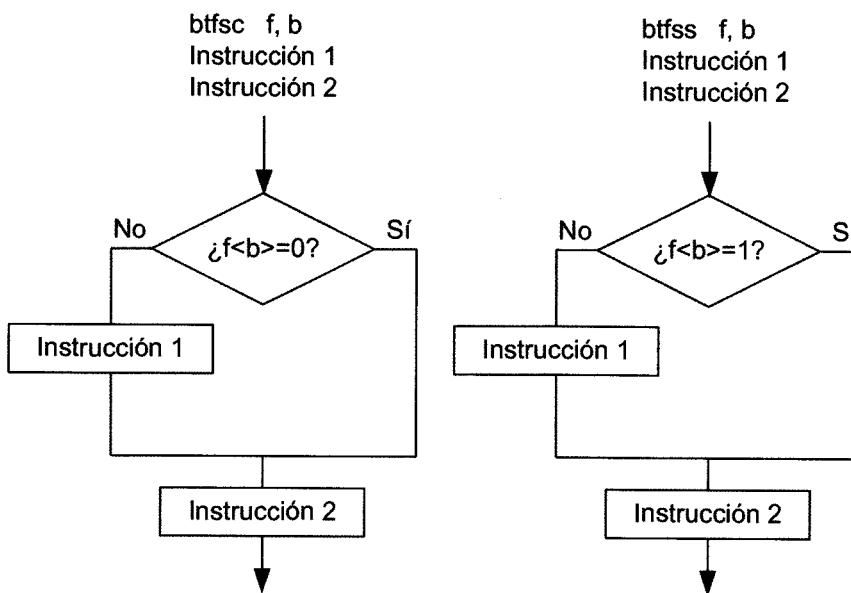


Figura 4.7 Semántica de las instrucciones de salto condicionado. En la parte superior de la figura se muestran segmentos de programas con las instrucciones btfsc y btfss y debajo los diagramas de bloques con la semántica de las instrucciones. El salto se produce según el valor del bit b del registro f, que puede ser cualquier registro de la memoria de datos.

Ejemplo 4.13

La figura 4.8 muestra el diagrama de bloques de la situación que se desea programar. Si se cumple la condición $f = 1$ se debe realizar la acción 1; en caso contrario, se ejecuta la acción 2; después de cualquiera de estas dos acciones, debe ejecutarse la acción 3. La programación de cada acción requiere un número indeterminado de instrucciones.

La siguiente estructura del programa resuelve la situación planteada:

```

btfs f, b
goto Acción2
Acción1:
;
; Aquí van las instrucciones que implementan la Acción 1.
;
goto Acción3
Acción2:
;

```

```

; Aquí van las instrucciones que implementan la Acción 2.
;
Acción3:
;
; Aquí van las instrucciones que implementan la Acción 3.
;
```

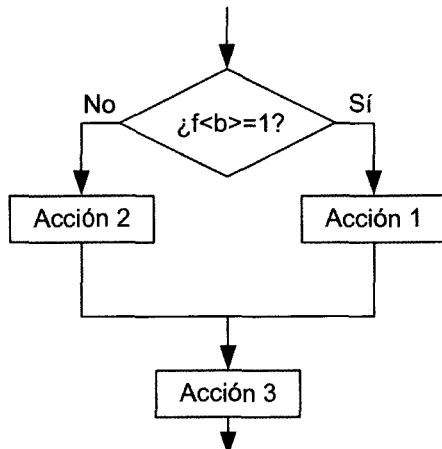


Figura 4.8 Diagrama de bloques con el algoritmo que se desea programar en el ejemplo 4.13.

Las instrucciones `incfsz f, d` y `decfsz f, d` combinan el incremento o decremento de un registro cualquiera con un salto condicionado al resultado de la operación aritmética realizada. Estas instrucciones incrementan o decrementan el registro `f` y si el resultado de esta operación es cero, se produce el salto; en caso contrario, se ejecuta la instrucción siguiente. El resultado se puede colocar en `W` (no se modifica `f`) o en el propio registro, que queda así modificado, lo cual se especifica con el parámetro `d` de la instrucción. El salto que producen es muy corto, al estilo de las instrucciones `btfsc` y `btfss` ya estudiadas. El salto es simplemente un brinco sobre la instrucción siguiente, que deja de ejecutarse si se cumple la condición de que el resultado del incremento o decrecimiento del registro sea cero. Estas instrucciones no alteran las banderas del registro `STATUS`.

Estas dos instrucciones son muy útiles pues, combinadas con la instrucción `goto`, sirven para programar lazos o iteraciones en los que el número de iteraciones se controla por el valor del registro `f` utilizado por la instrucción.

Ejemplo 4.14

Este ejemplo ilustra la forma de programar un lazo.

`movlw veces`

```

        movwf  CONTADOR
Lazo:
;
; Aquí van las instrucciones del lazo
;
decfsz  CONTADOR, f    ; ¿CONTADOR = 0?, Sí – saltar a Fin.
goto    Lazo             ; No – ir a una nueva iteración.

Fin:
;
;
```

En el segmento de programa mostrado, el registro que ha sido identificado con el nombre CONTADOR, almacena inicialmente la cantidad de iteraciones que se desea realizar. El número de iteraciones ha sido representado por la constante designada como "veces". La instrucción decfsz decrementa y actualiza el valor de CONTADOR: mientras el valor de CONTADOR no sea cero, se ejecuta la instrucción goto Lazo y comienza una nueva iteración; cuando CONTADOR llegue a cero, se salta a la dirección con etiqueta Fin, con lo cual termina la ejecución del lazo.

4.2.4 Instrucciones que operan con bits

Las instrucciones que operan con bits son sólo dos, aunque muy potentes, pues permiten poner en 0 o en 1 cualquier bit de cualquier registro del microcontrolador. La tabla 4.5 muestra estas instrucciones y sus principales características de interés para el programador. El ejemplo 4.15 ilustra cómo utilizarlas.

Tabla 4.5 *Instrucciones de operaciones con bits.*

Mnemotécnico	Significado	Afecta	Ciclos
bcf f, b	0 => f	-	1
bsf f, b	1 => f	-	1

Ejemplo 4.15

Seleccionar el banco 1 de la memoria de datos.

El banco de la memoria de datos que va a ser utilizado se selecciona mediante los bits RP1 y RP0 del registro STATUS. El banco 1 se selecciona haciendo RP1 = 0 y RP0 = 1, lo cual puede hacerse con el siguiente segmento de programa:

```

bcf      STATUS, RP1
bsf      STATUS, RP0
```

Otras instrucciones que operan con bits son btfsc y btfss, que realizan saltos condicionados por el estado de un bit, y se han incluido en el grupo de instrucciones de transferencia de control.

4.2.5 Otras instrucciones

La tabla 4.6 muestra las instrucciones que no pertenecen a ninguno de los grupos anteriores. La instrucción `nop`, tal como indica su nombre, no realiza ninguna operación, salvo la de ocupar tiempo del microcontrolador: un ciclo de instrucción.

Tabla 4.6 Otras instrucciones de control general del microcontrolador.

Mnemotécnico	Significado	Afecta	Ciclos
<code>nop</code>	no operación	-	1
<code>clrwdt</code>	0 => WDT	TO#, PD#	1
<code>sleep</code>	ir a modo bajo consumo	TO#, PD#	1

La instrucción `clrwdt` ejecuta un “*reset*” del perro guardián, es decir, borra (pone en 0) el temporizador del perro guardián y borra también el pre-divisor. Los bits TO# y PD# del registro STATUS son puestos ambos a 1. TO# es el indicador de desbordamiento del perro guardián y PD# es el indicador de que el microcontrolador ha entrado en el modo de bajo consumo.

La tercera instrucción de este grupo, `sleep`, pone al microcontrolador en el modo de bajo consumo, produciendo además un *reset* del perro guardián y poniendo a cero el valor del pre-divisor. El bit TO# es puesto a 1 y el bit PD# es puesto a 0.

4.3 Elementos del lenguaje ensamblador (para el ensamblador MPASM de Microchip)

Como todo lenguaje de programación, el lenguaje ensamblador tiene sus propias reglas para escribir las palabras y combinarlas para formar instrucciones. Estas reglas constituyen la sintaxis del lenguaje. En este apartado se estudia la sintaxis del lenguaje ensamblador de los microcontroladores PIC de gama media.

4.3.1 Introducción

Un programa en lenguaje ensamblador es una secuencia de líneas de texto, cada una de las cuales puede ser:

- Una instrucción del repertorio de instrucciones del microcontrolador.
- Una directiva del ensamblador.
- Una macroinstrucción, o simplemente “macro”.
- Un comentario.

- Una etiqueta.
- Una línea en blanco.

Una *directiva* es una instrucción que se escribe en el programa fuente y está dirigida al programa ensamblador. Es una orden para el programa ensamblador, no para el microcontrolador. En un programa escrito en lenguaje ensamblador se mezclan las instrucciones para el microcontrolador con las instrucciones para el programa ensamblador o directivas. Las directivas transmiten al ensamblador diversas indicaciones, como, por ejemplo, la definición de símbolos variables y constantes, la ubicación del programa en la memoria, la ubicación de variables en memoria de datos, etc., según se estudia más adelante en este capítulo.

Una *macroinstrucción* o *macro* es una instrucción definida por el usuario. En la definición de una macro se utilizan instrucciones del repertorio del microcontrolador y directivas del ensamblador. Una vez definida una macroinstrucción, basta con citarla en el texto del programa para que el ensamblador sustituya la llamada a la macro por el conjunto de instrucciones y directivas que la definieron.

Un *comentario* es un texto cuyo objetivo es informar, haciendo con ello más fácil la lectura y comprensión del programa fuente. Va precedido por el carácter punto y coma (:). El ensamblador, cuando encuentra este carácter, ignora todo lo que sigue hasta el final de la línea.

Una *etiqueta* (*label*) es el símbolo que identifica a una línea de programa fuente y que representa la dirección de una instrucción. Las etiquetas deben colocarse a partir de la columna 1 para que el ensamblador las identifique como tales y pueden ser seguidas por el carácter dos puntos (:).

Ejemplo 4.16

En el segmento de programa que se muestra a continuación, hay dos instrucciones: una dirigida al ensamblador y otra al microcontrolador.

```
dato_5 equ      0xA8      ; Se define el símbolo dato_5 y se le asigna un valor.
prog1:
    movlw     dato_5      ; La instrucción carga el registro W con el valor
                          ; del símbolo dato_5.
```

El símbolo dato_5 representa un dato constante cuyo valor se asigna mediante la directiva equ. El símbolo prog1 es una etiqueta y representa la dirección de la memoria de programa donde se ubica la instrucción movlw dato_5. La primera línea es una instrucción al programa ensamblador mientras que la tercera es una instrucción dirigida al microcontrolador. Obsérvese que la segunda y la cuarta líneas contienen, respectivamente, sólo una etiqueta y un comentario.

Las líneas que contienen instrucciones para el microprocesador se estructuran en varios campos, algunos de ellos opcionales, con la sintaxis siguiente:

[etiqueta[:]] mnemotécnico [operando1][, operando2] [; comentario]

Los campos indicados entre corchetes son opcionales. Los campos de una instrucción se separan entre sí con uno o más espacios en blanco, o con caracteres de tabulación. Si la instrucción tiene dos operandos, estos se separan por una coma (,). En el campo de los operandos se utilizan constantes, símbolos y expresiones.

Las directivas y el mnemotécnico de las instrucciones se pueden escribir con letras mayúsculas o minúsculas. Por ejemplo, `movlw` y `MOVLW` son dos formas correctas y similares de escribir un mismo mnemotécnico.

Las *constantes* son valores numéricos que se utilizan en el programa. Una constante puede ser numérica o ASCII. Al escribir una constante numérica se pueden emplear los sistemas de numeración decimal, hexadecimal, octal o binario. Una constante ASCII se forma con el código binario de un carácter ASCII. Las constantes son tratadas por el ensamblador como números binarios de 32 bits. El valor de la constante es truncado si se intenta colocarla en un campo de menor longitud. La tabla 4.7 da ejemplos de la sintaxis de las constantes en el lenguaje ensamblador de los PIC.

Tabla 4.7 Tipos de constantes permitidas en la programación en lenguaje ensamblador y su sintaxis. Las letras D, H, O y B usadas para indicar el tipo de constante, pueden ser mayúsculas o minúsculas.

Constante	Sintaxis	Ejemplo	Valor de la constante
Decimal	D'dígitos_decimales' 'dígitos_decimales'	D'167' '167'	0x000000A7
Hexadecimal	H'dígitos_hexadecimales' 0xdígitos_hexadecimales dígitos_hexadecimalesH	H'A7' 0xA7 0A7H	0x000000A7
Octal	O'dígitos_octales' dígitos_octalesO	O'247' 247O	0x000000A7
Binaria	B'dígitos_binarios'	B'10100111'	0x000000A7
ASCII	A'carácter_ASCII' 'carácter_ASCII'	A'Z' 'Z'	0x0000005A

Ejemplo 4.17

Las siguientes instrucciones ejemplifican la escritura de constantes de diferentes tipos. Todas ellas hacen lo mismo: poner el valor decimal 167 en el registro W.

```
movlw .167  
movlw 0a7h  
movlw 2470  
movlw b'10100111'
```

Obsérvese que las constantes hexadecimales escritas en el formato *digitos_hexadecimalesH* deben comenzar por un dígito para no ser confundidas con etiquetas.

Las constantes numéricas pueden ir precedidas por los signos más (+) o menos (-) para indicar cantidades positivas o negativas, respectivamente. Si no se coloca ningún signo, se supone que la constante es positiva.

En los programas en ensamblador se utilizan también *símbolos*. Un símbolo es una cadena de caracteres alfanuméricos (letras y números) de hasta 32 caracteres, que comienza siempre con una letra o con el carácter guión bajo (_). Los símbolos se usan para nominar:

- Las direcciones de instrucciones. En este caso el símbolo recibe el nombre de etiqueta.
- Los datos constantes.
- Los registros de la memoria de datos, ya sean registros de funciones especiales o de propósito general.
- Los bits de los registros de la memoria de datos.

Excepto las etiquetas, los símbolos deben definirse antes de ser utilizados en el programa fuente. Para definir los símbolos se usan, entre otras formas, las directivas equ y set o los operadores de asignación que se estudian en el apartado 4.3.2.4.

En realidad, los símbolos que se emplean para nominar registros representan direcciones de la memoria de datos. Esto es así porque en las instrucciones se hace referencia a los registros a través de la dirección que tienen en la memoria de datos. El ensamblador codifica el nombre de la variable usando la dirección que dicho nombre representa. Así, en el ejemplo 4.18, si en una instrucción aparece la variable REG1, el ensamblador la codifica utilizando el valor 20h. Los símbolos que dan nombre a bits de registros, deben definirse con valores del 0 al 7 (pues los registros de los PIC son de 8 bits).

Ejemplo 4.18

En el segmento de programa que se muestra a continuación, se usan los símbolos CONST, REG1 VAR y prog1.

CONST1	equ	0xA5	; Se define el símbolo CONST1 y se le asigna el valor A5h.
REG1	equ	20h	; Se define el símbolo REG1 y se le asigna el valor 20h.

BIT3	equ	3	; Se define el símbolo BIT3 y se le asigna el valor 3.
	org	0x10	; Dirección de memoria donde comienza el programa.
prog1:	movlw	CONST1	; Se carga el registro W con el valor A5h.
	movwf	REG1	; El valor de W se guarda en el registro REG1.
	bcf	REG1, BIT3	; Se pone a 0 el bit 3 de REG1

Por la forma en que se usa en el programa, el símbolo CONST1 representa un dato constante de 8 bits que valen A5h.

REG1 designa el registro de propósito general situado en la dirección 20h del banco de memoria activo; el valor del símbolo REG1 es 20h.

BIT3 es una constante que vale 3 y, por la forma en que se usa, es el nombre del bit 3 del registro REG1.

El símbolo prog1 es una etiqueta y representa la dirección donde comienza el programa. Su valor es 10h, dado por la directiva org 0x10 que le precede.

En el lenguaje ensamblador de los microcontroladores PIC, los nombres de los registros de funciones especiales, así como los nombres de bits de esos registros, no constituyen símbolos predefinidos o reservados del lenguaje ensamblador. Es decir, el programador debe definir en el programa fuente los símbolos que utilizará para nominar esos registros y sus bits, so pena de tener que utilizar sus direcciones numéricas para referirse a ellos. Para facilitar la programación, Microchip suministra para cada dispositivo su archivo de definición. Es un archivo de texto con los símbolos que utiliza el fabricante para nominar todos los registros de funciones especiales y bits de aquel dispositivo. Así, por ejemplo, para el microcontrolador PIC16F873, está el archivo P16F873.INC con las definiciones de los nombres de los registros y bits de ese dispositivo. Para cada microcontrolador se ofrecen los archivos respectivos. En el programa fuente en lenguaje ensamblador, el programador sólo tiene que “incluir” el archivo del dispositivo, para así poder utilizar en el programa los símbolos definidos en ese archivo. Esta inclusión se hace mediante la directiva #include, tal como ilustra el ejemplo 4.19

Ejemplo 4.19

Al trabajar con el microcontrolador PIC16F84, se dispone del archivo P16F84.INC, que contiene la definición de los nombres de todos los registros de funciones especiales y bits de dichos registros de ese microcontrolador.

Para incorporar todos esos nombres al programa fuente, basta con escribir una línea con la indicación al ensamblador de que incluya el contenido del archivo en el programa fuente. Esto se hace mediante la directiva include, así:

```
#include P16F84.INC
```

Esta línea debe escribirse antes de utilizar los nombres de los registros del microcontrolador en el programa. Una vez dada esta indicación al ensamblador, se puede hacer referencia a los registros de funciones especiales y a sus bits, mediante los nombres definidos en el archivo incluido.

A continuación se muestra el listado (parcial) del archivo P16F84.INC. Microchip suministra archivos semejantes a este para cada microcontrolador PIC.

```
; P16F84.INC Standard Header File, Version 2.00  Microchip Technology, Inc.
; 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.
```

```
=====
; 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 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'
<hr/>		
— OPTION Bits -----		
NOT_RBPU	EQU	H'0007'
INTEDG	EQU	H'0006'
TOCS	EQU	H'0005'
TOSE	EQU	H'0004'
PSA	EQU	H'0003'
PS2	EQU	H'0002'
PS1	EQU	H'0001'
PS0	EQU	H'0000'
<hr/>		
— EECON1 Bits -----		
EIF	EQU	H'0004'
WRERR	EQU	H'0003'
REN	EQU	H'0002'
WR	EQU	H'0001'
RD	EQU	H'0000'

4.3.2 Expresiones, operaciones y operadores

Las *expresiones* son combinaciones de constantes y símbolos mezclados con operadores aritméticos y lógicos. En su escritura se permite usar paréntesis con una función similar a la que tienen en las expresiones algebraicas del lenguaje matemático común.

Las expresiones se pueden utilizar en el campo de los operandos de instrucciones y directivas, y se evalúan siempre durante el ensamblaje. El resultado de la evaluación de una expresión pasa a ser el valor del operando a los efectos del ensamblaje.

Ejemplo 4.20

Las expresiones se evalúan durante el ensamblaje, no durante la ejecución del programa. En el siguiente segmento de programa, REG1 es el registro de la memoria de datos situado en la dirección 20h de uno de los bancos de memoria de datos del microcontrolador:

```
REG1      equ      20h
          movwf   REG1 + 1
```

La expresión REG1 + 1, NO incrementa el contenido del registro REG1, sino el valor del símbolo REG1, por lo que el resultado de la expresión es 21h. Este resultado se obtiene durante el ensamblaje del programa. Por lo tanto, 21h es entonces el operando de la instrucción movwf. Cuando se ejecute el programa, la instrucción movwf copiará el contenido del registro W en el registro 21h del banco de memoria activo.

Los *operadores* son los símbolos que denotan las operaciones aritméticas y lógicas definidas en el lenguaje ensamblador.

4.3.2.1 Operadores aritméticos

La tabla 4.8 muestra los operadores aritméticos y las operaciones que representan. Los operadores aritméticos indican la realización de operaciones aritméticas básicas con símbolos y constantes. Se pueden utilizar para construir expresiones que formen parte de los operandos de instrucciones, o en directivas del ensamblador. Las operaciones aritméticas se realizan en 32 bits, aunque el resultado puede quedar truncado al número de bits que tenga el registro o la dirección a la que sea asignada la expresión.

Tabla 4.8 Operadores y operaciones aritméticas.

Operador	Operación	Ejemplo
+	Suma	A1 + A2
-	Resta	A1 - A2
*	Multiplicación	A1 * A2
/	División	A1 / A2
%	Módulo (resto en una división)	A1 % A2

Ejemplo 4.21

El siguiente segmento de programa ilustra el uso de los operadores aritméticos, en particular la operación módulo.

```
DATO1    equ      .18
DATO2    equ      .7
;
        movlw    DATO1 % DATO2
```

Al ensamblar el programa se evalúa la expresión DATO1 % DATO2 con los valores de los símbolos DATO1 y DATO2. La operación módulo entrega como resultado el residuo de la división entre los argumentos de la operación. En este ejemplo, al dividir 18 entre 7 el residuo es 4. Ese será el valor que se depositará en el registro W cuando se ejecute el programa. Aunque el ensamblador calcula la expresión usando aritmética de 32 bits, en W se guardan sólo los 8 bits menos significativos.

Los operadores de suma y resta también sirven para indicar si una constante es positiva o negativa. En particular, el operador “-” colocado delante de un símbolo o una constante, genera el complemento a 2 del valor del símbolo o constante en 32 bits. Si se intenta asignar ese valor a un registro de menor tamaño, el valor se trunca y sólo se toman los bits menos significativos.

Ejemplo 4.22

En el siguiente segmento de programa se define el símbolo DATO y se le asigna el valor decimal 3. ¿Qué valor toma la expresión - DATO? ¿Qué valor se almacena en el registro W cuando se ejecute el programa?

```
DATO    equ      .3
```

`movlw` - DATO

La evaluación de las expresiones se hace con aritmética de 32 bits, y por ello la expresión - DATO toma el valor FFFFFFFFdh, que es la representación de -3 en complemento 2 usando 32 bits. Al codificar la instrucción `movlw`, el ensamblador toma sólo los 8 bits menos significativos de ese valor, pues se trata de un operando de 8 bits. Es decir, la instrucción `movlw` se codifica con el valor FDh como su operando. Cuando se ejecute el programa, en W se depositará ese valor, que es la representación de -3 en complemento 2 usando 8 bits.

4.3.2.2 Operadores lógicos y de relación

La tabla 4.9 muestra los operadores lógicos y de relación. Los símbolos y constantes que intervienen en las operaciones lógicas y de relación sólo pueden tomar uno de dos valores lógicos posibles: VERDADERO o FALSO. Un símbolo o constante tiene valor lógico VERDADERO si su valor numérico es diferente de cero y tiene valor lógico FALSO si su valor numérico es cero.

Tabla 4.9 Operadores lógicos y de relación.

Operador	Operación	Ejemplo
!	Negación lógica (NOT)	!A1
&&	Producto lógico (AND)	A1 && A2
	Suma lógica (OR)	A1 A2
>	Mayor que	A1 > A2
<	Menor que	A1 < A2
>=	Mayor o igual que	A1 >= A2
<=	Menor o igual que	A1 <= A2
==	Igual a	A1 == A2
!=	No igual a	A1 != A2

Al evaluar una expresión lógica o de relación, sólo puede tomar los valores VERDADERO o FALSO. Si el valor lógico es VERDADERO, la expresión toma el valor numérico 1 y si es FALSO, toma el valor 0.

Ejemplo 4.23

Si los símbolos A1 y A2 tienen los valores 20h y 21h respectivamente, ¿qué valor toma la expresión que los utiliza? ¿Qué valor se deposita en el registro W cuando se ejecuten las instrucciones indicadas? La tabla siguiente responde a estas preguntas.

Expresión lógica o de relación	Valor lógico al ser evaluada, si A1=20h y A2=21h	Instrucción que emplea la expresión lógica	Valor en W cuando se ejecuta la instrucción
!A1	FALSO	<code>movlw !A1</code>	00h
A1 && A2	VERDADERO	<code>movlw A1 && A2</code>	01h
A1 A2	VERDADERO	<code>movlw A1 A2</code>	01h
A1 > A2	FALSO	<code>movlw A1 > A2</code>	00h
A1 < A2	VERDADERO	<code>movlw A1 < A2</code>	01h
A1 >= A2	FALSO	<code>movlw A1 >= A2</code>	00h

A1 <= A2	VERDADERO	movlw A1 <= A2	01h
A1 == A2	FALSO	movlw A1 == A2	00h
A1 != A2	VERDADERO	movlw A1 != A2	01h

4.3.2.3 Operadores lógicos que operan directamente con bits

La tabla 4.10 muestra los operadores lógicos que trabajan directamente con bits. Estos operadores realizan las operaciones lógicas indicadas con los bits de símbolos y constantes. En las operaciones AND, OR y XOR, que se realizan entre dos símbolos o constantes, la operación lógica tiene lugar entre bits de igual posición de ambos símbolos. El resultado de una operación lógica directa con bits es un número binario.

Tabla 4.10 Operadores lógicos que trabajan directamente con bits.

Operador	Operación	Ejemplo
~	NOT	~A1
&	AND	A1 & A2
	OR	A1 A2
^	Or exclusivo (XOR)	A1 ^ A2
>>	Desplazar a la derecha	A1 >> 1
<<	Desplazar a la izquierda	A1 << 2

Ejemplo 4.24

Si los símbolos A1 y A2 tienen los valores 3 y 5 respectivamente, ¿qué valor toma la expresión que los utiliza? ¿Qué valor se deposita en el registro W cuando se ejecuten las instrucciones indicadas? La tabla siguiente da respuesta a estas preguntas.

Expresión lógica o de relación	Valor numérico resultante de la evaluación, si A1=3 y A2=5	Instrucción que emplea la expresión lógica	Valor en W cuando se ejecuta la instrucción
~ A1	FFFFFFFFFFCh	movlw ~ A1 + 1	FDh (es decir, -3)
A1 & A2	00000001h	movlw A1 & A2	01h
A1 A2	00000007h	movlw A1 A2	07h
A1 ^ A2	00000006h	movlw A1 ^ A2	06h
A1 >> 1	00000001h	movlw A1 >> 1	01h
A1 << 2	0000000Ch	movlw A1 << 2	0Ch

4.3.2.4 Operadores de asignación

Los *operadores de asignación* permiten asignar un valor a un símbolo. El valor asignado puede ser el resultado de la evaluación de una expresión aritmética o lógica. Con estos operadores se puede definir un símbolo a la vez que se le asigna un valor inicial, o se puede modificar el valor de un símbolo ya definido. En este último caso es necesario que el símbolo sea modificable

por el ensamblador, pues de lo contrario el ensamblador emitirá un mensaje de error al intentar cambiar su valor.

Para definir símbolos cuyo valor puede ser modificado en el transcurso del ensamblaje, hay que hacerlo con la directiva `set` o mediante operadores de asignación; no se debe usar la directiva `equ` porque esta directiva da al símbolo definido el atributo de constante (su valor no se puede modificar). En cambio, la directiva `set` y los operadores de asignación definen símbolos con atributos de variables. La tabla 4.11 muestra los operadores de asignación.

Tabla 4.11 Operadores de asignación.

Operador	Operación	Ejemplo	Significado
=	Asignación aritmética o lógica	var = 0	var = 0
++	Incrementar	var ++	var = var + 1
--	Decrementar	var --	var = var - 1
+=	Sumar y asignar	var += k	var = var + k
-=	Restar y asignar	var -= k	var = var - k
*=	Multiplicar y asignar	var *= k	var = var * k
/=	Dividir y asignar	var /= k	var = var / k
%=	Módulo y asignar	var %= k	var = var % k
&=	AND y asignar	var &= k	var = var & k
=	OR y asignar	var = k	var = var k
^=	XOR y asignar	var ^= k	var = var ^ k
>>=	Desplazar a la derecha y asignar	var >>= k	var = var >> k
<<=	Desplazar a la izquierda y asignar	var <<= k	var = var << k

Las expresiones de asignación se utilizan en lugar de directivas `set` o como parte de los argumentos de directivas. No se admiten expresiones de asignación en el campo de operandos de una instrucción al microprocesador.

Ejemplo 4.25

El siguiente segmento de programa ilustra el uso de los operadores de asignación. El programa almacena en los registros 20h y 21h los valores 10 y 15 respectivamente.

```
DATO = .10 ; Se define el símbolo DATO con valor inicial 10.
REGISTRO = 0x20 ; Se define el símbolo REGISTRO con valor inicial 20h.
```

```
        movlw   DATO      ; En W se coloca un 10
        movwf   REGISTRO ; que se guarda en el registro 20h.
DATO += .5          ; Ahora el símbolo DATO vale 15
REGISTRO ++
        movlw   DATO      ; En W se coloca un 15
        movwf   REGISTRO ; que se guarda en el registro 21h.
```

En las primeras dos líneas se definen los símbolos variables DATO y REGISTRO, con sus valores iniciales respectivos. El símbolo DATO se usa en el programa como un dato constante, mientras que REGISTRO representa la dirección de un registro de la memoria de datos. Ambos símbolos cambian sus valores durante el ensamblaje del programa, por lo que las instrucciones de las líneas 7 y 8 se codifican con operandos diferentes de los de las instrucciones de las líneas 3 y 4.

4.3.2.5 Operadores de dirección

La tabla 4.12 muestra los operadores de dirección. Estos operadores trabajan con direcciones de memoria. El operador \$ usado como operando de una instrucción, significa la dirección actual de la instrucción.

Tabla 4.12 Operadores de dirección.

Operador	Operación	Ejemplo
\$	Dirección actual	goto \$
low	Byte bajo de la dirección	movlw low etiqueta
high	Byte alto de la dirección	movlw high etiqueta
upper	Byte más alto de la dirección	movlw upper etiqueta

Ejemplo 4.26

La instrucción:

goto \$

tiene el mismo efecto que la instrucción:

prog: goto prog

Los operadores low, high y upper, devuelven respectivamente, los bits 0 al 7, 8 al 15 y 16 al 21 de la etiqueta sobre la que operan. Como las direcciones de memoria de los microcontroladores PIC de gama media tienen como máximo 13 bits, el operador upper no se usa en los programas de estos microcontroladores y el operador high devuelve los bits 8 al 12 de la dirección sobre la que opera. El ejemplo 4.10 (apartado 4.2.3.1) ilustra cómo utilizar los operadores low y high.

4.3.3 Directivas

En este apartado se estudian las directivas empleadas en situaciones de programación comunes. La descripción detallada de todas las directivas del MPASM se puede consultar en el sitio web de Microchip Technology.

Las *directivas* son instrucciones dirigidas al programa ensamblador, no al microcontrolador que ejecutará el programa objeto resultante del ensamblaje. En un programa fuente escrito en lenguaje ensamblador, se mezclan las

líneas que contienen directivas con las que contienen instrucciones al microcontrolador.

Mediante directivas se puede controlar la operación del ensamblador y se le puede informar de diversos aspectos de interés para el correcto proceso de ensamblaje. Por ejemplo, mediante las directivas apropiadas se indica al ensamblador el tipo de microcontrolador; se definen los símbolos empleados en el programa para nominar a datos, registros y bits; la dirección a partir de la cual se colocan las instrucciones de un programa o de parte de él, etc.

La sintaxis general de las directivas es la siguiente:

[etiqueta[:]] directiva [operandos] [: comentario]

Si la directiva tiene varios operandos, estos se separan entre sí por una coma (,). En el campo de los operandos, se utilizan constantes, símbolos y expresiones. Los campos indicados entre corchetes son opcionales.

La tabla 4.13 muestra las directivas más utilizadas, que se estudian en este apartado. El listado completo de las directivas admitidas por el ensamblador MPASM, está en el archivo de ayuda de dicho ensamblador, disponible en el sitio web de Microchip Technology.

Tabla 4.13 Directivas más utilizadas por el ensamblador MPASM de los PIC.

Directivas de uso general	
Operación que se desea realizar	Directivas
Definir el microcontrolador y el sistema de numeración	list, processor, radix
Incluir un archivo en el cuerpo del programa fuente (por ejemplo, un archivo .inc con la definición de símbolos)	#include
Definir símbolos	equ, set,
Asignar un valor al seudocuentador de programa	org
Terminar el programa fuente	end
Directivas que se usan en la codificación relocalizable	
Operación que se desea realizar	Directivas
Indicar el inicio de un bloque de instrucciones	code
Indicar el inicio de un bloque de datos	udata, udata_shr
Reservar espacio de memoria de datos	res
Indicar el alcance de los símbolos	global, extern
Seleccionar una página de la memoria de programa	pagesel
Seleccionar un banco de la memoria de datos	banksel, bankisel

4.3.3.1 Directivas de uso general

Las directivas de uso general las emplean casi todos los programas en lenguaje ensamblador. Con estas directivas se instruye al ensamblador sobre:

- Cuál es el microcontrolador PIC para el que se escribe el programa.
- Cuál es el sistema de numeración que se utiliza por defecto.
- Cuál es el archivo que contiene las definiciones de los símbolos de registros y bits del microcontrolador empleado.
- Cuáles son los símbolos que se utilizan para denominar cada registro de propósito general y los bits empleados por el programador.
- Cuál es la dirección de la memoria de programa a partir de la cual el ensamblador debe codificar las instrucciones del programa.

Directivas list, processor y radix

La sintaxis de la directiva list (opciones de listado) es:

`list [opción1][, opción2][, ...]`

La directiva list activa la generación del archivo de listado (.lst) y controla su formato. Sin embargo, las opciones más utilizadas de esta directiva no guardan relación con el formato del archivo de listado, sino con el control del ensamblaje. Estas opciones se muestran en la tabla 4.14. El ejemplo 4.27 ilustra cómo utilizar esta directiva.

Tabla 4.14 Algunas opciones posibles de la directiva list.

Opción	Significado
<code>p = tipo_procesador</code>	Informa del tipo de microcontrolador. Ejemplo: <code>p = 16f873</code> informa al ensamblador que el microcontrolador es un PIC16F873. Esta opción no tiene ningún valor supuesto por defecto.
<code>r = sistema_numeración</code>	Informa del sistema de numeración que se utiliza en el programa al escribir una constante numérica (decimal: DEC, hexadecimal: HEX, octal: OCT). Ejemplo: <code>r = DEC</code> . Si no se especifica la opción r, se presupone que el sistema de numeración es el hexadecimal.
<code>f = formato_hex</code>	Especifica el formato del archivo hexadecimal (hexadecimal de 8 bits estándar: INHX8M, hexadecimal de 8 bits separado: INHX8S, hexadecimal extendido a 32 bits: INHX32) resultante del ensamblaje absoluto. Ejemplo: <code>f = INHX8M</code> . Si no se especifica la opción f, el archivo hexadecimal que se genera es el hexadecimal de 8 bits estándar.

Ejemplo 4.27

El ensamblador MPASM supone por defecto que las constantes numéricas están escritas con el sistema

de numeración hexadecimal y que el archivo hexadecimal que puede generar el ensamblador tendrá el formato hexadecimal de 8 bits estándar de Intel. Por lo tanto, si no se desea cambiar estos parámetros, sólo es necesario especificar en la directiva list el tipo de microcontrolador. Si se trata de un PIC16F873, la línea de programa con la directiva es:

```
list      p = 16f873
```

Hay que recordar que la palabra list se debe escribir al menos a partir del segundo espacio de la línea, para que no se confunda con una etiqueta, en cuyo caso se produciría un mensaje de error, pues list es una palabra reservada del lenguaje.

Otra forma de declarar el tipo de microcontrolador y el sistema de numeración que se utilizará en la escritura de expresiones es, respectivamente, con las directivas processor y radix. Su sintaxis es:

processor	tipo_procesador
radix	sistema_numeración

El ejemplo 4.28 ilustra cómo usar estas dos directivas.

Ejemplo 4.28

Se desea declarar que el microcontrolador que ejecutará el programa que se está elaborando es un PIC16F84A y que en el programa se utiliza el sistema de numeración decimal para escribir las constantes numéricas.

Hay dos formas de hacer estas declaraciones. La primera utiliza las directivas processor y list:

processor	16f84a
radix	dex

La segunda forma es mediante la directiva list:

```
list      p = 16f84a, r = dec
```

Si en el programa se escriben constantes sin especificar cuál es el sistema de numeración, el ensamblador considera que están escritas en el sistema de numeración declarado en la directiva radix o en la opción r de la directiva list escrita antes de las constantes.

Ejemplo 4.29

El siguiente segmento de programa (inspirado en uno similar que se ofrece en la ayuda del ensamblador MPASM), ilustra cómo se interpreta el valor de las constantes en relación con las directivas radix y list.

```
list      r = dec ; A partir de aquí, las constantes son decimales, si no se
                  ; declara otra cosa.

      movlw    50H      ; Este 50 es hexadecimal.
      movlw    0x50     ; Otra forma de declarar el hexadecimal 50.
```

movlw	50O	; Este 50 es octal.
movlw	50	; Este 50 es decimal, pues no se especifica en qué sistema de numeración está escrito.
radix	oct	; A partir de aquí, las constantes son octales, si no se declara otra cosa.
movlw	50H	; Este 50 es hexadecimal.
movlw	0x50	; Otra forma de declarar el hexadecimal 50.
movlw	.50	; Este 50 es decimal.
movlw	50	; Este 50 es octal, pues no se especifica en qué sistema de numeración está escrito.
radix	hex	; A partir de aquí, las constantes son hexadecimales, si no se declara otra cosa.
movlw .	50	; Este 50 es decimal.
movlw	50O	; Este 50 es octal.
movlw	50	; Este 50 es hexadecimal, pues no se especifica en qué sistema de numeración está escrito.

Directivas equ y set

La sintaxis de las directivas equ (definir una constante) y set (definir una variable) es:

símbolo	equ	expresión
símbolo	set	expresión

Estas directivas asignan el valor de la expresión al símbolo. La diferencia entre ellas está en que el valor de un símbolo definido mediante la directiva equ no puede ser modificado posteriormente por el ensamblador; en cambio, si el símbolo se define con la directiva set, su valor puede cambiar en el transcurso del ensamblaje del programa.

Es importante no confundir el carácter constante o variable que otorgan las directivas equ y set al símbolo, a los efectos del ensamblaje, con el significado que pueda tener el símbolo para el programador durante la ejecución del programa. Por ejemplo, se puede definir un registro de propósito general mediante una directiva equ (lo cual lo define como constante durante el ensamblaje), aunque el valor que se almacena en el registro cambie durante la ejecución del programa (es una variable en la ejecución del programa).

La directiva `equ` se utiliza comúnmente para definir símbolos que se refieren a aspectos invariantes del hardware del microcontrolador, como, por ejemplo, los nombres de los registros de funciones especiales y sus direcciones en la memoria de datos. La directiva `equ` también se usa para nominar datos constantes.

En programas elaborados para su ensamblaje absoluto (donde no se usa el enlazador o *linker*), la directiva `equ` se emplea para nominar los registros de propósito general que se utilicen en el programa y asignar a estos nombres la dirección RAM correspondiente. Si el programa se elabora para ensamblaje relocalizable (donde se emplea el enlazador), no se recomienda usar este método para definir registros de propósito general; es preferible usar en su lugar la directiva `res` dentro de un bloque de datos declarado con las directivas `udata` o `udata_shr`.

Ejemplo 4.30

El siguiente segmento de programa ilustra el uso de las directivas `equ` y `set`.

```

REG1    equ      20h      ; REG1 es el registro 20h de la memoria de datos.
REG2    equ      21h      ; REG2 es el registro 21h de la memoria de datos.
DATO    set      .15       ; DATO es un dato cuyo valor inicial es 15.

        movlw   DATO
        movwf   REG1      ; En el registro REG1 se almacena 15.

DATO++                         ; Se modifica el valor del símbolo DATO.

        movlw   DATO
        movwf   REG2      ; En el registro REG2 se almacena 16.

```

Si `DATO` se define con una directiva `equ`, entonces en la línea donde aparece la expresión `DATO++`, que intenta modificar el símbolo `DATO`, el ensamblador genera un mensaje de error.

Directiva `#include`

La sintaxis de la directiva `#include` (incluir un archivo fuente adicional) es:

```

#include archivo
#include "archivo"
#include <archivo>

```

En esta directiva, archivo es el nombre completo de un archivo de texto. Si el nombre del archivo contiene espacios en blanco, entonces se deben utilizar las comillas ("archivo") o los corchetes (<archivo>).

La directiva #include inserta el texto completo del archivo especificado, en la posición que ocupa la directiva dentro del programa fuente.

La directiva #incluye se usa comúnmente para insertar en el programa fuente el archivo de definición (.inc) que contiene las definiciones de los nombres de los registros de funciones especiales y bits del microcontrolador previamente declarado en las directivas list o processor. Así, el usuario no tiene necesidad de declarar esos nombres en el programa fuente.

Ejemplo 4.31

Al comenzar la escritura de un programa fuente en lenguaje ensamblador, es usual declarar el tipo de procesador y definir los nombres de los registros especiales y bits y sus direcciones. Para ello se utilizan las directivas list e #include. El siguiente segmento de programa ilustra cómo hacerlo si el microcontrolador seleccionado es el PIC16F873 y su archivo de definición se nombra P16F873.INC.

```
list      p = p16f873
#include    p16f873.inc
```

Directiva org

La sintaxis de la directiva org (origen del programa) es:

[etiqueta]org expresión

Esta directiva asigna al seudocontador de programa del ensamblador el valor de la expresión, es decir, hace que las instrucciones que siguen a la directiva sean ensambladas a partir de la dirección dada por el valor de la expresión. Si la directiva usa una etiqueta, ésta recibe el valor de la expresión.

La directiva org se usa cuando se quiere colocar un programa o una porción de programa a partir de una dirección absoluta determinada.

Ejemplo 4.32

La directiva org se usa comúnmente para indicarle al ensamblador las direcciones de memoria correspondientes al reset (dirección 0) y al vector de interrupciones (dirección 4). El siguiente segmento de programa ilustra esta situación.

```
list      p = p16f873
#include    p16f873.inc

org      0      ; El seudo PC se pone en 0.
movlw   high PP ; Esta instrucción se coloca en la dirección 0.
```

```

movwf PCLATH ; Esta instrucción se coloca en la dirección 1.
goto PP       ; Esta instrucción se coloca en la dirección 2.

org    4      ; El seudo PC se pone en 4.

;
; Aquí se colocan las instrucciones de la subrutina de atención a interrupciones,
; que se ensamblan a partir de la dirección 4.
;

;
; Ahora se quiere comenzar el programa principal (PP) a partir de la dirección 800h
;
PP:   org    800h   ; El seudo PC se pone en 800h.
;
; A partir de aquí se escribe el programa principal
;
```

Directiva end

La sintaxis de la directiva end (fin del programa fuente) es:

end

Con esta directiva se indica al ensamblador que finalice el ensamblaje del programa fuente. La directiva end ocupa la última línea del programa fuente: el ensamblador ignora las líneas de programa que siguen a la línea donde está la directiva end.

4.3.3.2 Directivas utilizadas en la codificación relocable

Las directivas que se estudian en este apartado son las más utilizadas en los programas escritos según la modalidad de codificación relocable. En estos programas, la traducción al lenguaje de máquina la comienza el ensamblador y la concluye el enlazador, pues en el programa fuente no se especifican, en general, direcciones absolutas, y es el enlazador el que coloca las direcciones de instrucciones y datos.

Con estas directivas se puede:

- Indicar simbólicamente el inicio de bloques de instrucciones o de datos.
- Reservar espacio en la memoria de datos para las variables del programa.
- Indicar el alcance de los símbolos, es decir, símbolos globales y externos, en proyectos con varios archivos fuente.

- Seleccionar cómodamente la página de la memoria de programa o el banco de registros en la memoria de datos.

Directiva code

La sintaxis de la directiva code (comenzar una sección de código relocalizable) es la siguiente:

[etiqueta]code [dirección_ROM]

Esta directiva declara el comienzo de una *sección* o grupo de instrucciones del programa fuente que dará lugar a un bloque de código relocalizable o absoluto. La dirección de comienzo de la sección es el valor del campo dirección_ROM; si este campo no es especificado en la directiva, la dirección inicial la decide el enlazador. El campo etiqueta sirve para denominar la sección.

Dos secciones no pueden tener el mismo nombre. Si no se usa el campo etiqueta, la sección recibe el nombre .code

Una sección de código finaliza cuando comienza otra sección de código o de datos o cuando se alcanza la directiva end.

Ejemplo 4.33

El segmento de programa siguiente ilustra cómo usar la directiva code para declarar secciones de código absoluto o relocalizable.

```
Rst_vector    code  0      ; Esta sección comienza en la dirección 0.
    movlw  high PP
    movwf  PCLATH
    goto   PP
Intr_vector   code  4      ; Esta sección comienza en la dirección 4.
    goto   SR_Int
Intr_Prog     code  5      ; Esta sección comienza en la dirección 5.
SR_Int:
;
; Aquí se ponen las instrucciones de la subrutina de atención a la interrupción.
;
Prog_Principal code   ; La dirección inicial de la sección la pone el enlazador.
PP:
;
; A partir de aquí se escribe el programa principal
;
```

Directivas udata, udata_shr y res

Las directivas **udata** (comenzar una sección de datos no iniciados) y **udata_shr** (comenzar una sección compartida de datos no iniciados) tienen la sintaxis siguiente:

[etiqueta]udata	[dirección_RAM]
[etiqueta]udata_shr	[dirección_RAM]

Estas directivas permiten declarar el inicio de secciones de datos. Se puede especificar una dirección de comienzo de la sección en la memoria de datos mediante el campo dirección_RAM; en caso contrario, el enlazador decide la ubicación de la sección.

La diferencia entre las directivas es la siguiente: **udata** se usa para secciones de registros que están en un único banco de registros de la memoria de datos; **udata_shr** se usa para declarar secciones que comparten más de un banco de memoria. Por ejemplo, en un PIC16F873, los registros que ocupan las direcciones 20h a 7Fh del banco 0, están repetidos en direcciones similares del banco 2 (figura 3.11); por lo tanto, al referirse a ese conjunto de registros hay que usar la directiva **udata_shr** en lugar de **udata**.

El término “datos no iniciados” significa que a los datos de la sección no se les coloca un valor inicial determinado en el momento de definirlos. Para definir datos no iniciados se usa la directiva **res**.

La sintaxis de la directiva **res** (reservar memoria de datos) es la siguiente:

[etiqueta] res	cantidad_memoria
----------------	------------------

Esta directiva hace avanzar el contador de localizaciones en un valor igual al parámetro **cantidad_memoria**. Sirve para separar espacio de memoria de datos, sin asignar un valor inicial al espacio separado. Se usa dentro de las secciones de datos declarados con las directivas **udata** y **udata_shr**.

Ejemplo 4.34

El siguiente segmento de programa ilustra el uso de las directivas **udata_shr** y **res**.

```
udata_shr
REG1    res      1
REG2    res      1
```

Se declara una sección compartida de datos no iniciados, con los símbolos REG1 y REG2 a los que se les reserva una celda de memoria para cada uno. Se deja que el enlazador asigne las direcciones a estos registros.

Directivas global y extern

La sintaxis de las directivas **global** (exportar un símbolo) y **extern** (declarar un símbolo definido externamente) es:

```
global      símbolo [, símbolo ...]
extern      símbolo [, símbolo ...]
```

Estas dos directivas se usan cuando se tienen varios módulos para enlazar y hay símbolos definidos en un módulo pero que se usan en otro. La directiva **global** declara símbolos definidos en el módulo actual pero que deben estar disponibles para otros módulos. La directiva **extern** declara símbolos que se usan en el módulo actual pero que han sido definidos en otro módulo (en el que han sido declarados como globales).

Ejemplo 4.35

Se tiene un proyecto con dos módulos. El primer módulo contiene el programa principal en el archivo pp.asm. En el segundo módulo están las subrutinas llamadas desde el programa principal; este módulo está en el archivo sr.asm. Entre las subrutinas definidas en este módulo está la subrutina Demora, la cual produce una demora proporcional al valor colocado en el registro nombrado con el símbolo REG. En el programa principal se hace una llamada a esta subrutina y se le pasa el valor de la demora en el registro REG.

Para que el proceso de ensamblaje y enlace transcurra sin errores, en el módulo del programa principal se declara el símbolo Demora como externo, se define el símbolo REG como parte de una sección de datos, y se declara como un símbolo global. Por otra parte, en el módulo de las subrutinas, el símbolo Demora se define (como una etiqueta) dentro de una sección de programa y se declara como símbolo global, mientras que el símbolo REG se declara externo.

En el módulo del programa principal (archivo pp.asm):

```
;
; Módulo con el programa principal.
; Ilustrando las directivas global y extern
;

list      p = 16f873
#include p16f873.inc

udata_shr
REG      res    1      ; Se define el símbolo REG.

global   REG    ; El símbolo REG se declara global en este módulo
              ; y externo en el módulo de las subrutinas.

extern   Demora ; El símbolo Demora se declara externo en este módulo
              ; y se define y declara global en el módulo de las subrutinas.

;
```

: En una de las secciones del programa fuente (por ejemplo, en la sección Program) está
: la llamada a la subrutina Demora:

;

Program code

;

```
    movlw 35h
    movwf REG
    call Demora
```

;

```
    end
```

En el módulo de las subrutinas (archivo sr.asm):

;

;

: Módulo con las subrutinas
: ilustrando las directivas global y extern.

;

```
list p = 16f873
#include p16f873.inc

global Demora ; El símbolo Demora se declara global en este módulo
; y externo en el módulo del programa principal.

extern REG ; El símbolo REG se declara externo en este módulo
; y global en el módulo del programa principal.
```

;

;

: En una de las secciones del programa fuente (por ejemplo, en la sección Program) está
: definida la subrutina Demora:

;

Program code

;

Demora:

```
decfsz REG,1
goto Demora
return
```

;

```
end
```

Directivas pagesel, banksel y bankisel

La sintaxis de las directivas pagesel (selección de página de memoria), banksel (selección directa de banco de registros) y bankisel (selección indirecta del banco de registros) es:

```
pagesel etiqueta
banksel etiqueta
bankisel etiqueta
```

La directiva pagesel produce el código necesario para seleccionar la página de la memoria de programa donde se encuentre la etiqueta especificada en la directiva. Esta directiva intercala en el programa instrucciones necesarias para modificar convenientemente el registro PCLATH. Si el microcontrolador tiene sólo una página de memoria de programa, la directiva no genera ningún código.

Las directivas banksel y bankisel producen el código necesario para seleccionar el banco de registros donde se encuentre la etiqueta especificada en la directiva, según se quiera usar direccionamiento directo o indirecto respectivamente. La directiva banksel equivale a intercalar instrucciones para manipular convenientemente los bits RP1 y RP0 del registro STATUS, mientras que bankisel manipula el bit IRP de ese registro para darle el valor apropiado.

Ejemplo 4.36

El siguiente programa ilustra el uso de las directivas pagesel, banksel y bankisel.

```
list      p=16f873
#include p16f873.inc

; Datos constantes:
DATO1 equ    0x55
DATO2 equ    .10

; Registros de la memoria de datos:
        udata_shr
REG1   res    1      ; REG1 es el registro 20h de la memoria de datos.
REG2   res    1      ; REG2 es el registro 21h de la memoria de datos.

; Programas:
Rst_vector code 0
    pagesel PP      ; Se selecciona la página donde está PP
    goto    PP      ; garantizando un salto al lugar correcto.
```

Prog_Principal code

PP:

pagesel SRutina ; Se selecciona la página donde está la subrutina
call SRutina ; garantizando una llamada correcta.

: Se opera con los registros TRISB y PORTB usando direccionamiento directo:

banksel TRISB ; Seleccionar el banco 1 pues TRISB está en ese banco
; asegurando el direccionamiento correcto de TRISB.
clrf TRISB ; Se opera con TRISB.
banksel PORTB ; Se vuelve al banco 0, pues PORTB está en ese banco.
movf PORTB, DATO1 ; Se opera con PORTB.

: Se opera con REG1 usando direccionamiento indirecto:

movlw REG1
movwf FSR ; Se carga la dirección de REG1 en FSR.
bankisel REG1 ; Se selecciona el banco donde está REG1.
movlw DATO2 ; Se escribe un 10 en
movwf INDF ; REG1 usando direccionamiento indirecto.

SRutina:

Aquí van las instrucciones de la subrutina.

return

end

4.3.4 Macroinstrucciones

Las *macroinstrucciones* o simplemente *macros*, son instrucciones definidas por el usuario sobre la base de las instrucciones del microcontrolador y las directivas del ensamblador. Una vez ha sido definida una macro, se la puede llamar o invocar en el programa fuente. Una macroinstrucción se define con la siguiente sintaxis:

```
nombre_macro macro [arg_def1, arg_def2, ...]
    [ local etiqueta [, etiqueta, etiqueta, ...] ]
;
; Cuerpo de la macro
;
endm
```

donde nombre_macro es el símbolo que da nombre a la macroinstrucción y arg_def1, arg_def2, etc. son los argumentos opcionales con los que se define la

macro. Estos argumentos de definición son símbolos utilizados en el cuerpo de la macro.

La primera línea de la definición de una macroinstrucción está constituida por la directiva `macro`, con la que se declaran el nombre de la macro (`nombre_macro`) y sus argumentos (`arg_def1`, `arg_def2`, etc), si los hay. La última línea la ocupa la directiva `endm`, con la que se le comunica al ensamblador que la macro ha llegado a su fin. Por cada directiva `macro` debe haber una directiva `endm`.

El cuerpo de la macro está constituido por el conjunto de instrucciones y directivas que programan el algoritmo que el usuario ha decidido programar como una macroinstrucción. En el cuerpo de la macro se usan, formando parte de los operandos de instrucciones y directivas, los argumentos declarados en la directiva `macro`.

Es común que en el cuerpo de la macro se utilicen etiquetas con carácter local. Estas etiquetas deben ser declaradas en el cuerpo de la macro con una directiva `local`. Una vez que una etiqueta se ha declarado como local, no importa si existe otra con el mismo nombre fuera de la macro.

Para invocar una macro, basta con escribir su nombre en una línea del programa y especificar en su caso los argumentos con los que se ha de invocar la macro, tal como se indica a continuación:

nombre_macro [arg1, arg2, ...]

Al invocar una macro, el ensamblador inserta el cuerpo de la macro en el programa fuente, en el lugar donde ha sido invocada. Al desplegar instrucciones y directivas, el ensamblador utiliza los argumentos `arg1`, `arg2`, ... para sustituir los símbolos empleados como argumentos en la definición de la macro. Los argumentos con los que se invoca a una macro pueden ser símbolos o expresiones.

Ejemplo 4.37

En el siguiente programa se define la macro `Convierte`, que se invoca dos veces en el transcurso del programa. Esta macro recibe un dígito hexadecimal en un registro nombrado `HEXA` y entrega en otro registro, nominado `ASCII`, el equivalente ASCII del dígito. Por ejemplo, si `HEXA` = `0Ah`, entonces `ASCII` = `41h`. La conversión a ASCII se realiza sumando `30h` al dígito hexadecimal si éste es menor o igual que `9` o sumándole `37h` si el dígito hexadecimal es mayor que `9`.

```
list      p=16f873
#include p16f873.inc
;
```

Definición de macros:

Esta macro convierte a ASCII el dígito hexadecimal (0 a F) dado en el registro denominado HEXA. El código ASCII se entrega en el registro denominado ASCII.

```
Convierte    macro      HEXA, ASCII ; Declaración de la macro.
local       suma30, suma37, fin; Etiquetas locales.
movf       HEXA, W      ; En W el dígito hexadecimal.
sublw      9             ; W > 9 ? (Se afecta STATUS<C>).
movf       HEXA, W      ; En W el dígito hexadecimal (no se afecta STATUS<C>).
btfsr      STATUS, C    ; Sí (C=0), sumar 37h a HEXA.
goto       suma30        ; No (C=1), sumar 30h a HEXA.

suma37:
addlw     37h
goto      fin

suma30:
addlw     30h

fin:
movwf    ASCII          ; El resultado se deposita en el registro ASCII.
endm      ; Fin de la macro.
```

Registros de la memoria de datos:

```
udata_shr
HEXA1    res    1
HEXA2    res    1
ASCII1   res    1
ASCII2   res    1

: Programas

Rst_vector code    0

pagesel  PP          ; Se selecciona la página donde está PP
goto     PP          ; garantizando un salto al lugar correcto.
```

```
Prog_Principal code  0x800

PP:
    movlw   9
    movwf  HEXA1
    movlw   0Ah
    movwf  HEXA2

    Convierte  HEXA1, ASCII1      ; Se invoca la macro Convierte. El ensamblador
                                    ; inserta en este punto las instrucciones de la macro.
```

```

;           nop
;
Convierte    HEXA2, ASCII2      ; Se invoca otra vez la macro Convierte. El ensamblador
;           ; inserta en este punto, de nuevo, las instrucciones de la
;           ; macro.
;
;           nop
goto     $                      ; Lazo infinito.
;
end          ; Fin del programa.

```

4.3.5 Organización de un programa en lenguaje ensamblador

Aunque para organizar un programa escrito en lenguaje ensamblador no hay reglas rígidas, es recomendable seguir un orden como el siguiente:

1. Definir el procesador y sus símbolos mediante las directivas `list` e `#include`.
2. Si se van a utilizar macroinstrucciones, escribir la definición de las que se vayan a emplear en el programa.
3. Definir los símbolos con los que se representarán datos constantes, mediante las directivas `equ` y `set`.
4. Definir el uso de la memoria de datos, es decir, hay que definir los símbolos que se emplean en el programa para representar registros de propósito general de la memoria de datos y sus direcciones.
5. Escribir el cuerpo del programa principal, que, en general, comienza por iniciar las variables que requieran de un valor inicial determinado (por ejemplo, cero).
6. Escribir las subrutinas, si las hay.
7. Terminar el programa fuente con la directiva `end`.

Puede observarse que la definición de símbolos y sus valores, ya sean datos o direcciones, precede a las instrucciones del programa.

La forma de definir la ubicación de las instrucciones en la memoria de programa y los datos en la memoria de datos, depende de si la codificación del programa fuente será absoluta o relocalizable.

En la codificación absoluta, el ensamblador debe tener toda la información necesaria para codificar completamente el programa fuente. Ello significa que deben definirse todas las direcciones que se utilicen en el programa

fuente. Es decir, desde un inicio deben quedar definidas las direcciones donde comienzan los bloques de instrucciones del programa, así como las direcciones de los registros de propósito general que se utilicen en el programa. La definición de las direcciones del programa se hace con directivas org. Las direcciones de los registros se especifican con la directiva equ.

En la codificación relocalizable, el ensamblador hace una codificación incompleta del programa fuente, pues no se especifican todas las direcciones de datos e instrucciones. El enlazador, sobre la base del modelo del microcontrolador seleccionado, ubica finalmente las instrucciones y los datos. En este caso, el cuerpo del programa se puede organizar en secciones, cada una de ellas comenzando con una directiva code. Para algunas de estas secciones, como por ejemplo las correspondientes al vector de reset y al vector de interrupciones, que deben quedar ubicadas en lugares fijos de la memoria de programa, se especifica la dirección de comienzo; para otras secciones, que pueden quedar ubicadas en cualquier lugar de la memoria de programa, no se especifica su dirección de comienzo y se deja al enlazador la labor de ubicarlas en la memoria. Con la memoria de datos sucede algo parecido. En este caso, en el programa fuente solamente se declaran los nombres de los registros que se usan en el programa y se reserva espacio de memoria sin especificar direcciones. Esto se hace con las directivas udata, udata_shr y res.

Los ejemplos 4.38 y 4.39 muestran la estructura recomendada para los programas fuente que se van a ensamblar con codificación absoluta y codificación relocalizable, respectivamente. Se recomienda al lector que los analice detenidamente.

Ejemplo 4.38

El programa siguiente muestra cómo organizar el programa fuente en lenguaje ensamblador cuando se utilice codificación absoluta.

```
list      p=16f873      ; Se declara el microcontrolador que se va a utilizar
#include <p16f873.inc>    ; y se definen sus variables.
```

Definición de constantes:

```
DATO1      EQU 0x1      ;
DATO2      EQU 0x2      ;
```

Definición de variables:

```
w_temp     equ 0x20    ; Variable usada para salvar W.
status_temp equ 0x21    ; Variable usada para salvar STATUS.
```

```

X           equ    0x22 ; Ejemplo.
Y           equ    0x23 ; Ejemplo.
;
; Cuerpo del programa:
;
        org    0x000      ; Dirección del vector de reset.
        movlw high PP     ; Preparar el salto al programa principal,
        movwf PCLATH      ; garantizando la selección de la página correcta.
        goto   PP          ; Saltar a la dirección donde comienza el programa principal.

        org    0x004      ; Dirección del vector de interrupciones.
        movwf w_temp       ; Salvar el contenido actual de W.
        movf   STATUS, W    ; Copiar el contenido actual de STATUS en W,
        bcf    STATUS, RP0   ; asegurar la selección del banco 0
        movwf status_temp   ; y salvar el contenido de STATUS

; aquí se ponen las instrucciones de la subrutina de atención a la interrupción

        bcf    STATUS, RP0   ; Asegurar la selección del banco 0.
        movf   status_temp, W ; Recuperar la copia de STATUS
        movwf STATUS         ; y restituirla.
        swapf w_temp, f     ; Recuperar la copia de W
        swapf w_temp, W     ; y restituirla sin alterar STATUS.
        retfie               ; Retornar desde la interrupción.

PP:
        clrf   X            ; Iniciar variables.
        clrf   Y            ; Iniciar variables.
;
; Aquí se escriben instrucciones del programa principal
;
        movlw high SR1      ; Si SR1 está en una página diferente,
        movwf PCLATH        ; garantizar la selección de la página correcta
        call   SR1          ; y llamar a la subrutina.
;
; Aquí se escriben instrucciones del programa principal
;
        goto   $              ; Ejemplo: lazo infinito para terminar el programa principal.
;
SR1:
;
; Aquí se escriben instrucciones de la subrutina SR1.
;
        movlw high SR2      ; Ejemplo: llamar a SR2 que está en otra página,
        movwf PCLATH        ; garantizar la selección de la página correcta
        call   SR2          ; y llamar a la subrutina SR2.
;
```

```
; Aquí se escriben instrucciones de la subrutina SR1.

    return          ; Retornar al programa principal desde SR1.

SR2:                      ; Aquí comienza la subrutina SR2.

; Aquí se escriben las instrucciones de la subrutina SR2.

    return          ; Retornar a la subrutina SR1 desde SR2.

    end            ; Fin del programa fuente.
```

Ejemplo 4.39

El programa siguiente ilustra cómo organizar el programa fuente en lenguaje ensamblador cuando se utilice codificación relocalizable.

```
list      p=16f873      ; Se declara el microcontrolador que se va a utilizar
#include <p16f873.inc> ; y se definen sus variables.
```

Definición de constantes:

```
DATO1      equ     0x1      ; Ejemplo.
DATO2      equ     0x2      ; Ejemplo.
```

Definición de variables:

```
udata_shr
■_temp      res     1      ; Variable usada para salvar W.
status_temp  res     1      ; Variable usada para salvar STATUS.
X           res     1      ; Ejemplo.
Y           res     1      ; Ejemplo.
```

Cuerpo del programa:

```
Reset_vector code   0      ; Vector de reset en la dirección 0.

        pagesel PP      ; Preparar el salto al programa principal y
        goto    PP      ; saltar a la dirección donde comienza.

Interrupt_vector code   4      ; Vector de interrupciones en la dirección 4.

        goto    SR_Int   ; Saltar a la subrutina de atención a interrupciones.
```

```
Intr_Prog      code   5      ; Sección con la subrutina de atención a interrupciones
SR_Int:
    movwf w_temp          ; Salvar el contenido actual de W.
    movf  STATUS, W        ; Copiar el contenido actual de STATUS en W,
    bcf   STATUS, RP0       ; asegurar la selección del banco 0
    movwf status_temp      ; y salvar el contenido de STATUS.
```

; Aquí se ponen las instrucciones de la subrutina de atención a la interrupción.

```
    bcf   STATUS, RP0       ; Asegurar la selección del banco 0.
    movf status_temp, W     ; Recuperar la copia de STATUS
    movwf STATUS            ; y restituirla.
    swapf w_temp, f         ; Recuperar la copia de W
    swapf w_temp, W         ; y restituirla sin alterar STATUS.
    retfie                  ; Retornar desde la interrupción.
```

;

Prog_Principal code ; En esta sección está el programa principal.

PP:

```
    clrf X                 ; Iniciar variables
    clrf Y                 ; Iniciar variables
```

;

; Aquí se escriben instrucciones del programa principal.

```
;
```

```
    pagesel SR1             ; Se selecciona la página donde está SR1.
    call   SR1               ; Se llama a la subrutina SR1.
```

;

; Aquí se escriben instrucciones del programa principal.

```
;
```

```
    goto   $                 ; Ejemplo: lazo infinito para terminar el programa principal.
```

Subrutinas code ; En esta sección están las subrutinas.
SR1: ; Aquí comienza la subrutina SR1.

;

; Aquí se escriben instrucciones de la subrutina SR1.

```
;
```

```
    pagesel SR2             ; Ejemplo: desde la subrutina SR1 se llama
    call   SR2               ; a la subrutina SR2.
```

;

; Aquí se escriben instrucciones de la subrutina SR1.

```
;
```

```
    return                  ; Retornar al programa principal.
```

SR2: ; Aquí comienza la subrutina SR2.

;

; Aquí se escriben las instrucciones de la subrutina SR2.

```

;                                         ; Retornar a la subrutina SR1.

return                                ; Fin del programa fuente.

```

Las macroinstrucciones y las subrutinas son dos valiosos recursos de programación que ayudan a lograr una adecuada estructura modular del programa fuente. Ambos recursos permiten al programador escribir un determinado algoritmo una sola vez y utilizarlo todas las veces que sea necesario, empleando en cada ocasión argumentos diferentes. Permiten, además, reducir el tiempo de programación, facilitan la detección de errores y hacen que los programas sean más fáciles de interpretar. La diferencia fundamental entre ambos recursos es que mientras las macroinstrucciones se invocan o llaman durante la etapa de ensamblaje del programa fuente, las subrutinas son llamadas durante la fase de ejecución del programa.

El grupo de instrucciones que definen una macroinstrucción no ocupan ningún espacio en la memoria de programa puesto que esas instrucciones son ubicadas en memoria con la llamada a la macro, lo cual tiene lugar durante el ensamblaje del programa fuente. Cuando el ensamblador encuentra una llamada a una macroinstrucción, inserta en el programa fuente, en el lugar de la llamada, las instrucciones que definen la macro, con la sustitución de los argumentos de definición correspondientes, si los hay. Así, la cantidad de memoria de programa utilizada aumentará con el número de llamadas. Si una macro es invocada un cierto número de veces, sus instrucciones se insertan en el programa fuente cada vez que se llame, haciendo así que crezca el tamaño del programa.

Las subrutinas se definen una vez y son ubicadas por el ensamblador o el enlazador en algún lugar de la memoria de programa. Las subrutinas se llaman durante la ejecución del programa mediante la instrucción call. El efecto de una llamada a una subrutina es derivar la ejecución del programa hacia las instrucciones que forman la subrutina, ejecutándose éstas en el orden que establezca el algoritmo programado en la subrutina. El retorno al programa desde donde se realizó la llamada se hace con una instrucción return o similar. La repetición de la llamada a una subrutina no aumenta el tamaño del programa en cuanto a cantidad de memoria que éste ocupa, sino que, en todo caso, aumenta el tiempo de ejecución del programa por el mayor número de instrucciones que se ejecutan.

La ejecución de una subrutina requiere tres elementos: la instrucción de llamada, la instrucción de retorno y la pila. El objetivo fundamental de la pila

es almacenar la dirección de retorno al programa desde donde se llamó a la subrutina. Las instrucciones de llamada y de retorno, que tienen que guardar y extraer de la pila, añaden un cierto valor, en general poco significativo, al tiempo de ejecución de las instrucciones que conforman el cuerpo de la subrutina.

Una ventaja adicional al trabajar con subrutinas es que se pueden construir y utilizar bibliotecas de subrutinas. Una *biblioteca* es un archivo que contiene una colección de programas (subrutinas) sobre un tema. Durante el proceso de ensamblaje, el enlazador toma de la biblioteca solamente las subrutinas que son solicitadas y las incorpora en el programa objeto.

La elección entre una macroinstrucción o una subrutina para programar un determinado algoritmo que forma parte de un programa, es una cuestión que depende mucho de los gustos y preferencias del programador. No obstante, en los microcontroladores, donde en general no abunda la memoria de programa disponible, si el algoritmo es llamado varias veces en el programa, puede ser mejor utilizar subrutinas que macroinstrucciones.

4.4 Recursos disponibles para programar en el lenguaje ensamblador de los microcontroladores PIC

Para programar aplicaciones en ensamblador sobre los microcontroladores PIC de gama media, se utilizan los siguientes programas:

Editor de textos. Se utiliza para crear el programa fuente (archivo .asm).

Ensamblador (MPASM.EXE o MPASMWIN.EXE). Toma el programa fuente (en el archivo .asm) y realiza su traducción completa o parcial al lenguaje de máquina. Si la traducción es completa, el ensamblador entrega un archivo hexadecimal (.hex) con el programa codificado; si la traducción es parcial, entrega un archivo objeto (.o) que sirve de entrada al enlazador MPLINK.

Enlazador (MPLINK.EXE). Conforma el programa en lenguaje de máquina (en un archivo .hex) al enlazar, en un único módulo, uno o más módulos objetos (archivos .o) producidos por el ensamblador, y archivos de bibliotecas (.lib) producidos por el gestor de bibliotecas.

Gestor de bibliotecas (MPLIB.EXE). Crea una biblioteca (archivo .lib) a partir de varios programas ensamblados. Una biblioteca es una colección de programas.

Simulador/depurador. Es un programa que permite simular en el ordenador personal el funcionamiento del microcontrolador. Dispone de órdenes que facilitan la prueba de programas y la depuración de errores. Este programa está incluido en el entorno de desarrollo integrado MPLAB.

Programador. Es el programa que, junto con algún hardware, realiza la programación del microcontrolador PIC. La información de entrada al programador es básicamente el archivo hexadecimal (.hex) producido por el ensamblador o el enlazador.

Entorno de desarrollo integrado MPLAB. Es un sistema que proporciona un entorno muy cómodo y amistoso para editar, ensamblar, enlazar y depurar programas para un microcontrolador PIC. Con el suplemento de hardware adecuado, permite también programar el microcontrolador.

4.4.1 El ensamblador MPASM

El ensamblador MPASM realiza la traducción al lenguaje de máquina del programa fuente (contenido en el archivo .asm). En el ordenador personal, el ensamblador MPASM se puede ejecutar aislado (como MPASM en una línea de órdenes del sistema operativo DOS, o sobre el sistema operativo Windows, como MPASMWIN), o dentro del entorno de desarrollo integrado MPLAB. La figura 4.9 muestra el proceso de ensamblaje y sus archivos asociados.

El ensamblador puede generar código absoluto o relocalizable, según que la traducción al lenguaje de máquina sea completa o parcial, respectivamente. Si la traducción es completa, el ensamblador entrega un archivo hexadecimal (.hex) con el programa codificado; si la traducción es parcial, entrega un archivo objeto (.o) que sirve de entrada al enlazador MPLINK.

4.4.1.1 Generación de código absoluto

En este caso, el MPASM realiza por sí mismo todo el proceso de traducción al lenguaje de máquina. El resultado del ensamblaje es un archivo con el programa completamente codificado. La codificación absoluta puede hacerse si el programa fuente contiene toda la información que necesita el ensamblador para traducir al lenguaje de máquina. Es decir, en el programa fuente están especificadas con exactitud las direcciones de la memoria donde se ubicarán las instrucciones y las direcciones de los registros de la memoria de datos que se utilizan en el programa. Al tener esta información, el ensamblador puede codificar completamente el programa fuente.

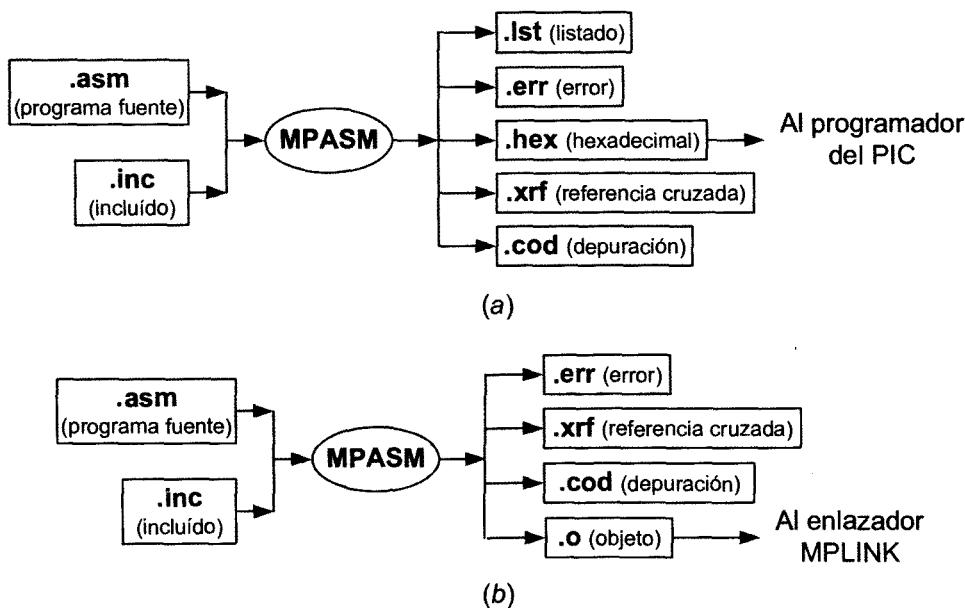


Figura 4.9 Proceso de ensamblaje con el MPASM y archivos involucrados en el proceso. (a) Ensamblado con codificación absoluta. El archivo hexadecimal (.hex) es el que contiene el programa traducido al lenguaje de máquina, listo para ser grabado en la memoria del microcontrolador. (b) Ensamblado con codificación relocalizable. La traducción al lenguaje de máquina es parcial y está contenida en el archivo objeto (.o), el cual debe ser procesado a continuación con el enlazador MPLINK.

Para la codificación absoluta, el programa fuente debe estar contenido completamente en un archivo fuente (.asm) o con partes del programa en otros archivos que sean incluidos en el proceso de ensamblaje desde el archivo fuente (con directivas `#include`). Como resultado del ensamblaje, el MPASM entrega el programa codificado completamente en lenguaje de máquina en un archivo hexadecimal (.hex). Este archivo puede ser transmitido al programador del microcontrolador para programarlo.

4.4.1.2 Generación de código relocalizable

En este caso, el MPASM realiza una codificación parcial del programa fuente, labor que es completada por el enlazador MPLINK. Si la información de dirección en el programa fuente no es completa, el ensamblador sólo puede realizar una codificación parcial e incompleta del programa fuente (codificación relocalizable) y hace falta que el enlazador MPLINK termine la codificación y obtenga el programa en lenguaje de máquina.

El resultado del ensamblaje por parte del MPASM es un archivo objeto (.o) que sirve de entrada al enlazador MPLINK. El enlazador puede recibir

varios archivos de este tipo y enlazarlos hasta formar un único programa objeto en lenguaje de máquina, el cual es entregado en un archivo hexadecimal (.hex) por el MPLINK. El archivo hexadecimal se puede transmitir al programador del microcontrolador para programarlo.

4.4.1.3 Archivos involucrados en el ensamblaje

En el ensamblaje intervienen varios archivos, unos requeridos por el ensamblador como archivos de entrada y otros producidos por el ensamblador como archivos de salida. Los archivos generados durante el proceso de ensamblaje están descritos en la ayuda del sistema integrado MPLAB, que se puede descargar desde el sitio web de Microchip Technology.

Los archivos más importantes son:

Archivo fuente (.asm). Archivo de texto con el programa fuente. Es el archivo que escribe el programador utilizando un editor de textos cualquiera, como puede ser el editor incluido en el entorno integrado MPLAB. Es un archivo de entrada al MPASM.

Archivo fuente incluido (.inc). Archivo de texto con parte del programa fuente, que es incluido en el archivo fuente (.asm) mediante una directiva #include. El uso más frecuente de este tipo de archivo es para definir los nombres y direcciones de los registros y bits del microcontrolador que se va a utilizar. Como parte de las herramientas y recursos para programar en ensamblador, Microchip suministra un archivo de este tipo para cada microcontrolador; por ejemplo, el archivo 16f873.inc contiene las definiciones de los nombres de los registros de funciones especiales, sus direcciones y los nombres de los bits en el PIC16F873. Para usar de los nombres definidos en el archivo, el programador sólo tiene que incluir ese archivo en su programa fuente. Ver el ejemplo 4.19.

Archivo de listado (.lst). Archivo de texto que contiene el listado del programa fuente, sus direcciones y la codificación de las instrucciones. Este archivo puede ser generado por el MPASM o por el MPLINK.

Archivo objeto (.o). Archivo producido por el ensamblador como resultado de la codificación relocalizable del programa fuente. Contiene la codificación incompleta del programa fuente realizada por el ensamblador. Es un archivo de entrada al enlazador MPLINK. No es un archivo de texto.

Archivo hexadecimal (.hex). Archivo de texto que contiene los códigos de las instrucciones y sus direcciones en el formato hexadecimal de Intel. Este archivo puede ser generado por el MPASM o por el MPLINK: es genera-

do por el MPASM si el programa fuente ha sido escrito para su codificación absoluta, y por el enlazador MPLINK en caso contrario. Este archivo es el producto final del ensamblaje o ensamblaje y enlace.

El archivo hexadecimal está constituido por un conjunto de líneas de texto (*records*), cada una de las cuales tiene el formato siguiente:

:LLAAAATTDDDD...DDSS

donde:

: es el carácter ASCII que indica el comienzo de un *record*.

LL Es la longitud del *record*. Son dos caracteres ASCII que indican la cantidad (en hexadecimal) de datos que contiene el *record*.

AAAA Es la dirección inicial del *record*. Son cuatro caracteres ASCII que representan la dirección del primer dato (byte) del *record*. En los PIC de gama media, dado que las celdas de memoria de programa son de 14 bits y su contenido se da en 2 bytes, las direcciones que aparecen en este campo son el doble de las reales.

TT Es el tipo de *record*. Básicamente, 00 indica que el *record* actual es de datos y 01 que se trata del último *record* del archivo hexadecimal.

DD... Son los datos. Dos caracteres ASCII en este campo representan el valor hexadecimal de un dato (un byte). En los PIC de gama media, dado que las celdas de memoria de programa son de 14 bits, el contenido de una celda se expresa con 4 caracteres ASCII.

SS Es la suma de comprobación. Se calcula sumando todos los bytes (no los códigos ASCII) del record; **SS** es el complemento a 2 de esa suma en 8 bits.

Ejemplo 4.40

Este ejemplo ilustra la estructura de los archivos de listado y hexadecimal obtenidos en el ensamblaje de un programa fuente trivial. El archivo de listado sólo se muestra parcialmente.

Archivo fuente (ejemplo.asm):

```

list      p=16f873
#include  <p16f873.inc>

X        equ      0x20
Y        equ      0x21

org      0x000
        high PP      ; Dirección del vector de reset.
        movlw

```

```

movwf PCLATH
goto PP

org 0x004 ; Dirección del vector de interrupciones.

org 0x0123

PP:
    clrf X ; Hacer X = 0.
    clrw
    addlw 1
    movwf Y ; Hacer X = 1.

end ; Fin del programa fuente.

```

Archivo de listado (ejemplo.lst) (se muestra parcialmente)

MPASM 03.90.01 Released EJEMPLO.ASM 8-7-2005 12:35:51 PAGE 1

LOC	OBJECT CODE	LINE	SOURCE	TEXT
				VALUE
00001			list	p=16f873
00002			#include	<p16f873.inc>
00001			LIST	
00002			P16F873.INC Standard Header File, Version 1.00 Microchip Technology, Inc.	
00358			LIST	
00003				
00000020	00004	X	equ	0x20
00000021	00005	Y	equ	0x21
00006				
0000	00007		org	0x000 ; Dirección del vector de reset.
0000 3001	00008		movlw	high PP
0001 008A	00009		movwf	PCLATH
0002 2923	00010		goto	PP
00011				
0004	00012		org	0x004 ; Dirección del vector de interrupciones.
0004 0009	00013		retfie	
00014				
0123	00015		org	0x0123
0123	00016	PP:		
0123 01A0	00017		clrf X	; Hacer X = 0.
0124 0103	00018		clrw	
0125 3E01	00019		addlw 1	
0126 00A1	00020		movwf Y	; Hacer X = 1.
00021				
00022			end	; Fin del programa fuente.

Archivo hexadecimal (ejemplo.hex)

```
:0600000001308A002329F3
:020008000900ED
```

:08024600A0010301013EA1002B
:00000001FF

4.4.2 El enlazador MPLINK

El enlazador MPLINK conforma el programa en lenguaje de máquina en un archivo hexadecimal (.hex) a partir de:

- Uno o más archivos objetos (.o) producidos por el ensamblador.
- Uno o más archivos de bibliotecas (.lib), producidos por el gestor de bibliotecas MPLIB.
- Un archivo auxiliar (.lkr) con la descripción de la memoria disponible en el microcontrolador u otros datos de interés para el enlace.

La figura 4.10 ilustra el proceso de enlace y los archivos involucrados. En el ordenador personal, el enlazador MPLINK se puede ejecutar aislado (como MPLINK en una línea de órdenes del sistema operativo DOS) o dentro del entorno de desarrollo integrado MPLAB.

El archivo auxiliar (.lkr) sirve básicamente para informar al enlazador sobre las direcciones de la memoria de programa y de datos disponibles en el microcontrolador seleccionado, de modo que el enlazador pueda asignar direcciones correctamente y completar con ello el ensamblaje del programa. También puede contener otra información de interés para el enlace, como por ejemplo, el nombre de los archivos que hay que enlazar, si éstos no han sido especificados cuando se llama al enlazador. Todo esto se hace mediante directivas al enlazador. La descripción completa de las directivas al MPLINK está en el archivo de ayuda que, como parte del entorno de desarrollo integrado MPLAB, está disponible en el sitio web de Microchip Technology.

En el archivo auxiliar, las regiones de memoria disponibles se describen mediante las directivas (al enlazador) databank, sharebank y codepage. La directiva databank informa del nombre y las direcciones inicial y final de una región de la memoria de datos que está en un banco de registros. La directiva sharebank hace lo mismo pero respecto a una región que comparte direcciones en más de un banco de registros.

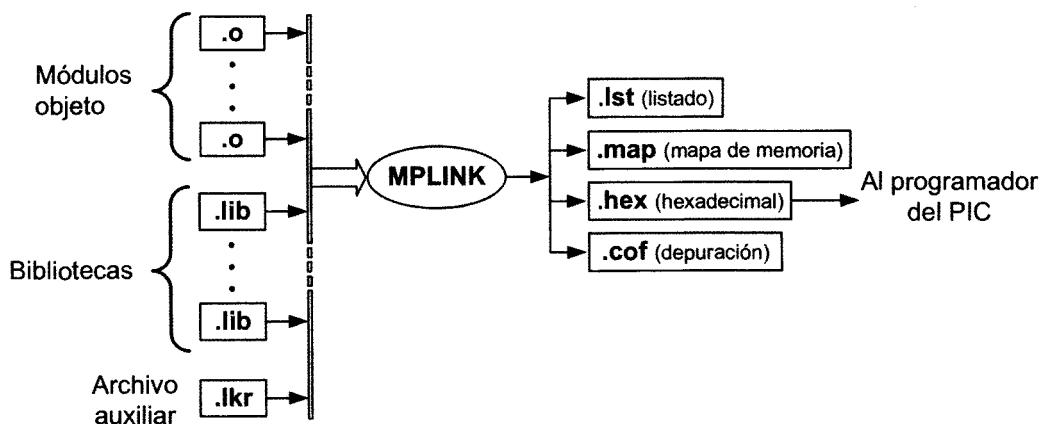


Figura 4.10 Proceso de enlace con el MPLINK y archivos involucrados en el proceso. Los archivos objeto (.o) se obtienen con el ensamblador MPASM como resultado de la codificación relocalizable de los respectivos módulos en lenguaje ensamblador. Los archivos de bibliotecas (.lib) se obtienen con el gestor de bibliotecas MPLIB. El archivo auxiliar (.lkr) contiene una descripción de la memoria disponible en el microcontrolador para que el enlazador pueda ubicar correctamente en ella el programa en lenguaje de máquina. El archivo hexadecimal (.hex) es el que contiene el programa traducido al lenguaje de máquina, listo para ser grabado en la memoria del microcontrolador.

La directiva codepage informa del nombre, las direcciones inicial y final de una región de la memoria de programa. Opcionalmente, se puede ordenar al enlazador que llene la región descrita con un dato (de 14 bits en los PIC de gama media). La región también puede ser declarada como “protegida”, lo cual implica ciertas restricciones en su uso desde el programa fuente.

La sintaxis de estas directivas es:

databank	name=nombre	start=dir_inicial	end=dir_final	[protected]
sharebank	name=nombre	start=dir_inicial	end=dir_final	[protected]
codepage	name=nombre	start=dir_inicial	end=dir_final	[protected] [fill=valor]

Para “conectar” las regiones de memoria declaradas mediante las directivas antes mencionadas con las secciones del programa fuente, se utiliza la directiva section. Esta directiva permite declarar una región de memoria como una sección lógica. Su sintaxis es:

section	name=nombre_sección	ROM=nombre
section	name=nombre_sección	RAM=nombre

donde nombre_sección es el símbolo que identifica a la sección y nombre es el nombre de la región de memoria, ya sea memoria de programa (ROM) o de datos (RAM), previamente declarada con las directivas databank, sharebank y co-

depage. Las secciones lógicas se pueden utilizar en el programa fuente mediante las directivas `udata`, `udata_shr` y `code`.

Ejemplo 4.41

En este ejemplo se muestra el contenido de un archivo auxiliar que contiene la descripción de las regiones de memoria de un microcontrolador PIC. También se ilustra cómo utilizar en el programa fuente las secciones lógicas declaradas en el archivo auxiliar `16f873.lkr` suministrado por Microchip como parte del sistema de desarrollo integrado MPLAB:

```
// Sample linker command file for 16F873
// $Id: 16f873.lkr, v 1.5 2002/11/07 23:16:07 sealep Exp $
```

```
LIBPATH .
```

CODEPAGE	NAME=vectors	START=0x0	END=0x4	PROTECTED
CODEPAGE	NAME=page0	START=0x5	END=0x7FF	
CODEPAGE	NAME=page1	START=0x800	END=0xFFFF	
CODEPAGE	NAME=.idlocs	START=0x2000	END=0x2003	PROTECTED
CODEPAGE	NAME=.config	START=0x2007	END=0x2007	PROTECTED
CODEPAGE	NAME=eedata	START=0x2100	END=0x217F	PROTECTED
DATABANK	NAME=sfr0	START=0x0	END=0x1F	PROTECTED
DATABANK	NAME=sfr1	START=0x80	END=0x9F	PROTECTED
DATABANK	NAME=sfr2	START=0x100	END=0x10F	PROTECTED
DATABANK	NAME=sfr3	START=0x180	END=0x18F	PROTECTED
SHAREBANK	NAME=gpr0	START=0x20	END=0x7F	
SHAREBANK	NAME=gpr0	START=0x120	END=0x17F	
SHAREBANK	NAME=gpr1	START=0xA0	END=0xFF	
SHAREBANK	NAME=gpr1	START=0x1A0	END=0x1FF	
SECTION	NAME=STARTUP	ROM=vectors	// Reset and interrupt vectors	
SECTION	NAME=PROG1	ROM=page0	// ROM code space - page0	
SECTION	NAME=PROG2	ROM=page1	// ROM code space - page1	
SECTION	NAME=IDLOCS	ROM=.idlocs	// ID locations	
SECTION	NAME=CONFIG	ROM=.config	// Configuration bits location	
SECTION	NAME=EEPROM	ROM=eedata	// Data EEPROM	

Puede observarse que en el archivo auxiliar hay declaradas varias secciones. Por ejemplo, la sección denominada `PROG1` corresponde a la región de memoria de programa denominada `page0` que comienza en la dirección `005h` y termina en la dirección `7FFh`.

En el programa fuente se puede indicar al enlazador que ubique instrucciones en esa sección lógica, precediendo dichas instrucciones con una directiva `code` acompañada por el nombre de la sección, de la forma siguiente:

```
PROG1 code
```

4.4.3 El gestor de bibliotecas MPLIB

Una *biblioteca* es aquí una colección de programas en un archivo. Comúnmente, es una colección de subrutinas relacionadas con algún tema, todas disponibles en un archivo de biblioteca (archivo .lib). Por ejemplo, se puede crear una biblioteca con un conjunto de subrutinas relacionadas con operaciones aritméticas, y llamarla math.lib.

El archivo con la biblioteca se puede enlazar con otros archivos resultantes del ensamblaje, tal como muestra la figura 4.10. Una ventaja que ofrece disponer de una biblioteca es que en un archivo único se puede tener la colección completa de programas relacionados con el tema. Al enlazar la biblioteca con otros archivos objeto, el enlazador toma de la biblioteca sólo aquellas subrutinas que han sido solicitadas desde otros módulos, de modo que el programa objeto en lenguaje de máquina no crece innecesariamente.

Para crear una biblioteca se utiliza el gestor de bibliotecas MPLIB. Este programa se puede invocar desde una línea de órdenes del sistema operativo DOS o desde el entorno de desarrollo integrado MPLAB. La figura 4.11 muestra este proceso.

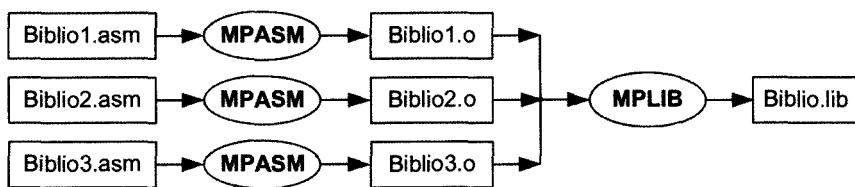


Figura 4.11 Proceso de obtención de una biblioteca con el MPLIB. Los diferentes componentes de la biblioteca son ensamblados con el MPASM, obteniéndose los archivos objetos (.o) correspondientes. Con el gestor de bibliotecas MPLAB, se unen en un archivo con extensión .lib.

Ejemplo 4.42

Procesos de obtención y uso de una biblioteca. La biblioteca que se va a construir (biblio.lib) estará formada por tres subrutinas triviales denominadas SR1, SR2 y SR3. Cada subrutina se ha programado en archivos independientes, denominados biblio1.asm, biblio2.asm y biblio3.asm. Los listados de estos archivos se muestran a continuación. Obsérvese el uso de la directiva global en los archivos.

Archivo biblio1.asm:

```

;
; Creación de una biblioteca.
; biblio1.asm: en este archivo está la subrutina SR1.
;
list      p = 16f873
```

```
#include p16f873.inc
global SR1
code
SR1:
nop
return
end
```

Archivo biblio2.asm:

```
;
; Creación de una Biblioteca.
; biblio2.asm: en este archivo está la subrutina SR2.
;
list      p = 16f873
#include  p16f873.inc
global   SR2
code
SR2:
nop
nop
return
end
```

Archivo biblio3.asm:

```
;
; Creación de una Biblioteca.
; biblio3.asm: en este archivo está la subrutina SR3.
;
list      p = 16f873
#include  p16f873.inc
global   SR3
code
SR3:
nop
nop
nop
return
end
```

Estos tres archivos se ensamblan con el MPASM y se procesan con el gestor de bibliotecas MPLIB, obteniéndose como resultado el archivo **biblio.lib**.

En el archivo **program.asm** está el programa fuente trivial, que utiliza una de las subrutinas de la biblioteca. El listado de este programa se muestra a continuación. Obsérvese el uso de la directiva **extern** en el archivo.

```
; Programa que utiliza una subrutina de la biblioteca biblio.lib
; Si se examina el archivo program.lst se verá que el enlazador
; incorpora en el programa objeto solamente la subrutina solicitada.
```

```
list      p = 16f873
#include  p16f873.inc
```

```

extern    SR2
Program code 0
;
call     SR2
;
end

```

Este programa se ensambla y enlaza con el archivo biblio.lib. Como resultado del enlace se obtiene el archivo program.hex con el programa objeto. También se obtiene un archivo de listado (program.lst). En este archivo se puede comprobar que el enlazador solamente ha incorporado la subrutina solicitada. El listado parcial del archivo program.lst se muestra a continuación:

Address	Value	Disassembly	Source
			; Programa que utiliza una subrutina de la biblioteca biblio.lib ; Si se examina el archivo program.lst se verá que el enlazador ; incorpora en el programa objeto solamente la subrutina solicitada.
			list p = 16f873
			#include p16f873.inc
			extern SR2
			Program code 0
			;
000000 2007	CALL 0x7	call SR2	
			;
			end
			;
			; Creación de una Biblioteca.
			; biblio2.asm: en este archivo está la subrutina SR2.
			;
			list p = 16f873
			#include p16f873.inc
			global SR2
			code
		SR2:	
000007 0000	NOP	nop	
000008 0000	NOP	nop	
000009 0008	RETURN	return	
			end

5 La entrada y salida en paralelo

Este capítulo trata sobre la entrada y salida (E/S) digital en paralelo, caracterizada porque todos los bits de un dato son transferidos simultáneamente; la entrada y salida en serie será tratada en el capítulo 8, y las entradas y salidas analógicas se estudiarán en el capítulo 9.

El capítulo comienza con la exposición de algunos conceptos básicos sobre los métodos de transferencia de datos y las técnicas existentes para el tratamiento de la entrada y salida digital de datos. A continuación se describen las características de los puertos paralelos de los microcontroladores PIC de clase media, con énfasis en su funcionamiento, aunque sin obviar aquellos aspectos de hardware más importantes. Finalmente, se ilustra la conexión y tratamiento de algunos periféricos de amplio uso en sistemas con microcontroladores PIC: interruptores, diodos LED, teclados matriciales, pantallas numéricas de 7 segmentos y módulos visualizadores LCD.

5.1 Conceptos básicos sobre entradas y salidas digitales

Un *periférico* es un dispositivo externo conectado al microcontrolador. Algunos posibles periféricos son: los interruptores (*switches*), diodos emisores de luz (LED: *Light Emitting Diode*), relés (relevadores), teclados matriciales (*keypad*), visualizadores (*displays*) de 7 segmentos o de cristal líquido, convertidores A/D o D/A, y también una impresora, un motor, etc. Por supuesto, todos estos periféricos deben incluir el circuito de interfaz necesario para conectarlos a los puertos del microcontrolador.

Un *puerto* (*port*) es un circuito que forma parte del microcontrolador y sirve de interfaz con algún dispositivo externo (un periférico). La figura 5.1 muestra el esquema general de la conexión entre un microcontrolador y un periférico a través de un puerto de E/S. En general, en esta conexión se dispone de n líneas para transportar el dato (lo usual es $n = 8$) y de m líneas adicionales para controlar la transferencia de los datos entre el periférico y el puerto. Las líneas de control pueden no ser necesarias, como sucede en la E/S simple sin sincronización, que se tratará más adelante en este mismo apartado. En la figura 5.1 se ha representado un puerto bidireccional, pero es común encontrar puertos que son sólo de entrada y puertos que son sólo de salida.

Desde el punto de vista de su programación, los puertos se identifican por sus direcciones, ubicadas por lo general en la memoria de datos. Para hacer referencia a los datos que entran o salen por un puerto, se necesita al menos una dirección. El manejo de las señales de control puede requerir al-

gunos bits adicionales, repartidos en una o dos direcciones más. En los microcontroladores PIC, el acceso a los puertos se realiza a través de los registros de funciones especiales de la memoria de datos.

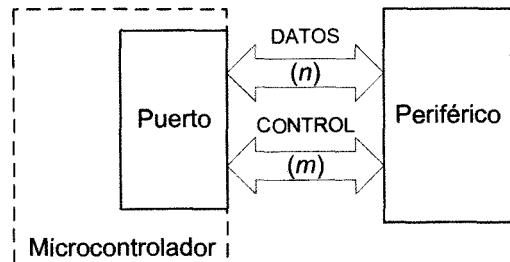


Figura 5.1 Conexión entre el microcontrolador y un periférico a través de un puerto de E/S. Obsérvese que hay n líneas de datos y m líneas de control.

El elemento básico en un puerto de E/S es el biestable tipo D *latch*, capaz de almacenar un bit. En un puerto de 8 bits, hay 8 biestables de este tipo. El biestable dispone de una entrada de datos D, una entrada de control G (*Gate*) y una salida de datos Q. Mientras la señal de control está en '0' ($G = '0'$), el biestable está 'bloquedado'; cuando G pasa al estado '1', el biestable se vuelve transparente y pone en su salida Q el valor de la entrada de datos D ($Q = D$); cuando G regresa a '0', el biestable captura el valor de la entrada D y lo mantiene en su salida Q.

A menudo se necesita disponer del dato a través de una salida tri-estado DO (*Data Output*); entonces se añade la entrada de control OE# (*Output Enable*), activa en '0', para habilitar la salida tri-estado. Si OE# = '1', la salida DO se mantiene en estado de alta impedancia ("tercer estado"), pero si OE# = 0, la salida DO = Q. La figura 5.2 muestra el esquema de este biestable tipo D latch con salida tri-estado.

En un puerto de entrada, las entradas D vienen del periférico y las salidas tri-estado O van a las líneas del bus de datos interno del microcontrolador. En un puerto de salida, la conexión es la contraria.

A continuación se estudian los métodos utilizados para transferir datos entre un puerto paralelo y un periférico, así como las diferentes técnicas de atención del puerto por parte del microcontrolador.

5.1.1 Métodos de transferencia de datos

La transferencia de datos entre un periférico y un puerto se realiza básicamente mediante uno de los dos métodos siguientes:

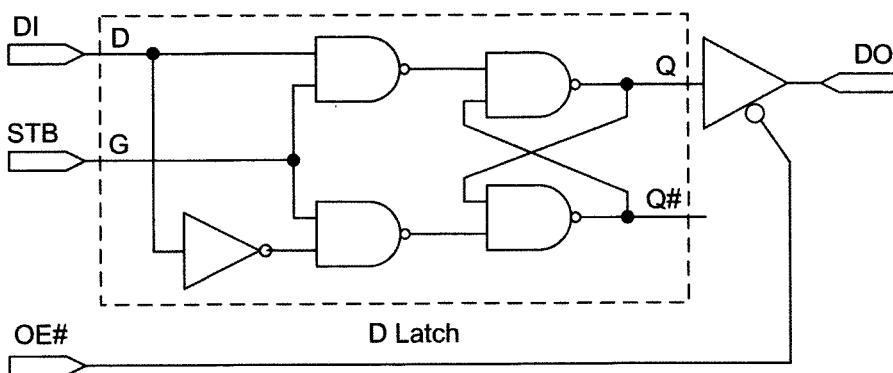


Figura 5.2 El elemento básico constitutivo de un puerto de E/S es el biestable tipo D latch, con el cual se puede almacenar un bit. La figura muestra este elemento junto con una salida tri-estado de datos. DI es la entrada de datos; con la señal STB, el latch captura el dato en DI y la señal OE# habilita la salida tri-estado DO.

- E/S simple.
- E/S controlada.

La E/S simple se caracteriza porque entre el puerto y el periférico solamente se transfieren los bits del dato, sin señales de control (figura 5.3a). Por ejemplo, la conexión de interruptores a las líneas de entrada o diodos LED a las líneas de salida de un puerto paralelo, son dos casos típicos de E/S simple.

A veces el dato va acompañado de una señal de sincronización (STB: *Strobe*). El objetivo básico de esta señal es indicar el momento en que el dato está disponible en los terminales de datos. Esta indicación puede hacerse atendiendo al nivel o a los flancos de la señal STB. Por ejemplo, si la indicación es por nivel, mientras el periférico (o el puerto) mantienen activa la señal STB (es decir, $STB = '1'$), el dato está estable en los terminales de datos del puerto o periférico. El dispositivo que recibe el dato (ya sea el periférico o el puerto), debe capturar el dato en sincronismo con la señal STB. La figura 5.3b muestra esta variante de E/S simple.

Si la indicación de que el dato está estable se realiza con uno de los flancos de la señal STB, por ejemplo, con el flanco de subida, entonces debe capturarse el dato sincronizadamente con esa transición de la señal STB.

En la E/S controlada se establece una “conversación” (*handshake*) entre el puerto y el periférico. Para realizar la E/S controlada se requieren algunas señales de control (al menos dos señales) y algún protocolo o reglas de entendimiento entre uno y otro dispositivo. La figura 5.4 ilustra dos variantes de la E/S controlada que emplean dos señales de control: una generada por el

emisor del dato, denominada STB (*strobe*), y otra generada por el receptor del dato, llamada ACK (*acknowledgment*). En la primera variante (figura 5.4a), denominada E/S controlada simplemente, el emisor de los datos (el periférico o el puerto, según sea la transferencia de entrada o salida) envía la señal STB al receptor, indicando que el dato está disponible en los terminales de datos (*el emisor dice: "aquí te envío el dato"*). Entonces el receptor captura el dato y reporta la acción al transmisor activando la señal ACK (ACK = '1') (*el receptor dice: "dato recibido, lo estoy procesando"*). El emisor no envía un nuevo dato hasta que la señal ACK se haya desactivado (*el receptor dice: "envía el siguiente dato"*). La señal ACK funciona, pues, como una señal informativa de que el receptor está ocupado procesando el dato recibido.

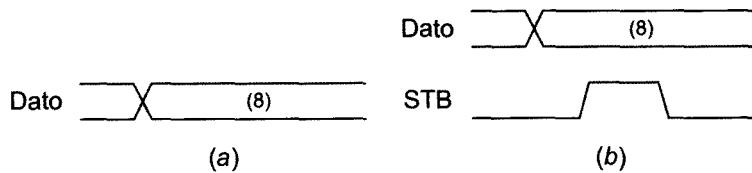


Figura 5.3 Formas de onda para la E/S simple en el caso de una transferencia paralela de 8 bits.
 (a) E/S simple sin sincronización, (b) E/S simple sincronizada con la señal STB.
 La señal STB indica el momento en que el dato está disponible en los terminales de datos. La indicación puede hacerse con el nivel de STB (por ejemplo, con STB = 1, como está indicado en la figura) o con cualquiera de los flancos de esta señal.

La segunda variante de E/S controlada mediante dos señales de control, es la denominada E/S controlada doblemente, e implica un protocolo algo más complejo entre el emisor y el receptor de los datos (figura 5.4b). En primer lugar, el emisor “avisa” al receptor de que va a enviarle un dato (aunque el dato aún no está disponible), activando la señal STB (*el emisor pregunta: “te voy a enviar un dato, ¿puedo hacerlo?”*). Una vez que el receptor detecta esta situación y se halla en condiciones de aceptar el dato, informa de ello al emisor activando la señal ACK (*el receptor responde: “envíalo”*). El emisor detecta que ACK está activa, lo cual es una indicación de que puede enviar el dato al receptor. Una vez que el emisor pone el dato en los terminales de datos, se lo indica al receptor con el flanco de bajada de la señal STB (*el emisor dice: “aquí va el dato”*). Entonces el receptor captura el dato y cuando está listo para recibir otro dato, desactiva la señal ACK (*el receptor dice: ya recibí el dato, puedes enviar un nuevo dato*).

Las señales de control y la lógica de la conversación entre puerto y periférico se pueden manipular por hardware o por software. La manipulación por hardware significa que el puerto dispone de circuitos capaces de gene-

rar y manipular las señales STB y ACK adecuadamente, sin la intervención de la CPU del microcontrolador. La manipulación por software significa en cambio que las señales se generan y manipulan mediante algún programa elaborado expresamente para ello.

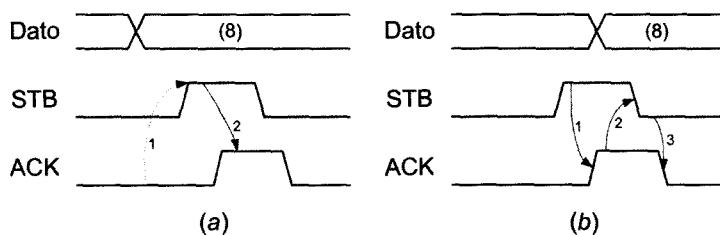


Figura 5.4 Formas de onda para la E/S controlada de datos de 8 bits. (a) E/S controlada simplemente y (b) E/S controlada doblemente.

Los microcontroladores PIC disponen de puertos paralelos de hasta 8 bits, independientes entre sí. No disponen de puertos especializados para implementar la E/S controlada por hardware; por tanto, la E/S controlada debe implementarse por software.

5.1.2 Técnicas de entrada y salida

Para atender a un periférico conectado a un microcontrolador, se emplea normalmente una de las dos técnicas siguientes:

- E/S programada.
- E/S por interrupción.

La E/S programada es una técnica básicamente de software. Supone la existencia de algún indicador del estado (listo o no listo) del periférico; este indicador no es más que un bit cuyo valor se puede averiguar desde el programa que atiende al periférico. El programa pregunta si el periférico requiere atención; si la respuesta es afirmativa, se hace la acción que corresponda, que en general consiste en la lectura o escritura del dato en el puerto conectado al periférico; si la respuesta es negativa, el programa pasa a otra tarea o simplemente espera a que el periférico esté listo para ser atendido.

La figura 5.5 muestra los algoritmos que siguen las dos variantes de la E/S programada: la consulta y la espera. En E/S por consulta (*polling*) (figura 5.5a), el programa pasa a otra tarea si el periférico no está listo. En la E/S por espera, el programa que atiende al periférico, espera a que el periférico esté

listo para atenderle. Es obvio que la E/S por consulta aprovecha mejor el tiempo del microcontrolador.

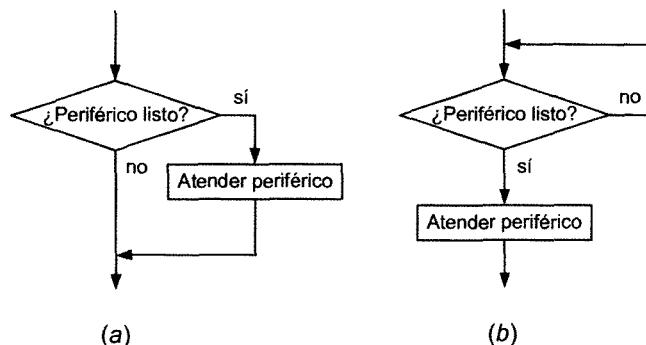


Figura 5.5 Variantes de la E/S programada. En (a) se muestra el algoritmo que sigue la E/S por consulta: si el periférico no está listo para recibir atención, el programa continúa y pasa a realizar otra tarea, que puede ser atender a otro periférico. En (b) se muestra el algoritmo de la E/S por espera. En este caso, el programa simplemente espera hasta que el periférico esté listo para ser atendido.

La técnica de E/S por interrupción se caracteriza porque el periférico "avisa" cuando necesita atención. Ese aviso se realiza mediante una solicitud de interrupción al microcontrolador. Al recibir una solicitud de interrupción generada por algún periférico, el microcontrolador interrumpe transitoriamente la ejecución del programa en curso y pasa a ejecutar el programa o más bien la subrutina de atención al periférico. Terminada la atención al periférico, se continúa con el programa que había sido interrumpido.

La E/S por interrupción se implementa mediante una combinación de software y hardware. La parte de hardware de esta técnica la constituyen los circuitos necesarios para solicitar y tratar la interrupción. La E/S por interrupción se estudiará con detalle en el capítulo 7.

En las técnicas de E/S programada y por interrupción, la velocidad de la transferencia de datos entre un periférico y el microcontrolador está limitada, en última instancia, por la velocidad con que se ejecutan las instrucciones, pues en ambas técnicas los datos se leen o escriben mediante instrucciones de un programa elaborado al efecto. Además, en general, los datos transferidos entre la memoria y los puertos de E/S tienen que pasar por la CPU del microcontrolador y eso, lógicamente, limita también la velocidad del proceso. En los sistemas basados en un microprocesador, se utiliza una tercera técnica de E/S para superar estas limitaciones; es el denominado *acceso directo a memoria* (DMA: *Direct Memory Access*).

El acceso directo a memoria es una técnica de E/S que se implementa fundamentalmente por hardware. Consiste básicamente en que la transferencia de los datos se realiza directamente entre el periférico y la memoria del sistema, sin la intervención de un programa ejecutado por el microprocesador. Se logran así elevadas tasas de transferencia de datos. Esta técnica se usa muy poco en los microcontroladores y los microcontroladores PIC no la emplean.

5.2 Los puertos paralelos en los PIC de clase media

Los microcontroladores PIC de clase media pueden tener hasta 7 puertos paralelos, nombrados con las letras A, B...G. Cada puerto puede ser de hasta 8 bits. Los terminales de los puertos se identifican como RA $<x>$, RB $<x>$,..., RG $<x>$, donde x es el número del bit ($x = 0, 1\dots7$). En general, cada línea de un puerto se puede programar como entrada o como salida.

La mayoría de los terminales de los puertos de E/S pueden realizar varias funciones. Por ejemplo, un mismo terminal puede servir como entrada o salida digital, o puede ser una entrada analógica al convertidor A/D, o puede ser portador de alguna señal de entrada o salida a uno de los temporizadores del microcontrolador.

Algunos microcontroladores PIC también tienen un puerto paralelo adicional denominado Puerto Paralelo Esclavo (PSP: *Parallel Slave Port*). El PSP se comporta como un bus periférico de 8 bits, con líneas para controlar la transferencia de datos entre el PIC y el periférico. Cuando existe, el PSP comparte sus terminales con los puertos D y E.

Para manipular los puertos paralelos, hay dos registros de funciones especiales por cada uno de los puertos, que se denominan PORT y TRIS (PORTA, TRISA, PORTB, TRISB, etc.). Los registros PORT almacenan el dato de salida del puerto mientras que los registros TRIS sirven para programar cada línea del puerto correspondiente como entrada o salida.

Cada bit de cualquier registro TRIS se programa de la siguiente forma:

TRIS $<x>$ = 1 programa la línea $<x>$ del puerto como entrada.

= 0 programa la línea $<x>$ del puerto como salida.

La figura 5.6 muestra el esquema básico de un terminal de E/S. El circuito consta de dos biestables D *latch*: uno para almacenar el bit del dato de salida y otro para almacenar el bit de control TRIS $<x>$. El terminal de E/S se manipula mediante dos transistores MOS en configuración *totem-pole*. En esta configuración, el transistor T1 (canal P) conduce cuando en su puerta hay

un '0' y se corta con un '1'; en el transistor T2 (canal N) ocurre lo contrario: conduce con un '1' y se corta con un '0' en su puerta. El *totem-pole* es excitado desde los biestables a través de dos puertas lógicas, una AND y otra OR. Se puede comprobar que si el biestable de control almacena un '1', ambos transistores están cortados, por lo que el terminal de E/S es puesto en estado de alta impedancia (3er. estado), y por tanto queda configurado como terminal de entrada de datos. Si el biestable de control almacena un '0', entonces el terminal de E/S queda configurado como salida y en este caso aparece en esa salida el valor del bit almacenado en el biestable de datos. La tabla 5.1 muestra la tabla de la verdad del circuito.

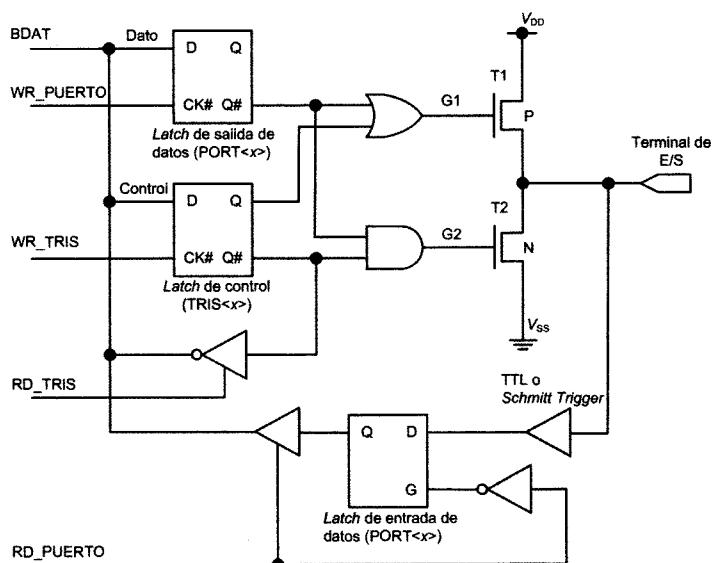


Figura 5.6 Esquema básico de un terminal de E/S típico en los microcontroladores PIC de clase media.

Tabla 5.1 Tabla de la verdad correspondiente al circuito de la figura 5.6.

Control	Dato	G1	G2	T1	T2	Terminal de E/S
1	x	1	0	cortado	cortado	Entrada: alta impedancia
0	0	0	0	cortado	conduce	Salida: V_{ss} (0)
	1	1	1	conduce	cortado	Salida: V_{dd} (1)

El circuito de la figura 5.6 incluye un tercer biestable que almacena el estado del terminal de E/S cuando éste ha sido configurado como entrada. La entrada D de este biestable recibe la tensión existente en el terminal de E/S a través de una puerta no inversora que puede ser TTL o *Schmitt Trigger*, según el puerto del microcontrolador.

Obsérvese que escribir un bit en un puerto es escribir en el *latch* correspondiente, mientras que leer (un bit) en un puerto es leer el estado lógico (nivel de tensión) que tiene físicamente el terminal. Esto significa que si se escribe un dato en un puerto de salida y luego se lee ese puerto, el valor leído puede no coincidir con el valor escrito con anterioridad. Esto puede ocurrir especialmente si se exceden los valores máximos de las corrientes de salida (I_{OH} e I_{OL}) recomendados por el fabricante.

Al conectar dispositivos a los terminales de los puertos, hay que tener en cuenta las limitaciones de potencia eléctrica del microcontrolador. Cada terminal de E/S puede suministrar o drenar una corriente máxima. Además, la corriente suministrada o drenada por el total de terminales de un puerto no puede exceder un cierto valor, que es generalmente menor que la suma de las corrientes individuales permitidas a través de los terminales del puerto.

Por otra parte, si se quiere que las tensiones en los terminales de salida permanezcan dentro de los límites establecidos para los estados lógicos '0' y '1', hay que mantener las corrientes de salida en nivel alto (I_{OH}) y nivel bajo (I_{OL}) dentro de los límites recomendados por el fabricante,

Todos los diseños deben cumplir estos requisitos, que están descritos en las hojas de especificaciones (*data sheet*) que suministra el fabricante para cada dispositivo.

Ejemplo 5.1

A continuación se dan algunos valores de las tensiones y corrientes en los terminales del PIC16F873 suministrados por su fabricante. Cualquier diseño de E/S debe tenerlos en cuenta.

Corriente máxima drenada por cualquier terminal: 25 mA.

Corriente máxima suministrada por cualquier terminal: -25 mA.

La corriente drenada o suministrada por el puerto C o por el conjunto de los puertos A y B no debe sobrepasar los 200 mA.

Valores típicos para las tensiones de entrada y salida:

Se interpreta que hay un estado lógico '0' en un terminal de entrada si la tensión en ese terminal es $V_{IL} < 0,75$ V (entrada TTL) o $V_{IL} < 1,0$ V (entrada Schmitt Trigger), cuando $V_{DD} = 5,0$ V.

Se interpreta que hay un estado lógico '1' en un terminal de entrada si la tensión en ese terminal es $V_{IH} > 2,0$ V (entrada TTL) o $V_{IH} > 4,0$ V (entrada Schmitt Trigger), cuando $V_{DD} = 5,0$ V.

La tensión de salida para el estado lógico '0' es $V_{OL} < 0,6$ V si la corriente de salida drenada por el terminal es $I_{OL} < 8,5$ mA (cuando $V_{DD} = 4,5$ V).

La tensión de salida para el estado lógico '1' es $V_{OH} > V_{DD} - 0,7$ V = 3,8 V si la corriente de salida suministrada por el terminal es $I_{OH} < -3,0$ mA (cuando $V_{DD} = 4,5$ V).

La modificación selectiva o individual de los bits de un puerto exige ciertas precauciones. En realidad, los microcontroladores PIC no disponen de hardware en sus puertos para modificar exclusiva y selectivamente un bit de salida, sino que, para modificar un bit, siempre se escriben la totalidad de los bits del puerto. Incluso las instrucciones de manipulación de bits, como bcf y bsf, que permiten poner a '0' y poner a '1' respectivamente un bit cualquiera de un registro, lo que hacen realmente es leer el registro, modificar el bit que se especifica en la instrucción y escribir la palabra resultante en el registro. Es decir, la modificación selectiva de un bit de un registro se realiza realmente como una operación de lectura —modificación— y escritura del registro completo. Esta forma de operar resulta transparente al programador cuando se trata de modificar un bit perteneciente a un registro de la memoria de datos. Si el bit modificado es del registro PORT de un puerto paralelo, entonces esa forma en que operan las instrucciones de manipulación individual de bits puede producir resultados inesperados en los otros bits del puerto. El problema está en que, cuando se lee un puerto, en realidad se lee el estado de sus terminales, no el valor almacenado en el registro PORT; por lo tanto, la lectura no tiene por qué coincidir necesariamente con el valor existente en el registro PORT.

5.2.1 El puerto A

El puerto A puede tener hasta 8 bits, aunque en la mayoría de los PIC de clase media (como el PIC16F873), sólo están implementados 6 bits, que corresponden a los terminales RA0 a RA5. Todos los terminales se pueden configurar como entradas o como salidas. RA4 tiene la particularidad de que es una entrada *Schmitt Trigger* y, si se programa como salida, es de drenador abierto. Los registros de funciones especiales asociados al puerto A son PORTA y TRISA.

Los terminales del puerto A pueden estar compartidos con las entradas del convertidor A/D si el microcontrolador dispone de este dispositivo, como sucede en el PIC16F873. En estos casos los terminales del puerto A pueden ser digitales o analógicos, y esto se programa mediante el registro de funciones especiales ADCON1. El terminal RA4 se utiliza también como entrada de reloj externo del temporizador Timer0. El terminal se denomina entonces RA4/T0CKI. La tabla 5.2 muestra las funciones de los terminales del puerto A en un PIC16F873.

Ejemplo 5.2

Los terminales del puerto A son también entradas analógicas en aquellos PIC que disponen de convertidor A/D. Este ejemplo muestra cómo se programa el puerto A tanto en los PIC que no tienen convertidor A/D como en aquellos que sí lo tienen.

Iniciación del puerto A (si el PIC no tiene convertidor A/D):

clrf STATUS	; Se selecciona el banco 0.
clrf PORTA	; Se pone 0 en el registro PORTA.
bsf STATUS, RP0	; Se selecciona el banco 1.
movlw 0xCF	; Valor que hay que cargar en TRISA para programar
movwf TRISA	; RA<3:0> como entradas y RA<5:4> como salidas.
bcf STATUS, RP0	; Se selecciona el banco 0.

Iniciación del puerto A (en un PIC con convertidor A/D):

bcf STATUS, RP0	; Se selecciona el
bcf STATUS, RP1	; banco 0.
clrf PORTA	; Se pone 0 en el registro PORTA.
bsf STATUS, RP0	; Se selecciona el banco 1.
movlw 0x06	; Se configuran todos los terminales del
movwf ADCON1	; puerto como entradas o salidas digitales.
movlw 0xCF	; Valor que hay que cargar en TRISA para programar
movwf TRISA	; RA<3:0> como entradas y RA<5:4> como salidas.
bcf STATUS, RP0	; Se selecciona el banco 0.

Tabla 5.2 Funciones de los terminales del puerto A en un PIC16F873.

Nombre	Función
RA0/AN0	Entrada/salida digital o entrada analógica.
RA1/AN1	Entrada/salida digital o entrada analógica.
RA2/AN2	Entrada/salida digital o entrada analógica.
RA3/AN3/VREF	Entrada/salida digital, o entrada analógica, o tensión de referencia del convertidor A/D.
RA4/T0CKI	Entrada/salida digital o entrada externa de reloj para el Timer0. Como salida, es de drenador abierto.
RA5/SS/AN4	Entrada/salida digital o entrada de selección del puerto serie sincrónico o entrada analógica.

5.2.2 El puerto B

El puerto B tiene 8 bits y sus terminales se denominan RB0 a RB7. Para escribir un dato en el puerto B se usa el registro de funciones especiales PORTB. Todos los terminales se pueden configurar como entradas o como salidas mediante el registro de funciones especiales TRISB. La tabla 5.3 muestra las funciones de los terminales del puerto B en un PIC16F873.

Cada terminal del puerto B cuenta con un circuito de *pull-up* interno, que se puede programar mediante el bit RBPU# del registro de funciones especiales OPTION (bit OPTION<7>). Con este bit se activan o desactivan los *pull-up* del puerto B.

Tabla 5.3 Funciones de los terminales del puerto B en un PIC16F873. Todos los terminales tienen un pull-up interno programable por software. RB0 puede utilizarse como entrada de interrupción externa. Un cambio de nivel en las entradas RB4 a RB7 puede generar una solicitud de interrupción. Los terminales RB3, RB6 y RB7 se usan en la programación del microcontrolador "en el circuito": se trata de un recurso disponible en los microcontroladores PIC para programar el microcontrolador en la propia placa de circuito impreso de la aplicación a la que está destinado.

Nombre	Función
RB0/INT	Entrada/salida digital. Entrada de interrupción externa.
RB1	Entrada/salida digital.
RB2	Entrada/salida digital.
RB3/PGM	Entrada/salida digital. Terminal de programación en circuito.
RB4	Entrada/salida. En entrada se programa una interrupción por cambio de nivel lógico.
RB5	Entrada/salida. En entrada se programa una interrupción por cambio de nivel lógico.
RB6/PGC	Entrada/salida. En entrada se programa una interrupción por cambio de nivel lógico. Terminal de programación en circuito.
RB7/PGD	Entrada/salida. En entrada se programa una interrupción por cambio de nivel lógico. Terminal de programación en circuito.

Un recurso que posee el puerto B es que puede generar una solicitud de interrupción por cambio en el nivel lógico de la señal en cualquiera de los terminales RB4 a RB7. Si estos terminales se programan como entradas, un cambio en el nivel lógico de la señal de entrada, de '0' a '1' o viceversa, genera una interrupción. Este cambio puede ser producido, por ejemplo, por la pulsación de una tecla conectada a uno de los terminales del puerto. Al producirse la interrupción por cambios en RB4 a RB7, el bit RBIF del registro INTCON (bit INTCON<0>) se pone a '1'. La interrupción puede habilitarse o inhabilitarse mediante el bit RBIE del registro INTCON (bit INTCON<3>). Esta interrupción puede servir para sacar al microcontrolador del modo de bajo consumo (modo *sleep*).

El terminal RB0 también sirve para aceptar una solicitud de interrupción externa, por flancos. En este caso el terminal se denomina RB0/INT. Esta interrupción queda reportada mediante el bit INTF del registro INTCON (bit INTCON<1>) y se habilita o inhabilita mediante el bit INTE del registro INTCON (bit INTCON<4>). El flanco que produce la interrupción puede ser el de subida o el de bajada, lo cual se programa mediante el bit INTEDG del registro OPTION (bit OPTION<6>).

Otra característica del puerto B es que los terminales RB3, RB6 y RB7 pueden ser utilizados como terminales para la programación en circuito del microcontrolador. La programación en circuito (ICSP: *In-Circuit Serial Programming*) es un recurso disponible en los microcontroladores PIC que per-

mite que el microcontrolador sea programado en la propia placa de circuito impreso de la aplicación a la que está destinado. El programa se introduce en la memoria OTP, EEPROM o FLASH del microcontrolador básicamente a través de los terminales del puerto B antes mencionados, utilizando un formato de transmisión serie. Se recomienda consultar las especificaciones de programación de cada dispositivo para utilizar este recurso.

5.2.3 El puerto C

El puerto C es un puerto paralelo de 8 bits cuyos terminales se nombran RC0 a RC7. Para escribir un dato en el puerto C se usa el registro de funciones especiales PORTC. Todos los terminales se pueden configurar como entradas de tipo *Schmitt Trigger* o como salidas digitales, mediante el registro de funciones especiales TRISC.

Los terminales del puerto C comparten funciones con otros dispositivos de entrada y salida: el temporizador Timer1, el módulo CCP (*Compare/Capture/PWM*) y los puertos serie SSP (*Synchronous Serial Port*) o MSSP (*Master Synchronous Serial Port*) y USART (*Universal Synchronous Asynchronous Transmitter Receiver*). Esta multiplicidad de funciones se ilustra en la tabla 5.4 para el PIC16F873.

Tabla 5.4 Funciones de los terminales del Puerto C en un PIC16F873. Todas las entradas son de tipo Schmitt Trigger.

Nombre	Función
RC0/T1OSO/T1CKI	Entrada/salida digital o salida del oscilador del Temporizador 1 o entrada de reloj del Temporizador 1.
RC1/T1OSI/CCP2	Entrada/salida digital o entrada del oscilador del Temporizador 1 o entrada de reloj del Temporizador 1 o terminal del módulo CCP2.
RC2/CCP1	Entrada/salida digital o terminal del módulo CCP1.
RC3/SCK/SCL	Entrada/salida digital o terminal del puerto serie sincrónico.
RC4/SDI/SDA	Entrada/salida digital o terminal del puerto serie sincrónico.
RC5/SDO	Entrada/salida digital o terminal del puerto serie sincrónico.
RC6/TX/CK	Entrada/salida digital o terminal del puerto serie USART.
RC7/RX/DT	Entrada/salida digital o terminal del puerto serie USART.

5.2.4 Los puertos D, E, F y G

Los puertos D y E son puertos paralelos de hasta 8 bits. Todos los terminales se pueden configurar como entradas o como salidas digitales. Si se configuran como entradas, entonces son de tipo *Schmitt Trigger*. En los microcontroladores donde existen, los terminales de estos puertos comparten sus funciones con el Puerto Paralelo Esclavo (PSP). A través de los terminales del puerto E también se pueden tener algunas entradas analógicas adicionales a

las disponibles en el puerto A. Los registros de funciones especiales asociados a los puertos D y E son PORTD y PORTE para datos y TRISD y TRISE para control. Los puertos D y E están implementados en el PIC16F874 pero no en el PIC16F873.

Los puertos F y G son puertos paralelos de hasta 8 bits de entrada de tipo *Schmitt Trigger*. Comparten funciones con las salidas de excitadores (*drivers*) de pantallas de cristal líquido (LCD: *Liquid Crystal Display*). Los puertos paralelos F y G sólo existen en los microcontroladores fabricados para excitar directamente dispositivos de cristal líquido, como el PIC16F946.

5.2.5 El Puerto Paralelo Esclavo

El Puerto Paralelo Esclavo (PSP: *Parallel Slave Port*) es un bus bidireccional de 8 bits con señales de control para leer y escribir datos desde un dispositivo externo al microcontrolador. El PSP se puede utilizar para conectar el microcontrolador directamente al bus de datos y de control de un sistema basado en un microprocesador u otro microcontrolador, convirtiendo al PIC poseedor del PSP en un puerto de E/S de ese sistema, tal como muestra la figura 5.7a.

En los microcontroladores que tienen PSP, como el PIC16F874, el PSP se realiza como una función alternativa de los terminales de los puertos D y E. El PSP cuenta con 8 líneas de datos (PSP<0:7>) y 3 líneas de control para la lectura (RD#) y escritura (WR#) de datos, y la selección (CS#) del PIC convertido en dispositivo periférico. Las líneas de datos del PSP se realizan sobre los terminales RD<0:7> y las de control utilizan los terminales RE<0:2>.

El PSP funciona de la siguiente manera. Si el sistema exterior al PIC va a leer o escribir un dato en el PIC usando el PSP, debe mantener seleccionado el dispositivo con CS# = '0' durante la lectura o escritura del dato. Si CS# = '1', las líneas de datos permanecen en estado de alta impedancia. Durante la lectura, los terminales de datos se comportan como salidas; mientras que durante la escritura de un dato, se comportan como entradas. La figura 5.7b ilustra el comportamiento de las señales del PSP durante ciclos de lectura y escritura de datos.

En el microcontrolador hay tres bits que informan del estado del PSP: los bits IBF y OBF del registro TRISE, y el bit PSPIF del registro PIR1. El **bit IBF (Input Buffer Full)** se pone a '1' cuando el registro PORTD contiene **un** dato escrito en el PSP desde el exterior; IBF se pone a '0' automáticamente cuando el programa lee el dato de entrada en PORTD.

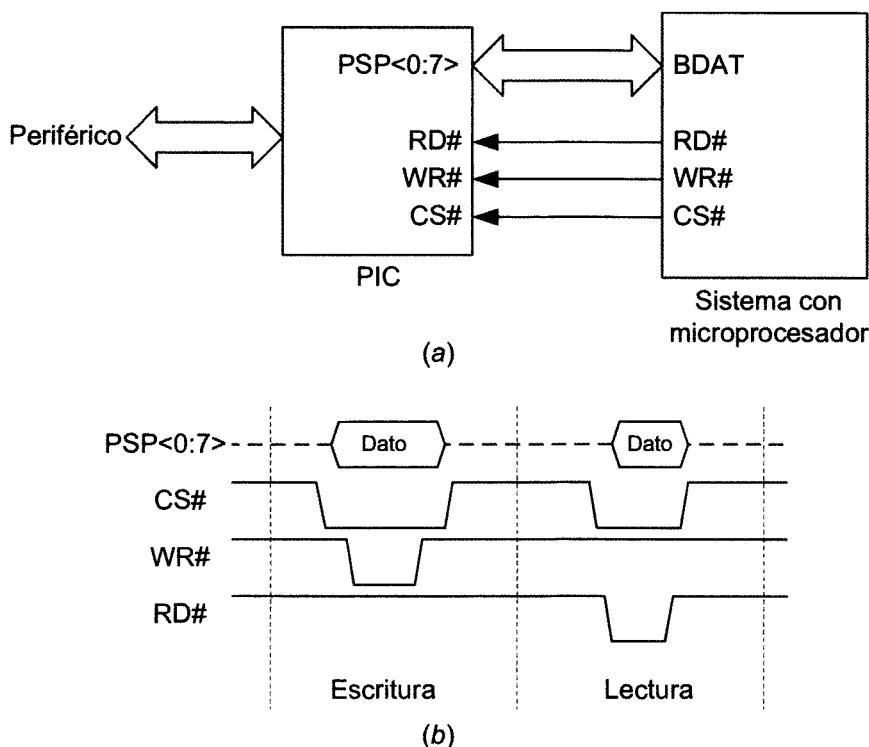


Figura 5.7 El Puerto Paralelo Esclavo (PSP) es un bus bidireccional de 8 bits con señales de control para la transferencia paralela de datos entre el PIC y un dispositivo externo al microcontrolador. En (a) se muestra un posible uso del PSP para conectar el PIC directamente al bus de datos (BDAT) de un sistema con un microprocesador o microcontrolador. En este caso, el PIC se convierte en un puerto de E/S de ese sistema, a través del cual se conecta el periférico al sistema con microprocesador. En (b) se muestran diagramas de tiempo con las señales que participan de las transferencias de E/S a través del PSP entre el PIC y el sistema.

El bit OBF (Output Buffer Full) se pone a '1' cuando el programa escribe un dato en el registro PORTD, el cual debe salir por los terminales del PSP; OBF se pone a '0' cuando el registro PORTD es liberado del dato de salida debido a que el dispositivo externo realizó una lectura del PSP.

El bit PSPIF (PSP Interrupt Flag) se pone a '1' cada vez que un dato es leído o escrito desde el exterior; debe ser puesto a '0' por software una vez que el programa atienda la transferencia por el PSP. PSPIF = '1' genera una solicitud de interrupción si la interrupción del PSP está habilitada, lo cual ocurre si el bit PSPIE (PSP Interrupt Enable) del registro PIE1 es '1'.

La atención en el PIC de la entrada y salida de datos por el PSP puede ser programada o por interrupción. La E/S programada se realiza mediante la consulta de los bits PSPIF, IBF y OBF. En la E/S por interrupción, el pro-

grama que atiende la interrupción del PSP debe consultar los bits IBF y OBF para conocer el tipo de transferencia que tuvo lugar entre el PSP y el exterior, y así actuar en consecuencia.

5.3 Conexión y tratamiento a periféricos comunes

5.3.1 Interruptores y diodos LED

Entre los dispositivos de E/S que se conectan a menudo a un microcontrolador están los interruptores y los diodos LED. La figura 5.8 muestra tres circuitos para conectar estos dispositivos a las líneas del puerto B de un microcontrolador PIC.

El diodo LED1 se activa con un nivel de tensión bajo ('0' lógico) en el terminal RB_i, configurado previamente como una salida digital. En este caso, la corriente I_1 que circula por el diodo LED1 "entra" en el terminal del puerto. Si se quiere que en ese terminal se mantenga el '0' lógico cuando se activa el LED, debe cumplirse

$$I_1 \leq I_{OLmax} \quad (5.1)$$

donde I_{OLmax} es la corriente máxima de salida del terminal cuando está en el nivel bajo.

El diodo LED2 se activa con un nivel alto ('1' lógico) en el terminal de salida RB_j. La corriente I_2 que circula por el diodo LED2 "sale" del terminal. Para que en estas condiciones el terminal mantenga el nivel lógico '1' debe cumplirse

$$I_2 \leq I_{OHmax} \quad (5.2)$$

donde I_{OHmax} es el valor máximo de la corriente de salida del terminal en nivel alto.

En general $|I_{OLmax}| > |I_{OHmax}|$, de modo que es más fácil cumplir la condición (5.1) que la (5.2). Por esta razón se prefiere conectar los diodos LED a los terminales de los puertos paralelos tal como está conectado el diodo LED1.

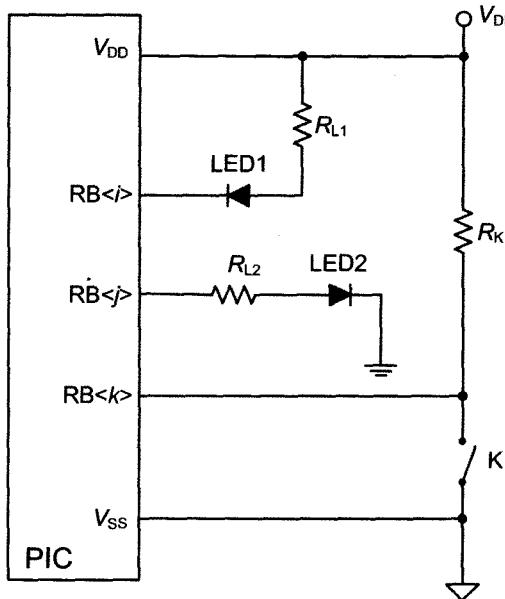


Figura 5.8 Posibles esquemas de conexión de interruptores y diodos LED al puerto B de un microcontrolador PIC.

Ejemplo 5.3

Se tiene un PIC16F873 al que se quieren conectar dos indicadores LED y un interruptor, en un esquema como el mostrado en la figura 5.8. Calcular los valores de las resistencias R_{L1} , R_{L2} si la tensión de alimentación es $V_{DD} = 5\text{ V}$.

Los diodos LED utilizados comúnmente como indicadores (3-5 mm de diámetro, colores rojo, verde, amarillo) operan con una corriente de conducción $I_F = 10\text{ mA}$ y una tensión de conducción $V_F = 2,0\text{ V}$ aproximadamente (depende del color).

Si $I_F = 10\text{ mA}$ y $V_{RB<j>} = V_{OL}$ ($V_{OL} = 0,35\text{ V}$ para $I_{OL} = 10\text{ mA}$, según el fabricante del PIC), el valor de R_{L1} cuando el LED es rojo ($V_F = 1,6\text{ V}$), es:

$$R_{L1} = \frac{V_{DD} - V_F - V_{OL}}{I_F} = \frac{5 - 1,6 - 0,35}{0,01} = 305\Omega$$

Si $I_F = 10\text{ mA}$ y $V_{RB<j>} = V_{DH}$ ($V_{DH} = 4,3\text{ V}$ para $I_{OH} = 10\text{ mA}$, según el fabricante del PIC), el valor de R_{L2} es:

$$R_{L2} = \frac{V_{DH} - V_F}{I_F} = \frac{4,3 - 2,0}{0,01} = 230\Omega$$

El valor de la resistencia en serie con el LED no es crítico y por tanto se pueden utilizar valores estandarizados con tolerancia del 5 % o el 10 %. En este ejemplo se toman los valores $R_{L1} = 300\Omega$ y $R_{L2} = 220\Omega$, con tolerancia del 5 %.

Los interruptores se conectan a entradas digitales. En la figura 5.8, el interruptor K está conectado a la entrada RB_k. La conexión es tal que cuando el interruptor está “cerrado” pone un nivel de tensión bajo (‘0’ lógico) en el terminal al que está conectado. En la conexión representada en la figura 5.8, cuando el interruptor K está cerrado, hay 0 V en el terminal RB_k. Si el interruptor está “abierto”, el circuito debe poner un nivel de tensión alto (‘1’ lógico) en el terminal correspondiente. Este nivel alto se garantiza con una resistencia conectada a la tensión V_{DD} del microcontrolador (*pull-up* externo). El valor de esta resistencia de *pull-up* externa (R_K en la figura 5.8) puede ser razonablemente alto (varias decenas de kiloohms), pues la corriente consumida por una entrada digital es sumamente pequeña.

Los terminales del puerto B de los microcontroladores PIC poseen un resistor de *pull-up* interno, el cual se puede conectar o desconectar por programa, mediante el bit RBPU del registro OPTION. Entonces, si los interruptores se conectan a los terminales del puerto B y se ha programado la conexión de los resistores de *pull-up* internos, se puede prescindir de las resistencias R_K externas.

Los interruptores mecánicos, que en esencia son dos piezas metálicas que se tocan o se separan, tienen el problema del rebote. Cuando se cierra o se abre un interruptor, las piezas metálicas no permanecen enseguida en su posición final sino que, de modo análogo a una bola que cae sobre una superficie dura, rebotan durante un cierto tiempo, con los consiguientes cambios rápidos de la resistencia de contacto, que corresponden a aperturas y cierres rápidos antes de que el interruptor alcance su estado estable (figura 5.9). El rebote puede hacer que un único cierre (o apertura) del interruptor, sea interpretado erróneamente por el microcontrolador como cierres y aperturas sucesivos.

El problema del rebote se puede solucionar por hardware y por software. Una primera solución es, obviamente, cambiar el interruptor por otro que no sea mecánico (de efecto Hall), o por interruptores de láminas con contactos de mercurio, que no tienen rebotes. No obstante, la variedad de modelos y su versatilidad es muy limitada respecto a los interruptores mecánicos. Otra solución hardware es conectar la salida del interruptor a un circuito monostable que alargue el primer pulso detectado durante un tiempo suficiente para ocultar los otros pulsos. Sin embargo, los circuitos añadidos aumentan el coste y el espacio ocupado.

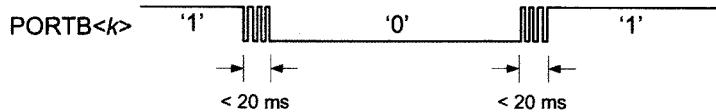


Figura 5.9 El problema del rebote en los interruptores mecánicos. Cuando se cierra o se abre el interruptor, las piezas metálicas vibran y se aprecien aperturas y cierres rápidos antes de alcanzar la posición estable. En general, ese estado transitorio dura menos de 20 ms.

La solución del rebote por software consiste básicamente en leer el estado del interruptor después de transcurrido un tiempo prudencial a partir del momento en que se manipuló el interruptor, tras haber detectado la primera transición. Un tiempo de espera de unos 20 ms es generalmente suficiente. El ejemplo 5.4 ilustra cómo aplicar esta solución.

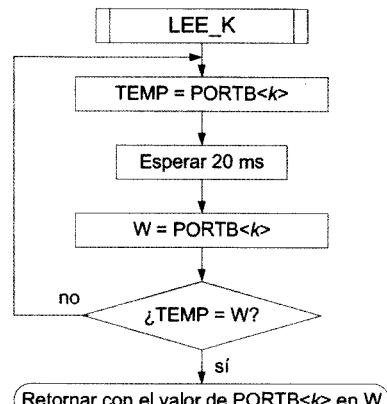
Ejemplo 5.4

En este ejemplo se muestra cómo leer con seguridad el estado de un interruptor conectado al bit k del puerto B según el esquema de la figura 5.8. El problema del rebote se soluciona por software, mediante el algoritmo presentado en la figura 5.10.

En este algoritmo, el interruptor se lee una primera vez y su estado se memoriza en un registro del microcontrolador (TEMP); se espera 20 ms para que se extinga el rebote y se repite la lectura. Si las dos lecturas coinciden, significa que son válidas y se da por concluida la lectura segura del interruptor. Si las lecturas difieren, significa que el rebote no ha concluido y hay que continuar leyendo el interruptor a intervalos (aquí de 20 ms) hasta que las lecturas sean iguales.

Figura 5.10 Algoritmo para leer el estado de un interruptor, solucionando el problema del rebote. El interruptor está conectado al bit k del puerto B, según el esquema de la figura 5.8. TEMP es un registro de la memoria de datos del microcontrolador y W es el registro de trabajo.

A continuación se da el listado de la subrutina LEE_K, que implementa el algoritmo de la figura 5.10.



```
list p=16f873
```

```
#include <p16f873.inc>
```

```
TEMP equ 0x20 ; Registro usado temporalmente por las subrutinas
k equ 3 ; Número del bit del puerto B al que está conectado
; el interruptor.
```

;LEE_K: Rutina para leer el estado de un interruptor conectado al bit k del puerto B.

; sin rebotes.

; Entradas: ninguna.

; Salidas: En W<0>, el valor del bit PORTB<k>

LEE_K:

```
    btfss  PORTB, k      ; Leer el puerto B. ¿El bit PORTB<k> = 1?  
    goto   K0              ; No: ir a poner 0 en TEMP.  
    movlw  1                ; Sí: poner 1 en TEMP.  
    movwf  TEMP              ; TEMP en 1.  
    goto   K1  
K0:  
    clrf   TEMP              ; TEMP en 0.  
K1:  
    call   DEM20            ; Aquí el valor de PORT<k> está en TEMP.  
    ;  
    call   DEM20            ; Esperar 20 ms.  
    ;  
    btfss  PORTB, k      ; Leer de nuevo el puerto B. ¿PORTB<k> = 1?  
    goto   K2              ; No: ir a poner 0 en W.  
    movlw  1                ; Sí: poner 1 en W.  
    goto   K3  
K2:  
    clrw                   ; W en 0.  
K3:  
    xorwf  TEMP, W          ; Aquí el valor de PORT<k> está en W.  
    ; Se comparan los valores leídos. Si son iguales, en W  
    ; se pone un 0 y se activa Z. TEMP no se altera.  
    btfss  STATUS, Z        ; ¿TEMP = W? ¿Z = 1?  
    goto   LEE_K            ; No: No ha terminado el rebote, leer PORTB<k> de nuevo.  
    ; Sí: Fin del rebote, terminar la rutina.  
K4:  
    movf   TEMP, W          ; Poner el valor de PORTB<k> en W y  
    return                 ; retornar.
```

;DEM20: Rutina que demora 20 ms.

DEM20

```
    ;  
    ; Aquí va el código de esta subrutina  
    ;  
    return
```

end

5.3.2 Teclados matriciales

Un teclado matricial está compuesto por teclas interconectadas formando una matriz. Las teclas son simples interruptores mecánicos y cada una ocupa la intersección de una fila con una columna. Cuando se pulsa una tecla, se ponen en contacto eléctrico la fila y la columna donde está dicha tecla. Las filas y columnas de esta matriz se pueden conectar a los terminales de uno o más puertos paralelos. La figura 5.11 muestra un teclado matricial de 16 teclas, dispuestas en 4 filas y 4 columnas.

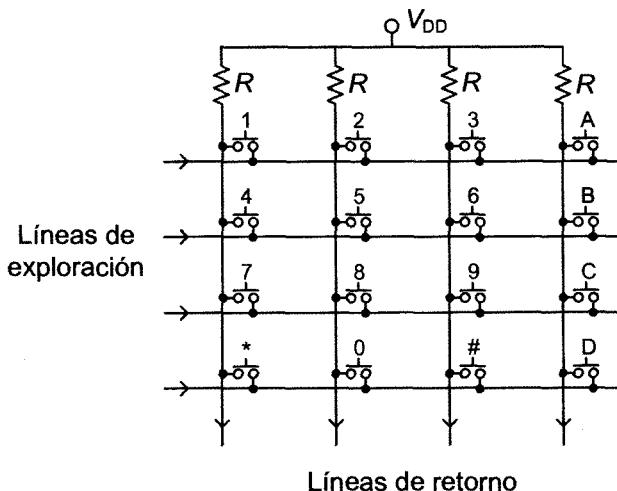


Figura 5.11 Teclado matricial de 16 teclas.

Para explorar un teclado matricial se envían señales hacia las filas de la matriz por las líneas de exploración y se recoge información por las columnas, que entonces constituyen las líneas de retorno. Básicamente se parte de que si no hay ninguna tecla pulsada, todas las líneas de retorno están en el nivel lógico '1'. Las líneas de exploración son puestas (sucesiva o simultáneamente) a '0'. Este valor lógico sólo aparece en la línea de retorno donde está la tecla pulsada, mientras que las restantes líneas de retorno mantienen el valor '1'. Con la información enviada hacia la matriz y la que retorna, se conforma un código único para cada tecla, llamado código de exploración.

Para garantizar que las líneas de retorno permanezcan en '1' si no hay tecla pulsada, se conectan resistencias entre cada línea de retorno y la tensión de alimentación (V_{DD}), según muestra la figura 5.11.

El mecanismo empleado para atender a los teclados matriciales tiene los pasos siguientes:

Paso 1. Esperar la liberación del teclado (debido al pulsado de la tecla anterior a la actual).

Paso 2. Detectar que hay una nueva tecla pulsada.

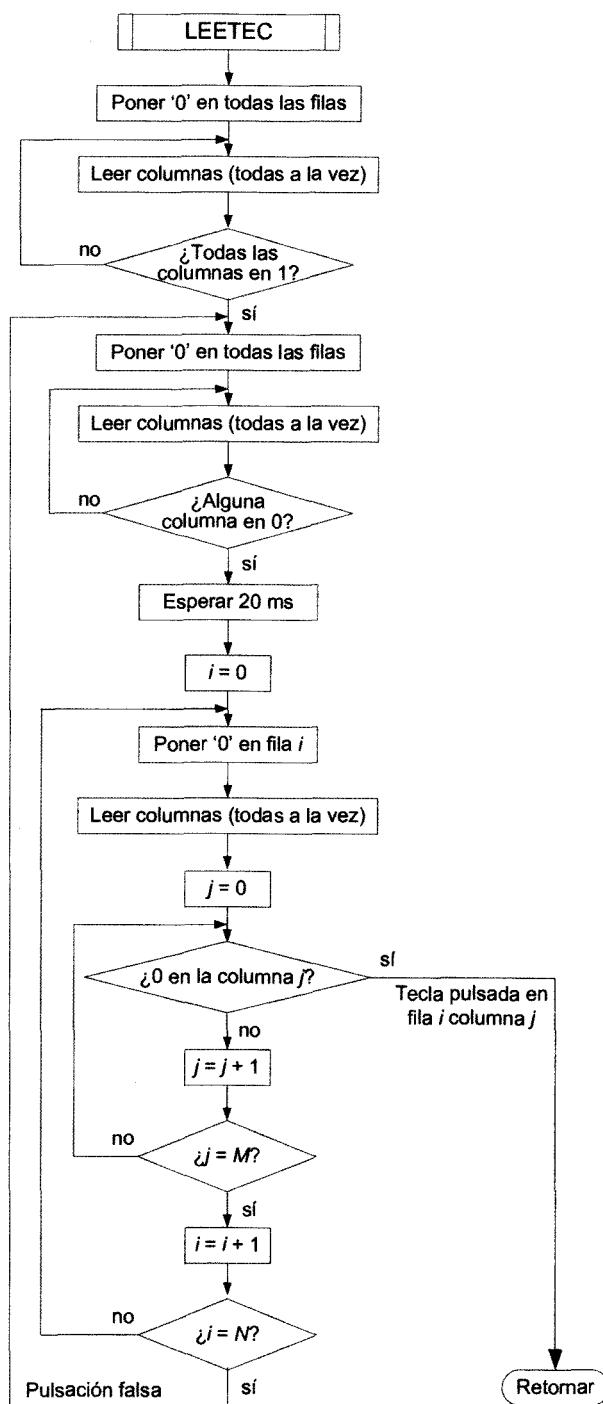
Paso 3. Si se detecta que se ha pulsado una tecla, se debe esperar un tiempo prudencial para que termine el rebote (acción denominada *debounce*). Una espera de unos 20 ms suele ser suficiente.

Paso 4. Se explora la matriz del teclado para determinar cuál es la tecla que ha sido pulsada. En este paso se genera el código de exploración que identifica la tecla pulsada, que contiene básicamente los números de la fila y la columna que ocupa la tecla en la matriz. La exploración del teclado se puede hacer mediante dos métodos diferentes:

- Por exploración secuencial de las filas. En este método se envía un '0' hacia la primera fila y se leen todas las columnas. Si la lectura no contiene ningún '0' (todas las columnas están en '1'), la tecla pulsada no está en esa fila. Entonces se envía el '0' a la siguiente fila y se leen todas las columnas. Se repite el proceso con las siguientes filas hasta encontrar algún '0' en la lectura de las columnas. Se conoce entonces en qué fila y columna está la tecla pulsada y con esto se puede conformar un código de exploración para la tecla.
- Por exploración simultánea de filas y columnas. En este método se envía un '0' a todas las filas simultáneamente y se leen todas las columnas, con lo que se detecta en qué columna está la tecla pulsada. A continuación se invierte el proceso: se envía simultáneamente un '0' a todas las columnas y se leen todas las filas, detectándose así la fila en que está la tecla pulsada. Conociendo en qué fila y columna está la tecla pulsada, se puede conformar un código de exploración para la tecla.

El método de exploración simultánea de filas y columnas tiene la ventaja de que permite la exploración rápida del teclado en sólo dos pasos, pero requiere que las líneas de exploración y retorno sean bidireccionales (aunque esto no es problema en los microcontroladores PIC). El método de exploración secuencial es en general más lento, pues el tiempo que demora detectar la pulsación de una tecla depende de su posición en la matriz, pero en cambio las líneas de exploración y retorno pueden ser unidireccionales.

Figura 5.12 Diagrama de bloques del algoritmo para leer un teclado matricial de N filas y M columnas, usando el método de exploración secuencial. El algoritmo espera hasta que se pulse una tecla y devuelve su posición (fila i , columna j).



La figura 5.12 muestra el algoritmo de atención a teclados matriciales mediante exploración secuencial de las filas del teclado. Este algoritmo se puede mejorar añadiéndole algún paso para comprobar la validez del código de exploración obtenido y determinar si se ha pulsado más de una tecla a la vez. También se puede convertir el código de exploración en algún código estándar, como por ejemplo el código ASCII, si se trata de un teclado alfanumérico.

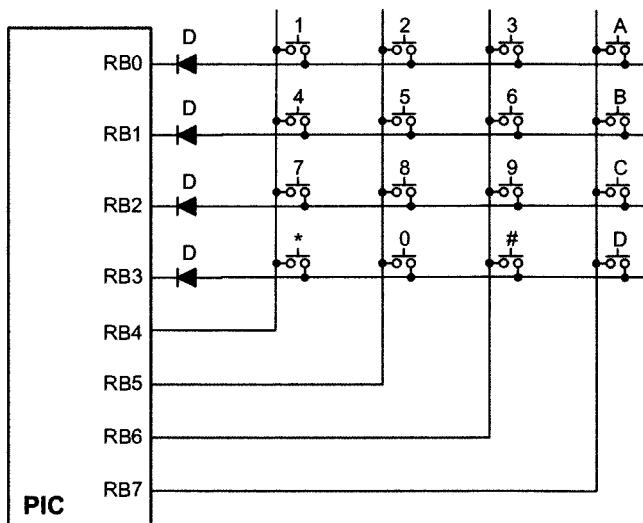


Figura 5.13 Teclado matricial de 16 teclas conectado al puerto B de un PIC. Los diodos evitan el cortocircuito accidental que se produce entre líneas de exploración si se pulsan simultáneamente dos o más teclas situadas en la misma columna. Se aprovecha el pull-up interno (no representado) disponible en el puerto B para mantener un nivel lógico '1' en las líneas de retorno cuando no hay tecla pulsada.

Ejemplo 5.5

Atención a un teclado matricial de 16 teclas conectado al puerto B de un microcontrolador PIC de clase media.

La figura 5.13 muestra una posible conexión del teclado matricial al puerto B del microcontrolador. Al comparar el esquema de esta figura con el de la figura 5.11, se observan dos diferencias fundamentales. En primer lugar, no se han utilizado las resistencias de *pull-up* en las líneas de retorno, porque las líneas del puerto B tienen un *pull-up* activo interno que hace la función de esas resistencias, garantizando que en las líneas de retorno haya un '1' si no hay ninguna tecla pulsada. La segunda diferencia que se observa es la presencia de los diodos D en las líneas de exploración, que se han colocado para evitar la corriente excesiva que se produce en los terminales de salida del puerto B (RB0 a RB3) si se pulsan simultáneamente dos teclas situadas en la misma columna durante el proceso de exploración. Estos diodos pueden sustituirse por resistencias de entre 1 kΩ y 2,2 kΩ.

La atención al teclado se realiza mediante la subrutina LEETEC, que explora el teclado matricial siguiendo el algoritmo representado en la figura 5.12. La rutina devuelve el código de exploración de la tecla pulsada. La tabla 5.5 muestra los códigos de exploración devueltos.

Tabla 5.5 Códigos de exploración devueltos por la rutina LEETEC al explorar el teclado de la figura 5.13.

Tecla	Fila (en binario)	Columna (en binario)	Código de exploración (en hexadecimal)
1	00	00	00
2	00	01	01

3	00	10	02
4	01	00	04
5	01	01	05
6	01	10	06
7	10	00	08
8	10	01	09
9	10	10	0A
0	11	01	0D
*	11	00	0C
#	11	10	0E
A	00	11	03
B	01	11	07
C	10	11	0B
D	11	11	0F

A continuación se presentan los listados de las subrutinas INICIO y LEETEC. La subrutina INICIO se usa para preparar el puerto B para la posterior atención al teclado.

```
list p=16f873
#include <p16f873.inc>
```

;Declaraciones:

TEMP	equ	0x20	; Registro usado temporalmente por las subrutinas.
FILA	equ	0x21	; Registro usado temporalmente por las subrutinas.
COLUMNA	equ	0x22	; Registro usado temporalmente por las subrutinas.

;INICIO: Rutina para programar el puerto B.

INICIO:

```
    clrf STATUS          ; Se selecciona el banco 0.
    bcf INTCON, INT0   ; Inhabilitar interrupción externa por RB0.
    bcf INTCON, RBIE   ; Inhabilitar interrupción por cambios en RB<7:4>.
    movlw 0FFh          ; Valor a poner en PORTB.
    movwf PORTB         ; Todas las salidas del puerto B en '1'.
    bsf STATUS, RP0     ; Se selecciona el banco 1.
    movlw 0F0h          ; Valor que hay que cargar en TRISB para programar
    movwf TRISB         ; RB<3:0> como salidas y RB<7:4> como entradas.
    bcf OPTION_REG, NOT_RBPU ; Habilitar los pull-up internos del puerto B.
    bcf STATUS, RP0     ; Se selecciona el banco 0.
    return
```

; LEETEC: Rutina de exploración del teclado matricial.

; Esta rutina espera la pulsación de una tecla, devolviendo en W ; su código de exploración.

; Entradas: ninguna.

; Salidas: El código de exploración en W. Los bits W<3:2> contienen la columna

; donde está la tecla pulsada y los bits W<1:0> contienen la fila.

;

LEETEC:

```
    movlw 0F0h
    movwf PORTB          ; Todas las filas en '0'.
```

	nop		
TEC10:	movf	PORTB, W	; Esperar la liberación del teclado: ; Se leen todas las columnas a la vez. La parte baja de W ; está en 0. En la parte alta, si hay tecla pulsada, el bit ; correspondiente a la línea de retorno de la tecla ; está en 0 y los restantes bits están en 1.
	xorlw	0F0h	; Se invierte la situación en la parte alta de W: el bit ; correspondiente a la línea de retorno de la tecla pulsada ; está en 1 y los restantes bits en 0. La parte baja de W ; no se altera. Se activa Z si no hay tecla pulsada.
	btfss	STATUS, Z	; ¿Abiertas todas las teclas? Z = 1?
	goto	TEC10	; No - esperar la liberación del teclado.
TEC20:	movlw	0F0h	; Sí - continuar. Esperar la pulsación de una tecla:
	movwf	PORTB	; Todas las filas en '0'.
	nop		
	movf	PORTB, W	; Se leen todas las columnas a la vez.
	xorlw	0F0h	; Z = 0 si hay tecla pulsada.
	btfsc	STATUS, Z	; ¿Hay alguna tecla pulsada? Z = 0?
	goto	TEC20	; No - esperar la pulsación de una tecla.
TEC30:	call	DEMORA20	; Sí - continuar.
TEC40:	movlw	0FFh	; Esperar 20 ms debido al rebote.
	movwf	PORTB	; Explorar las filas del teclado sucesivamente para ; averiguar qué tecla se pulsó:
	nop		; Todas las filas en '1'.
FILA0:			; Explorar fila 0:
	movlw	0	
	movwf	FILA	
	bcf	PORTB, 0	; Poner 0 en fila 0 (RB0).
	nop		
	movf	PORTB, W	; Se leen todas las columnas a la vez.
	call	IDENTIFICA	; Si hay tecla pulsada, se identifica la columna donde está ; la tecla .
	btfsc	STATUS, C	; ¿Hay tecla pulsada? C = 1?
	goto	TEC50	; Sí - Se encontró una tecla pulsada. Terminar exploración.
	bsf	PORTB, 0	; No - Poner 1 en fila explorada y pasar a la siguiente.
	nop		
FILA1:			; Explorar fila 1:
	movlw	1	
	movwf	FILA	
	bcf	PORTB, 1	; Poner 0 en fila 1 (RB1).
	nop		
	movf	PORTB, W	; Se leen todas las columnas a la vez.
	call	IDENTIFICA	; Si hay tecla pulsada, se identifica la columna donde está ; la tecla.

btfsc	STATUS, C	; ¿Hay tecla pulsada? ¿C = 1?
goto	TEC50	; Sí - Se encontró una tecla pulsada. Terminar exploración.
bsf	PORTB, 1	; No - Poner 1 en fila explorada y pasar a la siguiente.
nop		
FILA2:		; Explorar fila 2:
movlw	2	
movwf	FILA	
bcf	PORTB, 2	; Poner 0 en fila 2 (RB2).
nop		
movf	PORTB, W	; Se leen todas las columnas a la vez.
call	IDENTIFICA	; Si hay tecla pulsada, se identifica la columna donde está la tecla .
btfsc	STATUS, C	; ¿Hay tecla pulsada? ¿C = 1?
goto	TEC50	; Sí - Se encontró una tecla pulsada. Terminar exploración.
bsf	PORTB, 2	; No - Poner 1 en fila explorada y pasar a la siguiente.
nop		
FILA3:		; Explorar fila 3:
movlw	3	
movwf	FILA	
bcf	PORTB, 3	; Poner '0' en fila 3 (RB3).
nop		
movf	PORTB, W	; Se leen todas las columnas a la vez.
call	IDENTIFICA	; Si hay tecla pulsada, se identifica la columna donde está la tecla.
btfsc	STATUS, C	; ¿Hay tecla pulsada? ¿C=1?
goto	TEC50	; Sí - Se encontró una tecla pulsada. Terminar exploración.
bsf	PORTB, 3	; No - Poner 1 en fila explorada.
nop		
goto	TEC20	; Si se ha llegado hasta aquí sin encontrar una tecla pulsada es porque la pulsación era falsa. En este caso, ir a esperar una nueva pulsación.
TEC50:		; Se encontró una tecla pulsada y se construye el código de exploración de 4 bits: FILA:COLUMNNA.
rff	FILA, f	; El número de la fila se pone en
rff	FILA, W	; los bits 2 y 3
andlw	0FCh	; del registro W.
iorwf	COLUMNNA, W	; El número de la columna se pone en los bits 0 y 1 de W.
andlw	0Fh	; Se pone 0 en la parte alta de W.
return		; Se retorna con el código de exploración en W<3:0>.

; IDENTIFICA: Rutina para identificar la columna donde está la tecla pulsada,

; Entradas: En W está la lectura del puerto B.

; Salidas: El bit C del registro STATUS es puesto a 1 si se encuentra una

; tecla pulsada, de lo contrario C es puesto a 0.

; El número de la columna donde está la tecla se coloca en
; el registro nombrado COLUMNNA.

;

IDENTIFICA:

movwf	TEMP	; La información de entrada es puesta en el registro TEMP.
btfsc	TEMP, 4	; ¿El bit de la columna 0 es 0?
goto	COL1	; No - Pasar a examinar la siguiente columna.
movlw	0	; Sí - Hay una tecla pulsada en esa columna.
movwf	COLUMNNA	; Guardar el número de la columna en el registro COLUMNNA.
goto	IDENT_FIN	; Ir a finalizar la rutina.

COL1:

btfsc	TEMP, 5	; ¿El bit de la columna 1 es 0?
goto	COL2	; No - Pasar a examinar la siguiente columna.
movlw	1	; Sí - Hay una tecla pulsada en esa columna.
movwf	COLUMNNA	; Guardar el número de la columna en el registro COLUMNNA.
goto	IDENT_FIN	; Ir a finalizar la rutina.

COL2:

btfsc	TEMP, 6	; ¿El bit de la columna 2 es 0?
goto	COL3	; No - Pasar a examinar la siguiente columna.
movlw	2	; Sí - Hay una tecla pulsada en esa columna.
movwf	COLUMNNA	; Guardar el número de la columna en el registro COLUMNNA.
goto	IDENT_FIN	; Ir a finalizar la rutina.

COL3:

btfsc	TEMP, 7	; ¿El bit de la columna 3 es 0?
goto	COL4	; No - Pasar a examinar la siguiente columna.
movlw	3	; Sí - Hay una tecla pulsada en esa columna.
movwf	COLUMNNA	; Guardar el número de la columna en el registro COLUMNNA.
goto	IDENT_FIN	; Ir a finalizar la rutina.

COL4:

bcf	STATUS, C	; Indicar que no hubo tecla pulsada haciendo C = 0.
return		; Retomar.

IDENT_FIN:

bsf	STATUS, C	; Indicar que hubo tecla pulsada haciendo C = 1.
return		; Retomar.

;DEMORA20: Rutina que demora 20 ms.

DEMORA20

;		
;	; Aquí va el código de esta subrutina que produce una demora de 20ms	
;		
return		

end

5.3.3 Visualizadores numéricos de 7 segmentos

Los visualizadores de 7 segmentos con diodos LED se usan sobre todo para representar información numérica. La figura 5.14 muestra su circuito interno y su representación simbólica. Hay dos tipos de elementos de 7 segmentos: los de ánodo común y los de cátodo común, según estén conectados entre sí todos los ánodos o todos los cátodos de los LED, respectivamente. En los elementos de ánodo común, para que se active (emita luz) un segmento, el terminal correspondiente debe excitarse con una tensión baja (correspondiente al nivel lógico '0', si la lógica es positiva), mientras que el ánodo común debe estar puesto a una tensión positiva alta (V_{DD}). En los elementos de cátodo común la situación es la inversa: cada segmento se activa con una tensión alta (correspondiente al nivel lógico '1') y el cátodo común debe estar a 0 V (V_{SS}).

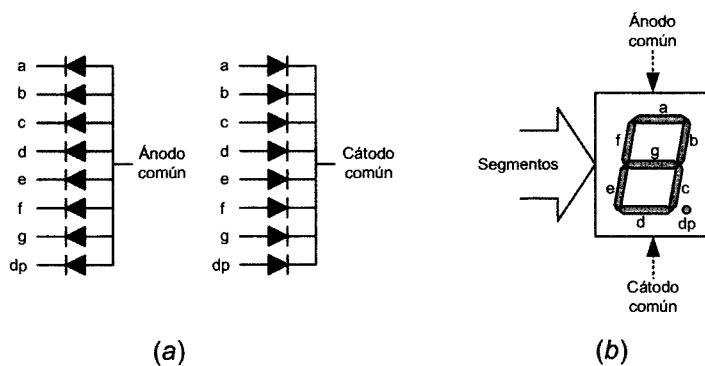


Figura 5.14 Los visualizadores de 7 segmentos están formados por un conjunto de diodos LED conectados entre sí según una de estas dos configuraciones: ánodo común o cátodo común. En (a) se muestran los circuitos internos de ambos tipos de visualizadores y en (b) su representación simbólica.

Cuando hay varios elementos de 7 segmentos formando una pantalla, visualizador o *display*, se prefiere la conexión multiplexada de estos dispositivos a los puertos paralelos del microcontrolador. La figura 5.15 muestra un posible esquema de conexión utilizando elementos de 7 segmentos de ánodo común. En este caso, todos los segmentos del mismo nombre han sido conectados entre sí y a los terminales del puerto B a través de resistencias limitadoras de corriente R_s . Cada segmento se activa (emite luz) cuando en el terminal correspondiente del puerto B hay un nivel lógico '0'. El ánodo común de cada elemento funciona como un terminal de selección gobernado desde el puerto A. La selección de un elemento se hace mediante un nivel lógico '0' en el terminal del puerto A correspondiente. Los visualizadores se activan sucesivamente, con una frecuencia de repetición (frecuencia de refrescamiento)

adecuada. En sincronismo con su activación, en los terminales de los segmentos se coloca la información correspondiente al elemento seleccionado. Si la frecuencia de refrescamiento es suficientemente alta, un observador humano verá todos los elementos de visualización activados a la vez. Para que no se vea el parpadeo, la frecuencia mínima de refrescamiento de cada dígito debe estar entre 40 Hz y 200 Hz.

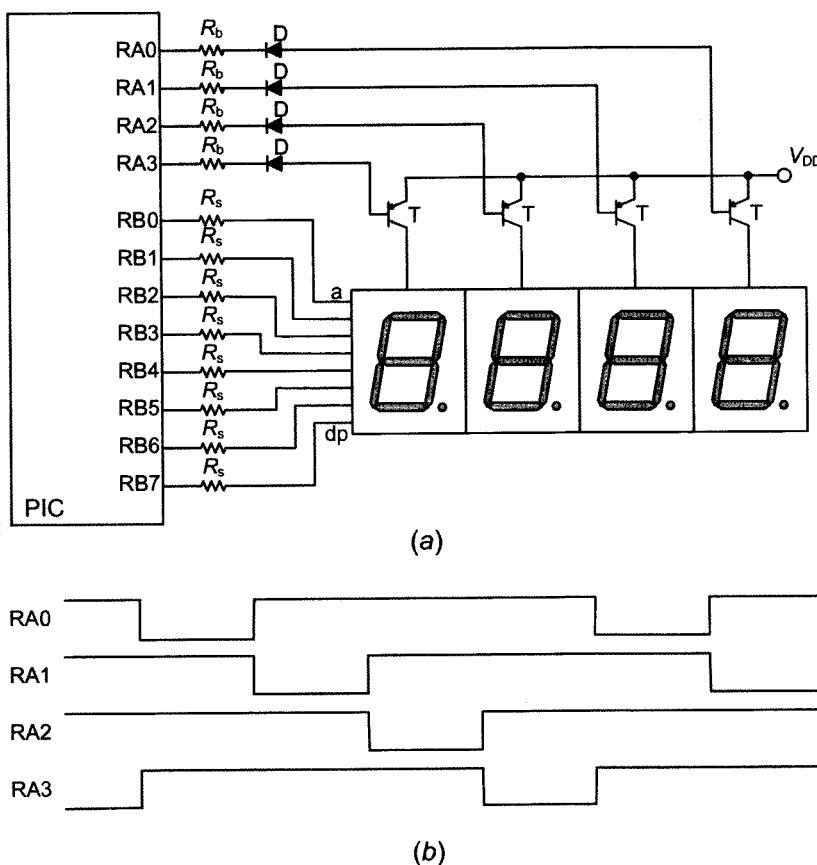


Figura 5.15 (a) Circuito de un visualizador de 4 elementos de 7 segmentos de ánodo común conectados a los puertos A y B de un microcontrolador PIC. Las resistencias R_s limitan la corriente que circula por los segmentos, mientras que las resistencias R_b deben garantizar la saturación de los transistores que activan los terminales de selección de cada elemento. Los diodos aseguran que los transistores vayan a corte con un nivel lógico '1' en los terminales del puerto A. (b) Formas de onda en cada uno de los terminales de selección. La frecuencia mínima de estas señales debe estar entre 40 Hz y 200 Hz para que no se perciba parpadeo alguno. Cada elemento está seleccionado durante una cuarta parte del tiempo.

Cada dígito de un visualizador multiplexado permanece encendido una fracción del tiempo total ($\frac{1}{4}$ en el ejemplo de la figura 5.15), por lo que hay que aumentar el nivel de corriente en los segmentos, aproximadamente en la misma proporción ($\times 4$), para que la iluminación sea satisfactoria. Es decir, si la corriente de trabajo de un LED es de unos 10 mA, al trabajar en forma multiplexada hay que suministrar pulsos de 40 mA por cada segmento. Este valor se puede ajustar con la resistencia R_s . La resistencia R_b debe garantizar la saturación de los transistores. Los diodos en las bases de los transistores tienen la misión de garantizar que los transistores se corten satisfactoriamente.

La atención al circuito de la figura 5.15 se puede hacer por interrupción, según se ilustra en la figura 5.16. Para un visualizador de cuatro elementos hace falta un oscilador de frecuencia igual a 4 veces la frecuencia de refreshamiento, para que genere las interrupciones al microcontrolador. Con cada interrupción se visualiza la información de un dígito. Se pueden reservar cuatro registros de la memoria RAM de datos para almacenar la "imagen" de la información visualizada en el *display*, en forma de una tabla (tabla 1). En cada posición de esta tabla se coloca el código de 7 segmentos del dígito que se desea visualizar. Por otra parte, en otros cuatro registros (tabla 2) se puede de tener la información de control para seleccionar los dígitos del *display*, es decir, las palabras que hay que enviar al puerto A para la selección correcta y secuencial de los dígitos. Por último, en otro registro se coloca un puntero al dígito que debe ser refrescado por el programa que atiende la interrupción. En cada interrupción se incrementa el puntero y se envía a los puertos A y B la información contenida en las tablas.

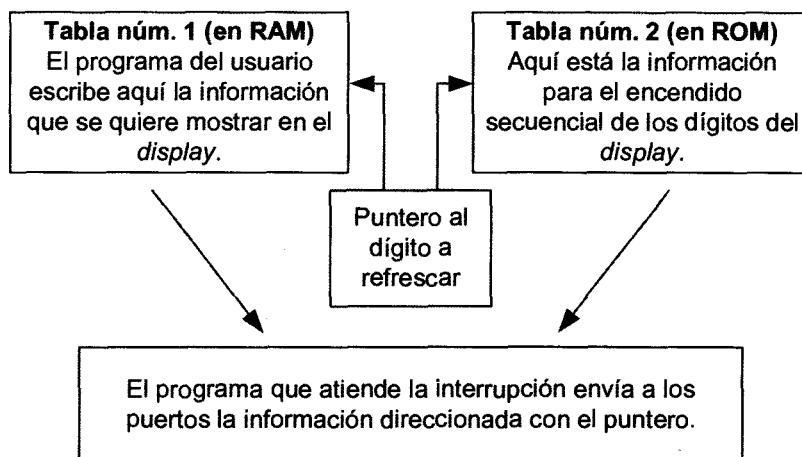


Figura 5.16 Tratamiento de software que se puede dar al visualizador de elementos de 7 segmentos de la figura 5.15.

Para el programa principal, la tabla 1 es la imagen del visualizador en memoria RAM. Cada nueva información que se desea visualizar debe ser escrita en esa tabla. Antes de este paso, hay que convertir al código de 7 segmentos la información original que puede estar en BCD, ASCII o en un código no estándar.

5.3.4 Visualizadores alfanuméricicos de cristal líquido

Los módulos visualizadores de cristal líquido o módulos LCD se emplean profusamente para mostrar información alfanumérica. Son muy populares los módulos compuestos básicamente por una pantalla o visualizador LCD y un microcontrolador especializado que se encarga de su manejo y ofrece una interfaz cómoda, en hardware y software, con el sistema que maneja el módulo visualizador. En la pantalla se pueden ver una o dos líneas con un cierto número de caracteres alfanuméricos por línea, según el modelo del módulo LCD. Un controlador especializado muy común es el HD44780 (Renesas, antes Hitachi).

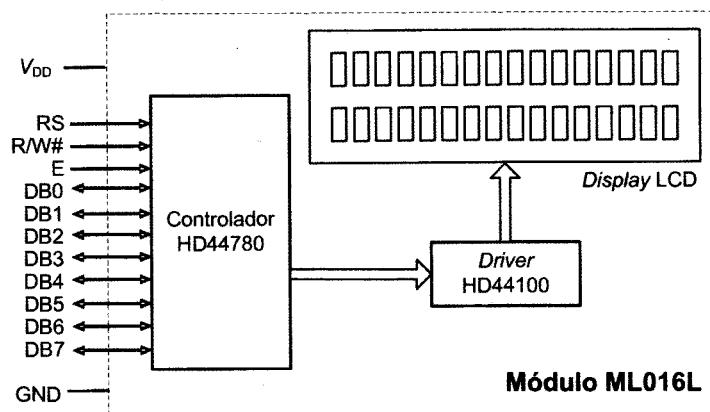


Figura 5.17 Componentes del módulo de cristal líquido ML016L para una interfaz de 8 bits.

El microcontrolador HD44780 puede manejar directamente una pantalla LCD de 1 ó 2 líneas de 8 caracteres cada una. Si la pantalla es mayor, hacen falta circuitos auxiliares, como el *driver* HD44100, que permite manejar otros ocho caracteres por línea. Se fabrican módulos LCD basados en el controlador HD44780 para visualizar 1 ó 2 líneas de 16, 32, etc. caracteres. Por ejemplo, la figura 5.17 muestra el esquema en bloques de un módulo LCD LM016L que utiliza un controlador HD44780 y un *driver* HD44100 para manejar una pantalla LCD de 2 líneas con 16 caracteres cada una.

El microcontrolador HD44780 tiene una memoria RAM interna para datos (DDRAM), en la que se pueden almacenar los códigos ASCII de hasta 80 caracteres alfanuméricos. Para visualizar los caracteres contenidos en la

DDRAM, esta memoria se puede organizar en 1 ó 2 líneas. Cada línea opera como una memoria circular. En una memoria (o *buffer*) circular, una vez se ha alcanzado la posición final, la escritura o lectura de un nuevo dato se realiza en la posición inicial de la memoria.

Si la DDRAM se organiza en una sola línea, entonces funciona como una memoria circular de 80 bytes, con las direcciones consecutivas que van desde 00h hasta 4Fh.

Si la DDRAM se organiza en 2 líneas, se tienen entonces dos memorias circulares independientes de 40 caracteres cada una y las direcciones no son exactamente consecutivas: la primera línea ocupa las direcciones 00h a 27h, mientras que la segunda ocupa las direcciones 40h a 67h de la DDRAM, según muestra la figura 5.18.

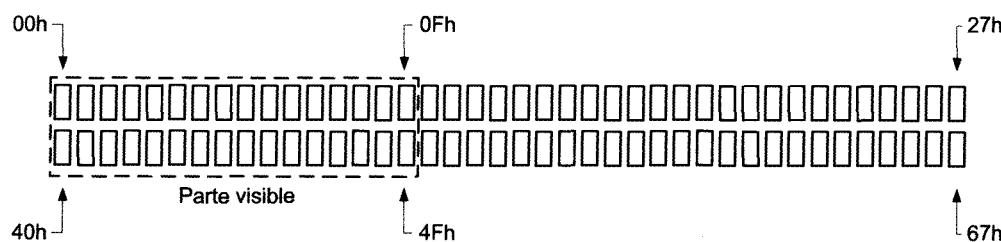


Figura 5.18 La memoria RAM de datos (DDRAM) del controlador HD44780 se puede organizar en una o dos líneas. Si está organizada en dos líneas (caso mostrado en esta figura), cada línea funciona como una memoria circular de 40 bytes, con las direcciones indicadas. La línea discontinua encierra la parte de la DDRAM que es visible en un módulo LM016L (2 líneas de 16 caracteres cada una) después de energizar el dispositivo. La parte visible se puede desplazar a lo largo de la DDRAM mediante las órdenes correspondientes (ver tabla 5.6).

El HD44780 también tiene un generador de caracteres en ROM (memoria ROM donde está almacenada la matriz de puntos de cada carácter, es decir, la información correspondiente al aspecto de los caracteres) y otro en RAM (CGRAM). El generador de caracteres en RAM permite que el usuario defina caracteres no estándares, que no están diseñados en el generador de caracteres en ROM.

El HD44780 tiene un repertorio de órdenes para manipular el módulo visualizador.

Algunas de las características de los módulos visualizadores de cristal líquido que utilizan el controlador HD44780 son las siguientes:

- Posibilidad de conectarse a puertos paralelos de 4 u 8 bits.
- Almacenamiento de hasta 80 caracteres en una memoria RAM interna (DDRAM) de 80 bytes (figura 5.18).
- Generador de caracteres en ROM interna con:
 - 160 caracteres del tipo 5 × 7 puntos.
 - 32 caracteres del tipo 5 × 10 puntos.
- Generador de caracteres en RAM interna, para definir nuevos caracteres por el usuario, con:
 - 8 caracteres del tipo 5 × 7 puntos.
 - 4 caracteres del tipo 5 × 10 puntos.
- Muy bajo consumo (usan tecnología CMOS), por lo que pueden alimentarse con baterías.
- Circuito de RESET interno automático en el encendido.
- Amplio repertorio de órdenes: borrado de display, parpadeo de caracteres y del cursor, desplazamientos del cursor y del display, apagado/encendido del cursor y el display, etc.

Los módulos LCD disponen de una interfaz digital para transferir datos y órdenes entre un sistema con un microprocesador o microcontrolador y el módulo LCD. Esta interfaz tiene 3 líneas de control y 4 u 8 líneas para datos (según la interfaz sea de 4 u 8 bits respectivamente).

Para conectar un módulo LCD a un PIC de clase media se pueden utilizar los puertos paralelos A y B. Mediante el software adecuado se generan las señales con la secuencia correcta. La figura 5.19 muestra la conexión para una interfaz de 8 bits, que tiene las señales siguientes.

- RS (Register Select, datos/control#). Esta señal indica al módulo LCD si el byte que se envía por DB0-DB7 es una orden (RS = '0') o un dato (RS = '1').
- R/W# (Read/Write, lectura/escritura#). Esta señal indica lectura si está en '1' y escritura si está en '0'.
- E (Enable, habilitar). Con esta señal en '1' se habilita el dispositivo. La captura de datos u órdenes por el módulo LCD se realiza con el flanco de caída de esta señal.
- DB0 a DB7 (Data bus, bus de datos). Por estas líneas transitan las órdenes y los datos en ambas direcciones.

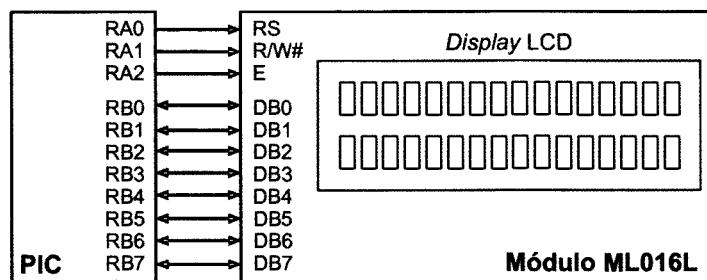


Figura 5.19 Conexión de un módulo de cristal líquido a un microcontrolador PIC, usando los puertos A y B.

La figura 5.20 ilustra la escritura de una orden o un dato en el módulo LCD.

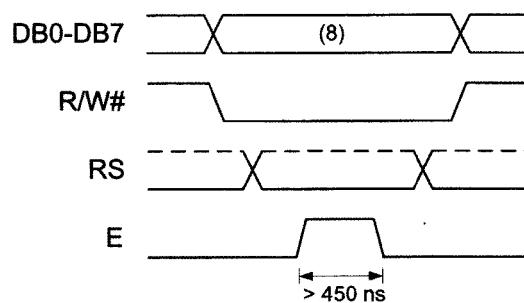


Figura 5.20 Señales para escribir datos u órdenes en el módulo de cristal líquido. La escritura de la orden (con RS = '0') o dato (con RS = '1') se hace efectiva en sincronismo con el flanco de caída de la señal E.

La tabla 5.6 muestra el repertorio de órdenes aceptadas por los módulos LCD que utilizan el controlador HD44780.

Tabla 5.6 Órdenes aceptadas por el controlador HD44780.

Órdenes	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPCION
Borrar display	0	0	0	0	0	0	0	0	0	1	Pone 20h (código ASCII de espacio) en la DDRAM, la selecciona y pone el AC en 0
Ir a la posición inicial	0	0	0	0	0	0	0	0	1	X	Selecciona la DDRAM y pone el AC en 0. Lleva el display a esa posición inicial. El contenido de la DDRAM no se modifica.
Seleccionar modo de entrada	0	0	0	0	0	0	0	1	I/D	S	Selecciona la dirección de movimiento del cursor (I/D) y si el display se desplaza o no (S). Estas operaciones se pueden efectuar durante la escritura o lectura de datos.
Controlar display	0	0	0	0	0	0	1	D	C	B	Enciende o apaga el display (D), el cursor (C) y activa el parpadeo del cursor (B).
Desplazar cursor o display	0	0	0	0	0	1	S/C	R/L	X	X	Desplaza el display o el cursor (S/C) en una dirección (R/L).

Seleccionar función	0	0	0	0	1	DL	N	F	X	X	Selecciona la longitud (4 ó 8) del BUS de datos (DL), el número de líneas del <i>display</i> (N) y el formato de los caracteres (F)
Seleccionar CGRAM	0	0	0	1	Dirección de la CGRAM que será puesta en el AC.				Pone una dirección de la CGRAM en el AC. Después de este orden, los datos escritos o leídos van a parar o provienen de la CGRAM.		
Seleccionar DDRAM	0	0	1	Dirección de la DDRAM, que será puesta en el AC.				Pone una dirección de la DDRAM en el AC. Después de este orden los datos escritos o leídos van a parar o provienen de la DDRAM.			
Leer BF y AC	0	1	BF	Contenido del AC				Lee el estado de la bandera de ocupado (BF) y el contenido del AC.			
Escribir dato.	1	0	Dato a escribir				Escribe el dato en la dirección apuntada por AC. El dato se escribe en la DDRAM o en la CGRAM, según la última selección realizada.				
Leer dato.	1	1	Dato leído.				Lee un dato en la dirección apuntada por AC. El dato se lee de la DDRAM o CGRAM, según la última selección realizada.				
I/D – 1: incrementa; 0: decrementa. S – 1: desplazamiento automático del <i>display</i> S/C – 1: desplazamiento del <i>display</i> ; 0: movimiento del cursor. R/L – 1: desplazamiento a la derecha; 0: desplazamiento a la izquierda. DL – 1: interfaz de 8 bits; 0: interfaz de 4 bits. N – 1: <i>display</i> de 2 líneas; 0: <i>display</i> de 1 línea. F – 1: caracteres de 5x10 puntos; 0: caracteres de 5x7 puntos. BF – Bandera de ocupado. 1: <i>display</i> ocupado, 0: <i>display</i> puede aceptar órdenes o datos. DDRAM – memoria RAM de datos del controlador. CGRAM – memoria RAM del generador de caracteres. AC – Contador de Direcciones											

El controlador HD44780 dispone de un registro interno denominado Contador de Direcciones (AC: *Address Counter*), donde está la dirección de la DDRAM o CGRAM donde se escribirá o leerá un dato. La procedencia del contenido del registro AC, es decir, si es una dirección de la DDRAM o de la CGRAM, depende de cuál de estas dos memorias fue la seleccionada más recientemente mediante la orden correspondiente. Una vez seleccionada una memoria, todas las escrituras y lecturas de datos se realizan sobre esa memoria, usando al registro AC como puntero.

Por ejemplo, la selección de la DDRAM, operación necesaria para proceder a escribir datos en ella, se puede hacer con las órdenes “Borrar *display*” e “Ir a posición inicial”, las cuales, además de seleccionar la DDRAM, ponen el AC en 0. La orden “Seleccionar DDRAM” permite, además, situar un valor en el AC.

Con la escritura (o lectura) de un dato, el registro AC se incrementa o decremente, según el modo de entrada seleccionado.

En general, cualquier operación de ejecución de una orden o de visualización de un dato, mantiene ocupado al controlador durante un tiempo relativamente grande, del orden de varios microsegundos. Durante ese tiempo no se debe enviar ninguna orden o dato al controlador. De ahí que antes de enviar una nueva orden o dato, resulte necesario indagar cuál es el estado del controlador. Para facilitar esta operación, se dispone del bit indicador BF (*Busy Flag*), el cual está en '1' mientras el controlador está ocupado y en '0' si el controlador se encuentra listo para recibir nuevos datos u órdenes. Este indicador se puede leer en cualquier momento con una operación de lectura (R/W# = '1') de una orden (RS = '0'); el bit 7 de la palabra devuelta (en DB0 a DB7) por el controlador es el indicador BF. Ver la rutina OCUPADO del ejemplo 5.6.

Ejemplo 5.6

En este ejemplo se presentan los listados en lenguaje ensamblador de algunas rutinas básicas para operar un módulo de cristal líquido que contiene un controlador HD44780. El módulo está conectado a un microcontrolador PIC a través de los puertos A y B, según el esquema de la figura 5.19.

Se presentan cuatro rutinas: INICIO para la programación inicial del *display*, WR_CMD y WR_DATO para escribir órdenes y datos en el *display*, respectivamente, y finalmente la rutina LCD_OCUPADO, usada para comprobar el estado del módulo LCD mediante la lectura del valor del indicador de ocupado (bit BF) y con ello saber si se puede o no escribir un dato o una orden en el módulo.

```
list      p=16f873
#include <p16f873.inc>
```

;Descripción del hardware:

P_DATOS	equ	PORTB	; Puerto para manejar las líneas de datos del display.
P_TRIS	equ	TRISB	
P_CTRL	equ	PORTA	; Puerto para manejar las líneas de control del display.
RS	equ	0	; Bit de control de la línea RS.
RW	equ	1	; Bit de control de la línea RW.
E	equ	2	; Bit de control de la línea E.

;Otras declaraciones:

TEMP	equ	0x020	; Registro temporal utilizado por las subrutinas.
------	-----	-------	---

;INICIO: Rutina de iniciación del display.

INICIO:

; La bandera de ocupado BF no está disponible aún.			
clrf	P_CTRL	;	Las líneas de control en 0.
call	DEMO15	;	Esperar 15 ms.
; La bandera BF está disponible a partir de aquí.			

```

movlw 38h ; Display de 8 bits, 2 líneas, caracteres 5 × 7.
call WR_CMD
movlw 08h ; Apagar el display y el cursor.
call WR_CMD
movlw 01h ; Borrar display y poner AC = 0.
call WR_CMD
movlw 0Ch ; Encender display con cursor apagado.
call WR_CMD
movlw 06h ; Modo de entrada: cursor se desplaza a la derecha.
call WR_CMD
return

```

; WR_CMD: Rutina para escribir una orden en el display.

; Entrada: la orden debe estar en W.

WR_CMD:

```

movwf TEMP ; Guardar la orden en TEMP.
call OCUPADO ; Esperar a que el display esté listo.
bcf P_CTRL, RW ; Preparar la escritura (RW = 0)
bcf P_CTRL, RS ; de una orden (RS = 0).
bsf P_CTRL, E ; Habilitar el display (E = 1).
movf TEMP, W ; Poner la orden en W.
movwf P_DATOS ; Enviarla al display.
bcf P_CTRL, E ; Inhabilitar el display (E = 0).
return

```

; WR_DATO: Rutina para escribir un dato en el display

; Entrada: el dato debe estar en W.

WR_DATO:

```

movwf TEMP ; Guardar la orden en TEMP
call OCUPADO ; Esperar a que el display esté listo
bcf P_CTRL, RW ; Preparar la escritura (RW = 0)
bsf P_CTRL, RS ; de un dato (RS = 1).
bsf P_CTRL, E ; Habilitar el display (E = 1).
movf TEMP, W ; Poner el dato en W.
movwf P_DATOS ; Enviarlo al display.
bcf P_CTRL, E ; Inhabilitar el display (E = 0).
return

```

; OCUPADO: Rutina para esperar si el display está ocupado

; Esta rutina comprueba la bandera BF, espera mientras BF = 1

; y retoma cuando BF = 0.

OCUPADO:

```

bsf STATUS,RP0 ; Seleccionar banco 1 de registros.
movlw 0FFh ; Poner el puerto de datos en entrada
movwf P_TRIS ; escribiendo FFh en el registro TRIS correspondiente.
bcf STATUS, RP0 ; Seleccionar el banco 0.

```

```
bsf      P_CTRL, RW ; Preparar la lectura (RW = 1).
bcf      P_CTRL, RS ; Se va a leer una orden (RS = 0).
OCUP10:
bsf      P_CTRL, E   ; Habilitar el display (E = 1).
nop
movf    P_DATOS, W ; Leer el display. El indicador BF es el bit 7.
bcf      P_CTRL, E   ; Inhabilitar el display (E = 0).
andlw   80h        ; Se aísla la bandera BF.
btfsf   STATUS, Z  ; ¿BF es 0?
goto    OCUP10     ; no - display ocupado: volver a chequear,
bcf      P_CTRL, RW ; sí - display desocupado, fin de lectura (RW = 1).
bsf      STATUS, RP0; Seleccionar banco 1.
movlw   0           ; Poner el puerto de datos en salida
movwf   P_TRIS      ; escribiendo 00h en el registro TRIS correspondiente.
bcf      STATUS, RP0 ; Seleccionar banco 0.
return
               ; Retornar.
```

;DEMO15: Rutina que demora 15ms.

DEMO15:

```

;
; Aquí va el código de esta subrutina
;
return
```

end

6 Los temporizadores

En muchas aplicaciones el microcontrolador debe trabajar con la variable tiempo. Por ejemplo, para generar señales de una determinada frecuencia, para medir la duración de una señal, o simplemente para llevar la fecha y la hora, el microcontrolador necesita algún recurso para contar el tiempo con precisión.

En los microcontroladores PIC de clase media hay hasta tres módulos básicos para temporizar, que se identifican con los nombres Timer0, Timer1 y Timer2. Todos los PIC disponen al menos del Timer0. Algunos PIC tienen uno o dos módulos adicionales para temporizar, que amplían las posibilidades de los módulos básicos. Son los denominados módulos de Comparación, Captura y Modulación Pulsos en Anchura, o módulos CCP (*Capture/Compare/PWM*), que comparten componentes y funciones con el Timer1 y el Timer2.

En este capítulo se estudia la estructura, funcionamiento y programación de cada uno de los módulos disponibles en los microcontroladores PIC para trabajar directamente con la variable tiempo. El capítulo incluye varios ejemplos ilustrativos del funcionamiento y programación de estos módulos.

6.1 Los temporizadores en los microcontroladores PIC

Cada uno de los temporizadores disponibles en un PIC de clase media tiene, como elemento esencial, un contador sincrónico ascendente de 8 ó 16 bits. Estos contadores se pueden programar para contar pulsos internos o externos, según se expondrá más adelante al estudiar cada temporizador por separado. El número almacenado en cada contador (valor de la cuenta) se puede leer o modificar mediante la lectura o escritura de registros de funciones especiales asociados al temporizador en cuestión. El desbordamiento de los contadores queda reportado en bits indicadores disponibles en esos registros, y puede generar también una solicitud de interrupción al microcontrolador.

Los temporizadores pueden disponer de un contador asincrónico auxiliar. Este contador auxiliar se inserta en el camino de los pulsos, antes del contador principal, en cuyo caso funciona como un pre-divisor (*prescaler*) o después del contador principal, funcionando entonces como post-divisor (*postscaler*). Los temporizadores Timer0 y Timer1 tienen solamente un pre-divisor; el Timer2, en cambio, dispone de un pre-divisor y un post-divisor. La figura 6.1 muestra el esquema general de los módulos para temporizar

Timer0, Timer1 y Timer2, cuyas principales características se resumen en la tabla 6.1 y se estudian a continuación.

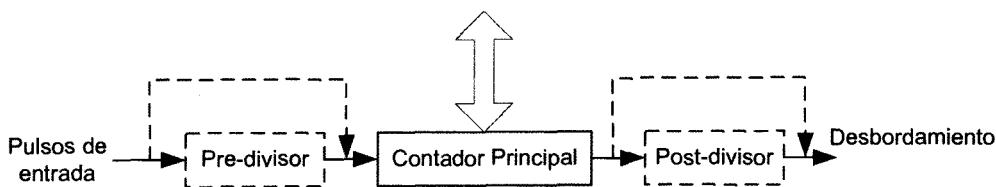


Figura 6.1 Esquema de bloques general de los temporizadores de los PIC de clase media. Todos tienen un contador principal ascendente de 8 ó 16 bits y un pre-divisor programable. El Timer2 tiene también un post-divisor. Los pulsos que se van a contar pueden ser internos o externos. El desbordamiento se reporta en un bit de un registro de funciones especiales y puede generar una solicitud de interrupción al PIC.

Tabla 6.1 Características principales de los temporizadores de los PIC de clase media.

Temporizador	Tamaño	Pre-divisor divide entre:	Post-divisor divide entre:	SFR donde está la cuenta	Desbordamiento
Timer0	8 bits	2, 4, ..., 256	NO	TMR0	bit T0IF de OPTION
Timer1	16 bits	1, 2, 4, 8	NO	TMR1H, TMR1L	bit TMR1IF de PIR1
Timer2	8 bits	1, 4, 8	1, 2, ..., 16	TMR2	bit TMR2IF de PIR2

6.1.1 El módulo Timer0

El módulo Timer0 consta básicamente de un pre-divisor y un contador ascendente de 8 bits (figura 6.2). El pre-divisor es un contador asincrónico ascendente con factor de división programable cuyo conteo no es visible para el programador. El contador ascendente de 8 bits se puede leer o escribir a través del registro de funciones especiales TMR0.

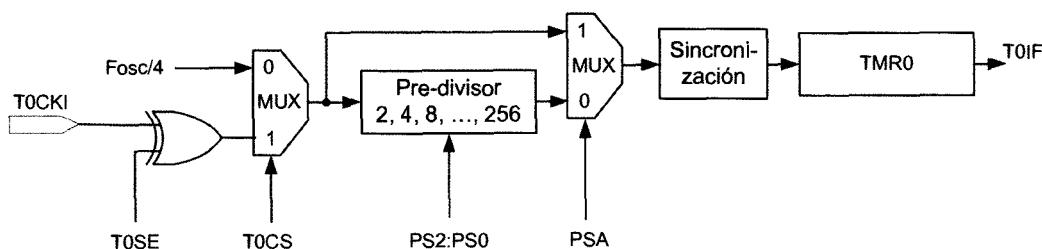


Figura 6.2 Diagrama de bloques del Timer0. El Timer0 puede trabajar como contador de ciclos de máquina (modo temporizador, T0CS = 0) o como contador de pulsos externos (modo contador, T0CS = 1). TMR0 es un contador ascendente de 8 bits, cuyo desbordamiento activa el indicador T0IF del registro INTCON. Antes de llegar a TMR0, los pulsos son sincronizados con el reloj del microcontrolador y pueden pasar o no por un pre-divisor cuyo factor de división es programable.

El Timer0 puede trabajar como contador de ciclos de máquina o como contador de pulsos externos. En el primer caso se dice que el módulo opera en el modo temporizador (*timer*). Si cuenta pulsos externos, el módulo opera en el modo contador (*counter*). Los pulsos externos llegan por el terminal T0CKI.

A su paso por el Timer0, los pulsos llegan al bloque de sincronización. En este bloque, los pulsos que entran son muestrados en dos instantes de tiempo dentro de cada ciclo de máquina, de lo que resulta una señal cuyos flancos ocurren en fase con el reloj del microcontrolador. Con esta señal sincronizada se excita el contador TMR0. Para que no se pierdan pulsos en la sincronización, es necesario que los pulsos que entran en el bloque de sincronización permanezcan en 1 o en 0 al menos durante la mitad del tiempo que dura un ciclo de máquina. Cuando el Timer0 trabaja en el modo contador, el bloque de sincronización determina el valor mínimo del período (o la frecuencia máxima) de los pulsos que entran por el terminal T0CKI. Si T_{osc} es el período del oscilador principal del microcontrolador y P es el factor de división del pre-divisor, el período T_i de los pulsos que entran por el terminal T0CKI debe cumplir

$$T_i > \frac{4 \times T_{osc}}{P} \quad (6.1)$$

donde $P = 1$ si no se usa el pre-divisor y $P = 2, 4, \dots, 256$ si se usa.

En el modo de bajo consumo, el oscilador del microcontrolador se paraliza y por lo tanto no funciona el bloque de sincronización del Timer0, de manera que el Timer0 tampoco funciona mientras el microcontrolador está en dicho modo.

Hay tres registros de funciones especiales asociados al Timer0: TMR0, OPTION e INTCON. La figura 6.3 muestra los nombres de los bits de los registros OPTION e INTCON. El registro TMR0 almacena el valor que tiene el contador del Timer0. Este valor se puede leer o escribir en cualquier momento desde el programa que ejecuta el microcontrolador. Cuando se escribe un valor en TMR0, la cuenta del pre-divisor, si está asignado al Timer0, se pone a 0. Además, una escritura en el registro TMR0 inhibe el conteo del Timer0 durante dos ciclos de máquina.

El desbordamiento del Timer0 hace que el indicador T0IF (bit INTCON<2>) pase a 1. Si la atención al Timer0 se hace usando la técnica de E/S programada, entonces se debe consultar el bit T0IF para saber si el Timer0 se ha desbordado.

Una vez comprobado el desbordamiento, este indicador debe ser puesto a 0 por software.

OPTION								
7	6	5	4	3	2	1	0	
RBPU#	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	

INTCON								
7	6	5	4	3	2	1	0	
GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	

Figura 6.3 Registros de funciones especiales OPTION e INTCON. En OPTION están los bits de configuración del Timer0: T0CS configura al Timer0 como temporizador o como contador; T0SE configura el flanco de la señal externa con el que se incrementa el Timer0 si ha sido programado como contador; PSA asigna el pre-divisor al Timer0 o al Perro Guardián y PS2, PS1 y PS0 programan el factor de división del pre-divisor. En INTCON está el indicador de desbordamiento del Timer0: TOIF; con el bit TOIE se habilita la solicitud de interrupción al microcontrolador por esta causa.

Si la interrupción del Timer0 está habilitada, lo cual se hace poniendo a 1 el bit TOIE (INTCON<5>), cuando se desborda el Timer0 se produce una solicitud de interrupción.

En el registro OPTION están los bits de control del Timer0. La fuente de los pulsos de reloj se selecciona con el bit T0CS. Si se selecciona una fuente de reloj externa en el terminal T0CKI, el bit T0SE sirve para hacer que el contador se incremente con los flancos de subida (con T0SE = 0) o de bajada (con T0SE = 1) de los pulsos en T0CKI.

El Timer0 y el WDT comparten el pre-divisor, en un esquema como el que se muestra en la figura 6.4. El pre-divisor, un contador asincrónico ascendente de 8 bits, puede ser asignado al Timer0 o al WDT. Esta asignación es excluyente, es decir, si el pre-divisor se asigna al WDT, el Timer0 no lo puede usar y viceversa. Cuando está asignado al WDT, el pre-divisor funciona como post-divisor del Perro Guardián.

El pre-divisor se asigna al Timer0 poniendo a 1 el bit PSA (OPTION<3>). Si PSA es puesto a 0, el pre-divisor queda asignado al WDT. El factor de división del pre-divisor asignado al Timer0 se selecciona con los bits PS2, PS1 y PS0 del registro OPTION.

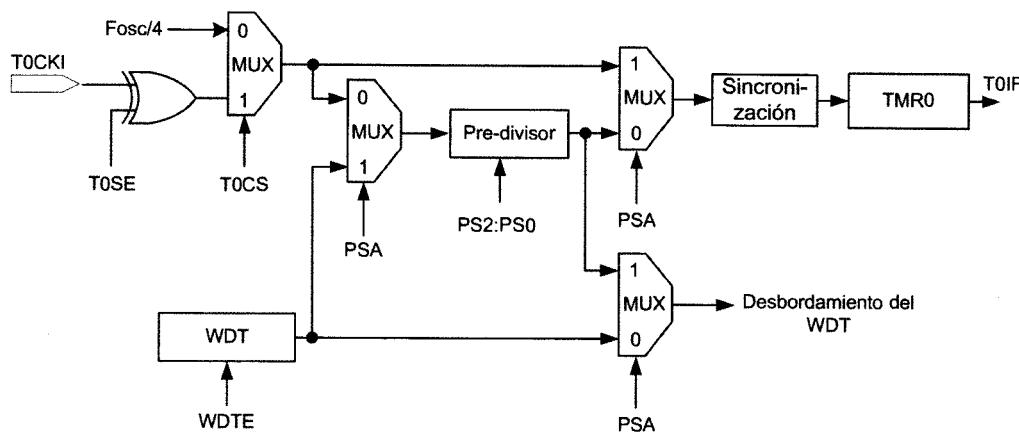


Figura 6.4 El pre-divisor se puede asignar al Timer0 o al Perro Guardián (WDT): con PSA = 0 se asigna al Timer0 y con PSA = 1 al WDT. Con el desbordamiento del Timer0 (cuando éste pasa de FFh a 00), se activa el bit TOIF del registro INTCON. T0SE, T0CS, PSA y PS2:PS0 son bits del registro OPTION. WDTE es uno de los bits de configuración del PIC, con el que se habilita el funcionamiento del WDT.

El factor de división P del pre-divisor asignado al Timer0 puede tomar los valores:

$$P = 2, 4, 8, \dots, 2^{n+1}, \dots, 256 \quad (6.2)$$

donde $n = 0, 1, \dots, 7$ es el valor situado en los bits PS2:PS0.

Si el pre-divisor es asignado al WDT, el factor de división es entonces $P = 1, 2, 4, \dots, 2^n, \dots, 128$ (tabla 2.1).

Para calcular el tiempo de desbordamiento del Timer0 se puede proceder de la forma siguiente. Sea N la cantidad de pulsos que deben llegar al Timer0 para que se desborde, P el factor de división del pre-divisor y T_i el período de los pulsos de entrada al pre-divisor. Si el Timer0 opera como temporizador, entonces T_i es la duración de un ciclo de máquina; si opera como contador, T_i es el período de los pulsos externos. El tiempo T_d que tarda en desbordarse el Timer0 es

$$T_d = P \times N \times T_i \quad (6.3)$$

Debe tenerse en cuenta que el valor que se carga en el registro TMR0 no es N , sino su complemento a 2 en 8 bits, es decir, lo que le falta a N para llegar a 256. Por lo tanto, el valor que se carga en TMR0 debe ser

$$N_{\text{TMR0}} = 256 - N \quad (6.4)$$

Si el Timer0 está trabajando en el modo temporizador, es decir, como contador de ciclos de máquina, entonces hay que tener en cuenta que el conteo se inhibe durante dos ciclos de máquina después de haber escrito un valor en el registro TMR0. Por lo tanto, para compensar esa demora, el valor que hay que depositar en TMR0 en este caso debe ser

$$N_{\text{TMR0}} = 256 - N + 2 \quad (6.5)$$

Obsérvese que si el valor N_{TMR0} se carga en el registro TMR0 cada vez que ocurre el desbordamiento, entonces el Timer0 trabaja como un contador/temporizador con módulo N .

Ejemplo 6.1

Uso del Timer0 para programar demoras. En este caso el Timer0 opera en el modo “temporizador”, pues los pulsos de reloj provienen del oscilador principal del microcontrolador y llegan al temporizador con una frecuencia $F_{\text{osc}}/4$, donde F_{osc} es la frecuencia del oscilador principal del PIC.

El siguiente programa ilustra cómo iniciar (rutina InicTimer0) la operación del Timer0 en este modo, asignándole el pre-divisor con un factor de división de 8. La rutina Dem1ms crea una espera (aproximada) de 1 ms. La rutina DemNms realiza una espera de N milisegundos ($N \leq 255$).

```
; Uso del Timer0 y su pre-divisor para lograr demoras.
;
; Hardware:
; Frecuencia del oscilador del PIC: 4 MHz, por tanto la duración de un ciclo de
; máquina (CM) es  $T_{\text{cm}} = 1 \mu\text{s}$ .
;
; Valores que hay que situar en el pre-divisor y en TMR0 para lograr una demora de 1 ms:
; 1 ms = 1000  $\mu\text{s}$ , pero  $1000 = 8 \times 125$ , y por lo tanto en el pre-divisor se puede
; situar el valor  $P = 8$  y en TMR0 el complemento 2 de 125 más 2.
; Es decir que TMR0 = 256 - 125 + 2 = 133.
```

```
List      p=16F873
include "P16F873.INC"

AUX    equ    0x20          ; Variable auxiliar.

; InicTimer0: Rutina para programar el Timer0 como temporizador con un
;             pre-divisor de 8.
InicTimer0:

    bcf    INTCON, T0IE    ; Inhabilitar interrupción del Timer0.
    bsf    STATUS, RP0     ; Seleccionar banco 1 de la memoria de datos
    movlw  0xC2            ; y configurar el Timer0 como temporizador con un
```

```

movwf OPTION_REG      ; pre-divisor de 8.
bcf STATUS, RP0        ; Seleccionar banco 0 de la memoria de datos.
clrf TMR0              ; Poner 0 en registro TMR0.
bcf INTCON, TOIF       ; Poner en 0 el indicador de desbordamiento.
; bsf INTCON, TOIE     ; Aquí se puede habilitar la interrupción del Timer0
                      ; si se trabaja por interrupción.

return

```

;Dem1ms: Rutina para crear una demora de 1 ms.

; Entradas: ninguna.
; Salidas: ninguna.

Dem1ms:

```

movlw .133            ; Complemento 2 de 125, más 2, valor con
movwf TMR0             ; el que se carga TMR0.

```

Dem1ms_01:

```

btfs  INTCON, TOIF    ; ¿TOIF = 1?
goto Dem1ms_01         ; No - esperar.
bcf   INTCON, TOIF    ; Sí - hacer TOIF = 0 y
return                  ; retornar pues ha transcurrido 1 ms.

```

;Dem500ms: Rutina para crear una demora de N milisegundos (N <= 255)

; Esta rutina llama *N* veces a la rutina Dem1ms.
; Entradas: en W el valor de *N*.
; Salidas: ninguna.

DemNms:

```

movwf AUX               ; Poner en AUX la demora en milisegundos.
DemNms_01:                ; Llamar a Dem1ms N veces:
call Dem1ms             ; Esperar 1 ms.
decfsz AUX, f            ; Decrementar AUX. ¿AUX = 0?
goto DemNms_01           ; No - seguir esperando.
return                   ; Sí - retornar pues han transcurrido N milisegundos.

```

end

6.1.2 El módulo Timer1

El Timer1 es un segundo módulo disponible para temporizar en muchos microcontroladores PIC de clase media. Su estructura se muestra en la figura 6.5.

El Timer1 consta fundamentalmente de un contador ascendente de 16 bits precedido por un pre-divisor programable con factor de división de 1, 2, 4 u 8.

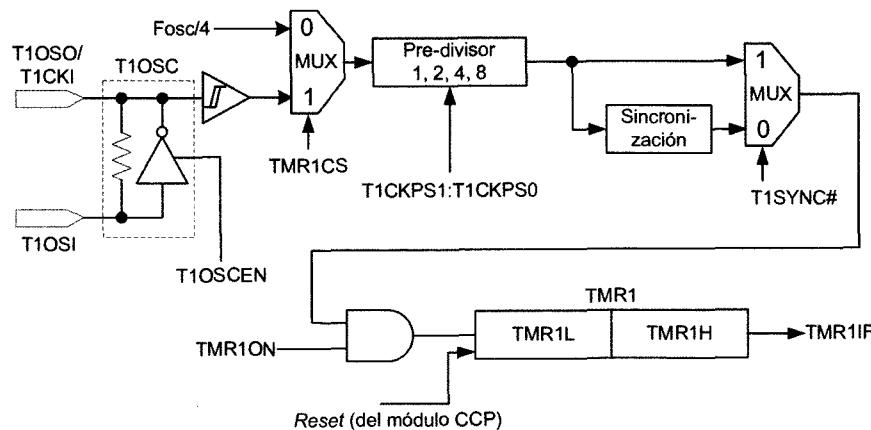


Figura 6.5 Diagrama de bloques del Timer1.

El Timer1 puede operar como temporizador (al contar ciclos de máquina) o como contador (si cuenta pulsos externos). Como contador, el Timer1 puede ser programado para que trabaje en modo sincronizado o en modo no sincronizado. Para ello se usa el bit T1SYNC# del registro T1CON. Si T1SYNC# = 0, el Timer1 trabaja como contador sincronizado, pues los pulsos de entrada al contador TMR1 pasan por el bloque de sincronización. En este bloque se muestrea la señal entrante y se sincroniza con el reloj interno del microcontrolador, de lo que resulta una señal cuyos flancos ocurren en fase con el reloj principal del PIC. La señal ya sincronizada excita el contador de 16 bits TMR1, formado por los registros TMR1L y TMR1H.

Para que no se pierdan pulsos en la sincronización, es necesario que los pulsos que entran en el bloque de sincronización permanezcan en 1 o en 0 al menos durante la mitad del tiempo que dura un ciclo de máquina. Ello determina el valor mínimo del período de los pulsos que entran en el Timer1 en el modo contador sincronizado.

Los modos sincronizado y no sincronizado se aplican sólo al trabajo del Timer1 como contador, pues cuando este módulo opera como temporizador, lo hace obviamente en modo sincronizado. De hecho, el bit T1SYNC# se ignora si el Timer1 se programa como temporizado con TMR1CS = 0.

Si T_{osc} es el período del oscilador principal del microcontrolador y P es el factor de división del pre-divisor ($P = 1, 2, 4, 8$), entonces el período T_i de los pulsos en T1CKI debe cumplir

$$T_i > \frac{4 \times T_{osc}}{P} \quad (6.6)$$

Si el Timer1 está programado como contador no sincronizado ($T1SYNC\# = 1$), continúa operando incluso cuando el microcontrolador esté en el modo de bajo consumo. Esta característica del Timer1 lo hace apropiado para implementar con él un reloj de tiempo real (RTC: *Real Time Clock*).

El valor del conteo se puede leer y/o escribir en los registros de funciones especiales TMR1H y TMR1L. Cuando se escribe en estos registros, la cuenta del pre-divisor va a 0.

En el registro T1CON están los bits de control del Timer1. La fuente de los pulsos de reloj se selecciona con el bit TMR1CS y puede ser interna o externa. Si se selecciona el reloj externo, éste puede ser pulsos que entren por el terminal T1CKI/T1OSO o también puede colocarse un cristal entre los terminales T1OSO y T1OSI. El bit T1OSCEN sirve para habilitar el oscilador si se usa un cristal. Con el bit TMR1ON se habilita el conteo.

El valor n ($n = 0, 1, 2, 3$) de los bits T1CKPS1 y T1CKPS0 del registro T1CON fija el factor de división P ($P = 1, 2, 4, 8 = 2^n$) del pre-divisor del Timer1. Por ejemplo, si se quiere un factor de división de 8, hay que poner el valor 3 en los bits mencionados.

El desbordamiento del Timer1 hace que el bit TMR1IF pase a 1. TMR1IF es un bit del registro PIR1. Debe ser puesto a 0 por software. Si la interrupción del Timer1 está habilitada, lo cual se hace poniendo a 1 el bit TMR1IE ($PIE1<x>$), el desbordamiento genera una interrupción.

T1CON								
7	6	5	4	3	2	1	0	
-	-	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC#	TMR1CS	TMR1ON	

Figura 6.6 T1CON: registro de control del Timer1.

El cálculo del tiempo de desbordamiento del Timer1 sigue un razonamiento semejante al utilizado en el Timer0. Si N es la cantidad de pulsos que deben llegar al contador de 16 bits (TMR1) para que éste se desborde, P el factor de división del pre-divisor y T_i el período de los pulsos de entrada, el tiempo T_d hasta que el Timer1 se desborda es

$$T_d = P \times N \times T_i \quad (6.7)$$

En la pareja de registros TMR1L y TMR1H se carga el complemento 2 de N en 16 bits, es decir,

$$N_{TMR1} = 2^{16} - N = 65536 - N \quad (6.8)$$

Si el Timer1 se deja correr libremente, entonces, de modo natural, $N = 2^{16}$. En este caso,

$$Td = 2^{16} \times P \times Ti \quad (6.9)$$

Ejemplo 6.2

Programación del Timer1. En este ejemplo se ilustra:

- Cómo programar el Timer1 como contador asincrónico de pulsos externos.
- Cómo escribir un número binario de 16 bits en los registros TMR1L y TMR1H.
- Cómo leer de forma segura los registros TMR1L y TMR1H.

```
List      p=16F873
include "P16F873.INC"

;
AUX_H equ 0x20          ; Variable auxiliar
AUX_L equ 0x21          ; Variable auxiliar
;

; InicTimer1: Rutina para programar el Timer1 0 como contador asincrónico
; de pulsos externos con pre-divisor de 8.
InicTimer1:
    clrf T1CON           ; Timer1 como temporizador, detenido, pre-divisor = 1.
    bsf STATUS, RP0        ; Seleccionar banco 1.
    bcf PIE1, TMR1IE       ; Inhabilitar interrupción del Timer1.
    bcf STATUS, RP0        ; Seleccionar banco 0.
    clrf TMR1H             ; Poner 0 en TMR1H.
    clrf TMR1L             ; Poner 0 en TMR1L.
    bcf PIR1, TMR1IF       ; Indicador de desbordamiento en 0.
    movlw 0x36              ; Timer1 como contador no sincronizado con P = 8,
    movwf T1CON             ; T1OSC inhabilitado, Timer1 detenido.
    bsf T1CON, TMR1ON       ; Iniciar el conteo del Timer1.
    return

;
; WR_TMR1a: Rutina para escribir un número binario de 16 bits en el Timer1
; deteniendo temporalmente el conteo del Timer1.
; Entradas: en AUX_H y AUX_L están los bytes alto y bajo
;           del número que se va a escribir en el Timer1.
; Salidas: ninguna.
WR_TMR1a:
    bcf T1CON, TMR1ON ; Detener el conteo del TMR1.
    movf AUX_H, W        ; Cargar el byte alto en
    movwf TMR1H           ; el registro TMR1H.
    movf AUX_L, W        ; Cargar el byte bajo en
    movlw TMR1L           ; en el registro TMR1L.
    bsf T1CON, TMR1ON ; Reiniciar el conteo del TMR1.
```

```

return

; WR_TMR1b: Rutina para escribir un número binario de 16 bits en el Timer1
; sin detener el conteo del Timer1.
; Entradas: en AUX_H y AUX_L están los bytes alto y bajo
; del número a escribir en el Timer1.
; Salidas: ninguna.

WR_TMR1b:
    clrf   TMR1L      ; Asegurar que TMR1L no se desborde mientras
    movf   AUX_H, W    ; se cargar el byte alto en
    movwf  TMR1H      ; el registro TMR1H.
    movf   AUX_L, W    ; Cargar el byte bajo en
    movlw   TMR1L      ; en el registro TMR1L.
    return

; RD_TMR1: Rutina para leer el valor del Timer1 mientras está contando.
; Entradas: ninguna.
; Salidas: en AUX_H se devuelve el valor de TMR1H y en
; AUX_L el valor de TMR1L.

RD_TMR1:
    movf   TMR1H, W    ; Leer TMR1H y poner el byte alto
    movwf  AUX_H        ; en AUX_H.
    movf   TMR1L, W    ; Leer el byte bajo en TMR1L y ponerlo
    movwf  AUX_L        ; en AUX_L.
    movf   TMR1H, W    ; Leer de nuevo TMR1H para ver si cambió.
    xorwf  AUX_H, W    ; Se comparan las dos lecturas.
    btfsc  STATUS, Z   ; ¿Son iguales?
    goto   RD_FIN      ; Sí - lectura válida: terminar la rutina.
    movf   TMR1H, W    ; No - lectura no válida: volver a leer TMR1H
    movwf  AUX_H        ; y TMR1L, de lo que resulta una lectura válida pues
    movf   TMR1L, W    ; no hay tiempo de que TMR1L se desborde y cambie
    movwf  AUX_L        ; el valor de TMR1H.

RD_FIN:
    return              ; Retornar con la lectura correcta.

;
end

```

6.1.3 El módulo Timer2

El Timer2 es un tercer módulo disponible para temporizar en muchos microcontroladores PIC de clase media. Está formado por un contador ascendente de 8 bits, un pre-divisor, un post-divisor y un registro para almacenar el módulo de conteo. El Timer2 sólo trabaja como temporizador y es un contador de ciclos de máquina. La figura 6.7 muestra su diagrama de bloques.

El pre-divisor puede ser programado con factores de división de 1, 4 u 8. Para el post-divisor, los factores de división son 1, 2, 3...16. Cuando se usan ambos divisores en sus valores máximos, el tiempo de desbordamiento del Timer2 es el de un temporizador de 16 bits.

El Timer2 es el temporizador que constituye la base de tiempos del modulador de pulsos en anchura que se constituye cuando el módulo CCP opera en el modo PWM. El Timer2 también puede ser seleccionado como generador de la velocidad de comunicación por el módulo SSP.

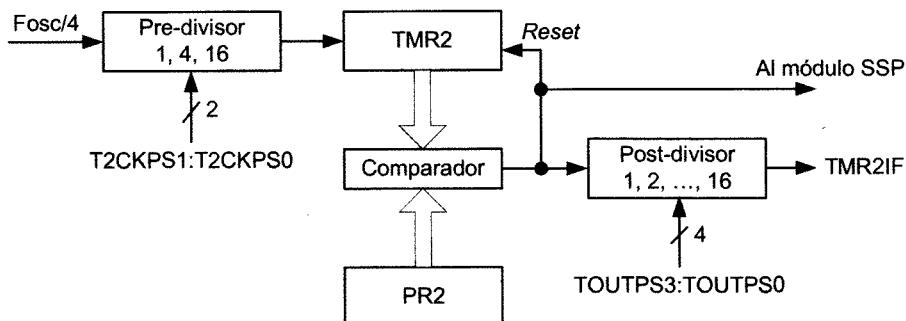


Figura 6.7 Diagrama de bloques del Timer2.

El Timer2 no se incrementa mientras el microcontrolador está en el modo de bajo consumo; el conteo se reanuda cuando el PIC despierta.

La figura 6.8 muestra los bits que componen el registro de control del Timer2, T2CON. Con los bits T2CKPS1 y T2CKPS0 se selecciona el factor de división del pre-divisor, mientras que los bits TOUTPS3:TOUTPS0 seleccionan el factor de división del post-divisor. Los valores que hay que poner en estos bits se muestran en la tabla 6.2. El conteo del Timer2 se puede habilitar poniendo a 1 el bit TMR2ON del registro T2CON y se inhabilita poniéndolo a 0.

T2CON							
7	6	5	4	3	2	1	0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

Figura 6.8 T2CON: registro de control del Timer2. Con los bits T2CKPS1 y T2CKPS0 se selecciona el factor de división del pre-divisor, mientras que los TOUTPS3: TOUTPS0 seleccionan el factor de división del post-divisor. El bit TMR2ON sirve para habilitar el conteo del Timer2.

El valor del conteo se puede leer y/o escribir en el registro TMR2. El valor cambiante del registro TMR2 es comparado continuamente con el valor depositado en PR2. Cuando ambos valores son iguales, el resultado de

la comparación es positivo y el registro TMR2 es puesto a 0 en el siguiente incremento, reiniciándose el proceso, a la vez que se envía un pulso al post-divisor. Si el factor de división del post-divisor es P₂, entonces al cabo de P₂ comparaciones con resultado positivo se desborda el Timer2 y el bit TMR2IF del registro PIR1 pasa a 1. Este bit debe ser puesto a 0 por software. Si el bit TMR2IE del registro PIE1 está en 1, se genera una solicitud de interrupción.

Tabla 6.2 Bits del registro T2CON y factores de división del pre-divisor y el post-divisor. El valor en T2CKPS1 y T2CKPS0 fija el factor de división del pre-divisor, mientras que el valor de TOUTPS3:TOUTPS0 selecciona el factor de división del post-divisor.

T2CKPS1:T2CKPS0	Factor de división del pre-divisor	TOUTPS3:TOUTPS0	Factor de división del post-divisor
00	1	0000	1
01	4	0001	2
1x	16	0010	3
	
		1111	16

El tiempo que tarda el Timer2 en desbordarse se puede calcular de la siguiente forma. Si N es el número depositado en el registro PR2, P₁ el factor de división del pre-divisor, P₂ el factor de división del post-divisor y T_i el período de los pulsos de entrada al módulo, entonces, tras un sencillo razonamiento se concluye que el tiempo T_d que tarda el Timer2 en desbordarse es

$$T_d = P_1 \times P_2 \times (N + 1) \times T_i \quad (6.10)$$

donde P₁ = 1, 4, 16 y P₂ = 1, 2, 3, ..., 16. Teniendo en cuenta que el Timer2 opera como temporizador,

$$T_i = 4 \times T_{osc} \quad (6.11)$$

Ejemplo 6.3

Programar el Timer2 para que se desborde cada milisegundo cuando la frecuencia del oscilador principal del microcontrolador es 4 MHz.

Si la frecuencia del oscilador del microcontrolador es 4 MHz, entonces el período de los pulsos que entran al Timer2 es T_i = 1 μs. Para obtener un tiempo de desbordamiento T_d = 1 ms, hay que hacer que el módulo de conteo del Timer2 sea 1000. Es decir, según (6.10),

$$\frac{T_d}{T_i} = P_1 \times P_2 \times (N + 1) = 1000$$

Este valor se puede obtener con: P₁ = 4, P₂ = 10 y N = 24.

El siguiente segmento de programa ilustra cómo programar el Timer2 con estos parámetros.

```

List          p=16F873
include      "P16F873.INC"
;

; InicTimer2: Rutina que programa al Timer1 para dividir por 1000.

; InicTimer1:
    clrf      T2CON      ; Se detiene el Timer2.
    clrf      TMR2       ; El registro TMR2 es puesto a 0.
    bsf       STATUS, RP0 ; Se selecciona el banco 1.
    bcf       PIE1, TMR2IE ; Se inhabilita la interrupción del Timer2.
    movlw     .24        ; Módulo de conteo de TMR2
    movwf     PR2         ; en PR2.
    bcf       STATUS, RP0 ; Se selecciona el banco 0.
    bcf       PIR1, TMR2IF ; El indicador de desbordamiento es puesto a 0.
    movlw     0x4A       ; Postdivisor = 10, pre-divisor = 16.
    movwf     T2CON      ; Timer2 detenido.
    bsf       T2CON, TMR2ON ;Se inicia el conteo en el Timer2.
    return

;

; Espera_Timer2: Rutina para esperar el desbordamiento del Timer2
; lo cual ocurre cada milisegundo.

; Espera_Timer2:
    btfss    PIR1, TMR2IF   ; ¿Se ha desbordado el Timer2?
    goto    Espera_Timer2  ; No - esperar.
    bcf     PIR1, TMR2IF   ; Sí - poner a 0 la bandera TMR2IF.
    return

end

```

6.2 El módulo CCP

Los módulos de Captura, Comparación y Modulación de Pulso en Anchura (CCP: *Capture/Compare/PWM*) son circuitos que junto con los módulos Timer1 y Timer2 permiten temporizar de otras formas.

En un mismo microcontrolador PIC pueden existir hasta dos módulos CCP, denominados CCP1 y CCP2. Un módulo CCP está formado básicamente por una pareja de registros de 8 bits denominados CCPR_xH y CCPR_xL. Aquí y en lo sucesivo, la letra “*x*” se debe sustituir por 1 ó 2, indicando el módulo CCP al que se esté haciendo referencia. En estos registros se puede almacenar, respectivamente, la parte alta y baja de un número de 16 bits. Cada módulo CCP utiliza también el registro CCP_xCON para el control y el bit

CCPxIF del registro PIR como indicador de que se ha producido un evento. Si la interrupción del módulo está habilitada (con el bit CCPxIE del registro PIE), cuando CCPxIF pasa a 1 se genera una solicitud de interrupción.

Cada módulo CCP puede operar en cualquiera de los siguientes modos:

- Modo de captura. El módulo CCP captura el valor del Timer1 cuando ocurre un evento externo en el terminal CCPx.
- Modo comparador. El registro del módulo CCP almacena un número de 16 bits que se compara con el valor del Timer1 y, según el resultado de la comparación, se genera un evento, que puede incluir un cambio en el terminal CCPx.
- Modo modulador de pulsos en anchura (PWM: *Pulse Width Modulation*). El módulo CCP y el Timer2 forman un modulador de ancho de pulsos con salida por el terminal CCPx.

Los terminales CCPx (CCP1 o CCP2) son entradas en el modo de captura y salidas en los modos comparador y PWM. Estos terminales, uno por cada módulo CCP existente en el PIC, comparten funciones con los terminales del puerto C.

En los modos de captura y comparador, el Timer1 es utilizado por los módulos CCP como base de tiempo. En estos modos, el Timer1 debe ser programado como temporizador o como contador en modo sincronizado. En el modo PWM, el Timer2, que opera siempre como temporizador, determina la frecuencia de la señal PWM.

Dado que los módulos CCP comparten funciones con los temporizadores Timer1 y Timer2, en los PIC que disponen de dos módulos CCP, como sucede en el PIC16F873, hay que tener en cuenta el uso compartido de estos temporizadores por ambos módulos. La tabla 6.3 muestra las posibles interacciones que hay que considerar al programar los módulos CCP.

Tabla 6.3 Interacción entre los módulos CCP.

Modo del módulo CCPx	Modo del módulo CCPy	Interacción
Captura	Captura	Ambos módulos utilizan la misma base de tiempo (Timer1)
Captura	Comparador	El módulo CCP que opera como comparador debe configurarse para la puesta a 0 del Timer1 cuando el resultado de la comparación es positivo.
Comparador	Comparador	Los comparadores deben configurarse para la puesta a 0 del Timer1 cuando el resultado de la comparación es positivo.
PWM	PWM	Ambas señales PWM tienen el mismo período, dado por el valor del registro PR2.

PWM	Captura	No hay interacción.
PWM	Comparador	No hay interacción.

Los registros de funciones especiales CCPxCON se utilizan para programar los módulos CCP. La figura 6.9 muestra los bits de estos registros. El modo de trabajo de los módulos CCP se programa mediante los bits CCPxM3: CCPxM0. Los bits DCxB1 y DCxB0 se usan sólo en el modo PWM.

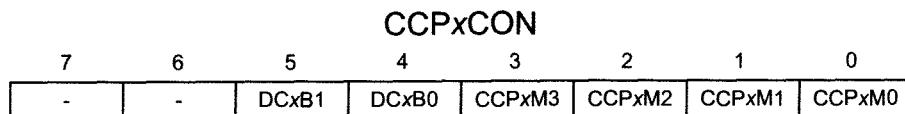


Figura 6.9 Registros CCPxCON, para el control de los módulos CCP.

La tabla 6.4 muestra los valores que hay que poner en los bits CCPxM3: CCPxM0 de los registros CCPxCON para programar los diferentes modos de trabajo de los módulos CCP, que se explican en los apartados siguientes.

Tabla 6.4 Valores de los bits CCPxM3:CCPxM0 del registro CCPxCON y modos de operación de los módulos CCP.

Bits CCPxM3 ...CCPxM0		Modo	
00	00	Desactivar módulo CCP	
01	00	Captura	Cada flanco de bajada
	01		Cada flanco de subida
	10		Cada 4 flancos de subida
	11		Cada 16 flancos de subida
	10	Comparador	El terminal CCPx es iniciado en bajo y se pone en alto cuando el resultado de la comparación es positivo. El bit CCPxF es puesto a 1.
	01		El terminal CCP es iniciado en alto y se pone en bajo cuando el resultado de la comparación es positivo. El bit CCPxF es puesto a 1.
	10		El bit CCPIF es puesto a 1 cuando el resultado de la comparación es positivo. No se afecta el terminal CCPx.
	11		El Timer1 es puesto a 0 (reset al Timer1). El resultado de la comparación es positivo. El bit CCPxF es puesto a 1. No se afecta el terminal CCPx.
11	xx	PWM	

6.2.1 Modo de captura

La figura 6.10 muestra el esquema de bloques del módulo CCP cuando trabaja en el modo de captura. El valor que tiene el Timer1 en el momento de producirse un cierto evento relacionado con el terminal CCPx del módulo,

es almacenado en los registros CCPxRH y CCPxRL. La captura del valor del Timer1 se puede programar para que se realice con cada flanco de subida o de bajada de los pulsos de entrada al terminal CCPx, o con los flancos de subida cada 4 ó 16 pulsos. La programación de la captura se hace con los bits CCPxM3:CCPxM0, tal como se ilustra en la tabla 6.4.

Al ocurrir la captura, el bit CCPxIF es puesto a 1, de modo que actúa como un indicador de la ocurrencia del evento. Este indicador puede ser consultado por programa. Además, si la interrupción del módulo CCP está habilitada (el bit CCPxIE del registro PIE está en 1), se genera una solicitud de interrupción.

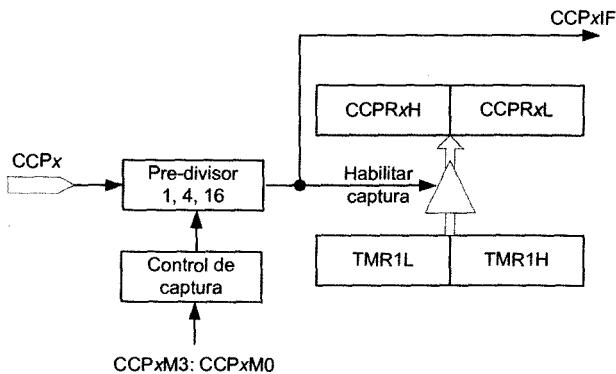


Figura 6.10 Módulo CCP operando como registro de captura del Timer1 ante un evento en el terminal CCPx.

El terminal CCPx debe ser configurado como entrada poniendo a 1 el bit correspondiente del registro TRIS del puerto paralelo donde se encuentra el CCPx, que normalmente es el puerto C.

Ejemplo 6.4

El modo de captura en un módulo CCP se puede utilizar para medir tiempos. En este ejemplo se ilustra cómo medir el período de un tren de pulsos periódicos utilizando el módulo CCP1 de un PIC16F873. La frecuencia del oscilador principal es de 4 MHz.

La figura 6.11 muestra el esquema simplificado que se utiliza, el cual está compuesto por el módulo CCP1 en modo de captura y el Timer1. Como elementos esenciales están el terminal CCP1 de entrada de la señal cuyo período T_x se quiere determinar; los registros CCPR1 (formado por la pareja CCPR1H y CCPR1L) y TMR1 (formado por los registros TMR1H y TMR1L) y los pre-divisores del módulo CCP y del Timer1, con factores de división P_c y P_1 , respectivamente.

Para el correcto funcionamiento de la captura, el Timer1 debe ser programado como temporizador, para que actúe como un contador de ciclos de máquina. La duración de un ciclo de máquina es T_{CM} .

Del esquema de la figura 6.11 se deduce que la captura del valor cambiante del Timer1 ocurre cada $P_c \times T_x$ segundos. Si N_1 y N_2 son los valores de TMR1 en dos capturas sucesivas, entonces, entre una captura

y otra, la cuenta del Timer1 ha avanzado en $N = N_2 - N_1$. En unidades de tiempo, eso significa que han transcurrido $N \times P_1 \times T_{CM}$ segundos entre una captura y otra. Entonces:

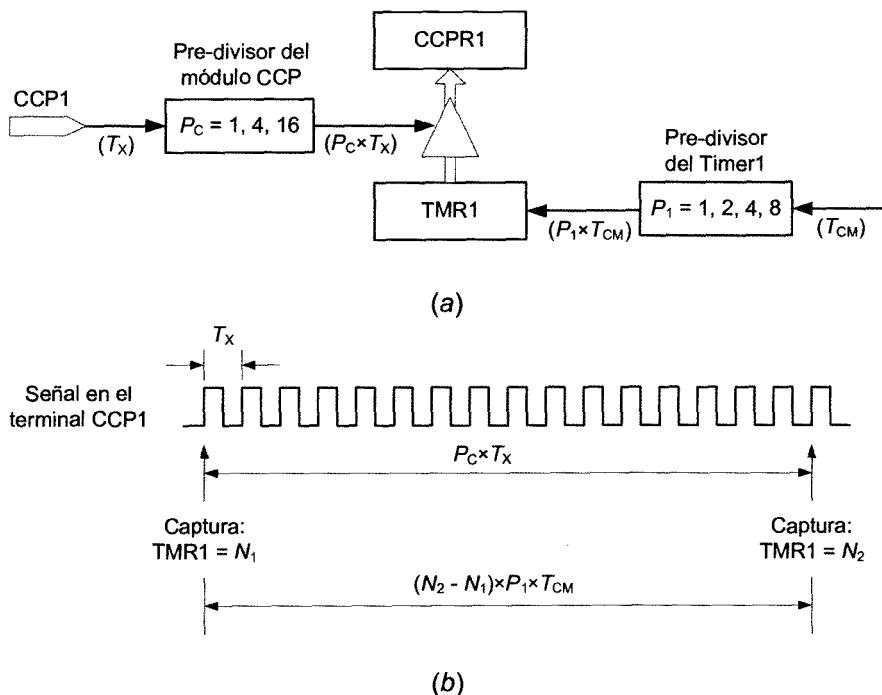


Figura 6.11 El módulo CCP en modo de captura. (a) Esquema simplificado del módulo CCP en modo de captura junto al Timer1. (b) Diagrama del tiempo transcurrido entre dos capturas sucesivas, en función del período T_x del tren de pulsos en el terminal CCP1, la duración T_{CM} de los ciclos de máquina y los factores de división de los pre-divisores.

$$P_c \times T_x = N \times P_1 \times T_{CM}$$

de donde se obtiene

$$T_x = N \times \frac{T_{CM}}{P_c / P_1}$$

El cociente P_c / P_1 de los factores de división de los pre-divisores, permite ajustar la resolución con que se determina el valor de T_x , pues en un microcontrolador la duración de los ciclos de máquina está fijada de antemano. Sabiendo que $P_c = 1, 4, 16$ y $P_1 = 1, 2, 4, 8$ la mejor resolución se obtiene con $P_c = 16$ y $P_1 = 1$. Con estos valores se puede medir el período T_x con una resolución de 1/16 de la duración de un ciclo de máquina. No obstante, para que esta afirmación sea cierta, es necesario que la señal de entrada mantenga su período estable, sin cambios, al menos durante los 16 períodos que dura la medición.

Por otra parte, si $P_c = P_1$, N representa la cantidad de ciclos de máquina que dura el período T_x de la señal. Si $T_{CM} = 1 \mu s$, N es directamente la duración del período T_x en microsegundos. Evidentemente,

en este caso, como N es un número de 16 bits, el mayor período que se puede medir con este esquema es de 65536 μs . Dejamos como ejercicio para el lector que determine cuál es el período máximo que se puede medir si $P_c/P_1 = 16$.

El listado que sigue ilustra cómo programar el Timer1 y el módulo CCP1 en modo de captura, para medir el período de los pulsos en el terminal CCP1 de un PIC16F873. La frecuencia del oscilador principal del PIC es 4 MHz y los factores de división seleccionados son $P_c = P_1 = 1$, de modo que la diferencia entre dos capturas sucesivas es el período buscado, en microsegundos.

```
List      p=16F873
include  "P16F873.INC"

N1H    equ    20h      ; Parte alta de la primera captura.
N1L    equ    21h      ; Parte baja de la primera captura.
NH     equ    22h      ; Parte alta de la diferencia.
NL     equ    23h      ; Parte baja de la diferencia.

; Inic_captura: Rutina para programar el módulo CCP1 en modo de captura
; con cada flanco de subida de los pulsos en el terminal CCP1. El Timer1
; se programa como temporizador, con pre-divisor = 1.

Inic_captura:
    clrf   T1CON      ; Timer1 como temporizador, pre-divisor=1, detenido.
    clrf   CCP1CON    ; Reset al módulo CCP1.
    bsf    STATUS, RP0 ; Seleccionar banco 1.
    bsf    TRISC, 2   ; Poner el terminal CCP1 como entrada.
    bcf    PIE1, TMR1IE ; Inhabilitar interrupción del Timer1.
    bcf    PIE1, CCP1IE ; Inhabilitar interrupción del módulo CCP1.
    bcf    STATUS, RP0 ; Seleccionar banco 0.
    clrf   PIR1        ; Poner a 0 las banderas de interrupción.
    movlw  0x05        ; Seleccionar modo captura, con cada flanco de subida.
    movwf  CCP1CON    ;
    bsf    T1CON, TMR1ON ; Iniciar conteo del Timer1.
    return

; Captura: Esta rutina captura dos valores del Timer1 y calcula la diferencia
; que hay entre ellos. La diferencia, que es un número de 16 bits, es devuelta
; en los registros NH (la parte alta) y NL (la parte baja).

Captura:
    bcf    PIR1, CCP1IF ; Poner a 0 el indicador de captura.
    btfss  PIR1, CCP1IF ; ¿CCP1IF = 1?
    goto   Captura     ; No - esperar.
    bcf    PIR1, CCP1IF ; Sí - poner a 0 el indicador de captura y
    movf   CCPR1L, W    ; guardar el valor capturado en N1H y N1L.
    movwf  N1L
    movf   CCPR1H, W
    movwf  N1H
```

; Captura: Esta rutina captura dos valores del Timer1 y calcula la diferencia
; que hay entre ellos. La diferencia, que es un número de 16 bits, es devuelta
; en los registros NH (la parte alta) y NL (la parte baja).

Captura:

```
    bcf    PIR1, CCP1IF ; Poner a 0 el indicador de captura.
    btfss  PIR1, CCP1IF ; ¿CCP1IF = 1?
    goto   Captura     ; No - esperar.
    bcf    PIR1, CCP1IF ; Sí - poner a 0 el indicador de captura y
    movf   CCPR1L, W    ; guardar el valor capturado en N1H y N1L.
    movwf  N1L
    movf   CCPR1H, W
    movwf  N1H
```

```

Captura2:          ; Capturar el siguiente valor:
    btfss  PIR1, CCP1IF   ; ¿CCP1IF = 1?
    goto   Captura2      ; No - esperar.
    bcf    PIR1, CCP1IF   ; Sí - poner a 0 el indicador de captura y
                          ; restar los valores capturados.

; Ahora se realiza la operación CCPR1 - N1 ==> N:
    movf  N1L, W
    subwf CCPR1L, W
    movwf NL
    btfss  STATUS, C
    goto   Resta1
    goto   Resta0

Resta1:
    decf  CCPR1H, f

Resta0:
    movf  N1H, W
    subwf CCPR1H, W
    movwf NH
    return

end

```

6.2.2 Modo comparador

La figura 6.12 muestra el esquema de bloques del módulo CCP operando en el modo comparador. En este modo se está comparando el valor cambiante del Timer1 con el valor almacenado en los registros CCPxRH y CCPxRL; cuando ambos valores son iguales, el resultado de la comparación es positivo y se genera un determinado evento, que depende de lo que haya sido programado en los bits CCPxM3:CCPxM0 del registro CCPxCON, según se muestra en la tabla 6.4.

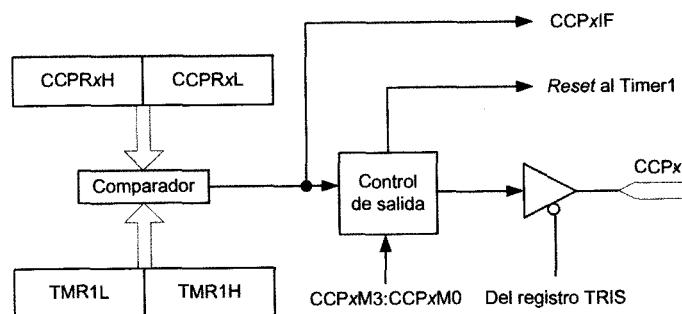


Figura 6.12 Módulo CCP como comparador. Si el valor en el registro CCPRx es igual al valor en el par TMR1, se genera un evento por el terminal CCPx. Las características del evento se programan con los bits CCPxM3:CCPxM0 del registro CCPxCON.

Los eventos que se pueden programar consisten en la puesta a 0 o a 1 del terminal CCP_x y el *reset* del Timer1. Si se usa el terminal CCP_x, entonces debe ser configurado como salida poniendo a 0 el bit correspondiente del registro TRIS del puerto paralelo donde se encuentra el terminal CCP_x, que normalmente es el puerto C. En todos los casos, cuando el resultado de la comparación es positivo, el bit CCP_xIF del registro PIR es puesto a 1. Este bit puede ser consultado por programa. Si la interrupción del módulo CCP está habilitada (mediante la puesta a 1 del bit CCP_xIE del registro PIE), se genera una solicitud de interrupción.

Uno de los eventos que se puede generar cuando el resultado de la comparación es positivo es la puesta a 0 (*reset*) del Timer1. Esta opción aumenta las posibilidades de ese temporizador, ya que entonces puede trabajar como un temporizador de 16 bits cuyo módulo de conteo es el valor de los registros CCPR_xH y CCPR_xL del módulo CCP.

Ejemplo 6.5

Uso del Timer1 como temporizador de 16 bits empleando los registros CCPR1H y CCPR1L para almacenar el módulo de conteo. Para ello se programa el módulo CCP1 en modo comparador, usando la variante de generar un *reset* al Timer1 cuando la comparación entre los registros CCPR1 (CCPR1H:CCPR1L) y TMR1 (TMR1H:TMR1L) sea positiva. En este ejemplo, el Timer1 se programa como temporizador, con un factor de división de 1 para su pre-divisor.

A continuación se da el listado de la programación en ensamblador.

```

List      p=16F873
include  "P16F873.INC"
;
; Inic_compara: Rutina para programar el módulo CCP1 en modo comparador
; con el reset del Timer1 al ser positiva la comparación. No se utiliza el
; terminal CCP1. El Timer1 se programa como temporizador, con pre-divisor = 1.
Inic_compara:
    clrf    T1CON        ; Timer1 como temporizador, pre-divisor=1, detenido.
    clrf    CCP1CON      ; Reset al módulo CCP1.
    bsf     STATUS, RP0   ; Seleccionar banco 1.
    bcf     PIE1, TMR1IE  ; Inhabilitar interrupción del Timer1.
    bcf     PIE1, CCP1IE  ; Inhabilitar interrupción del módulo CCP1.
    bcf     STATUS, RP0   ; Seleccionar banco 0.
    clrf    PIR1          ; Poner a 0 las banderas de interrupción.
    movlw  0x0B          ; Seleccionar modo comparador, con reset al Timer1.
    movwf  CCP1CON      ;
    bsf     T1CON, TMR1ON ; Iniciar conteo del Timer1.
    return
;
; Compara: Esta rutina espera a que la comparación hecha en el módulo CCP1

```

; programado como comparador, sea positiva. En este modo se comparan los registros ; CCPR1 (CCPR1H y CCPR1L) y TMR1 (TMR1H y TMR1L). Cuando CCPR1 = TMR1, ; el resultado de la comparación es positivo y termina la rutina.
; Entradas: En CCPR1H y CCPR1L, el módulo de conteo del Timer1.

Compara:

```

btfs  PIR1, CCP1IF    ; ¿CCPR1 = TMR1?
goto Compara      ; No - esperar.
bcf   PIR1, CCP1IF    ; Sí - poner a 0 el indicador de comparación.
return          ; Retornar.

end

```

6.2.3 Modo PWM

En el modo PWM (*Pulse Width Modulation*), cada módulo CCP y el Timer2 forman un modulador de pulsos en anchura con salida por el terminal CCP_x del módulo.

Una señal PWM es un tren de pulsos de duración T_{ON} variable y período T constante (figura 6.13).

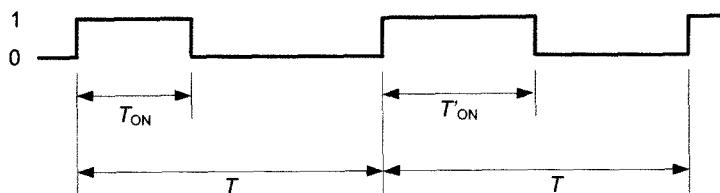


Figura 6.13 Señal PWM. La duración de los pulsos es variable (T_{ON}, T') pero el período T es constante.

Para caracterizar un tren de pulsos periódicos se utilizan los términos *ciclo de trabajo, ciclo activo, o ciclo útil (duty cycle)*, que se define como el porcentaje del período durante el cual la señal permanece en 1, es decir,

$$\text{Ciclo de trabajo} \% = \frac{T_{ON}}{T} \times 100 \quad (6.12)$$

Así, por ejemplo, si los pulsos están la mitad del tiempo en 1, el ciclo de trabajo es del 50 %; si los pulsos están todo el tiempo en 1, el ciclo de trabajo es del 100 %. En una señal PWM, el ciclo de trabajo varía de acuerdo con la señal moduladora.

Otro parámetro utilizado para caracterizar una señal PWM es la *resolución*. En los sistemas digitales, el tiempo es normalmente una variable discreta, de modo que en una señal PWM generada por un sistema digital, la duración T_{ON} de los pulsos no varía continuamente, sino a saltos, en intervalos discretos de tiempo ΔT_{ON} . Si este intervalo es un submúltiplo del período T de la señal, la duración T_{ON} de los pulsos en una señal PWM puede tomar sólo los siguientes valores discretos:

$$T_{ON} = 0, \Delta T_{ON}, 2\Delta T_{ON}, 3\Delta T_{ON}, \dots, T \quad (6.13)$$

La cantidad R de veces en que se puede subdividir el período T define la resolución de la señal PWM,

$$R = \frac{T}{\Delta T_{ON}} \quad (6.14)$$

Si este cociente es una potencia de 2, es decir, $R = 2^r$, entonces r es la resolución expresada en bits,

$$r = \frac{\lg R}{\lg 2} \quad (6.15)$$

La resolución r de una señal PWM da la cantidad de bits necesarios para determinar cualquier posible duración T_{ON} de la señal. Es una medida de la cantidad de partes en que se puede dividir la duración de los pulsos PWM. Por ejemplo, si $R = 1024$, la resolución de la señal PWM es de 10 bits. Esto significa que la duración T_{ON} de los pulsos PWM se puede variar a saltos, con un paso de T/R unidades de tiempo.

En un módulo CCP programado como modulador de ancho de pulsos, el período T de la señal PWM está determinado por el valor del registro PR2 del Timer2. Dentro de cada período, el tiempo T_{ON} durante el cual la señal PWM permanece en 1, está determinado por los bits DCxB9:DCxB0. Estos bits se distribuyen entre los registros CCPRxL y CCPxCON de la siguiente forma: del registro CCPRxL se toman sus 8 bits, para formar los 8 bits de mayor peso, nombrados DCxB9:DCxB2; del registro CCPxCON se toman los bits CCPxCON<5:6>, que entonces constituyen los dos bits de menor peso, nombrados DCxB1:DCxB0.

La figura 6.14 muestra la configuración que adquieren los módulos CCP junto al Timer2 en el modo PWM. El funcionamiento es el siguiente. Se compara el valor almacenado en el registro PR2 con el valor cambiante de TMR2. Cuando ambos valores son iguales, significa que ha transcurrido el tiempo

correspondiente a un período T de la señal PWM. Entonces se realizan varias acciones:

- El registro TMR2 es puesto a 0 en el siguiente ciclo de máquina.
- La entrada S del biestable RS se activa y con ello su salida Q pasa a 1. Por lo tanto, si el terminal CCPx está programado como salida mediante el bit correspondiente del registro TRIS, pasa también a 1.
- El valor situado en los bits DCxB9:DCxB0 se carga en CCPRxH y en dos bits internos del módulo. Este valor es proporcional a la duración T_{ON} de los pulsos de la señal PWM.

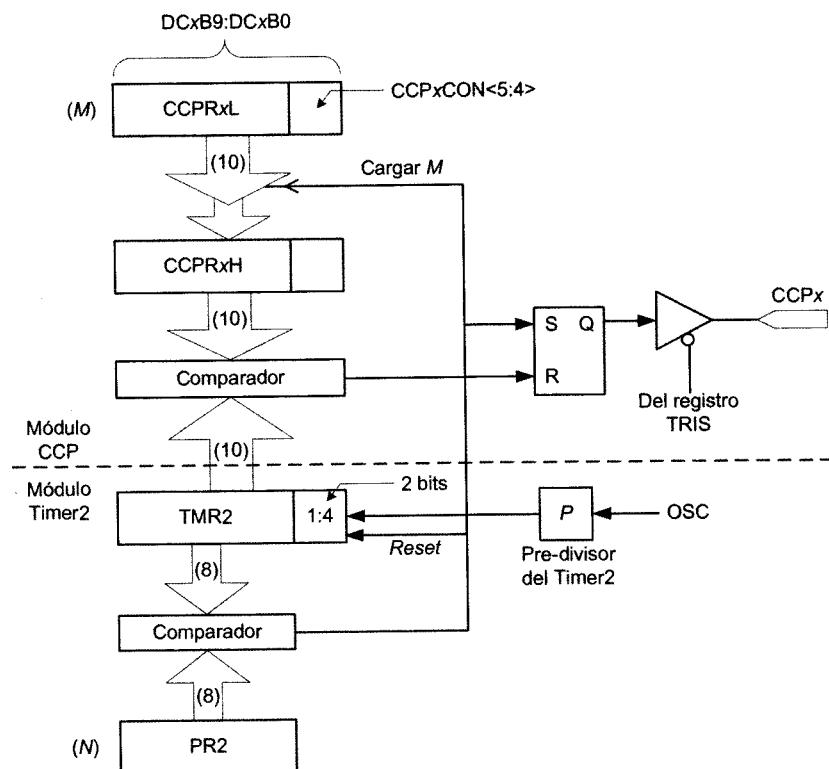


Figura 6.14 Módulo CCP trabajando como modulador de pulsos en anchura. La señal PWM se puede obtener en el terminal CCPx del módulo. El período de la señal queda determinado por el valor N depositado en el registro PR2 del Timer2 y la duración de los pulsos la define el valor M formado por los bits DCxB9:DCxB0.

Realizadas estas acciones, se inicia un nuevo período de la señal PWM. El valor DCxB9:DCxB0, proporcional al ciclo de trabajo de la señal PWM, es comparado con el valor cambiante de TMR2 (más dos bits internos). Cuando

ambos valores son iguales, la entrada R del biestable RS se activa y con ello la salida Q pasa a 0. Si el terminal CCPx ha sido programado como salida, pasa también a 0. La señal PWM resultante y sus parámetros se muestran en la figura 6.15.

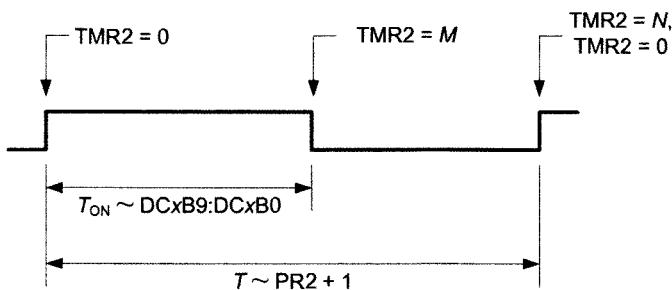


Figura 6.15 Parámetros de la señal PWM generada por el módulo CCP. El ciclo de trabajo de la señal PWM está determinado por el valor M de los bits DCxB9:DCxB0, mientras que el período de la señal lo determina el valor N almacenado en PR2. El registro TMR2 se incrementa, en cada ciclo de máquina, desde 0 a N ; al alcanzar este valor, es forzado a 0 en el ciclo de máquina siguiente, con lo que se inicia un nuevo período de la señal PWM. La señal inicia su ciclo de trabajo cuando el TMR2 es puesto a 0, y finaliza cuando TMR2 alcanza el valor M .

El período (T) y la duración (T_{ON}) de los pulsos en la señal PWM se pueden calcular mediante

$$T = (N + 1) \times P \times 4 \times T_{OSC} \quad (6.16a)$$

$$T_{ON} = M \times P \times T_{OSC} \quad (6.16b)$$

donde N es el valor depositado en el registro PR2, M es el número formado por los bits DCxB9:DCxB0, $P = 1, 4, 16$ es el factor de división del pre-divisor del Timer2 y T_{OSC} es el período del oscilador principal del microcontrolador. Dado que M es un número de 10 bits, se puede expresar como

$$M = 4 \times M_8 + M_2 \quad (6.17)$$

donde M_8 es el número de 8 bits almacenado en CCPRxL y M_2 es el número de 2 bits almacenado en CCPxCON<5:4>. Si, por comodidad, para variar la duración de los pulsos de la señal PWM se quiere manipular sólo el registro CCPRxL, entonces se pueden mantener los bits M_2 en 0. De esta forma, si $M_2 = 0$, y teniendo en cuenta (6.16b) y (6.17), la duración de los pulsos es

$$T_{ON} = 4 \times M_8 \times P \times T_{OSC} \quad (6.18)$$

Para determinar la resolución de la señal PWM obtenida, obsérvese que el tiempo de duración de los pulsos PWM se puede dividir en M partes. Dado que M es un número de 10 bits, se podría pensar que la resolución del mo-

dulador PWM es de 10 bits. Sin embargo, la resolución real es menor pues el mayor valor que puede alcanzar M está limitado por el valor N depositado en PR2. Por ello, la resolución queda determinada realmente por N . Para calcular la resolución del módulo PWM se procede así. El valor más pequeño de T_{ON} que se puede obtener es

$$\Delta T_{ON} = P \times T_{OSC} \quad (6.19)$$

Por tanto, la cantidad de veces que este valor cabe en el período T es

$$R = \frac{(N+1) \times P \times 4 \times T_{OSC}}{P \times T_{OSC}} = 4 \times (N+1) \quad (6.20)$$

Como $(N+1)$ es un número de n bits ($n = 1, 2, \dots, 8$), teniendo en cuenta (6.15), la resolución en bits que se puede alcanzar es

$$r = n + 2 \quad (6.21)$$

Ejemplo 6.6

Programación del módulo CCP1 de un PIC16F873 para generar una señal PWM. Se desea una señal PWM con un período constante de 1 ms y ciclo de trabajo variable. El oscilador principal del microcontrolador es de 4 MHz ($T_{OSC} = 0,25 \mu s$).

El período T de la señal PWM se fija mediante el valor N depositado en el registro PR2. La duración T_{ON} de los pulsos queda determinada por el valor M de los 10 bits distribuidos entre los 8 bits del registro CCPR1L (que constituyen los 8 bits más significativos de M , es decir, M_8) y los 2 bits CCP1CON<5:4>, que forman los 2 bits menos significativos de M , es decir, M_2 . En este ejemplo, se quiere utilizar sólo el registro CCPR1L para fijar el valor de T_{ON} . Los bits CCP1CON<5:4> se mantendrán permanentemente en 0.

Teniendo en cuenta que $T_{OSC} = 0,25 \mu s$ y $T = 1 \text{ ms}$, en (6.16a) se obtiene que

$$(N+1) \times P = 1000$$

donde P es el factor de división del pre-divisor del Timer2, $P = 1, 4, 16$. Se escoge $P = 4$, de modo que el valor que hay que colocar en el registro PR2 es $N = 249$.

La duración (variable) de los pulsos PWM se logra colocando un valor entre 0 y 249 en el registro CCPR1L.

A continuación se muestra el listado del programa en lenguaje ensamblador.

```
List      p=16F873
include "P16F873.INC"
;
; Inic_pwm: Rutina para programar el módulo CCP1 en modo PWM
; para generar una señal PWM de período 1ms y ciclo de trabajo del 50 %.
; El valor del ciclo de trabajo se puede variar dinámicamente con posterioridad.
```

```
Inic_pwm:  
    movlw 0x01          ; Programar Timer2 detenido  
    movwf T2CON         ; y pre-divisor = 4.  
    clrf CCP1CON       ; Reset al módulo CCP1.  
    clrf TMR2          ; Poner a 0 el Timer2.  
    movlw .124          ; La señal PWM saldrá con  
    movwf CCPR1L        ; ciclo de trabajo del 50 %.  
    bsf STATUS, RP0     ; Seleccionar banco 1.  
    movlw .249          ; Módulo de conteo del Timer2  
    movwf PR2           ; que es el período de la señal PWM.  
    bcf PIE1, TMR2IE   ; Inhabilitar interrupción del Timer2.  
    bcf PIE1, CCP1IE   ; Inhabilitar interrupción del módulo CCP1.  
    bcf TRISC, 2        ; Poner el terminal CCP1 como salida.  
    bcf STATUS, RP0     ; Seleccionar el banco 0.  
    clrf PIR1           ; Poner a 0 las banderas de interrupción.  
    movlw 0x0C           ; El módulo CCP1 en modo PWM,  
    movwf CCP1CON        ; con los bits DC1B1:DC1B0 en 0 (M2 = 0).  
    bsf T2CON, TMR2ON   ; Iniciar conteo del Timer2.  
    return  
  
;  
; TON_pwm: Esta rutina varía el ciclo útil de la señal PWM  
; según el valor depositado en W.  
; Entradas: En W un número entre 0 y 249.  
TON_pwm:  
    movwf CCPR1L  
    return  
  
end
```

En un microcontrolador hay varias fuentes de interrupción, unas internas y otras externas. Las interrupciones internas tienen su origen en los módulos de entrada y salida del microcontrolador, la memoria o la CPU. Los temporizadores y otros módulos de entrada y salida son fuentes de interrupción comunes. Menos comunes son las interrupciones causadas por algún evento que tenga lugar en la memoria (por ejemplo, por escribir en la EEPROM de datos) o en la propia CPU (una división por cero). Las interrupciones externas se originan en un periférico y llegan al microcontrolador por alguno de sus terminales.

Los microcontroladores tienen recursos para recibir y procesar las solicitudes de interrupción. Generalmente, cada dispositivo que es fuente de una posible interrupción tiene asociados dos bits, que pueden estar en un mismo registro o en registros diferentes. El primer bit tiene una función informativa: es un indicador que es activado (es puesto a 1, por ejemplo) por el dispositivo que solicita la interrupción. Este bit se puede consultar por programa si para atender al dispositivo se usa la técnica de consulta o espera. El otro bit tiene una función de control y se usa para permitir o impedir el paso de la solicitud de interrupción hacia la CPU, lo que equivale a habilitar o inhabilitar la generación de interrupciones por la fuente en cuestión. Este bit de control se puede manejar por programa.

Los microcontroladores disponen además de un bit para el control global del sistema de interrupción. Con este bit se permite o impide el paso de cualquier interrupción hacia la CPU, lo cual equivale a habilitar o inhabilitar globalmente el sistema de interrupción. Para que una solicitud de interrupción llegue a la CPU y sea atendida, tanto el sistema en su conjunto como la interrupción en particular deben estar habilitados.

Los bits de control utilizados para permitir o no el paso de las solicitudes de interrupción hacia la CPU se denominan *máscaras*; de ahí que las interrupciones que se pueden habilitar o inhabilitar por programa se llamen *interrupciones enmascarables (maskable interrupts)* y las interrupciones que no se pueden inhabilitar por programa (es decir, que están siempre habilitadas), si las hay, se denominan *interrupciones no enmascarables (non-maskable interrupts)*.

La figura 7.2 ilustra el camino que siguen las solicitudes de interrupción, enmascarables o no, para llegar a la CPU. Las interrupciones enmascarables disponen de bits de control asociados a cada fuente de interrupción y del bit para el control global del sistema. Para que una solicitud de interrupción enmascarable progrese hacia la CPU, tanto el bit de control individual corres-

pondiente como el bit de control global deben estar en 1. Si la interrupción es no enmascarable, llegará a la CPU con independencia del bit de control global, y será atendida.

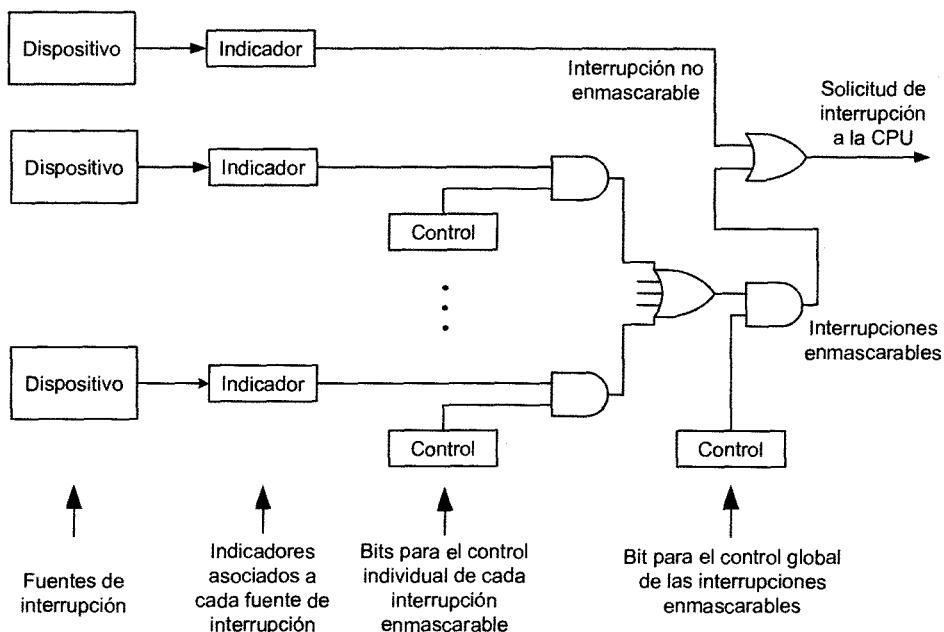


Figura 7.2 Camino que siguen las solicitudes de interrupción en un microcontrolador. Cada fuente de interrupción tiene un indicador que se activa (pasa a 1) con la solicitud de interrupción. Las interrupciones enmascarables deben ser habilitadas para que las solicitudes lleguen a la CPU. Para ello están los bits de control individual y global correspondientes. Una solicitud de interrupción no enmascarable siempre llega a la CPU para ser atendida.

En general, cuando una solicitud de interrupción que llega a la CPU es atendida, el sistema de interrupción queda inhabilitado (el bit de control global es puesto a 0). Entonces, según el esquema de la figura 7.2, no llegará a la CPU ninguna nueva solicitud de interrupción que se produzca. Para poder atender otras solicitudes de interrupción, el programador debe habilitar nuevamente el sistema; esto suele hacerlo el propio programa que atiende la interrupción.

7.1.2 Atención a las solicitudes de interrupción

Atender una solicitud de interrupción es interrumpir la ejecución del programa en curso y pasar a ejecutar otro programa, tal como ilustra la figura 7.1. Cuando termina este segundo programa, hay que continuar con el programa interrumpido.

La solicitud de interrupción que llega a la CPU (en el supuesto de que la interrupción y el sistema están habilitados), se atiende cuando termina la ejecución de la instrucción en curso. Como en general no se conoce de antemano cuál es esta instrucción, hay que encontrar la forma de recordar la dirección de la instrucción que le sigue, para regresar a ella cuando termine el programa que atiende a la interrupción. Esa dirección está en el contador de programa (PC). La forma de recordarla es guardar el contenido del PC en la pila, tal como lo hacen las instrucciones de llamada a una subrutina. Conviene por ello que el programa de atención a una interrupción tenga la estructura de una subrutina, porque la instrucción de retorno que pone fin a la ejecución de esta subrutina hará que se regrese satisfactoriamente al programa interrumpido.

El programa de atención a una interrupción es, pues, una subrutina que se “llama” por interrupción. También se puede decir que una solicitud de interrupción equivale a insertar una instrucción de llamada a una subrutina (la que atiende la interrupción) en algún lugar no previsible del programa.

En principio, la estructura de la subrutina que atiende a una interrupción no es diferente de la de cualquier subrutina “convencional”. No obstante, dado que no es posible conocer de antemano el lugar del programa desde donde se producirá la llamada a la subrutina, el programador debe tomar ciertas precauciones al escribirla. La ejecución de la subrutina que atiende una interrupción debe dejar intactos los valores de los registros y bits con los que el programa estaba trabajando. Por ejemplo, los valores que tienen el acumulador (o el registro de trabajo) y el registro de estado u otro similar, como los indicadores aritméticos, no deben alterarse por la ejecución de la subrutina de atención a una interrupción. Para preservar esos y otros registros que no deban ser alterados, hay que guardar sus valores al iniciar la subrutina. En muchos microcontroladores se usa la pila para esta acción. Antes de retornar al programa interrumpido, se restituyen los valores originales de esos registros. Hay que desactivar el indicador de la interrupción (es decir, ponerlo a 0) para que no se repita continuamente, pues, por lo general, el indicador no se desactiva automáticamente, y sólo entonces habilitar el sistema de interrupción para retornar al programa interrumpido. La figura 7.3 muestra el diagrama de bloques de la estructura general de la subrutina de atención a una interrupción.

Las etapas que transcurren al atender a una solicitud de interrupción son las siguientes:

1. El microcontrolador completa la ejecución de la instrucción en curso.

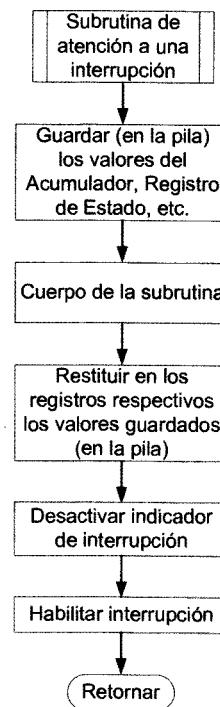


Figura 7.3 Estructura general de la subrutina de atención a una solicitud de interrupción.

2. El valor del PC se guarda en la pila, para “recordar” la dirección de la instrucción a la que hay que regresar cuando termine la atención a la interrupción.
3. La dirección de la subrutina de atención a la interrupción se pone en el PC, con lo cual se salta a esa dirección y comienza la ejecución de la subrutina.
4. Se ejecuta la subrutina de atención a la interrupción. Como toda subrutina, termina (su ejecución) con una instrucción de retorno.
5. Con la ejecución de la instrucción de retorno, se regresa al programa interrumpido.

7.1.3 Interrupciones fijas y vectorizadas

Para informar a la CPU de la dirección donde comienza la subrutina que atiende la interrupción, hay dos alternativas:

1. La subrutina que atiende a la interrupción se coloca en un lugar fijo de la memoria de programa, conocido de antemano por la CPU.

7.2 Las interrupciones en los microcontroladores PIC

7.2.1 Fuentes de interrupción y registros asociados

Las interrupciones de los microcontroladores PIC de la gama media son interrupciones enmascarables y fijas, de modo que se pueden habilitar o inhabilitar globalmente y además cada fuente de interrupción se puede habilitar o inhabilitar individualmente (figura 7.7). Para que una solicitud de interrupción se haga efectiva, tanto la fuente de la interrupción como el sistema en su conjunto deben estar habilitados. El sistema se habilita o inhabilita globalmente mediante el bit GIE del registro de funciones especiales INTCON. Los dispositivos con posibilidad de interrumpir se habilitan o inhabilitan individualmente mediante bits de los registros INTCON, PIE1 y PIE2.

Al ser todas las interrupciones fijas, todas las solicitudes, si están habilitadas, hacen que el microcontrolador pase a ejecutar la instrucción que esté en la dirección 4 de la memoria de programa. Dentro de la rutina de atención a la interrupción, el programador debe averiguar la fuente de la interrupción consultando los bits apropiados en los registros de funciones especiales (INTCON, PIR1 y PIR2) asociados al sistema de interrupción del PIC.

Cuando se produce una solicitud de interrupción, si el sistema en su conjunto y la fuente en particular están habilitados, el microcontrolador termina la instrucción en curso, guarda en la pila el valor del contador de programa y salta a la dirección 4 de la memoria de programa.

El tiempo transcurrido entre el momento en que se produce la solicitud de interrupción y el comienzo de la ejecución de la primera instrucción de la subrutina de atención a la interrupción (la que está en la dirección 4), está entre 3 y 3,75 ciclos de máquina. Este es el denominado *tiempo de latencia* de la interrupción, cuyo valor exacto depende del momento en que se produce la solicitud dentro de un ciclo de máquina y de la fuente de la interrupción (si es interna o externa).

La figura 7.5 ilustra las operaciones que tienen lugar en el microcontrolador durante el tiempo de latencia. En primer lugar se completa la instrucción que estaba en curso cuando se produjo la solicitud de interrupción; a continuación se guarda en la pila el valor del contador de programa (PC), que apunta a la siguiente instrucción, y finalmente se pone el valor 0004 en el PC, con lo cual se salta a la primera instrucción de la rutina que atiende la solicitud de interrupción. Al atender una solicitud de interrupción, el sistema de interrupciones queda inhabilitado (el bit GIE es puesto a 0).

Cada módulo de entrada y salida puede generar al menos una solicitud de interrupción. Entre las posibles fuentes de interrupción están las siguientes:

- Interrupción externa por el terminal INT del microcontrolador.
- Interrupción por cambio en el nivel lógico de las entradas RB4:RB7 del Puerto B.
- Interrupción por desbordamiento de los temporizadores Timer0, Timer1 y Timer2.
- Interrupción por algún evento en el módulo CCP.
- Interrupción por el puerto serie USART.
- Interrupción por el convertidor A/D.

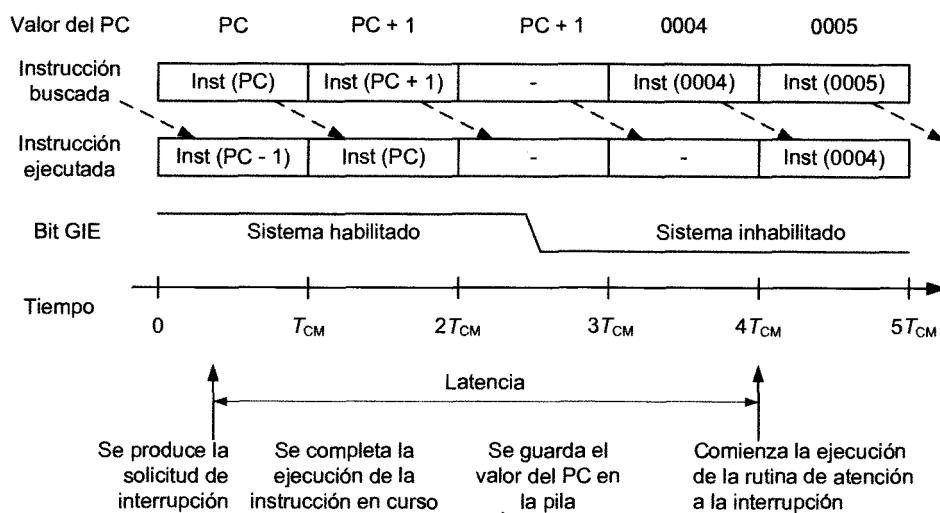


Figura 7.5 Operaciones que tienen lugar en el microcontrolador durante el tiempo de latencia: se completa la instrucción en curso, se guarda el valor del PC en la pila y se pone el valor 0004 en el PC, con lo cual se ejecuta la primera instrucción de la rutina de atención a la interrupción. Cuando se ejecuta esta rutina, el sistema de interrupción queda inhabilitado (se habilita nuevamente con la instrucción `retfie` de retorno al programa interrumpido).

Todos los PIC de clase media usan al menos un registro de funciones especiales para controlar las interrupciones: el registro INTCON. En este registro se controlan la interrupción externa (proveniente del terminal INT), la interrupción por cambio en los terminales RB4 a RB7 y la interrupción por desbordamiento del Timer0. Hay también un bit (GIE) para habilitar globalmente el sistema de interrupción. Las restantes fuentes de interrupción se

controlan con los registros PIE1, PIR1, PIE2 y PIR2. La figura 7.6 muestra los bits del registro de control de las interrupciones INTCON.

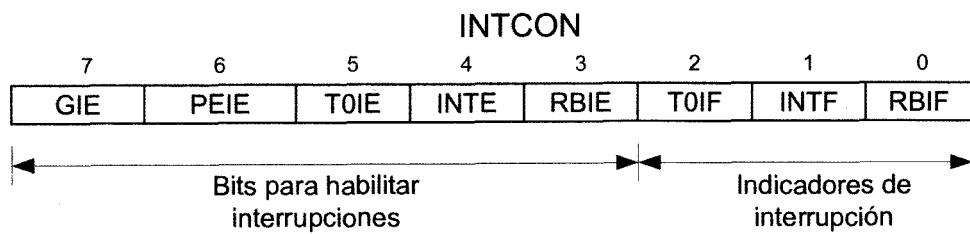


Figura 7.6 El registro INTCON, para el control de las interrupciones.

El significado de cada bit del registro INTCON es el siguiente:

GIE (Global Interrupt Enable). Con este bit se habilita (con GIE = 1) o inhabilita (con GIE = 0) el sistema de interrupción del microcontrolador. Cuando se produce una solicitud de interrupción, este bit pasa automáticamente a 0, con lo cual se inhabilita el sistema de interrupción, que no atenderá nuevas solicitudes hasta que GIE = 1. La instrucción de retorno de una subrutina de atención a una interrupción (`retfie`) pone el bit GIE en 1 y así el sistema de interrupción del PIC queda nuevamente habilitado. Al producirse un reset, el bit GIE es puesto a 0, quedando así inhabilitado el sistema de interrupción.

PEIE (Peripheral Interrupt Enable). Con este bit se habilitan las interrupciones de otras fuentes que no están presentes en el registro INTCON sino en los registros PIE.

TOIE, TOIF. Bits relacionados con la interrupción del Timer0. Con el bit TOIE puesto a 1 se habilita la interrupción del Timer0. TOIF es el indicador de desbordamiento del Timer1: cuando este bit pasa a 1, se genera la solicitud de interrupción correspondiente, si TOIE = 1.

INTE, INTF. Bits relacionados con la interrupción externa. Con el bit INTE se habilita esta interrupción, mientras que INTF es el indicador de que se ha detectado un flanco (de subida o de bajada) en el terminal de entrada INT.

RBIE, RBIF. Bits relacionados con la interrupción por cambio de nivel en los bits RB4 a RB7. El bit RBIE puesto a 1 habilita esta interrupción. RBIF indica que se ha producido un cambio en alguno de los terminales RB4 a RB7; si RBIE = 1, se produce la interrupción.

La figura 7.7 ilustra cómo operan los bits del registro INTCON. La solicitud de interrupción externa se hace a través del terminal INT, que comparte funciones con el terminal RB0 del puerto B en la mayoría de los microcontroladores PIC de clase media. Esta interrupción se produce con los flancos de la señal en el terminal INT. El flanco que produce la interrupción puede ser el de subida o el de bajada, lo cual se programa mediante el bit INTEDG del registro OPTION (bit OPTION<6>). La interrupción externa queda reportada mediante el bit INTF del registro INTCON y se habilita o inhabilita mediante el bit INTE de ese mismo registro.

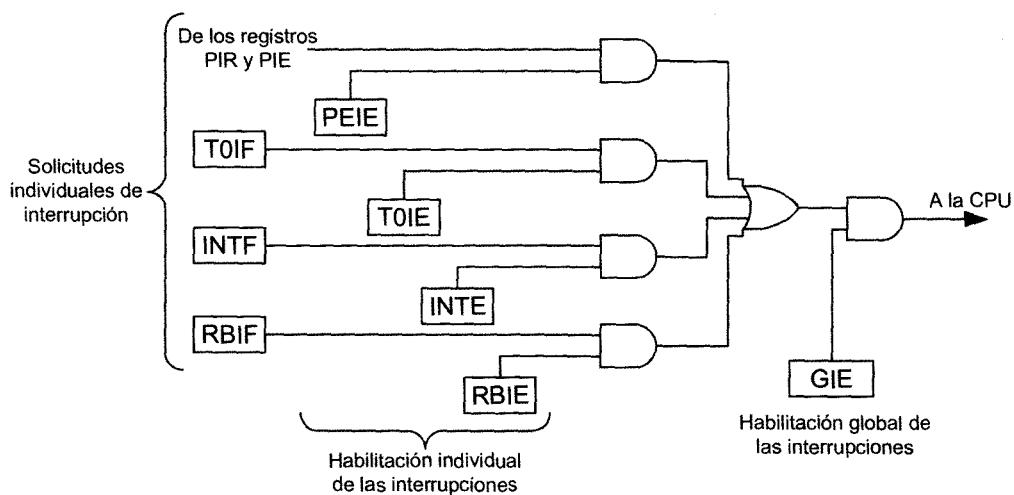


Figura 7.7 Papel de los bits del registro INTCON en el sistema de interrupción de los microcontroladores PIC de clase media. El sistema se habilita o inhabilita mediante el bit GIE. Cada posible fuente de interrupción tiene un bit de control para habilitar o no la solicitud de interrupción de la fuente. Las solicitudes quedan plasmadas en los indicadores correspondientes (T0IF, INTF y RBIF), independientemente de que la solicitud progrese o no.

Si los terminales RB4 a RB7 están programados como entradas y se produce un cambio en el nivel lógico de la señal en cualquiera de estas entradas, es decir, de '0' a '1' o viceversa, hay una solicitud de interrupción. Al producirse la interrupción por cambios en RB4 a RB7, el bit RBIF del registro INTCON se pone a '1'. La interrupción puede habilitarse o inhabilitarse mediante el bit RBIE de ese mismo registro. Esta interrupción se puede utilizar para "despertar" al microcontrolador si se halla en el modo de bajo consumo (modo *Sleep*).

La figura 7.8 muestra el circuito simplificado con el que se genera la interrupción por cambios en el nivel de cualquiera de las entradas RB4 a RB7.

La señal en el terminal se muestrea en dos instantes de tiempo diferentes correspondientes a los estados Q1 y Q3 de cada ciclo de máquina (figura 2.2). Si el nivel lógico de la señal en el terminal ha cambiado entre los muestreos, las salidas Q de los biestables D tendrán valores diferentes. En este caso, la salida de la puerta or-exclusivo pasará a 1 y con ello el bit RBIF será puesto también a 1, con lo que se genera la solicitud de interrupción. Para que esto ocurra, el terminal debe estar configurado como entrada (el bit TRISB*<i>* debe ser 1).

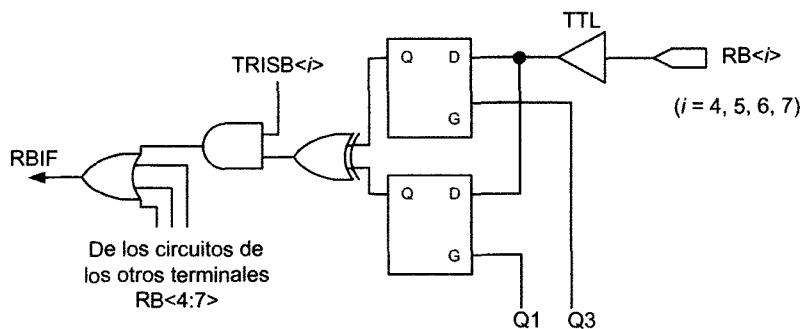


Figura 7.8 Circuito (simplificado) asociado a los terminales RB4 a RB7 programados como entradas, que ilustra cómo se genera la interrupción por cambios en el nivel lógico de esas entradas. Q1 y Q3 son las señales internas correspondientes a los estados de igual nombre en cada ciclo de máquina (ver figura 2.2).

Los registros PIE y PIR controlan las interrupciones de los diferentes módulos periféricos incorporados en el microcontrolador PIC. En los registros PIE (PIE1, PIE2) están los bits para habilitar o inhabilitar las interrupciones de los periféricos. En los registros PIR (PIR1, PIR2) están los bits indicadores de que ha sido solicitada una interrupción por los diferentes periféricos. La estructura de estos registros, es decir, el significado y posición de los bits dentro del registro, depende del dispositivo en particular y puede variar de un PIC a otro.

Ejemplo 7.2

Registros PIE y PIR en los microcontroladores PIC16F87x. La estructura y composición de los registros PIE y PIR es particular de cada PIC, pues depende de los módulos de entrada y salida disponibles en el microcontrolador. La familia de microcontroladores PIC16F87x tiene dos registros PIE (PIE1 y PIE2) y dos registros PIR (PIR1 y PIR2), cuya estructura en bits se ilustra en la figura 7.9. La tabla 7.2 muestra los nombres de los bits y los módulos de entrada y salida asociados a ellos.

Tabla 7.2 Nombre de algunos bits de los registros PIE y PIR y módulos de entrada y salida asociados a ellos.

Bit de control en registro PIE	Bit indicador en registro PIR	Módulo fuente de la interrupción
TMR1IE	TMR1IF	Timer1
TMR2IE	TMR2IF	Timer2
CCP1IE	CCP1IF	CCP1
CCP2IE	CCP2IF	CCP2
RCIE	RCIF	USART (recepción)
TXIE	TXIF	USART (transmisión)
ADIE	ADIF	Convertidor A/D
PSPIE	PSPIF	PSP
SSPIE	SSPIF	SSP
BCLIE	BCLIE	SSP (colisión en el bus I ² C)
EEIE	EEIF	EEPROM de datos (escritura)

	7	6	5	4	3	2	1	0
PIE1	PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
PIE2	-	-	-	EEIE	BCLIE	-	--	CCP2IE
PIR1	PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
PIR2	-	-	-	EEIF	BCLIF	-	--	CCP2IF

Figura 7.9 Registros PIE y PIR en los microcontroladores PIC16F87x.

Para que una solicitud de interrupción cause efectivamente la interrupción del programa en curso, es necesario, en primer lugar, que el sistema de interrupción del PIC esté habilitado, es decir, el bit GIE del registro INTCON debe ser 1. En segundo lugar, la fuente de la interrupción debe estar también habilitada. Por ejemplo, si se trata de una interrupción externa, el bit INTE del registro INTCON debe ser 1. Si se dan estas condiciones, la solicitud progresará y el microcontrolador guarda en la pila el contador de programa y salta a la dirección 4 de la memoria de programa. El bit GIE pasa automáticamente a 0, con lo cual se inhabilita el sistema de interrupción y no se atenderán nuevas solicitudes. El sistema se habilita de nuevo cuando se ejecuta la instrucción `retfie` de retorno al programa interrumpido desde la subrutina de atención a la interrupción. La instrucción `retfie` pone el bit GIE a 1 y así el sistema de interrupción queda nuevamente habilitado, y con ello el PIC está listo para atender nuevas solicitudes de interrupción.

Al producirse un reset, el bit GIE es puesto a 0, de modo que el microcontrolador no atenderá ninguna solicitud de interrupción después de haberse producido un reset por cualquiera de las causas posibles (apartado 2.1.6). Así, por ejemplo, cuando se energiza el microcontrolador, el dispositivo inicia su trabajo con el sistema de interrupción inhabilitado. Debe ser el programador quien lo habilite, poniendo el bit GIE a 1.

Los bits de control individual de las interrupciones (T0IE, INTE, etc.) normalmente no se modifican cuando hay una solicitud de interrupción.

Los indicadores individuales de solicitudes de interrupción (T0IF, INTF, etc.) pasan a 1 automáticamente para dar cuenta de una solicitud de interrupción, pero normalmente deben ser puestos a 0 en el programa que atiende la interrupción.

7.2.2 Estructura del subprograma de atención a una interrupción

Las interrupciones son eventos que pueden ocurrir en cualquier momento mientras se ejecuta una instrucción cualquiera del programa. No es posible, en general, prever durante qué instrucción habrá una solicitud de interrupción. Esta situación obliga al programador a tomar ciertas precauciones para preservar los valores de los registros que, siendo usados por el programa interrumpido, sean también utilizados por la rutina que atiende la interrupción. Los registros W y STATUS son de los que se utilizan con mayor frecuencia en cualquier programa, de modo que si hay una solicitud de interrupción, con toda seguridad convendrá preservar sus valores. Esto significa que se deben guardar (puede que en la memoria) los valores que tenían esos registros cuando fue solicitada la interrupción, y una vez finalizada la atención a la interrupción, se deben restituir dichos valores en los registros respectivos para que el programa interrumpido pueda continuar su curso normal.

En muchos microprocesadores y microcontroladores, la forma de preservar los registros mientras se atiende una solicitud de interrupción se reduce a guardar sus valores en la pila. Para esto se usan instrucciones del tipo PUSH registro (para guardar en la pila el contenido del registro) y POP registro (para extraer de la pila un valor y colocarlo en el registro). El uso de la pila (con su estructura LIFO: *last in first out*) para preservar registros permite anidar subrutinas de atención a interrupciones. Esto significa que mientras se está atendiendo una solicitud de interrupción, se puede procesar una segunda solicitud sin que haya terminado la primera. Todo esto puede dar lugar a potentes y complejos sistemas de interrupción.

Desafortunadamente, en los microcontroladores PIC de clase media no existen las instrucciones PUSH ni POP, ni se puede emplear la pila para almacenar otra información que no sea el contador de programa. La ausencia de una pila para resguardar otros registros (como W y STATUS) durante la atención a una solicitud de interrupción, dificulta mucho el anidamiento de subrutinas de atención a interrupciones, por lo que el sistema de interrupción de los PIC de clase media resulta relativamente simple. Si el microcontrolador está atendiendo una solicitud de interrupción y en el transcurso de la rutina que atiende esa interrupción se produce una nueva solicitud, ésta deberá esperar a que finalice la primera para poder ser atendida. De hecho, esto queda garantizado en los PIC de clase media porque el sistema de interrupción queda inhabilitado mientras se está atendiendo a una solicitud de interrupción y sólo se habilita de nuevo con la instrucción `retfie`, que termina la atención a la interrupción.

Al no haber una pila de propósito general en la que resguardar cualquier registro del microcontrolador, hay que emplear la memoria de datos para resguardar a los registros W y STATUS, o cualquier otro que sea necesario. Esta operación no está exenta de dificultades, dada la estructura en bancos que tiene la memoria de programa y que la mayoría de las instrucciones de transferencia de datos pueden alterar algún bit del registro STATUS. El ejemplo 7.3 ilustra varias soluciones propuestas al efecto por el fabricante de los PIC.

Ejemplo 7.3

Estructura recomendada para la rutina de atención a una interrupción. La rutina utiliza los registros TEMP_W y TEMP_ST para guardar los registros W y STATUS respectivamente. Esta operación, que puede parecer sencilla con la secuencia:

```
movwf TEMP_W
movf STATUS, W
movwf TEMP_ST
```

No es posible, pues la instrucción `movf` afecta el bit Z del registro STATUS, por lo que el valor almacenado en TEMP_ST puede no ser original de STATUS. Hay que valerse de otras instrucciones que no afecten ningún bit de STATUS. Tal es el caso de la instrucción `swapf`. Con esta instrucción, el segmento de programa queda así:

```
movwf TEMP_W
swapf STATUS, W
movwf TEMP_ST
```

que guarda el registro STATUS en TEMP_ST sin alterar el valor original de los bits pero con los cuartetos intercambiados.

El hecho de que el registro STATUS se guarde con sus cuartetos intercambiados no representa ningún

problema: basta con intercambiar los cuartetos de TEMP_ST antes de salir de la rutina de atención a la interrupción para recuperar intacto el registro STATUS. El segmento de programa que restituye los valores de W y STATUS antes de finalizar la rutina, no utiliza la instrucción movf y queda así:

```
swapf TEMP_ST, W
movwf STATUS
swapf TEMP_W, F
swapf TEMP_W, W
```

Otro elemento que hay que tener en cuenta es que al menos el registro TEMP_W debe estar situado en el banco que está activo en el momento de producirse la interrupción. Como, en general, esto no es posible precisarlo de antemano, hay diferentes soluciones según que el microcontrolador disponga o no de zonas de memoria RAM de datos comunes a todos los bancos. Una zona RAM común a todos los bancos es un área de memoria que es direccionable desde cualquier banco como la misma área física de memoria de datos.

A continuación se muestran los listados de las rutinas SRAI1 y SRAI2 con las estructuras recomendadas por Microchip para las rutinas de atención a una interrupción según el PIC disponga o no de zonas comunes de memoria de datos:

```
; Esta rutina es para los PIC que tienen RAM común (Ejemplo: PIC16F84).
; RAM común: un área de RAM que es la misma en todos los bancos.
; TEMP_W y TEMP_ST se definen en esa RAM común.
;
SRAI1:
    movwf TEMP_W          ; Guardar W en TEMP_W.
    swapf STATUS, W       ; Intercambiar cuartetos de STATUS
    movwf TEMP_ST          ; y guardar el resultado en TEMP_ST.
;
; Aquí se coloca el cuerpo de la subrutina.
;
    swapf TEMP_ST, W      ; Recuperar TEMP_ST e intercambiar cuartetos
    movwf STATUS           ; y poner el resultado en STATUS.
    swapf TEMP_W, F        ; Recuperar TEMP_W y ponerlo en
    swapf TEMP_W, W        ; W sin alterar STATUS.
    retfie                 ; Retornar al programa interrumpido.
;
;
; Esta rutina es para los PIC que no tienen RAM común (Ejemplo: PIC16F873).
; RAM común: un área de RAM que es la misma en todos los bancos.
; El registro TEMP_W está definido en cualquier banco.
; El registro TEMP_ST está definido en el banco 0.
```

SRAI2:

```

movwf TEMP_W      ; Guardar W en TEMP_W.
swapf STATUS, W   ; Intercambiar cuartetos de STATUS,
bcf   STATUS, RP0  ; seleccionar el banco 0
movwf TEMP_ST     ; y guardar el resultado en TEMP_ST.

;
; Aquí se coloca el cuerpo de la subrutina
;

swapf TEMP_ST, W  ; Recuperar TEMP_ST y intercambiar cuartetos
movwf STATUS       ; poner en STATUS. El banco seleccionado
                   ; es ahora el original, donde está TEMP_W.
swapf TEMP_W, F    ; Recuperar TEMP_W y ponerlo en
swapf TEMP_W, W    ; W sin alterar STATUS.
retfie             ; Retornar de la interrupción.
;
```

La figura 7.10 ilustra la estructura general que tiene la subrutina de atención a una interrupción en los microcontroladores PIC. Después de preservar los registros W y STATUS mediante alguno de los procedimientos ilustrados en el ejemplo 7.3, el programador debe averiguar cuál es la fuente de la interrupción. Para ello se encuestan los bits indicadores correspondientes, para ver si alguno de ellos tiene el valor 1. Una vez identificada la fuente, se atiende la interrupción. Un punto importante es la puesta a 0 del indicador asociado a la fuente de interrupción (el mismo que fue encontrado con valor 1 durante la encuesta). Finalmente, se restituyen los valores originales de los registros W y STATUS y se retorna al programa interrumpido al ejecutar la instrucción `retfie`. Esta instrucción habilita el sistema de interrupción (pone a 1 el bit GIE).

Las etapas que transcurren en la atención a una solicitud de interrupción en un PIC de la gama media son:

1. El microcontrolador completa la ejecución de la instrucción en curso.
2. El valor del PC se guarda en la pila.
3. El PC toma el valor 0004, con lo cual se salta a esa dirección y comienza la ejecución de la subrutina.
4. Se guardan los registros W y STATUS. (Ver el ejemplo 7.3).
5. Se determina la fuente de la interrupción, encuestando los indicadores de las posibles fuentes.

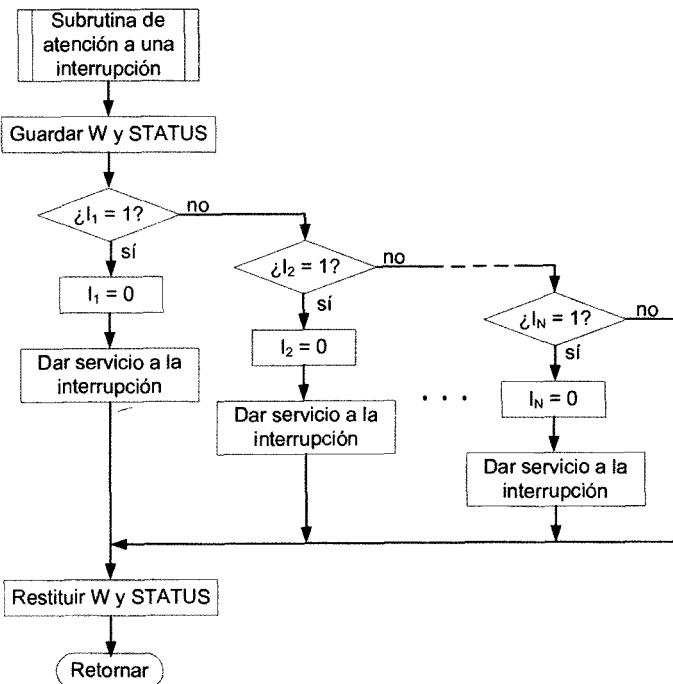


Figura 7.10 Estructura de la subrutina de atención a una interrupción en los microcontroladores PIC de la gama media. Si hay N posibles fuentes, se consultan los indicadores I_1, I_2, \dots, I_N asociados a esas interrupciones. Estos indicadores son los bits TOIF, INTF, etc. de los registros INTCON, PIR1 y PIR2.

6. Una vez que la fuente ha sido identificada, el indicador correspondiente a esa fuente se desactiva (se pone a 0).
7. Se restituyen los valores de los registros W y STATUS. (Ver el ejemplo 7.3).
8. Se retorna al programa interrumpido con una instrucción `retfie`, la cual extrae de la pila el valor del PC y habilita el sistema de interrupción del PIC (pone a 1 el bit GIE).

Las tres primeras etapas se realizan automáticamente en el microcontrolador, mientras que las etapas 4 a 8 se deben implementar dentro de la subrutina de atención a la interrupción.

7.3 Ejemplos de uso de las interrupciones

7.3.1 Reloj de tiempo real

Una *base de tiempos* es un conjunto de variables cuyos valores reflejan el valor del tiempo real en el microcontrolador. Por ejemplo, una base de tiempos puede estar formada por las variables TICS, SEG, MIN, HOR, las cuales

llevan el conteo de las décimas de segundo, los segundos, los minutos y las horas respectivamente. Las variables no son más que registros del microcontrolador en la memoria de datos.

El *reloj de tiempo real* (RTC: *Real Time Clock*) es un mecanismo de software, basado en una interrupción periódica, que permite actualizar la base de tiempos y sincronizar eventos con ella. Por ejemplo, con cada interrupción periódica, el programa del RTC incrementa el valor de la variable TICS y, según corresponda, actualiza las restantes variables.

Los eventos externos que se sincronicen con la base de tiempos pueden ser periódicos. Por ejemplo, son eventos periódicos leer un canal del convertidor A/D cada 5 s, poner un valor en el puerto B cada 8 s, etc. En el ejemplo 7.5 se explica cómo se realizan estas sincronizaciones.

El elemento fundamental en un reloj de tiempo real es la interrupción periódica con la que se actualiza la base de tiempos. El valor T del período de esta interrupción determina la resolución que tiene la variable tiempo en el sistema. Por ejemplo, si $T = 0,1$ s, el sistema no puede discernir intervalos de tiempo menores que 0,1 s. Otro elemento importante es que la ejecución del programa que atienda a la interrupción del RTC debe tomar muy poco tiempo del procesador, de manera que la tarea de llevar la cuenta del tiempo real no limite la realización de otras tareas en el microcontrolador.

Para diseñar un reloj de tiempo real se puede utilizar la interrupción de uno de los temporizadores del microcontrolador. Por ejemplo, se puede programar el Timer0 para obtener una interrupción periódica (tic del reloj) cada cierto número de milisegundos. Se emplea entonces una variable (variable contadora de los tics del reloj) que se incremente (o decremente) con cada interrupción. Con otras variables se puede llevar el conteo de los segundos, minutos, horas, etc.

El ejemplo siguiente ilustra cómo implementar un reloj de tiempo real en un PIC16F873.

Ejemplo 7.4

Reloj de tiempo real. Se tiene un sistema basado en un PIC16F873 con un cristal de 4 MHz. Se desea implementar un reloj de tiempo real (RTC) con una base de tiempos con variables para contar los tics del reloj, los segundos, minutos y horas.

Si el Timer0 se programa con un módulo de conteo de $N = 256$ y el pre-divisor con una razón de división $P = 32$, la interrupción del Timer0 ocurre cada 8,192 ms (122,07 Hz).

Para alcanzar un tiempo aproximado de 1 s, hay que contar 122 interrupciones (tics) del contador. Para ello se puede utilizar un registro (TICS) como contador de los tics. En el registro SEG se implementa el contador de segundos, en MIN el contador de minutos y en HOR el contador de horas.

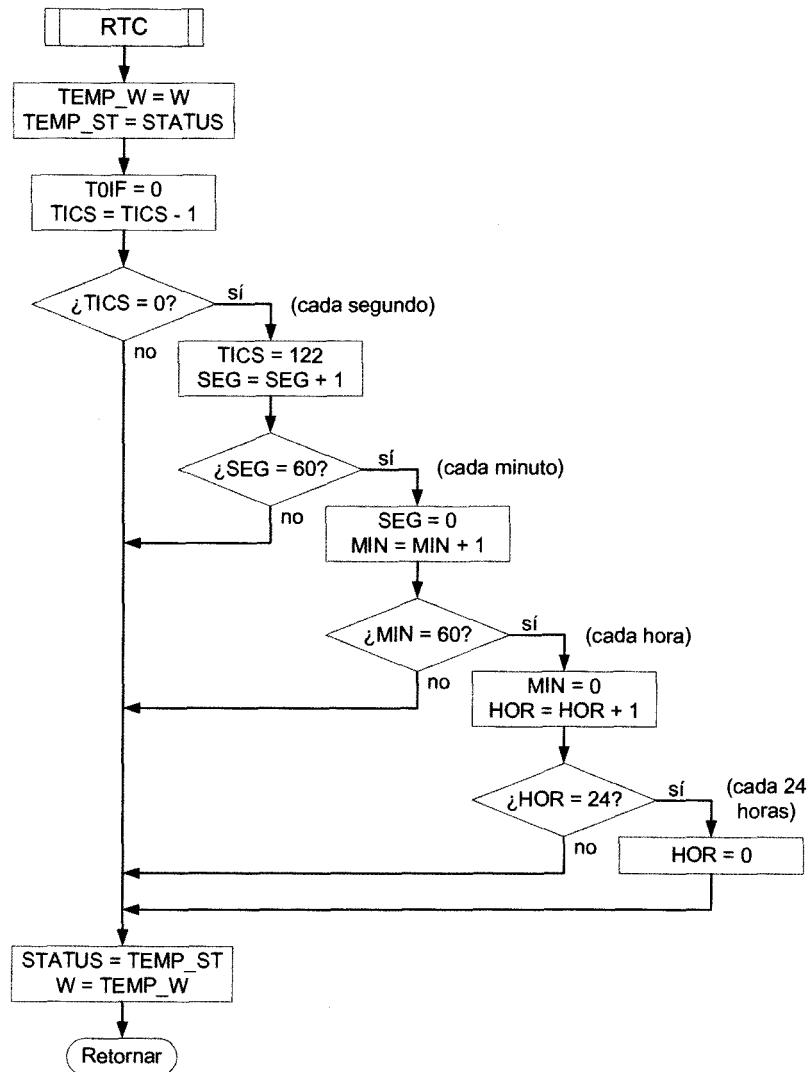


Figura 7.11 Diagrama de bloques con el algoritmo del reloj de tiempo real del ejemplo 7.4. La base de tiempos está constituida por las variables TICS, SEG, MIN y HOR que llevan la cuenta de los tics de reloj, los segundos, los minutos y las horas. El algoritmo se ejecuta muy rápidamente pues la gran mayoría de las veces toma el camino del NO en la primera decisión. El algoritmo se ejecuta completo (tomando los caminos de todos los SÍ solamente en un TIC de reloj al día (justo a medianoche).

La figura 7.11 muestra el algoritmo seguido por el RTC. Aunque pudiera parecer que el algoritmo consume mucho tiempo del microcontrolador, en realidad no es así, pues la inmensa mayoría de las veces el camino que toma la ejecución del algoritmo en las decisiones es el NO. De las 122 interrupciones que ocurren en cada segundo, sólo una toma el camino del Sí (en el bloque de decisión ¿TICS = 0?); de los 60 s de 1 min, sólo en uno se toma el camino del Sí en el bloque de decisión ¿MIN = 0? y así sucesivamente. Por ello, el algoritmo consume realmente muy poco tiempo del procesador, lo cual es una característica muy deseable en todo RTC.

En el algoritmo mostrado en la figura 7.11, la variable TICS se decremente mientras que SEG, MIN y HOR se incrementan en los instantes en que corresponde hacer estas operaciones. El motivo de esta diferencia al operar con esas variables está en que se ha querido dar velocidad a la ejecución del algoritmo, de modo que ocupe el menor tiempo de procesador posible. La operación de incrementar la variable TICS y comparar su valor con 122 necesita más instrucciones que decrementar y preguntar si el valor es cero, por lo que esta última opción es más rápida.

A continuación se da el listado del programa del RTC que sigue el algoritmo de la figura 7.11.

```
; Realización de un reloj de tiempo real usando la
; interrupción del timer 0.

;
list p=16f873
#include      <p16f873.inc>

;
; Variables de la base de tiempos:
;

TICS      equ    0x20 ; Contador de tics.
SEG       equ    0x21 ; Contador de segundos.
MIN       equ    0x22 ; Contador de minutos.
HOR       equ    0x23 ; Contador de horas.
;

; Otras variables:
;

TEMP_W    equ    0x24
TEMP_ST   equ    0x25
;

org      0
goto    inicio
org      4
goto    rtc
;

inicio:
    clrf   INTCON      ; Inhabilitar todas las interrupciones.
    bsf    STATUS, RP0   ; Seleccionar banco 1.
    movlw  0xC4          ; Pre-divisor de 32 asignado
    movwf  OPTION_REG    ; al Timer0.
    bcf    STATUS, RP0   ; Seleccionar banco 0.
    movlw  .0             ; Módulo de conteo de 256
    movwf  TMRO          ; en el Timer0.
    movlw  .122           ; Cantidad de tics por segundo.
    movwf  TICS           ; en el contador de tics.
    clrf   SEG            ; Contador de segundos en 0.
    clrf   MIN            ; Contador de minutos en 0.
    clrf   HOR            ; Contador de horas en 0.
    bsf    INTCON, TOIE   ; Habilitar interrupción del Timer0.
```

```

bsf    INTCON, GIE      ; Habilitar el sistema de interrupción.

;
; prog:
nop
goto  prog              ; Programa principal trivial.

;
; rtc:
movwf TEMP_W            ; Guardar W en TEMP_W.
swapf STATUS, W          ; Intercambiar cuartetos de STATUS,
bcf    STATUS, RP0        ; seleccionar el banco 0
movwf TEMP_ST             ; y guardar el resultado en TEMP_ST.

;
bcf    INTCON, T0IF      ; 0 en indicador de desbordamiento de Timer0.
decfsz TICS, f           ; ¿Se alcanzó el segundo?
goto   fin_rtc           ; No, salir de la interrupción.
;
;rtc_seg:
movlw .122
movwf TICS
incf  SEG, f             ; Sí, se alcanzó el segundo, entonces
                          ; recargar la variable TICS
                          ; con la la cantidad de tics/s e
                          ; incrementar los segundos.

;
xorlw .60                 ; ¿SEG = 60?
btfsr STATUS, Z           ;
goto   fin_rtc           ; No, retornar.
;
;rtc_min:
clrf  SEG
incf  MIN, f             ; Sí, se alcanzó el minuto, entonces
                          ; poner los segundos en 0 e
                          ; incrementar los minutos.

;
xorlw .60                 ; ¿MIN = 60?
btfsr STATUS, Z           ;
goto   fin_rtc           ; No, retornar.
;
;rtc_hor:
clrf  MIN
incf  HOR, f              ; Sí, se alcanzó la hora, entonces
                          ; poner los minutos en 0 e
                          ; incrementar las horas.

;
xorlw .24                 ; ¿HOR = 24?
btfsr STATUS, Z           ;
goto   fin_rtc           ; No, retornar
;
;rtc_dia:
clrf  HOR                 ; Sí, transcurridas 24 horas, entonces
                          ; poner las horas en 0.

;
fin_rtc:
swapf TEMP_ST, W           ; Recuperar TEMP_ST y intercambiar cuartetos
movwf STATUS                ; poner en STATUS. El banco seleccionado
                            ; es ahora el original, donde está TEMP_W.
swapf TEMP_W, F             ; Recuperar TEMP_W y ponerlo en
swapf TEMP_W, W             ; W sin alterar STATUS.

```

```

retfie           ; Retornar al programa interrumpido.

;
end             ; Fin del programa fuente.

```

7.3.2 Sincronización de eventos al reloj de tiempo real

Si se dispone de un RTC con una base de tiempos que lleve el conteo de las fracciones de segundos, los segundos, los minutos, etc., es posible con relativa facilidad sincronizar con esa base diversos eventos, de forma que cada evento se realice periódicamente a intervalos determinados, iguales o diferentes, para cada evento.

La figura 7.12 ilustra, mediante un caso particular, cómo proceder en general para sincronizar eventos con una base de tiempos realizada con un RTC. En este caso se han ideado dos eventos denominados EVENTO1 y EVENTO2, que deben ejecutarse periódicamente cada 3 s y 5 s respectivamente. La forma de proceder es la siguiente.

En el programa del RTC se incorporan tantas variables contadoras como eventos se quieran sincronizar. Cada una de estas variables marca el período de repetición de un evento. Así, por ejemplo, si se quieren sincronizar dos eventos, para que uno se ejecute cada 3 s y el otro cada 5 s, la solución es añadir al programa del RTC dos variables que se incrementan cada segundo desde 0 hasta llegar a 3 y 5 respectivamente. En la figura 7.12 éstas variables son SEG3 y SEG5.

Dentro del programa del RTC se añade también un indicador o bandera por cada evento que se desea sincronizar. Estas banderas se ponen a 1 cuando se alcanza el tiempo en que se debe ejecutar el evento correspondiente. Estos indicadores pueden ser bits de un registro del microcontrolador. En el caso mostrado en la figura 7.12 se han utilizado para este fin los bits 0 y 1 de un registro del microcontrolador, que se denominado FLAGS.

Por otra parte, en el programa principal se consulta continuamente el estado de estas banderas; si se encuentra que alguna de ellas está en 1, ello significa que el evento se debe ejecutar, y esto se puede hacer llamando a una subrutina que implemente la acción correspondiente. Al ejecutar un evento, su bandera debe ser puesta a 0. Desde el punto de vista de la programación, estas banderas son variables globales pues deben ser accesibles tanto desde el programa de atención a la interrupción del reloj como desde el programa principal.

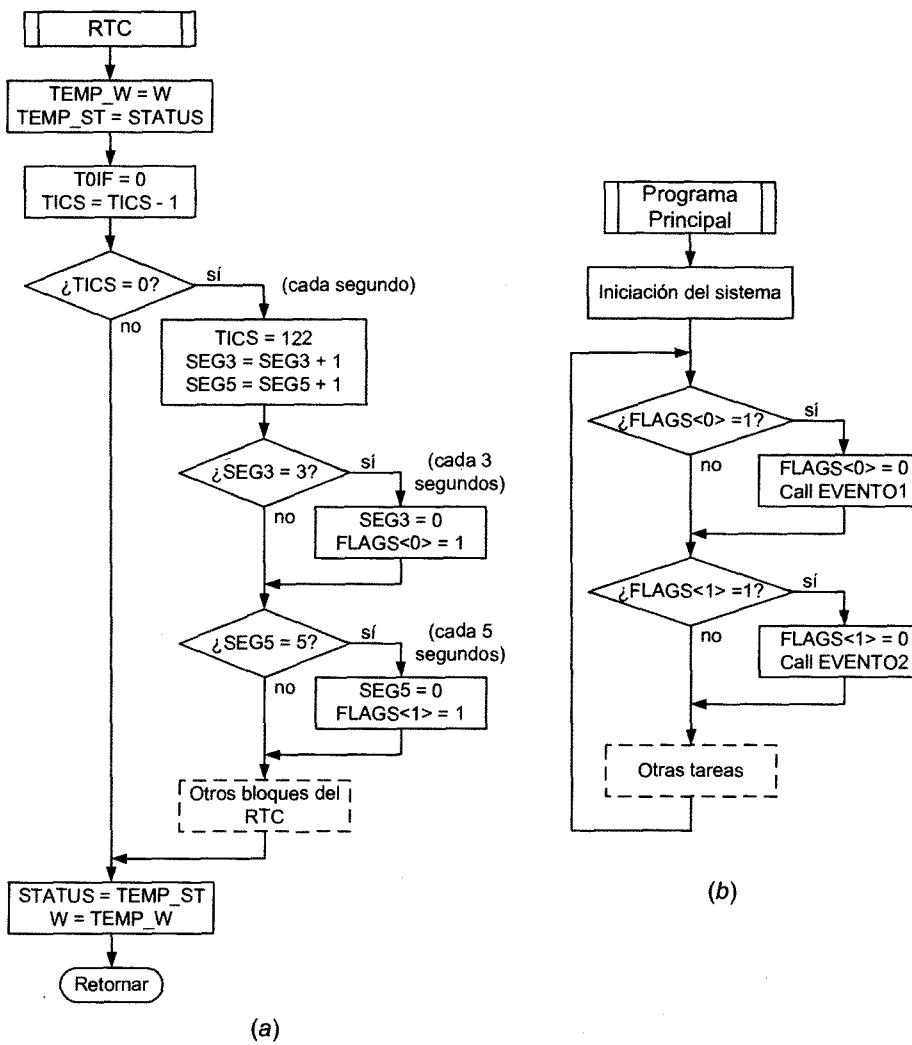


Figura 7.12 Sincronización de eventos al RTC. Los eventos denominados EVENTO1 y EVENTO2 se ejecutan cada 3 s y 5 s respectivamente. (a) Diagrama de bloques del RTC, donde se destacan los contadores de segundos SEG3 y SEG5, que se incrementan cada segundo hasta llegar a 3 s y 5 s respectivamente; además, están los bits 0 y 1 del registro FLAGS que operan como indicadores de que han transcurrido esos tiempos. (b) Diagrama de bloques del programa principal, donde se encuesta continuamente el valor de los indicadores de tiempo transcurrido y, cuando corresponde, se ejecuta el evento correspondiente. Los bits indicadores del tiempo se ponen a 1 en el RTC y se encuestan y ponen a 0 en el programa principal.

Ejemplo 7.5

Sincronización de dos eventos a una base de tiempos. Se tiene un sistema basado en un PIC16F873 con un cristal de 4 MHz. Se desea implementar un RTC y una base de tiempos y realizar dos eventos sincronizada con ella:

1. Alternar el valor del terminal RB0 entre 0 y 1 cada 3 s.
2. Alternar el valor del terminal RB1 entre 0 y 1 cada 5 s.

La solución de este problema sigue el algoritmo de la figura 7.12. Si el cristal es de 4 MHz, y el Timer0 se programa con un módulo de conteo $N = 256$ y el pre-divisor se sitúa con un factor de división $P = 32$, la interrupción del Timer0 ocurre cada 8,192 ms (122,07 Hz). Entonces, para alcanzar un tiempo de un segundo hay que contar 122 interrupciones (tics) del contador. Para ello se puede utilizar un registro (TICS) como contador de los tics. En los registros SEG3 y SEG5 se implementan contadores de 0 s a 3 s y 0 s a 5 s, respectivamente. Como bandera indicadora de que han transcurrido los 3 s o los 5 s, se han utilizado los bits 0 y 1 del registro FLAGS (FLAGS<0>, FLAGS<1>). Estos bits son puestos a 1 en el RTC cada 3 s y 5 s, respectivamente. En el programa principal son consultados y puestos a 0 cuando se efectúa el evento correspondiente.

El listado del programa es el siguiente:

```

;
; Realización de un reloj de tiempo real usando la
; interrupción del Timer0.
; A esta base se sincronizan los eventos EVEN1 y EVEN2,
; que se realizan cada 3 y 5 segundos respectivamente.
;

list p=16f873
#include <p16f873.inc>

TICS    equ    0x20      ; Contador de tics.
SEG3   equ    0x21      ; Contador de segundos hasta 3.
SEG5   equ    0x22      ; Contador de segundos hasta 5.
FLAGS  equ    0x23      ; Banderas de eventos realizadas en bits 0 y 1.
TEMP_W  equ    0x24
TEMP_ST equ    0x25

org     0
goto   inicio
org     4
goto   rtc

inicio:
    clrf  PORTB
    clrf  INTCON      ; Inhabilitar todas las interrupciones.
    bsf   STATUS, RP0 ; Seleccionar banco 1.
    movlw 0xC4        ; Pre-divisor de 32 asignado
    movwf OPTION_REG ; al Timer0.
    clrf  TRISB       ; Puerto B en salida.
    bcf   STATUS, RP0 ; Seleccionar banco 0.
    movlw .0           ; Módulo de conteo de 256
    movwf TMR0         ; en el Timer0.
    movlw .122         ; Cantidad de tics por segundo.
    movwf TICS         ;

```

```

clrf    SEG3      ; Contador SEG3 en 0.
clrf    SEG5      ; Contador SEG5 en 0.
clrf    FLAGS     ; Banderas de eventos en 0.
bsf     INTCON, T0IE ; Habilitar interrupción del Timer0.
bsf     INTCON, GIE  ; Habilitar el sistema de interrupción.

prog:
    btfsc  FLAGS, 0   ; ¿FLAGS<0>=0?
    call    evento1    ; No, entonces hacer evento1.
    btfsc  FLAGS, 1   ; ¿FLAGS<1>=0?
    call    evento2    ; No, entonces hacer evento2.
    goto   prog

evento1:
    bcf    FLAGS, 0   ; Poner FLAGS<0> en 0
    btfsc  PORTB, 0   ; ¿PORTB<0> = 0?
    goto   even1_pon0  ; No, es 1, entonces poner en 0.
even1_pon1:
    bsf    PORTB, 0   ; Sí, es 0, entonces poner en 1.
    return
even1_pon0:
    bcf    PORTB, 0   ; Poner en 0
    return

evento2:
    bcf    FLAGS, 1   ; Poner FLAGS<1> en 0
    btfsc  PORTB, 1   ; ¿PORTB<0> = 0?
    goto   even1_pon0  ; No, es 1, entonces poner en 0.
even2_pon1:
    bsf    PORTB, 1   ; Sí, es 0, entonces poner en 1.
    return
even2_pon0:
    bcf    PORTB, 1   ; Poner en 0.
    return

rtc:
    movwf  TEMP_W    ; Guardar W en TEMP_W.
    swapf  STATUS, W  ; Intercambiar cuartetos de STATUS,
    bcf    STATUS, RP0  ; seleccionar el banco 0
    movwf  TEMP_ST   ; y guardar el resultado en TEMP_ST.
    ;
    bcf     INTCON, T0IF ; Borrar flag del Timer0.
    decfsz TICS, f    ; ¿Llegamos al segundo?
    goto   fin_rtc    ; No, salir de la interrupción.
    ;

```

```

rtc_seg:
    movlw .122          ; Sí, recargar el valor de TICS
    mowwf TICS          ; con la cantidad de tics/s.
    incf SEG3, f         ; Incrementar el contador de 3 s.
    incf SEG5, f         ; Incrementar el contador de 5 s.
    ;
    movf SEG3, w         ;
    xorlw .3              ; ¿SEG3 = 3?
    btfsc STATUS, Z       ;
    goto rtc_seg1         ; No, continuar.
    clrf SEG3             ; Sí, transcurrieron 3 s: poner SEG3 en 0,
    bsf  FLAGS, 0          ; poner a 1 la bandera FLAGS<0>
    ; y continuar.

rtc_seg1:
    movf SEG5, w         ;
    xorlw .5              ; ¿SEG5 = 5?
    btfsc STATUS, Z       ;
    goto fin_rtc           ; No, continuar.
    clrf SEG5             ; Sí, transcurrieron 5 s, poner SEG5 en 0,
    bsf  FLAGS, 1          ; poner a 1 la bandera FLAGS<1>
    ; y continuar.

fin_rtc:
    swapf TEMP_ST, W      ; Recuperar TEMP_ST e intercambiar cuartetos
    mowwf STATUS           ; poner en STATUS. El banco seleccionado
                           ; es ahora el original, donde está TEMP_W.
    swapf TEMP_W, F         ; Recuperar TEMP_W y ponerlo en
    swapf TEMP_W, W         ; W sin alterar STATUS.
    retfie                  ; Retornar al programa interrumpido.

end

```

7.3.3 Protección contra fallos de hardware

En general es conveniente proteger al software que debe esperar alguna señal proveniente del hardware externo al microcontrolador, pues si fallara dicha señal se podría caer en un lazo de espera infinito. En estos casos resulta conveniente limitar el tiempo de espera a un valor razonable. Este valor es conocido en inglés como *time-out*.

Para ilustrar este problema, supóngase que se debe atender a un periférico y para ello hay que esperar alguna señal externa al microcontrolador. Si todo el hardware funcionara siempre correctamente, un algoritmo como el mostrado en la figura 7.13a sería suficiente para atender al periférico. Pero si se produjera un fallo en el hardware externo, el tiempo de espera con este algoritmo podría ser infinito, lo cual sería inadmisible.

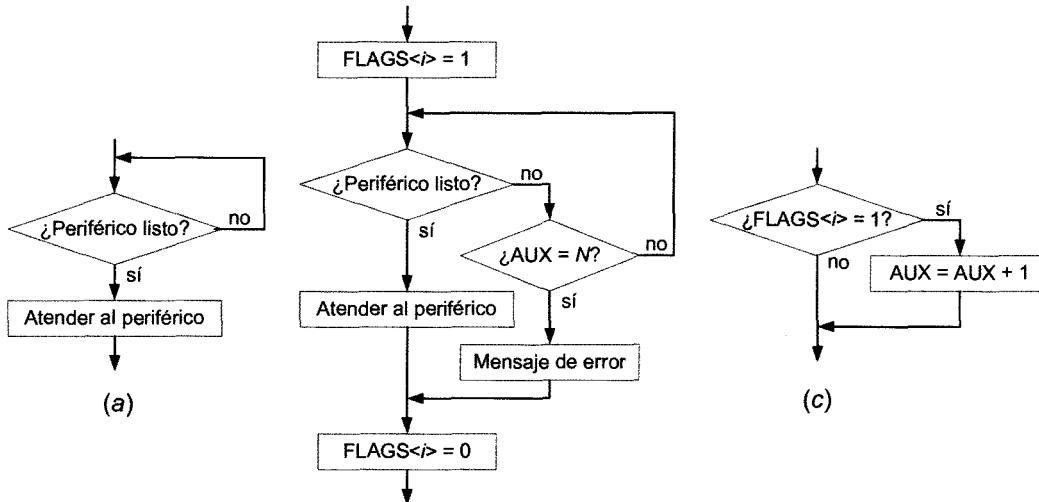


Figura 7.13 Protección contra fallos de hardware en la atención a periféricos. Si para atender a un periférico hay que esperar una señal externa al microcontrolador, existe el peligro de que la espera se haga infinita si se emplea el algoritmo representado en (a). En (b) se muestra un algoritmo modificado, que limita el tiempo de espera. Para ello se usa una variable (AUX) que lleva la cuenta del tiempo de espera. Si AUX alcanza el valor N (que corresponde a un tiempo de espera máximo, fijado convenientemente), entonces se ha producido un fallo en el hardware y se emite un mensaje de error. En (c) se muestra una sección del RTC donde AUX es incrementada. AUX sólo debe incrementarse si se ha entrado en el lazo de espera de atención al periférico, lo cual se le indica al RTC mediante el bit de control $\text{FLAGS}_{<i>}$. La ubicación dentro del RTC de la sección mostrada en (c) depende de si la variable AUX se va a incrementar con cada tic de reloj o con otro intervalo de tiempo.

Una solución para limitar el tiempo de espera en una situación como la representada en la figura 7.13a, es usar en el RTC una variable que lleve la cuenta del tiempo de espera transcurrido. En el programa de atención al periférico, el lazo se interrumpe cuando esta variable alcanza un valor determinado que corresponde a un tiempo de espera que se juzgue adecuado para la aplicación. Las figuras 7.13b y 7.13c ilustran esta solución. En el programa de atención al periférico (figura 7.13b), justo antes de comenzar el lazo de espera, se activa una variable de control (el bit i del registro FLAGS) que indica al RTC que debe comenzar el conteo del tiempo de espera, que será llevado en la variable nombrada AUX. Es decir, la variable AUX con la que se cuenta el tiempo de espera se incrementa convenientemente en el RTC sólo si la variable de control está activada. Los bloques mostrados en la figura 7.13c deben colocarse en el RTC de tal modo que AUX se incremente con cada tic del reloj, o a cada segundo, etc. según convenga. Si la variable AUX alcanza el valor N , que corresponde a un

tiempo de espera prudencial, ello significa que algo anda mal en el hardware, por lo que se interrumpe el lazo de espera y se puede emitir algún mensaje de error indicando la ocurrencia del fallo.

8 La entrada y salida en serie

Este capítulo estudia la entrada y salida en serie en los microcontroladores. Comienza con una exposición de los conceptos básicos sobre la transmisión de información en serie, sus formatos, parámetros e interfaces. A continuación se estudian los puertos serie disponibles en los microcontroladores PIC de la gama media y se dan ejemplos de cómo programarlos.

8.1 Conceptos básicos sobre entradas y salidas en serie

8.1.1 Introducción a la transmisión de datos en serie

La transmisión en serie de información binaria consiste en enviar, uno a uno y de forma sucesiva, los bits de una palabra, a través de los mismos terminales. Así, por ejemplo, la palabra de 8 bits B2h = 10110010b puede ser representada y eventualmente transmitida mediante una señal de datos que represente el 0 con un nivel de tensión bajo (V_L) y el 1 con un nivel de tensión alto (V_H). Esta señal de datos se genera en sincronismo con una señal de reloj cuyo período determina la duración de un bit de la señal de datos, tal como muestra la figura 8.1.

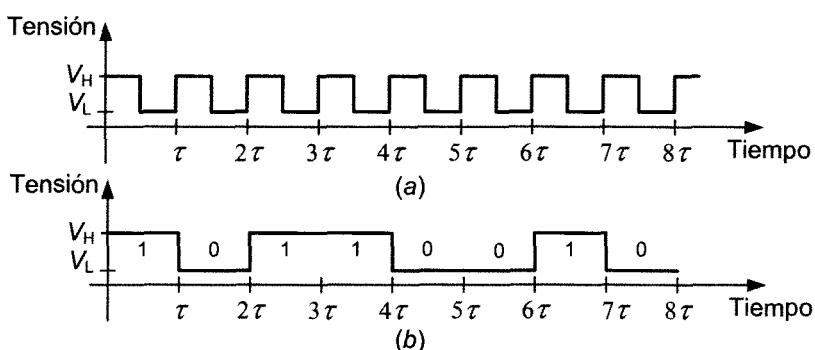


Figura 8.1 Transmisión serie de un byte. (a) Señal de reloj. (b) Señal de datos. Cada bit de una palabra, representado por una tensión alta (V_H) o baja (V_L), es transmitido sucesivamente, es decir, un bit a continuación del otro.

La señal de datos se caracteriza mediante la *velocidad de transmisión* (v_T), que se define como el inverso de la duración de un bit. Si cada bit dura τ segundos, la velocidad de transmisión es

$$v_T = \frac{1}{\tau} \text{ bit/s} \quad (8.1)$$

Resulta evidente que entre el transmisor y el receptor debe haber alguna forma de sincronización para que la información transmitida pueda ser interpretada correctamente por el receptor. En aplicaciones en las que los datos se transmiten a cortas distancias, la señal de reloj puede acompañar a la de datos, lo cual facilita dicha sincronización. Cuando la señal de reloj es transmitida, se dice que la comunicación es *sincrónica*.

Cuando las distancias son grandes, por lo general resulta inadmisible transmitir la señal de reloj, por el coste adicional de los medios de transmisión. Pero aunque el reloj del transmisor no esté disponible en el receptor, la sincronización debe permitir a este último conocer la duración de cada bit y el momento en que comienza cada palabra transmitida. La primera de estas condiciones puede lograrse si tanto el transmisor como el receptor utilizan un reloj de la misma frecuencia, lo que equivale a haber acordado de antemano la velocidad de transmisión. La segunda condición, es decir, el conocimiento por parte del receptor del momento en que comienza una nueva palabra, puede lograrse de dos formas diferentes: "marcando" de alguna forma el inicio de cada nueva palabra, o marcando el inicio de cada bloque de palabras. Esto da lugar a dos modalidades de la comunicación digital conocidas como *comunicación asincrónica* y *comunicación sincrónica*. En la comunicación asincrónica, la sincronización entre transmisor y receptor se realiza palabra a palabra, mientras que en la comunicación sincrónica la sincronización se hace por bloques de palabras. En ambas modalidades es necesario introducir cierta cantidad de información redundante en los datos transmitidos, para lograr la sincronización necesaria entre el transmisor y el receptor.

El término comunicación sincrónica se usa pues para identificar, indistintamente, una transmisión y/o recepción de datos en serie en la cual se transmite la señal de reloj, y una comunicación en la que no se transmite el reloj y la sincronización se hace por bloques de palabras. En cambio, el término comunicación asincrónica se aplica solamente a la transmisión y/o recepción de datos en serie sin transmisión del reloj y donde la sincronización se realiza palabra a palabra.

La coordinación entre el transmisor y el receptor se realiza siguiendo un determinado *protocolo de comunicación*, que es un conjunto de reglas acordadas entre transmisor y receptor que aseguran la transferencia ordenada de los datos. Hay dos tipos diferentes de protocolos de comunicación:

- Protocolos orientados a bytes, en los que todas las palabras transmitidas son de 8 bits. Ejemplo: el protocolo BISYNC (IBM Binary Synchronous Communications Protocol).

- Protocolos orientados a bits, en los que los bloques de datos transmitidos no están necesariamente formados por palabras de 8 bits, es decir, los bloques de datos son conjuntos de bits más que conjuntos de bytes. Ejemplos: los protocolos HDLC (*High -level Data Link Control Protocol*), SDLC (*Synchronous Data Link Control Protocol*), y CSMA/CD (*Carrier Sense, Multiple Access with Collision Detection*) muy utilizado en redes locales de ordenadores que sigan la norma IEEE 802.3: *Ethernet Network Standard*.

8.1.2 Comunicación asincrónica

La comunicación asincrónica se caracteriza por introducir un elemento de sincronización en cada dato transmitido, que consiste en un bit con valor 0 para indicar el comienzo de cada palabra y otro bit con valor 1 para indicar el final de las palabras. El 0 inicial se denomina bit o pulso de inicio (*start*) o *espacio*; el 1 final se denomina bit o pulso de parada (*stop*) o *marca*. Cuando el transmisor hace una pausa porque no tiene palabras para transmitir, mantiene en su terminal de salida una secuencia de bits de parada, es decir, la salida permanece en 1 mientras dura la pausa. El formato de la señal asincrónica se muestra en la figura 8.2, donde puede apreciarse que la sincronización del receptor ocurre en cada dato transmitido.

8.1.3 Comunicación sincrónica

La comunicación sincrónica sin transmisión del reloj se caracteriza por la sincronización de datos por bloques de palabras. A diferencia de la comunicación asincrónica, las palabras no son sincronizadas individualmente, sino que para iniciar la transmisión de un bloque de datos (un conjunto de bytes o de bits), el transmisor introduce un elemento de sincronización, que puede ser una palabra o un patrón único de bits, según el sistema utilizado. Cuando el transmisor termina de enviar un bloque de datos y no hay más datos para enviar, se produce una pausa en la que el transmisor debe mantener la línea en un estado determinado. En general, si hay una pausa entre bloques, se transmite continuamente un bit en 1. Las figura 8.3 ilustra el formato general de la señal en la comunicación sincrónica.

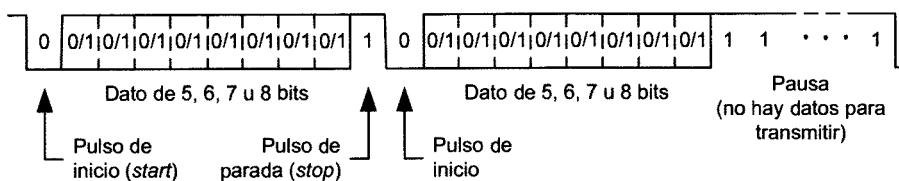


Figura 8.2 Formato de la señal en la transmisión asincrónica. El pulso de inicio siempre tiene la duración de un bit; en cambio el pulso de parada puede tener la duración de 1, 1 $\frac{1}{2}$ ó 2 bits.

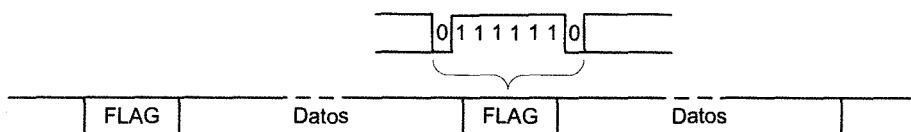


Figura 8.3 Formato de la señal en la comunicación sincrónica. La sincronización se produce al inicio de cada bloque de datos, mediante una secuencia única de bits (FLAG). Como FLAG se utiliza mucho el valor 7Eh. Cada bloque puede estar formado por una secuencia de palabras (bytes) o simplemente ser un conjunto de bits, necesariamente múltiplo de 8.

El elemento de sincronización, muchas veces denominado FLAG, es una secuencia única de bits, que no se repite entre los datos. Este elemento debe comenzar con un bit en 0, para que el receptor pueda determinar que ha finalizado la pausa. Muchas veces se utiliza la palabra 7Eh (01111110), que contiene una secuencia de seis bits en 1. Entonces, para evitar que esta secuencia se repita entre los datos, el transmisor añade previamente un bit 0 a toda secuencia de cinco bits 1, y el receptor lo quita.

En la comunicación asincrónica, al realizarse la sincronización carácter a carácter (con los bits *start* y *stop*), se pierde un 20 % del tiempo de transmisión pues se transmiten 10 bits por cada 8 bits de datos. En cambio, la comunicación sincrónica aprovecha el tiempo de forma más eficiente, pues la sincronización se realiza por bloques de datos y en cuanto el transmisor y el receptor se han sincronizado, se transmiten o reciben sólo datos.

8.1.4 Conexión entre equipos: interfaz RS-232C

El establecimiento de una comunicación a distancia requiere la participación de varios equipos que pueden agruparse en:

- Equipos Terminales de Datos (DTE: *Data Terminal Equipment*). Son los equipos que producen la señal de datos o son los receptores finales de la señal de datos.
- Equipos de Comunicación de Datos (DCE: *Data Communication Equipment*). Son los equipos que adecuan la señal de datos al medio de transmisión utilizado o reciben esta señal del medio de transmisión ofreciéndola de forma apropiada al receptor final.

Un equipo terminal de datos muy común es el ordenador personal, que puede, por ejemplo, generar la señal de datos con el formato asincrónico. Si esta señal ha de transmitirse a otro ordenador a través de un canal telefónico, entonces hay que utilizar algún equipo intermedio para adecuar la señal de datos al canal telefónico en el lado transmisor y viceversa en el lado receptor.

Este equipo es denominado *módem* (*modulator - demodulator*) y está clasificado dentro de los equipos de comunicación de datos.

Según la anterior clasificación de los equipos, el esquema general de un sistema de comunicación de datos es el ilustrado en la figura 8.4.

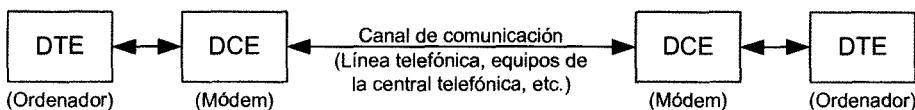


Figura 8.4 Diagrama de bloques muy simplificado de un sistema de comunicación. El equipo terminal de datos (DTE) puede ser un ordenador personal y el equipo de comunicación de datos (DCE) puede ser un módem. En el canal de comunicación puede haber otros DCE.

La conexión entre el DTE y el DCE fue normalizada en el sexto decenio del siglo XX por el entonces llamado Comité Consultivo Internacional de Telefonía y Telegrafía (CCITT), hoy incorporado a la Unión Internacional de Telecomunicaciones (UIT). Una de las normas más utilizadas es la relativa a la interfaz de comunicación en modo asincrónico para velocidades bajas y medias, conocida como la Recomendación V.24 del Libro Blanco de 1969. Esta interfaz es conocida popularmente como RS-232C (*Recommended Standard* número 232 revisión C) pues fue propuesta originalmente en 1962 por la EIA (*Electronic Industries Association*) de los Estados Unidos, para la conexión entre equipos de datos a corta distancia (originalmente menos de 50 pies o 16,4 m) en un entorno ruidoso. La revisión C es del año 1969; se han hecho otras revisiones, como la D en 1986, la E en 1991 y la F en 1997, pero a menudo se mantiene la designación RS-232C a pesar de los cambios. Esta interfaz se ha popularizado de tal forma que hasta hace poco, prácticamente todos los ordenadores personales estaban equipados con una interfaz "RS-232" y su conector para la comunicación con un *módem* u otro periférico serie. Desde 2004, los únicos puertos serie de los ordenadores personales suelen ser USB (*Universal Serial Bus*).

Las señales de la interfaz RS-232C utilizan lógica negativa:

- Nivel lógico 0: entre +3 V y +15 V con carga, hasta +25 V sin carga.
 - Nivel lógico 1: entre -3 V y -15 V con carga, hasta -25 V sin carga.

Las señales más utilizadas de la interfaz RS232C se muestran en la tabla 8.1.

Tabla 8.1 Algunas señales de la interfaz RS-232C. Las señales de datos son RxD y TxD. Las señales restantes se utilizan para controlar la comunicación DTE - DCE o DTE - DTE.

Conector	Nombre de la señal	Sentido de la señal	
		Desde el DCE	Hacia el DCE
25D	Protective Ground (GND)		
9D			
1	Transmitted Data (TxD)		x
2	Received Data (RxD)	x	
3	Request to Send (RTS)		x
4	Clear to Send (CTS)	x	
5	Data Set Ready (DSR)	x	
6	Signal Ground (SG)		
7	Rcvd Line Signal Detect (Data Carrier Detect: DCD)	x	
8	-		
9	-		
10	-		
11	Select Standby		x
12	-		
13	-		
14	-		
15	Transmit Signal Element Timing	x	
16	-		
17	Receiver Signal Element Timing	x	
18	Test	x	
19	-		
20	Data Terminal Ready (DTR)		x
21	-		
22	Ring Indicator (RI)	x	
23	Speed Select		x
24	-		
25	-		

Para conectar equipos mediante la interfaz RS-232C, se utilizan los esquemas de la figura 8.5, aunque hay esquemas para conectar las señales de control de otras formas. La conexión entre un DTE que puede ser un ordenador y un DCE (por ejemplo, un módem), se realiza según el esquema de la figura 8.5a. Para conectar dos equipos terminales de datos entre sí, por ejemplo un ordenador con una impresora o un ordenador con otro ordenador, es común emplear la conexión de la figura 8.5b. Obsérvese que estas dos conexiones son diferentes, pues algunos cables están cruzados. La conexión de la figura 8.5b se denomina *null modem*, aunque suele darse este nombre a cualquier conexión diferente de la mostrada en la figura 8.5a.

8.1.5 El bus I²C

El bus I²C (*Inter-Integrated Circuit*) fue desarrollado por *Philips* para interconectar circuitos integrados de una misma placa de circuito impreso,

utilizando muy pocas líneas para la conexión (tres). Este bus se ha convertido, de facto, en un estándar para la interconexión y transferencia sincrónica de datos en serie entre diferentes dispositivos cercanos: microcontroladores, memorias, convertidores A/D y D/A, etc. Aparte de la línea de masa, en la conexión entre los dispositivos se utilizan sólo dos líneas: una para transferir los datos (SDA: *Serial Data Line*) y otra para la señal de reloj (SCL: *Serial Clock Line*). Se pueden alcanzar velocidades de transferencia de datos elevadas: hasta 100 kbit/s en el modo estándar de baja velocidad (*low-speed mode*), 400 kbit/s en el modo rápido (*fast-mode*) y 3,4 Mbit/s en el modo de alta velocidad (*high-speed mode, hs-mode*). La figura 8.6 ilustra la conexión de varios dispositivos mediante el bus I²C.

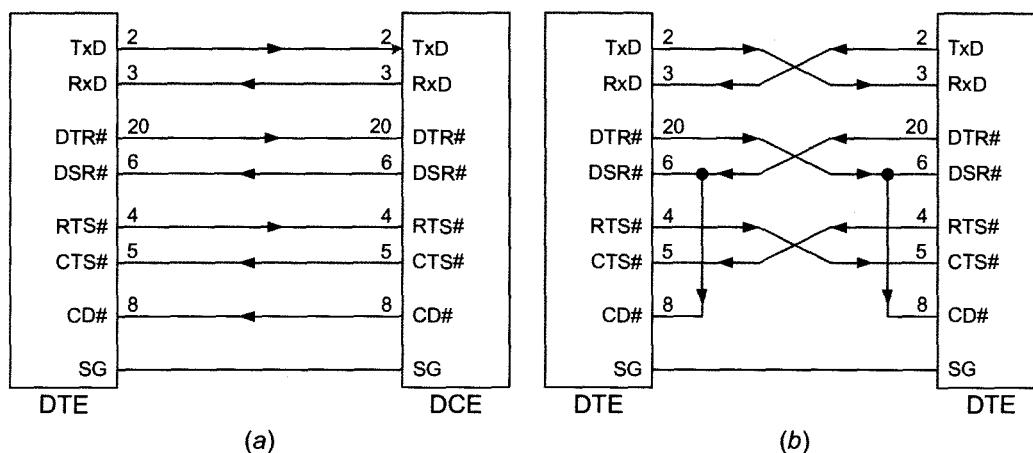


Figura 8.5 Conexión entre equipos utilizando la interfaz RS-232C. (a) Conexión entre un equipo terminal de datos (DTE) y un equipo de comunicación de datos (DCE), que puede ser un módem. (b) Conexión "null modem" entre dos DTE, que pueden ser dos ordenadores, un ordenador y un periférico, dos microcontroladores, etc.

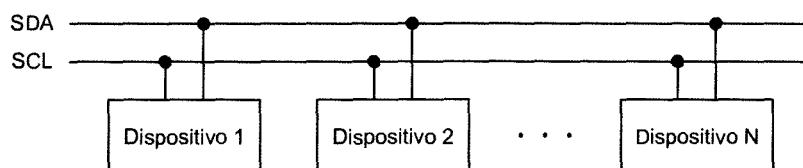


Figura 8.6 Conexión de dispositivos mediante el bus I²C. SDA es la línea de datos y SCL es la de reloj. Los dispositivos pueden ser microcontroladores, memorias, visualizadores, etc. Cada dispositivo se identifica por su dirección. En un momento dado, uno de los dispositivos es servidor y los demás son clientes. La señal de reloj la genera el servidor, que puede ser transmisor o receptor. La señal de datos la genera principalmente el transmisor, que puede ser servidor o cliente. En la figura no se ha representado la conexión a masa de cada dispositivo.

En una comunicación, uno de los dispositivos se comporta como servidor (*master*) y los restantes como clientes (*slaves*). Servidores y clientes pueden ser indistintamente transmisores o receptores. El dispositivo servidor es el que inicia una comunicación, genera la señal de reloj y termina la comunicación. El bus I²C es multi-servidor (*multi-master*), lo que significa que pueden existir varios servidores conectados al bus (aunque en un momento dado, sólo uno actúa como tal). Cada dispositivo tiene una dirección única, que lo identifica durante la comunicación. Las direcciones pueden ser de 7 ó 10 bits, según se explica más adelante.

La figura 8.7 ilustra los circuitos que conectan los dispositivos al bus I²C. Los circuitos de salida son del tipo drenador (o colector) abierto, lo que unido al hecho de que cada línea del bus lleva una resistencia R de *pull-up*, permite realizar la función lógica AND entre las salidas conectadas al bus (*AND cableada*). Para que una línea del bus sea puesta a 1, todos los transistores de salida deben estar cortados; en cambio, cuando un transistor de salida se satura, pone un 0 en la línea correspondiente y con ello ese dispositivo “domina” la línea del bus. Esta es una característica fundamental del bus I²C. Además, los terminales de los dispositivos conectados al bus son, a la vez, entradas y salidas, de modo que cada línea del bus es bidireccional.

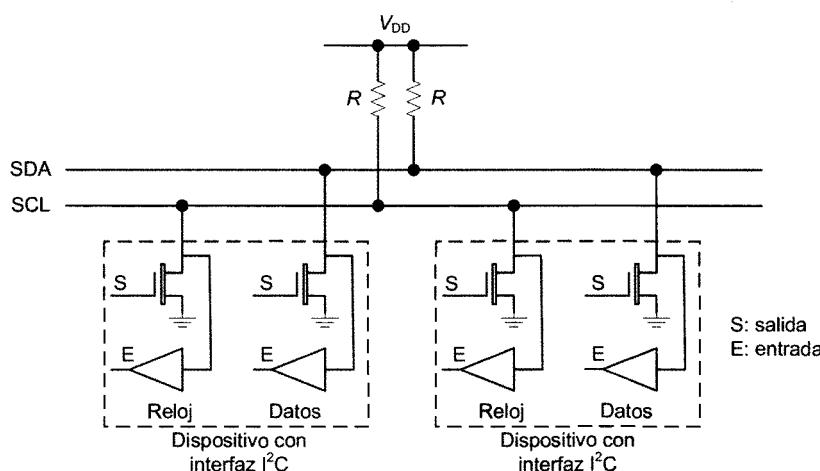


Figura 8.7 Entradas y salidas de los dispositivos conectados al bus I²C. Cada línea del bus es bidireccional. Las salidas de drenador abierto y las resistencias R de *pull-up* permiten la conexión AND cableada en el bus. Cuando un transistor de salida se satura, pone un 0 en la línea correspondiente, dominando la línea frente a los demás dispositivos.

Por la línea SDA del bus se transfieren datos y direcciones de los dispositivos. Toda esta información está organizada en palabras de 8 bits. Cada vez

que se completa la transmisión de un byte por la línea SDA, el receptor debe responder con un bit de reconocimiento (A). Este bit de reconocimiento es un 0 puesto por el receptor en la misma línea SDA durante el siguiente pulso de reloj en SCL (figura 8.8). Para permitir que el receptor coloque el bit de reconocimiento, el transmisor libera la línea SDA al finalizar la transmisión del último bit del dato o dirección, y espera el bit A = 0 en SDA antes de continuar la transmisión de un nuevo byte.

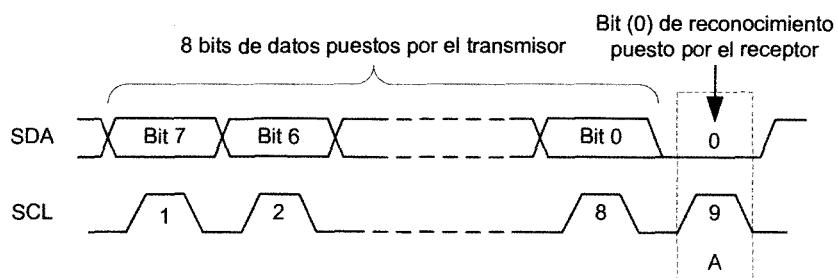


Figura 8.8 La transferencia de datos y direcciones por el bus I²C se organiza en palabras de 8 bits. Cada palabra enviada por el transmisor es sucedida por un bit de reconocimiento (A = 0) puesto por el receptor en la línea SDA durante el noveno pulso de reloj en la línea SCL. Cada bit en SDA debe ser estable mientras el pulso de reloj está en 1 y sólo debe cambiar mientras el reloj está en 0.

Según la figura 8.8, cada bit en SDA se transfiere en sincronismo con la señal de reloj SCL, de modo que mientras la señal de reloj está en 1, el bit en SDA debe permanecer estable (en 0 o en 1) y los cambios de estado en la línea SDA se producen cuando la señal de reloj en SCL está en 0.

La comunicación entre dos o más dispositivos del bus I²C es iniciada y terminada siempre por el dispositivo que funciona como servidor. La comunicación comienza cuando el servidor genera la condición de inicio (S: *start*) y termina cuando el servidor genera la condición de parada (P: *stop*). Ambas condiciones se identifican por una transición en la señal de datos SDA mientras la señal de reloj permanece en 1, tal como ilustra la figura 8.9. La condición de inicio se identifica por una transición de 1 a 0 en la línea SDA mientras SCL está en 1. La condición de parada se identifica, en cambio, por la transición de 0 a 1 de la línea SDA mientras SCL está en 1.

Una vez que el servidor genera la condición de inicio, coloca en la línea SDA la dirección del cliente con el que se quiere comunicar. A partir de ese momento, el servidor indica que va a transmitir o recibir datos con el bit R/W# que transmite al final de la dirección. El bit R/W# en 0 indica que el servidor es un transmisor, mientras que un 1 en R/W# indica que el servidor es

un receptor. Hecha esta indicación, comienza la transferencia de los bytes de datos, siempre sucedidos por el bit de reconocimiento emitido por el receptor. La figura 8.10 ilustra el formato de la comunicación a través del bus I²C en tres situaciones posibles: (a) un servidor transmisor envía datos a un cliente receptor; (b) un servidor receptor recibe datos de un cliente transmisor; y (c) un servidor que inicialmente es un transmisor y luego es receptor. Para pasar de transmisor a receptor, el servidor repite la condición de inicio (S) y envía la dirección del cliente, poniendo el bit R/W# en el valor correspondiente. Independientemente de que el servidor sea transmisor o receptor, la dirección siempre la pone el servidor y, además, éste inicia y termina la comunicación.

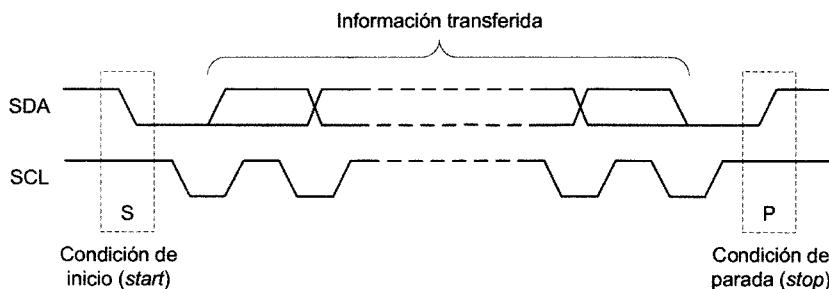


Figura 8.9 Condiciones de inicio (S) y parada (P), con las que el servidor inicia y termina una comunicación.

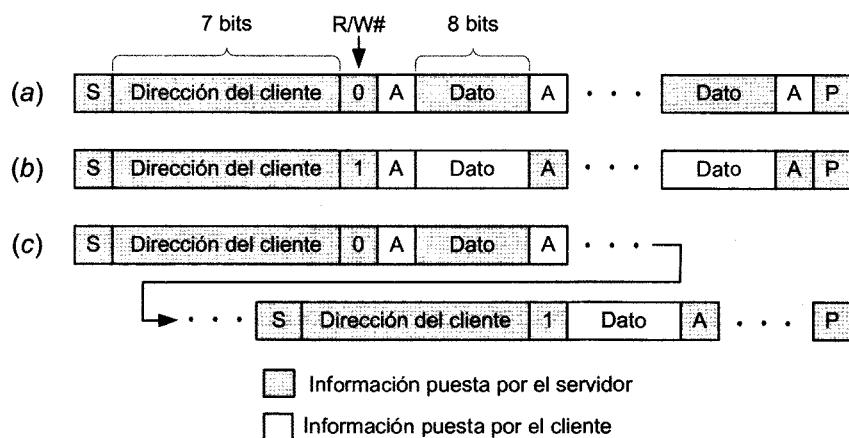


Figura 8.10 Formato de la comunicación en tres posibles situaciones. (a) El servidor transmisor escribe varios datos en el cliente receptor. (b) El servidor receptor solicita varios datos al cliente transmisor. (c) El servidor, inicialmente es transmisor (R/W# = 0) y luego se convierte en servidor receptor, para lo cual repite la condición de inicio y envía la dirección del cliente, poniendo el bit R/W# en 1.

En la versión inicial del bus I²C, las direcciones eran de 7 bits (bits A₆, ..., A₀), con lo que se podían conectar al bus, en principio, hasta 128 dispositivos.

En versiones posteriores, el número de bits de dirección se amplió a 10 (bits A_9, \dots, A_0), para permitir un mayor número de dispositivos conectados al bus. En estas versiones se admiten tanto las direcciones de 7 bits como las de 10 bits. La figura 8.11 ilustra el formato general de dichas direcciones.

En general, las direcciones de 7 bits se dan en un byte único, excepto cuando el servidor realice una “llamada general”, que es una llamada de atención a todos los dispositivos del bus seguida de alguna información que especifica el objetivo de la llamada. En la llamada general se utilizan dos bytes: el servidor emite un primer byte 0, seguido de un segundo byte que precisa el objetivo de la llamada. Las acciones que se pueden realizar se clasifican según el valor del bit menos significativo del segundo byte (bit B).

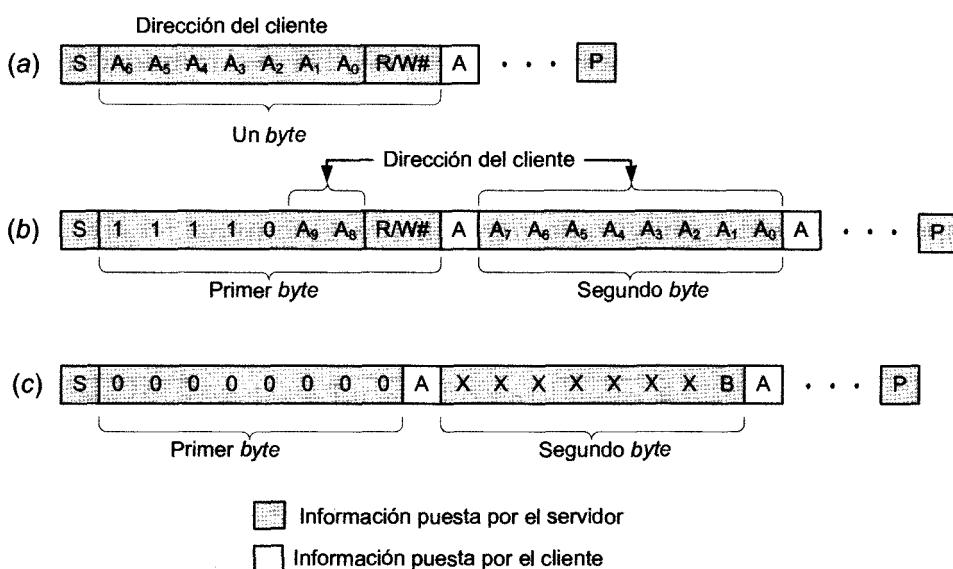


Figura 8.11 Las direcciones de los dispositivos conectados al bus I²C pueden ser de: (a) 7 bits o (b) 10 bits. (c) Formato de una “llamada general”.

Según la figura 8.11, no todas las direcciones están disponibles para su libre uso por los dispositivos conectados al bus I²C: las direcciones de 7 bits 78h a 7Bh (en binario: 11110XX) no se pueden usar para identificar a ningún dispositivo pues esos números están destinados a formar el primer byte de una dirección de 10 bits. Además, hay un número importante de direcciones reservadas para futuros desarrollos o con otros propósitos. Estos y otros detalles del bus I²C están en las especificaciones del bus publicadas por *Philips* (en www.nxp.com).

8.2 El puerto serie USART en los microcontroladores PIC

Los microcontroladores PIC de la gama media poseen un puerto serie para comunicaciones denominado USART (*Universal Synchronous Asynchronous Receiver Transmitter*) o SCI (*Serial Communication Interface*), que puede ser configurado para establecer una comunicación asincrónica bidireccional simultánea (*full duplex*) o sincrónica (con transmisión de la señal de reloj) bidireccional no simultánea (*half duplex*).

8.2.1 Descripción general

El puerto serie USART utiliza los dos terminales TX/CK y RX/DT del microcontrolador, que generalmente comparten funciones con dos terminales del puerto paralelo C. En modo asincrónico, TX/CK es el terminal del transmisor por donde sale la señal de datos y RX/DT es el terminal del receptor por donde entra la señal de datos.

En modo sincrónico, TX/CK es el terminal de salida de reloj si el dispositivo ha sido configurado como servidor, o el terminal de entrada de reloj si el dispositivo ha sido configurado como cliente. El terminal RX/DT es bidireccional y por él entra o sale la señal de datos. La figura 8.12 ilustra el uso de estos terminales en la conexión entre dos microcontroladores PIC.

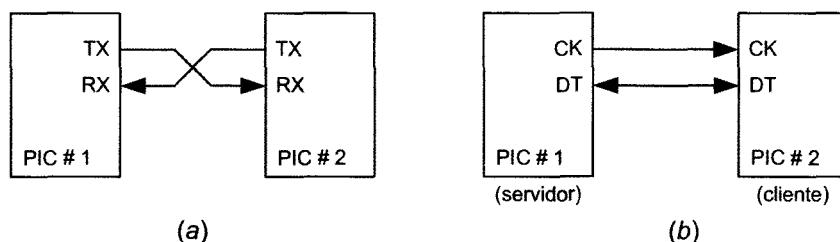


Figura 8.12 Terminales TX/CK y RX/DT utilizados por el puerto serie USART. (a) Conexión entre dos PIC en modo asincrónico. TX y RX son los terminales de datos que se transmiten y reciben, respectivamente. No se transmite la señal de reloj. (b) Conexión en modo sincrónico. CK es el terminal de reloj, que es salida en el dispositivo servidor y entrada en el cliente. DT es el terminal de datos, que es salida en el transmisor y entrada en el receptor.

El puerto serie USART utiliza los registros de funciones especiales TXREG y RXREG para almacenar el dato que se va a transmitir o el dato recibido, respectivamente. También utiliza los registros TXSTA y RXSTA para controlar el puerto y el registro SPBRG para establecer la velocidad de transmisión. Además se usan algunos bits de los registros PIE y PIR para controlar y señalar las solicitudes de interrupción que puede generar el puerto.

8.2.2 Funcionamiento en modo asincrónico

El funcionamiento del puerto serie USART en modo asincrónico se caracteriza porque permite la comunicación bidireccional simultánea de datos (*full duplex*). Esto significa que durante la comunicación entre dos dispositivos USART en modo asincrónico (figura 8.12a), cada dispositivo puede transmitir y recibir datos a la vez.

La señal que se transmite o recibe está constituida por 8 bits precedidos por el bit de inicio o *start* con valor 0 y sucedidos por el bit de parada o *stop* con valor 1. También es posible programar el puerto para que transmita o reciba un noveno bit de datos, tal como muestra la figura 8.13.

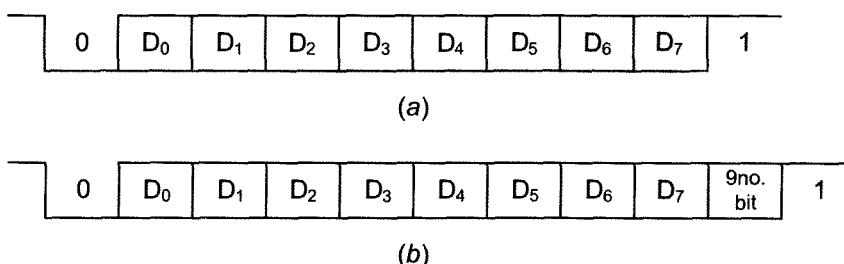


Figura 8.13 Formato de la señal asincrónica en el puerto serie de los PIC de la gama media. (a) Señal de datos. (b) Señal de datos con la adición de un noveno bit.

El puerto serie tiene registros de funciones especiales para manipular los datos que se va a transmitir o recibir: TXREG y RCREG. TXREG es el registro donde debe colocarse el dato que será transmitido por el terminal TX. RCREG es el registro donde queda depositado el dato recibido por el terminal RX. En estos registros no está incluido el noveno bit.

La figura 8.14 es el esquema del transmisor del puerto serie USART. En modo asincrónico y en transmisión, una vez que se vacía el registro TXREG (porque el dato ha pasado al registro de desplazamiento TSR y se está transmitiendo), el bit TXIF del registro PIR se pone a 1, indicando con ello que el puerto está listo para transmitir un nuevo dato (que debe ponerse en TXREG). Si la interrupción por transmisión está habilitada, es decir, TXIE = 1 en el registro PIE, entonces se genera una interrupción con la puesta a 1 del bit TXIF. El bit TXIF pasa a 0 automáticamente cuando se carga un nuevo dato en TXREG.

La figura 8.15 es el esquema del receptor del puerto serie USART. En modo asincrónico y en recepción, el dato recibido por el terminal RX es muestreado a una frecuencia 64 veces mayor que la del reloj y pasa al registro de

desplazamiento RSR, desde donde es colocado en el registro RCREG, en el que puede ser leído por el programa. En realidad hay dos registros para almacenar sendos datos, y que están organizados con una estructura FIFO (*first in first out*), de modo que mientras se está recibiendo un dato en el registro RSR puede haber dos datos más en espera de ser leídos por el programa en RCREG. Cuando hay algún dato en RCREG, el bit RCIF de registro PIR se pone a 1, indicando con ello que se ha recibido un dato. Si el bit RCIE del registro PIE está en 1, se genera una solicitud de interrupción. El bit RCIF pasa a 0 cuando el registro RCREG no contiene ningún dato.

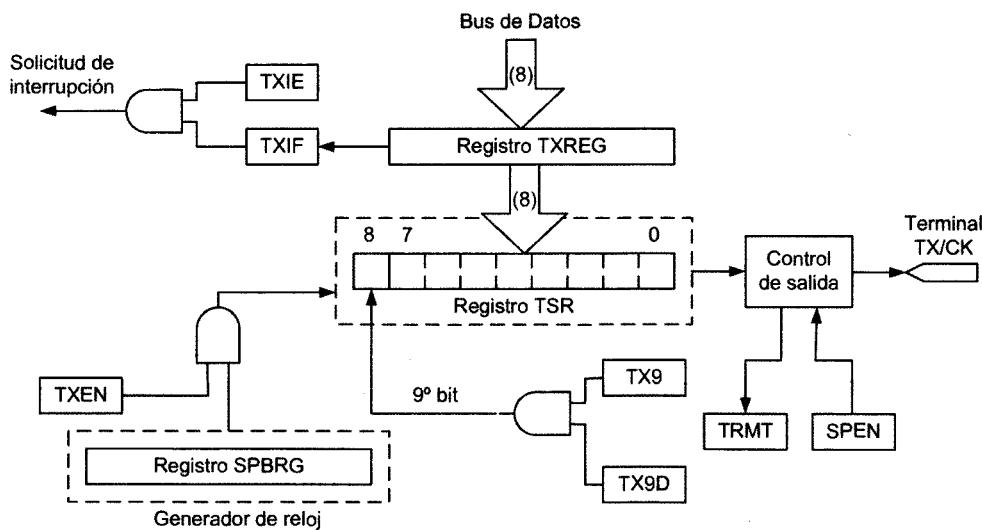


Figura 8.14 Transmisor del puerto serie USART.

El control de la transmisión y la recepción se establece mediante los registros TXSTA y RCSTA, que se muestran en la figura 8.16. Con el registro TXSTA se puede: seleccionar el modo de transmisión entre sincrónico (bit SYNC = 1) o asincrónico (bit SYNC = 0); habilitar la transmisión (bit TXEN = 1); habilitar la transmisión del noveno bit (bit TX9 = 1) y colocar su valor en el bit TX9D; y seleccionar la velocidad de transmisión en modo asincrónico con el bit BRGH (*high baud rate select bit*) como alta (BRGH = 1) o baja (BRGH = 0). El estado del registro de desplazamiento TSR del transmisor se puede conocer mediante el bit TRMT (*transmit shift register status bit*); un 1 en este bit indica que TSR está vacío. El bit CSRC (*clock source select bit*) se utiliza para programar el puerto serie USART como servidor (CSRC = 1) o cliente (CSRC = 0) en modo sincrónico, pero no tiene ningún significado en el modo asincrónico.

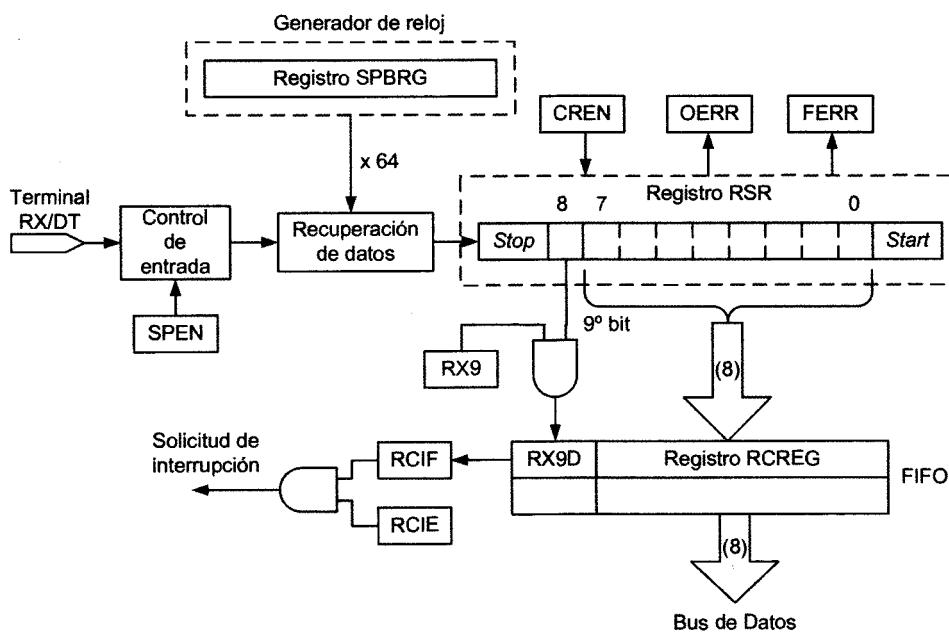


Figura 8.15 Receptor del puerto serie USART.

TXSTA								
7	6	5	4	3	-	2	1	0
CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D	

RCSTA								
7	6	5	4	3	-	2	1	0
SPEN	RX9	SREN	CREN	-	FERR	OERR	RX9D	

Figura 8.16 Registros TXSTA y RCSTA, para controlar la transmisión y recepción de datos por el puerto serie USART, respectivamente.

Con el registro RCSTA se puede: habilitar el puerto serie con el bit SPEN (*serial port enable bit*) puesto a 1, lo cual implica que los terminales RX y TX quedan configurados como terminales del puerto serie; habilitar la recepción de datos con el bit CREN (*continuous receive enable bit*) puesto a 1; y habilitar la recepción del noveno bit con RX9 = 1 y obtener su valor en el bit RX9D. El bit FERR (*framing error bit*) da cuenta de que se ha recibido un dato no válido (el bit de stop no es 0) si toma el valor 1. El bit OERR (*overrun error bit*) pasa a 1 si se ha dejado de leer un dato recibido. El bit SREN (*single receive enable bit*) habilita la recepción de un dato único en modo sincrónico, pero no tiene ningún significado en el modo asincrónico.

8.2.3 Funcionamiento en modo sincrónico

El funcionamiento del puerto serie USART en modo sincrónico se caracteriza por: (a) la comunicación bidireccional no simultánea de datos (*half duplex*); (b) la transmisión (o recepción) simultánea de las señales de datos y de reloj; y (c) la comunicación entre dos dispositivos según un esquema servidor – cliente.

En el modo sincrónico, la transmisión de datos es bidireccional pero no simultánea (*half duplex*). Cada dispositivo puede transmitir y recibir datos, pero las dos operaciones no pueden ser simultáneas: mientras el puerto serie está transmitiendo no puede recibir, y viceversa.

El transmisor y el receptor funcionan según los esquemas de las figuras 8.14 y 8.15, respectivamente. En este modo, la señal de reloj está disponible en un terminal del microcontrolador (TX/CK) junto a la señal de datos (disponible en TX/DT). La señal de datos está formada por la secuencia de bits procedentes de las palabras que se van a transmitir. La figura 8.17 muestra las señales típicas de reloj y de datos manejados por el puerto serie USART en la transmisión sincrónica. Puede observarse que en este caso no hay los bits de inicio y parada característicos de la señal de datos en el modo asincrónico.

El puerto serie USART en modo sincrónico puede funcionar como servidor o como cliente. El servidor es el dispositivo que genera la señal de reloj tanto si es transmisor como si es receptor. El servidor inicia y termina la comunicación con el cliente poniendo o quitando la señal de reloj. El terminal TX/CK sirve como entrada o salida de la señal de reloj: en el servidor es salida y en el cliente es entrada. La figura 8.12b ilustra la conexión entre dos microcontroladores usando los terminales del puerto serie USART en modo sincrónico. Una característica importante es que un dispositivo cliente puede recibir o transmitir un dato incluso cuando está en modo de bajo consumo (*sleep*). En este caso, el PIC despierta y se puede generar una solicitud de interrupción si el bit de habilitación global de las interrupciones así lo permite (GIE = 1).

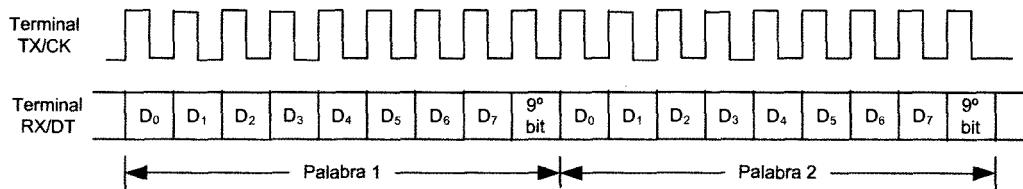


Figura 8.17 Señales de reloj y datos en la transmisión sincrónica por el puerto serie USART.
La transmisión del noveno bit es opcional.

El modo sincrónico del puerto serie USART queda programado poniendo a 1 el bit SYNC del registro TXSTA. Al poner a 1 el bit SPEN del registro RCSTA se habilitan los terminales del puerto serie USART: CK para el reloj y DT para la señal de datos. En la transmisión, una vez que se vacía el registro TXREG porque el dato ha pasado al registro de desplazamiento TSR y se está transmitiendo, el bit TXIF del registro PIR se pone a 1, indicando con ello que el puerto está listo para transmitir un nuevo dato. Si la interrupción por transmisión está habilitada, es decir, TXIE = 1 en el registro PIE, se genera entonces una interrupción con la puesta a 1 del bit TXIF. El bit TXIF pasa a 0 automáticamente cuando se carga un nuevo dato en TXREG. En la recepción, cuando hay algún dato disponible en RCREG, el bit RCIF de registro PIR se pone a 1. Si el bit RCIE del registro PIE está en 1, se genera una solicitud de interrupción. El bit RCIF pasa a 0 cuando el registro RCREG es vaciado (leído) por el programa.

8.2.4 Velocidad de la comunicación.

El reloj que determina la velocidad de transmisión por el puerto serie se deriva del oscilador principal del microcontrolador. Su valor se ajusta mediante el registro SPBRG y el bit BRGH del registro TXSTA. La velocidad de transmisión en el modo asincrónico es

$$v_T = \frac{4^{BRGH}}{64 \times (SPBRG + 1)} \times f_{osc} \quad (8.2)$$

donde v_T es la velocidad de transmisión (bit/s), f_{osc} es la frecuencia en hertz del oscilador principal del microcontrolador, BRGH es 0 ó 1 y SPBRG es un número entre 0 y 255. La tabla 8.2 da algunos valores calculados mediante (8.2). El error se ha calculado mediante

$$\text{Error}(\%) = \frac{v_{T, \text{calculada}} - v_{T, \text{deseada}}}{v_{T, \text{deseada}}} \times 100 \quad (8.3)$$

Si el puerto serie USART ha sido programado en modo sincrónico, la velocidad de transmisión se determina mediante

$$v_T = \frac{f_{osc}}{4 \times (SPBRG + 1)} \quad (8.4)$$

En este caso no interviene el bit BRGH del registro TXSTA.

Tabla 8.2 Valores que deben colocarse en el bit BRGH y en el registro SPBRG para obtener, en el modo asincrónico, las velocidades de comunicación deseadas, según la frecuencia del oscilador principal del microcontrolador, y error cometido en la velocidad en cada caso.

$V_T, \text{deseada}^a /(\text{bit/s})$	$f_{\text{osc}}/\text{MHz}$	BRGH	SPBRG (decimal)	$V_T, \text{calculada}^a /(\text{bit/s})$	Error/%
1200	4	0	51	1201,92	0,16
1200	16	0	207	1201,92	0,16
2400	4	0	25	2403,85	0,16
2400	16	0	103	2403,85	0,16
9600	16	0	25	9615,38	0,16
19200	4	1	12	19230,77	0,16
19200	16	1	51	19230,77	0,16
1200	5,0688	0	65	1200,00	0
2400	5,0688	0	32	2400,00	0
9600	5,0688	1	32	9600,00	0

Ejemplo 8.1

Programación básica del puerto serie USART de un PIC16F873 en modo asincrónico. La frecuencia del oscilador principal del PIC es de 4 MHz.

La programación tiene tres partes: La primera es iniciar el puerto serie en el modo asincrónico, estableciendo una velocidad de transmisión determinada; la segunda es elaborar una subrutina para transmitir un dato; y la tercera parte es elaborar una subrutina de recepción de un dato. Se usará la técnica de entrada y salida por consulta.

A continuación se muestra el listado de instrucciones de las subrutinas para la operación del puerto serie:

```

;
; INIC_SCI: Rutina de iniciación del puerto serie USART en modo asincrónico,
; 19200 bit/s, habilitando transmisión y recepción por espera.
;
INIC_SCI:
    bsf      STATUS, RP0      ; Seleccionar el banco 1.
    movlw   0Ch                ; Velocidad de transmisión 19200 bit/s.
    movwf   SPBRG
    movlw   0CFh              ; Programar los terminales RC7/RX como entrada
    movwf   TRISC              ; y RC6/TX como salida.
    movlw   24h                ; Modo asincrónico, 8 bits de datos,
    movwf   TXSTA              ; transmisión habilitada, BRGH = 1.
    bcf     PIE1, TXIE         ; Interrupción por transmisión inhabilitada.
    bcf     PIE1, RCIE         ; Interrupción por recepción inhabilitada.
    bcf     STATUS, RP0        ; Seleccionar banco 0.
    movlw   90h                ; Recepción habilitada, 8 bits de datos.
    movwf   RCSTA              ; USART listo para transmitir y recibir datos.
    return
;
```

```

; TXDATA: Rutina de transmisión de un dato de 8 bits por espera.
; El dato que se va a transmitir debe estar en el registro W.
;
TXDATA:
    btfss    PIR1, TXIF      ;¿TXIF=1?
    goto    TXDATA          ;No, esperar.
    movwf   TXREG           ;Sí, poner el dato en TXREG.
    return
;

; RCDATO: Rutina de recepción de un dato de 8 bits por espera.
; El dato recibido es devuelto en el registro W.
;
RCDATO:
    btfss    PIR1, RCIF      ;¿RCIF=1?
    goto    RCDATO          ;No, esperar.
    movf    RCREG, W         ;Sí, leer el dato.
    return
        ;Retornar con el dato en W.

```

8.3 El puerto serie SSP en los microcontroladores PIC

Los microcontroladores PIC de la gama media poseen un puerto serie para comunicaciones sincrónicas a corta distancia con transmisión de la señal de reloj, del cual existen dos versiones denominadas SSP (*Synchronous Serial Port*) y MSSP (*Master Synchronous Serial Port*). Cualquiera de estos puertos puede ser configurado para trabajar como interfaz sincrónica serie SPI (*Serial Peripheral Interface*) o como interfaz I²C (*Inter-Integrated Circuit*). En el puerto SSP, la interfaz I²C puede operar sólo como cliente, mientras que en el puerto MSSP se puede programar como servidor o como cliente. Tanto la interfaz SPI (implementada inicialmente en los microcontroladores 68HCxx de *Motorola*) como la I²C, fueron desarrolladas para la comunicación entre dispositivos cercanos (microcontroladores, memorias externas, visualizadores, convertidores A/D, etc.) mediante un número pequeño de líneas de interconexión. A continuación se estudia el funcionamiento de estas dos modalidades del puerto serie sincrónico en los microcontroladores PIC.

8.3.1 Interfaz SPI

El puerto serie SSP (o MSSP) programado como interfaz SPI realiza la transmisión y recepción sincrónica y simultánea de datos de 8 bits. La interfaz SPI puede funcionar como servidor (*master*) o cliente (*slave*). El servidor es el dispositivo que genera la señal de reloj, sea transmisor o receptor. Para esta comunicación se utilizan fundamentalmente tres terminales del microcontrolador, que generalmente comparten funciones con terminales del puerto paralelo C. Estos terminales son: SDO y SDI para la salida y entrada de

datos respectivamente y SCK para el reloj. El terminal de reloj SCK es salida en el servidor y entrada en el cliente. Un cuarto terminal, denominado SS# (*slave select*) se puede utilizar opcionalmente para que un dispositivo servidor seleccione uno de varios dispositivos programados como clientes. La figura 8.18 ilustra el uso de estos terminales en la conexión entre dos microcontroladores PIC.

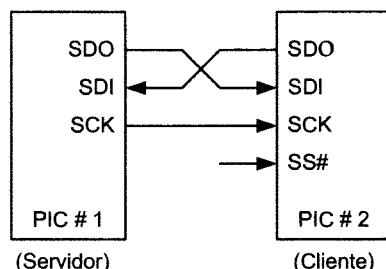


Figura 8.18 Conexión entre dos microcontroladores PIC utilizando la interfaz SPI.

La interfaz SPI utiliza el registro de funciones especiales SSPBUF para almacenar transitoriamente el dato que se va a transmitir o el dato recibido. También utiliza los registros SSPCON y SSPSTAT para controlar la interfaz. Además se usan, respectivamente, los bits SSPIE y SSPIF de los registros PIE y PIR para el control y señalización de las solicitudes de interrupción que puede generar la interfaz.

La figura 8.19 muestra el uso de los bits de los registros SSPCON y SSPSTAT por parte de la interfaz SPI. Estos registros también son utilizados para controlar la interfaz I²C, pero entonces algunos bits cambian su significado respecto al que tienen en la interfaz SPI.

SSPCON								
7	6	5	4	3	2	1	0	
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	

SSPSTAT								
7	6	5	4	3	2	1	0	
SMP	CKE	0	0	0	0	0	BF	

Figura 8.19 Registros SSPCON y SSPSTAT para controlar la transmisión y recepción de datos con la interfaz SPI.

El bit SSPEN (*SSP enable bit*) asigna los terminales SCK, SDO, SDI y SS# al puerto serie sincrónico, aunque deben ser definidos como entradas o salidas mediante la puesta a 1 o a 0 de los bits correspondientes del registro

TRIS. Los bits SSPM3:SSPM0 del registro SSPCON programan el dispositivo como servidor o cliente, habilitan o no el uso del terminal SS# como control del cliente, y seleccionan la frecuencia de reloj de la interfaz SPI. En el servidor, la señal de reloj en SCK puede ser una fracción (1/4, 1/16 ó 1/64) de la frecuencia del oscilador principal del microcontrolador o puede obtenerse a partir del Timer2. El bit WCOL (*write collision detect bit*) del registro SSPBUF informa si se ha producido una colisión en el registro SSPBUF, es decir, si se ha intentado escribir un dato en SSPBUF cuando aún estaba el dato anterior en el registro. El bit SSPOV (*receive overflow indicador bit*) indica si se ha dejado de leer un dato recibido y depositado en SSPBUF. El bit CKP (*clock polarity select bit*) programa el estado (0 ó 1) de la señal de reloj cuando no hay datos que transmitir.

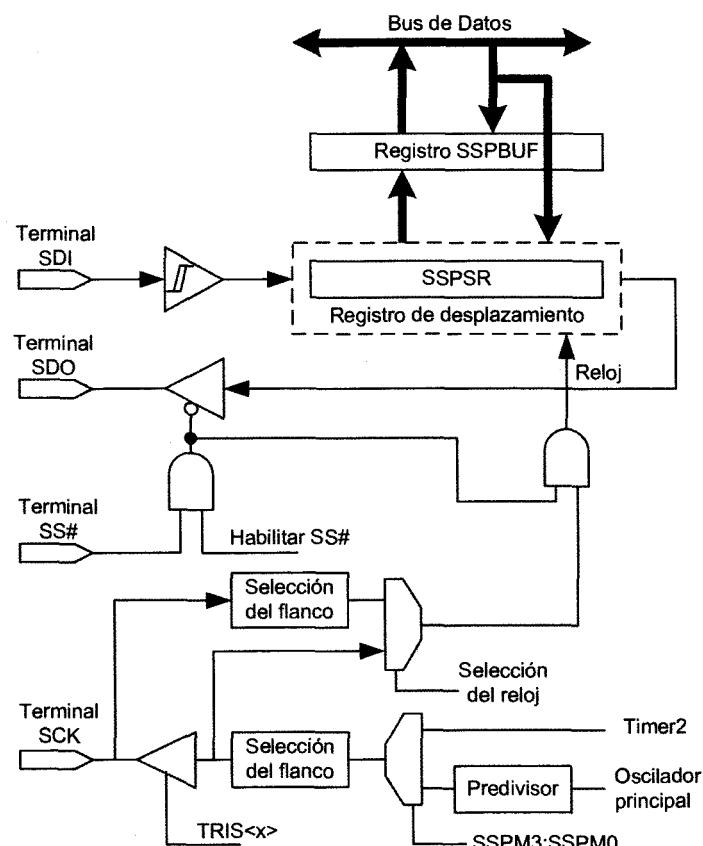


Figura 8.20 Diagrama de bloques de la interfaz SPI.

La interfaz SPI opera de la siguiente forma. En el servidor, la transferencia se inicia con la escritura de un dato en el registro SSPBUF. Cuando sucede esto, el dato pasa al registro de desplazamiento SSPSR (figura 8.20) y

comienza a ser transmitido de inmediato por el terminal SDO; simultáneamente comienza la recepción de un dato por SDI. Al completarse la recepción del dato, el bit BF (*buffer full status bit*) del registro SSPSTAT es puesto a 1 y el dato recibido queda disponible en el registro SSPBUF. Además, el bit SSPIF del registro PIR se pone a 1, con lo que se puede generar una solicitud de interrupción si la interrupción del puerto SSP está habilitada (el bit SSPIE del registro PIE está en 1). Una vez que el dato es extraído del registro SSPBUF, se puede escribir un nuevo dato en ese registro, con lo que continúa el proceso de transmisión y recepción de datos.

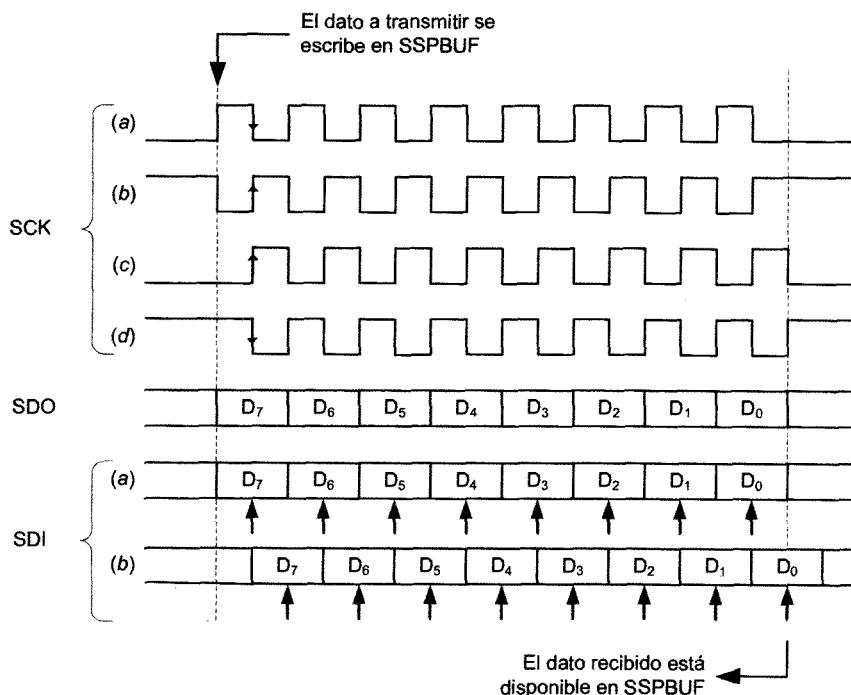


Figura 8.21 Señales de la interfaz SPI del dispositivo programado como servidor. La transmisión se inicia cuando el dato que se va a transmitir se escribe en el registro SSPBUF. Se pueden programar cuatro variantes de la señal de reloj SCK: en (a) y (c) el estado inactivo de SCK es 0, mientras que en (b) y (d) es 1. También se programa el flanco de la señal de reloj al que está sincronizada la señal de datos, lo cual se ha indicado con una flecha en la señal SCK. En (a) y (d) la señal de datos es estable con el flanco de bajada de SCK, mientras que en (b) y (c) lo es con el flanco de subida. La señal de datos recibida en SDI se puede muestrear en instantes de tiempo situados (a) a mitad de cada bit transmitido o (b) al final de cada bit transmitido. Al muestrear el último bit de la señal de entrada, el dato recibido queda disponible en el registro SSPBUF.

La figura 8.21 muestra las señales de la interfaz SPI en modo servidor. Si no hay datos para transmitir, la señal de reloj en SCK permanece en el estado

inactivo (*idle state*). El estado inactivo puede ser 0 ó 1 según se programe con el bit CKP del registro SSPCON. Los bits del dato se pueden transmitir sincrónicamente con los flancos de subida o bajada de la señal de reloj, lo cual se programa mediante el bit CKE (*SPI clock edge select bit*) del registro SSPSTAT. El muestreo de la señal de datos en SDI se puede hacer a mitad o al final de cada bit transmitido, lo cual se programa con el bit SMP (*SPI data input sample phase bit*) del registro SSPSTAT.

En el dispositivo que ha sido programado como cliente, la transmisión y la recepción de un dato se inician cuando se recibe la señal de reloj. El dato que se va a transmitir debe estar depositado en SSPBUF y al finalizar su transmisión, el dato recibido durante ese mismo tiempo queda disponible en el mismo registro SSPBUF, donde puede ser leído. La recepción del dato se señala con la puesta a 1 del bit BF del registro SSPSTAT. También el bit SSPIF del registro PIR se pone a 1, con lo que se puede generar una solicitud de interrupción si la interrupción del puerto SSP está habilitada (el bit SSPIE del registro PIE está en 1).

Si el dispositivo cliente está en el modo de bajo consumo y le llega la señal de reloj, entonces recibe un dato y transmite el dato que esté en SSPBUF. En este caso, el bit SSPIF del registro PIR pasa a 1 y si la interrupción está habilitada (el bit SSPIE del registro PIE está en 1), el PIC la atiende y sale del modo de bajo consumo.

Ejemplo 8.2

Transmisión y recepción de datos por el puerto SPI de un PIC16F873. La frecuencia del oscilador principal del PIC es de 4 MHz.

Se presenta la programación del puerto SPI con los parámetros siguientes: tratamiento del puerto por espera o consulta, modo servidor, frecuencia de reloj igual a $F_{osc}/16$, el estado inactivo de la señal de reloj SCK es 1, y el dato se transmite con el flanco de bajada de SCK (caso (d) en la figura 8.21). Se presenta también el listado de una rutina para recibir y transmitir un dato. Se supone que el dato que se va a transmitir debe ser depositado previamente en el registro DATOTX y el dato recibido es devuelto en el registro DATORX.

A continuación se da el listado de las subrutinas.

```
;  
;  
; Programación del puerto serie SPI. Fosc de 4 MHz.  
;  
list p=16f873  
#include <p16f873.inc>  
;  
DATOTX      equ    20h      ; Dato que se va a transmitir.  
DATORX      equ    21h      ; Dato recibido.
```

```

; ; INIC_SPI: Rutina de iniciación del puerto serie SPI,
;

INIC_SPI:
    bsf    STATUS, RP0      ; Se selecciona el banco 1.
    movlw  40h              ; SMP=0, CKE=1, BF=0.
    movwf  SSPSTAT
    movlw  0D7h             ; Programar terminales RC5/SDO y RC3/SCK como salidas
    movwf  TRISC            ; y RC4/SDI como entrada.
    bcf    STATUS, RP0      ; Se selecciona el banco 0.
    movlw  31h              ; SSPEN=1, CKP=1, SPI master, reloj Fosc/16 (250 kbit/s).
    movwf  SSPCON
    bsf    STATUS, RP0      ; Se selecciona el banco 1.
    bcf    PIE1, SSPIE       ; La interrupción del SPI es inhabilitada.
                           ; El SPI está listo para transmitir y recibir datos.
    bcf    STATUS, RP0      ; Se selecciona el banco 0
    movf   DATOTX, W        ; y se transmite el
    movwf  SSPBUF           ; el primer dato.
    return

; ; TRANSFER: Rutina de transmisión y recepción de un dato de 8 bits por espera.
; Entradas: el dato a transmitir debe estar en DATOTX.
; Salidas: el dato recibido es devuelto en DATORX.

TRANSFER:
    bsf    STATUS, RP0      ; Se selecciona el banco 1
ESPERA:
    btfss  SSPSTAT, BF      ; ¿Dato recibido?
    goto   ESPERA           ; No – esperar.
    bcf    STATUS, RP0      ; Si – seleccionar banco 0 y
    movf   SSPBUF, W        ; se lee el dato recibido, se pone en W y
    movwf  DATORX           ; se deposita finalmente en DATORX.
    movf   DATOTX, W        ; El dato que se va a transmitir se coloca en
    movwf  SSPBUF           ; SSPBUF, con lo que se inicia la transmisión
                           ; (y recepción) de un nuevo dato.
    return
end

```

8.3.2 Interfaz I²C

El puerto SSP puede funcionar como interfaz I²C en modo cliente. Para realizar una interfaz I²C que funcione como servidor, se requiere un puerto MSSP en el microcontrolador. En un puerto MSSP, la interfaz I²C se puede configurar como servidor o como cliente. La interfaz dispone de los termina-

les SDA y SCL para datos y reloj, respectivamente, que normalmente comparten funciones con terminales del puerto C.

La figura 8.22 es el diagrama de bloques de la interfaz I²C. Un elemento clave en la interfaz es el registro de desplazamiento SSPSR. Cada byte transmitido o recibido, sea un dato o una dirección o parte de ella (en las direcciones de 10 bits), transita por ese registro. La comunicación entre este registro y el programador se realiza a través del registro de funciones especiales SSPBUF. Cualquier dato o dirección que se va a transmitir al bus debe ser colocado en SSPBUF. Cualquier dato o dirección que se reciba por el bus puede ser leído en SSPBUF. El registro SSPSR es invisible para el programador. Cada vez que se transmite o recibe un byte, el bit SSPIF del registro PIR es puesto a 1, lo que puede generar una solicitud de interrupción si el bit SSPIE del registro PIE está en 1.

La interfaz cuenta también con el registro SSPADD, que realiza funciones diferentes en los modos servidor y cliente. Si el dispositivo es cliente, el registro de funciones especiales SSPADD almacena la dirección del dispositivo. Ante una llamada del servidor, que debe poner la dirección del cliente en el bus, se compara la dirección enviada por el servidor con la dirección del cliente, la cual está en SSPADD. Si ambas direcciones son iguales, el cliente realiza alguna acción para responder a la llamada del servidor y establecer la comunicación. La acción puede ser simplemente generar una solicitud de interrupción (el bit SSPIF del registro PIR se pone a 1).

En un dispositivo que opera como servidor, el registro SSPADD sirve para fijar la frecuencia de la señal de reloj que genera el servidor. En este caso, la dirección, que el servidor debe poner en el bus para llamar a un cliente, llega al registro de desplazamiento SSPSR a través del registro de funciones especiales SSPBUF.

El dispositivo servidor cuenta también con circuitos para generar las condiciones de inicio (S) y parada (P). En un dispositivo cliente, sólo hay circuitos para detectar estas condiciones. El control y observación de estos circuitos se realiza mediante los registros de funciones especiales SSPSTAT y SSPCON2.

En un dispositivo servidor, la frecuencia de la señal de reloj en la línea SCL se obtiene mediante la expresión:

$$f_{\text{SCL}} = \frac{f_{\text{osc}}}{4 \times (\text{SSPADD} + 1)} \quad (8.5)$$

donde f_{SCL} es la frecuencia (en hertz) del reloj en la línea SCL del bus, f_{osc} es la frecuencia (en hertz) del oscilador principal del microcontrolador, y SSPADD es el número formado por los 7 bits menos significativos del registro SSPADD (SSPADD<6:0>).

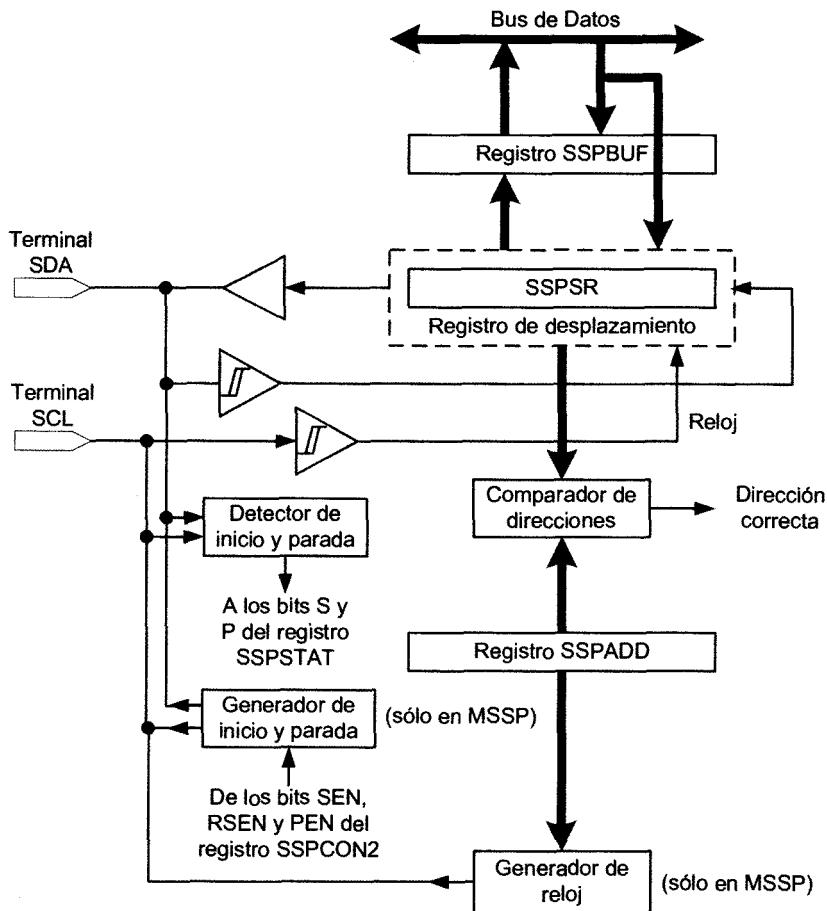


Figura 8.22 Diagrama de bloques de la interfaz SSP en el modo I²C. Los datos y direcciones se transfieren hacia o desde el bus mediante el registro SSPBUF. El registro de desplazamiento SSPSR es invisible para el programador. El registro SSPADD almacena la dirección del dispositivo cliente o la velocidad de la comunicación si el dispositivo es servidor. Los generadores de las condiciones de inicio y parada, así como el generador de reloj sólo existen en los dispositivos que son servidores.

La figura 8.23 muestra los bits de los registros SSPCON, SSPSTAT y SSPCON2, según son utilizados por la interfaz I²C. El registro SSPCON2 sólo existe en los dispositivos que pueden ser servidores, es decir, que tienen un puerto del tipo MSSP.

Los bits SSPM3:SSPM0 del registro SSPCON programan el dispositivo como interfaz I²C servidor o cliente, con dirección de 7 ó 10 bits. El bit SSPEN (*SSP enable bit*) asigna los terminales SDA y SCL al interfaz I²C o al puerto paralelo C. El bit WCOL (*write collision detect bit*) del registro SSPCON informa si se ha producido una colisión en el registro SSPBUF, es decir, si se ha intentado escribir un dato en SSPBUF cuando el dato anterior aún estaba en el registro. El bit SSPOV (*receive overflow indicador bit*) indica si se ha dejado de leer un dato recibido y depositado en SSPBUF. El bit CKP (*clock polarity select bit*) se utiliza para el control del reloj en el cliente.

En el registro SSPSTAT, los bits S (*start bit*) y P (*stop bit*) indican la detección de las condiciones de inicio y parada, respectivamente. R/W# (*read/write bit information*) es el bit que acompaña a la dirección, e indica si el cliente es receptor o transmisor. UA (*update address*) se utiliza sólo con direcciones de 10 bits, indicando que se debe actualizar la dirección del cliente en el registro SSPADD. El bit BF (*buffer full status bit*) indica si el registro SSPBUFF está lleno o vacío. D/A# (*data/address bit*) indica si el último byte transmitido o recibido fue un dato o una dirección. El bit SMP (*sample bit*) permite establecer el control de la velocidad de transición (*slew rate*) de las señales en el bus en el modo de alta velocidad (400 kHz).

SSPCON							
7	6	5	4	3	2	1	0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0

SSPSTAT							
7	6	5	4	3	2	1	0
SMP	CKE	D/A#	P	S	R/W#	UA	BF

SSPCON2							
7	6	5	4	3	2	1	0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN

Figura 8.23 Registros SSPCON, SSPSTAT y SSPCON2 utilizados en el control de la transmisión y recepción de datos por la interfaz I²C. El registro SSPCON2 sólo existe en los dispositivos que tienen un puerto del tipo MSSP.

El registro SSPCON2 se usa para controlar la interfaz I²C configurada como servidor. Los bits SEN (*start condition enable bit*), RSEN (*repeated start condition enable bit*) y PSEN (*stop condiction enable bit*) generan las condiciones de inicio, inicio repetido y parada. RCEN (*receive enable bit*) habilita la

recepción del servidor. ACKEN (*acknowledge sequence enable bit*) habilita la generación de un bit de reconocimiento (A); el valor de este bit debe estar en ACKDT (*acknowledge data bit*). ACKSTAT (*acknowledge status bit*) informa qué bit de reconocimiento se ha recibido. GCEN (*general call enable bit*) habilita la solicitud de interrupción por una llamada general.

9 Las entradas y salidas analógicas. Adquisición y distribución de señales

En los capítulos anteriores se ha explicado cómo los microcontroladores pueden adquirir, procesar y generar señales digitales, por ejemplo para comunicarse con otros circuitos y subsistemas. En este capítulo se estudia cómo se pueden adquirir y generar señales analógicas empleando módulos o dispositivos integrados en el propio microcontrolador, o en circuitos periféricos, y se dan los criterios básicos para diseñar el sistema.

Las señales analógicas tienen la información en su amplitud o duración, de modo que para procesarlas con circuitos digitales, primero hay que digitalizarlas y esto exige adaptar sus características a las de los digitalizadores. Además, dado que los terminales dispuestos para las entradas y salidas analógicas pueden tener conexiones a puntos relativamente alejados del microcontrolador, hay que prever la instalación de protecciones que eviten los daños que producirían las tensiones o corrientes excesivas.

9.1 Funciones y estructura de un sistema de adquisición y distribución de señales

Las señales analógicas pertenecen a sistemas de medida y control, y también están presentes en las comunicaciones con intervención humana, por ejemplo mediante micrófonos, altavoces o cámaras.

9.1.1 Funciones básicas en los sistemas de medida y control

Las señales que se desea medir son normalmente magnitudes físicas o químicas no eléctricas. Las magnitudes que se desea controlar (como la temperatura de una sala o la posición de un cabezal de impresora), o las señales empleadas para comunicar un resultado de medida al usuario, tampoco son magnitudes eléctricas. Por ello, las funciones básicas en un sistema de medida son: detectar la magnitud (con un *sensor*), procesar la información, y comunicarla (al usuario u otra máquina). Cuando el sistema es electrónico, a estas tres funciones hay que añadirles el suministro de una alimentación eléctrica, y el control u organización de las funciones, según se indica en la figura 9.1. El procesador está basado en un dispositivo digital, tal como un microcontrolador, que puede realizar también las funciones de control. Si el objetivo del sistema es controlar una magnitud, tras la medida y el procesamiento hay una acción con un *actuador*, que convierte una señal eléctrica en

la acción física deseada; por ejemplo, poner en marcha un calefactor o activar un motor.

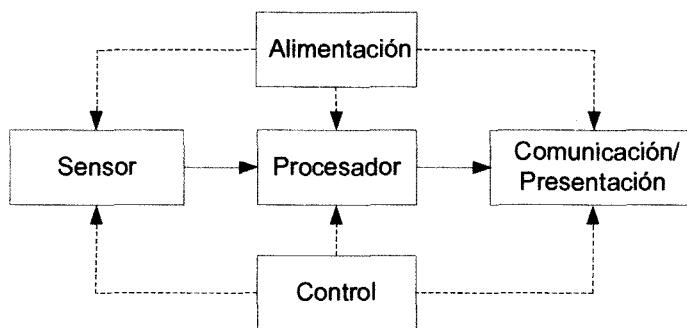


Figura 9.1 Funciones básicas en un sistema de medida. Las fluctuaciones de la alimentación y las señales de control pueden repercutir (indebidamente) en las señales de salida de cada bloque.

Las señales de salida de los sensores suelen ser analógicas y de baja amplitud. Sólo algunos sensores digitales, como los codificadores de posición, ofrecen una salida digital, que si tiene los niveles de tensión adecuados se puede conectar directamente a un puerto de entrada del microcontrolador. Para digitalizar las señales analógicas se utiliza un *convertidor analógico-digital* (CAD). Para adaptar la señal de salida del sensor al rango de amplitudes de entrada del CAD, se emplea un *acondicionador de señal* y, en algunos casos, un procesador analógico (por ejemplo, para obtener el valor eficaz de la señal antes de digitalizarla, o para desmodular su amplitud). La salida del CAD es procesada para obtener la información deseada, por ejemplo el valor medio de una señal durante un cierto tiempo, o bien se combina la información de varios sensores para tomar una decisión, por ejemplo sobre la presencia o ausencia de un intruso en un recinto. La cadena de bloques principales de la figura 9.1 se puede ampliar según muestra la figura 9.2. Globalmente hay, pues, dos tipos de funciones: las de conversión (de una magnitud física en una señal eléctrica en el sensor; de una tensión analógica en un código digital, en el CAD), y las de adaptación de la señal de salida de una etapa a las características de entrada de la etapa siguiente, como sucede en los acondicionadores de señal.

La conversión analógico-digital (A/D) es en esencia la comparación de una tensión desconocida V_x con una tensión de referencia, V_{ref} (figura 9.3). En la denominada conversión A/D directa, la comparación se realiza entre V_x y fracciones de V_{ref} de valor $L \times V_{\text{ref}}/2^N$, donde L y N son números enteros. Esta comparación se puede hacer de forma simultánea con todos los valores entre

0 V y V_{ref} (*convertidores flash* o paralelos), o de forma sucesiva con valores fraccionarios elegidos en un orden que agilice el proceso de decisión (*convertidores de aproximaciones sucesivas*). Los CAD integrados en microcontroladores suelen ser de aproximaciones sucesivas. La característica de transferencia de la conversión se muestra en la figura 9.3 (para $N = 3$).

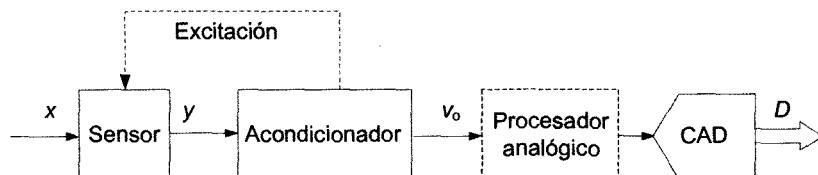


Figura 9.2 La etapa frontal en la adquisición de señales: un acondicionador de señal adapta la señal de salida del sensor al rango de entrada del CAD. Un procesador analógico puede extraer un parámetro de la señal antes de digitalizarla, por ejemplo, su valor eficaz, o puede desmodular su amplitud.

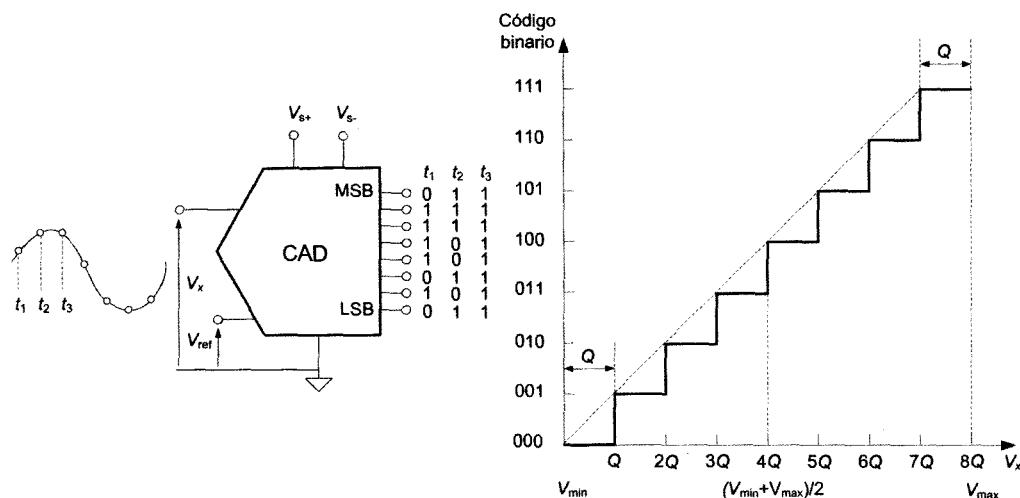


Figura 9.3 El proceso de conversión analógica-digital (directa) y su característica de transferencia.

En la conversión A/D indirecta, un circuito genera, por ejemplo, un intervalo de tiempo de duración proporcional a la tensión de entrada, y dicha duración se compara con la de un intervalo de tiempo generado a partir de la tensión de referencia y el mismo circuito. Ambos intervalos se miden con el mismo contador digital. Otros convertidores obtienen una frecuencia proporcional a la tensión de entrada, y luego miden la frecuencia con un contador digital. Aquellos sensores que ofrecen una salida cuya información está en

la frecuencia, periodo, intervalo de tiempo, ancho de pulso, ciclo de trabajo, fase, etc. se denominan *casidigitales* (o *cuasidigitales*), porque, aunque su salida no es digital, basta un contador para obtener un código que represente la información de entrada; es decir, el propio sensor realiza parte de la conversión A/D indirecta. Dado que el resultado de contar es un número entero, la característica de transferencia de la conversión indirecta es también la de la figura 9.3, que se puede describir mediante

$$D_x = \text{ent}\left(\frac{V_x}{V_{\text{ref}}} 2^N\right) + 1 \quad (9.1)$$

donde D_x es el número de orden del código de salida (entre 1 y 2^N), $\text{ent}(a)$ designa el mayor entero menor o igual que a , $V_x < V_{\text{ref}}$, y N es el número de bits del convertidor. En el código binario natural, sin signo, el primer código es 000...0 y el código superior es 111...1. Si el valor de V_{ref} se puede seleccionar, como sucede en algunos microcontroladores, el rango de tensiones de entrada variará según la selección.

La forma escalonada de la característica de transferencia de un CAD significa que todas las tensiones de entrada que caen dentro de un intervalo $[V_u, V_u + Q]$ son asignadas al mismo código; V_u es cualquiera de las tensiones umbral a partir de las cuales el código asignado es otro, y Q es el denominado *intervalo de cuantificación*,

$$Q = \frac{V_{\text{ref}}}{2^N} \quad (9.2)$$

En un CAD, $Q = 1$ LSB. La cuantificación conlleva por una parte una incertidumbre, por cuanto a partir de un código de salida no se puede saber a ciencia cierta cuál ha sido la tensión de entrada que lo ha producido. Por otra parte, la cuantificación determina el menor cambio de tensión que el sistema puede detectar, es decir, su *resolución*.

9.1.2 Margen o rango dinámico

Las etapas necesarias entre el sensor y el CAD, que constituyen la denominada etapa analógica frontal (*Analog Front End*, AFE), se diseñan teniendo en cuenta el *margen o rango dinámico* (*dynamic range*, DR) de medida, definido como

$$\text{DR}_x = \frac{\text{Rango de medida}}{\text{Resolucion}} = \frac{x_{\text{max}} - x_{\text{min}}}{\Delta x} \quad (9.4)$$

Para cada etapa se puede definir su rango dinámico, a su entrada o a su salida,

$$DR = \frac{V_{\max} - V_{\min}}{\Delta V} \quad (9.5)$$

donde ΔV es la resolución de la etapa. La *resolución en la entrada* de una etapa es el mínimo cambio de la entrada que produce un cambio correspondiente en su salida. La *resolución en la salida* es la resolución en la entrada multiplicada por la ganancia de tensión de la etapa. Por consiguiente, las etapas con ganancia unidad tienen la misma resolución en su entrada y en su salida. Para un CAD,

$$DR_{\text{CAD}} = \frac{V_{\max} - V_{\min}}{Q} = \frac{(2^N - 1)Q}{Q} \approx 2^N \quad (9.6)$$

El diseño de la etapa frontal parte entonces del rango dinámico de la medida, y a partir de éste se determina el número de bits del CAD. Éste tendrá un rango y un nivel de tensiones de entrada determinados, y la etapa frontal deberá adaptar la salida del sensor a esas características. Así pues, las etapas de la figura 9.2 no se diseñan de forma sucesiva de izquierda a derecha, sino que la elección del CAD y del sensor determinan las etapas intermedias. Para tener un diseño correcto, el rango dinámico de entrada de cada etapa entre el sensor y el CAD debe ser igual o mayor que el rango dinámico de salida de la etapa anterior. La figura 9.4 describe esta situación. El “*ruido*” es el valor eficaz de las fluctuaciones de tensión erráticas que se observan a la salida de una etapa cuando la entrada es nula, y es el factor que determina en último término la resolución. Si una etapa tiene una ganancia muy elevada, el ruido que tenga la entrada de la etapa siguiente será insignificante frente a la señal que ofrece la etapa anterior.

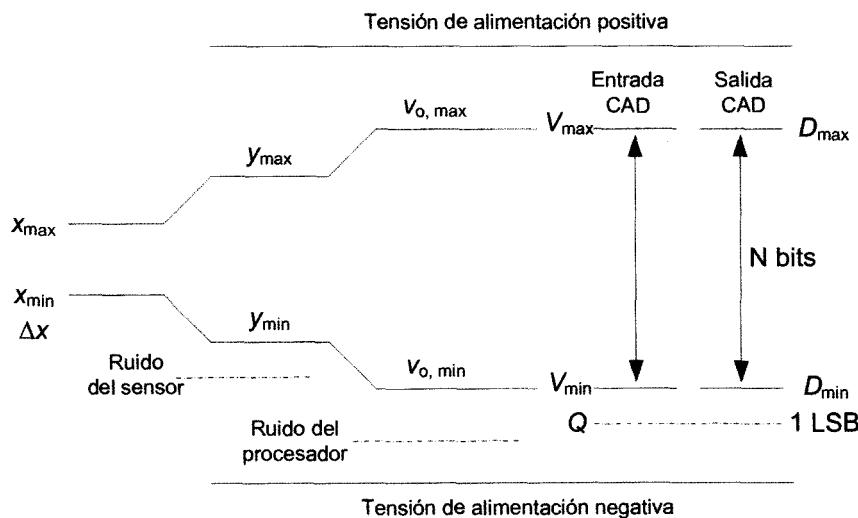


Figura 9.4 El rango dinámico del CAD debe ser superior al rango dinámico del sensor, y las etapas entre ambos deben tener mayor rango dinámico que el CAD.

ñal de, por ejemplo, 1 MHz a 2 MHz es de banda estrecha, una señal de 1 Hz a 100 Hz es de banda ancha. Las señales de banda ancha son más difíciles de procesar, porque las características del procesador deben ser uniformes en todo su ancho de banda. Las señales que sólo tienen componentes de muy baja frecuencia, menores de 0,1 Hz por ejemplo, se denominan *continuas*, y las de frecuencias superiores se denominan *alternas*.

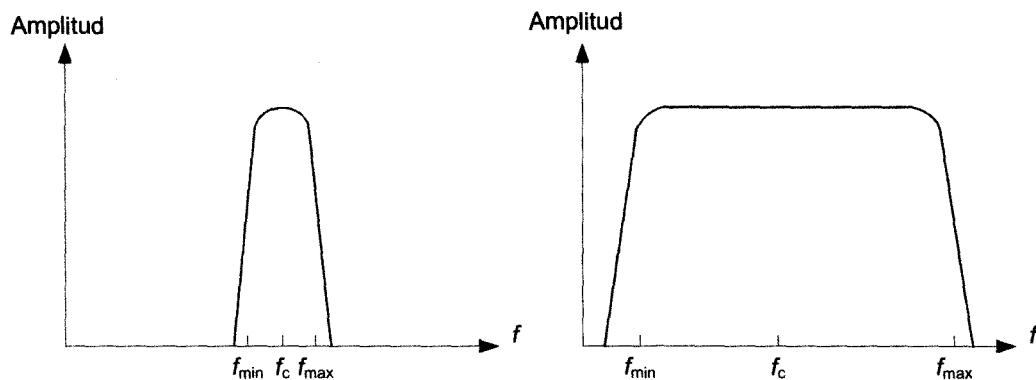


Figura 9.5 Señal de banda estrecha (izquierda) y de banda ancha (derecha). El criterio para determinar f_{max} y f_{min} depende del tipo de señal concreto. La máxima amplitud no corresponde necesariamente a f_c ni las amplitudes son necesariamente uniformes para todas las frecuencias entre f_{max} y f_{min} .

El *ancho de banda de un circuito* es el campo de frecuencias en el que la respuesta a una señal de entrada está dentro de ± 3 dB de la respuesta a las frecuencias en el centro del intervalo de frecuencias (figura 9.6). La *frecuencia central* se define como la media geométrica de las frecuencias a -3 dB.

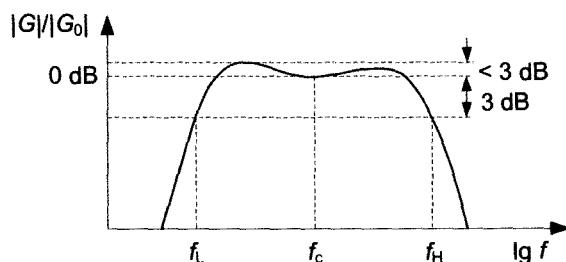


Figura 9.6 El ancho de banda de un circuito se define respecto a las frecuencias donde la ganancia está dentro de ± 3 dB respecto a la respuesta en las frecuencias centrales.

Cuando la conexión de una señal a un circuito o entre circuitos se realiza sin interponer condensadores en serie, se dice que los circuitos están *acoplados en continua*. Si en cambio se intercala un condensador en serie, los circuitos, o

la señal, están *acoplados en alterna*. En este caso, la amplitud de las frecuencias inferiores puede quedar atenuada, según se verá en el apartado 9.2.3.

9.1.4 Muestreo de señales

En los convertidores A/D directos, el valor digitalizado es el que tiene la tensión de entrada durante el proceso de conversión. Por ello, dicha tensión debe permanecer constante mientras está siendo digitalizada, y esto se consigue con el amplificador de muestreo y retención (*sample and hold amplifier, SHA*) (apartado 9.2.6). El número de muestras que hay que tomar por unidad de tiempo, y por tanto la velocidad de conversión, dependen de cómo se procesen los valores muestreados. En el mejor de los casos, que sería al emplear un filtro interpolador digital ideal, la frecuencia de muestreo viene dada por el *criterio de Nyquist*, según el cual dicha frecuencia debe ser mayor que el doble del ancho de banda de la señal de entrada (que incluirá siempre la señal deseada y una cierta cantidad de ruido no deseado). En caso contrario, aparecerán *señales falsas (alias)*, tal como muestra la figura 9.7.

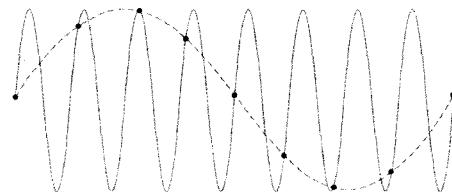


Figura 9.7 Señal falsa (alias) que aparece cuando se muestrea una señal a una frecuencia inferior a la frecuencia de Nyquist.

Si la señal de entrada es periódica, las muestras se pueden tomar de ciclos sucesivos (figura 9.8), lo cual reduce la velocidad de muestreo necesaria. Se habla entonces de *muestreo repetitivo secuencial*, en lugar de muestreo en tiempo real, y es común en los osciloscopios digitales.

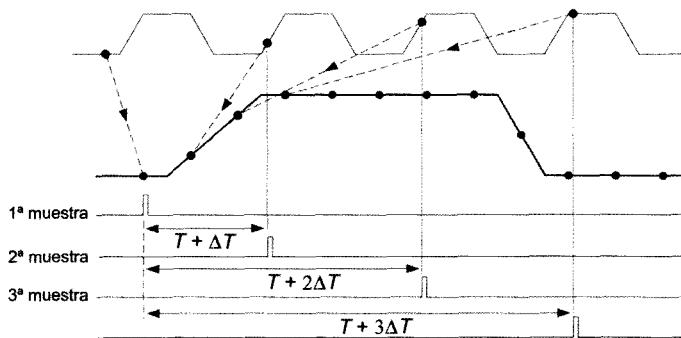


Figura 9.8 En el muestreo repetitivo secuencial, se toma muestras en instantes progresivamente alejados y que corresponden a ciclos sucesivos de la señal periódica de entrada.

9.1.5 Arquitecturas para la adquisición de señales. Sistemas de alto y bajo nivel

Cuando se debe digitalizar varias señales y el CAD es suficientemente rápido, no es necesario emplear un CAD para cada señal, sino que éste puede ser compartido por todas las señales mediante un *multiplexor analógico*, que es un conjunto de interruptores analógicos cuyo terminal de salida es común (apartado 9.2.4), y donde en cada instante de tiempo sólo uno está cerrado: aquél cuya entrada será digitalizada.

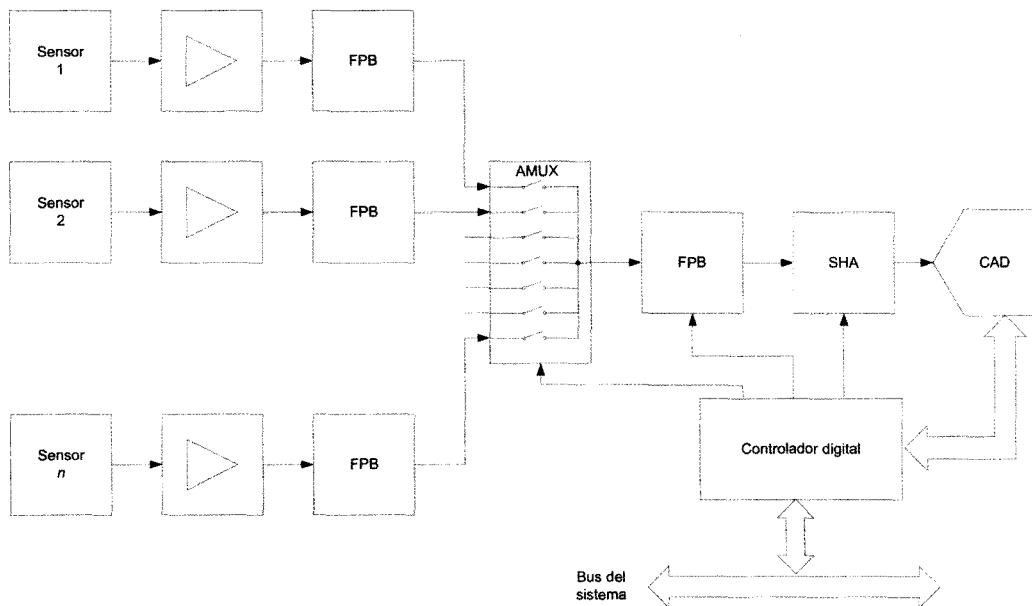


Figura 9.9 En el multiplexado de alto nivel, cada señal debe ser acondicionada antes de ser multiplexada. (FPB: filtro de paso bajo; SHA: amplificador de muestreo y retención).

La ubicación del multiplexor dentro de la cadena de funciones previas al CAD, da lugar a distintas arquitecturas para la adquisición de señales. Las dos más comunes son las denominadas de alto nivel y de bajo nivel. Los sistemas de adquisición de alto nivel (figura 9.9) aceptan sólo señales grandes y ofrecen una ganancia pequeña, normalmente programable (1, 2, 4, 8), posterior al multiplexor. Es necesario, pues, acondicionar previamente cada señal, y esto tiene un coste asociado. La ventaja es que la velocidad de conmutación del sistema (y por lo tanto, la velocidad de muestreo) puede ser rápida, pues el acondicionamiento previo al multiplexado estará adaptado a las características específicas de cada señal. Los microcontroladores que incluyen un CAD con varios canales de entrada analógicos (hasta 16), integran el multiplexor pero no el amplificador ni el filtro, por lo que estas funciones deben ser pro-

vistas externamente. Algunos convertidores A/D de varios canales tampoco incluyen el amplificador ni el filtro. Si se desea muestrear varias señales simultáneamente, hay que disponer un amplificador de muestreo y retención para cada una de ellas, antes de multiplexarlas.

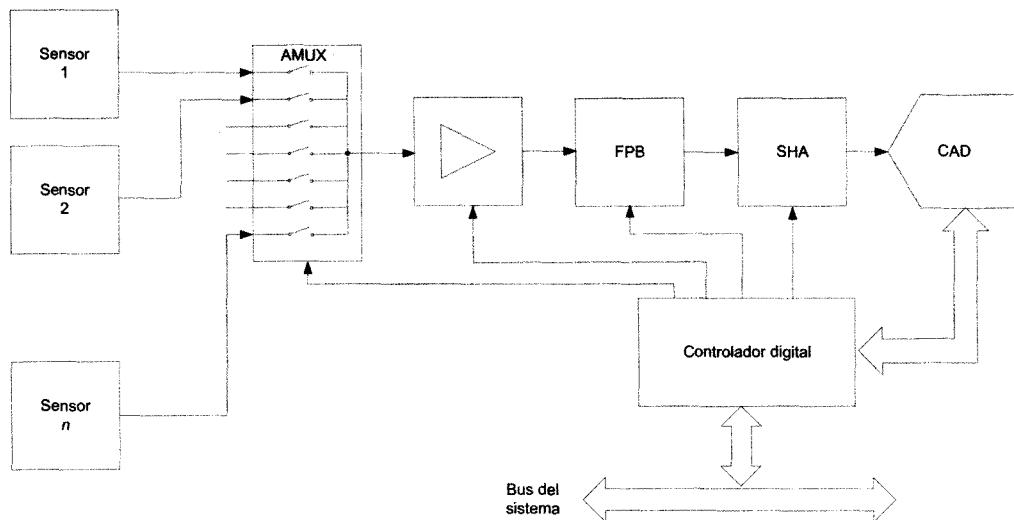


Figura 9.10 En el multiplexado de bajo nivel, no hace falta acondicionar cada entrada antes de multiplexarla, pero la velocidad de comutación entre canales es más lenta, porque hay que esperar a que se estabilicen las salidas del amplificador y del filtro. (FPB: filtro de paso bajo).

Los sistemas de adquisición de datos de bajo nivel (figura 9.10) incluyen un amplificador de ganancia programable con ganancias entre 1 y 500, grosso modo. Las señales de entrada pueden tener menor amplitud que en la adquisición de alto nivel, pero los errores del multiplexor se les sumarán, y la suma será amplificada. Además, al conmutar de uno a otro canal habrá que esperar a que se estabilicen las salidas del amplificador y del filtro (que puede ser también programable), y este tiempo de espera reduce la máxima velocidad de exploración de canales. Estos sistemas están disponibles como circuitos integrados periféricos.

9.2 La etapa frontal para la adquisición de señales

La etapa frontal en la adquisición de señales debe adaptar los niveles de tensión, corriente y potencia de la señal de entrada a las características del CAD, y también debe adaptar el ancho de banda de la señal según el procesamiento posterior previsto.

9.2.1 Atenuadores

Si la amplitud de la tensión de entrada excede el valor máximo que admite el CAD, hay que atenuarla. La figura 9.11 muestra la estructura general de un atenuador de tensión, formado por dos impedancias Z_1 y Z_2 , cuando la impedancia equivalente de salida de la fuente de señal es Z_o y la impedancia equivalente de entrada es Z_i . La atenuación es

$$A = \frac{V_i}{V_o} = \frac{Z_2 \| Z_i}{Z_o + Z_1 + Z_2 \| Z_i} \approx \frac{Z_2}{Z_1 + Z_2} \quad (9.8)$$

donde la aproximación es válida siempre y cuando $Z_{in} \gg Z_o$ y $Z_i \gg Z_2$, y el símbolo \parallel representa la combinación en paralelo de dos impedancias. Z_{in} es la impedancia equivalente de entrada del atenuador, una vez conectado a la etapa siguiente, es decir, Z_{in} es la impedancia que ve la fuente de señal, y vale

$$Z_{in} = Z_1 + Z_2 \| Z_i \quad (9.9)$$

El cumplimiento del criterio "mucho mayor" (o "mucho menor") hay que juzgarlo según la resolución del CAD: un parámetro será suficientemente grande, si el hecho de que no tenga un valor infinito pasa desapercibido para el CAD. Es decir, el efecto de su carácter finito es, en el peor caso, inferior a Q (o a una fracción de Q si se están considerando varios factores simultáneamente). Para evaluar el efecto de un parámetro que idealmente debiera ser nulo, se procede de forma análoga.

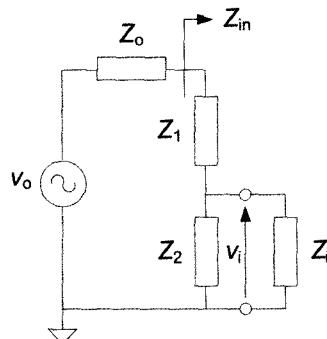


Figura 9.11 Atenuador de tensión conectado a una etapa cuya impedancia equivalente de entrada es Z_i .

Para atenuar una tensión continua bastan dos resistencias conectadas según muestra la figura 9.12a. La atenuación y la resistencia equivalente de entrada del atenuador son, respectivamente,

$$A = \frac{V_i}{V_o} = \frac{R_2 \| R_i}{R_o + R_1 + R_2 \| R_i} \approx \frac{R_2}{R_1 + R_2} \quad (9.10)$$

$$R_{in} = R_1 + R_2 \| R_i \quad (9.11)$$

Estas dos ecuaciones permiten calcular el valor de R_1 y R_2 . La tolerancia en sus valores, y el efecto de su coeficiente de temperatura, crearán una incertidumbre en el valor real de A . Para conocer este valor hay que calibrar el sistema (apartado 9.4).

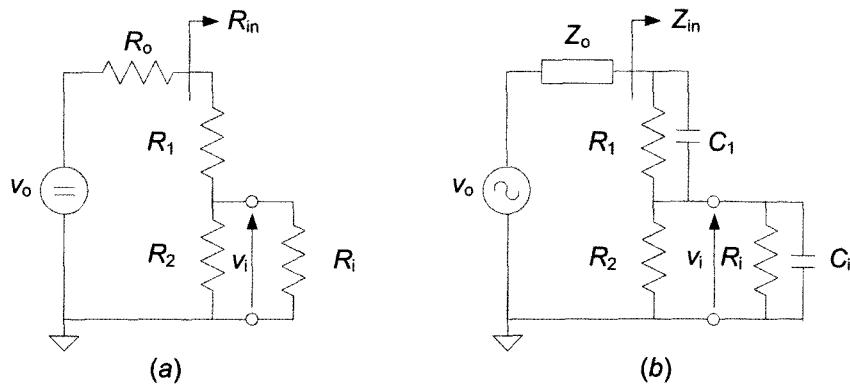


Figura 9.12 Atenuador de tensión para señales (a) continuas y (b) alternas.

Ejemplo 9.2

Diseñar un atenuador resistivo que presente una resistencia de entrada de $1 \text{ M}\Omega$ y que permita medir una tensión de 42 V con un circuito que admite hasta 5 V y que tiene una resistencia de entrada de $1 \text{ M}\Omega$.

Las condiciones de atenuación (9.10) y resistencia de entrada (9.11) llevan a

$$A = \frac{5 \text{ V}}{42 \text{ V}} = \frac{R_{eq}}{R_1 + R_{eq}}$$

$$R_{in} = R_1 + R_{eq} = 1 \text{ M}\Omega$$

donde $R_{eq} = R_2 \| R_i$. De aquí se obtiene, $R_1 = 881 \text{ k}\Omega$ y $R_2 = 135 \text{ k}\Omega$. Si se eligen resistencias con tolerancia del 1 %, los valores más convenientes son $R_1 = 887 \text{ k}\Omega$ y $R_2 = 133 \text{ k}\Omega$. La atenuación será un poco mayor, pero así aseguraremos que no se excede el rango de señal.

Si para atenuar señales alternas se utilizara el mismo circuito que para señales continuas, la capacidad equivalente de entrada (en paralelo con R_i) atenuaría más las señales de alta frecuencia que las de baja frecuencia. Para evitar esta atenuación desigual, que impediría determinar el valor de la señal de entrada cuando fuera de banda ancha, los atenuadores de alterna incluyen un condensador C_1 en paralelo con R_1 (figura 9.12b). Si C_1 se elige de forma que se cumpla $R_1 C_1 = (R_2 \parallel R_i) C_i$, se tendrá

$$A = \frac{V_i}{V_o} = \frac{Z_{eq}}{Z_o + Z_1 + Z_{eq}} \approx \frac{Z_{eq}}{Z_1 + Z_{eq}} = \frac{R_2 \parallel R_i}{R_1 + R_2 \parallel R_i} \quad (9.12)$$

$$Z_{in} = (R_1 + R_2 \parallel R_i) \parallel (C_1 \oplus C_i) \quad (9.13)$$

donde Z_{eq} es la combinación en paralelo de R_2 , R_i y C_i , y el símbolo \oplus representa la combinación en serie de dos condensadores, es decir, el recíproco de la suma de sus recíprocos. La atenuación es, pues, constante y se dice que el atenuador está *compensado en frecuencia*. Con estas dos ecuaciones y la condición de compensación se puede calcular R_1 , R_2 y C_1 .

Ejemplo 9.3

Diseñar un atenuador que permita dividir por 10 una tensión alterna (sea cual sea su frecuencia) y presente una resistencia de entrada de $10 \text{ M}\Omega$ a baja frecuencia, cuando se conecta a un sistema de adquisición de datos cuya impedancia de entrada es de $1 \text{ M}\Omega \parallel 125 \text{ pF}$.

Para obtener la atenuación deseada, de (9.12) resulta

$$A = \frac{R_{eq}}{R_1 + R_{eq}} = 0,1$$

y de aquí, $R_1 = 9R_{eq}$. La condición de igualdad de las constantes de tiempo implica $C_1 = C_i/9 = 125 \text{ pF}/9 \approx 14 \text{ pF}$, que es un valor estándar. Para tener la impedancia de entrada deseada a baja frecuencia, deberá ser

$$R_{in} = R_1 + R_{eq} = 10 \text{ M}\Omega = 10 R_{eq}$$

Luego, $R_{eq} = 1 \text{ M}\Omega$, que implica $R_2 = \infty$ (circuito abierto). Entonces, $R_1 = 9 \text{ M}\Omega$. Podríamos elegir $R_i = 8,98 \text{ M}\Omega (\pm 0,5\%)$, o $R_i = 9,09 \text{ M}\Omega (\pm 1\%)$.

Si la impedancia de salida de la fuente de señal (Z_o) en la figura 9.12 no es suficientemente pequeña, por ejemplo porque a la frecuencia de la señal, Z_{in} es baja debido a las capacidades parásitas, la aproximación en (9.12) deja de ser válida, y la atenuación aumenta con la frecuencia debido al denominado *efecto de carga*, que es la atenuación indeseada de una tensión causada por la impedancia finita de entrada del circuito donde se aplica. La situación general se puede describir mediante la figura 9.13a, que para tensiones alter-

nas procedentes de una fuente resistiva se convierte en la figura 9.13b, que a su vez se puede simplificar según se muestra en la figura 9.13c, donde $R_{eq} = R_o \parallel R_{in}$. La tensión a la entrada no es V_o sino V_{in} .

$$V_{in} = V_o \frac{Z_{in}}{Z_o + Z_{in}} = V_o \frac{R_{in}}{R_o + R_{in}} \frac{1}{1 + j2\pi f(R_o \parallel R_{in})C_{in}} = V_{eq} \frac{1}{1 + j2\pi fR_{eq}C_{in}} \quad (9.14)$$

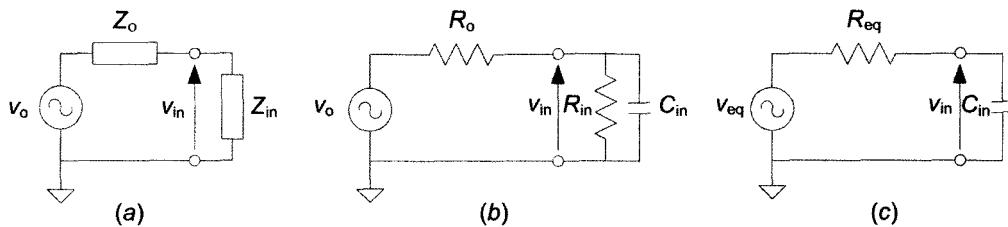


Figura 9.13 Circuitos equivalentes para analizar el efecto de carga en tensiones alternas.

La diferencia entre los módulos de la tensión obtenida y aplicada será el error absoluto, que dividido por el módulo de la tensión de entrada será el error relativo. Si se desea que éste sea menor que una cierta cota ε , se deberá cumplir

$$\frac{\|V_{in}\| - \|V_o\|}{\|V_o\|} < \varepsilon \quad (9.15)$$

que lleva a la condición,

$$2\pi f R_{eq} C_{in} < \frac{\sqrt{A_0^2 - 1 - \varepsilon^2 + 2\varepsilon}}{1 - \varepsilon} \approx \frac{\sqrt{A_0^2 - 1 + 2\varepsilon}}{1 - \varepsilon} = \frac{\sqrt{2\varepsilon}}{1 - \varepsilon} \quad (9.16)$$

donde $A_0 = R_{in}/(R_o + R_{in})$, y donde la primera aproximación es válida si $\varepsilon \ll 1$, y la segunda es válida cuando $A_0 \approx 1$. Con esta ecuación se puede determinar, por ejemplo, la máxima frecuencia de la tensión para que el error relativo sea inferior a ε .

Ejemplo 9.4

Calcular la máxima frecuencia de una sinusoidal que procede de una fuente de 600Ω de resistencia interna y se conecta a un circuito que tiene una impedancia de entrada de $10 M\Omega \parallel 100 \text{ pF}$, si el efecto de carga debe ser imperceptible en un sistema con 12 bits de resolución.

Para que el efecto de carga sea imperceptible, la atenuación (indeseada) que experimente la señal debe ser inferior a 1 LSB, y $1 \text{ LSB} = V_{FS}/2^{12}$. La diferencia entre la tensión del generador y la tensión a la entrada del circuito será máxima cuando la tensión sea máxima. Por lo tanto, $\varepsilon = 1/2^{12}$. La condición (9.16) se puede escribir como

$$f < \frac{1}{2\pi R_{eq} C_{in}} \frac{\sqrt{2}\epsilon}{1-\epsilon}$$

donde, en este caso, $R_{eq} \approx 600 \Omega$ y $C_{in} = 100 \text{ pF}$. De aquí resulta, $f < 58,6 \text{ kHz}$.

9.2.2 Amplificadores

Los amplificadores sirven para adaptar el rango dinámico, el nivel y la configuración de los terminales de la señal de entrada, a las características correspondientes de la etapa siguiente, a la vez que ofrecen una alta impedancia de entrada para minimizar el efecto de carga.

La configuración de los terminales de una señal se refiere a la relación entre dichos terminales y el terminal de referencia para la medida de tensiones propio del sistema (terminal común, 0 V, "masa eléctrica") (figura 9.14). Una tensión que se mide entre dos terminales, uno de los cuales está conectado directamente al terminal común, se denomina *asimétrica (single-ended voltage)*. Una tensión que se mide entre dos terminales, ninguno de los cuales está conectado directamente al terminal común, se denomina *diferencial* si se cumple,

$$\left. \begin{array}{l} v_H = v_c + \frac{v_d}{2} \\ v_L = v_c - \frac{v_d}{2} \end{array} \right\} \quad (9.17)$$

donde v_c es la denominada tensión en modo común y v_d es la tensión en modo diferencial. En un sistema alimentado entre 0 V y 5 V, por ejemplo, si una señal diferencial tiene $v_c = 2,5 \text{ V}$, se pueden procesar tensiones v_d positivas y negativas, porque sólo cambia la polaridad de éstas, sin que se rebasen las tensiones de alimentación. Si no se cumple la condición (9.17) y la señal no es asimétrica, se denomina *seudodiferencial*. En cualquier caso, si $Z_o = Z'_o$, se dice que la señal está *balanceada o equilibrada*. Si Z_{mc} es muy alta, la señal se califica como *flotante*. Tanto las señales diferenciales como las seudodiferenciales vienen descritas mediante tres terminales: alto, bajo y común, mientras que una señal asimétrica viene descrita mediante un terminal alto y un terminal común.

Un amplificador lineal ofrece a su salida una tensión proporcional a la tensión que hay entre sus dos terminales de entrada, "alto" y "bajo". Pero, de forma análoga a las señales, la entrada de los amplificadores puede ser asimétrica, diferencial o seudodiferencial (figura 9.15). En una *entrada asimétrica*,

el terminal bajo es el terminal común (0 V) de la fuente de alimentación del amplificador, que suele estar conectado a la masa metálica del equipo, y ésta a la toma de tierra de la instalación. En una *entrada diferencial*, ninguno de los dos terminales de medida está conectado a masa, y la impedancia entre cada terminal y masa es similar. En una *entrada seudodiferencial*, el terminal bajo está conectado a la masa del amplificador, pero ésta no está conectada directamente, con una impedancia baja, al terminal de toma de tierra de la instalación. Si tanto Z_L como Z_H son muy elevadas, se tiene una *entrada flotante*.

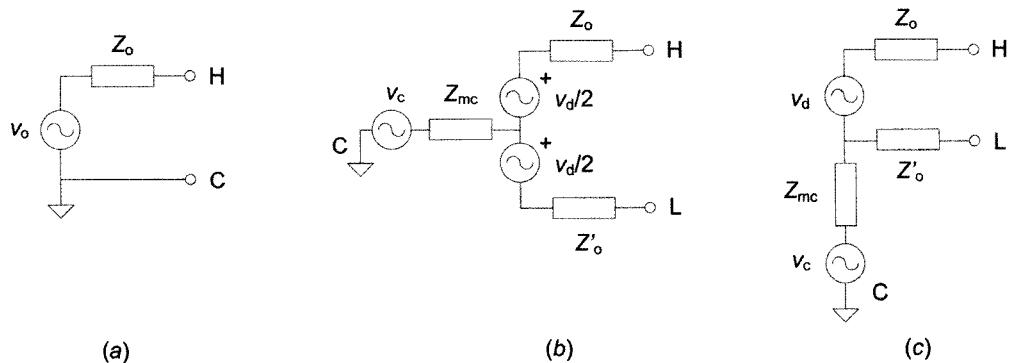


Figura 9.14 Tipos de señales según la configuración de sus terminales. (a) Asimétrica (single ended). (b) Diferencial. (c) Seudodiferencial. Z_{mc} es la impedancia en modo común, y puede ser nula.

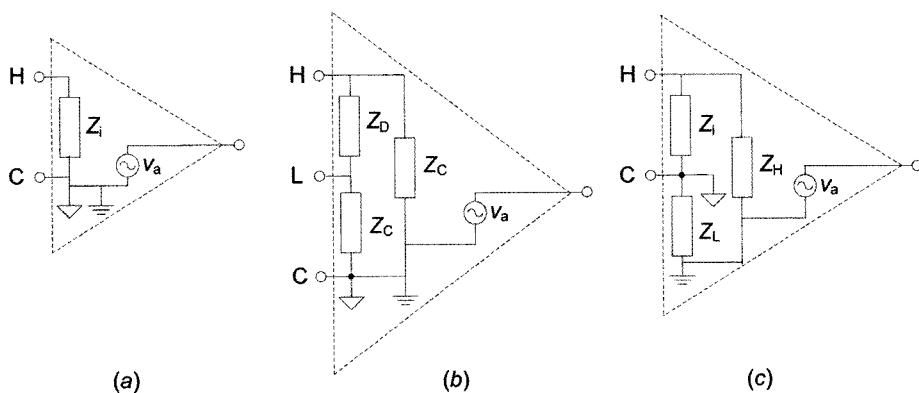


Figura 9.15 Tipos de amplificadores según su entrada: (a) asimétrica (single ended), (b) diferencial o (c) seudodiferencial; si Z_L y Z_H son muy elevadas, la entrada es flotante.

Para minimizar el efecto de carga, la impedancia entre los dos terminales de medida (Z_i o Z_D , según el caso), debe ser suficientemente alta. Para que la tensión de modo común no influya en la tensión de salida, en un amplificador diferencial Z_C debe ser lo mayor posible e igual para cada terminal de

entrada. En caso contrario, Z_o y Z_c por una parte, y Z'_o y Z'_c por otra, formarían sendos divisores de tensión con atenuación distinta, y una señal de modo común produciría una señal diferencial a la entrada del amplificador. En un amplificador seudodiferencial, Z_L y Z_H deben ser cuanto más altas, mejor. Obviamente, una señal diferencial o seudodiferencial no se debe conectar a una entrada asimétrica, pues “sobraría” un terminal de la señal. Además, al conectar una señal asimétrica a una entrada también asimétrica, hay que respetar la polaridad de los terminales, mientras que la polaridad de las señales diferenciales, y la de las seudodiferenciales en muchos casos, es irrelevante. No obstante, siempre hay que asegurar que las tensiones de modo común no contribuyan de forma significativa a la tensión amplificada.

Cuando el terminal o terminales de entrada de un amplificador se conectan a masa, la tensión de salida no es nula, sino que tiene un cierto nivel de continua que depende de su ganancia y de las resistencias dispuestas entre sus entradas y masa. Para considerar estos dos efectos, el amplificador se modela anteponiéndole una fuente de tensión continua, denominada *tensión de offset*, y una fuente de corriente continua (de polarización o de fugas) entre cada terminal y masa. La figura 9.16 muestra el circuito equivalente al conectar una señal diferencial a un amplificador diferencial que tiene unas impedancias de entrada muy altas, denominado *amplificador de instrumentación*. La presencia de las fuentes de corriente continua obliga a que entre cada uno de los dos terminales y masa haya un camino de resistencia suficientemente baja. Por esta razón, el terminal común de la señal se conecta normalmente a la masa del amplificador. Además, no se puede conectar sin más la señal al amplificador con un condensador en serie con cada terminal, porque los dos condensadores se irían cargando y acabarían saturando la salida del amplificador.

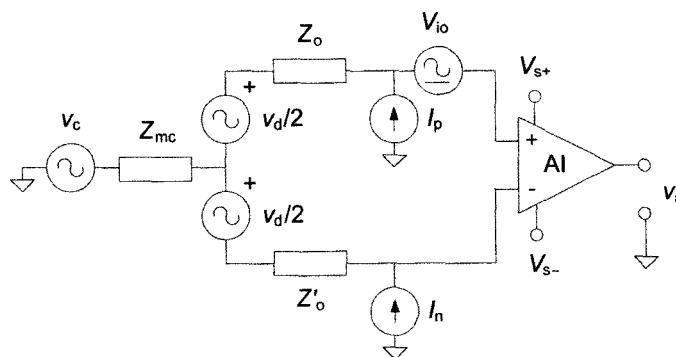


Figura 9.16 Circuito equivalente al conectar una señal diferencial a un amplificador de instrumentación. Obsérvese que la señal y el amplificador están conectados al mismo terminal de masa.

Si las corrientes de entrada son suficientemente pequeñas, y las tensiones de alimentación están bien filtradas, la tensión a la salida del amplificador se puede expresar, en primera aproximación, de la forma

$$v_a = G(1 + \varepsilon_G) \left(v_d + \frac{v_c}{CMRR} + V_{io} \right) + e_{NLG} \quad (9.18)$$

donde ε_G es el error relativo en la ganancia, e_{NLG} es el error de no linealidad en la ganancia, especificado a veces como una cota de tensión, y CMRR es la denominada *relación de rechazo del modo común*. El CMRR del amplificador se define como el cociente entre la tensión de salida producida por una determinada tensión aplicada en modo diferencial, y la tensión producida cuando la misma tensión de entrada se aplica en modo común. Si las impedancias de entrada en modo común no son muy grandes o están desequilibradas, el CMRR efectivo es menor que el del amplificador sólo. Los errores de ganancia y offset habituales en amplificadores de instrumentación no impiden obtener rangos dinámicos de 100 dB e incluso superiores, pero hacen muy difícil conseguir, sin calibración, una exactitud adecuada para sistemas de más de 10 bits. Con calibración, las dificultades crecen cuando se desea obtener más de unos 14 bits, a baja frecuencia o para señales de banda estrecha. Para señales de banda ancha, es difícil conseguir una exactitud de más de 12 bits.

La ganancia de cualquier amplificador se reduce a partir de una determinada frecuencia, tanto menor cuanto mayor sea la ganancia. La figura 9.17 muestra este efecto para un amplificador de instrumentación. En muchos amplificadores, la reducción de la ganancia con la frecuencia se puede describir mediante

$$G(f) < G_0 \frac{f_a}{f_a + jf} \quad (9.19)$$

donde G_0 es la ganancia a baja frecuencia y f_a es la *frecuencia de corte* (-3 dB). A efectos prácticos, esta dependencia significa que, si se acepta un error relativo ε , la máxima frecuencia permitida para una entrada sinusoidal es

$$f_{max} < f_a \frac{\sqrt{2\varepsilon - \varepsilon^2}}{1-\varepsilon} = f_a \frac{\sqrt{2\varepsilon}}{1-\varepsilon} \quad (9.20)$$

Ejemplo 9.5

Si se amplifica una señal sinusoidal con un amplificador de instrumentación que tiene un producto GBW de 1 MHz y ganancia 1000, ¿cuál puede ser la máxima frecuencia para que la atenuación sea imperceptible en un sistema con 12 bit de resolución?

Para $G = 1000$, la frecuencia de corte (a -3 dB) será $f_a = 1 \text{ MHz}/1000 = 1 \text{ kHz}$. Aplicando (9.20), con $\varepsilon = 1 \text{ LSB}/V_{FS}$,

$$f < f_a \frac{\sqrt{2\varepsilon}}{1-\varepsilon} = 1 \text{ kHz} \sqrt{2 \times 2^{-12}} = 22 \text{ Hz}$$

Esta frecuencia es muy baja, pero hay que tener en cuenta que -3 dB es una atenuación del 30 %, y aquí se acepta una atenuación de sólo 0,024 %.

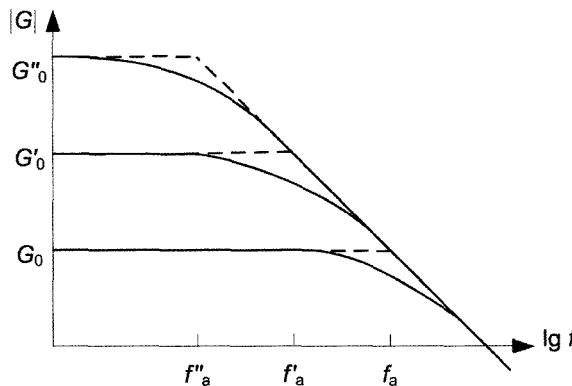


Figura 9.17 Relación entre la ganancia y la frecuencia en un amplificador de instrumentación típico: la ganancia empieza a decrecer a una frecuencia tanto menor cuanto mayor sea la ganancia a bajas frecuencias.

9.2.3 Filtros y protecciones de entrada

La tensión máxima que se puede aplicar directamente a la entrada de cualquier dispositivo electrónico sin dañarlo, está siempre limitada (cuando menos por debajo de las tensiones de alimentación del dispositivo). La máxima corriente en los terminales del dispositivo está también limitada (< 20 mA en muchos terminales de los PIC, < 10 mA o 1 mA en amplificadores y multiplexores, según su tecnología). Por ello, es necesario incorporar circuitos de protección en aquellos terminales que acepten conexiones a puntos externos al circuito. La figura 9.18 muestra que los limitadores de corriente se conectan en serie y los limitadores de tensión se conectan en paralelo con la entrada que se desea proteger. Si la entrada es diferencial, hay que añadir un limitador de corriente en serie con el terminal bajo, y un limitador de tensión (de modo común) entre cada terminal y masa. Cuando actúa el limitador de tensión, la tensión a la entrada del dispositivo es constante, e independiente de la señal, de forma que no hay daños, pero se pierde la información.

Otra precaución en las entradas analógicas es filtrar las interferencias acopladas desde otros circuitos, cuya amplitud puede ser suficiente para saturar la entrada del amplificador. Este filtro debe ser necesariamente pasivo, porque los filtros activos no aceptan tensiones mayores que la tensión de alimentación de los circuitos integrados. Para diseñar un filtro de paso bajo de primer orden, la resistencia del filtro puede ser el resistor que limita la corriente y para tener una capacidad suficientemente grande, se puede conectar un condensador en paralelo con el limitador de tensión. La máxima frecuencia de medida cuando se acepta un error relativo ϵ se puede calcular con (9.20), donde f_a es ahora la frecuencia de corte del filtro a -3 dB ($f_a = 1/(2\pi RC)$).

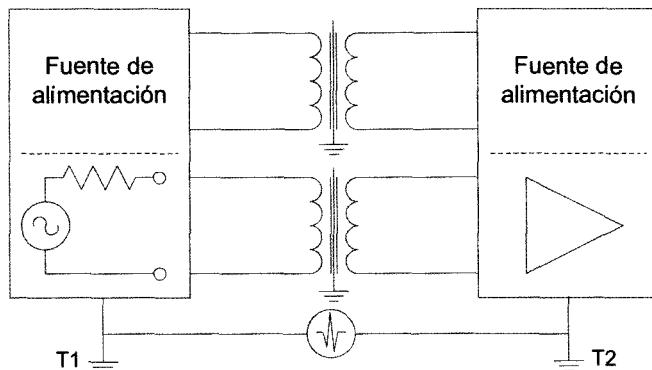


Figura 9.20 Cuando la diferencia de potencial entre tomas de tierra es excesiva, hay que emplear aisladores para la señal y para la tensión de alimentación, para que la tensión entre tomas de tierra no fuerce una corriente peligrosa en el circuito.

Cuando la señal que se desea adquirir está conectada a tierra en un punto distinto de la toma de tierra de la fuente que alimenta al amplificador, si la diferencia de potencial entre tomas de tierra es superior a la tensión de alimentación (por ejemplo, entre tomas alejadas y en un entorno industrial), los limitadores de tensión actuarían continuamente y no se podría adquirir información alguna. La solución pasa entonces por evitar toda conexión galvánica (óhmica) entre la señal y el amplificador: se emplea un aislador lineal o un amplificador de aislamiento para la señal analógica, o bien se digitaliza la señal con un sistema cuyo terminal de referencia no esté conectado a tierra, y luego se comunica la señal digital mediante optoacopladores u otro tipo de aislador digital. En cualquier caso, hay que evitar la continuidad óhmica tanto en el circuito de la tensión de alimentación, como en el circuito de la señal (figura 9.20).

9.2.4 Multiplexores analógicos

Un multiplexor analógico (AMUX) es un circuito que contiene un conjunto de interruptores analógicos cuyo terminal de salida es común, y cuya activación está organizada para que en cada instante sólo una de las señales de entrada esté conectada a la salida. La figura 9.21 muestra la estructura de un multiplexor (periférico) para señales asimétricas y otro para señales diferenciales. Para éstas, los dos interruptores involucrados se cierran al unísono. Los multiplexores de algunos microcontroladores permiten medir la diferencia entre diversos pares de señales, no sólo entre pares predeterminados como en la figura 9.21b.

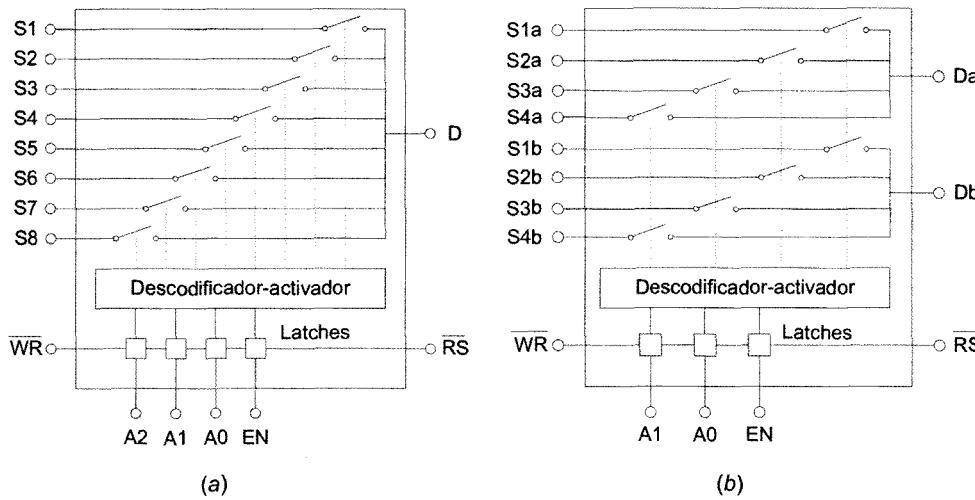


Figura 9.21 Estructura funcional de un multiplexor analógico para (a) señales asimétricas, y (b) señales diferenciales.

Los interruptores de un multiplexor ideal tienen resistencia nula cuando están cerrados e impedancia infinita cuando están abiertos, que les confiere un aislamiento total entre su entrada respectiva y la salida común. En un multiplexor real, los interruptores, que normalmente son transistores CMOS, tienen una resistencia finita R_{ON} cuando están cerrados, y una capacidad finita (C_{DS} , C_{OFF} , C_{iso}) entre su entrada y la salida común. R_{ON} provoca un efecto de carga (adicional al debido a la resistencia interna de la fuente de señal, R_o), denominado *pérdidas por inserción*, expresado a veces en decibelios para una resistencia de carga determinada (la resistencia equivalente de entrada de la etapa siguiente). Los multiplexores que integran una resistencia de protección en serie con cada interruptor, tienen mayores pérdidas por inserción. R_{ON} (y R_o) introducen además un retardo cuando se conmuta de un canal a otro, porque limitan la corriente de carga de la capacidad equivalente cone-

tada a la salida del multiplexor. Con los términos de la figura 9.22, si la señal a la entrada del interruptor es de baja frecuencia, al cerrarlo, la tensión a la salida aumenta paulatinamente según

$$v_L(t) = V_o \left(1 - e^{-t/\tau}\right) \quad (9.21)$$

donde $\tau = [(R_o + R_{ON})||R_L]C_L$. Entonces, si se desea que la diferencia relativa entre la tensión de salida en un instante dado y su valor final sea inferior a ϵ , después de cerrar el interruptor hay que esperar un tiempo

$$t_\epsilon = -\tau \ln \epsilon \quad (9.22)$$

Este tiempo limita la máxima velocidad de exploración de canales, y no viene especificado porque no es inherente al multiplexor, pues depende en parte de factores externos (R_o y C_L).

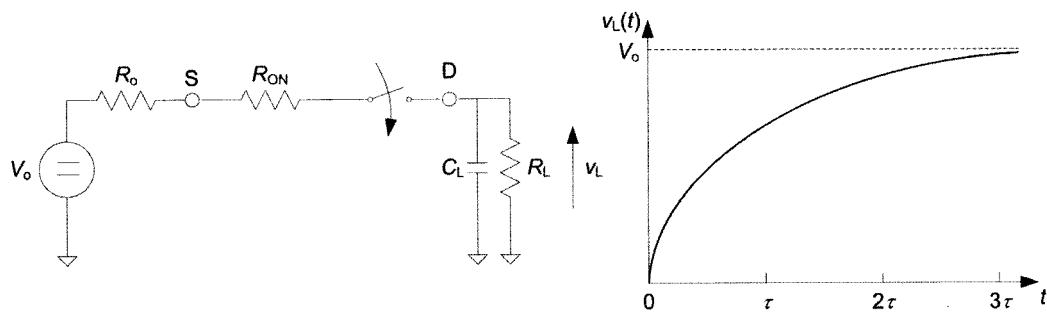


Figura 9.22 Circuito equivalente de un interruptor analógico que se cierra, y evolución de la tensión a su salida. C_L es la capacidad equivalente entre la salida del multiplexor y masa, e incluye la capacidad de salida del propio multiplexor y la capacidad equivalente de entrada de la carga (la etapa siguiente).

Ejemplo 9.6

Un determinado multiplexor analógico, con cada canal protegido frente a sobrecorrientes, tiene una resistencia equivalente de canal de $1 \text{ k}\Omega$, y su salida está conectada a una carga de unos 100 pF y resistencia muy alta. ¿Cuánto hay que esperar después de conmutar un canal para que la salida se aproxime suficientemente al valor final si el sistema es de 12 bits?

Hay que aplicar (9.22) con $\tau = 1 \text{ k}\Omega \times 100 \text{ pF} = 100 \text{ ns}$, y $\epsilon = 1 \text{ LSB}/2^{12}$. Resulta,

$$t_\epsilon = -\tau \ln \epsilon = -(100 \text{ ns}) \ln 2^{12} = 832 \text{ ns}$$

Este tiempo es mucho mayor que el tiempo de conmutación entre canales en un multiplexor común.

El aislamiento finito en los interruptores de un multiplexor produce la *diáfonía estática (static crosstalk)*: cuando un interruptor está abierto, la señal de su entrada contribuye inadvertidamente a la salida del multiplexor, donde

tendría que haber sólo la señal de entrada del interruptor (canal) seleccionado. Según el circuito equivalente de la figura 9.23, cuanto mayores sean la frecuencia de la señal aplicada a los canales abiertos (desconectados) y las resistencias en el canal 1 (R_{i1} y R_{ON1}), mayor será la diafonía del canal 2 sobre el canal 1.

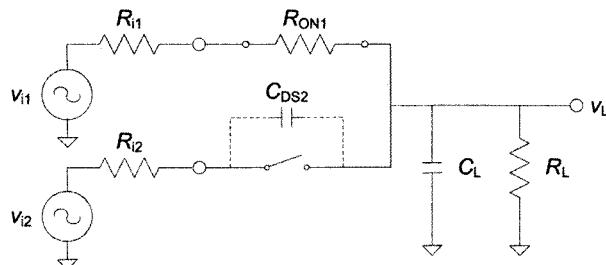


Figura 9.23 Circuito equivalente para analizar la diafonía estática en un multiplexor analógico: cuando se selecciona el canal 1, el canal 2, y todos los demás canales abiertos contribuyen (indebidamente) a la tensión de salida porque el aislamiento de los interruptores abiertos no es infinito.

Otras limitaciones de los multiplexores son el tiempo que transcurre desde que se da una orden de cambio de canal hasta que el canal queda efectivamente cerrado, denominado *tiempo de conmutación*, y las tensiones espurias inyectadas a su salida por las señales de control (selección de canal). Además, al cambiar de un canal al siguiente hay un breve intervalo de tiempo durante el cual no hay ningún canal cerrado, y esto hace que a la salida conserve durante dicho tiempo la tensión del canal seleccionado anteriormente. La *máxima velocidad de conmutación (throughput rate)* especificada por los fabricantes se refiere a veces a la situación ideal en la que hay una misma tensión continua aplicada a todos los canales y el único factor limitante es el tiempo de establecimiento de la salida cerca de su valor final, sin tiempos de espera adicionales ni valoraciones de la posible diafonía que sufren y provocan las señales alternas.

9.2.5 Filtros anti-alias

Para evitar la aparición de señales falsas al muestrear, es necesario que la amplitud de cualquier señal de frecuencia igual o superior a la frecuencia de Nyquist (la mitad de la frecuencia de muestreo), sea imperceptible para el CAD. Los circuitos que discriminan las señales según su frecuencia (u otro criterio), se denominan *filtros*. Un filtro que atenúa las señales de alta frecuencia respecto a las de baja frecuencia, tal como se requiere para que no aparezcan señales falsas al muestrear, se denomina *filtro de paso bajo, o filtro pasabajos*. Un

filtro ideal (figura 9.24a) elimina totalmente las señales no deseadas, y no modifica ni la frecuencia ni la fase de las señales de interés. En un filtro real (figura 9.24b), la atenuación en la banda rechazada es limitada; la respuesta en la banda de paso no es constante para todas las frecuencias; y la transición de una a otra banda no es abrupta sino progresiva. Además, el desfase no es necesariamente proporcional a la frecuencia, y esto distorsiona la forma de la señal. En cuanto a la respuesta transitoria (frente a una entrada con un cambio de amplitud brusco), el filtro ideal sería aquél que respondiera inmediatamente y sin rebasar la amplitud final. Un filtro ideal en su respuesta en frecuencia tiene una respuesta temporal que dista mucho de ser ideal.

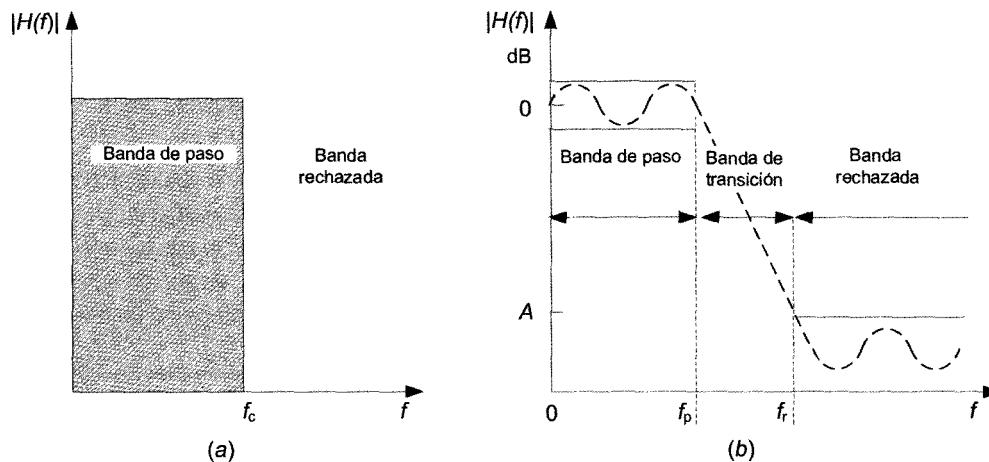


Figura 9.24 Respuesta en frecuencia de un filtro de paso bajo (a) ideal, y b) real.

Un filtro se caracteriza en primer lugar por su orden n . En la banda rechazada, los filtros comunes ofrecen una atenuación que aumenta en $20 \times n$ dB por cada década que aumente la frecuencia. En la banda de transición, en cambio, la atenuación inicial depende del tipo de filtro, que recibe un nombre según el polinomio del denominador de su respuesta en frecuencia. Los más comunes son los filtros de Butterworth, Chebishev y Bessel. Los filtros de Butterworth ofrecen una respuesta muy plana en la banda de paso. Los filtros de Chebishev ofrecen una mayor atenuación en la banda de transición, pero a costa de una respuesta que dista de ser plana en la banda de paso, y un desfase muy no lineal. Los filtros de Bessel tienen un desfase casi lineal con la frecuencia y no rebasan la amplitud final cuando se aplica un escalón a su entrada, pero tienen una mayor atenuación en la banda de paso y una menor atenuación en la banda de transición.

La elección del tipo de filtro es una responsabilidad del diseñador. Una vez decidido, los fabricantes ofrecen programas de ayuda para determinar el orden en función de la frecuencia de corte (-3 dB) en la banda de paso, la frecuencia de muestreo y la relación señal a ruido deseada. Para realizar el filtro, hay varios circuitos posibles, cuyo diseño es también automático mediante programas como FilterWizard® (Analog Devices), FilterCAD® (Linear Technology), Filterlab® (Microchip), o FilterPro® (Texas Instruments).

Ejemplo 9.7

Diseñar un filtro anti-alias para una señal que tiene un ancho de banda (a -3 dB) de 70 Hz, y una relación señal a ruido de 40 dB, cuando se muestrea a 1500 Hz, con un convertidor de 12 bits.

Con la herramienta "Anti-Aliasing Wizard" de Filterlab®, si se elige una frecuencia de corte del filtro de 70 Hz (para no modificar la señal) y se introducen la frecuencia de muestreo, el número de bits y la relación de señal a ruido en la entrada, se obtiene como respuesta un filtro de segundo orden, tipo Butterworth, que da una atenuación de 41,3 dB a 750 Hz (1500 Hz/2).

Grosso modo, para que a la mitad de la frecuencia de muestreo, la amplitud del ruido sea inferior a 1 LSB, en un sistema de 12 bits hace falta que el ruido esté unos 72 dB por debajo de la señal (supuesta con amplitud de fondo de escala). Dado que a la entrada el ruido ya está 40 dB por debajo de la señal, hacen falta 32 dB de atenuación adicionales. Si la frecuencia de corte se toma a 70 Hz, al cabo de una década (700 Hz) se tendrá una atenuación de $20 \times n$ dB. Luego, debe ser $n = 2$, y a 750 Hz se tendrá incluso más atenuación que la mínima necesaria.

9.2.6 Amplificador de muestreo y retención

El muestreo ideal de una señal consiste en tomar su valor instantáneo en un instante específico. En la práctica, tomar una muestra de tensión lleva un tiempo finito, que debe ser suficientemente breve para que la incertidumbre en el valor de la tensión muestreada sea imperceptible para el CAD. El valor máximo permitido para el tiempo de muestreo es tanto más breve cuanto mayores sean la pendiente de la señal muestreada y el número de bits del CAD.

Ejemplo 9.8

Se desea muestrear una señal sinusoidal de 1kHz con un CAD de 12 bits sin que la variación de la señal durante el tiempo de muestreo sea mayor que el máximo error de cuantificación. ¿Cuál debe valer como máximo dicho tiempo según se muestree en el pico o en el cruce por cero?

Si se toma la muestra cerca del valor de pico, la condición es

$$\Delta V = A - A \sin\left(\frac{\pi}{2} + 2\pi f t_m\right) < 1 \text{ LSB} = \frac{2A}{2^{12}}$$

que lleva a

$$\frac{\pi}{2} + 2\pi f t_m > 1,5395 \text{ rad}$$

$$t_m < \frac{0,03125 \text{ s}}{2\pi(10^3)} \approx 5 \mu\text{s}$$

que es un tiempo relativamente grande. Si en cambio se toma la muestra en el punto de cruce por cero, la derivada de una tensión sinusoidal $v(t) = A\sin(2\pi ft)$ en dicho punto es $2\pi fA$. Si se desea que el cambio de tensión ΔV durante un tiempo $\Delta t = t_m$ (tiempo de muestreo), sea inferior a 1 LSB, se deberá cumplir

$$\frac{\Delta V}{t_m} = 2\pi fA < \frac{1 \text{ LSB}}{t_m} = \frac{2A}{2^{12}}$$

$$t_m < \frac{1}{\pi 2^{12} (1 \text{ kHz})} = 7,8 \text{ ns}$$

que es un tiempo muy breve.

Si el CAD tuviera que digitalizar (cuantificar y codificar) la entrada en un tiempo tan breve como el permitido para el muestreo, la resolución posible sería muy pobre. Por esto, al CAD le precede un amplificador de muestreo y retención (*sample and hold amplifier, SHA*), cuya estructura funcional se muestra en la figura 9.25. Se puede describir como un interruptor que se cierra para cargar un condensador de retención (C_H) al valor de la tensión de entrada, precedido de un amplificador seguidor, para ofrecer alta impedancia de entrada, y con un seguidor de salida, para no descargar el condensador mientras dure la conversión. Los convertidores que integran un amplificador de muestreo y retención se denominan, en inglés, *sampling converters*.

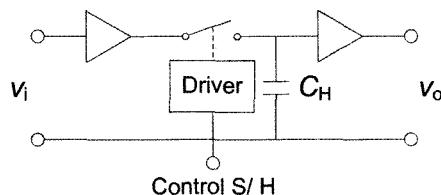


Figura 9.25 Estructura funcional de un amplificador de muestreo y retención. En los SHA integrados en microcontroladores no suele haber ninguno de los dos seguidores de tensión.

Cuando el SHA está en la fase de muestreo, idealmente debería comportarse como un amplificador de ganancia unidad. Cuando está en la fase de retención, la tensión en bornes del condensador debería mantenerse constante; sin embargo, las corrientes de entrada del buffer de salida, y las corrientes de fuga del interruptor y del propio condensador, lo van descargando lentamente. La velocidad de descarga (*droop rate*) es

$$\frac{dv_c}{dt} = \frac{i_d}{C_H} \quad (9.22)$$

donde i_d es la corriente de descarga (resultado de todas las fugas). La descarga será tanto más lenta cuanto mayor sea C_H .

En la transición de retención a muestreo, el condensador tarda un cierto tiempo en cargarse al valor de la tensión de entrada, denominado *tiempo de adquisición*. Este tiempo será tanto más breve cuanto menor sea C_H , por lo que hay un compromiso entre el tiempo de adquisición y la velocidad de descarga. Si el tiempo de conversión del CAD es suficientemente breve, este compromiso se puede resolver utilizando el llamado *modo de seguimiento y retención (track and hold)*: en lugar de tomar una muestra rápidamente y retener su valor durante un tiempo relativamente largo, se cierra el interruptor durante un tiempo largo que permite al condensador cargarse y seguir las fluctuaciones de la tensión de entrada; en un instante dado, se abre el interruptor y se digitaliza la tensión del condensador durante un tiempo relativamente breve. El tiempo que tarda el interruptor en abrirse desde que se da la orden hasta que la tensión en el condensador se “desconecta” de la entrada, se denomina *tiempo de apertura*, y conlleva una incertidumbre en el instante en que se toma realmente la muestra, porque no es un tiempo constante sino sujeto a fluctuaciones erráticas (y breves).

9.2.7 Convertidores A/D

El convertidor A/D integrado en los microcontroladores, y la mayor parte de los convertidores A/D periféricos, están basados en el algoritmo de aproximaciones sucesivas (figura 9.26): primero se compara la tensión de entrada (v_x) con la mitad de la tensión de fondo de escala ($V_{FS} = V_{ref} = 2^N \times Q$, para un CAD de N bits); si $v_x > V_{FS}/2$, el bit de más peso de la salida (MSB) del CAD se pone a 1, y se incrementa la tensión de comparación en $V_{FS}/4$; si $v_x < V_{FS}/2$, se toma MSB = 0, y la nueva tensión de comparación es $V_{FS}/4$. Para decidir el siguiente bit se procede de forma análoga: si el resultado de la segunda comparación es positivo ($v_x > V_{comparador}$), el bit se pone a 1; en caso contrario, se pone a cero. El tercer nivel de comparación será, respectivamente, el anterior más o menos $V_{FS}/8$, y así sucesivamente. v_x debe permanecer constante durante todo el tiempo de conversión, que será tanto más largo cuanto mayor sea N . Esos niveles de comparación los obtiene un convertidor digital-analógico (CDA) (apartado 9.6.1).

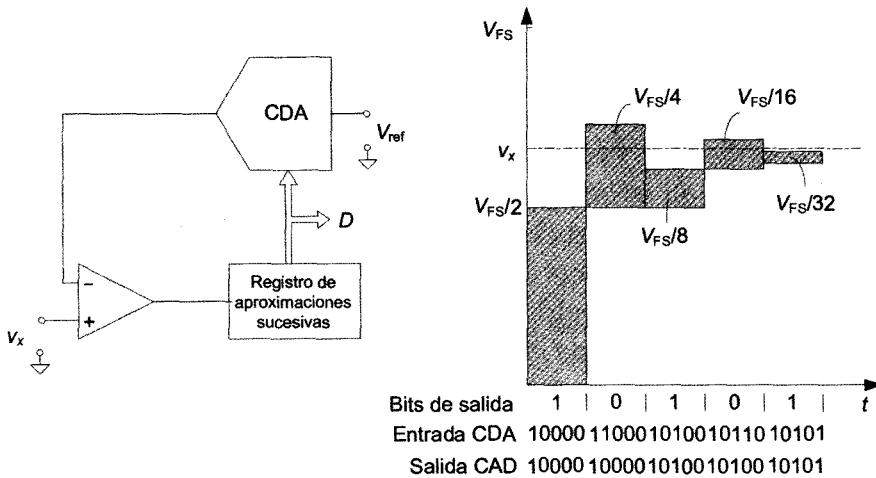


Figura 9.26 Estructura funcional de un CAD de aproximaciones sucesivas, y proceso de decisión del valor de los bits de salida. V_{FS} : tensión de fondo de escala.

La relación entre la tensión de entrada del CAD, v_x , y el código de salida D , queda descrita mediante la *característica de transferencia* de la figura 9.3. Muchos convertidores se diseñan con dicha característica desplazada $Q/2$ hacia la izquierda, tal como se muestra en la figura 9.27 (línea a trazos); es decir, los umbrales de transición de un código a otro son tensiones múltiplo de $Q/2$, no de Q . En realidad, la transición de un código de salida a otro no se produce siempre para el mismo valor de la tensión de entrada, sino que unas veces se produce para una tensión ligeramente superior y otras para una tensión ligeramente inferior. El rango de tensiones que llevan a un mismo código de salida se denomina *anchura de código* (*code width*). Como *umbral de transición* se toma aquella tensión para la que hay una probabilidad del 50 % de que la transición se produzca en una tensión mayor o en una tensión menor. Para un CAD ideal, la anchura de código es 1 LSB para todos los códigos, y la línea que une los centros de los escalones de la característica de transferencia es una recta de pendiente unidad que pasa por el origen.

En un CAD real, la línea a través del centro de los escalones puede presentar un error de cero (*offset*) y un error de ganancia (*gain error, full scale error*). Si hay un error de cero, todos los umbrales de transición están desplazados en el mismo sentido y en la misma magnitud (figura 9.27a). Si hay un error de ganancia, una vez corregido el error de cero, resulta que dicha línea es una recta que no tiene pendiente unidad (figura 9.27b).

También puede suceder que la anchura de código cambie de uno a otro código, en cuyo caso se habla de *no linealidad diferencial* (DNL, *differential non-*

linearity). DNL se define como la diferencia entre la anchura de cada código y la anchura ideal (1 LSB), y puede ser positiva o negativa. Si para un código concreto, $DNL = -1$ LSB, quiere decir que o bien el código precedente no se observa nunca, o bien DNL está especificada en unas condiciones extremas en las que difícilmente se observaría el código precedente. El resultado global de DNL es la denominada *no linealidad integral* (*integral nonlinearity*, INL), que es una medida de cuánto se aparta la línea a través del centro de los escalones de la característica de transferencia actual respecto a la línea recta de pendiente unidad, una vez se han corregido los errores de cero y de ganancia, si los hubiere. Si los errores de cero y de ganancia se corrigen mediante calibración, INL es el factor que (junto con la cuantificación) limita la incertidumbre sobre el valor de la entrada que ha dado lugar al código de salida observado. Si no hay calibración, la incertidumbre viene determinada por el *error absoluto*, que es la suma de los errores de cero, ganancia y no linealidad.

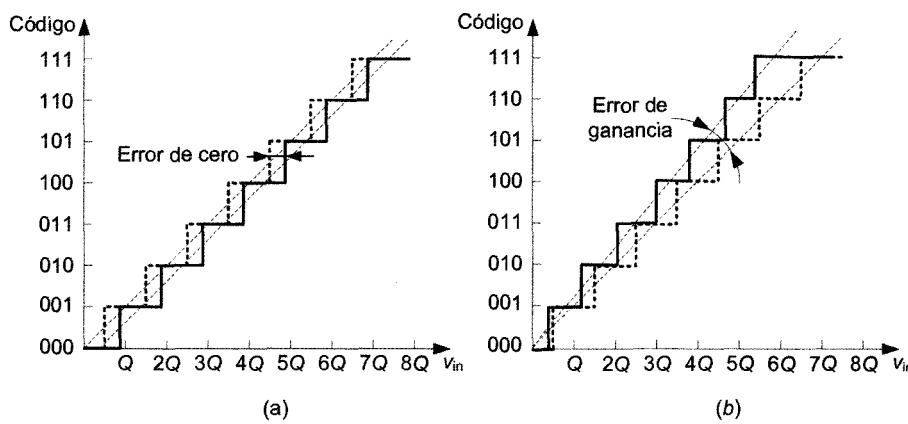


Figura 9.27 Característica de transferencia de un CAD que presenta (a) un error de cero, y (b) un error de ganancia. La característica de transferencia ideal es la línea gruesa a trazos.

Para las señales alternas es importante conocer cuánto ruido y distorsión introduce el CAD. En un CAD ideal, las muestras de salida reflejarían sólo el *ruido de cuantificación*, que es una forma de describir la incertidumbre sobre el valor de la tensión de entrada a partir del código de salida, debido a que todas las tensiones del mismo intervalo de cuantificación producen el mismo código. Su valor eficaz es $Q/\sqrt{12}$. El valor eficaz del ruido a la salida de un CAD real (medido con un procedimiento estandarizado), es mayor. Se define entonces el *número de bits efectivo* (*Effective Number of Bits*, ENOB) como

$$\text{ENOB} = N - \text{lb} \frac{\text{Ruido CAD}}{Q/\sqrt{12}} = N - \text{lb} \frac{\text{Ruido CAD}}{\frac{V_{\text{FS}}}{2^N}/\sqrt{12}} = \text{lb} \frac{V_{\text{FS}}/\sqrt{12}}{\text{Ruido CAD}} \quad (9.23)$$

En general, para un mismo valor de N , ENOB es mayor en un CAD periférico que en uno integrado en el microcontrolador.

La configuración de los terminales de entrada de un CAD se puede describir con los mismos términos empleados para los amplificadores (apartado 9.2.2), pero aquí dicha configuración va ligada al código digital de salida. Si la entrada es asimétrica (*single-ended input*) o seudodiferencial, sólo se admiten tensiones positivas y el código de salida suele ser binario directo (*unipolar straight binary*). Si la entrada es diferencial, se aceptan tensiones positivas y negativas respecto a la tensión de masa, y el código de salida suele ser binario con complemento a 2, porque es más cómodo para los cálculos aritméticos. El rango de tensiones de entrada (*full-scale input range*, FSR) es V_{ref} si la entrada es asimétrica y $2 \times V_{\text{ref}}$ si es diferencial, pero para un número de bits determinado N , el intervalo de cuantificación tiene doble anchura para una convertidor con entrada diferencial que para uno con entrada asimétrica.

Según (9.1), la salida del CAD representa la relación (cociente) entre la tensión de entrada v_x y la tensión de referencia V_{ref} por lo que la incertidumbre que se tenga en V_{ref} por su tolerancia y sus variaciones con el tiempo y la temperatura, repercutirá directamente en la salida. Ahora bien, si la tensión de entrada (v_x) proviene de un sensor de resistencia variable alimentado con una tensión de referencia, de modo que $v_x = xV_{\text{ref}}$ con x proporcional a la magnitud medida, si se emplea la misma V_{ref} para el sensor y el CAD, la salida no dependerá de la incertidumbre de V_{ref} . Se habla entonces de *medidas por relación*, por contraposición a las medidas respecto a una tensión de referencia independiente, que se denominan *medidas absolutas*. Pero conviene recordar que un CAD siempre mide una relación de tensiones, nunca tensiones absolutas.

9.3 El módulo de conversión A/D de 10 bits en los microcontroladores PIC

9.3.1 Arquitectura del módulo de conversión A/D

Los microcontroladores PIC de gama media tienen convertidores A/D de aproximaciones sucesivas, normalmente de 10 bits, cuya estructura interna simplificada se muestra en la figura 9.28. Los componentes principales de este módulo son:

- Multiplexor analógico de hasta 8 canales de entrada.

- Amplificador de muestreo y retención (sin seguidor de entrada ni salida).
- Convertidor A/D de aproximaciones sucesivas, de 10 bits.
- Registros para controlar el módulo (ADCON0 y ADCON1) y para almacenar el resultado de la conversión A/D (ADRESH y ADRESL).

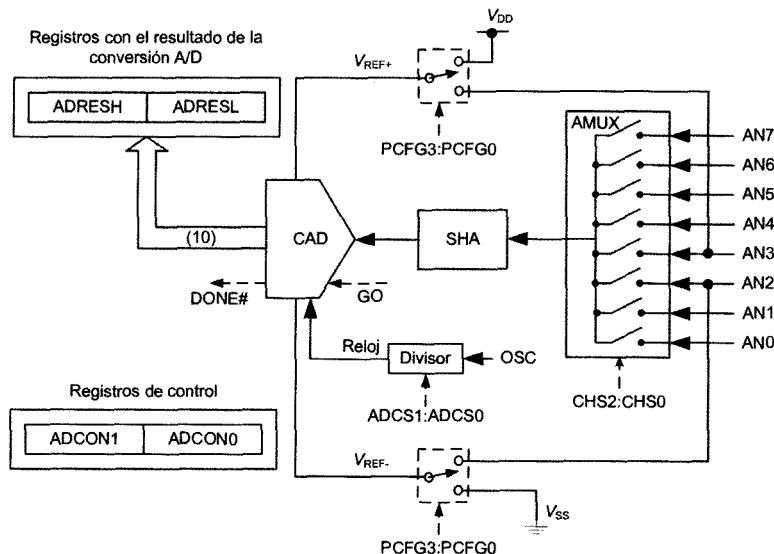


Figura 9.28 Bloques que componen el módulo de conversión A/D de 10 bits en los microcontroladores PIC de gama media. CAD: Convertidor A/D de aproximaciones sucesivas, de 10 bits; SHA: amplificador de muestreo y retención; AMUX: multiplexor analógico.

Este módulo puede tener hasta ocho entradas analógicas, que están disponibles como funciones alternativas de los terminales de los puertos paralelos. El número de entradas analógicas o canales de entrada depende del PIC en particular. Por ejemplo, el PIC16F873 tiene 5 entradas analógicas, disponibles en igual número de terminales del puerto paralelo A (RA0/AN0, RA1/AN1, RA2/AN2/VREF-, RA3/AN3/VREF+ y RA5/AN4). En los PIC que tienen más de cinco entradas analógicas, como el PIC16F874, que tiene ocho, se utilizan también tres terminales del puerto E para las entradas analógicas AN5, AN6 y AN7. La selección del canal se realiza con los bits CHS2:CHS0 del registro ADCON0.

El amplificador de muestreo y retención está compuesto básicamente por un condensador (sin amplificadores de entrada ni de salida), que empieza a cargarse en cuanto se selecciona en el multiplexor el canal deseado. La tensión en el condensador sigue la evolución de la tensión de entrada (modo

track), y cuando se da una orden el condensador se desconecta de la entrada analógica y empieza la conversión.

El resultado de una conversión se deposita en los registros ADRESH y ADRESL. Dentro del espacio de 16 bits conformado por este par de registros, los 10 bits resultantes de una conversión se pueden depositar justificados “a la izquierda” o “a la derecha”, como muestra la figura 9.29. La opción de depositar el resultado justificado a la izquierda (figura 9.29a) resulta muy apropiada para operar el CAD como un convertidor de 8 bits, con el resultado de la conversión en el registro ADRESH.

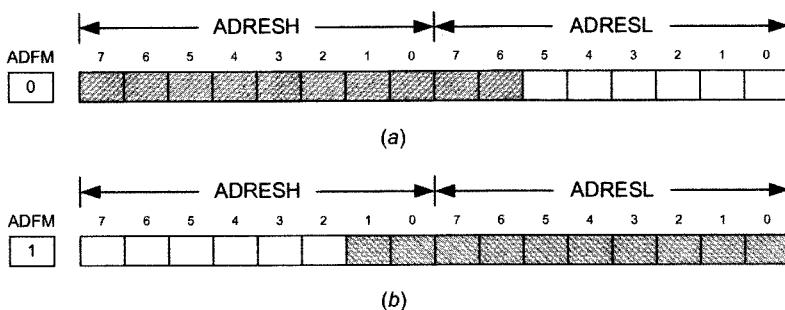


Figura 9.29 El resultado de una conversión A/D se puede leer en los registros ADRESH y ADRESL, en un formato que se especifica con el bit ADFM del registro ADCON1. En (a) el bit ADFM = 1 y el resultado queda “justificado a la izquierda”, por lo que en ADRESH quedan los 8 bits más significativos de la conversión. En (b) el bit ADFM = 0 y el resultado queda “justificado a la derecha”, con los 8 bits menos significativos en ADRESL.

La tensión de referencia para la conversión A/D puede ser la tensión de alimentación del microcontrolador o una tensión externa que se aplique entre los terminales AN3/VREF+ y AN2/VREF-. La selección se hace por software, con los bits PCFG3:PCFG0 del registro ADCON1. En la opción por defecto (es decir, la que se tiene después de una acción de *reset*), la tensión de referencia se toma de la alimentación del microcontrolador.

Las conversiones A/D se realizan en sincronismo con una señal de reloj. Este reloj se obtiene o bien del oscilador principal del microcontrolador mediante un divisor programable, o bien de un oscilador RC interno de frecuencia fija, no representado en la figura 9.28. Con los bits ADCS1 y ADCS0 del registro ADCON0 se selecciona la fuente del reloj y se programa el divisor de frecuencia si la fuente es el oscilador principal del microcontrolador. Para que el CAD funcione mientras el microcontrolador está en el modo de bajo consumo (*sleep*), hay que seleccionar el oscilador RC interno.

Para iniciar una conversión A/D hay que activar el bit de control GO. Cuando ha terminado la conversión, se activa el bit de estado DONE#. En realidad, GO y DONE# están realizados en un mismo bit: el bit GO/DONE# del registro ADCON0 (figura 9.30). El programador debe poner a 1 este bit para iniciar una conversión A/D y cuando este bit toma el valor 0, el resultado de la conversión está disponible en ADRESH y ADRESL. Cuando finaliza una conversión A/D, se activa también el bit ADIF del registro PIR para solicitar interrupción. Si el bit ADIE del registro PIE está activo y el sistema de interrupción del microcontrolador está habilitado (el bit GIE del registro INTON es 1), la solicitud de interrupción se hace efectiva.

ADCON0								
7	6	5	4	3	2	1	0	
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DINE#	-	ADON	

ADCON1								
7	6	5	4	3	2	1	0	
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0	

Figura 9.30 Registros ADCON0 y ADCON1 en un PIC16F873.

La figura 9.30 muestra los bits que componen los registros de funciones especiales ADCON0 y ADCON1 en un PIC16F873, con los cuales se controla la operación del módulo de conversión A/D. En el registro ADCON0, el par de bits ADCS1:ADCS0 selecciona la fuente del reloj del convertidor A/D y su frecuencia, según la tabla 9.1. La entrada analógica se selecciona con los bits CHS2, CHS1 y CHS0. GO/DONE# es el bit de control/estado para iniciar una conversión A/D y conocer si ha finalizado. El bit ADON = 1 habilita el funcionamiento del módulo de conversión A/D del microcontrolador.

Tabla 9.1 Selección de la fuente del reloj del convertidor A/D y su frecuencia con los bits ADCS1:ADCS0 del registro ADCON0. F_{osc} es la frecuencia del oscilador principal del microcontrolador. La frecuencia del oscilador RC interno es fija y puede estar entre los valores especificados en la tabla.

ADCS1:ADCS0	Fuente de reloj	Frecuencia del reloj del convertidor A/D
00	Oscilador principal	$F_{osc}/2$
01	Oscilador principal	$F_{osc}/8$
10	Oscilador principal	$F_{osc}/32$
11	Oscilador RC interno	167 kHz a 500 kHz

En el registro ADCON1, el bit ADFM determina la justificación (a la derecha o a la izquierda) del resultado de la conversión A/D en los registros ADRESH y ADRESL. Los bits PCFG3:PCFG0 configuran los terminales del

microcontrolador utilizados por el módulo de conversión A/D como entradas analógicas del módulo o como terminales digitales de los puertos paralelos correspondientes. A modo de ejemplo, la tabla 9.2 muestra los valores que toman estos bits en un PIC16F873. Después de un reset, los bits PCFG3:PCFG0 quedan en el valor 0, por lo que los terminales RA5, RA4:RA0 están asignados al módulo de conversión A/D. Para asignar estos terminales al puerto paralelo A, hay que programar la opción correspondiente en el registro ADCON1.

Tabla 9.2 Asignación de funciones a los terminales de entrada del puerto A en un PIC16F873 mediante los bits PCFG3:PCFG0 del registro ADCON1. A: entrada analógica, D: entrada digital.

PCFG3:PCFG0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	Número de canales analógicos/digitales
00x0	A	A	A	A	A	V_{DD}	V_{SS}	5/0
1001	A	A	A	A	A	V_{DD}	V_{SS}	5/0
00x1	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/0
1010	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/0
1x00	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	3/0
1011	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	3/0
0100	D	A	D	A	A	V_{DD}	V_{SS}	3/2
0101	D	V_{REF+}	D	A	A	RA3	V_{SS}	2/2
1101	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2	2/1
1110	D	D	D	D	A	V_{DD}	V_{SS}	1/4
1111	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2	1/2
011x	D	D	D	D	D	V_{DD}	V_{SS}	0/5

9.3.2 Tiempos de una conversión A/D

La digitalización de una señal analógica se realiza en dos etapas sucesivas: el muestreo y retención, que tiene lugar en el circuito de igual nombre, y la conversión A/D. Cada una de estas etapas dura un tiempo determinado.

En los PIC de la gama media, el condensador de retención es de 120 pF y el tiempo que demora su carga es el tiempo de adquisición (T_{ACQ}). Una vez transcurrido este tiempo puede comenzar la conversión en el convertidor A/D de 10 bits, que dura un tiempo T_{CONV} (tiempo de conversión). La figura 9.31 muestra estos tiempos.

Según el fabricante, el tiempo de adquisición en los PIC de gama media está entre 10 μ s y 20 μ s, es decir,

$$10 \mu\text{s} \leq T_{ACQ} \leq 20 \mu\text{s} \quad (9.24)$$

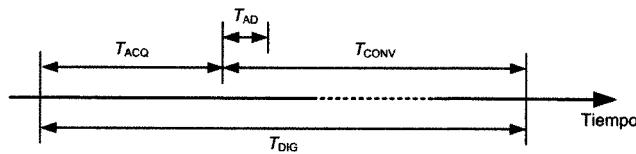


Figura 9.31 Tiempos relacionados con la digitalización de una señal analógica en el módulo de conversión A/D. T_{ACQ} : tiempo de adquisición, T_{AD} : tiempo de conversión de un bit en el convertidor A/D, T_{CONV} : tiempo de conversión A/D de un dato, T_{DIG} : tiempo de digitalización. El tiempo de adquisición está entre 10 μs y 20 μs y depende en gran medida de la resistencia de salida de la fuente de señal. El tiempo de conversión de un bit debe ser mayor que 1,6 μs . El tiempo de conversión de 10 bits es 11,5 veces el tiempo de conversión de un bit.

La falta del seguidor de tensión de entrada mostrado en la figura 9.25 hace que T_{ACQ} dependa mucho de la resistencia de salida (R_s) de la fuente de señal. El fabricante de los PIC recomienda que R_s sea siempre menor que 10 k Ω . Si $R_s = 10$ k Ω , resulta $T_{ACQ} = 20 \mu s$, y para $R_s = 50 \Omega$ se obtiene $T_{ACQ} = 10 \mu s$.

En el convertidor A/D de 10 bits de los PIC de gama media, según el fabricante, el tiempo de conversión es:

$$T_{CONV} = 1,5 \times T_{AD} \quad (9.25)$$

donde T_{AD} es el tiempo de conversión de un bit.

Para la operación correcta del convertidor A/D, el fabricante recomienda:

$$T_{AD} > 1,6 \mu s \quad (9.26)$$

Para $T_{AD} = 1,6 \mu s$, el tiempo de conversión A/D de 10 bits es $T_{CONV} = 18,4 \mu s$.

El valor de T_{AD} es igual al período del reloj del convertidor A/D. Dado que el reloj de convertidor A/D se puede obtener del oscilador principal del microcontrolador (tabla 9.1), el valor de la frecuencia de este oscilador debe establecerse de modo que se cumpla la condición expresada en (9.26). La tabla 9.3 señala el valor máximo de la frecuencia del oscilador principal del microcontrolador en las diferentes configuraciones posibles.

Tabla 9.3 Valor (máximo) que debe tener la frecuencia (F_{osc}) del oscilador principal del microcontrolador para que el tiempo de adquisición (mínimo) por bit (T_{AD}) sea de 1,6 μs , en las diferentes configuraciones.

ADCS1:ADCS0 en ADCON0	T_{AD}	F_{osc}	F_{osc} (para $T_{\text{AD}} = 1,6 \mu\text{s}$)
00	$2/F_{\text{osc}}$	$2/T_{\text{AD}}$	1,25 MHz
01	$8/F_{\text{osc}}$	$8/T_{\text{AD}}$	5 MHz
10	$32/F_{\text{osc}}$	$32/T_{\text{AD}}$	20 MHz
11	2 μs a 6 μs	-	-

La suma de los tiempos de adquisición y conversión constituye el tiempo de digitalización (T_{DIG}) de la señal analógica,

$$T_{\text{DIG}} = T_{\text{ACQ}} + T_{\text{CONV}} \quad (9.27)$$

Puede calcularse que, en los PIC, el menor valor de T_{DIG} está entre 20,4 μs y 38,4 μs .

Si una señal analógica se digitaliza periódicamente con un período T_s , denominado *período de muestreo*, la frecuencia de muestreo es $F_s = 1/T_s$. T_s debe ser mayor o igual que la duración del proceso de digitalización. Se recomienda esperar un tiempo igual a $2T_{\text{AD}}$ segundos antes de comenzar una nueva conversión A/D. Por tanto:

$$T_s \geq (T_{\text{DIG}} + 2T_{\text{AD}}) \quad (9.28)$$

En términos de la frecuencia de muestreo, esta expresión se convierte en

$$F_s \leq \frac{1}{T_{\text{DIG}} + 2T_{\text{AD}}} \quad (9.29)$$

Ejemplo 9.9

Cálculo del tiempo de digitalización para una conversión A/D de 10 bits en un PIC16F873 con un oscilador principal de 4 MHz.

Si la frecuencia del oscilador principal es $F_{\text{osc}} = 4 \text{ MHz}$, teniendo en cuenta la tabla 9.3, se selecciona la configuración dada por los bits ADCS1:ADCS0 = 01, con lo cual resulta $T_{\text{AD}} = 8/F_{\text{osc}} = 8/(4 \text{ MHz}) = 2,0 \mu\text{s}$ que cumple con la condición expresada en (9.26). El tiempo de una conversión A/D de 10 bits es entonces

$$T_{\text{CONV}} = 11,5 \times T_{\text{AD}} = 11,5 \times 2,0 \mu\text{s} = 23 \mu\text{s}$$

Si se considera el peor tiempo de adquisición, es decir, $T_{\text{ACQ}} = 20 \mu\text{s}$, que corresponde al caso en que la resistencia de salida del sistema medido sea la más alta posible (10 k Ω), el tiempo de muestreo resultante es

$$T_{\text{DIG}} = T_{\text{ACQ}} + T_{\text{CONV}} = 20 \mu\text{s} + 23 \mu\text{s} = 43 \mu\text{s}$$

Si la conversión A/D se realiza periódicamente, el muestreo de la señal analógica debe hacerse a una frecuencia menor o igual que $1/(T_{\text{DIG}} + T_{\text{AD}})$ Hz, de donde resulta que $F_s \leq 22\,222$ Hz.

Si se considera el mejor tiempo de adquisición, es decir, $T_{\text{ACQ}} = 10 \mu\text{s}$, que corresponde al caso en que la resistencia de salida del sistema medido sea de 50Ω , el tiempo de muestreo resultante es

$$T_{\text{DIG}} = T_{\text{ACQ}} + T_{\text{CONV}} = 10 \mu\text{s} + 23 \mu\text{s} = 33 \mu\text{s}$$

En estas condiciones, si la conversión A/D se realiza periódicamente, la frecuencia de muestreo debe ser inferior a 28 571 Hz.

9.3.3 Programación del módulo de conversión A/D

El módulo de conversión A/D se puede atender utilizando las técnicas de entrada por consulta o espera, o por interrupción. Los pasos que hay que seguir para medir la tensión analógica en un canal de entrada son:

1. Configurar el módulo de conversión A/D:
 - Configurar los terminales de los puertos A y C como entradas analógicas, referencias de tensión, o entradas o salidas digitales, colocando los valores apropiados en los bits PCFG3:PCFG0 de ADCON1 y en los registros TRISA y TRISC.
 - Configurar el formato del resultado de la conversión A/D con el bit ADFM del registro ADCON1.
 - Seleccionar la procedencia del reloj del módulo de conversión A/D y el valor apropiado del tiempo de conversión por bit (T_{AD}) mediante los bits ADCS1:ADCS0 de ADCON0.
 - Seleccionar el canal analógico de entrada mediante los bits CHS2:CHS0 de ADCON0.
 - Activar el módulo A/D mediante el bit ADON del registro ADCON0.
2. Si el módulo se atiende por interrupción, configurar la interrupción del módulo de conversión A/D.
 - Poner a 0 el indicador de interrupción del convertidor A/D, que es el bit ADIF del registro PIR.
 - Habilitar la interrupción del convertidor A/D, poniendo a 1 el bit ADIE del registro PIE.
 - Habilitar el sistema de interrupción del PIC, poniendo a 1 el bit GIE del registro INTCON.
3. Esperar el tiempo de adquisición (TACQ) requerido.

4. Comenzar la conversión A/D al poner en 1 el bit GO/DONE# del registro ADCON0.
5. Esperar que se complete la conversión A/D:
 - Si el tratamiento es por consulta o espera: cuando el bit GO/DONE# del registro ADCON0 vaya a 0 o el bit ADIF del registro PIR vaya a 1.
 - Si el tratamiento es por interrupción: cuando se produce la interrupción del convertidor A/D.
6. Leer el resultado de la conversión en los registros ADRESH y ADRESL. Poner el bit ADIF a 0 si fuese necesario.
7. Para realizar otra adquisición, ir a los pasos 1 ó 2 según proceda. Esperar al menos un tiempo igual a 2TAD antes de comenzar otra adquisición.

El ejemplo 9.10 muestra cómo programar el módulo de conversión A/D para adquirir señales analógicas utilizando la técnica de entrada por espera.

Ejemplo 9.10

Programación del módulo de conversión A/D en un PIC16F873 con un oscilador principal de 4 MHz, para adquirir la señal de cualquiera de los 5 canales analógicos de entrada en este microcontrolador.

La programación consta básicamente de 3 partes:

1. El bloque de iniciación del módulo de conversión A/D, con etiqueta Inicio, que configura las entradas del puerto A como analógicas y establece el formato del resultado de la conversión A/D.
2. La rutina Canal, que configura la palabra que ha de colocarse en el registro ADCON0, según el canal que se desee medir.
3. La rutina Mide, que recibe en W el número del canal que se desea medir y devuelve, también en W, el valor de la medición. Esta rutina le pasa a la rutina Canal el número del canal que se desea medir y obtiene de ella la palabra que se coloca en el registro ADCON0 para terminar así la configuración del módulo de conversión A/D, seleccionar el canal y comenzar su adquisición. A continuación, hace una pausa acorde al tiempo de adquisición recomendado y una vez transcurrido este tiempo, inicia la conversión A/D al poner a 1 el bit GO/DONE# de ADCON0; espera a que este bit vaya a 0, lo cual ocurre cuando finaliza la conversión A/D, y entonces devuelve los 8 bits más significativos del resultado en W.

A continuación, se da el listado del programa en lenguaje ensamblador.

; Ejemplo de programación del conversor A/D de 10 bits.

; Fosc = 4 MHz

```
List p=16F873
include "P16F873.INC"
AUX equ 0x20      ; Variable auxiliar.

org 0x00
goto Inicio
```

```
org 0x04
retfie
```

Inicio:

```
bsf STATUS, RP0 ; Seleccionar banco 1.
movlw 0xff      ; W con fff.
movwf TRISA     ; PORTA en entrada.
clr ADCON1      ; Todas las entradas de PORTA como analógicas y
                 ; el resultado de la conversión justificado a la izquierda.
```

;

; Aquí va el programa principal.

;

; Rutina Mide:

; Esta rutina realiza la conversión A/D de 10 bits del canal analógico seleccionado.
; La rutina selecciona el canal deseado y espera a que el resultado esté listo en los
; registros ADRESH y ADRESL. Devuelve en W los 8 bits más significativos de la
; conversión, es decir, el valor de ADRESH.
; La rutina supone un tiempo de conversión por bit Tad = 2 microsegundos y un reloj
; principal de 4 MHz en el PIC.
; Entradas: en W el número del canal a medir.
; Salidas: en W el resultado, en 8 bits, de la medición.

;

Mide:

```
nop          ; Esperar 2Tad = 4 microsegundos
nop
nop
nop
bcf STATUS, RP0 ; Seleccionar banco 0.
call Canal    ; Seleccionar la palabra que hay que depositar en
                ; ADCON0, según el canal.
movwf ADCON0   ; Esperar un tiempo de adquisición de 10 microsegundos
call Dem10us   ; Comenzar la conversión A/D.
```

Mide01:

```
btfsc ADCON0, GO ; Terminó la conversión?
goto Mide01       ; No - seguir esperando.
movf ADRESH, W    ; Sí - poner resultado en W.
bcf ADCON0, ADON ; Inhabilitar el convertidor A/D.
return            ; Retornar.
```

;

Rutina Canal.

; Esta rutina recibe en W el número del canal y devuelve en W la palabra
; que debe ponerse en ADCON0 para habilitar el convertidor A/D, seleccionar un
; canal de entrada y fijar el reloj del módulo de conversión A/D.
; El reloj del módulo de conversión A/D ha sido fijado en Fosc/8.
; Si Fosc = 4 MHz, entonces el tiempo de conversión por bit es Tad = 2 microsegundos.

```

;

Canal:
addwf    PCL, f
retlw    41h      ; Palabra para seleccionar el canal 0.
retlw    49h      ; Palabra para seleccionar el canal 1.
retlw    51h      ; Palabra para seleccionar el canal 2.
retlw    59h      ; Palabra para seleccionar el canal 3.
retlw    61h      ; Palabra para seleccionar el canal 4.

; Rutina Dem10us.
; Esta rutina demora más de 10 microsegundos.
;

Dem10us:
    movlw   .3
    movwf   AUX
Dem01:
    decfsz  AUX
    goto    Dem01
    return
    end

```

9.4 Calibración

Para poder interpretar el código obtenido en la conversión A/D en términos de tensión de entrada, es necesario conocer la característica de transferencia real de la etapa frontal y del propio convertidor A/D. Idealmente, si se prescinde de la cuantificación, esta característica es una línea recta cuya pendiente es la ganancia del sistema, de modo que se tiene

$$D = G \times V_x + V_0 \quad (9.30)$$

En la práctica, la característica real en unas condiciones determinadas (tensión de alimentación, temperatura ambiente, frecuencia de entrada) puede ser una recta con pendiente distinta de G y ordenada en el origen (offset) distinta de V_0 (figura 9.32a). Para determinar estos dos parámetros se puede aplicar a la entrada dos tensiones V_1 y V_2 conocidas, y a partir de las lecturas respectivas (D_1 y D_2), obtener la relación entre cualquier código D y la tensión V_x que lo produce:

$$V_x = \frac{V_2 - V_1}{D_2 - D_1} (D - D_1) + V_1 \quad (9.31)$$

Este proceso se denomina *calibración* y permite corregir las desviaciones sistemáticas (constantes) entre la respuesta ideal y la real. A menudo, se toma $V_1 = 0$ V, y V_2 es la tensión que produce la tensión a fondo de escala a la en-

trada del convertidor, V_{FS} , que si $V_0 = 0$ V, será $V_2 = V_{FS}/G = V_{ref}/G$. La figura 9.32b muestra una forma de realizar esta calibración, válida cuando las corrientes de entrada del multiplexor (o las resistencias de salida de los canales medidos) son suficientemente pequeñas para que su repercusión sea inferior a 1 LSB. En caso contrario, tanto en la conexión a masa como en la salida del generador de tensión de referencia hay que incluir una resistencia igual a la de salida de los canales medidos. Obsérvese que sólo se calibra la parte del sistema entre los puntos de aplicación de las entradas conocidas y la salida, en el momento de aplicar las tensiones conocidas.

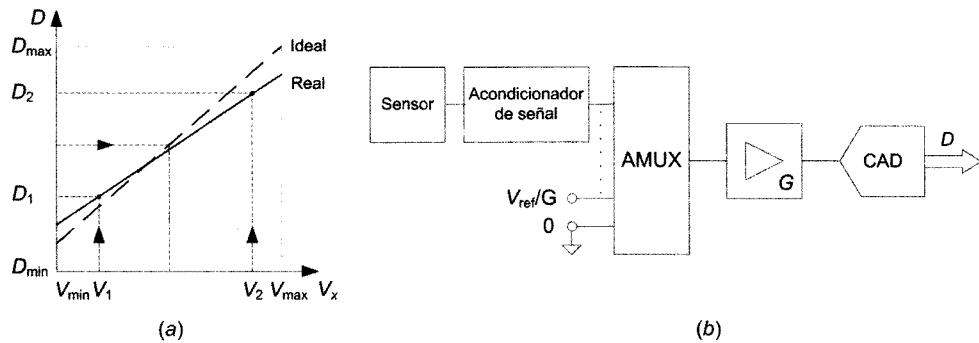


Figura 9.32 La calibración de un sistema de adquisición de datos en dos puntos consiste en aplicar dos tensiones conocidas (V_1 , V_2) para determinar la recta de transferencia real.

Si la característica de transferencia es una recta de pendiente muy próxima a la característica ideal, puede ser suficiente un solo punto de calibración, que suele ser $V_1 = 0$ V.

Si la respuesta real del sistema no es una línea recta, se puede dividir el rango de tensiones de entrada en dos o más intervalos contiguos (subrangos) en los que la característica de transferencia real sea una recta, y calibrar en cada uno de dichos subrangos. A partir de una lectura D , hay que determinar primero a qué subrango corresponde, y después aplicar la ecuación respectiva. La posible no linealidad de la respuesta se puede determinar aplicando una tensión conocida V_3 , tal que $V_1 < V_3 < V_2$: si el valor obtenido al aplicar (9.31) no coincide con V_3 , la respuesta no es lineal.

9.5 Interfaces directas entre sensor y microcontrolador

Las señales analógicas con información en su duración, no en su amplitud, pueden ser digitalizadas mediante un simple temporizador (o contador) digital, como los disponibles en muchos microcontroladores, siempre y cuando su entrada sea del tipo *Schmitt Trigger* (ST). En caso contrario, hay que

poner un ST externo. La duración de la señal se determina contando ciclos del reloj interno desde un flanco de la señal hasta el siguiente. Si los flancos no son abruptos y tienen superpuesto ruido de amplitud mayor que el ancho de la banda de histéresis del ST, el inicio y el final del conteo se pueden producir en instantes erróneos.

Para introducir la información en la duración de una señal, cuando el punto de partida es un sensor basado en la variación de una resistencia, se puede emplear el circuito de la figura 9.33: primero se carga el condensador con la tensión de salida alta ("1", V_{OH}) del terminal 1, mientras se mantiene el terminal P en estado de alta impedancia; después se pone el terminal 1 en estado de alta impedancia y el terminal P a 0 (V_{OL}), con lo cual C se descarga a través de R_x . La duración de la descarga (hasta V_{TL}), detectada con el ST, es proporcional al producto $R_x C$, y depende también de las tensiones de salida V_{OH} y V_{TL} . Aunque en principio se podría aplicar el mismo método midiendo los tiempos de carga, está demostrado que el umbral de detección "bajo" del Schmitt Trigger (V_{TL}) es menos susceptible al ruido que el umbral de detección "alto" (V_{TH}).

Para tener un resultado independiente de C y de las tensiones V_{OH} y V_{TL} , se puede emplear el circuito de calibración de la figura 9.34a, donde R_{c1} y R_{c2} son resistores conocidos. Se procede tres veces como en la figura 9.33, pero la descarga se hace cada vez a través de un resistor distinto. La resistencia desconocida se puede estimar mediante

$$R_x = \frac{N_x - N_{c1}}{N_{c2} - N_{c1}} (R_{c2} - R_{c1}) + R_{c1} \quad (9.32)$$

donde N_x , N_{c1} y N_{c2} son los tiempos de descarga (en ciclos de reloj) a través de, respectivamente, R_x , R_{c1} y R_{c2} . Uno de los dos resistores conocidos puede ser un cortocircuito, pero es mejor elegir valores próximos a los valores extremos de R_x .

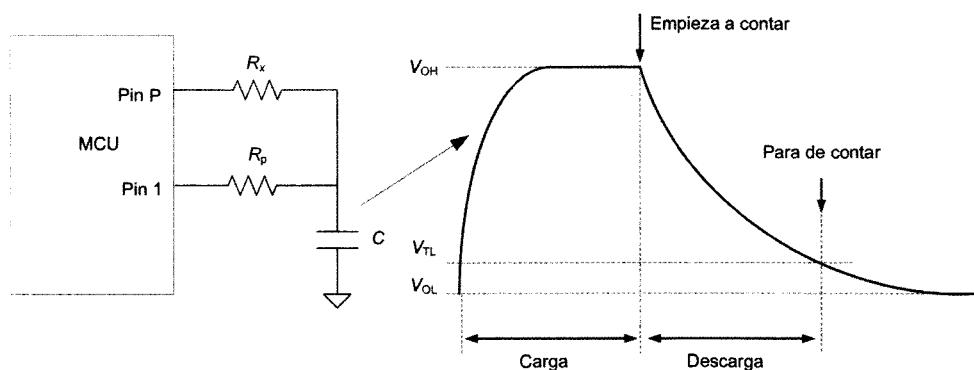


Figura 9.33 Medida de un resistor desconocido a partir del tiempo necesario para descargar un condensador. R_p limita la corriente de carga del condensador a menos de 20 mA.

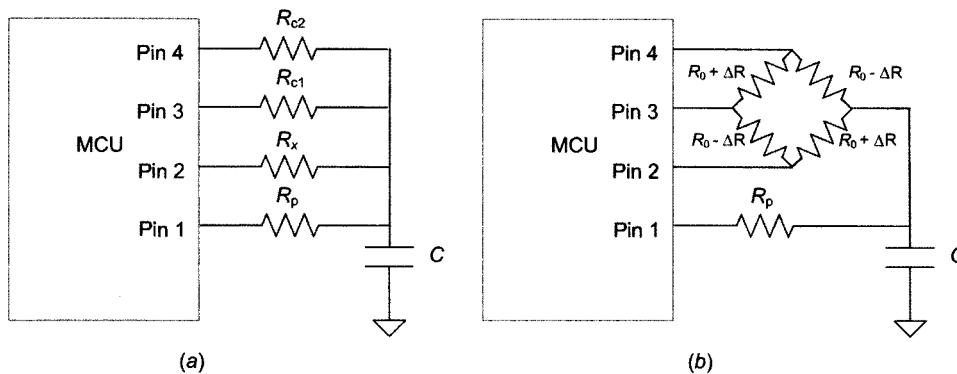


Figura 9.34 (a) Si con el método de la figura 9.33 se miden dos resistencias conocidas, se puede determinar R_x sin conocer C ni las tensiones de salida del microcontrolador. (b) Un puente de sensores se puede conectar como un circuito con tres terminales de entrada y uno de salida, y aplicarle el mismo método.

Si se aplica este mismo método a un puente de sensores conectado tal como se muestra en la figura 9.34b, si las lecturas respectivas al descargar el condensador a través de cada terminal (de 2 a 4), son N_1 , N_2 y N_3 , se tiene

$$\frac{N_1 - N_3}{N_2} = \frac{\Delta R}{R_0} \quad (9.33)$$

En la figura 9.34, el sensor de resistencia variable y el puente de sensores están conectados al microcontrolador sin otro elemento que un condensador, que no influye en el resultado. Se habla entonces de *interfaz directa*, porque no media ningún circuito integrado entre el sensor y el microcontrolador. Los microcontroladores que integran un convertidor A/D de resolución suficiente para la aplicación pretendida, pueden conectarse directamente a sensores dispuestos en circuitos cuya salida sea una tensión, sin necesidad de amplificarla; pero si el sensor es un puente de resistencias, la entrada del convertidor debe ser diferencial. En la figura 9.34, basta que el microcontrolador tenga un temporizador, no hace falta un CAD.

El método de la figura 9.33 se puede aplicar también a sensores basados en una variación de capacidad, intercambiando las posiciones de la resistencia y el condensador. El circuito de calibración correspondiente incluye entonces dos condensadores conocidos, uno de los cuales puede ser un circuito abierto ($C = 0$, figura 9.35). Entonces, en la fase de carga se cargan las capacidades parásitas entre el nodo N y masa, y en la fase de descarga se descarga dicha capacidad a través de R . Si para tener un tiempo de descarga suficientemente largo, R se elige grande (más de $1 \text{ M}\Omega$), el circuito será muy susceptible a interferencias, incluso si las conexiones a C_x son muy cortas.

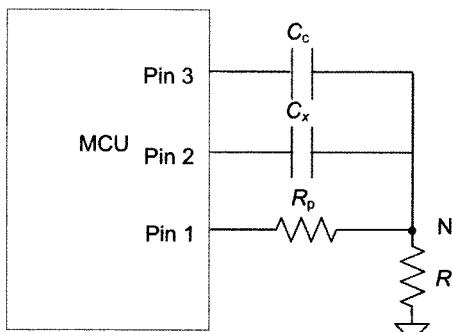


Figura 9.35 Medida de un condensador desconocido a partir del tiempo necesario para descargarlo a través de un resistor R . R_p limita la corriente de carga del condensador a menos de 20 mA. Si con el mismo método se miden dos capacidades conocidas (circuito abierto y C_c), se puede determinar C_x sin conocer R ni las tensiones de salida del microcontrolador.

Este método de medida de resistencias y capacidades viene limitado por una parte por la cuantificación al contar (el resultado sólo puede ser un múltiplo entero del ciclo de reloj), y por otra parte por la incertidumbre en el instante de cruce del umbral de detección debida al ruido superpuesto a la tensión de descarga del condensador o a la propia tensión umbral, V_{TL} .

La cuantificación depende de cómo se atienda a la detección del cruce del umbral del *Schmitt Trigger* (apartado 5.1.2). Si se hace por consulta, puede que la detección ocurra justo después de haber consultado, con lo cual la indeterminación puede ser de entre un ciclo de reloj y el número de ciclos entre consultas. Si se atiende mediante interrupciones, la indeterminación puede ser de entre un ciclo de reloj y el número de ciclos de la instrucción más larga, cuya ejecución debe acabar antes de atender la interrupción. Si se dispone de un módulo de captura (apartado 6.2), el valor del temporizador que va contando el tiempo de descarga, es capturado en cuanto se detecta el cruce del umbral V_{TL} .

El efecto del ruido en el disparo del *Schmitt Trigger* depende de la pendiente de la señal: para una amplitud de ruido determinada, cuanto más lenta sea la señal, mayor será el error en el tiempo de disparo. Por ello, si la constante de tiempo del circuito RC formado es grande (para tener un número de cuentas alto), la pendiente será más lenta y el error aumentará. Pero si la constante de tiempo es muy breve, para tener una pendiente alta, el número de cuentas obtenido será también pequeño, y la cuantificación tendrá un mayor efecto relativo. El valor óptimo de la constante de tiempo depende del nivel de ruido en el umbral y en la señal. En un circuito impreso bien diseñado, y con la tensión de alimentación desacoplada, puede estar entre 1 ms y 3 ms,

y obtener una resolución de 10 a 12 bits. Si predomina el efecto del ruido, se puede mejorar la resolución promediando varias lecturas.

9.6 La etapa de salida para salidas analógicas

Para disponer de una señal de salida analógica a partir de los datos internos del microcontrolador, debe procederse de forma inversa a como se procede en la etapa frontal: hay que reconstruir una señal a partir de su valor en instantes determinados, y a veces hay que suministrar o distribuir varias señales a la vez. Estas funciones las realiza la etapa de salida (*analog back end*)

9.6.1 Convertidores D/A

Un convertidor digital-analógico (CDA) (figura 9.36a), ofrece en cada instante de tiempo una tensión (o una corriente) de salida cuya amplitud corresponde al código digital de entrada ($B_{n-1}B_{n-2}\dots B_1B_0$), según fracciones de una tensión de referencia,

$$V_o = V_{\text{ref}} \left(\frac{B_{n-1}}{2} + \frac{B_{n-2}}{2^2} + \dots + \frac{B_1}{2^{n-1}} + \frac{B_0}{2^n} \right) \quad (9.34)$$

Cada bit B_i puede valer 1 ó 0. La función realizada es, pues, el producto de una tensión analógica (V_{ref}) y un código digital. La figura 9.36b muestra un circuito clásico para realizarla: cada bit controla (abre o cierra) un interruptor cuya posición en la red $R-2R$ está tanto más lejos de la salida cuanto mayor sea la significación (o peso) del bit. Los convertidores D/A actuales, incluidos los integrados en microcontroladores, se realizan con una red de capacidades (C-2C), que son más fáciles de integrar, y cuyo conexionado es ligeramente distinto, pero que tiene el mismo efecto final.

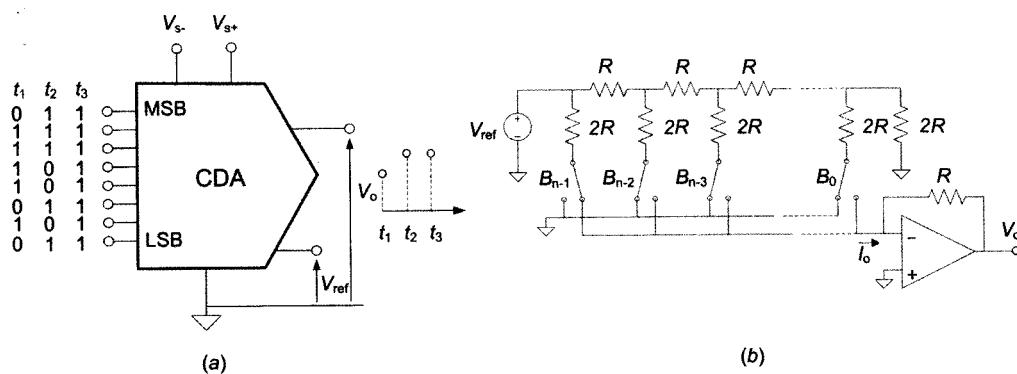


Figura 9.36 Convertidor D/A: función y circuito clásico basado en una red de resistencias de valor R y $2R$.

Las características de transferencia ideal y real de un CDA se pueden describir con los mismos parámetros empleados para un CAD (figura 9.27).

9.6.2 Desmultiplexado analógico

Para tener simultáneamente varias señales analógicas de salida, hay convertidores D/A múltiples de hasta 16 canales y 16 bits. Pero si la frecuencia de las señales no es muy alta, se puede emplear un sólo CDA rápido y un desmultiplexor analógico, con función inversa a la de los multiplexores de entrada (figuras 9.9 y 9.10). La figura 9.37 muestra la conexión: la tensión de salida correspondiente a cada canal es conectada al amplificador de muestreo y retención respectivo, cerrando el interruptor correspondiente del desmultiplexor. El condensador de retención de cada canal debe ser suficientemente grande para no descargarse de forma apreciable hasta que se refresque o actualice la tensión desde el convertidor D/A, pero cuanto mayor sea, más tardará en cargarse. La desmultiplexación analógica se hace con los multiplexores descritos en el apartado 9.2.4, pues, a diferencia de los multiplexores digitales, son reversibles (figura 9.21).

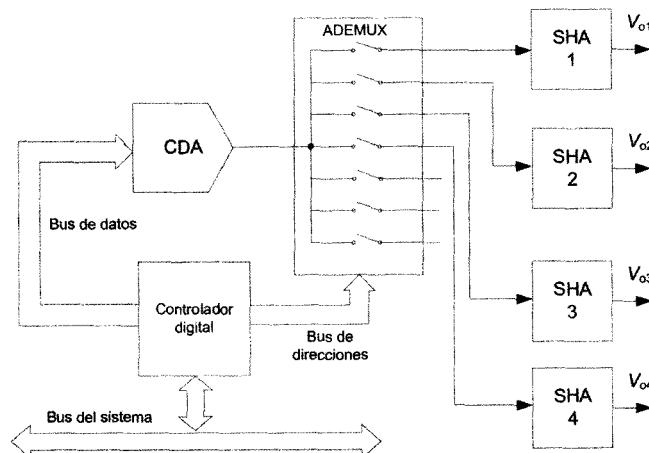


Figura 9.37 Desmultiplexado analógico para obtener varias señales analógicas de salida con un solo convertidor D/A.

9.6.3 Métodos de extrapolación

La tensión de salida del convertidor digital-analógico corresponde al código aplicado a su entrada en cada instante. Si el CDA tiene un registro de entrada, su salida permanecerá constante hasta que se actualice el valor de dicho registro. Del mismo modo, si el CDA se conecta brevemente a un amplificador de muestreo y retención, la salida de éste permanecerá (aproximadamente) constante durante el tiempo que dure la conexión.

madamente) constante hasta que se muestree otra tensión. En ambos casos se dice que hay una *extrapolación, o retención, de orden cero*, (*zero-order hold, ZOH*), porque el valor de la salida se mantiene constante hasta que llega otro valor distinto. La salida tiene forma de señal escalonada y si se desea obtener una forma aparentemente lisa, hay que actualizar a menudo el valor de la salida (figura 9.38). El módulo de la función de transferencia de la extrapolación de orden cero es

$$|H(f)| = \left| T \frac{\sin \pi f T}{\pi f T} \right| \quad (9.35)$$

donde T es el tiempo que dura la retención. A efectos prácticos, esto significa, por ejemplo, que para que la amplitud de una sinusoides de amplitud A y frecuencia f no difiera de la sinusoides "escalonada" en más de $A/2^{10}$, el periodo de retención debe cumplir: $fT < 41$; es decir, que hay que aproximar la sinusoides mediante 41 tramos escalonados.

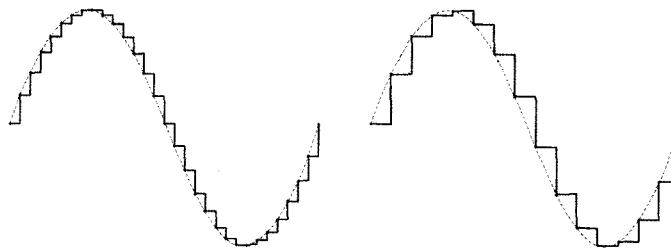


Figura 9.38 Efecto de la frecuencia de las muestras de salida en la forma de onda de una señal reconstruida cuando se emplea extrapolación de orden cero. Para que la forma de onda reconstruida se aproxime suficientemente a la forma deseada, hay que proporcionar un número de muestras elevado (izquierda). Si se sacan pocas muestras por unidad de tiempo, la forma reconstruida dista mucho de la deseada (derecha).

9.6.4 Salidas PWM

El nivel de continua (valor medio) de un pulso de anchura modulada (figura 9.39) depende de la duración relativa del pulso, o ciclo de trabajo (apartado 6.2.3), según

$$V_0 = (V_{OH} - V_{OL}) \frac{T_{ON}}{T} \approx V_{DD} \frac{T_{ON}}{T} \quad (9.36)$$

donde T es el periodo del pulso. Se ha supuesto que durante T_{ON} la salida es V_{DD} y durante T_{OFF} la salida es 0 V. En la práctica, según el procesador y la corriente de salida (I_{OH}, I_{OL}), V_{OH} puede tener un valor mínimo de hasta $V_{DD} - 0,7$ V, y V_{OL} un valor máximo de hasta 0,6 V. En cualquier caso, ajustando el ciclo de trabajo se puede obtener una tensión continua del nivel deseado, con una estabilidad que dependerá directamente de la de V_{DD} . Si la corriente que puede dar la puerta de salida no es suficiente, se puede conectar un compara-

dor de tensión entre la salida PWM y el filtro de paso bajo. Para obtener una salida de mayor amplitud, se puede emplear un comparador con salida de colector o drenador abierto.

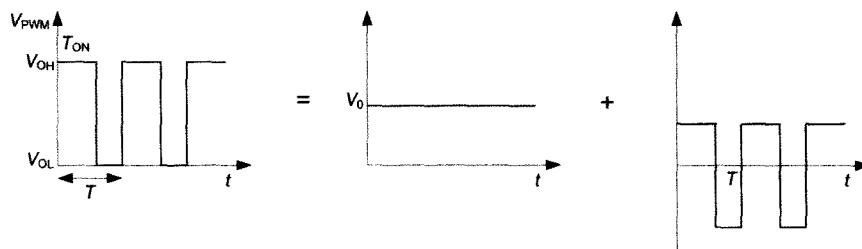


Figura 9.39 Una señal PWM se puede descomponer en una tensión continua y una tensión rectangular sin nivel de continua, que a su vez se puede descomponer en varias sinusoides armónicas, la primera con amplitud $(V_{OH} - V_{OL})2/\pi$.

Si la señal PWM se filtra con un filtro de paso bajo de primer orden y frecuencia de corte f_c , la amplitud, de pico a valle, del primer armónico a la salida del filtro será

$$V_1 = \frac{2V_{DD}}{\pi} \frac{1}{\sqrt{1 + \left(\frac{f}{f_c}\right)^2}} \quad (9.37)$$

Los armónicos de orden superior tienen una amplitud que decrece con el cuadrado de su orden, y sufrirán una atenuación mayor. Si se desea que el rizado de la salida debido al primer armónico sea inferior a $1 \text{ LSB}/2^q$, en un sistema de N bits, la frecuencia de corte del filtro deberá cumplir,

$$f_c < \sqrt{\frac{\pi}{2}} \frac{f}{2^{N+q} - 1} \approx \frac{1,25f}{2^{N+q}} \quad (9.38)$$

Por ejemplo, si $f = 20 \text{ kHz}$ y $N = 8$, para que el rizado sea inferior a $1/4 \text{ LSB}$, deberá ser $f_c < 25 \text{ Hz}$. Un filtro activo permite obtener una menor impedancia de salida que un filtro pasivo, pero añade una tensión de offset a V_0 y limita los valores máximo y mínimo de V_0 a los disponibles a la salida del amplificador operacional.

Además del rizado, también la resolución en T_{ON} (ΔT_{ON}), que es de unos 8 ó 10 bits en los PIC de gama media (apartado 6.2.3), limitará la resolución en V_0 . Según (9.36), ΔV_0 se puede hacer más pequeña aumentando T , pero esto conlleva la necesidad de reducir la frecuencia de corte del filtro de paso bajo, y por lo tanto, una respuesta transitoria más lenta cuando se deseé cambiar de un valor de V_0 a otro.

Si interesa obtener una tensión de salida sinusoidal (con un nivel de continua superpuesto), basta que la moduladora de la señal PWM tenga la frecuencia deseada. Ahora bien, el filtro paso bajo de salida deberá ser necesariamente activo y de orden 3 o superior si se desea un rizado compatible con un sistema de unos 8 bits o más. Si la máxima frecuencia disponible para la señal PWM es de unos 20 kHz, con este método se pueden generar señales de hasta 1 kHz, válidas como señales de prueba o para comunicación acústica.

9.6.5 Protecciones de salida

La tensión, corriente y potencia que pueden suministrar y soportar los terminales de salida de los microcontroladores están limitados a valores relativamente bajos, del orden de $V_{SS} - 0,3\text{ V}$ a $V_{DD} + 0,3\text{ V}$, y $\pm 25\text{ mA}$. Esto obliga, por una parte, a emplear activadores (*drivers*) para accionar las cargas de mayor potencia, y por otra parte a proteger dichos terminales frente a sobretensiones y sobrecorrientes. Ahora bien, a diferencia de las protecciones de entrada, que deben diseñarse en función del nivel de protección que se desee obtener en una situación anómala (apartado 9.2.3), en la salida pueden aparecer tensiones y corrientes peligrosas en condiciones ordinarias.

Cuando se interrumpe el suministro de corriente a una carga inductiva, la tendencia de la corriente a seguir circulando provoca una diferencia de potencial entre los terminales del interruptor abierto, que puede ser de varias decenas de voltios incluso para corrientes e inductancias pequeñas. Para evitar dicha tensión, se puede poner un diodo (o un varistor) conectado según muestra la figura 9.40a: al abrir el interruptor (i.e., pasar de 1 a 0), la corriente que circulaba por la bobina circulará inicialmente por el diodo, y se irá extinguiendo.

Para las cargas capacitivas, el problema es la corriente al conectarlas, pues viene limitada sólo por la resistencia interna de la fuente de tensión. La solución es poner un limitador de corriente, tal como una resistencia con coeficiente de temperatura negativo (NTC) (figura 9.40b): el elevado valor de esta resistencia en frío (unos 250Ω como mínimo), limita la corriente inicial; conforme se va calentando, su resistencia disminuye y permite la carga rápida del condensador.

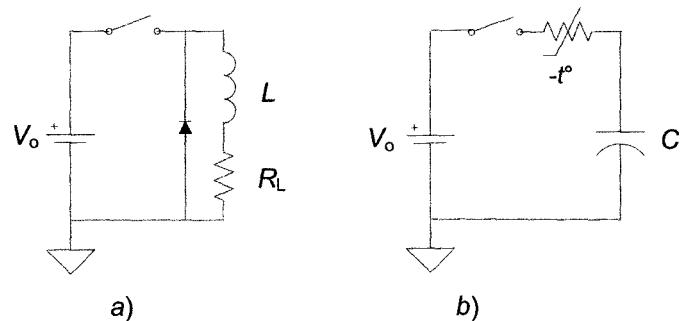


Figura 9.40 Protecciones de salida al alimentar (a) una carga inductiva: limitación de la tensión, y (b) una carga capacitiva: limitación de la corriente. El interruptor representa la acción de pasar de 0 a 1 (cerrar) y de 1 a 0 (abrir) en la puerta de salida.

Bibliografía

- [1] Anónimo, *EDN's 2005 Microprocessor/Microcontroller Directory*. EDN. Agosto 5, 2004.
- [2] Anónimo, *Embedded Control Handbook Update 2000*. Chandler (AZ): Microchip Technology, 1999.
- [3] Anónimo, *Fundamentals of RS-232 Serial Communications*. Application Note 83. Dallas (TX): Dallas Semiconductor.
- [4] Anónimo, *MCS®51 Microcontroller Family User's Manual*. Santa Clara (CA): Intel, Febrero 1994.
- [5] Anónimo, *MPLAB® IDE user's guide*. Chandler (AZ): Microchip Technology, 2005.
- [6] Anónimo, *PIC16F87X Data Sheet 28/40-Pin 8-Bit CMOS FLASH Microcontrollers*. Chandler (AZ): Microchip Technology, 2001.
- [7] Anónimo, *PICmicro™ Mid-Range MCU Family Reference Manual*. Chandler (AZ): Microchip Technology, Diciembre 1997/DS33023A.
- [8] Anónimo, *The I²C-bus specification. Version 2.1*. Eindhoven: Philips Semiconductors, Enero 2000.
- [9] B. Baker, *Anti-aliasing. Analog Filters for Data Acquisition Systems*. AN699. Chandler (AZ): Microchip Technology, 1999.
- [10] F. Pardo Carpio, *Edu-PIC: Tarjeta de desarrollo para sistemas basados en PIC. Manual del usuario*, Universitat de València, (<http://tapec.uv.es/edu-pic>), Julio 2002.
- [11] F. Reverter y R. Pallàs Areny, *Direct Sensor-to-Microcontroller Interface Circuits: Design and Characterization*. Barcelona: Marcombo, 2005.
- [12] F. Valdés Pérez y A. Selva Barthelemy, *Fundamentos Técnicos de Computación*. La Habana (Cuba): Editorial ISPJAE, 1986.
- [13] J. Wharton, *An introduction to the Intel® MCS-51™ single-chip microcomputer family*. AP-69. Santa Clara (CA): Intel, 1980.
- [14] J. A. González Vázquez, *Introducción a los microcontroladores: hardware, software y aplicaciones*. Madrid: McGraw-Hill/Interamericana de España, 1992.
- [15] J. B. Peatman, *Design with microcontrollers*. Nueva York: McGraw Hill, 1988.

- [16] J. B. Peatman, *Design with PIC microcontrollers*. Upper Saddle River (NJ): Prentice-Hall, 1997.
- [17] J. M. Irazabal, S. Blozis, *I²C Manual*. AN10216-01. Eindhoven: Philips Semiconductors, Marzo 24, 2003.
- [18] R. Cravotta, *The 32nd annual microprocessor directory*. EDN, August 4, 2005.
- [19] R. Pallàs Areny, *Adquisición y distribución de señales*. Barcelona: Marcombo, 1993, reimpresión 2003.
- [20] R. Richey, *How to Implement ICSPTM Using PIC16CXXX OTP MCUs*. Chandler (AZ): Microchip Technology, 1999.
- [21] S. Bowling, *Understanding A/D Converter Performance Specifications*. AN693. Chandler (AZ): Microchip Technology, 2002.

Anexo. Siglas y acrónimos utilizados en el libro

A/D	Analógico/Digital.
ACC	<i>Accumulator.</i> Acumulador.
ADEMUX	Desmultiplexor analógico.
AFE	<i>Analog front end.</i> Etapa de entrada analógica.
ALU	<i>Arithmetic and Logic Unit.</i> Unidad aritmética y lógica.
AMUX	Multiplexor analógico.
ASCII	<i>American Standard Code for Information Interchange.</i> Código americano normalizado para el intercambio de información.
BISYNC	<i>IBM Binary Synchronous Communications Protocol.</i> Protocolo binario de IBM para comunicaciones sincrónicas.
BOR	<i>Brown - out Reset.</i> Reset por fallo de la alimentación.
CAD	Convertidor analógico-digital.
CCITT	Comité Consultivo Internacional de Telefonía y Telegrafía.
CCP	<i>Compare/Capture/PWM.</i> Comparación/Captura/PWM.
CDA	Convertidor digital-analógico.
CEI	Comisión Electrotécnica Internacional.
CISC	<i>Complex Instruction Set Computer.</i> Computador con repertorio de instrucciones complejas.

CM	Ciclo de máquina.
CMOS	<i>Complementary Metal Oxide Semiconductor.</i> Semiconductor complementario de óxido de metal.
CMRR	<i>Common Mode Rejection Ratio.</i> Relación de rechazo del modo común.
CPU	<i>Central Processing Unit.</i> Unidad central de procesamiento.
CSMA/CD	<i>Carrier Sense, Multiple Access with Collision Detection.</i> Acceso múltiple por detección de portadora con detección de colisión.
DCE	<i>Data Communication Equipment.</i> Equipo de comunicación de datos.
DIP	<i>Dual In line Package.</i> Encapsulado de doble fila.
DMA	<i>Direct Memory Access.</i> Acceso directo a memoria.
DNL	<i>Differential NonLinearity.</i> No linealidad diferencial.
DR	<i>Dynamic Range.</i> Margen o rango dinámico.
DSP	<i>Digital Signal Processor.</i> Procesador digital de señales.
DTE	<i>Data Terminal Equipment.</i> Equipo terminal de datos.
EEPROM	<i>Electrical Erasable Programmable Read – Only Memory.</i> Memoria de solo lectura que se programa y borra eléctricamente.
EIA	<i>Electronic Industries Association.</i> Asociación de Industrias Electrónicas.
ENOB	<i>Equivalent Number of Bits.</i> Número de bits equivalente.
EPROM	<i>Erasable Programmable Read – Only Memory.</i> Memoria de solo lectura que se programa y borra.
FIFO	<i>First In First Out.</i> Primero en entrar, primero en salir.

FPB	Filtro de paso bajo, o filtro pasabajas.
FPGA	<i>Field Programmable Gate Array</i> . Matriz de puertas programable in situ.
FS	<i>Full Scale</i> . Fondo de escala.
FSR	<i>Full Scale (Input) Range</i> . Valor de fondo de escala (de entrada).
GIE	<i>Global Interrupt Enable bit</i> . Bit de habilitación global de las interrupciones.
GPR	<i>General Purpose Register</i> . Registro de propósito general.
HDLC	<i>High - level Data Link Control</i> . Control de enlace de alto nivel de datos.
I ² C	<i>Inter-Integrated Circuit</i> . Bus para conectar circuitos integrados.
ICSP	<i>In-Circuit Serial Programming</i> . Programación “en el circuito”.
INL	<i>Integral NonLinearity</i> . No linealidad integral.
LCD	<i>Liquid Crystal Display</i> . Pantalla de cristal líquido.
LED	<i>Light Emitting Diode</i> . Diodo emisor de luz.
LIFO	<i>Last In First Out</i> . Último en entrar, primero en salir.
LSB	<i>Least Significant Bit</i> . Bit menos significativo.
módem	<i>Modulator – demodulator</i> . Modulador – desmodulador.
MOV	<i>Metal-Oxide resistor</i> . Resistor de óxido metálico.
MPASM	Macro ensamblador para los microcontroladores PIC.
MSB	<i>Most Significant Bit</i> . Bit más significativo.

MSSP	<i>Master Synchronous Serial Port.</i> Puerto serie sincrónico servidor.
NTC	<i>Negative Temperature Coefficient.</i> Coeficiente de temperatura negativo.
OST	<i>Oscillator Start - up Timer.</i> Temporizador de arranque del oscilador.
OTP	<i>One Time Programmable.</i> Programable una vez.
PC	<i>Program Counter.</i> Contador de programa.
PCON	<i>Power Control.</i> Control de la alimentación
PIC	<i>Programmable Integrated Circuit.</i> Acrónimo de PICmicro, nombre utilizado por Microchip para denominar sus microcontroladores.
PLD	<i>Programmable Logic Devices.</i> Dispositivos lógicos programables.
POR	<i>Power - on Reset.</i> Reset por encendido.
PSP	<i>Parallel Slave Port.</i> Puerto paralelo esclavo.
PTC	<i>Positive Temperature Coefficient.</i> Coeficiente de temperatura positivo.
PWM	<i>Pulse Width Modulation.</i> Modulación de pulsos en anchura.
PWRT	<i>Power - up Timer.</i> Temporizador de arranque de encendido.
RAM	<i>Random Access Memory.</i> Memoria de acceso aleatorio.
RDD	Registro de direcciones de datos.
RI	Registro de Instrucción.
RISC	<i>Reduced Instruction Set Computer.</i> Computador con repertorio de instrucciones reducidas.
ROM	<i>Read Only Memory.</i> Memoria de solo lectura.

RS-232C	<i>Recommended Standard</i> número 232 revisión C.
RTC	<i>Real Time Clock</i> . Reloj de tiempo real.
SCI	<i>Serial Communication Interface</i> . Interfaz de comunicación serie. Nombre alternativo del puerto USART en los microcontroladores PIC.
SCL	<i>Serial Clock Line</i> . Línea de reloj del interfaz I ² C.
SDA	<i>Serial Data Line</i> . Línea de datos del interfaz I ² C.
SDLC	<i>Synchronous Data Link Control</i> . Control de enlace sincrónico de datos.
SFR	<i>Special Function Register</i> . Registro de funciones especiales.
SI	Sistema Internacional de Unidades.
SP	<i>Stack Pointer</i> . Puntero de la pila.
SPI	<i>Serial Peripheral Interface</i> . Interfaz serie para periférico.
SPP	<i>Slave Parallel Port</i> . Puerto paralelo esclavo.
SSP	<i>Synchronous Serial Port</i> . Puerto serie sincrónico.
ST	<i>Schmitt Trigger</i> . Disparador de Schmitt.
STATUS	Registro o bits de estado.
UIT	Unión Internacional de Telecomunicaciones.
USART	<i>Universal Synchronous Asynchronous Transmitter Receiver</i> . Transmisor receptor universal sincrónico asincrónico.
USB	<i>Universal Serial Bus</i> . Bus serie universal.
W	<i>Working Register</i> . Registro de trabajo.

WDT *Watchdog Timer.* Perro guardián.

XTAL Cristal.

ZOH *Zero-order hold,* Retención de orden cero.

Índice alfabético

- \$ [operador \$], . . . 114, 130, 132, 134
- Acarreo, 15, 30, 74, 94, 99
- ACC, ver acumulador.
- Acceso directo a memoria, . . . 154
- Accumulator*, ver acumulador.
- Acondicionador de señal, . . . 276
- Acoplamiento en alterna, . . . 282
- Acoplamiento en continua, . . 281
- Actuador, 275
- Acumulador, 14 - 15, 24, 31, 88, 220
- Aislamiento (galvánico), 295
- Alias, ver señales falsas
- Amplificador de instrumentación, 291 - 293
- Amplificador de muestreo y retención, 282 - 284, 300 - 301, 306, 321
- Ancho de banda de un circuito, 281, 294, 300
- Ancho de banda de una señal, 280, 284
- Anchura de código, 303
- Anidamiento de subrutinas, 87, 231
- Archivo hexadecimal, . . . 136 - 143
- Arithmetic and Logic Unit* (ALU), ver unidad aritmética y lógica.
- Atenuador compensado en frecuencia, 287
- Atenuador de tensión alterna, 286 - 287
- Atenuador de tensión continua, 285 - 286
- Banco, 68, 69, 70
- Banda de paso, 299
- Banda de transición, 299
- Banda rechazada, 299
- Bank*, ver banco.
- Base de la pila, 85
- Base de tiempos, . . . 200, 234 - 240
- Biblioteca, 82, 136, 145 - 147
- Biestable almacenador de un bit, 150 - 151
- Biestable D *latch*, 150 - 151
- Bits de configuración, 36 - 37
- Borrado de la memoria FLASH, 60
- Bus de datos, . 11, 22, 162 - 163, 182
- Bus de direcciones, 11, 22
- Byte, ver octeto.
- Calibración, 292, 304, 315 - 318
- Capacidad de la memoria, 45, 55 - 56, 62
- Característica de transferencia, 277 - 278, 303 - 304, 315 - 316
- Carga capacitiva, 297, 317 - 318, 325
- Carga inductiva, 324 - 325
- Central Processing Unit* (CPU), ver unidad central de procesamiento.
- Ciclo activo, ver ciclo de trabajo.
- Ciclo de instrucción, 30, 76, 88, 91, 103
- Ciclo de máquina, 32 - 34
- Ciclo de trabajo, 210 - 215, 278, 322

- Ciclo útil, ver ciclo de trabajo.
- Circuitos integrados independientes, 25
- Code width*, ver anchura de código.
- Código absoluto, 80 - 82, 122, 137 - 138
- Código binario con complemento a 2, 305
- Código binario directo, 305
- Código de operación, 82 - 83
- Código relativo o relocalizable, 80 - 81, 115, 121 - 122, 130 - 131, 138 - 139
- Comentario, 104
- Complemento a 2, 110, 140, 193, 305
- Comunicación bidireccional no simultánea, 262
- Comunicación bidireccional simultánea, 259
- Condensador de retención, 301, 309, 321
- Constantes, 105 - 106
- Contador de programa, 15, 18, 32, 62 - 64, 72, 86 - 87, 95, 220, 224, 229, 231
- Controlador incrustado, 12
- Convertidor analógico-digital, 276 - 277, 302 - 306, 318
- Convertidor de aproximaciones sucesivas, 277, 302 - 306
- Convertidor digital-analógico, 320, 321
- Convertidor flash, 277
- Core*, ver núcleo.
- Criterio de Nyquist, 282
- CRM, ver relación de rechazo del modo común.
- Cuantificación al contar, 319
- D latch*, ver biestable almacenador de un bit.
- Debounce*, ver rebote.
- Desbordamiento de la pila, 88
- Desmultiplexor analógico, 321
- Desplazamiento, 57 - 58, 63, 70 - 71, 184
- Diafonía estática, 297 - 298
- Differential nonlinearity*, ver no linealidad diferencial.
- Digital Signal Processor (DSP)*, ver procesador digital de señales.
- Dirección de una celda, 56 - 57
- Dirección lógica, 57
- Direccionamiento directo, 70 - 71, 73, 84 - 85, 95, 126 - 127
- Direccionamiento indirecto, 15, 69, 70 - 73, 84 - 85, 91, 93, 127
- Direccionamiento inmediato, 84 - 85
- Direct Memory Access (DMA)*, ver acceso directo a memoria.
- Directiva, 103 - 104
- Display*, 149, 177
- Droop rate*, ver velocidad de descarga.
- Duty cycle*, ver ciclo de trabajo.
- Dynamic range*, ver rango dinámico.
- E/S por consulta, 153 - 154
- Efecto de carga, 287 - 290, 294, 296
- Effective number of bits*, ver número de bits efectivo.
- Embedded controller*, ver controlador incrustado.

- Encapsulados, 60
Enlazador, 81 - 82
Ensamblaje, 80
Entrada asimétrica, . 289 - 291, 305
Entrada diferencial, . . . 289 - 290
Entrada flotante, 290
Entrada seudodiferencial, . . . 290
Error de cero, 303 - 304
Error de disparo, 319
Error de ganancia, 303 - 304
Etapa analógica frontal, 278
Etiqueta, 104
Extrapolación de orden cero, . 322
Fases de una instrucción, 32
Fetch, 32
Filtro de Bessel, 299
Filtro de Butterworth, 299
Filtro de Chebishev, 299
Filtro de paso bajo,
 298 - 299, 323 - 324
Filtro, 298 - 300
Frecuencia central, 280 - 281
Frecuencia de corte,
 292 - 293, 295, 300, 323
Frecuencia de muestreo,
 282, 298, 300, 311 - 312
Fuente de alimentación,
 13, 38, 40 - 42, 290
Full duplex, ver comunicación asincrónica bidireccional simultánea.
Full scale error, ver error de ganancia.
Gain error, ver error de ganancia.
Half duplex, ver comunicación bidireccional no simultánea.
- Handshake*, 151
In-Circuit Serial Programming (ICSP),
 ver programación en circuito.
Include, 119 - 120
Insertion loss, ver pérdidas por inserción.
Integral nonlinearity, ver no linealidad integral.
Interferencias, 295
Interrupción, [E/S por], 154
Interrupciones enmascarables, . .
 218 - 219, 224
Interrupciones no enmascarables,
 218 - 219
Intervalo de cuantificación,
 278, 304 - 305
Label, ver etiqueta.
Limitador de corriente,
 293 - 294, 324 - 325
Limitador de tensión, 293 - 295
Linker, ver enlazador.
Longitud de palabra, 55
Macro, ver macroinstrucción.
Macroinstrucción,
 104, 127 - 130, 135 - 136
Margen dinámico, 278 - 279, 289, 292
Maskable interrupts, ver interrupciones enmascarables.
Master - slave, ver servidor - cliente.
Términos utilizados para denotar relaciones de subordinación entre dos dispositivos. En castellano se utilizan los términos amo - esclavo, maestro - seguidor, principal - subordinado, servidor - cliente.
Medidas absolutas, 305
Medidas por relación, 305

- Memoria de datos, .22 - 24, 29 - 30
Memoria de lectura y escritura, 15, 22, 59
Memoria de programa, . 22, 29 - 30
Memoria no volátil, . . 16, 36, 60, 75
Microcontrolador, 12
Modelo de programación, . 88 - 89
Módem, 251 - 253
Modem, ver módem.
Muestreo en tiempo real, 282
Muestreo repetitivo secuencial, 282
Multiplexor analógico, 283, 296 - 298, 305 - 306
No linealidad diferencial, 303 - 304
No linealidad integral, 304
Non-maskable interrupts, ver interrupciones no enmascarables.
Núcleo, 24 - 27, 72
Null modem, 252 - 253
Número de bits efectivo, 304 - 305
Octeto, 55
Offset, ver desplazamiento.
Operandos, 82 - 83, 105, 115
Organización LIFO de la pila, 85, 87, 230
Oscilador de cristal, 14, 34 - 35, 197
Oscilador externo, 34, 36
Oscilador RC, 34 - 36, 39, 307 - 308
Oscillator Start-up Timer (OST), ver temporizador de arranque del oscilador.
Páginas de la memoria, 45, 47, 56 - 58, 62, 64, 68, 84, 95 - 96
Palabra de configuración, ver bits de configuración.
Pérdidas por inserción, 296
Periférico, 149, 164
Periodo de muestreo, 311 - 312
Pipeline, ver segmentado.
Polling, ver E/S por consulta.
Port, ver puerto.
Post-divisor, 44, 189 - 190, 192, 199 - 201
Postscaler, ver post-divisor.
Power-on Reset (POR), ver reset por encendido.
Power-up Timer (PWRT), ver temporizador de arranque de encendido.
Pre-divisor, 74 - 75, 103, 189 - 202, 205 - 207, 209, 213 - 215, 235, 237, 241
Prescaler, ver pre-divisor.
Procesador digital de señales, 26 - 27
Profundidad de la pila, 86
Program Counter (PC), ver contador de programa.
Programa fuente, 80 - 81
Programa objeto, 80 - 82
Programación en circuito, . 30, 160
Protocolo de comunicación, . . 248
Puente de sensores, 318
Puerto, 149 - 150
Pull-up, 159 - 160, 166, 172 - 173, 254
Puntero de la pila, . . . 14 - 15, 30, 86
Rango dinámico, 278 - 279, 289, 292
Real Time Clock (RTC), ver reloj de tiempo real.
Rebote, 166 - 168, 170, 176

-
- Record*, 140
 Refrescamiento, 59, 177 - 179
 Registros de funciones especiales, 29, 68 - 70, 72, 107
 Relación de rechazo del modo común, 292
 Reloj de tiempo real, 197, 234 - 237, 239 - 241
 Reset por encendido, 18, 36, 39 - 42, 74
 Resolución de la señal PWM, 211, 213 - 214
 Resolución, 278 - 280
 Retención de orden cero, 322
 Ruido de cuantificación, 304
 Ruido, 279, 300, 317, 319
Sample and hold amplifier, ver amplificador de muestreo y retención.
Sampling converter, 301
Schmitt trigger, 156 - 158, 161 - 162, 316 - 317, 319
 Sección lógica, 143 - 144
 Segmentado, 29, 32 - 34
 Seguimiento y retención, 302
 Sensor casidigital, 278
 Sensor de capacidad variable, 318
 Sensor resistivo, 305, 317 - 318
 Sensor, 275 - 277
 Señal alterna, 281, 286 - 287, 298, 304
 Señal analógica, 12, 275 - 276, 295, 309 - 312, 316, 321
 Señal asimétrica, 289 - 291, 296
 Señal continua, 281, 285 - 287
 Señal de banda ancha, 280 - 281, 287, 292
 Señal de banda estrecha, 280 - 281, 292
 Señal diferencial, 289 - 291, 296
 Señal flotante, 289
 Señal PWM, 203, 210 - 213, 323 - 324
 Señal seudodiferencial, 289 - 291
 Señales falsas, 282, 298, 300
 Servidor - cliente, 253 - 254, 258, 262, 265 - 266, 270
 Símbolos, 56, 106 - 107, 111, 113, 124
Single-ended input, ver entrada asimétrica.
Single-ended signal, ver señal asimétrica.
 Sistema de adquisición de datos de alto nivel, 283 - 284
 Sistema de adquisición de datos de bajo nivel, 283 - 284
 Sistema de numeración hexadecimal, 56, 105 - 106, 116 - 118
Stack Pointer (SP), ver puntero de la pila.
Stand-alone devices, ver circuitos integrados independientes
Static crosstalk, ver diafonía estática.
 Subrutina que atiende a una interrupción, 220 - 221
 Tamaño de las instrucciones, 29, 45, 88
 Temporizador de arranque de encendido, 39 - 42
 Temporizador de arranque del oscilador, 39 - 40
 Tensión de offset, 291, 323
 Tensión de referencia, 159, 276 - 277, 305, 307, 316, 320

- Termistor PTC, 294
Throughput rate, ver velocidad de
comutación máxima.
Tiempo de adquisición,
 302, 309 - 314
Tiempo de apertura, 302
Tiempo de comutación, 297 - 298
Tiempo de conversión,
 302, 309 - 312, 314
Tiempo de espera, . . 167, 243 - 245
Tiempo de establecimiento,
 40 - 41, 298
Tiempo de latencia, 224 - 225
Time-out, ver tiempo de espera.
Tope de la pila, 64, 86
Totem-pole, 155 - 156
Track and hold, ver seguimiento y re-
tención.
Umbral de transición, 303
Unidad aritmética y lógica,
 12, 14 - 15, 30 - 31
Unidad central de procesamiento,
 11 - 12, 14 - 15, 20 - 25,
 30, 32, 43, 48, 153 - 154, 218 - 222
Varistor de óxido metálico, . . 294
Velocidad de comutación
 máxima, 298
Velocidad de descarga, . . 301 - 302
Velocidad de transmisión,
 247 - 248, 258, 260, 263 - 264
Zero-order hold, ver retención de or-
den cero.

MICROCONTROLADORES

FUNDAMENTOS Y APLICACIONES CON PIC

Los microcontroladores están presentes en muchos de los productos electrónicos que empleamos en nuestra vida cotidiana. Su enseñanza es un reto debido a la variedad de modelos existentes en el mercado y a la gran cantidad de aplicaciones posibles. Sin embargo, a pesar de su diversidad, hay unidad en los principios de funcionamiento y las arquitecturas de muchos microcontroladores. Este libro aprovecha esa unidad presente en la diversidad para mostrar los fundamentos del diseño y la programación de los microcontroladores.

El objetivo del libro es enseñar la arquitectura y la programación de los microcontroladores en general, tomando como ejemplos los microcontroladores PIC de Microchip. La documentación que ofrecen los fabricantes es tan abundante que su mero acopio ocuparía varios volúmenes. En este libro se han seleccionado los temas de forma fundamentada, buscando el rigor en las descripciones y la claridad en la exposición de los conceptos. Se han incluido figuras que complementan el texto de forma sustancial, evitando fotografías u otro material gráfico que aumenta el número de páginas pero aporta poca información útil.

Cada tema es tratado con un enfoque que va de lo general a lo particular. Primero se explican las cuestiones propias del tema que son comunes a la mayoría de los microcontroladores, y seguidamente se particulariza para los microcontroladores PIC. Las explicaciones se ilustran con ejemplos prácticos. En nueve capítulos se explican la estructura y componentes de los microcontroladores, y en particular: la memoria, la entrada y salida paralelas, el repertorio de instrucciones y la programación en lenguaje ensamblador, los temporizadores, las interrupciones, la entrada y salida serie y la adquisición y distribución de señales con las entradas y salidas analógicas.

El libro está dirigido especialmente a estudiantes y a profesionales de la electrónica, pero también resultará útil a los lectores interesados en conocer el fascinante mundo de los microcontroladores, en particular de los PIC, y utilizarlos en un sinfín de aplicaciones.

FERNANDO EUDALDO VALDÉS PÉREZ es Ingeniero Electricista (1977) y Máster en Automática (2001) por la Universidad de Oriente (UO) en Santiago de Cuba. Es Profesor Titular de Centro de Estudios de Neurociencias y Procesamiento de Imágenes y Señales (CENPIS) de la Facultad de Ingeniería Eléctrica en la UO. Con 29 años de experiencia académica, ha impartido asignaturas de pregrado y postgrado relacionadas con la arquitectura y la programación de microprocesadores, microcontroladores y las computadoras personales, así como los circuitos eléctricos, la electrónica digital y el procesamiento estadístico de señales, con énfasis en las aplicaciones biomédicas. Es el autor principal del libro de texto para la carrera de Automática "Fundamentos Técnicos de Computación", publicado por la Editorial ISPJAE, Ciudad de La Habana, 1986. Ha trabajado en el diseño de monitores de hemodiálisis y plasmáferesis. Su actividad de investigación actual se centra en la adquisición y procesamiento de señales del sistema cardiovascular.

RAMÓN PALLÁS -ARENY es profesor de la Universidad Politécnica de Cataluña (UPC) desde 1975 y catedrático de universidad desde 1989. Actualmente trabaja en la Escuela Politécnica Superior de Castelldefels, donde dirige el Grupo de Instrumentación, Sensores e Interfaces (<http://isi.upc.es>). En temas de instrumentación electrónica y médica, ha impartido cursos para formación de profesorado y postgraduados en España, Argentina, Cuba, Italia, México y Rumanía. Ha recibido, entre otros galardones, el Premio a la Calidad en la Docencia Universitaria, otorgado por el Consejo Social de la UPC (2000) y la Medalla Narcís Monturiol de la Generalitat de Cataluña (2003).

MICROCONTROLADORES

VALDÉS 2007 PIC1

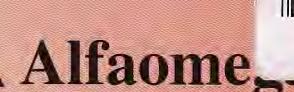


A014CO0013

ISBN 978-970-15-1149-7



9 789701 511497

 Alfaomega  Sistemas de Información