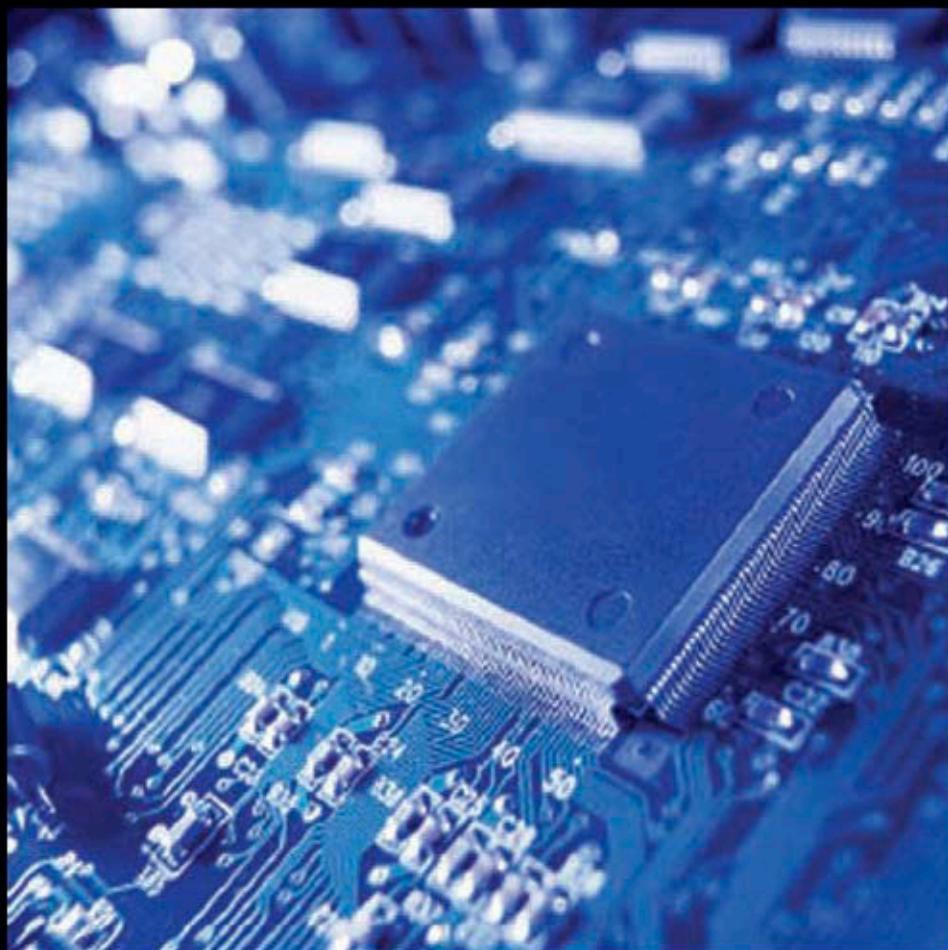


Cuarta edición

MICROCONTROLADOR 8051



I. SCOTT MacKENZIE
RAPHAEL C.-W. PHAN

MICROCONTROLADOR 8051

CUARTA EDICIÓN

I. Scott MacKenzie

York University

Raphael C.-W. Phan

*Swinburne University of Technology
(Sarawak Campus)*

TRADUCCIÓN

Ricardo Javier Romero Elizondo

Ingeniero en sistemas electrónicos

*Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Monterrey*

REVISIÓN TÉCNICA

Cuauhtémoc Carbajal Fernández

Profesor del Departamento de Ingeniería Electrica y Electrónica

Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

José Miguel Morán Loza

Presidente de la Academia de Sistemas Digitales Avanzados

Centro Universitario de Ciencias Exactas e Ingenierías

Universidad de Guadalajara



México • Argentina • Brasil • Colombia • Costa Rica • Chile • Ecuador
España • Guatemala • Panamá • Perú • Puerto Rico • Uruguay • Venezuela

www.FreeLibros.me

Datos de catalogación bibliográfica

MACKENZIE, I. SCOTT; C.-W. PHAN, RAPHAEL
Microcontrolador 8051. Cuarta edición

PEARSON EDUCACIÓN, México, 2007

ISBN: 978-970-26-1021-2

Área: Ingeniería

Formato: 19 × 23.5 cm

Páginas: 552

Authorized translation from the English language edition, entitled The 8051 microcontroller by I. Scott Mackenzie and Raphael C. -W. Phan published by Pearson Education, Inc., publishing as PRENTICE HALL, INC., Copyright © 2007. All rights reserved.

ISBN 0130195626

Traducción autorizada de la edición en idioma inglés, The 8051 microcontroller por I. Scott Mackenzie y Raphael C. -W. Phan publicada por Pearson Education, Inc., publicada como PRENTICE HALL INC., Copyright © 2007. Todos los derechos reservados.

Esta edición en español es la única autorizada.

Edición en español

Editor: Luis Miguel Cruz Castillo

e-mail: luis.cruz@pearsoned.com

Editor de desarrollo: Bernardino Gutiérrez Hernández

Supervisor de producción: Rodrigo Romero Villalobos

Edición en inglés

Editor-in-Chief: Vernon Anthony

Production Editor: Rex Davidson

Design Coordinator: Diane Ernsberger

Editorial Assistant: Lara Dimmick

Cover Designer: Candace Rowley

Cover art: Getty Images

Production Manager: Matt Ottenweller

Senior Marketing Manager: Ben Leonard

Marketing Assistant: Les Roberts

Senior Marketing Coordinator: Liz Farrell

CUARTA EDICIÓN, 2007

D.R. © 2007 por Pearson Educación de México, S.A. de C.V.

Atlacomulco No. 500, 5° piso

Col. Industrial Atoto

53519, Naucalpan de Juárez, Edo. de México

E-mail: editorial.universidades@pearsoned.com

Cámara Nacional de la Industria Editorial Mexicana. Reg. Núm. 1031.

Prentice Hall es una marca registrada de Pearson Educación de México, S.A. de C.V.

Reservados todos los derechos. Ni la totalidad ni parte de esta publicación pueden reproducirse, registrarse o transmitirse, por un sistema de recuperación de información, en ninguna forma ni por ningún medio, sea electrónico, mecánico, fotoquímico, magnético o electroóptico, por fotocopia, grabación o cualquier otro, sin permiso previo por escrito del editor.

El préstamo, alquiler o cualquier otra forma de cesión de uso de este ejemplar requerirá también la autorización del editor o de sus representantes.



ISBN10: 970-26-1021-4

ISBN 13: 978-970-26-1021-2

Impreso en México. Printed in Mexico.

1 2 3 4 5 6 7 8 9 0 - 10 09 08 07

PREFACIO

Este libro, en el que examinaremos las características de hardware y software de la familia de microcontroladores MCS-51, está enfocado hacia los estudiantes universitarios ubicados en los campos de electrónica y tecnología computacional, hacia los ingenieros en electricidad o computación, y hacia los técnicos o ingenieros interesados en aprender acerca de los microcontroladores.

Durante el desarrollo del texto se llevó a cabo el proceso de prueba y depuración de los medios necesarios para satisfacer la necesidad de información de la audiencia a la que va dirigido este libro, el cual, en su versión preliminar, fue base de un curso para estudiantes universitarios del quinto semestre de ingeniería computacional. Los estudiantes construyeron una computadora de un solo tablero basada en el microcontrolador 8051 como parte de dicho curso, tal y como se describe en el capítulo 11. A su vez, hemos utilizado esa computadora como sistema para implementar un “proyecto” final de sexto semestre en el cual los estudiantes puedan diseñar, implementar y documentar un “producto”, controlado por el microcontrolador 8051, que incorpore software y hardware originales.

En este libro se enfatiza la arquitectura y la programación más que los detalles eléctricos del microcontrolador 8051 debido a que, como todos los microcontroladores, éste contiene un alto grado de funcionalidad. Los temas referentes al software se presentan bajo el contexto del ensamblador de Intel (ASM51) y el enlazador/localizador (RL51).

Esta edición contiene cuatro nuevos capítulos, y su característica adicional más importante es la información que presenta acerca de cómo utilizar el lenguaje C para programar el 8051 como una alternativa al lenguaje ensamblador aplicado en ediciones anteriores. La programación en C permite el uso de programas estructurados y es muy útil para codificar proyectos grandes y complejos basados en el 8051.

Todos los ejemplos contienen anotaciones para ayudar tanto al estudiante como al instructor. Los ejemplos parten con el enunciado del problema, seguido de una solución simple y directa. Después de la solución se presenta un análisis en el que se exploran los detalles internos del problema y de la solución. La intención es explicar y crear, tomando en cuenta las diferentes perspectivas involucradas en los ejemplos.

Consideramos que los cursos sobre microprocesadores o microcontroladores son más difíciles de presentar que, por decir, los cursos de sistemas digitales, ya que no resulta fácil definir una secuencia lineal de los temas o conceptos a tratar. El primer programa que se muestra a los estudiantes trae considerables suposiciones, tales como el conocimiento previo del modelo de programación y modos de direccionamiento de la CPU, la distinción entre una dirección y el contenido de una dirección, etc. Por todo esto, es posible que un curso basado en este libro no siga de manera estricta la secuencia que aquí presentamos. Sin embargo, el capítulo 1 es un buen punto para comenzar, ya que sirve como introducción general a los microcontroladores, con particular énfasis en las diferencias que hay entre los microcontroladores y los microprocesadores.

En el capítulo 2 se presenta la arquitectura del hardware del microcontrolador 8051 y sus contrapartes que forman la familia MCS-51. También se presentan ejemplos concisos utilizando secuencias cortas de instrucciones. Los instructores deberán prepararse en este punto para introducir, en forma paralela, temas de los capítulos 3 y 7, y de los apéndices A y C, para reforzar el conocimiento del software requerido para estos ejemplos. El apéndice A es de un valor especial, ya que contiene en una sola figura todo el conjunto de instrucciones del 8051.

En el capítulo 3 se presenta el conjunto de instrucciones, empezando con las definiciones de los modos de direccionamiento del 8051. El conjunto de instrucciones contiene distintas categorías (transferencia de datos, bifurcación, etc.), las cuales facilitan su presentación paso por paso. Cada modo de direccionamiento y cada tipo de instrucción se demuestran mediante varios ejemplos breves.

Los capítulos 4, 5 y 6 tratan acerca de las características internas del 8051, comenzando con los temporizadores, después con el puerto serial (el cual requiere un temporizador que se utiliza como generador de velocidad en baudios), y por último con las interrupciones. Los ejemplos incluidos en estos capítulos son más largos y complejos que los presentados antes. Recomendamos a los instructores que no se abalancen sobre estos capítulos, pues resulta esencial que los estudiantes obtengan primero una sólida comprensión de la arquitectura y del conjunto de instrucciones del 8051 antes de ver estos temas.

Muchos de los temas que se presentan en el capítulo 7 se cubrirán por necesidad a lo largo de los primeros seis capítulos. Sin embargo, quizás este capítulo sea el de mayor importancia para cimentar en el estudiante la capacidad de llevar a cabo proyectos a gran escala. Temas avanzados tales como evaluación de expresiones en la fase de ensamblado, programación modular, enlazamiento y localización, y programación de macros representarán un reto considerable para muchos estudiantes. Es aquí donde tenemos que enfatizar la importancia de la experiencia práctica. Recomendamos que los estudiantes codifiquen los ejemplos incluidos en este capítulo y observen los mensajes de salida y de errores proporcionados por el ASM51, el RL51, y la herramienta para convertir objetos en código hexadecimal (OH).

En el capítulo 8 se establecen las bases para programar el 8051 mediante el lenguaje C. Se describen las diferencias entre este lenguaje de alto nivel y el lenguaje ensamblador, y las correspondientes entre el lenguaje C para sistemas computacionales y el lenguaje C para microcontroladores incrustados tales como el 8051.

En los capítulos 9 y 10 se presentan temas avanzados relacionados con métodos y estilos de programación y con el ambiente de desarrollo. Estos capítulos se dedican a tratar temas amplios y conceptuales que son importantes en los ambientes de desarrollo profesionales.

El capítulo 11 ofrece varios ejemplos de diseño, los cuales incorporan una selección de hardware con software de apoyo. El verdadero enfoque de estos ejemplos es el software, sobre el cual incluimos suficientes notas. Esta cuarta edición contiene varias interfaces adicionales: una pantalla de cristal líquido (LCD), el 8255, una interfaz serial RS-232, una interfaz paralela Centronics, sensores, relevadores, y un motor por pasos. La SBC-51, una computadora de un solo tablero basada en el 8051, es uno de los diseños que se presenta en el capítulo 11. La SBC-51 podría ser base para un curso sobre el microcontrolador 8051. El apéndice G incluye un programa corto de monitoreo, pero suficiente para comenzar a practicar. Un ambiente de desarrollo también requiere de una computadora anfitriona, la cual además funciona como una terminal para controlar la SBC-51 luego de descargar los programas que serán ejecutados.

Docenas de estudiantes de ingeniería computacional han construido versiones prototípico de la SBC durante el tiempo en que Scott les ha impartido cursos basados en el 8051. Por su parte, Raphael desea agradecer la colaboración de sus estudiantes de Fundamentos de los microprocesadores, Aplicaciones de los microprocesadores y Microprocesadores incrustados, quienes con gran entusiasmo han llevado a cabo tareas y proyectos basados en el 8051.

El nuevo capítulo 12 trata sobre los ejemplos de diseño e interfaces del capítulo 11, pero con las soluciones en lenguaje C en vez de usar lenguaje ensamblador.

El capítulo 13 presenta algunos ejemplos más avanzados de proyectos con el 8051; además se concentra en una exposición de las opciones de diseño y la importancia del seudocódigo durante el proceso de diseño, antes de escribir el código real.

El capítulo 14 trata brevemente algunos dispositivos derivados del 8051, los cuales son descendientes de este microcontrolador, pero que incluyen mejoras tales como un aumento en la velocidad y el tamaño de la memoria, periféricos adicionales integrados, capacidades de red mejoradas, y mecanismos de seguridad.

También es importante mencionar el tratamiento sobre tarjetas inteligentes y seguridad de los datos incluido en esta edición, especialmente en los capítulos 12, 13 y 14, así como en el apéndice J. Incluimos esta información debido a la creciente popularidad de las tarjetas inteligentes que utilizan microcontroladores de 8 bits, tales como el 8051, para ejecutar software de seguridad y proteger información confidencial.

En este libro se hace un uso extenso de la literatura de Intel sobre los dispositivos MCS-51, además de tomarla como base. En especial, el apéndice C contiene las definiciones de todas las instrucciones del 8051, mientras que el apéndice E contiene la hoja de datos del 8051. Agradecemos mucho la cooperación de Intel.

Todos los ejemplos presentados en lenguaje C para el 8051 se compilaron, depuraron y evaluaron con el IDE μ Vision2 de Keil, el cual se puede descargar del sitio web <http://www.keil.com>, administrado por los desarrolladores del software. También agradecemos a las siguientes personas por sus revisiones e invaluosables comentarios, críticas y sugerencias: Dwight Egbert, de University of Nevada; Marty Kaliski, de Cal Polytech State University; Claude Kansaku, del Oregon Institute of Technology; y Ron Tinkham, de Santa Fe Community College. Raphael agradece a su esposa Grace por su comprensión y paciencia, y por sacrificar muchas noches, fines de semana y días festivos para hacerle compañía mientras preparaba esta edición. De hecho, sin sus suaves y ligeros codazos no se hubiera podido terminar el libro. Esta edición está dedicada a ella.

I. Scott MacKenzie
Raphael C.-W. Phan

CONTENIDO

1 INTRODUCCIÓN A LOS MICROCONTROLADORES 1

1.1	Introducción	1
1.2	Terminología	3
1.3	La unidad central de procesamiento	4
1.4	Memorias de semiconductor: RAM y ROM	5
1.5	Los buses de direcciones, datos y control	6
1.6	Dispositivos de entrada/salida	
1.6.1	Dispositivos de almacenamiento masivo	1.6.2 Dispositivos de interfaz humana
1.6.3	Dispositivos de control/monitoreo	
1.7	Los programas: grandes y pequeños	8
1.8	Micros, minis y supercomputadoras	9
1.9	Comparación entre microprocesadores y microcontroladores	10
1.9.1	Arquitectura del hardware	1.9.2 Aplicaciones
		1.9.3 Características del conjunto de instrucciones
1.10	Nuevos conceptos	12
1.11	Ganancias y pérdidas: un ejemplo de diseño	13
	Problemas	14

2 RESUMEN DEL HARDWARE 17

2.1	Información general sobre la familia MCS-51™	17
2.2	Descripción de las terminales	18
2.2.1	Puerto 0	2.2.2 Puerto 1
	2.2.3 Puerto 2	2.2.4 Puerto 3
2.2.5	PSEN (Habilitación de programa almacenado)	
2.2.6	ALE (Habilitación de latch de dirección)	2.2.7 EA (Acceso externo)
2.2.8	RST (Reset, o reinicio)	2.2.9 Entradas del oscilador incorporado en el chip
2.2.10	Conexiones de energía	
2.3	Estructura de los puertos de E/S	22
2.4	Sincronización y el ciclo de máquina	23
2.5	Organización de la memoria	24

2.5.1	Memoria RAM de propósito general
2.5.2	RAM accesible por bits individuales 2.5.3 Bancos de registros
2.6	Registros con funciones especiales 28
2.6.1	Palabra de estado del programa 2.6.2 Registro B 2.6.3 Apuntador de pila
2.6.4	Apuntador de datos 2.6.5 Registros de puerto
2.6.6	Registros de temporizadores 2.6.7 Registros de puerto serial
2.6.8	Registros de interrupciones 2.6.9 Registro de control de energía
2.7	Memoria externa 36
2.7.1	Acceso a la memoria externa para código 2.7.2 Acceso a la memoria externa
2.7.3	Decodificación de direcciones 2.7.4 Traslape de los espacios externos para código y para datos
2.8	Mejoras del 8032/8052 41
2.9	Operación de reinicio 43
	Resumen 44
	Problemas 44

3 RESUMEN DEL CONJUNTO DE INSTRUCCIONES 49

3.1	Introducción 49
3.2	Modos de direccionamiento 50
3.2.1	Direccionamiento por registro 3.2.2 Direccionamiento directo
3.2.3	Direccionamiento indirecto 3.2.4 Direccionamiento inmediato
3.2.5	Direccionamiento relativo 3.2.6 Direccionamiento absoluto
3.2.7	Direccionamiento largo 3.2.8 Direccionamiento indexado
3.3	Tipos de instrucciones 59
3.3.1	Instrucciones aritméticas 3.3.2 Instrucciones lógicas 3.3.3 Instrucciones para transferencia de datos 3.3.4 Instrucciones booleanas 3.3.5 Instrucciones de bifurcación de programa
	Resumen 78
	Problemas 78

4 OPERACIÓN DE LOS TEMPORIZADORES 87

4.1	Introducción 87
4.2	Registro de modo del temporizador (TMOD) 89
4.3	Registro de control del temporizador (TCON) 89
4.4	Modos del temporizador y la bandera de desbordamiento 90
4.4.1	Modo de temporizador de 13 bits (Modo 0) 4.4.2 Modo de temporizador de 16 bits (Modo 1) 4.4.3 Modo de autorrecarga de 8 bits (Modo 2)
4.4.4	Modo de temporizador dividido (Modo 3)
4.5	Fuentes de reloj 92
4.5.1	Temporización de intervalos 4.5.2 Conteo de eventos

4.6	Inicio, detención y control de los temporizadores	93
4.7	Inicialización y acceso a los registros del temporizador	95
4.7.1	Lectura de un temporizador “al instante”	
4.8	Intervalos cortos, medianos y largos	96
4.9	Generación de frecuencias exactas	102
4.9.1	Eliminación de los errores de redondeo	4.9.2 Compensación de la pérdida de tiempo debido a las instrucciones
4.10	Temporizador 2 del 8052	105
4.10.1	Modo de autorrecarga	4.10.2 Modo de captura
4.11	Generación de tasa en baudios	106
Resumen		107
Problemas		107

5 OPERACIÓN DEL PUERTO SERIAL 111

5.1	Introducción	111
5.2	Comunicación serial	111
5.3	Registro de búfer del puerto serial (SBUF)	112
5.4	Registro de control del puerto serial (SCON)	113
5.5	Modos de operación	113
5.5.1	Registro de desplazamiento de 8 bits (Modo 0)	5.5.2 UART de 8 bits con velocidad en baudios variable (Modo 1)
		5.5.3 UART de 9 bits con velocidad en baudios fija (Modo 2)
		5.5.4 UART de 9 bits con velocidad en baudios variable (Modo 3)
5.6	Escollos acerca de la comunicación serial tipo full duplex	117
5.7	Inicialización y acceso a los registros del puerto serial	118
5.7.1	Habilitación del receptor	5.7.2 El noveno bit de datos
		5.7.3 Agregar un bit de paridad
		5.7.4 Banderas de interrupción
5.8	Comunicaciones entre múltiples procesadores	119
5.9	Velocidades en baudios del puerto serial	120
5.9.1	Uso del temporizador 1 como el reloj de velocidad en baudios	
Resumen		127
Problemas		128

6 LAS INTERRUPCIONES 131

6.1	Introducción	131
6.2	Organización de las interrupciones en el 8051	132
6.2.1	Habilitar y deshabilitar interrupciones	6.2.2 Prioridad de las interrupciones
6.2.3	Secuencia de sondeo	

6.3	Procesamiento de interrupciones	136
6.3.1	Vectores de interrupción	
6.4	Diseño de programas mediante el uso de interrupciones	137
6.4.1	Rutinas de servicio de interrupción pequeñas 6.4.2 Rutinas de servicio de interrupción grandes	
6.5	Interrupciones del temporizador	139
6.6	Interrupciones del puerto serial	142
6.7	Interrupciones externas	143
6.8	Sincronización de interrupciones	148
Resumen	149	
Problemas	150	

7 PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR 151

7.1	Introducción	151
7.2	Operación de un ensamblador	152
7.2.1	Paso uno 7.2.2 Paso dos	
7.3	Formato de un programa en lenguaje ensamblador	155
7.3.1	Campo de etiqueta 7.3.2 Campo de mnemónico 7.3.3 Campo de operando	
7.3.4	Campo de comentario 7.3.5 Símbolos especiales del ensamblador	
7.3.6	Dirección indirecta 7.3.7 Datos inmediatos 7.3.8 Dirección de datos	
7.3.9	Dirección de bit 7.3.10 Dirección de código 7.3.11 Saltos genéricos y llamadas	
7.4	Evaluación de expresión al tiempo de ensamblado	160
7.4.1	Bases de números 7.4.2 Cadenas de caracteres 7.4.3 Operadores aritméticos	
7.4.4	Operadores lógicos 7.4.5 Operadores especiales 7.4.6 Operadores relacionales 7.4.7 Ejemplos de expresiones 7.4.8 Precedencia de los operadores	
7.5	Directivas del ensamblador	164
7.5.1	Control del estado del ensamblador 7.5.2 Definición de símbolos	
7.5.3	Inicialización/reservación de almacenamiento 7.5.4 Enlazamiento de programas 7.5.5 Directivas de selección de segmentos	
7.6	Controles del ensamblador	173
7.7	Operación del enlazador	173
7.8	Ejemplo con notas: enlace de segmentos y modulos relocalizables	176
7.8.1	PANTALLA.LST 7.8.2 ES.LST 7.8.3 EJEMPLO.M51	
7.9	Macros	183
7.9.1	Transferencia de parámetros 7.9.2 Etiquetas locales	
7.9.3	Operaciones de repetición 7.9.4 Operaciones de flujo de control	
Resumen	188	
Problemas	188	

8 PROGRAMACIÓN EN C DEL 8051 191

8.1	Introducción	191
8.2	Ventajas y desventajas del lenguaje C del 8051	191
8.3	Compiladores para C del 8051	192
8.4	Tipos de datos	193
8.5	Tipos y modelos de memoria	197
8.6	Arreglos	198
8.7	Estructuras	199
8.8	Apuntadores	199
8.8.1	El tipo de memoria de un apuntador	8.8.2 Apuntadores con tipo
8.8.3	Apuntadores sin tipo	
8.9	Funciones	202
8.9.1	Transferencia de parámetros	8.9.2 Valores de retorno
8.10	Algunos ejemplos en lenguaje C del 8051	204
8.10.1	El primer programa	8.10.2 Temporizadores
8.10.3	El puerto serial	8.10.4 Interrupciones
	Resumen	214
	Problemas	214

9 Estructura y diseño de programas 217

9.1	Introducción	217
9.2	Ventajas y desventajas de la programación estructurada	219
9.3	Las tres estructuras	220
9.3.1	Instrucciones	9.3.2 La estructura de ciclo
9.3.3	La estructura opción	
9.4	Sintaxis del seudocódigo	234
9.5	Estilo de programación en lenguaje ensamblador	237
9.5.1	Etiquetas	9.5.2 Comentarios
9.5.3	Bloques de comentarios	9.5.4 Almacenamiento de registros en la pila
9.5.5	El uso de igualdades	9.5.6 El uso de subrutinas
9.5.7	Organización de programas	
9.6	Estilo de programación en lenguaje C del 8051	243
9.6.1	Comentarios	9.6.2 El uso de definiciones
9.6.3	El uso de funciones	9.6.4 El uso de arreglos y apuntadores
9.6.5	Organización de los programas	
	Resumen	245
	Problemas	245

10 HERRAMIENTAS Y TÉCNICAS PARA EL DESARROLLO DE PROGRAMAS 247

10.1	Introducción	247
10.2	El ciclo de desarrollo	247
	10.2.1 Desarrollo de software 10.2.2 Desarrollo de hardware	
10.3	Integración y verificación	251
	10.3.1 Simulación en software 10.3.2 Emulación de hardware	
	10.3.3 Ejecución desde la RAM 10.3.4 Ejecución desde una EPROM	
	10.3.5 El proceso de máscara de fábrica	
10.4	Comandos y ambientes	255
	Resumen	257
	Problemas	257

11 EJEMPLOS DE DISEÑO E INTERFACES 259

11.1	Introducción	259
11.2	La SBC-51	259
11.3	Interfaz a un teclado numérico hexadecimal	265
11.4	Interfaz con varios LEDs de 7 segmentos	267
11.5	Interfaz con pantallas de cristal líquido (LCDs)	273
11.6	Interfaz con un altavoz	276
11.7	Interfaz con RAM no volátil	277
11.8	Expansión de entradas/salidas	282
	11.8.1 Uso de registros de desplazamiento 11.8.2 Uso del 8255	
11.9	Interfaz serial RS232 (EIA-232)	291
11.10	Interfaz paralela centronics	294
11.11	Salida analógica	296
11.12	Entrada analógica	300
11.13	Interfaz a sensores	303
11.14	Interfaz a relevadores	306
11.15	Interfaz a motor paso a paso	310
	Resumen	315
	Problemas	315

12 EJEMPLOS DE DISEÑO E INTERFACES EN C 319

12.1	Introducción	319
12.2	Interfaz de teclado numérico hexadecimal	319
12.3	Interfaz para varios LEDs de 7 segmentos	323

12.4	Interfaz para las pantallas de cristal líquido (LCD)	325
12.5	Interfaz para un altavoz	327
12.6	Interfaz para una RAM no volátil	329
12.7	Ampliación de entradas/salidas	333
12.8	Interfaz serial RS232 (EIA-232)	337
12.9	Interfaz paralela centronics	339
12.10	Salidas analógicas	341
12.11	Entradas analógicas	342
12.12	Interfaz para sensores	344
12.13	Interfaz para relevadores	346
12.14	Interfaz para un motor paso a paso	347
	Problemas	350

13 PROYECTOS DE EJEMPLO PARA LOS ESTUDIANTES 353

13.1	Introducción	353
13.2	Sistema de seguridad residencial	353
	13.2.1 Descripción del proyecto	
	13.2.2 Especificaciones del sistema	
	13.2.3 Diseño del sistema	
	13.2.4 Diseño del software	
13.3	Sistema de elevador	355
	13.3.1 Descripción del proyecto	
	13.3.2 Especificaciones del sistema	
	13.3.3 Diseño del sistema	
	13.3.4 Diseño del software	
13.4	Tres en raya	358
	13.4.1 Descripción del proyecto	
	13.4.2 Especificaciones del sistema	
	13.4.3 Diseño del software	
13.5	Calculadora	363
	13.5.1 Descripción del proyecto	
	13.5.2 Especificaciones del sistema	
	13.5.3 Diseño del software	
13.6	Microrratón	366
	13.6.1 Descripción del proyecto	
	13.6.2 Especificaciones del sistema	
	13.6.3 Diseño del sistema	
	13.6.4 Diseño del software	
13.7	Un robot que juega fútbol soccer	369
	13.7.1 Descripción del proyecto	
	13.7.2 Especificaciones del sistema	
	13.7.3 Diseño del sistema	
	13.7.4 Diseño del software	
13.8	Una aplicación de tarjeta inteligente	371
	13.8.1 Conceptos básicos de seguridad	
	13.8.2 Descripción del proyecto	
	13.8.3 Especificaciones del sistema	
	13.8.4 Diseño del software	
	Resumen	373
	Problemas	374

14 Derivados del 8051 377

14.1	Introducción	377
14.2	MCS-151™ y MCS-251™	377
14.3	Microcontroladores con memoria flash y NVRAM	377
14.4	Microcontroladores con ADCs y DACs	378
14.5	Microcontroladores de alta velocidad	378
14.6	Microcontroladores para redes	379
14.7	Microcontroladores seguros	379
	Resumen	379
	Problemas	380

APÉNDICES

A	Diagrama de referencia rápida	381
B	Mapa de códigos de operación	383
C	Definiciones de las instrucciones	385
D	Registros con funciones especiales	431
E	Hoja de datos del 8051	439
F	Diagrama de códigos en ASCII	455
G	MON51—un programa de monitoreo para el 8051	457
H	Una guía al IDE μ Vision2 de Keil	499
I	Una guía para el simulador del 8052	507
J	El Advanced Encryption Standard	515
K	Fuentes de productos de desarrollo para el 8051	521

BIBLIOGRAFÍA 527

ÍNDICE 529

Introducción a los microcontroladores

1.1 INTRODUCCIÓN

Aunque las computadoras han estado con nosotros tan sólo unas cuantas décadas, su impacto ha sido profundo, rivalizando en importancia con el teléfono, el automóvil o la televisión. Todos sentimos su presencia, no importa si programamos computadoras o si únicamente recibimos estados de cuenta mensuales impresos por un inmenso sistema de cómputo y que se entregan por correo convencional. Por esto tendemos a clasificar a las computadoras como “procesadores de datos”, cuya función es efectuar operaciones numéricas con una competencia sin limitaciones.

Encontramos computadoras de índole bastante diferente en un contexto más sutil, donde ejecutan tareas de modo tranquilo, eficiente e, incluso, humilde, y a menudo no se advierte su presencia. Podemos encontrar a las computadoras como un componente central de muchos productos industriales, automotrices y de consumo, tales como cajas registradoras y balanzas en un supermercado; hornos de microondas, lavadoras, relojes despertadores y termostatos en el hogar; juguetes, reproductoras de video, aparatos estereofónicos e instrumentos musicales; máquinas de escribir y fotocopiadoras en la oficina; tableros de instrumentos y sistemas de ignición en los automóviles; taladros de banco y fotocomponedoras en plantas industriales. En estos ambientes las computadoras ejecutan funciones de “control” a través de una interfaz con el “mundo real” para encender o apagar dispositivos y monitorear determinadas condiciones. Los **microcontroladores** (a diferencia de las microcomputadoras o los microprocesadores) se utilizan con frecuencia en aplicaciones tales como las antes mencionadas.

Es difícil imaginar el mundo actual de los juguetes electrónicos sin los microprocesadores. Pero esta maravilla integrada en un solo chip está apenas llegando a su aniversario número 35. En 1971 Intel Corporation presentó el 8080, el primer microprocesador exitoso. Poco después, Motorola, RCA, MOS Technology y Zilog introdujeron dispositivos similares: los 6800, 1801, 6502 y Z80, respectivamente. Estos circuitos integrados (CIs) por sí solos no eran muy útiles; pero como parte de una computadora de una sola tarjeta (SBC) se convirtieron en el componente central de útiles productos empleados para diseñar y aprender más acerca de los microprocesadores. Estas SBC, entre las cuales está la D2 de Motorola, la KIM-I de MOS Technology, y la SDK-85 de Intel, pronto encontraron uso en los laboratorios de diseño en colegios, universidades y compañías de componentes electrónicos.

El microcontrolador es un dispositivo similar al microprocesador. En 1976 Intel introdujo el 8748, el primer dispositivo en la familia de microcontroladores MCS-48™. El 8748 incluía una CPU, una EPROM de 1K bytes, 64 bytes de RAM, 27 terminales de entrada/salida, y un temporizador de 8 bits, todo esto contenido en un solo circuito integrado con 17,000 transistores. Este circuito integrado y los subsiguientes dispositivos MCS-48™ pronto se convirtieron en un estándar industrial en aplicaciones orientadas al control. Una de las aplicaciones populares al principio (y lo sigue siendo) fue el reemplazo de componentes electromecánicos en productos tales como lavadoras y controladores de señales de tránsito (semáforos). Otros productos donde podemos encontrar microcontroladores incluyen automóviles, equipos industriales, productos de consumo para el entretenimiento y dispositivos periféricos para computadoras. (Los propietarios de una computadora personal IBM encontrarán un ejemplo de un diseño basado en microcontrolador, con un mínimo de componentes adicionales, con tan sólo examinar el interior del teclado).

La potencia, el tamaño, y la complejidad de los microcontroladores avanzaron de manera drástica en 1980 cuando Intel anunció el 8051, el primer dispositivo en la familia de microcontroladores MCS-51™. En comparación con el 8048, este dispositivo cuenta con más de 60,000 transistores, 4K bytes de ROM, 128 bytes de RAM, 32 líneas de entrada/salida, un puerto serial, y dos temporizadores de 16 bits; una cantidad formidable de circuitos presentes en un solo circuito integrado (vea la figura 1-1). Se han agregado nuevos miembros a la familia MCS-51™, y en la actualidad existen versiones que casi doblan la capacidad de estas especificaciones. Siemens Corporation, la segunda proveedora de componentes para la familia MCS-51™, ofrece el SAB80515, una versión mejorada del 8051 con 68 terminales que representan seis puertos de entrada/salida de 8 bits, 13 fuentes de interrupción, y un convertidor analógico-digital de 8 bits con ocho canales de entrada. En el capítulo 14 abordaremos también otras variantes mejoradas del 8051. La familia del 8051 está muy bien establecida como base de uno de los microcontroladores de 8 bits más versátiles y poderosos; su posición como líder de los microcontroladores está asegurada por muchos años.

Este libro trata acerca de la familia de microcontroladores MCS-51™, y en los siguientes capítulos presentaremos la arquitectura de su hardware así como su programación y demostraremos,

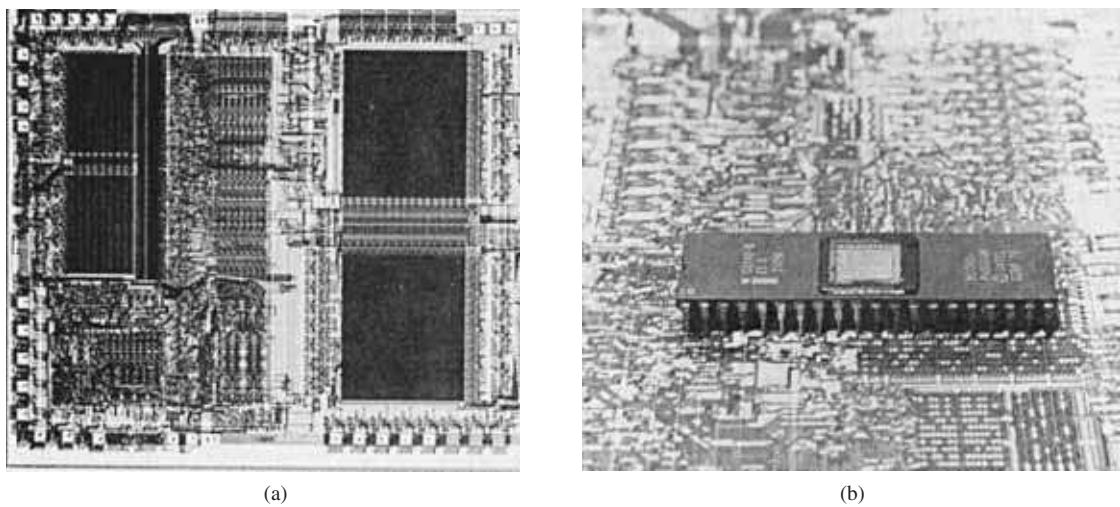


FIGURA 1-1

El microcontrolador 8051. (a) Una pastilla 8051, (b) un 8751 con EPROM integrada al chip (cortesía de Intel Corporation)

mediante una gran cantidad de ejemplos de diseño, cómo pueden participar todos estos dispositivos en diseños electrónicos con un mínimo de componentes adicionales.

En las siguientes secciones desarrollaremos, mediante una breve introducción a la arquitectura computacional, un vocabulario funcional de los muchos acrónimos y términos populares que prevalecen (y muchas veces confunden) en este campo. Nuestro enfoque es más práctico que académico debido a que muchos de los términos presentan definiciones vagas y que se trasladan, además de estar sujetas a los prejuicios de las grandes corporaciones y a los caprichos de diversos autores. Presentaremos cada término en su entorno más común con una explicación simple y directa.

1.2 TERMINOLOGÍA

Para comenzar, una **computadora** se define mediante dos características clave: (1) la capacidad de ser programada para efectuar operaciones sobre ciertos datos sin ninguna intervención humana, y (2) la capacidad de almacenar y recuperar datos. Un **sistema computacional** incluye también **dispositivos periféricos** para permitir la comunicación con las personas, además de **programas** que procesan datos. El equipo se define como **hardware** y los programas como **software**. Examinemos la figura 1-2 para empezar a conocer el hardware computacional.

La falta de detalles en la figura es intencional para poder representar los diferentes tamaños de las computadoras. Como se muestra, un sistema computacional contiene una **unidad central de procesamiento** (CPU) que se conecta a la **memoria de acceso aleatorio** (RAM) y a la **memoria de sólo lectura** (ROM) a través de un **bus de direcciones**, un **bus de datos**, y un **bus de control**. Los **circuitos de interfaz** conectan los buses del sistema con los **dispositivos periféricos**. Veamos con detalle cada uno de estos componentes.

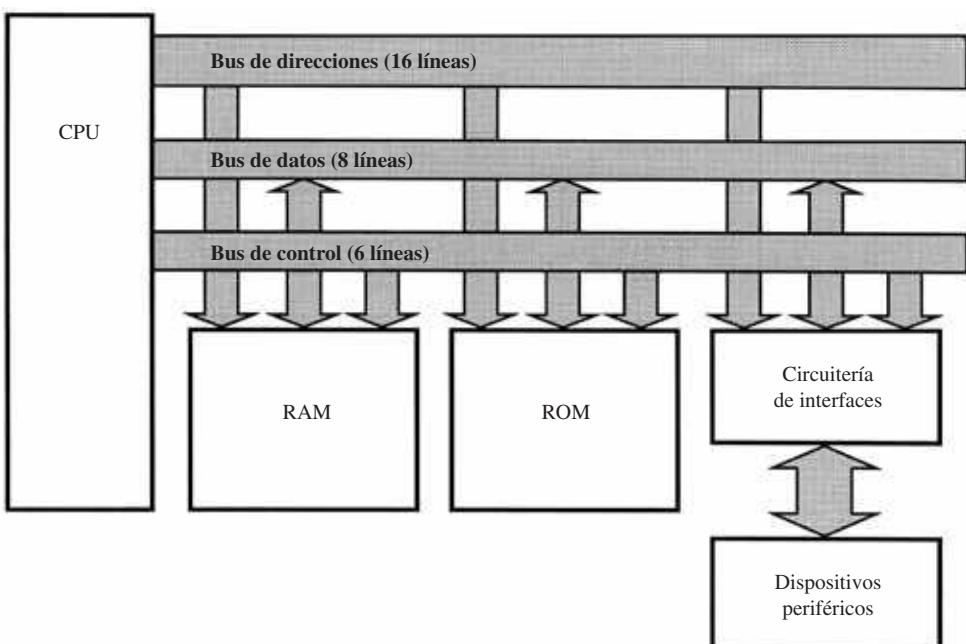


FIGURA 1-2

Diagrama de bloques de un sistema de microcomputadora

1.3 LA UNIDAD CENTRAL DE PROCESAMIENTO

La CPU actúa como el “cerebro” de un sistema computacional; administra toda actividad realizada en el sistema y lleva a cabo todas las operaciones sobre los datos. La mayor parte del misterio que rodea a una CPU no está justificada, ya que ésta constituye sólo una colección de circuitos lógicos que ejecutan dos operaciones de manera continua: la búsqueda (fetch) de instrucciones y su ejecución. La CPU tiene capacidad para interpretar y ejecutar instrucciones codificadas en forma binaria, cada una de las cuales representa una operación simple. Por lo general, estas instrucciones son aritméticas (suma, resta, multiplicación, división), lógicas (AND, OR, NOT, etc.), de movimiento de datos, o de operaciones de bifurcación, y se representan mediante una serie de códigos binarios llamada **conjunto de instrucciones**.

La figura 1-3 es una representación bastante simplificada del interior de una CPU. Esta figura muestra un conjunto de **registros** para el almacenamiento temporal de información; una **unidad aritmética-lógica (ALU)** para llevar a cabo operaciones sobre esta información; una **unidad de decodificación y control de instrucciones** que determina la operación a ejecutar e inicia las acciones necesarias para realizarla, y dos registros adicionales. El **registro de instrucciones (IR)** almacena el código binario para cada instrucción a medida que se ejecuta, y el **contador de programa (PC)** almacena la dirección de memoria de la siguiente instrucción a ejecutar.

La búsqueda de una instrucción en la ROM del sistema es una de las operaciones fundamentales realizadas por la CPU. Involucra los siguientes pasos: (a) el contenido del contador de programa se transfiere al bus de direcciones, (b) se activa una señal de control de LECTURA, (c) los datos (el código de operación de la instrucción) se leen de la ROM y se transfieren al bus de datos, (d) el código de operación se almacena en el registro interno de instrucciones de la CPU, y (e) el contador de programa se incrementa como preparación para la siguiente búsqueda en la memoria. La figura 1-4 ilustra el flujo de información que se presenta durante la búsqueda de una instrucción.

La etapa de ejecución involucra la decodificación (o interpretación) del código de operación y la generación de señales de control para permitir la entrada y salida de datos de los registros

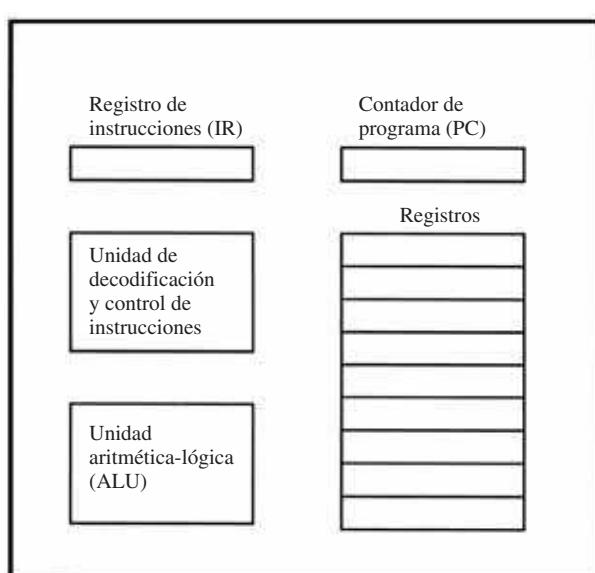
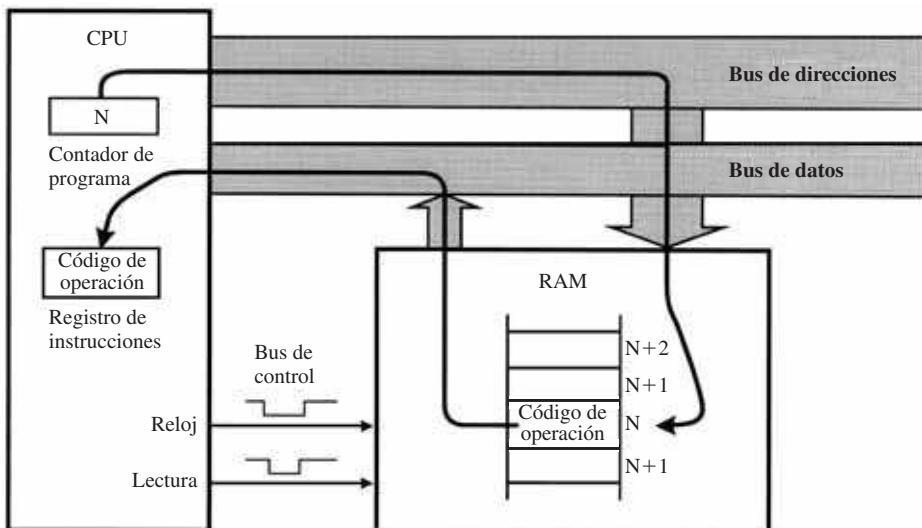


FIGURA 1–3
La unidad central de procesamiento (CPU)

**FIGURA 1–4**

Actividad en el bus durante un ciclo de búsqueda de un código de operación

internos de la ALU y enviar una señal a la ALU de que ejecute la operación especificada. Esta explicación es un tanto limitada en su alcance debido a la gran variedad de posibles operaciones. Es aplicable a una operación simple, tal como “incrementar el registro”. Instrucciones más complejas requieren de pasos adicionales, como la lectura de un segundo y tercer bytes que también forman parte de los datos necesarios para ejecutar la operación.

Un **programa**, o **software**, es una serie de instrucciones que se combinan para ejecutar una tarea útil; es aquí donde reside el verdadero misterio. La calidad del software (y no el grado de sofisticación de la CPU) es lo que determina el nivel de eficiencia y exactitud con que se ejecutan las tareas. Entonces, los programas “controlan” la CPU, y algunas veces fallan al hacerlo, imitando las debilidades de sus autores. Frases como “la computadora cometió un error” son un tanto inexactas. Aunque es inevitable la falla del equipo, los resultados erróneos por lo general son indicios de programas mediocres o de errores del operador.

1.4 MEMORIAS DE SEMICONDUCTOR: RAM Y ROM

Los programas y los datos se almacenan en la memoria. La variedad de memorias para computadoras es muy vasta, los términos que las acompañan son tan abundantes y los avances en la tecnología tan frecuentes que se requiere un estudio extensivo y continuo para mantenerse al tanto de los últimos avances. Los dispositivos de memoria a los que la CPU puede acceder en forma directa están constituidos por CI's (circuitos integrados), a los cuales se conoce como RAM y ROM. Existen dos características que distinguen una RAM de una ROM: en primer lugar, la RAM es una memoria de lectura/escritura, mientras que la ROM es memoria de sólo lectura; en segundo lugar, la RAM es volátil (el contenido se pierde cuando se interrumpe la energía), mientras que la ROM es no volátil.

La mayoría de los sistemas computacionales posee una unidad de disco y una cantidad pequeña de ROM, la suficiente como para almacenar las cortas rutinas de software que se utilizan con más frecuencia y llevan a cabo operaciones de entrada/salida. Los programas y datos del

usuario se almacenan en disco y se cargan en la RAM para su ejecución. Gracias a la continua reducción en el costo por byte de la memoria RAM, hasta los sistemas computacionales pequeños contienen a menudo millones de bytes de RAM.

1.5 LOS BUSES DE DIRECCIONES, DATOS Y CONTROL

Un **bus** es una colección de alambres que transfieren información con un propósito común. El acceso a los circuitos que rodean la CPU se lleva a cabo mediante el uso de tres buses: el **bus de direcciones**, el **bus de datos**, y el **bus de control**. Para cada operación de lectura o escritura, la CPU especifica la ubicación de los datos (o instrucción); para ello coloca una dirección en el bus de direcciones y después activa una señal en el bus de control, con lo cual indica si la operación es de lectura o de escritura. En las operaciones de lectura se recupera un byte de datos de la ubicación de memoria especificada y se coloca en el bus de datos. La CPU lee los datos y los coloca en uno de sus registros internos. Para realizar una operación de escritura, la CPU envía los datos mediante el bus de datos. Gracias a la señal de control, la memoria reconoce la operación como un ciclo de escritura y almacena los datos en la ubicación especificada.

La mayoría de las computadoras pequeñas cuenta con 16 o 20 líneas de dirección. Dado un número n de líneas de dirección, con la posibilidad para cada línea de estar en un nivel alto (1) o bajo (0), se puede acceder a 2^n ubicaciones. Por lo tanto, un bus de direcciones de 16 bits puede acceder a $2^{16} = 65,536$ ubicaciones; una dirección de 20 bits tiene acceso a $2^{20} = 1'048,576$ ubicaciones. La abreviatura K (kilo) representa el valor $2^{10} = 1024$; en consecuencia, 16 bits pueden acceder a $2^6 \times 2^{10} = 64K$ ubicaciones, mientras que 20 bits acceden a 1024K o 1M ubicaciones. La abreviatura M (mega) representa el valor de $2^{20} = 1024 \times 1024 = 1,048,576$.

El bus de datos transfiere información entre la CPU y la memoria, o entre la CPU y los dispositivos de entrada/salida. Se ha realizado una extensa investigación para determinar el tipo de actividades que consumen el valioso tiempo de ejecución de una computadora. Es evidente que las computadoras invierten hasta dos tercios de su tiempo tan sólo en efectuar las operaciones necesarias para mover datos. La cantidad de líneas (anchura) del bus de datos es importante para el rendimiento en general, ya que la mayoría de las operaciones necesarias para mover datos tiene lugar entre un registro de la CPU y la memoria externa, RAM, o la ROM. Esta limitación debida a la anchura resulta en un embudo: aunque exista una gran cantidad de memoria en el sistema y la CPU posea un poder computacional enorme, el acceso a los datos (el movimiento de datos entre la memoria y la CPU a través del bus de datos) puede encontrarse con un embudo, dependiendo de la anchura del bus de datos.

Esta característica es tan importante que resulta muy común añadir un prefijo en el que se indica el grado de dicha aglomeración. La frase “computadora de 16 bits” se refiere a una computadora que tiene 16 líneas en su bus de datos. La mayoría de las computadoras cae dentro de las clasificaciones de 4, 8, 16 o 32 bits, donde el poder de cómputo en general aumenta a medida que se incrementa la anchura del bus de datos.

Observe que el bus de datos mostrado en la figura 1-2 es bidireccional, y el bus de direcciones es unidireccional. La CPU siempre proporciona información acerca de las direcciones (como indica la flecha en la figura 1-2), y sin embargo los datos pueden viajar en cualquier dirección dependiendo de si la intención es una operación de lectura o de escritura.¹ Observe también que el término “datos” se utiliza en sentido general: la “información” que viaja en el bus de datos pueden ser las instrucciones de un programa, una dirección añadida a una instrucción, o los datos utilizados por el programa.

¹En ocasiones también los circuitos de acceso directo a memoria (DMA) proporcionan información acerca de las direcciones (además de la CPU).

El bus de control es una mezcolanza de señales, cada una de las cuales tiene una tarea específica en el ordenado control de actividades del sistema. En general, las señales de control son señales de sincronización que la CPU proporciona para coordinar el movimiento de la información en los buses de direcciones y de datos. Aunque por lo común existen tres señales, tales como las de CLOCK (reloj), READ (lectura) y WRITE (escritura), para realizar el movimiento básico de datos entre la CPU y la memoria, los nombres y la operación de estas señales dependen en gran parte de la CPU que se utilice. Para obtener más detalles, consulte las hojas de datos del fabricante.

1.6 DISPOSITIVOS DE ENTRADA/SALIDA

Los dispositivos de entrada/salida, o “periféricos de computadora”, proveen una ruta para establecer la comunicación entre el sistema computacional y el “mundo real”. Sin estos dispositivos, los sistemas computacionales serían máquinas bastante introvertidas y de poca utilidad para quienes las utilizaran. Tres clases de dispositivos de entrada/salida son: **almacenamiento masivo, interfaz humana, y control/monitoreo**.

1.6.1 Dispositivos de almacenamiento masivo

Al igual que las memorias de semiconductor RAM y ROM, los dispositivos de almacenamiento masivo forman parte del campo de la tecnología de memoria, la cual está en crecimiento y mejora constantes. Tal como implica su nombre, estos dispositivos almacenan grandes cantidades de información (programas o datos) que no caben en la relativamente pequeña memoria RAM o memoria “principal” de la computadora. Esta información debe cargarse en la memoria principal para que la CPU pueda acceder a ella. Los dispositivos de almacenamiento masivo se clasifican de acuerdo con su facilidad de acceso; pueden ser **en línea o de archivo**. El almacenamiento en línea, provisto casi siempre en disco magnético, está disponible para la CPU cuando un programa efectúa una petición sin necesidad de que haya intervención humana. El almacenamiento de archivo contiene datos que rara vez se necesitan, por lo cual es necesario cargarlos manualmente en el sistema. Por lo general, el almacenamiento de archivo utiliza cintas o discos magnéticos, aunque discos ópticos tales como el CD-ROM o la tecnología WORM están emergiendo y pueden llegar a alterar el concepto del almacenamiento de archivo gracias a su confiabilidad, alta capacidad y bajo costo.²

1.6.2 Dispositivos de interfaz humana

La unión del humano y la máquina se implementa mediante una multitud de dispositivos de interfaz humana, de los cuales los más comunes son el teclado, la terminal de video (VDT) y la impresora. Las impresoras son dispositivos de salida que generan copias impresas. Las VDT también son dispositivos de salida y están constituidas básicamente por un tubo de rayos catódicos (CRT). La necesidad de diseñar estos dispositivos periféricos teniendo a los humanos en mente ha propiciado la evolución de un nuevo campo de ingeniería llamado “ergonomía” o “factores humanos”, el cual tiene la meta de acoplar de manera segura, cómoda y eficiente las características de las personas con las máquinas que utilizan. Desde luego, existen más compañías que producen esta clase de dispositivos periféricos que fabricantes de computadoras. Podemos encontrar por lo menos tres clases de estos dispositivos en la mayoría de los sistemas computacionales: un teclado, un monitor (CRT), y una impresora. Otros ejemplos de dispositivos de interfaz humana son la palanca de mando (o de juegos), la pluma óptica, el ratón, el micrófono y altavoces.

²El término “CD-ROM” significa memoria de sólo lectura de disco compacto. El término “WORM” significa escritura solamente una vez y lectura múltiple. Un CD-ROM puede contener 700 Mbytes, lo cual es suficiente como para almacenar los 32 volúmenes de la *Enciclopedia Británica*.

1.6.3 Dispositivos de control/monitoreo

Las computadoras pueden ejecutar un sinnúmero de tareas orientadas al control mediante el uso de dispositivos de control/monitoreo (y de dispositivos electrónicos de interfaz y software de meticuloso diseño), todo esto sin fatigarse y sobrepasando las capacidades humanas. Aplicaciones tales como el control de la temperatura en un edificio, la seguridad en el hogar, el control de elevadores, el control de aparatos electrodomésticos, e incluso la soldadura de las partes de un automóvil son posibles gracias a estos dispositivos.

Los dispositivos de control son salidas, o **actuadores**, que afectan lo que las rodea cuando se les suministra un voltaje o corriente (por ejemplo, motores y elevadores). Los dispositivos de monitoreo son entradas, o **sensores**, estimuladas mediante calor, luz, presión, movimiento, etc., y convierten esta energía en voltaje o corriente que una computadora puede leer (por ejemplo, fototransistores, termistores e interruptores). Los circuitos de interfaz convierten voltaje o corriente en datos binarios, o viceversa, y establecen una relación ordenada entre las entradas y salidas mediante el uso de software. El hardware y el software que permiten la interfaz de estos dispositivos con los microcontroladores constituyen uno de los temas principales de este libro.

1.7 LOS PROGRAMAS: GRANDES Y PEQUEÑOS

La disertación anterior se enfocó en el hardware de los sistemas computacionales con sólo una breve mención de los programas, o software, que los hacen funcionar. El relativo énfasis puesto en el hardware, en vez de en el software, sorprendentemente ha cambiado su curso en los años recientes. Mientras que los primeros tiempos de la computación fueron testigos de que el costo de los materiales, la fabricación, y el mantenimiento del hardware computacional sobrepasaban el costo del software, en la actualidad el costo del hardware ha disminuido gracias a los circuitos de integración a gran escala (LSI) que se producen masivamente. El trabajo intensivo de codificar, documentar, mantener, actualizar y distribuir el software constituye la mayor parte del costo de automatizar un proceso mediante el uso de computadoras.

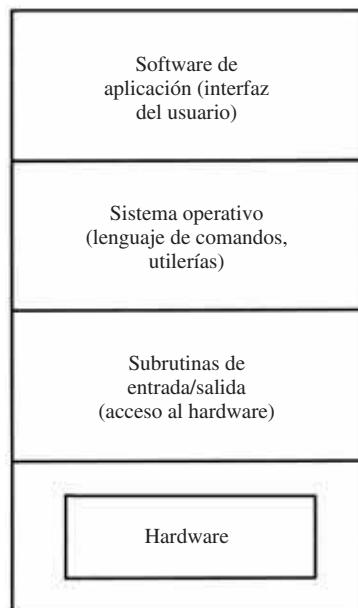
Vamos a examinar ahora los diferentes tipos de software. La figura 1-5 muestra tres niveles de software entre el usuario y el hardware de un sistema computacional: el **software de aplicación**, el **sistema operativo**, y las **subrutinas de entrada/salida**.

En el nivel más bajo, las subrutinas de entrada/salida manipulan directamente el hardware del sistema, leen los caracteres del teclado, escriben caracteres en el CRT, leen bloques de información del disco, y así por el estilo. Estas subrutinas son codificadas por los diseñadores del hardware, y (casi siempre) se almacenan en memoria ROM debido a que están íntimamente entrelazadas con el hardware. Un ejemplo de estas subrutinas es el BIOS (sistema básico de entrada/salida) incluido en la PC de IBM.

Se definen ciertas condiciones explícitas para las subrutinas de entrada/salida, de manera que los programadores tengan acceso directo al hardware del sistema. Lo único que debe hacerse es inicializar los valores en los registros de la CPU y llamar a la subrutina; la acción se lleva a cabo y los resultados regresan a los registros de la CPU o se colocan en la RAM del sistema.

La memoria ROM contiene un conjunto completo de subrutinas de entrada/salida y un programa de inicio que ejecuta cuando el sistema se enciende o el operador lo reinicializa manualmente. La naturaleza no-volátil de la memoria ROM es esencial pues dicho programa debe existir al momento del encendido. El programa de inicio ejecuta todas las tareas cotidianas, tales como validación de opciones, inicialización de la memoria, diagnósticos, etc. Por último, la rutina del **cargador de arranque** (bootstrap loader) lee la primera pista (un programa pequeño) del disco y la carga en la memoria RAM; después le entrega el control al programa. Éste, a su vez, carga del disco la porción residente en RAM del sistema operativo (un programa grande),

FIGURA 1–5
Niveles de software



al cual le transmite entonces el control, y de esta forma se completa el ciclo de inicialización del sistema.

El sistema operativo es una inmensa colección de programas que se incluyen con el sistema computacional y provee el mecanismo necesario para tener acceso a los recursos de la computadora, así como para manejarlos y utilizarlos de manera efectiva. Estas capacidades existen gracias al **lenguaje de comandos** y a **programas de utilería**, los cuales facilitan el desarrollo del software de aplicación. El usuario puede interactuar con la computadora aun cuando no tenga conocimientos sobre el sistema operativo, siempre que el software de aplicación esté bien diseñado. El objetivo principal del diseño del software de aplicación es proveer al usuario una interfaz efectiva, significativa y segura.

1.8 MICROS, MINIS Y SUPERCOMPUTADORAS

Si utilizamos el tamaño y el poder de cómputo como base, podemos clasificar a las computadoras en microcomputadoras, minicomputadoras, o supercomputadoras. Una característica primordial de las microcomputadoras es el tamaño y el encapsulado de la CPU: está contenida en un solo circuito integrado —un **microprocesador**—. Por otra parte, las minicomputadoras y las supercomputadoras, aparte de ser más complejas en cada detalle de su arquitectura, contienen CPUs constituidas por múltiples circuitos integrados, y abarcan desde varios CIs (minicomputadoras) hasta varias tarjetas de CIs (supercomputadoras). Esta mayor capacidad es necesaria para alcanzar las altas velocidades y el poder computacional que tienen las computadoras más grandes.

Las microcomputadoras típicas como la IBM PC, la Apple *Macintosh*, y la Commodore *Amiga* incorporan un microprocesador como su CPU. La RAM, la ROM, y los circuitos de interfaz requieren de muchos circuitos integrados, lo cual causa que el número de componentes aumente de acuerdo con el poder computacional. Los circuitos de interfaz varían de manera considerable en complejidad, dependiendo de los dispositivos de entrada/salida. Por ejemplo, para operar la bocina que contienen la mayoría de las computadoras se requiere sólo de un par de

compuertas lógicas. En comparación, la interfaz de disco casi siempre involucra muchos circuitos integrados, algunos de ellos en encapsulados LSI.

Otra característica que separa a las microcomputadoras de minis y supercomputadoras es que las microcomputadoras son sistemas para un solo usuario y una sola tarea; en otras palabras, interactúan solamente con un usuario y ejecutan sólo un programa a la vez. Por otro lado, minis y supercomputadoras son sistemas multisuarios y multitareas; pueden atender muchos usuarios y programas al mismo tiempo. En realidad, la ejecución simultánea de programas es una ilusión que resulta de la “compartición de tiempo” de los recursos de la CPU. (Sin embargo, los sistemas de multiprocesamiento utilizan varias CPU para ejecutar tareas en forma simultánea).

1.9 COMPARACIÓN ENTRE MICROPROCESADORES Y MICROCONTROLADORES

En secciones anteriores mencionamos que los microprocesadores son CPUs de un solo chip que se utilizan en las microcomputadoras. ¿Cómo podemos diferenciar los microcontroladores de los microprocesadores? Podemos contestar la pregunta desde tres puntos de vista: **arquitectura del hardware, aplicaciones, y características del conjunto de instrucciones**.

1.9.1 Arquitectura del hardware

Para destacar la diferencia entre los microcontroladores y los microprocesadores, repetimos la figura 1-2 en la figura 1-6, la cual se muestra con mayor detalle.

Mientras que un microprocesador es una CPU de un solo chip, un microcontrolador contiene, en un solo CI, una CPU y la mayor parte del resto de los circuitos de un sistema de microcomputadora completo. Los componentes ubicados dentro de la línea punteada en la figura 1-6 son parte integral de la mayoría de los CIs de los microcontroladores. Además de la CPU, los microcontroladores incluyen la RAM, la ROM, una interfaz serial, una interfaz paralela, un temporizador, y circuitos para la programación de interrupciones; todo dentro del mismo CI. Desde luego, la cantidad de RAM incorporada en el chip no se acerca siquiera a la de un modesto sistema de microcomputadora; pero como pronto aprenderemos, no representa una limitación ya que los microcontroladores se dedican a aplicaciones totalmente distintas.

Una característica importante de los microcontroladores es el sistema de interrupciones que incorporan. Los microcontroladores a menudo deben responder a ciertos estímulos externos (interrupciones) en tiempo real debido a que son dispositivos orientados al control. Deben llevar a cabo un cambio rápido de contexto mediante la suspensión de un proceso, mientras se ejecuta otro proceso en respuesta a algún “evento”. Ejemplo de un evento que puede causar interrupciones en un producto basado en un microcontrolador es cuando abrimos la puerta de un horno de microondas. Aunque la mayoría de los microprocesadores también pueden implementar poderosos esquemas de interrupción, a menudo se requieren componentes externos. Los circuitos incorporados en un microcontrolador incluyen todo lo necesario para el manejo de las interrupciones.

1.9.2 Aplicaciones

Los microprocesadores se utilizan con frecuencia como la CPU en un sistema de microcomputadora. Es para lo que están diseñados y donde residen sus fortalezas. Por el contrario, los microcontroladores se encuentran en diseños pequeños con un mínimo de componentes desarrollados para realizar actividades orientadas al control. Estos diseños se implementaban antes utilizando frecuentemente docenas o incluso cientos de CIs digitales. Un microcontrolador puede ayudar a reducir el total de componentes utilizados. Todo lo que se requiere es un microcontrolador, un pequeño número de componentes de soporte, y un programa de control en ROM. Los microcontroladores

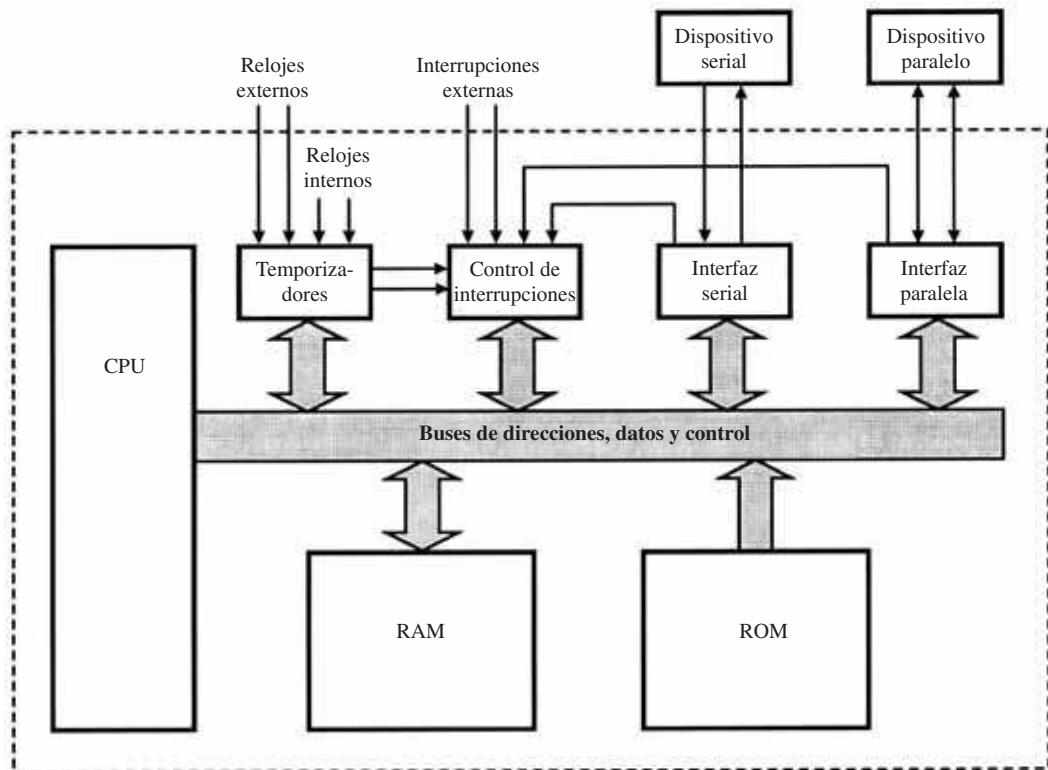
**FIGURA 1–6**

Diagrama de bloques detallado de un sistema microcomputacional

están adaptados para “controlar” dispositivos de entrada/salida en diseños que requieren la mínima cantidad de componentes, mientras que los microprocesadores son adaptados para “procesar” información en sistemas computacionales.

1.9.3 Características del conjunto de instrucciones

Los microcontroladores requieren de un conjunto de instrucciones un tanto distinto al de los microprocesadores debido a las deficiencias en las aplicaciones. Los conjuntos de instrucciones de los microprocesadores son de “procesamiento intensivo”, lo cual indica que tienen poderosos modos de direccionamiento con instrucciones dedicadas a realizar operaciones sobre grandes volúmenes de datos. Sus instrucciones operan en nibbles, bytes, palabras, o incluso en dobles palabras.³ Los modos de direccionamiento proporcionan acceso a grandes arreglos de datos, en donde se utilizan apuntadores de dirección y desplazamientos. Los modos automáticos de incremento y decremento simplifican el avance en arreglos de bytes, palabras, o palabras dobles. Las instrucciones privilegiadas no pueden ejecutarse dentro de un programa de usuario. Y la lista continúa.

³La interpretación más común de estos términos es 4 bits = 1 nibble, 8 bits = 1 byte, 16 bits = 1 palabra, y 32 bits = 1 palabra doble.

Por otra parte, los microcontroladores poseen conjuntos de instrucciones dedicados al control de entradas y salidas. La interfaz para muchas de las entradas y salidas utiliza un solo bit. Por ejemplo, un solenoide energizado por un puerto de salida de 1 bit puede encender y apagar un motor. Los microcontroladores cuentan con instrucciones para cambiar bits individuales a 1 o a 0 y llevar a cabo operaciones orientadas a bits, tales como las operaciones lógicas AND, OR o EXOR, para saltar si un bit es 1 o 0, y así sucesivamente. Esta poderosa característica se presenta raras veces en los microprocesadores, ya que están diseñados para operar sobre bytes o unidades de datos más grandes.

En el control y monitoreo de dispositivos (tal vez con una interfaz de 1 bit), los microcontroladores poseen circuitos incorporados e instrucciones para realizar operaciones de entrada/salida, de sincronización de eventos, y habilitar y establecer niveles de prioridad para las interrupciones causadas por estímulos externos. Los microprocesadores a veces requieren de circuitos adicionales (circuitos integrados de interfaz en serie, controladores de interrupciones, temporizadores, etc.) para llevar a cabo operaciones similares. Pero la capacidad absoluta de procesamiento de un microcontrolador nunca se acerca a la de un microprocesador (si todo lo demás es igual), debido a que dentro de un chip la mayor parte del espacio está ocupado por las funciones incorporadas, con lo cual se sacrifica el poder de procesamiento.

Las instrucciones de un microcontrolador deben ser bastante compactas, y la mayoría debe implementarse en un solo byte, debido a que el espacio disponible en el chip tiene gran demanda. Por lo general, un criterio de diseño es que el programa de control debe caber en la ROM incorporada en el chip, ya que agregar incluso una sola memoria ROM externa eleva demasiado el costo del producto final. Un esquema de codificación compacto es imprescindible para implementar el conjunto de instrucciones. Ésta es una característica rara en los microprocesadores; sus poderosos modos de direccionamiento traen consigo una codificación de instrucciones nada compacta.

1.10 NUEVOS CONCEPTOS

Los microcontroladores existen, como otros productos que en retrospectiva se consideran avances tecnológicos, gracias a dos fuerzas complementarias: la demanda del mercado y nueva tecnología. La nueva tecnología es justo lo que mencionamos antes: dispositivos de semiconductor que incluyen más transistores en un menor espacio y son producidos en masa a bajo costo. La demanda del mercado es el apetito que muestran la industria y los consumidores por herramientas y juguetes más sofisticados.⁴ Esta demanda abarca muchos espacios. El ejemplo más ilustrativo es quizás el panel de instrumentos de un automóvil. Examinemos cómo se ha transformado el “centro de control” del automóvil en la última década, lo cual ha sido posible gracias al microcontrolador y a otros avances tecnológicos. Hasta hace poco los conductores se contentaban con poder saber la velocidad a que conducían; hoy pueden observar qué tanto combustible gastan y el tiempo estimado en que llegarán a su destino. Alguna vez fue suficiente con saber si un cinturón de seguridad estaba desabrochado al momento de encender el automóvil; hoy nos “dicen” cuál es el culpable. Si una puerta está entreabierta, se nos informa de ello mediante una voz electrónica. (Tal vez el cinturón está atorado en la puerta).

Esto nos trae a la mente un comentario indispensable. Los microprocesadores (y en el mismo sentido los microcontroladores) han sido denominados como “soluciones en busca de un problema”. Parece que han demostrado ser tan efectivos al reducir la complejidad de los circuitos

⁴Podemos discrepar acerca de que la “demanda del mercado” es en realidad el “deseo del mercado”, al cual estimula el crecimiento autopropulsado de la tecnología.

en productos (del consumidor), que los fabricantes a menudo están ansiosos por incluir características superfljas sólo porque resultan fáciles de incorporar al diseño del producto. Esto ocasiona resultados faltos de eficacia, ya que al principio parecen una maravilla pero terminan siendo una molestia. Uno de los ejemplos más evidentes de este enfoque puede encontrarse en los productos que “hablan”. No importa si son automóviles, juguetes o tostadores, siguen siendo ejemplos de un diseño malo e inútil. Podemos estar seguros de que al desaparecer la novedad quedará únicamente lo sutil y apropiado del producto.

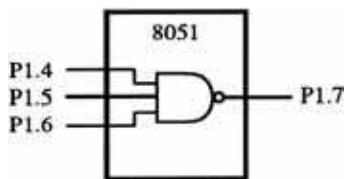
Los microcontroladores son especializados. Se utilizan en productos industriales y para el consumidor, no sólo en computadoras. Los usuarios de dichos productos no están enterados de la existencia de los microcontroladores: para ellos, los componentes internos son un detalle de diseño sin consecuencias. Consideremos como ejemplo los hornos de microondas, los termostatos programables, las balanzas electrónicas, e incluso los automóviles. Por lo general, los componentes electrónicos incluidos dentro de cada uno de estos productos incorporan un microcontrolador como interfaz para presionar botones, interruptores, luces y alarmas en el panel de control; la operación del usuario se asemeja a la de algunos antecedentes electromecánicos, a excepción de ciertas características adicionales. El microcontrolador es invisible para el usuario.

A diferencia de los sistemas computacionales, definidos por su capacidad de programación y reprogramación, los microcontroladores se programan para ejecutar una sola tarea de manera permanente. Esta comparación resulta a partir de las evidentes diferencias detectadas en la arquitectura de ambos sistemas. Los sistemas computacionales cuentan con una proporción alta de RAM a ROM; los programas de usuario se ejecutan en el gran espacio de la memoria RAM y las rutinas de interfaz con el hardware tienen lugar en un pequeño espacio de ROM. Por otro lado, los microcontroladores tienen una proporción alta de ROM a RAM. El programa de control, aun cuando sea un tanto grande, se almacena en la memoria ROM, mientras que la RAM solamente es utilizada para almacenamiento temporal. El programa de control se ha denominado **firmware** debido a que está almacenado de manera permanente (firme) en memoria ROM. En grados de “firmeza”, el firmware se encuentra entre el software (programas en memoria RAM que se pierden cuando se suspende la energía) y el hardware (circuitos físicos). La diferencia entre software y hardware es parcialmente análoga a la diferencia que hay entre una hoja de papel (hardware) y las palabras escritas en ella (software). Podemos representar al firmware como una carta con formato estándar, la cual está diseñada e impresa para cumplir un propósito específico.

1.11 GANANCIAS Y PÉRDIDAS: UN EJEMPLO DE DISEÑO

Las tareas ejecutadas por los microcontroladores no son nuevas. Lo que sí es nuevo son los diseños que pueden implementarse en la actualidad con menos componentes de lo que antes era posible. Los diseños que requerían decenas o incluso cientos de circuitos integrados ahora se implementan con sólo un pequeño número de componentes, incluyendo un microcontrolador. La reducida cantidad de componentes es resultado directo de la capacidad de programación y del alto grado de integración del microcontrolador, lo cual se traduce en menor tiempo de desarrollo, costos de manufactura más bajos, menos consumo de energía, y alta confiabilidad. Con frecuencia, las operaciones lógicas que requieren varios circuitos integrados se pueden implementar dentro del mismo microcontrolador al agregar un programa de control.

La velocidad es un problema. Las soluciones basadas en microcontroladores nunca son tan veloces como las implementaciones discretas. Los microcontroladores no resultan eficientes en el manejo de situaciones que requieren una respuesta en extremo rápida a ciertos eventos (un número pequeño de aplicaciones). Consideremos como ejemplo la implementación, en apariencia trivial, de una operación NAND en la que se utiliza un microcontrolador 8051, ilustrada en la figura 1-7.

**FIGURA 1-7**

Implementación de una operación lógica simple mediante un microcontrolador

No resulta muy evidente que podamos utilizar un microcontrolador para implementar tal operación, pero sí se puede. El software debe llevar a cabo las operaciones que se muestran en el diagrama de flujo de la figura 1-8. A continuación presentamos el programa en lenguaje ensamblador del 8051 para efectuar esta operación lógica.

CICLO:	MOV C, P1.4	;LEE BIT P1.4 A BANDERA DE CARRY
	ANL C, P1.5	;AND CON P1.5
	ANL C, P1.6	;AND CON P1.6
	CPL C	;COMPLEMENTA EL RESULTADO
	MOV P1.7, C	;ENVÍA AL BIT DE SALIDA P1.7
	SJMP CICLO	;SE REPITE

Si ejecutamos este programa en un microcontrolador 8051, sin duda se realizará la función NAND de 3 entradas. (Podemos verificar el resultado mediante un voltímetro o un osciloscopio). El tiempo de propagación de la transición de una entrada al nivel correcto de salida es demasiado largo, por lo menos en comparación con un circuito TTL (lógica de transistor-transistor) equivalente. El retraso es de 3 a 17 microsegundos, dependiendo del instante en que cambie la entrada en relación con el momento en que el programa detecta el cambio. (Asumimos la operación estándar del 8051, en la que se utiliza un cristal de 12 MHz). El tiempo de propagación equivalente en TTL es de unos 10 nanosegundos; casi tres órdenes menos de magnitud. Es claro que no existe competencia alguna cuando comparamos la velocidad de los microcontroladores con las implementaciones en TTL de la misma función.

En muchas ocasiones, en particular cuando se requiere la operación humana, no importa si los retrasos se miden en nanosegundos, microsegundos o milisegundos. (Cuando en nuestro automóvil disminuye la presión del aceite, ¿acaso necesitamos saberlo durante los primeros microsegundos?) El ejemplo de la compuerta lógica ilustra que los microcontroladores pueden implementar operaciones lógicas. Más aún, las ventajas de los diseños basados en microcontroladores comienzan a relucir conforme los diseños se vuelven más complicados. Como ya mencionamos, reducir la cantidad de componentes tiene sus ventajas; pero las operaciones en el programa de control también hacen posible la introducción de cambios en el diseño con sólo modificar el software. Una modificación de este tipo produce un impacto mínimo en el ciclo productivo.

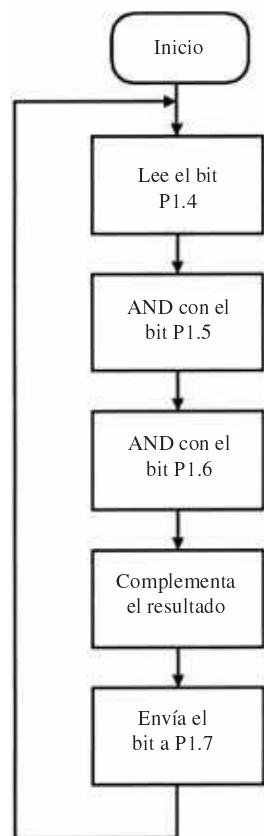
Con esto concluimos nuestra introducción a los microcontroladores. En el siguiente capítulo empezaremos a examinar la familia de dispositivos MCS-51™.

PROBLEMAS

- 1.1** ¿Cuál fue el primer microprocesador ampliamente utilizado? ¿En qué año y qué compañía lo introdujo?
- 1.2** En la década de 1970 había dos pequeñas compañías de microprocesadores llamadas MOS Technology y Zilog. Mencione el microprocesador que introdujo cada una de estas empresas.

FIGURA 1–8

Diagrama de flujo de un programa para una compuerta lógica



- 1.3** ¿En qué año se introdujo el microcontrolador 8051? ¿Cuál fue el predecesor del 8051 y en qué año se introdujo?
- 1.4** Mencione los dos tipos de memoria de semiconductor analizados en este capítulo. ¿Cuál es el tipo que retiene su contenido cuando se desconecta de la energía? ¿Cuál es el término más común que describe esta propiedad?
- 1.5** ¿Cuál es el registro de la CPU que siempre contiene una dirección? ¿Qué dirección está contenida en este registro?
- 1.6** ¿Qué información está contenida en los buses de direcciones y datos durante la búsqueda de un código de operación? En estos buses, ¿cuál es el sentido del flujo de la información mientras se busca un código de operación?
- 1.7** ¿A cuántos bytes de datos puede acceder un sistema computacional con un bus de direcciones de 18 bits y un bus de datos de 8 bits?
- 1.8** ¿Cuál es el significado más común del término “16 bits” en la frase “una computadora de 16 bits”?
- 1.9** ¿Cuál es la diferencia entre almacenamiento en línea y almacenamiento de archivo?
- 1.10** Además de cintas y discos magnéticos, ¿qué otro tipo de tecnología se utiliza en el almacenamiento de archivo?
- 1.11** En relación con los sistemas computacionales, ¿cuál es la meta del campo de ingeniería conocido como “factores humanos”?

- 1.12** Considere los siguientes dispositivos de interfaz humana: una palanca de mando (o de juegos), una pluma óptica, un ratón, un micrófono, y un altavoz. ¿Cuáles son dispositivos de entrada? ¿Cuáles son dispositivos de salida?
- 1.13** ¿Cuál es el más bajo de los tres niveles de software que presentamos en este capítulo? ¿Cuál es el propósito de tal nivel de software?
- 1.14** ¿Cuál es la diferencia entre un actuador y un sensor? Proporcione un ejemplo de cada uno.
- 1.15** ¿Qué es el firmware? Si comparamos un sistema basado en microcontroladores con un sistema basado en microprocesadores, ¿cuál será el que tenga más probabilidad de depender del firmware? ¿Por qué?
- 1.16** ¿Cuál es una característica importante del conjunto de instrucciones de un microcontrolador que lo distingue de un microprocesador?
- 1.17** Nombre cinco productos no mencionados en este capítulo y que muy probablemente utilizan un microcontrolador.

Resumen del hardware

2.1 INFORMACIÓN GENERAL SOBRE LA FAMILIA MCS-51™

La compañía Intel Corporation desarrolla, fabrica y comercializa la familia de microcontroladores MCS-51™. Otros fabricantes de circuitos integrados, tales como Siemens, Advanced Micro Devices, Fujitsu y Philips, tienen licencia como “proveedores secundarios” de dispositivos de la familia MCS-51™. Cada microcontrolador que integra esta familia posee características que satisfacen algún propósito específico de diseño.

En este capítulo presentaremos la arquitectura del hardware de la familia MCS-51™. El apéndice E contiene la hoja de datos de Intel para los dispositivos de nivel básico (por ejemplo, el 8051 AH). Recomendamos consultar este apéndice para obtener más detalles sobre las propiedades eléctricas de estos dispositivos.

Muchas de las características del hardware se ilustran mediante secuencias cortas de instrucciones. Cada ejemplo está acompañado por una breve descripción, pero diferimos todos los detalles acerca del conjunto de instrucciones hasta el capítulo 3. Consulte el apéndice A para obtener un resumen del conjunto de instrucciones del 8051, y el apéndice C para encontrar definiciones de cada una de las instrucciones del 8051.

El circuito integrado más genérico de la familia MCS-51™ es el 8051, y fue el primer dispositivo de la familia que se ofreció comercialmente. A continuación resumimos sus características.

- 4K bytes de memoria ROM
- 128 bytes de memoria RAM
- Cuatro puertos de E/S (Entrada/Salida) de 8 bits
- Dos temporizadores de 16 bits
- Una interfaz serial
- 64K de espacio para código en memoria externa
- 64K de espacio para datos en memoria externa
- Procesador booleano (opera sobre bits individuales)
- 210 ubicaciones direccionables para bits
- Multiplicación y división en $4\ \mu s$

TABLA 2-1

Comparación de los circuitos integrados de la familia MCS-51™

Número de pieza	Memoria para código incorporada en el chip	Memoria para datos incorporada en el chip	Temporizadores
8051	4K ROM	128 bytes	2
8031	0K	128 bytes	2
8751	4K EPROM	128 bytes	2
8052	8K ROM	256 bytes	3
8032	0K	256 bytes	3
8752	8K EPROM	256 bytes	3

Los demás miembros de la familia MCS-51™ ofrecen diferentes combinaciones de memoria ROM o EPROM incorporada en el chip, memoria RAM incorporada en el chip, o un tercer temporizador. Cada circuito integrado de la familia MCS-51™ también cuenta con una versión CMOS de bajo consumo de energía (consulte la tabla 2-1).

Utilizamos el término “8051” para referirnos a toda la familia de microcontroladores MCS-51™. Cuando el análisis se concentre en una versión mejorada de la versión básica del dispositivo 8051, utilizaremos el número de pieza específico. En el diagrama de bloques de la figura 2-1 se muestran las características antes mencionadas. (También puede consultarse el apéndice D).

2.2 DESCRIPCIÓN DE LAS TERMINALES

En esta sección se introduce la arquitectura de hardware del 8051, desde una perspectiva externa, mediante la asignación de las terminales (consulte la figura 2-2). A continuación exponemos una breve descripción de la función de cada terminal.

Como podemos observar en la figura 2-2, 32 de las 40 terminales del 8051 funcionan como líneas para puertos de E/S. 24 de estas líneas son de propósito dual (26 en el 8032/8052). Cada una de estas líneas puede operar como E/S o como línea de control, o como parte del bus de direcciones o del bus de datos.

Los diseños que requieren de muy poca memoria externa o de otros componentes externos utilizan estos puertos como E/S de propósito general. Las ocho líneas ubicadas en cada puerto se pueden tratar como una sola unidad de interfaz para dispositivos paralelos tales como impresoras, convertidores digitales-analógicos, etc. Cada línea también puede operar de manera independiente como interfaz para dispositivos de un solo bit, como interruptores, LEDs (diodos de emisión de luz), transistores, solenoides, motores y altavoces.

2.2.1 Puerto 0

El puerto 0 es un puerto de propósito dual en las terminales 32-39 del circuito integrado 8051. Se utiliza como un puerto de E/S de propósito general en diseños que requieren un mínimo de componentes. Este puerto se puede convertir en un bus de direcciones y datos multiplexados en diseños más complejos que requieran de memoria externa.

2.2.2 Puerto 1

El puerto 1 es un puerto dedicado de E/S en las terminales 1-8. Las terminales, designadas como P1.0, P1.1, P1.2, etc., están disponibles para utilizarse como interfaces para dispositivos externos, en caso de requerirse. Ninguna de las terminales del puerto 1 tiene otra función asignada, por lo

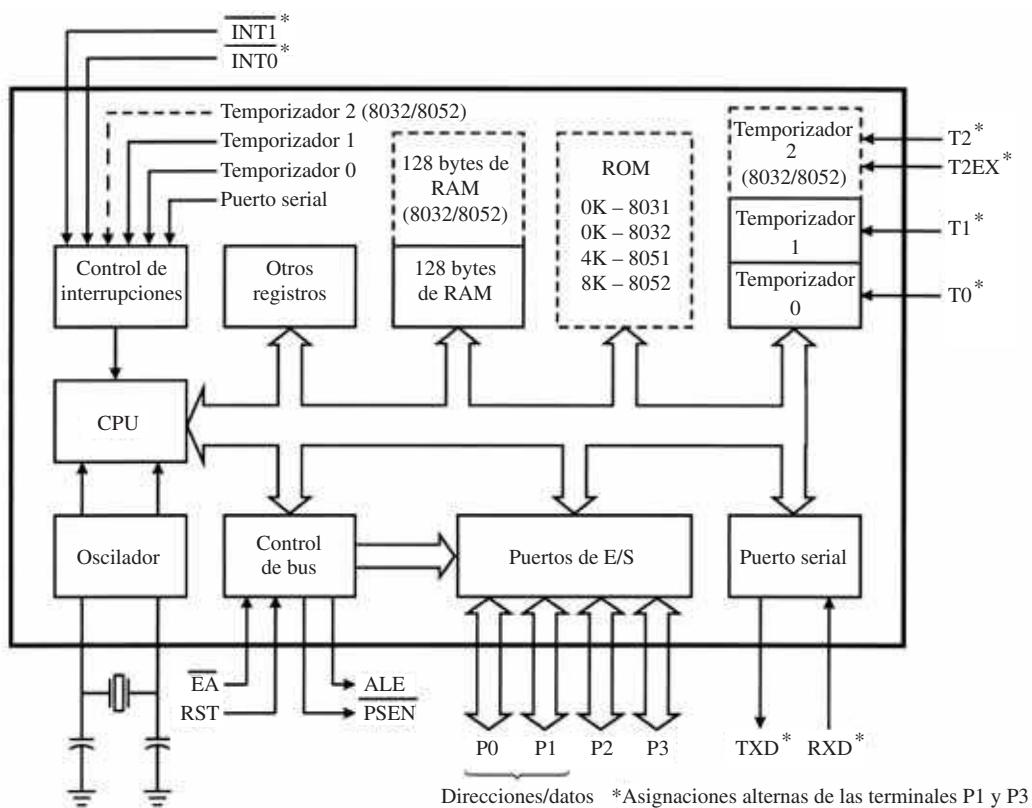


FIGURA 2–1
Diagrama de bloques del 8051

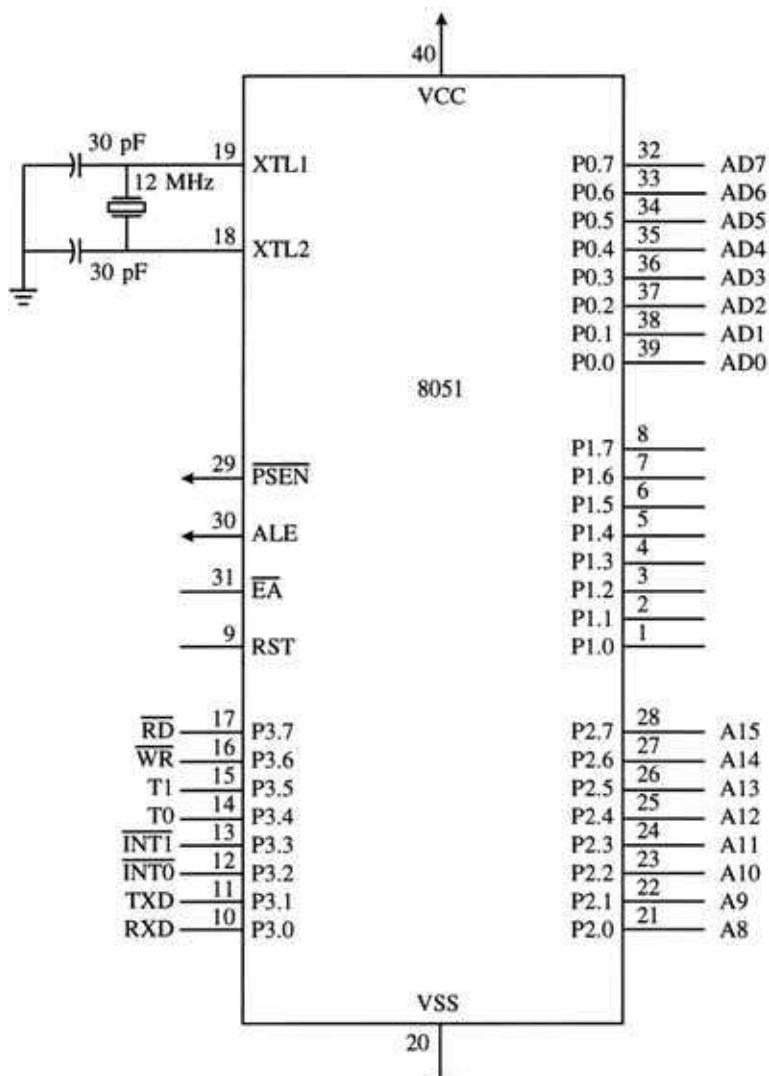
tanto, sólo se utilizan como interfaces para dispositivos externos. Los circuitos integrados 8032/8052 son la excepción, pues utilizan las terminales P1.0 o P1.1 ya sea como líneas de E/S o como entradas externas del tercer temporizador.

2.2.3 Puerto 2

El puerto 2 (terminales 21-28) es un puerto de propósito dual que sirve como E/S de propósito general, o como el byte superior del bus de direcciones en diseños que utilizan memoria externa para código o más de 256 bytes de memoria externa para datos. (Consulte la sección 2.7 Memoria externa).

2.2.4 Puerto 3

El puerto 3 es un puerto de propósito dual en las terminales 10-17. Se puede utilizar como E/S de propósito general, pero también cumple múltiples funciones ya que sus terminales tienen un propósito alterno relacionado con las características especiales del 8051. En la tabla 2-2 se sintetiza el propósito alterno de los puertos 3 y 1.

**FIGURA 2-2**

Asignación de terminales del 8051

2.2.5 PSEN (Habilitación de programa almacenado)

El 8051 tiene cuatro señales dedicadas al control del bus. La señal de habilitación de programa almacenado (PSEN) es una señal de salida en la terminal 29. Sirve como señal de control que permite utilizar memoria externa para programas (código). A menudo se conecta a la terminal de habilitación de salida (OE) de una memoria EPROM (Memoria borrable y programable de sólo lectura) para permitir la lectura de bytes de un programa.

La señal (PSEN) envía un pulso de nivel bajo durante la etapa de búsqueda (fetch) de una instrucción, la cual se encuentra almacenada en la memoria externa para programas. Los códigos binarios de un programa (códigos de operación) se leen de la memoria EPROM, viajan a lo

TABLA 2-2

Funciones alternas para las terminales de los puertos

Bit	Nombre	Dirección de bit	Función alterna
P3.0	RXD	B0H	Recepción de datos para el puerto serial
P3.1	<u>TXD</u>	B1H	Transmisión de datos para el puerto serial
P3.2	<u>INT0</u>	B2H	Interrupción externa 0
P3.3	<u>INT1</u>	B3H	Interrupción externa 1
P3.4	T0	B4H	Entrada externa al temporizador/contador 0
P3.5	T1	B5H	Entrada externa al temporizador/contador 1
P3.6	<u>WR</u>	B6H	Señal de sincronización de escritura a memoria externa para datos
P3.7	<u>RD</u>	B7H	Señal de sincronización de lectura de memoria externa para datos
P1.0	T2	90H	Entrada externa al temporizador/contador 2
P1.1	T2EX	91H	Captura/recarga del temporizador/contador 2

largo del bus de datos, y se almacenan en el registro de instrucciones del 8051 para su decodificación. La señal (PSEN) se mantiene en el estado inactivo (alto) cuando se ejecuta un programa almacenado en la memoria ROM interna (8051/8052).

2.2.6 ALE (Habilitación de latch de dirección)

La señal de salida ALE en la terminal 30 debe ser conocida para cualquiera que haya trabajado con los microprocesadores 8085, 8088 u 8086 de Intel. Igual que estos microprocesadores, el 8051 utiliza la señal ALE para demultiplexar los buses de direcciones y de datos. Cuando se utiliza el puerto 0 en su modo alterno (como bus de datos y como byte inferior del bus de direcciones), la señal ALE permite retener la dirección en un registro externo durante la primera mitad de un ciclo de memoria. Después de esta acción, las líneas del puerto 0 están disponibles para actuar como entradas o salidas durante la segunda mitad del ciclo de memoria, que es cuando se lleva a cabo la transferencia de datos. (Consulte la sección 2.7 Memoria externa).

La señal ALE envía pulsos a una tasa igual a 1/6 de la frecuencia del oscilador incorporado en el chip y se puede utilizar como reloj de propósito general para el resto del sistema. La señal ALE oscila a 2 MHz si se aplican pulsos de reloj al 8051 mediante un cristal de 12 MHz. Una excepción a esta regla es cuando se ejecuta la instrucción MOVX, ya que se pierde un pulso de la señal ALE. (Consulte la figura 2-11). Esta terminal también se utiliza para enviar el pulso de la entrada de programación de la memoria EPROM en las versiones del 8051 que cuentan con ella.

2.2.7 EA (Acceso externo)

La señal de entrada EA en la terminal 31 se mantiene por lo general a nivel alto (+5 V) o a nivel bajo (tierra). Si está a nivel alto, el 8051/8052 ejecuta los programas de su memoria ROM interna. Cualquier intento de ejecución de un programa en una zona de memoria más allá de sus 4K/8K de memoria interna, se dirigirá automáticamente a la memoria externa. Si la señal está a nivel bajo, únicamente se ejecutan los programas de la memoria externa (y la señal PSEN envía un pulso de nivel bajo como resultado). La EA debe mantenerse a nivel bajo en los microprocesadores 8031/8032, ya que no cuentan con memoria de programa incorporada al chip. La memoria ROM interna se deshabilita y los programas se ejecutan desde una EPROM externa cuando la señal EA se mantiene a nivel bajo. Las versiones del 8051 que tienen memoria EPROM también utilizan la línea EA para aplicar un voltaje de +21 volts (V_{pp}), necesario para programar la memoria EPROM interna.

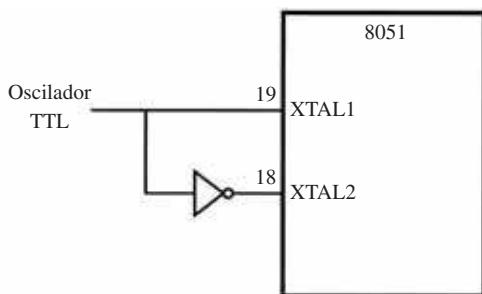


FIGURA 2-3
Control del 8051 mediante un oscilador TTL

2.2.8 RST (Reset, o reinicio)

La señal de entrada RST en la terminal 9 es la señal maestra de reinicio del 8051. Los registros internos del 8051 se cargan con los valores apropiados para efectuar un ciclo de inicio ordenado del sistema cuando esta señal se mantiene a nivel alto, durante por lo menos dos ciclos de máquina. La señal RST se mantiene a nivel bajo durante la operación normal del 8051. (Consulte la sección 2.9 Operación de reinicio).

2.2.9 Entradas del oscilador incorporado en el chip

Tal como se muestra en la figura 2-2, el 8051 incorpora en el chip un oscilador que por lo general se controla mediante un cristal conectado a las terminales 18 y 19. También se requieren condensadores de estabilización, como indica la figura.

La frecuencia nominal del cristal es de 12 MHz para la mayoría de los circuitos integrados en la familia MCS-51TM, aunque el 80C31BH-1 puede operar a través de cristales a frecuencias de hasta 16 MHz. El oscilador incorporado al chip no requiere de un cristal para controlarlo. Se puede conectar una fuente de reloj TTL a las terminales XTAL1 y XTAL2, según muestra la figura 2-3.

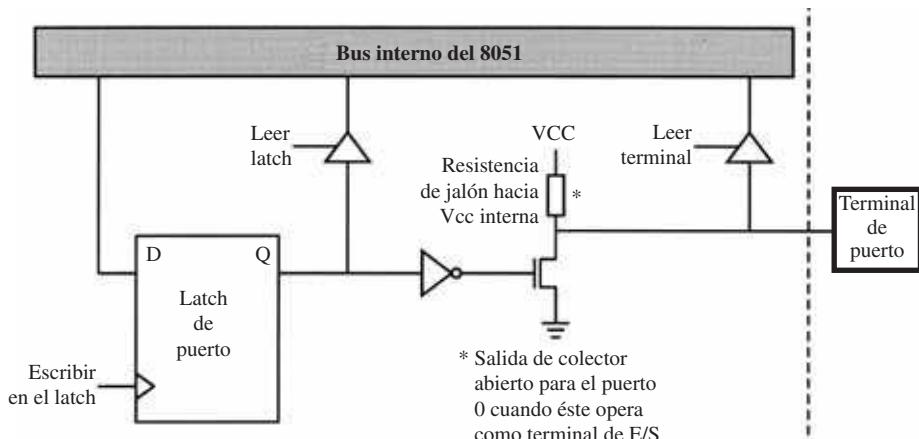
2.2.10 Conexiones de energía

El 8051 opera mediante una sola fuente de alimentación de + 5 voltios. La conexión V_{CC} está en la terminal 40 y la conexión V_{SS} (tierra) en la terminal 20.

2.3 ESTRUCTURA DE LOS PUERTOS DE E/S

En la figura 2-4 se muestra de manera simplificada la circuitería interna de las terminales de los puertos. Si escribimos a la terminal de un puerto se cargan datos en un latch del puerto, el cual controla un transistor de efecto de campo conectado a la terminal. Los puertos 1, 2 y 3 tienen capacidad para controlar cuatro unidades de carga del tipo Schottky TTL de bajo consumo de energía, y el puerto 0, 8 unidades del mismo tipo. (Consulte el apéndice E para obtener más detalles). Observe que la resistencia de jalón hacia V_{CC} (pull-up) no está presente en el puerto 0, a menos que se requiera que éste funcione como bus de direcciones/datos externo. Tal vez se requiera una resistencia externa de jalón hacia V_{CC} (pull-up), dependiendo de las características de entrada del dispositivo controlado por la terminal.

Existe capacidad tanto para “leer el latch” como para “leer la terminal”. Las instrucciones que requieren de una operación de lectura-modificación-escritura (CPL P1.5, por ejemplo) leen el latch para evitar que el nivel de voltaje se malinterprete en el caso de que la terminal tenga una carga pesada (por ejemplo, cuando se controla la terminal de base de un transistor). Las instrucciones que envían datos de entrada a un bit de puerto (MOV C,P1.5, por ejemplo) leen la terminal. El latch del puerto debe contener el valor 1, en este caso, de otra manera el controla-

**FIGURA 2–4**

Circuitería para los puertos de E/S

dor del FET está ENCENDIDO y lleva a la salida al nivel bajo. El reinicio de un sistema establece a 1 todos los latches de los puertos, así que se pueden utilizar las terminales de los puertos como entradas sin tener que ponerlos a 1 de manera explícita. No obstante, si un latch de puerto es puesto a cero (digamos, mediante la instrucción CLR P1.5), no podrá funcionar después como entrada a menos que se establezca en 1 primero (por ejemplo, SETB P1.5).

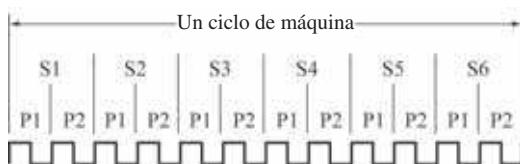
La figura 2-4 no muestra la circuitería necesaria para ejecutar las funciones alternas de los puertos 0, 2 y 3. Cuando la función alterna está en efecto, los controladores de salida se comunican a una dirección interna (puerto 2), a datos o direcciones (puerto 0), o a una señal de control (puerto 3), según sea apropiado.

2.4 SINCRONIZACIÓN Y EL CICLO DE MÁQUINA

El oscilador incorporado en el chip del 8051 se controla mediante un cristal de cuarzo externo a través de las terminales 18 y 19. Este cristal tiene una frecuencia típica de 12 MHz, lo cual significa que genera 12 millones de ciclos de reloj por segundo. Estos ciclos de reloj del oscilador constituyen la base de la sincronización del 8051: cada operación realizada por el 8051 se lleva a cabo de manera acompañada con estos ciclos.

Con el reloj del oscilador como referencia, el 8051 requiere de dos ciclos de reloj para ejecutar una sola operación discreta, que puede ser la búsqueda (fetching), decodificación o ejecución de una instrucción. A la duración de estos dos ciclos se le conoce también como **estado**. Por lo general, el 8051 requiere de seis estados, o 12 ciclos de reloj, para procesar una instrucción por completo, ya que primero tiene que buscar y decodificar la instrucción antes de ejecutarla. A la duración de estos seis estados también se le llama **ciclo de máquina**. Se requieren más ciclos de máquina para llevar a cabo una instrucción entre más compleja sea ésta. Los apéndices B y C presentan una lista de la cantidad de ciclos de máquina necesarios para ejecutar cada una de las instrucciones del 8051. Este número varía de uno a cuatro ciclos de máquina. La figura 2-5 muestra la relación que hay entre los ciclos de reloj del oscilador (P), los estados (S), y un ciclo de máquina.

El oscilador incorporado en el chip del 8051, f_{osc} está controlado por un cristal que oscila a 12MHz, por lo que el periodo de un ciclo de reloj es, $T_{clock} = 1/f_{osc} = 1/12 \text{ MHz} = 83.33 \text{ ns}$. Un ciclo de máquina consta de 12 pulsos de reloj, por lo tanto su duración es de $83.33 \text{ ns} \times 12 = 1 \mu\text{s}$.

**FIGURA 2–5**

Relación entre los ciclos de reloj del oscilador, los estados, y el ciclo de máquina

2.5 ORGANIZACIÓN DE LA MEMORIA

La mayoría de los microprocesadores implementan un espacio de memoria que se comparte entre los datos y los programas. Esto es razonable, ya que los programas a menudo se almacenan en un disco y se cargan a la memoria RAM para su ejecución; es por eso que tanto los datos como los programas residen en la memoria RAM. Por otra parte, los microcontroladores raras veces se utilizan como la CPU en “sistemas computacionales”, se usan más bien como el componente central en diseños orientados al control. La capacidad de memoria es limitada y no existe una unidad de disco o un sistema operativo en disco. El programa de control debe residir en la memoria ROM.

Es por esta razón que el 8051 implementa espacios separados de memoria para los programas (código) y los datos. Tanto el código como los datos pueden estar almacenados de manera interna, como se muestra en la tabla 2-1; sin embargo, ambos pueden expandirse mediante componentes externos hasta llegar a un máximo de 64K de memoria para código y 64K de memoria para datos.

La memoria interna consta de memoria ROM incorporada en el chip (sólo en el 8051/8052) y memoria RAM incorporada en el chip. La memoria RAM contiene áreas de almacenamiento de propósito general para bytes y bits individuales, bancos de registros, y registros con funciones especiales.

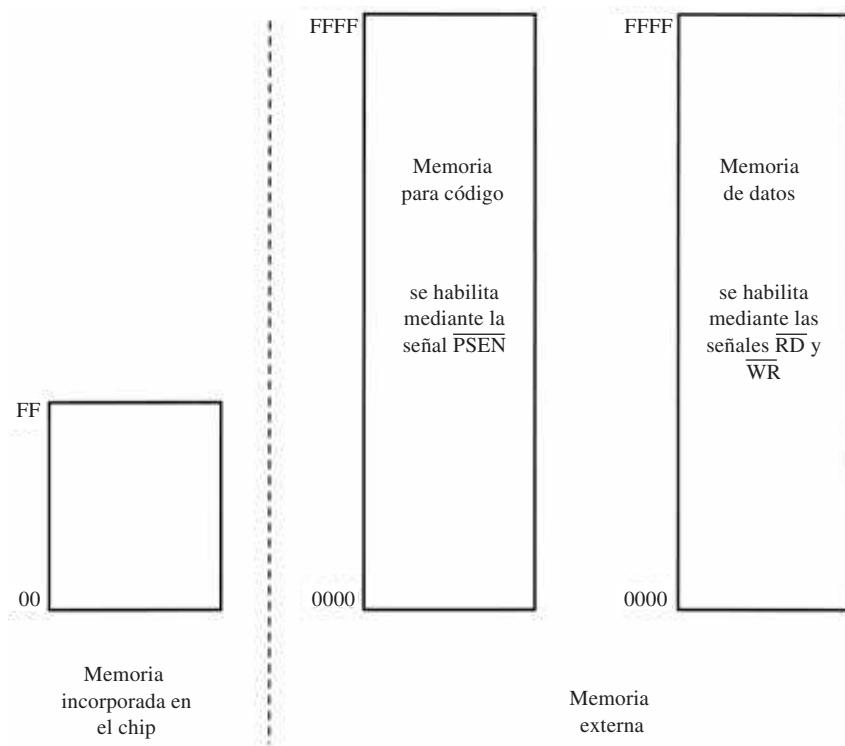
Hay que observar dos características importantes: (a) los registros y los puertos de entrada/salida están “mapeados” a memoria, a los cuales se puede acceder como a cualquier otra localidad de memoria, y (b) la pila reside dentro de la RAM interna en lugar de en la RAM externa, como es típico en el caso de los microprocesadores.

En la figura 2-6 se presenta un resumen de los diferentes espacios de memoria disponibles en el dispositivo 8031 (el cual no cuenta con memoria ROM) sin mostrar ningún detalle sobre la memoria de datos incorporada en el chip. (Más adelante sintetizaremos las características de los dispositivos mejorados del 8032/8052).

La figura 2-7 presenta detalles sobre la memoria de datos incorporada en el chip. Como se muestra en la figura, el espacio de memoria interna de datos se divide entre la **memoria RAM interna** (00H-7FH) y los **registros con funciones especiales** (80H-FFH). La diferencia entre el concepto de memoria de datos interna (incorporada en el chip) y la RAM interna puede causar confusión algunas veces. El espacio de memoria de datos interno del 8051 tiene un rango que abarca las direcciones 00H-FFH, lo cual constituye un espacio de 256 bytes. Sin embargo, solamente la parte baja (00H-7FH) del espacio de memoria interna se utiliza para datos de uso general, mientras que la parte alta (80H-FFH) se usa casi siempre para propósitos específicos y no para datos de uso general; es por ello que únicamente la parte baja se considera como RAM interna. La RAM interna de datos se subdivide en bancos de registros (00H-1FH), RAM direccionable por bits (20H-2FH), y RAM de propósito general (30H-7FH). A continuación veremos cada una de estas secciones de la memoria interna.

2.5.1 Memoria RAM de propósito general

Aunque la figura 2-7 muestra 80 bytes de memoria RAM de propósito general en el rango de direcciones de 30H a 7FH, los 48 bytes más bajos en el rango de 00H a 2FH pueden usarse de forma similar (aunque estas ubicaciones tienen otros propósitos, como veremos en los siguientes párrafos).

**FIGURA 2-6**

Resumen de los espacios de memoria del 8031

Podemos acceder a cualquier ubicación en la memoria RAM de propósito general, sin ningún problema, mediante el uso de los modos de direccionamiento directo o indirecto. Por ejemplo, para leer el contenido de la dirección de RAM interna 5FH y transferirlo al acumulador, se utiliza la siguiente instrucción:

```
MOV A, 5FH
```

Esta instrucción transfiere un byte de datos utilizando el modo de direccionamiento directo para especificar la “ubicación de origen” (en otras palabras, la dirección 5FH). El destino de los datos se especifica de manera implícita en la instrucción de código, mediante la referencia al acumulador A. (Nota: en el capítulo 3 hablaremos con más detalle sobre los modos de direccionamiento).

También se puede acceder a la RAM interna mediante el uso del modo de direccionamiento indirecto, a través de R0 o R1. Por ejemplo, las siguientes dos instrucciones realizan la misma operación que la instrucción anterior:

```
MOV R0, #5FH  
MOV A, @R0
```

La primera instrucción utiliza el modo de direccionamiento inmediato para transferir el valor 5FH al registro R0, y la segunda instrucción usa el modo de direccionamiento indirecto para transferir los datos “a los que apunta R0” al acumulador.

Dirección de byte	Dirección de bit	Dirección de byte	Dirección de bit	
7F	Memoria RAM de propósito general	FF		
30		F0	F7 F6 F5 F4 F3 F2 F1 F0 B	
2F		E0	E7 E6 E5 E4 E3 E2 E1 E0 ACC	
2E	7F 7E 7D 7C 7B 7A 79 78	D0	D7 D6 D5 D4 D3 D2 - D0 PSW	
2D	77 76 75 74 73 72 71 70	B8	- - - BC BB BA B9 B8 IP	
2C	6F 6E 6D 6C 6B 6A 69 68	B0	B7 B6 B5 B4 B3 B2 B1 B0 P3	
2B	67 66 65 64 63 62 61 60	A8	AF - - AC AB AA A9 A8 IE	
2A	5F 5E 5D 5C 5B 5A 59 58	A0	A7 A6 A5 A4 A3 A2 A1 A0 P2	
29	57 56 55 54 53 52 51 50	99	no accesible por bits SBUF	
28	4F 4E 4D 4C 4B 4A 49 48	98	9F 9E 9D 9C 9B 9A 99 98 SCON	
27	47 46 45 44 43 42 41 40	90	97 96 95 94 93 92 91 90 P1	
26	3F 3E 3D 3C 3B 3A 39 38	8D	no accesible por bits TH1	
25	37 36 35 34 33 32 31 30	8C	no accesible por bits TH0	
24	2F 2E 2D 2C 2B 2A 29 28	8B	no accesible por bits TL1	
23	27 26 25 24 23 22 21 20	8A	no accesible por bits TL0	
22	1F 1E 1D 1C 1B 1A 19 18	89	no accesible por bits TMOD	
21	17 16 15 14 13 12 11 10	88	8F 8E 8D 8C 8B 8A 89 88 TCON	
20	0F 0E 0D 0C 0B 0A 09 08	87	no accesible por bits PCON	
1F	07 06 05 04 03 02 01 00	83	no accesible por bits DPH	
18	Banco 3	82	no accesible por bits DPL	
17		81	no accesible por bits SP	
10	Banco 2	80	87 86 85 84 83 82 81 80 P0	
0F		REGISTROS CON FUNCIONES ESPECIALES		
08	Banco 1			
07	Banco de registros por defecto para R0-R7			
00	RAM			

FIGURA 2-7

Resumen de la memoria de datos incorporada en el chip del 8051

2.5.2 RAM accesible por bits individuales

El 8051 cuenta con 210 ubicaciones accesibles para almacenar bits individuales, de los cuales 128 están en las direcciones de byte 20H a 2FH, y el resto se encuentra en los registros con funciones especiales (que veremos más tarde).

La idea de poder acceder a los bits en forma individual mediante el uso de software es una característica poderosa de la mayoría de los microcontroladores. Los bits se pueden cambiar a 1, a 0, es posible realizar operaciones lógicas con ellos, como AND, OR, etc., utilizando una sola instrucción. La mayoría de los microprocesadores requiere de una secuencia de instrucciones de lectura-modificación-escritura para causar el mismo efecto. Además, los bits de los puertos de E/S del 8051 pueden direccionarse individualmente, lo cual simplifica la interfaz de software para las entradas y salidas de un solo bit.

Existen 128 localizaciones direccionables por bits individuales de propósito general en las direcciones de byte 20H a 2FH (8 bits/byte × 16 bytes = 128 bits). Se puede acceder a estas direcciones como bytes o como bits, dependiendo de la instrucción. Por ejemplo, podemos utilizar la siguiente instrucción para establecer en 1 el bit 67H:

```
SETB 67H
```

Si consultamos la figura 2-7 podremos observar que la “dirección de bit 67H” es el bit más significativo en la “dirección de byte 2CH”. La instrucción anterior no afecta a ninguno de los otros bits localizados en la misma dirección. La mayoría de los microprocesadores realizaría la misma operación de la siguiente forma:

<pre>MOV A, 2CH</pre>	<i>;LEE BYTE COMPLETO</i>
<pre>ORL A,#10000000B</pre>	<i>;ESTABLECE EN 1 BIT MÁS SIGNIFICATIVO</i>
<pre>MOV 2CH,A</pre>	<i>;ESCRIBE BYTE COMPLETO</i>

EJEMPLO 2.1

¿Qué instrucción se utilizaría para establecer en 1 el bit 3 de la dirección de byte 25H?

Solución

```
SETB 2BH
```

Análisis

La dirección de byte 25H se encuentra dentro del área direccionable por bits de la memoria interna (consulte la figura 2-7). Las direcciones de bit dentro de este byte, empezando por el bit 0, son 28H, 29H, etc. El bit 3 dentro de la dirección de byte 25H es la dirección de bit 2BH.

2.5.3 Bancos de registros

Las 32 ubicaciones inferiores de la memoria interna contienen los bancos de registros. El conjunto de instrucciones del 8051 soporta ocho registros, denominados R0, R1, R2, R3, R4, R5, R6 y R7, y éstos se encuentran en las direcciones 00H-07H de manera predeterminada (después de un reinicio del sistema). La siguiente instrucción lee entonces el contenido de la dirección 05H al acumulador:

```
MOV A,R5
```

Esta instrucción es de un solo byte y utiliza direccionamiento por registros. Desde luego que podría realizarse la misma operación en una instrucción de 2 bytes, mediante el uso de la dirección directa como el byte 2:

```
MOV A,05H
```

Las instrucciones que utilizan los registros R0 a R7 son más cortas que las equivalentes que utilizan direccionamiento directo. Es conveniente utilizar uno de estos registros para los valores de datos usados con frecuencia.

El banco de registros activo puede cambiarse si se modifican los bits de selección de banco de registros en la palabra de estado de programa (lo cual veremos a continuación). La siguiente instrucción escribe el contenido del acumulador en la ubicación 18H, si asumimos que el banco de registros 3 está activo:

```
MOV R0,A
```

El concepto de “bancos de registros” permite realizar una rápida y efectiva “conmutación de contexto”, lo cual posibilita que secciones individuales del software puedan utilizar un conjunto de registros privado, independiente de las otras secciones del software.

EJEMPLO ¿Cuál es la dirección del registro 5 en el banco de registros 3?

2.2

Solución

1DH

Análisis

El banco de registros 3 ocupa las ubicaciones de memoria interna 18H a 1FH (consulte la figura 2-7), en donde el registro R0 está localizado en la dirección 18H, R1 en la dirección 19H, etc. El registro 5 (R5) está en la dirección 1DH.

2.6 REGISTROS CON FUNCIONES ESPECIALES

En la mayoría de los microprocesadores se accede a los registros internos de manera implícita mediante el conjunto de instrucciones. Por ejemplo, en el microprocesador 6809 la instrucción “INCA” incrementa el contenido del acumulador A. La operación está especificada de manera implícita dentro de la instrucción de operación de código. En el microcontrolador 8051 también se utiliza un acceso similar a los registros. De hecho, la instrucción “INC A” del 8051 realiza la misma operación.

Los registros internos del 8051 están configurados como parte de la RAM incorporada en el chip, por lo que cada registro tiene también una dirección.¹ Esto es aceptable en el 8051, ya que tiene muchos registros. Además de los registros R0 a R7 existen 21 más con funciones especiales (SFR) en la parte superior de la RAM interna, en el rango de direcciones 80H a FFH. (Consulte la figura 2-7 y el apéndice D). Observe que la mayoría de las 128 direcciones en el rango de 80H a FFH no están definidas. Sólo 21 direcciones de SFR están definidas (26 en el 8032/8052).

Se puede acceder a la mayoría de los SFR mediante el modo de direccionamiento directo, aun cuando es posible ingresar al acumulador (A o ACC) de manera implícita, como se mostró antes. Observe que en la figura 2-7 algunos de los SFR se pueden direccionar tanto por bits como por bytes. Los programadores deben tener cuidado al acceder a bits en lugar de a bytes. Por ejemplo, la instrucción

SETB 0E0H

restablece en 0 el bit 0 del acumulador, pero no cambia ninguno de los otros bits. El truco está en reconocer que la dirección E0H es la dirección de byte del acumulador y también la dirección del bit menos significativo del acumulador. Sólo se afecta el bit direccionado porque la instrucción SETB opera con bits (no con bytes). Observe que los bits direccionables dentro de los SFR son los cinco bits de dirección de mayor orden que concuerdan con los del SFR. Por ejemplo, el puerto 1 está en la dirección de byte 90H o 10010000B. Los bits localizados dentro del puerto 1 tienen las direcciones de 90H a 97H, o 10010xxxB.

¹El contador de programa y el registro de instrucciones son la excepción. No hay ninguna ganancia si éstos se incluyen en la memoria RAM incorporada en el chip, debido a que casi nunca se manipulan directamente.

EJEMPLO 2.3 ¿Cuál es la instrucción que puede utilizarse para establecer en 1 el bit más significativo en el acumulador B sin afectar a los otros bits?

Solución

SETB 0F7H

Análisis

El acumulador B se encuentra en la dirección de byte F0H en el espacio para registros con funciones especiales de la memoria interna (consulte la figura 2-7). Se puede acceder a los bits individuales del acumulador B. En efecto, el bit 0 se ubica en la dirección de bit F0H, el bit 1 en la dirección F1H, etc. Entonces el bit 7 del acumulador B se encuentra en la dirección de bit F7H.

Hablaremos sobre la PSW con todo detalle en la siguiente sección. Los otros SFR serán presentados brevemente después de la sección de la PSW, y los trataremos con más detalle en capítulos posteriores.

2.6.1 Palabra de estado del programa

La palabra de estado del programa (PSW) ubicada en la dirección D0H contiene los bits de estado, como se muestra en la tabla 2-3. Examinaremos cada bit de la PSW en las siguientes secciones.

2.6.1.1 Bandera de acarreo La bandera de acarreo (C o CY) es de doble propósito. Se utiliza en la manera tradicional para realizar operaciones aritméticas: se vuelve 1 si hay un acarreo del bit 7 durante una suma o si hay un préstamo hacia el bit 7 durante una resta. Por ejemplo, si el acumulador contiene el valor FFH, entonces la instrucción

ADD A, #1

deja al acumulador con un valor de 00H y establece en 1 la bandera de acarreo en la PSW.

TABLA 2-3

Resumen de los registros de la PSW (palabra de estado del programa)

Bit	Símbolo	Dirección	Descripción del bit
PSW.7	CY	D7H	Bandera de acarreo
PSW.6	AC	D6H	Bandera auxiliar de acarreo
PSW.5	F0	D5H	Bandera 0
PSW.4	RS1	D4H	Selección de banco de registros 1
PSW.3	RS0	D3H	Selección de banco de registros 0 00 = banco 0; direcciones 00H–07H 01 = banco 1; direcciones 08H–0FH 10 = banco 2; direcciones 10H–17H 11 = banco 3; direcciones 18H–1FH
PSW.2	OV	D2H	Bandera de desbordamiento
PSW.1	—	D1H	Reservado
PSW.0	P	D0H	Bandera de paridad par

EJEMPLO 2.4 ¿Cuál es el estado de la bandera de acarreo y el contenido del acumulador después de que se ejecuta la siguiente secuencia de instrucciones?

```
MOV R5, #55H
MOV A, #0AAH
ADD A, R5
```

Solución

$$C = 0, ACC = FFH$$

Análisis

La suma binaria llevada a cabo en la tercera instrucción se muestra enseguida.

$$\begin{array}{r} 01010101 \quad (R5 = 55H) \\ +10101010 \quad (ACC = AAH) \\ \hline 11111111 \quad (\text{Resultado en ACC} = FFH) \end{array}$$

Esta suma no genera un acarreo del bit más significativo (bit 7), por lo tanto el bit de acarreo se restablece en 0. El resultado final en el acumulador es el valor $FFH = 255_{10}$.

La bandera de acarreo también se conoce como “acumulador booleano”, y funciona como un registro de 1 bit para ejecutar instrucciones booleanas que operan sobre bits. Por ejemplo, la siguiente instrucción realiza una operación AND entre el bit 25H y la bandera de acarreo y transfiere el resultado a la bandera de acarreo:

```
ANL C, 25H
```

2.6.1.2 Bandera auxiliar de acarreo La bandera auxiliar de acarreo (AC) se establece en 1 cuando se realiza una suma de valores decimales codificados en binario (BCD), siempre y cuando se haya generado un acarreo del bit 3 al bit 4 o si el resultado en el nibble inferior se encuentra en el rango 0AH-0FH. Si los valores que se suman son BCD, la instrucción de suma debe ir seguida de la instrucción DA A (acumulador de ajuste decimal) para que los resultados mayores a 9 vuelvan a quedar dentro del rango válido.

EJEMPLO 2.5 ¿Cuál es el estado de la bandera auxiliar de acarreo y el contenido del acumulador después de que se ejecuta la siguiente secuencia de instrucciones?

```
MOV R5, #1
MOV A, #9
ADD A, R5
```

Solución

$$AC = 1, ACC = 0AH$$

Análisis

La suma binaria que ocurre en la tercera instrucción se muestra enseguida.

$$\begin{array}{r} 1 \\ 00000001 \quad (R5 = 01H) \\ +00001001 \quad (ACC = 09H) \\ \hline 00001010 \quad (\text{Resultado en ACC} = 0AH) \end{array}$$

Aunque no ocurre ningún acarreo en esta suma binaria, el nibble inferior del resultado es $1010B = AH$. La bandera auxiliar de acarreo cambia a 1 debido a que este resultado es mayor que 9_{10} . Si la instrucción de suma va seguida de una instrucción de ajuste decimal (DA A), el resultado final en el acumulador es $0001000B = 10H$. Como este número es un decimal codificado en binario, $10H = 10_{10}$ lo cual es el resultado correcto de la suma $9_{10} + 1_{10}$.

2.6.1.3 Bandera 0 La bandera 0 (F0) es un bit de bandera de propósito general disponible para aplicaciones de usuario.

2.6.1.4 Bits de selección del banco de registros Los bits de selección del banco de registros (RS0 y RS1) determinan el banco de registros que está activo. Estos bits se restablecen en 0 después de un reinicio del sistema y el software puede cambiarlos de acuerdo a sus necesidades. Por ejemplo, las siguientes tres instrucciones habilitan el banco de registros 3 y transfieren el contenido del registro R7 (dirección de byte 1FH) al acumulador:

```
SETB RS1
SETB RS0
MOV A, R7
```

Las direcciones de bit correctas reemplazan a los símbolos “RS1” y “RS0” cuando el programa se ensambla, por lo que la instrucción SETB RS1 resulta ser la misma que SETB 0D4H.

EJEMPLO 2.6 Muestre una secuencia de instrucciones para hacer que el banco de registros 2 sea el banco de registros activo. Suponga que se desconoce cuál es el banco de registros que estaba activo antes.

Solución

```
SETB RS1
CLR RS0
```

Análisis

El resultado de estas instrucciones transfiere el valor $10_2 = 2_{10}$ a los bits de selección del banco de registros en la palabra de estado del programa. El banco de registros 2 se convierte en el banco de registros activo. La segunda instrucción no sería necesaria si supiéramos que los bits del banco de registros no han cambiado desde la última vez que se realizó una operación de reinicio en la CPU; sin embargo, tuvimos que inicializar de manera explícita ambos bits de selección del banco de registros pues asumimos que su estado era desconocido.

2.6.1.5 Bandera de desbordamiento La bandera de desbordamiento (OV) cambia a 1 después de una operación de suma o resta, si hubo un desbordamiento aritmético. El software puede examinar esta bandera para determinar si el resultado se encuentra en el rango apropiado cuando se suman o restan números con signo. Podemos ignorar el bit OV cuando se realizan sumas con número sin signo. Los resultados con valores mayores que +127 o menores que -128 cambiarían a 1 el bit OV. Por ejemplo, la siguiente suma produce un desbordamiento y establece en 1 el bit OV en la PSW

HEX :	OF	Decimal :	15
	<u>+7F</u>		<u>+127</u>
	8E		142

El valor 8EH representa el número con signo -116, el cual no es el resultado correcto de 142; por lo que esto establece en 1 el bit OV.

EJEMPLO 2.7 ¿Cuál es el estado de la bandera de desbordamiento y el contenido del acumulador después de que se ejecuta la siguiente secuencia de instrucciones?

```
MOV R7, #0FFH
MOV A, #0FH
ADD A, R7
```

Solución

$$OV = 0, ACC = 0EH$$

Análisis

El registro R7 se inicializa con el valor FFH, el cual representa un número con signo y valor de -1_{10} . El acumulador se inicializa con el valor 0FH, que equivale a 15_{10} . El resultado de la suma es $15 + (-1) = 14 = 0EH$. No ocurre ningún desbordamiento, y el bit OV se restablece en 0 debido a que el valor 14 está dentro del rango permitido para los números con signo de 8 bits (-28 to $+127$). (Observe, sin embargo, que el bit C se establece en 1 debido a que la suma genera un acarreo del bit 7).

2.6.1.6 Bit de paridad El bit de paridad (P) cambia a 1 o a 0 de manera automática en cada ciclo de máquina para establecer la paridad par con el acumulador. La suma del número de bits en 1 en el acumulador más el bit P siempre es un resultado par. Por ejemplo, el bit P contendrá el valor 1 si el acumulador contiene el valor 10101101 (lo cual establece un total de seis bits iguales a 1; en otras palabras, un número par de unos). El bit de paridad se utiliza con mayor frecuencia junto con las rutinas del puerto serial para incluir un bit de paridad antes de la transmisión o para verificar la paridad después de una recepción.

EJEMPLO 2.8 ¿Cuál es el estado del bit P después de que se ejecuta la siguiente instrucción?

```
MOV A, #55H
```

Solución

$$P = 0$$

Análisis

El valor 55H equivale a 01010101B en binario. Este patrón presenta cuatro bits con valor de 1. El bit de P se pone en 0 debido a que el número 4 es un número par. El número total de bits que equivalen a un valor de 1, tomando en cuenta los que están en el acumulador y el bit P, es cuatro, lo cual resulta en una paridad par.

2.6.2 Registro B

El registro B, o acumulador B, en la dirección F0H se utiliza junto con el acumulador para realizar operaciones de multiplicación y división. La instrucción MUL AB multiplica los valores sin signo de 8 bits en A y B, y deja el resultado de 16 bits en A (byte inferior) y en B (byte superior). La instrucción DIV AB divide A entre B, dejando el resultado entero en A y el residuo en B. El registro B puede tratarse también como un registro auxiliar de propósito general. Es posible acceder a los bits individuales de este registro usando las direcciones F0H a F7H.

2.6.3 Apuntador de pila

El apuntador de pila (SP) es un registro de 8 bits en la dirección 81H, y contiene la dirección del dato que se encuentra actualmente en la parte superior de la pila. Las operaciones de pila incluyen operaciones para “meter” datos a la pila y “sacar” datos de la misma. Al meter datos en la pila se incrementa el SP antes de escribir los datos, y al sacar datos de la pila se leen los datos y entonces disminuye el SP. La pila del 8051 se mantiene dentro de la RAM interna y está limitada a las direcciones accesibles mediante el modo de direccionamiento indirecto. Éstas corresponden a los primeros 128 bytes en el 8031/8051 o el total de los 256 bytes en la RAM incorporada al chip del 8032/8052.

Para reinicializar el SP y que la pila comience en la dirección 60H, se utiliza la siguiente instrucción:

```
MOV SP, #5FH
```

Esto limitaría la pila a 32 bytes en el 8031/8051, ya que la dirección más alta disponible en la RAM incorporada al chip es 7FH. Utilizamos el valor 5FH debido a que el SP se incrementa al valor 60H antes de la primera operación de meter datos.

EJEMPLO 2.9

¿Cuál es la instrucción utilizada para inicializar el apuntador de pila en el 8052 y crear una pila de 48 bytes en la parte superior de la memoria interna?

Solución

```
MOV SP, #0CFH
```

Análisis

Como el 8052 sólo tiene 256 bytes de memoria interna, una pila de 48 bytes ubicada en la parte superior ocuparía las ubicaciones D0H a FFH. Debido a que el SP se incrementa antes de colocar el primer elemento en la pila, debe inicializarse con la dirección que está justo debajo de la ubicación inicial, CFH.

Los diseñadores pueden decidir que no es necesario reinicializar el apuntador de pila y dejar que éste retenga su valor predeterminado después de reiniciar el sistema. El valor de reinicio de 07H mantiene la compatibilidad con el predecesor del 8051: el 8048, ello causa que la primera operación de escritura en la pila almacene los datos en la ubicación 08H. El banco de registros 1 (y quizás los bancos 2 y 3) no estará disponible si el software de aplicación no reinicializa el SP, debido a que esta área de RAM interna es el área de la pila.

Las instrucciones PUSH (meter) y POP (sacar) acceden a la pila de manera explícita para almacenar y recuperar datos de manera temporal, o de manera implícita mediante la llamada a la subrutina (ACALL, LCALL) y las instrucciones de retorno (RET, RETI) para almacenar y restaurar el contador del programa.

2.6.4 Apuntador de datos

El apuntador de datos (DPTR) se utiliza para acceder a la memoria externa para código o para datos; es un registro de 16 bits ubicado en las direcciones 82H (DPL, byte inferior) y 83H (DPH, byte superior). Las siguientes tres instrucciones escriben el valor 55H en la ubicación de RAM externa 1000H:

```
MOV A, #55H
MOV DPTR, #1000H
MOVX @DPTR, A
```

La primera instrucción utiliza el modo de direccionamiento inmediato para cargar el valor constante 55H en el acumulador. La segunda instrucción también utiliza el modo de direccionamiento inmediato, pero carga el valor constante de la dirección de 16 bits 1000H en el apuntador de datos.

La tercera instrucción utiliza el modo de direccionamiento indirecto para transferir el valor almacenado en A (55H) a la ubicación de RAM externa cuya dirección se encuentra en el DPTR (1000H). La “X” en el nemónico “MOVX” indica que la instrucción de transferencia accede a la memoria externa para datos.

2.6.5 Registros de puerto

Los puertos de E/S del 8051 incluyen al puerto 0 en la dirección 80H, al puerto 1 en la dirección 90H, al puerto 2 en la dirección A0H, y al puerto 3 en la dirección B0H. Los puertos 0, 2 y 3 pueden no estar disponibles para uso de E/S cuando se está utilizando la memoria externa o algunas de las características especiales del 8051 (interrupciones, puerto serial, etc.). No obstante, las terminales P1.2 a P1.7 siempre están disponibles como líneas de E/S de propósito general.

Todos los bits de los puertos se pueden direccionar individualmente. Esta capacidad proporciona poderosas posibilidades de interfaz. Por ejemplo, si conectamos un motor a través de un solenoide y un controlador de transistor al bit 7 del puerto 1, podríamos encender o apagar este motor utilizando una sola instrucción del 8051:

```
SETB P1.7
```

la cual puede encender el motor, y

```
CLR P1.7
```

que puede apagarlo.

Las instrucciones anteriores utilizan el operador punto para direccionar un bit dentro de una ubicación de byte direccionable mediante bits. El ensamblador realiza la conversión necesaria; por ende, las siguientes dos instrucciones son iguales:

```
CLR P1.7
```

```
CLR 97H
```

En el capítulo 7 veremos con detalle el uso de los símbolos de ensamblador predefinidos (por ejemplo, P1).

Veamos otro ejemplo. Considere la interfaz para un dispositivo con un bit de estado llamado OCUPADO, el cual se establece en 1 cuando el dispositivo está ocupado y se vuelve 0 cuando está listo. Por decir, si la señal OCUPADO se conecta al bit 5 del puerto 1, podemos usar el siguiente ciclo para esperar a que el dispositivo esté listo:

```
WAIT: JB P1.5, WAIT
```

Esta instrucción significa “si el bit P1.5 es 1, salta a la etiqueta ESPERA”. En otras palabras, “salta hacia atrás y vérifícalo de nuevo”.

2.6.6 Registros de temporizadores

El 8051 posee dos contadores/temporizadores de 16 bits para manejar intervalos de tiempo o para contar eventos. El temporizador 0 está localizado en las direcciones 8AH (TL0, byte inferior) y 8CH (TH0, byte superior); y el temporizador 1 se localiza en las direcciones 8BH (TL1, byte inferior) y 8DH (TH1, byte superior). La operación de los temporizadores se establece mediante el registro de modo de temporizadores (TMOD) en la dirección 89H y el registro de control de temporizadores (TCON) en la dirección 88H. Sólo los bits del registro TCON se puede direccionar individualmente. En el capítulo 4 veremos con detalle los temporizadores.

2.6.7 Registros de puerto serial

El 8051 contiene un puerto serial incorporado en el chip para permitir la comunicación con dispositivos seriales como terminales o módems, o para implementar interfaces con otros circuitos integrados que cuentan con una interfaz serial (convertidores A/D, registros de desplazamiento,

memorias RAM no volátiles, etc.). Un registro, el buffer de datos serial (SBUF) en la dirección 99H, almacena tanto los datos para transmisión como los datos recibidos. Cargamos datos para transmisión si escribimos al SBUF; cuando leemos el SBUF, podemos acceder a los datos recibidos. Es posible programar varios modos de operación a través del registro de control de puertos seriales, mediante sus bits direccionables individualmente (SCON), el cual se encuentra en la dirección 98H. En el capítulo 5 veremos más a fondo esta operación del puerto serial.

2.6.8 Registros de interrupciones

El microprocesador 8051 tiene una estructura de interrupciones de niveles de cinco fuentes y 2 prioridades. Las interrupciones se deshabilitan después de un reinicio del sistema y se habilitan escribiendo en el registro de habilitación de interrupciones (IE) en la dirección A8H. El nivel de prioridad se establece a través del registro de prioridad de interrupción (IP) en la dirección B8H. Ambos registros son direccionables por bit. En el capítulo 6 hablaremos con detalle sobre las interrupciones.

2.6.9 Registro de control de energía

El registro de control de energía (PCON) en la dirección 87H contiene diversos bits de control. La tabla 2-4 presenta un resumen de estos bits.

El bit SMOD dobla la tasa de transmisión y recepción en baudios del puerto serial cuando utilizamos los modos 1, 2 o 3. (Consulte el capítulo 5). Los bits 6, 5 y 4 del PCON no están definidos. Los bits 3 y 2 son bits de bandera de propósito general y están disponibles para aplicaciones de usuario.

Los bits de control de energía, apagado (PD) y suspensión (IDL), estaban disponibles originalmente en toda la familia de circuitos integrados MCS-51TM, pero ahora sólo se implementan en las versiones CMOS. Los bits del registro PCON no son direccionables individualmente.

2.6.9.1 Modo de suspensión La última instrucción ejecutada antes de iniciar el modo de suspensión será la instrucción que establecerá en 1 el bit IDL. En el modo de suspensión la señal de reloj interna está desconectada de la CPU mediante una compuerta, pero no de las funciones de interrupción, temporizador y puerto serial. El estado de la CPU se conserva y todos los registros mantienen su contenido. Las terminales de puerto también retienen todos sus niveles lógicos. Las señales ALE y (\overline{PSEN}) se mantienen a nivel alto.

El modo de suspensión termina si se habilita cualquier interrupción o cuando el sistema se reinicia. Cualquiera de estas condiciones restablece en 0 el bit IDL.

TABLA 2-4

Resumen del registro PCON

Bit	Símbolo	Descripción
7	SMOD	Bit para doblar la tasa en baudios; cuando se establece en 1, la tasa en baudios se duplica en los modos 1, 2 o 3 del puerto serial
6	—	No definido
5	—	No definido
4	—	No definido
3	GF1	Bit de bandera de propósito general 1
2	GF0	Bit de bandera de propósito general 0
1*	PD	Modo de apagado; establecer en 1 para activar el modo de apagado; la única salida es reiniciar
0*	IDL	Modo de reposo; establecer en 1 para activar el modo de reposo; la única salida es mediante una interrupción o el reinicio del sistema

*Sólo las versiones CMOS implementan estas funciones

2.6.9.2 Modo de apagado La última instrucción ejecutada antes de iniciar el modo de apagado será la que establecerá en 1 el bit PD. Lo siguiente ocurre en el modo de apagado: (1) se detiene el oscilador incorporado en el chip, (2) se detienen todas las funciones, (3) se retiene todo el contenido de la RAM incorporada en el chip, (4) las terminales de puerto retienen sus niveles lógicos, y (5) las señales ALE y (PSEN) se mantienen a nivel bajo. La única manera de salir de este modo es mediante el reinicio del sistema.

El voltaje en la terminal V_{cc} puede ser tan bajo como 2V durante el modo de apagado. Hay que tener cuidado de no reducir el voltaje en V_{cc} hasta que el modo de apagado haya iniciado, y de no restablecer un voltaje de 5V en V_{cc} sino hasta que hayan transcurrido, por lo menos, 10 ciclos de oscilador antes de que la terminal de RST llegue de nuevo al nivel bajo (después de salir del modo de apagado).

2.7 MEMORIA EXTERNA

Es importante que los microcontroladores tengan capacidad de expandirse más allá de los recursos incorporados al chip para evitar un potencial embotellamiento en el diseño. Esta capacidad debe existir si cualquiera de los recursos deben expandirse (memoria, E/S, etc.). La arquitectura de la familia MCS-51TM provee esta capacidad mediante los espacios de 64K en la memoria externa para código y para datos. Se puede añadir memoria ROM y RAM adicional conforme sea necesario. También es posible agregar circuitos integrados de interfaz periférica para expandir la capacidad de E/S. Estos circuitos integrados se convierten en parte del espacio en memoria externa para datos mediante el uso de E/S por asignación de memoria.

El puerto 0 no está disponible para uso como puerto de E/S cuando se utiliza la memoria externa. Este puerto se convierte en un bus multiplexado de direcciones (A0-A7) y de datos (D0-D7), y la señal ALE permite fijar el byte inferior de la dirección al principio de cada ciclo de memoria externa. El puerto 2 se utiliza a menudo (pero no siempre) para el byte superior del bus de direcciones.

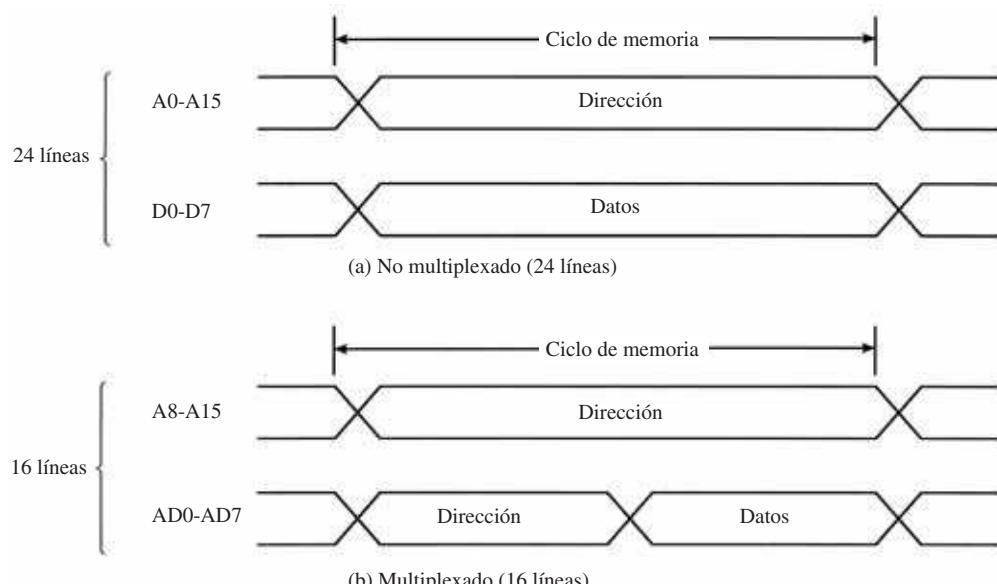


FIGURA 2–8

Multiplexaje de los buses de direcciones (byte inferior) y de datos

Antes de hablar sobre los detalles específicos del multiplexado de los buses de direcciones y de datos, presentaremos la idea general en la figura 2-8. Un arreglo no multiplexado utiliza 16 líneas de dirección dedicadas y ocho líneas de datos dedicadas, para un total de 24 terminales. El arreglo multiplexado combina ocho líneas para el bus de datos y el byte inferior del bus de direcciones, con ocho líneas más para el byte superior del bus de direcciones —un total de 16 terminales—. El ahorro en el número de terminales utilizadas permite que se puedan ofrecer otras funciones en un DIP (encapsulado dual en línea) de 40 terminales.

El arreglo multiplexado trabaja de la siguiente manera: el puerto 0 provee el byte inferior del bus de direcciones durante la primera mitad de cada ciclo de memoria y se fija mediante la señal ALE. Un latch 74HC373 (o su equivalente) almacena y mantiene estable el byte inferior de la dirección durante todo un ciclo de memoria. El puerto 0 se utiliza como bus de datos durante la segunda mitad del ciclo de memoria, y los datos se leen o escriben dependiendo de la operación realizada.

2.7.1 Acceso a la memoria externa para código

La memoria externa para código es memoria de sólo lectura y se habilita mediante la señal de ($\overline{\text{PSEN}}$). Los puertos 0 y 2 no están disponibles para su uso como puertos de E/S de propósito general cuando se utiliza una memoria EPROM externa. La figura 2-9 muestra las conexiones del hardware para una memoria EPROM externa.

Un ciclo de máquina del 8051 consta de 12 períodos de oscilación. El ciclo de máquina dura 1 μs si el oscilador incorporado en el chip está controlado por un cristal de 12 MHz. Durante un ciclo de máquina ordinario, la señal ALE envía pulsos dos veces y se leen 2 bytes de la memoria para programa. (El segundo byte se desecha si la instrucción en curso es de 1 byte). La figura 2-10 muestra la sincronización para esta operación, conocida como búsqueda de código de operación (fetch).

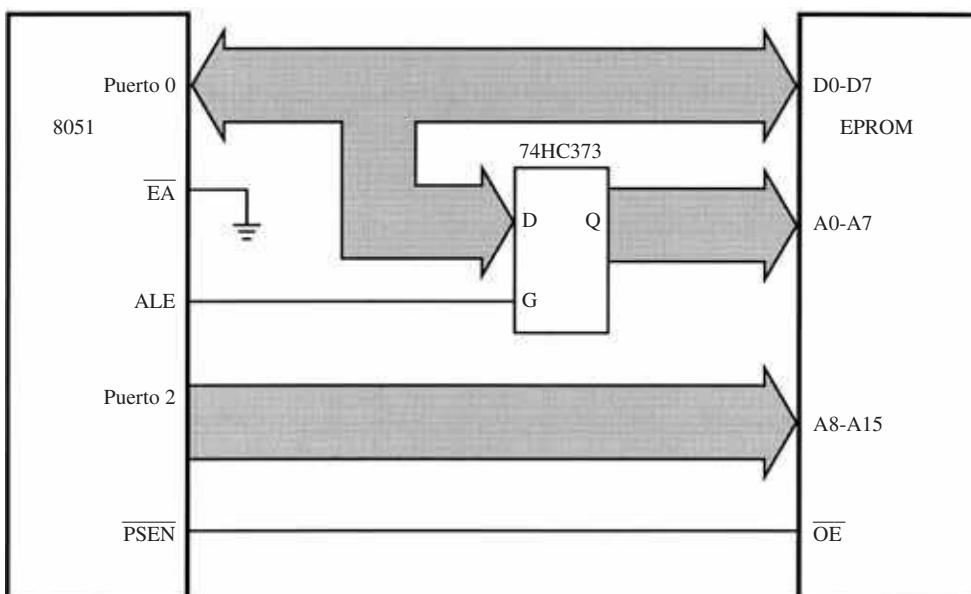
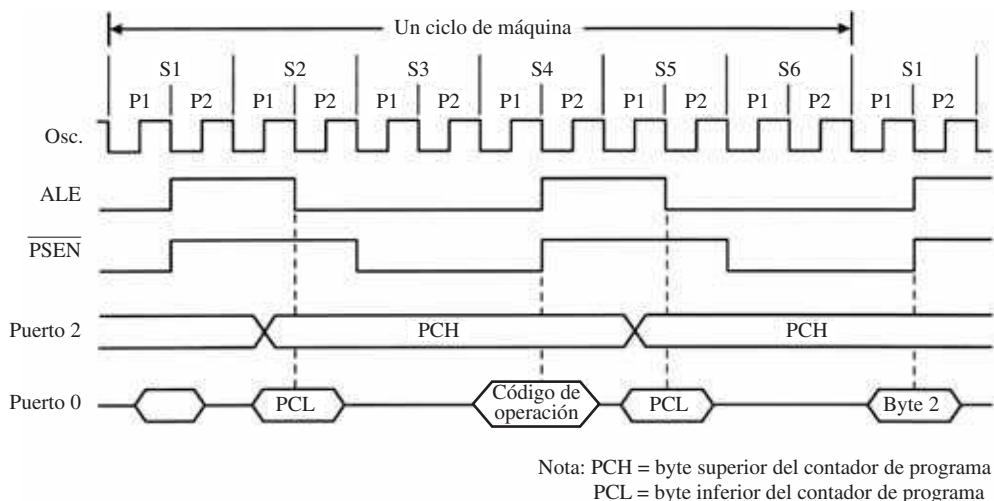


FIGURA 2-9

Acceso a la memoria externa para código

**FIGURA 2–10**

Sincronización de lectura de la memoria externa para código

2.7.2 Acceso a la memoria externa

La memoria externa para datos es memoria de lectura/escritura que se habilita mediante las señales RD y WR—las cuales son funciones alternas de las terminales P3.7 y P3.6. El único acceso a la memoria externa para datos es mediante la instrucción MOVX, utilizando ya sea el apuntador de datos de 16 bits (DPTR), el registro R0, o el registro R1 como registros de dirección.

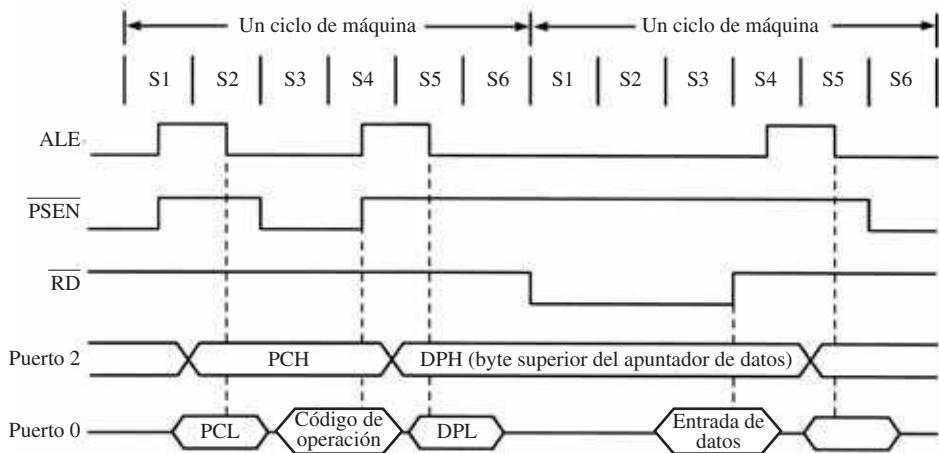
Es posible que varias memorias RAM puedan tener una interfaz con el 8051 en la misma forma que una memoria EPROM, a excepción de que la línea RD se conecta a la línea de habilitación de salida (\overline{OE}) de la memoria RAM y la señal WR se conecta a la línea de escritura (\overline{W}) de la RAM. Las conexiones del bus de direcciones y del bus de datos son las mismas que se utilizan para conectar una memoria EPROM. Se pueden conectar hasta 64K bytes de memoria RAM externa para datos al 8051 mediante el uso de los puertos 0 y 2, como se mencionó antes.

La figura 2-11 muestra un diagrama de tiempos de una operación de lectura en la memoria externa para datos para la instrucción MOVX A, @DPTR. Observe que se omiten un pulso de la señal ALE y un pulso de la señal (\overline{PSEN}) y en su lugar aparece un pulso en la línea RD para habilitar la memoria RAM.²

El diagrama de tiempos de un ciclo de escritura (MOVX @DPTR,A) es bastante similar, con la excepción de que la línea WR envía pulsos a bajo nivel y los datos utilizan el puerto 0 como salida. (La línea RD se mantiene a nivel alto).

Los sistemas que cuentan con el mínimo de componentes liberan al puerto 2 de su función alterna (suministrar el byte superior del bus de direcciones), ya que estos sistemas no utilizan memoria externa para código y sólo una pequeña cantidad de memoria externa para datos. Mediante direcciones de 8 bits se puede acceder a la memoria externa de datos en configuraciones pequeñas

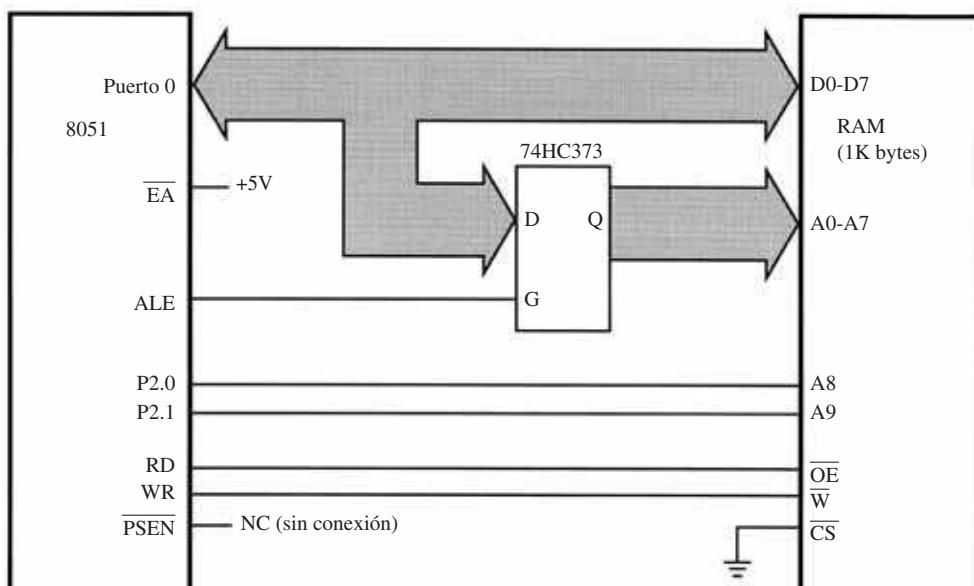
²La señal ALE envía pulsos de manera consistente a una frecuencia equivalente a 1/6 de la frecuencia del cristal oscilador si nunca se utilizan instrucciones MOVX (y memoria RAM externa).

**FIGURA 2-11**

Sincronización para la instrucción MOVX

de memoria organizadas en páginas. Algunos bits del puerto 2 (o de algún otro puerto) pueden seleccionar una página si se utiliza más de una página de 256 bytes de memoria RAM. Por ejemplo, una memoria RAM de 1K byte (es decir, cuatro páginas de 256 bytes) puede integrarse con el 8051 como se muestra en la figura 2-12.

Los bits 0 y 1 del puerto 2 deben inicializarse para seleccionar una página, y entonces se utiliza una instrucción MOVX para leer o escribir datos dentro de dicha página. Por ejemplo, si

**FIGURA 2-12**

Interfaz para la memoria RAM de 1K

suponemos que P2.0 = P2.1 = 0, podemos utilizar las siguientes instrucciones para leer el contenido de la dirección 0050H de la memoria RAM externa y colocarlo en el acumulador:

```
MOV R0, #50H
MOVX A, @R0
```

Los dos bits de selección de página deben ser puestos en 1 para poder leer la última dirección, 03FFH, en esta memoria RAM. Podemos utilizar la siguiente secuencia de instrucciones:

```
SETB P2.0
SETB P2.1
MOV R0, #0FFH
MOVX A, @R0
```

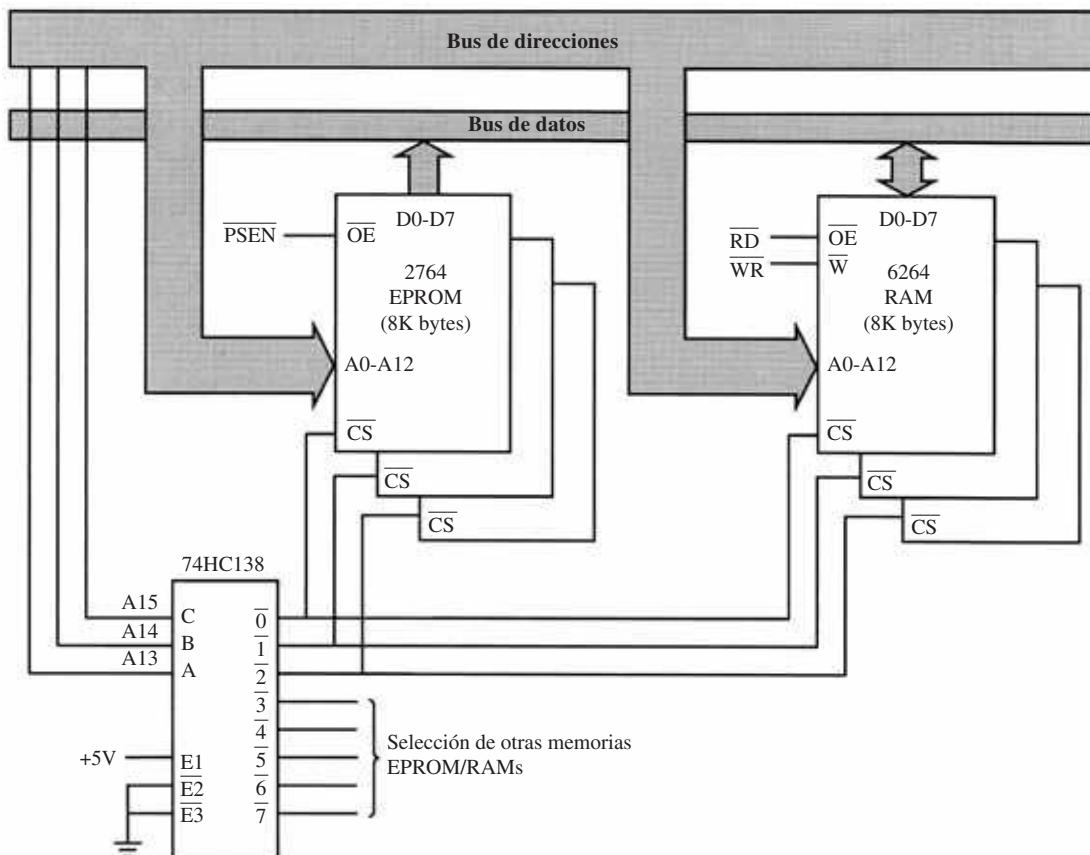
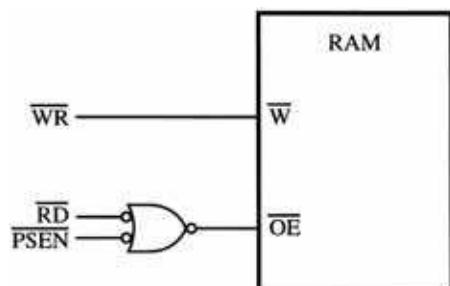


FIGURA 2-13
Decodificación de dirección

FIGURA 2-14

Traslape del espacio externo para código y para datos



Una característica de este diseño es que los bits 2 y 7 del puerto 2 no se necesitan en la forma de bits de dirección, como sería si el DPTR fuera el registro de dirección. Las terminales P2.2 a P2.7 están disponibles para propósitos de E/S.

2.7.3 Decodificación de direcciones

La decodificación de direcciones es necesaria si el 8051 tiene una interfaz para varias memorias EPROM y/o RAM. La decodificación es parecida a la que requieren la mayoría de los microprocesadores. Por ejemplo, si se utilizan memorias EPROM o RAM de 8K bytes, el bus de direcciones tendrá que decodificarse para poder seleccionar circuitos integrados de memoria sobre límites de 8K: 0000H-FFFFH, 2000H-3FFFH, etcétera.

Por lo general, se utiliza un CI decodificador tal como el 74HC138, con sus propias salidas conectadas a las entradas de selección de chip (\overline{CS}) en los CIs de memoria. Esto se ilustra en la figura 2-13 para un sistema con varias memorias EPROM 2764 de 8K y RAM 6264 de 8K. Recuerde que el 8051 puede aceptar hasta 64K de *cada* memoria EPROM y RAM, debido a que existen líneas separadas de habilitación (\overline{PSEN} para la memoria del código; \overline{RD} y \overline{WR} para la memoria de los datos).

2.7.4 Traslape de los espacios externos para código y para datos

El desarrollo de software para el 8051 presenta una situación complicada, debido a que la memoria para código es de sólo lectura. ¿Cómo podemos “escribir” software en cierto sistema para depurarlo cuando sólo es posible ejecutarlo desde el espacio de código de “sólo lectura”? Un truco muy común para resolver este problema es el de traslapar los espacios de memoria externos para código y para datos. Dado que \overline{PSEN} se utiliza para leer memoria de código y \overline{RD} para leer memoria de datos, una memoria RAM puede ocupar espacio de memoria tanto para código como para datos mediante la conexión de su línea \overline{OE} a la compuerta AND lógica (NOR de entrada negativa) de \overline{PSEN} y \overline{RD} . La figura 2-14 muestra el circuito que permite escribir en el CI de RAM como memoria para datos y leer ya sea como memoria de datos *o* de código. Por ende, un programa puede cargarse en la RAM (si se escribe en ella como memoria de datos) y ejecutarse (si se accede a la RAM como memoria de código).

2.8 MEJORAS DEL 8032/8052

Los circuitos integrados 8032/8052 (y las versiones de CMOS y/o EPROM) ofrecen dos mejoras para los CI 8031/8051. En primer lugar, existen 128 bytes adicionales de RAM incorporada al chip en el rango de direcciones 80H a FFH. Para no crear ningún conflicto con los SFR

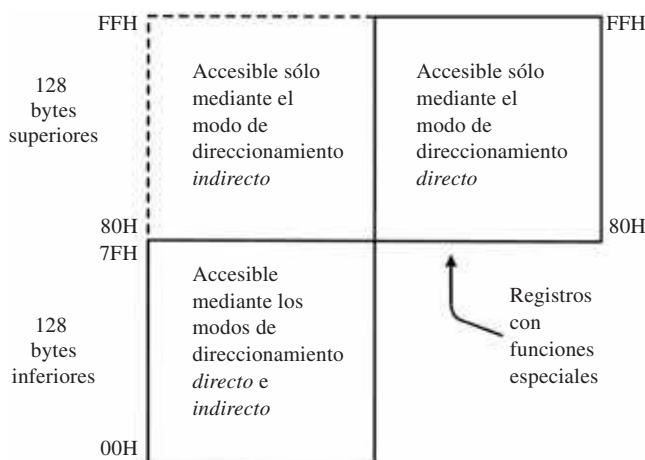


FIGURA 2-15
Espacios de memoria del 8032/8052

(que utilizan las mismas direcciones), sólo se puede acceder a ese 1/8K adicional de memoria RAM mediante el modo de direccionamiento indirecto. Una instrucción tal como

```
MOV A, 0FOH
```

transfiere el contenido del registro B al acumulador en todos los CI de la familia MCS-51TM. La secuencia de instrucciones

```
MOV R0, #0FOH
MOV A, @R0
```

lee el contenido de la dirección interna F0H y lo transfiere al acumulador en los circuitos integrados 8032/8052. Note que esta dirección no está definida en los CI 8031/8051. La figura 2-15 presenta un resumen de la organización de la memoria interna en los circuitos integrados 8032/8052.

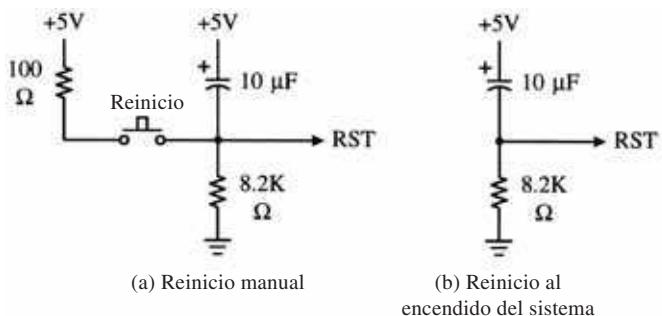
La segunda mejora del 8032/8052 es un temporizador adicional de 16 bits, denominado Temporizador 2, el cual se programa a través de cinco registros adicionales de función especial, sintetizados en la tabla 2-5. Consulte el capítulo 4 para obtener más detalles.

TABLA 2-5
Registros del Temporizador 2

Registro	Dirección	Descripción	Direccionable por bits
T2CON	C8H	Control	Sí
RCAP2L	CAH	Byte inferior de Captura	No
RCAP2H	CBH	Byte superior de Captura	No
TL2	CCH	Byte inferior del Temporizador 2	No
TH2	CDH	Byte inferior del Temporizador 2	No

FIGURA 2-16

Dos circuitos para el reinicio del sistema.
 (a) Reinicio manual
 (b) Reinicio al encendido del sistema



2.9 OPERACIÓN DE REINICIO

Para reiniciar el 8051, se mantiene la señal RST en nivel alto durante por lo menos dos ciclos de máquina, y entonces se regresa al nivel bajo. La señal RST puede ser puesto en 1 manualmente empleando un interruptor o, al momento de encender el sistema, mediante una red RC (resistencias-condensadores). La figura 2-16 muestra dos circuitos que implementan un reinicio del sistema.

La tabla 2-6 presenta un resumen del estado de todos los registros del 8051 después de un reinicio del sistema. El contador de programa es quizás el más importante de estos registros, y se carga con el valor 0000H. Cuando la señal RST regresa al nivel bajo, la ejecución de un programa siempre comienza en la primera ubicación de la memoria de código: la dirección 0000H. El contenido de la RAM incorporada en el chip no se ve afectado por una operación de reinicio.

TABLA 2-6

Valores de los registros después de un reinicio del sistema

Registro(s)	Contenido
Contador del programa	0000H
Acumulador	00H
Registro B	00H
PSW	00H
SP	07H
DPTR	0000H
Puertos 0-3	FFH
IP (8031/8051)	XXX00000B
IP (8032/8052)	XX000000B
IE (8031/8051)	0XX00000B
E (8032/8052)	0X000000B
Registros de temporizador	00H
SCON	00H
SBUF	00H
PCON (HMOS)	0XXXXXXXB
PCON (CMOS)	0XXX0000B

RESUMEN

En este capítulo se presentó un resumen de la arquitectura del hardware del 8051. Antes de poder desarrollar aplicaciones útiles, debemos comprender el conjunto de instrucciones del 8051. El siguiente capítulo se enfoca en las instrucciones y los modos de direccionamiento del 8051. En este capítulo tratamos muy poco sobre el temporizador, el puerto serial, y los SFR de interrupciones ya que más adelante veremos capítulos completos dedicados a cada uno de estos elementos, y entonces los examinaremos con todo detalle.

PROBLEMAS

- 2.1** Nombre cuatro fabricantes del microcontrolador 8051, además de Intel.
- 2.2** ¿Qué dispositivo de la familia MCS-51™ podría utilizarse en un producto a fabricar en grandes cantidades con un programa extenso incorporado en el chip?
- 2.3** ¿Qué instrucción podría utilizarse para establecer en 1 el bit menos significativo en la dirección de byte 25H?
- 2.4** ¿Cuál es la secuencia de instrucciones que podría utilizarse para realizar la operación lógica OR de los bits ubicados en las direcciones de bit 00H y 01H y almacenar el resultado en la dirección de bit 02H?
- 2.5** ¿Qué secuencia de instrucciones podría utilizarse para leer el bit 0 del puerto 0 y escribir el estado del bit leído en el bit 0 del puerto 3?
- 2.6** Muestre una secuencia de instrucciones útil para leer los bits 0 y 1 del puerto 0 y escribir una condición de estado en el bit 0 del puerto 3 como se indica a continuación: si los dos bits leídos son 1, escriba un 1 en el bit de estado de salida; en caso contrario escriba un 0.
- 2.7** Muestre una secuencia de instrucciones para leer los bits 0 y 1 del puerto 0 y escribir una condición de estado en el bit 0 del puerto 3 como se indica a continuación: si cualquiera de los bits es 1, pero no ambos, escriba un 1 en el bit de estado de salida; en caso contrario escriba un 0.
- 2.8** Muestre una secuencia de instrucciones para leer los bits 0 y 1 del puerto 0 y escribir una condición de estado en el bit 0 del puerto 3 como se indica a continuación: si cualquiera de los bits leídos es 1, escriba un 0 en el bit de estado de salida; en caso contrario escriba un 1.
- 2.9** Muestre las operaciones necesarias para resolver los tres problemas anteriores utilizando compuertas lógicas.
- 2.10** ¿Qué direcciones de bit se establecerán en 1 como resultado de las siguientes instrucciones?
 - a. `MOV 26H, #26H`
 - b. `MOV R0, #26H`
`MOV @R0, #7AH`
 - c. `MOV A, #13H`
 - d. `MOV 30H, #55H`
`XRL 30H, #0AAH`
 - e. `SETB P1.1`
 - f. `MOV P3, #0CH`
- 2.11** ¿Qué instrucción de 1 byte causa el mismo efecto que la siguiente instrucción de 2 bytes?
`MOV 0E0H, #55H`
- 2.12** Muestre una secuencia de instrucciones para almacenar el valor 0ABH en la dirección 9A00H de RAM externa.

- 2.13** ¿Cuántos registros con funciones especiales están definidos en el 8052?
- 2.14** ¿Cuál es el valor del apuntador de pila del 8051 justo después de un reinicio del sistema?
- 2.15** ¿Qué instrucción podría utilizarse para inicializar el apuntador de pila y crear una pila de 64 bytes en la parte superior de la memoria interna para (a) el 8031 o (b) el 8032?
- 2.16** ¿Qué instrucción podría utilizarse para inicializar el apuntador de pila y crear una pila de 32 bytes en la parte superior de la memoria interna para (a) el 8051 o (b) el 8052?
- 2.17** Cierta subrutina hace uso extensivo de los registros R0-R7. Muestre cómo la subrutina podría, al momento de su inicio, cambiar el banco de registros activo al banco 3 y restaurar el banco de registros previamente activo al momento en que terminara la subrutina.
- 2.18** ¿Cuál es el banco de registros que queda activo después de ejecutar cada una de las siguientes instrucciones?
- MOV PSW, #0FDH
 - MOV PSW, #18H
 - MOV PSW, #08H
- 2.19** ¿Cuál es el banco de registros que queda activo después de ejecutar cada una de las siguientes instrucciones?
- MOV PSW, #0C8H
 - MOV PSW, #50H
 - MOV PSW, #10H
- 2.20** El 80C31BH-1 puede operar mediante el uso de un cristal de 16 MHz conectado a sus entradas XTAL1 y XTAL2. Si no se utilizan instrucciones MOVX, ¿cuál es la frecuencia de la señal en ALE?
- 2.21** ¿Cuál es la duración del ciclo de máquina cuando se opera un 8051 mediante un cristal de 4 MHz?
- 2.22** ¿Cuál es la frecuencia de la forma de onda en la señal ALE si se opera un 8051 mediante un cristal de 10 MHz? Suponga que el software no está accediendo a la memoria RAM externa.
- 2.23** ¿Cuál es el ciclo de trabajo de ALE? Suponga que el software no está accediendo a la memoria RAM externa. (Nota: El ciclo de trabajo se define como la proporción de tiempo durante el cual una forma de onda pulsante se encuentra en nivel alto en relación al periodo).
- 2.24** La sección 2.9 menciona que el 8051 se reinicia si la terminal de RST se mantiene en nivel alto durante al menos dos ciclos de máquina. (Nota: Tal como se menciona en la sección Características DC del 8051 en el apéndice E, un nivel “alto” en la señal RST tiene un valor mínimo de 2.5 voltios).
- ¿Durante cuánto tiempo la señal RST debe mantenerse en nivel alto para lograr un reinicio del sistema, si se opera un 8051 con un cristal de 8 MHz?
 - La figura 2-16a muestra un circuito RC para reinicio manual. Mientras el botón para reinicio se mantiene presionado, RST = 5 voltios y el sistema se mantiene en estado de reinicio. ¿Qué tanto tiempo permanecerá en estado de reinicio el 8051 después de soltar el botón de reinicio?
- 2.25** ¿Cuántas unidades de carga del tipo Schottky de bajo consumo de energía puede controlar la línea de puerto P1.7 en la terminal 8?
- 2.26** Nombre las señales de control de bus del 8051 que se utilizan para seleccionar memorias EPROM y RAM externas.
- 2.27** ¿Cuál es la dirección del bit más significativo localizado en la dirección de byte 25H de la memoria interna de datos del 8051?
- 2.28** ¿Cuál es la dirección del bit 3 en la dirección de byte 2FH de la memoria interna de datos del 8051?

- 2.29** Algunas de las ubicaciones direccionables por bit en la memoria de datos incorporada al chip del 8031 corresponden también a señales en el circuito integrado 8031. ¿Cuáles son estas ubicaciones y señales? ¿Cuál es el número de estas terminales y cuáles son sus direcciones de bit?
- 2.30** Identifique la posición de bit y la dirección de byte para cada una de las siguientes instrucciones SETB.
- SETB 37H
 - SETB 77H
 - SETB 0F7H
- 2.31** Identifique la posición de bit y la dirección de byte para cada una de las siguientes instrucciones SETB.
- SETB 0A8H
 - SETB 84H
 - SETB 63H
- 2.32** ¿Cuál es la instrucción que establece en 1 el bit menos significativo del acumulador sin afectar los otros siete bits?
- 2.33** ¿Cuál es el estado del bit P en la PSW después de ejecutar cada una de las siguientes instrucciones?
- MOV A, #55H
 - MOV A, #0F8H
 - MOV A, #0FFH
- 2.34** ¿Cuál es el estado del bit P en la PSW después de ejecutar cada una de las siguientes instrucciones?
- CLR A
 - MOV A, #03H
 - MOV A, #0ABH
- 2.35** ¿Qué secuencia de instrucciones podría utilizarse para copiar el contenido de R7 en la ubicación de RAM externa 100H?
- 2.36** Muestre una secuencia de instrucciones para leer la dirección de memoria RAM externa 08F5H y colocar el byte leído en el acumulador B.
- 2.37** Asuma que la primera instrucción que se ejecuta después de un reinicio del sistema es la llamada a una subrutina. ¿En qué direcciones de memoria RAM interna se almacena el contador del programa antes de que la ejecución se bifurque hacia la subrutina?
- 2.38** Considere la instrucción MOV SP,#08FH. (a) ¿Qué efecto tiene esta instrucción si se ejecuta en el 8032? (b) ¿Qué efecto tiene si se ejecuta en el 8031?
- 2.39** El apuntador de pila no tiene que inicializarse si en el 8031 un programa está diseñado para utilizar sólo el banco de registros 0. Es imperativo, sin embargo, que el apuntador de pila se inicialice de manera explícita si un programa localizado en el 8031 está diseñado para utilizar los cuatro bancos de registros. ¿Por qué?
- 2.40** ¿Cuál es la diferencia entre los modos de suspensión y de apagado del 8051?
- 2.41** ¿Qué instrucción podría utilizarse para forzar al 8051 a que entre en el modo de apagado?
- 2.42** Muestre cómo podrían integrarse dos RAM estáticas de 32 Kbytes con el 8051 de manera que ocuparan todo el espacio de datos externo de 64K.
- 2.43** Si tenemos lo siguiente:

A = 55H

B = 11H

Ubicación de memoria RAM interna 30H = 33H
SP = 00H

- ¿Cuál será el contenido de cada uno luego de un reinicio?
- 2.44** Explique el término “expansión de E/S”.
- 2.45** ¿Cuáles son las diferencias entre una E/S por asignación de memoria y un puerto de E/S? Explique su respuesta.
- 2.46** Un microcontrolador 8051 debe ejecutar programas desde una memoria ROM externa. Especifique las señales de control y sus valores lógicos para habilitar el acceso a la memoria ROM externa. Especifique también el registro (y su valor inicial después de un reinicio) que identificará la primera instrucción a buscar (fetch).
- 2.47** Se dice que el 8051 cuenta con 128 bytes de memoria interna para datos. Pero si nos referimos al mapa de memoria presentado en la figura 2-7, la memoria para datos incorporada en el chip del 8051 tiene un rango de 00H a FFH, lo cual produce 256 ubicaciones. ¿Por qué pasa esto?
- 2.48** Explique la diferencia entre la pila y el apuntador de pila (SP) por medio de un ejemplo.

<http://www.ingeelectronico.blogspot.com>

Resumen del conjunto de instrucciones

3.1 INTRODUCCIÓN

Así como una oración está formada por palabras, los programas de cómputo se forman a partir de instrucciones. Podemos crear programas rápidos, eficientes, y hasta elegantes si los construimos utilizando secuencias de instrucciones lógicas y bien planeadas. El conjunto de instrucciones es único para cada familia de computadoras, y es un repertorio de operaciones básicas tales como “sumar”, “transferir”, o “saltar”. En este capítulo se introduce el conjunto de instrucciones de la familia MCS-51™ mediante la exploración de los modos de direccionamiento y de ejemplos que resultan de situaciones típicas de la programación. El apéndice A contiene un diagrama que sintetiza todas las instrucciones del 8051. El apéndice C proporciona descripciones detalladas de cada instrucción. Recomendamos consultar ambos apéndices como referencia.

En este capítulo no trataremos técnicas de programación ni utilizaremos el ensamblador para convertir programas codificados en lenguaje ensamblador (nemáticos, etiquetas, etc.) en programas de lenguaje de máquina (códigos binarios). En el capítulo 7 presentaremos estos temas.

El conjunto de instrucciones de la familia MCS-51™ está optimizado para aplicaciones de control de 8 bits. Cuenta con una variedad de modos de direccionamiento rápidos y compactos para acceder a la memoria RAM interna de manera que se faciliten las operaciones sobre estructuras de datos pequeñas. El conjunto de instrucciones ofrece un soporte extenso para variables de 1 bit y permite la manipulación directa de bits en sistemas de control y de lógica que requieren de procesamiento booleano.

Las instrucciones del 8051 tienen códigos de operación de 8 bits, como es común en los procesadores de 8 bits. Esta estructura provee $2^8 = 256$ instrucciones posibles. De éstas, 255 están implementadas y una no está definida. Algunas instrucciones, además del código de operación, tienen uno o dos bytes adicionales para datos o direcciones. En total, existen 139 instrucciones de 1 byte, 92 instrucciones de 2 bytes, y 24 instrucciones de 3 bytes. El *Mapa de códigos de operación* incluido en el apéndice B muestra, para cada una de las instrucciones, su nemático, el número de bytes contenidos en la instrucción, y el número de ciclos de máquina necesarios para ejecutarla.

3.2 MODOS DE DIRECCIONAMIENTO

Cuando las instrucciones operan sobre los datos, surge la siguiente pregunta: “¿En dónde están los datos?”; cuya respuesta encontraremos en los “modos de direccionamiento” del 8051. Existen varios modos de direccionamiento posibles y varias respuestas posibles a dicha pregunta, tales como “en el byte 2 de la instrucción”, “en el registro R4”, “en la dirección directa 35H”, o tal vez “en la memoria externa para datos en la dirección contenida en el apuntador de datos”.

Los modos de direccionamiento son parte integral del conjunto de instrucciones de cada computadora, ya que nos permiten especificar la fuente o el destino de los datos de varias maneras, dependiendo de la situación de programación. En este apartado, examinaremos los modos de direccionamiento del 8051 y presentaremos ejemplos para cada uno. Existen ocho modos disponibles:

- Registro
- Directo
- Indirecto
- Inmediato
- Relativo
- Absoluto
- Largo
- Indexado

3.2.1 Direccionamiento por registro

El programador del 8051 tiene acceso a ocho “registros funcionales”, enumerados del R1 al R7. Las instrucciones que utilizan direccionamiento por registro se codifican utilizando los tres bits menos significativos del código de operación de la instrucción para especificar un registro concreto. Por lo tanto, podemos combinar un código de función y la dirección de un operando para formar una instrucción corta (de 1 byte). (Consulte la figura 3-1a).

El lenguaje ensamblador del 8051 indica el direccionamiento por registro mediante el símbolo *Rn*, donde *n* varía de 0 a 7. Por ejemplo, para añadir el contenido del registro 7 al acumulador se utiliza la siguiente instrucción:

```
ADD A, R7
```

y el código de operación es 00101111B. Los cinco bits superiores, 00101, indican la instrucción y los tres bits inferiores, 111, señalan el registro. Consulte el apéndice C para confirmar que éste es el código de operación correcto.

EJEMPLO 3.1 ¿Cuál es el código de operación para la siguiente instrucción? ¿Cuál es la función de esta instrucción?

```
MOV A, R7
```

Solución

EFH. Esta instrucción transfiere el contenido de 8 bits del registro 7 (en el banco de registros activo) al acumulador.

Análisis

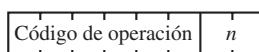
El apéndice C presenta una lista de todas las instrucciones del 8051 en orden alfabético por nemónico. La forma general de las instrucciones para transferir bytes es:

```
MOV byte_destino,byte_origen
```

Existen 15 variaciones identificadas. Este ejemplo se enfoca en la instrucción MOV A,Rn. El código de operación binario aparece como 11101rrr. Los tres bits de orden inferior identifican el registro fuente, el cual es R7 en este ejemplo. Cuando sustituimos el valor “111” por “rrr”, el resultado es un código de operación 11101111B = EFH.

Existen cuatro “bancos” de registros funcionales, pero sólo uno de ellos está activo en un momento dado. Los bancos de registros ocupan el espacio físico de los primeros 32 bytes de la memoria RAM para datos incorporada en el chip (el rango de direcciones 00H-1FH) y los bits 4 y 3 de la PSW determinan el banco activo. Un reinicio por hardware habilita el banco 0, pero podemos seleccionar un banco diferente si modificamos los bits 4 y 3 de la PSW en la forma apropiada. Por ejemplo, la instrucción

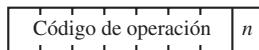
```
MOV PSW, #00011000B
```



(a) Direcciónamiento por registro (por ejemplo, ADD A,R5)



(b) Direcciónamiento directo (por ejemplo, ADD A,55H)



(c) Direcciónamiento indirecto (por ejemplo, ADD A, @R0)



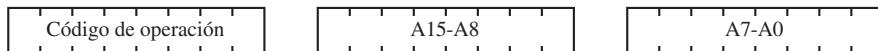
(d) Direcciónamiento inmediato (por ejemplo, ADD A,#44H)



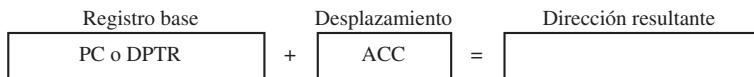
(e) Direcciónamiento relativo (por ejemplo, SJMP ADELANTE)



(f) Direcciónamiento absoluto (por ejemplo, AJMP ATRÁS)



(g) Direcciónamiento largo (por ejemplo, LJMP ADELANTE_LEJOS)



(h) Direcciónamiento indexado (por ejemplo, MOVC A,@A+PC)

FIGURA 3-1

Modos de direcciónamiento del 8051. (a) Direcciónamiento por registro. (b) Direcciónamiento directo. (c) Direcciónamiento indirecto. (d) Direcciónamiento inmediato. (e) Direcciónamiento relativo. (f) Direcciónamiento absoluto. (g) Direcciónamiento largo. (h) Direcciónamiento indexado

activa el banco de registros 3 al establecer en 1 los bits de selección de banco de registros (RS1 y RS0) en las posiciones de bit 4 y 3 de la PSW.

Ciertas instrucciones son específicas para un registro dado, como el acumulador, el apuntador de datos, etc., así que los bits de dirección no son necesarios. En el mismo código de operación se indica el registro. Estas instrucciones “específicas para el registro” se refieren al acumulador como “A”, al apuntador de datos como “DPTR”, al contador de programa como “PC”, a la bandera de acarreo como “C”, y al par de registros acumulador-B como “AB”. Por ejemplo,

```
INC DPTR
```

es una instrucción de 1 byte que incrementa el apuntador de datos de 16 bits. Consulte el apéndice C para determinar el código de operación de esta instrucción.

EJEMPLO 3.2

(a) ¿Cuál es el código de operación de la siguiente instrucción? (b) ¿Cuál es la función de esta instrucción?

```
MUL AB
```

Solución

(a) A4H. (b) Esta instrucción multiplica el valor sin signo de 8 bits que contiene el acumulador por el valor sin signo de 8 bits que contiene el registro B. El producto de 16 bits queda en el acumulador (byte inferior) y el registro B (byte superior).

3.2.2 Direccionamiento directo

El direccionamiento directo puede acceder a cualquier variable incorporada en el chip o a cualquier registro del hardware. Se añade un byte adicional al código de operación para especificar la localidad a utilizar. (Consulte la figura 3-1b).

Se selecciona uno de dos espacios de memoria incorporados en el chip, dependiendo del bit de mayor orden de la dirección directa. Cuando el bit 7 = 0, la dirección directa se encuentra en el rango de 0 a 127 (00H-7FH) y se hace referencia a las 128 ubicaciones de menor orden en la memoria RAM incorporada en el chip. Sin embargo, todos los puertos de E/S y los registros con funciones especiales, de control o de estado, tienen asignadas direcciones en el rango de 128 a 255 (80H-FFH). Cuando el byte de dirección directa está dentro de estos límites (bit 7 = 1), se accede al registro de funciones especiales correspondiente. Por ejemplo, los puertos 0 y 1 tienen asignadas las direcciones directas 80H y 90H, respectivamente. La mayoría de las veces no es necesario conocer la dirección de estos registros; el ensamblador acepta y entiende las abreviaturas de los nemáticos (“P0” para el puerto 0, “TMOD” para el registro de modo del temporizador, etc.). Algunos ensambladores, como el ASM51 de Intel, incluyen de manera automática la definición de símbolos predefinidos. Otros ensambladores pueden utilizar un archivo fuente separado que contiene las definiciones. Para exemplificar el direccionamiento directo, la instrucción

```
MOV P1,A
```

transfiere el contenido del acumulador al puerto 1. La dirección directa del puerto 1 (90H) está determinada por el ensamblador y se añade como el byte 2 de la instrucción. La fuente de los datos (el acumulador) se especifica de manera implícita en el código de operación. Si utilizamos el apéndice C como referencia, la codificación completa de esta instrucción es

```
10001001 – primer byte (código de operación)  
10010000 – segundo byte (dirección de P1)
```

EJEMPLO 3.3 En lenguaje de máquina, ¿cuáles son los bytes para la siguiente instrucción?

MOV SCON, #55H

Solución

75H, 98H, 55H

Análisis

La forma general de esta instrucción es

MOV directo, #dato

La instrucción es de tres bytes, tal como se menciona en el apéndice C. El primer byte es el código de operación, 75H. El segundo byte es la dirección directa del registro de función especial SCON, 98H (consulte la figura 2-6 o el apéndice D). El tercer byte es el dato inmediato, 55H.

3.2.3 Direcciónamiento indirecto

¿Cómo podemos identificar una variable cuando su dirección se determina, calcula o modifica mientras el programa se está ejecutando? Esta situación emerge al manipular ubicaciones secuenciales en memoria, valores indexados dentro de tablas en memoria RAM, números de precisión múltiple, o cadenas de caracteres. Los direccionamientos por registro o directo no pueden utilizarse, pues requieren que se conozcan las direcciones de operando durante el tiempo de ensamblaje.

En el 8051, la solución es el direcciónamiento indirecto. Los registros R0 y R1 pueden operar como registros “apuntadores”; su contenido indica la dirección de la memoria RAM interna en donde se escriben o leen los datos. El bit menos significativo del código de operación de la instrucción determina cuál es el registro (R0 o R1) que se utiliza como apuntador. (Consulte la figura 3-1c).

El direcciónamiento indirecto en el lenguaje ensamblador del 8051 se representa empleando el signo de arroba (@) como prefijo para R0 o R1. Por ejemplo, si el registro R1 contiene el valor 40H y la dirección de memoria interna 40H contiene el valor 55H, la instrucción

MOV A, @R1

transfiere el valor 55H al acumulador.

EJEMPLO 3.4 (a) ¿Cuál es el código de operación de la siguiente instrucción? (b) ¿Cuál es la función de la instrucción?

MOV A, @R0

Solución

(a) E6H. (b) Esta instrucción transfiere un byte de datos de la memoria RAM interna al acumulador. El registro R0 incluye la dirección que contiene los datos a transferir.

Análisis

La forma general de esta instrucción es

MOV A, @Ri

El código de operación binario, de acuerdo con el apéndice C, es 1110011i. El valor “0” sustituye al término “i” en el código de operación, ya que especificamos el registro indirecto utilizando el término “R0”. Por lo tanto, el código de operación es 11100110B = E6H.

El modo de direccionamiento indirecto resulta esencial cuando recorremos localidades consecutivas en la memoria. Por ejemplo, la siguiente secuencia de instrucciones borra las direcciones de RAM interna en el rango de 60H a 7FH:

```
CICLO:    MOV     R0, #60H
           MOV     @R0, #0
           INC     R0
           CJNE   R0, #80H, CICLO
           (continúa)
```

La primera instrucción inicializa el registro R0 con la dirección inicial del bloque de memoria; la segunda instrucción utiliza el direccionamiento indirecto para transferir el valor 00H hacia la ubicación a donde apunta R0; la tercera instrucción incrementa el apuntador (R0) a la siguiente dirección; y la última instrucción prueba el apuntador para ver si el programa llegó al fin del bloque de ejecución. La prueba utiliza el valor 80H, en lugar de 7FH, ya que el incremento ocurre después de la transferencia indirecta. Esto asegura que se escriba en la ubicación final (7FH) antes de terminar la ejecución.

3.2.4 Direccionamiento inmediato

Cuando el operando de origen es una constante en vez de una variable (es decir, que la instrucción utiliza un valor conocido en tiempo de ensamblaje), entonces la constante puede incorporarse a la instrucción como un byte de dato “inmediato”. El valor está contenido en un byte de instrucción adicional. (Consulte la figura 3-1d).

En el lenguaje ensamblador, los operandos inmediatos están precedidos del símbolo de número (#). El operando puede ser una constante numérica, una variable simbólica, o una expresión aritmética que utilice constantes, símbolos y operadores. El ensamblador interpreta el valor, sustituye los datos inmediatos y los incorpora a la instrucción. Por ejemplo, la instrucción

```
MOV A, #12
```

carga el valor 12 (0CH) en el acumulador. (Asumimos que la constante “12” está en notación decimal, ya que no va seguida de la letra “H” para representar números hexadecimales).

Todas las instrucciones que utilizan direccionamiento inmediato, a excepción de una, usan una constante de 8 bits para el dato inmediato. Se requiere de una constante de 16 bits cuando se inicializa el apuntador de datos. Por ejemplo,

```
MOV DPTR, #8000H
```

es una instrucción de 3 bytes que carga la constante de 16 bits 8000H al apuntador de datos.

EJEMPLO 3.5 En lenguaje de máquina, ¿cuáles son los bytes para esta instrucción, en hexadecimal y en binario?

```
ADD A, #15
```

Solución

En binario: 00100100B, 00001111B. Hexadecimal: 24H, 0FH.

Análisis

La forma general de esta instrucción es

```
ADD A, #datos
```

El código de operación, de acuerdo con el apéndice C, es 00100100B = 24H. El segundo byte de la instrucción representa los datos inmediatos. Esto se especifica en la instrucción como $15_{10} = 00001111\text{ B} = 0FH$.

3.2.5 Direcciónamiento relativo

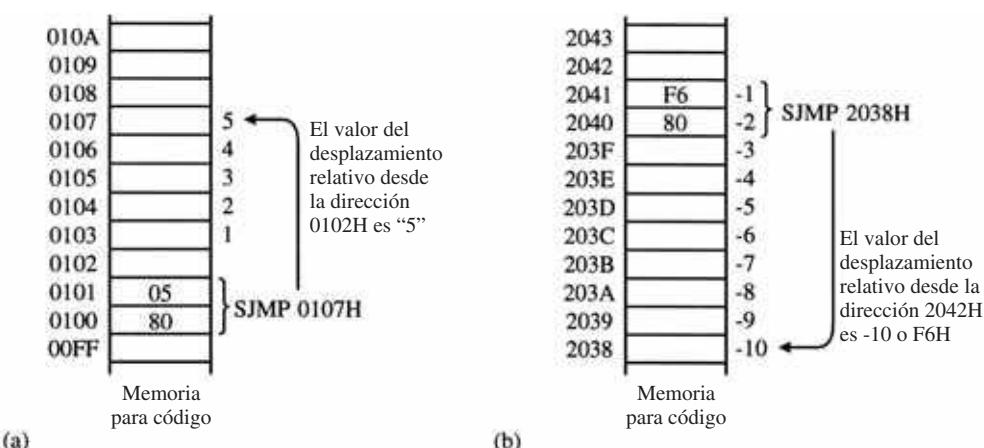
El direcciónamiento relativo sólo se utiliza con ciertas instrucciones de salto. Una dirección relativa (o desplazamiento) es un valor con signo de 8 bits que se agrega al contador de programa para construir la dirección de la siguiente instrucción a ejecutar. El rango de salto es de -128 a +127 ubicaciones, ya que se utiliza un desplazamiento con signo de 8 bits. El desplazamiento relativo se añade a la instrucción en forma de un byte adicional. (Consulte la figura 3-1e).

Antes de añadir el desplazamiento, el contador de programa se incrementa a la dirección que va después de la instrucción de salto; por lo tanto, la nueva dirección es relativa a la siguiente instrucción, *no* a la dirección de la instrucción de salto. (Consulte la figura 3-2).

Este detalle no es importante para el programador en la mayoría de los casos, debido a que por lo general el destino del salto se especifica mediante el uso de etiquetas y el ensamblador calcula el desplazamiento relativo en forma adecuada. Por ejemplo, si la etiqueta AQUI representa una instrucción en la ubicación 1040H, y la instrucción

```
SJMP AQUI
```

está ubicada en las direcciones 1000H y 1001H de la memoria, el ensamblador asignará un desplazamiento relativo con el valor 3EH como el byte 2 de la instrucción ($1002H + 3EH = 1040H$).

**FIGURA 3-2**

Cálculo del desplazamiento para direcciónamiento relativo. (a) Salto corto hacia delante en la memoria.
(b) Salto corto hacia atrás en la memoria

El direccionamiento relativo ofrece la ventaja de proveer código que es independiente de su posición (ya que no se utilizan direcciones “absolutas”), pero también existe la desventaja de que el destino de los saltos tiene un rango limitado.

EJEMPLO La instrucción

3.6

`SJMP 9030H`

está ubicada en las direcciones 9000H y 9001H de la memoria. En lenguaje de máquina, ¿cuáles son los bytes para esta instrucción?

Solución

80H, 2EH

Análisis

La forma general de la instrucción es

`SJMP dirección_relativa`

Si consultamos el apéndice C veremos que la instrucción es de dos bytes, comenzando con el código de operación 10000000B = 80H. El segundo byte es un valor con signo de 8 bits que representa el desplazamiento relativo. El desplazamiento tiene signo positivo, ya que para este ejemplo estaremos brincando “hacia delante” en la memoria. La figura 3-2 muestra que la dirección de origen es la dirección “después” de la instrucción de salto. El desplazamiento se añade a este valor para obtener la dirección de destino para el salto. Podemos calcular el desplazamiento en forma aritmética, en lugar de realizar un dibujo y descontar el desplazamiento (como en la figura 3-2):

`dirección_origen + desplazamiento = dirección_destino`

o bien

`desplazamiento = dirección_destino - dirección_origen`

En este ejemplo, la dirección de origen es 9002H (la dirección que va *después* de la instrucción de salto) y la dirección de destino es 9030H, por lo tanto

`desplazamiento = 9030H - 9002H = 2EH`

EJEMPLO
3.7

Una instrucción SJMP con representación en lenguaje de máquina de 80H F6H se encuentra en las direcciones 0802H y 0803H de la memoria. ¿A qué dirección saltará la ejecución?

Solución

07FAH

Análisis

El desplazamiento para esta instrucción de salto es F6H, que es un número negativo. El salto es “hacia atrás” en la memoria. La dirección de origen para el salto es la que se encuentra después de la instrucción de salto, y en este ejemplo es 0804H. Para calcular la dirección de destino, simplemente añadimos el desplazamiento a la dirección de origen; sin embargo, esto requiere de un truco. Debemos “extender el signo” del desplazamiento ya que es negativo y la dirección es de

16 bits, así que lo expresaremos como un número de 16 bits: FFF6H. La dirección de destino se calcula como sigue:

$$\begin{array}{l} 0804 \text{ (dirección de origen)} \\ +\underline{\text{FFF6}} \text{ (desplazamiento)} \\ \hline 07FA \text{ (dirección de destino)} \end{array}$$

Nota: Al añadir un desplazamiento negativo a la dirección de origen se generará un acarreo del bit más significativo, el cual se desecha.

3.2.6 Direccionamiento absoluto

Las instrucciones ACALL y AJMP son las únicas que utilizan el direccionamiento absoluto. Estas instrucciones de 2 bytes proporcionan los 11 bits menos significativos de la dirección de destino en el código de operación (A10-A8) y el byte 2 de la instrucción (A7-A0) para permitir la bifurcación dentro de la página de 2K actual de la memoria para código. (Consulte la figura 3-1f).

Los cinco bits superiores de la dirección de destino son los cinco bits más significativos actuales del contador de programa, así que tanto la instrucción que sigue de la instrucción de bifurcación como la dirección de destino de la instrucción de bifurcación deben estar localizadas en la misma página de 2K, pues los bits A15-A11 no cambian. (Consulte la figura 3-3). Por ejemplo, si la etiqueta AQUI representa una instrucción en la dirección 0F46H, y la instrucción

AJMP AQUI

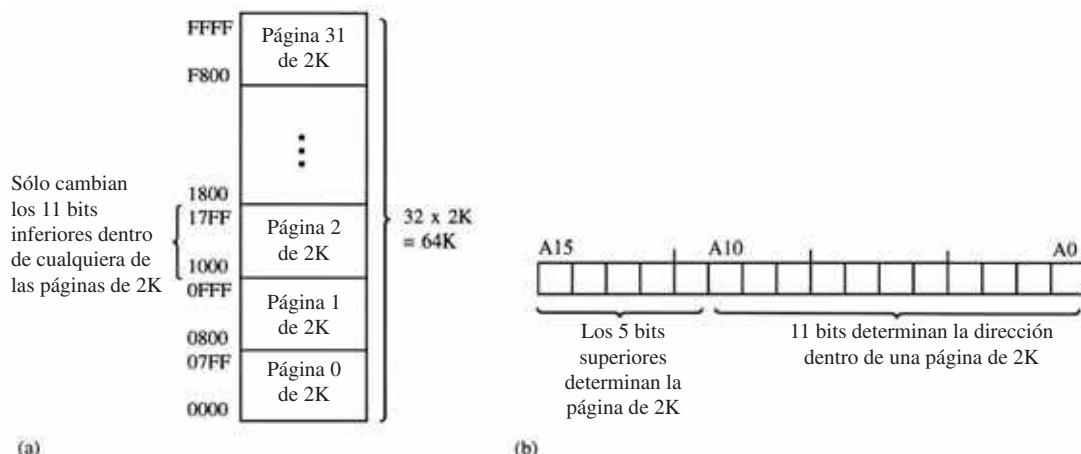


FIGURA 3-3

Codificación de instrucciones para direccionamiento absoluto. (a) Mapa de memoria con páginas de 2K. (b) Los cinco bits de dirección superiores son iguales en las direcciones de origen y destino dentro de cualquiera de las páginas de 2K. La instrucción proporciona los 11 bits inferiores de la dirección de destino.

está ubicada en las direcciones de memoria 0900H y 0901H, el ensamblador codificará la instrucción como

11100001 - primer byte (A10-A8 + código de operación)

01000110 - segundo byte (A7-A0)

Los bits subrayados son los 11 bits de menor orden de la dirección de destino, 0F46H = 0000111101000110B. Los cinco bits superiores del contenido del contador de programa no cambian cuando se ejecuta esta instrucción. Observe que la dirección de destino y la instrucción AJMP están dentro de la página de 2K limitada por las direcciones 0800H y 0FFFH (consulte la figura 3-3), y por lo tanto tienen los cinco bits de dirección superior en común.

El direccionamiento absoluto ofrece la ventaja de utilizar instrucciones cortas (de 2 bytes), pero tiene la desventaja de limitar el rango posible para la dirección destino, además de la necesidad de utilizar código que depende de su posición.

EJEMPLO 3.8

Una instrucción ACALL está ubicada en las direcciones de memoria 1024H y 1025H. La subrutina a llamar comienza en la dirección de memoria 17A6H. ¿Cuáles son los bytes en lenguaje de máquina para esta instrucción ACALL?

Solución

F1H, A6H

Análisis

Al consultar el apéndice C encontramos que la codificación de la instrucción ACALL es

aaa10001 aaaaaaaaa

Los 11 bits de orden inferior de la dirección de destino se insertan en la instrucción, donde los bits 10-8 representan los bits de orden superior del código de operación y los bits 7-0 representan el segundo byte de la instrucción. En seguida se muestra en binario la dirección de destino (17A6H) con los 11 bits de orden inferior identificados como un grupo de tres bits (10-8) y un grupo de ocho bits (7-0).

00010111 10100110 = 17A6H
aaa aaaaaaaaa

Aquí la tarea es posicionar en forma correcta los 11 bits de la dirección de destino en los bytes de la instrucción. El posicionamiento se ilustra como sigue:

11110001 10100110 = F1A6H
 aaa..... aaaaaaaaa

Nota: El direccionamiento absoluto sólo puede utilizarse cuando los cinco bits de orden superior son iguales en las direcciones de origen y destino. Esta propiedad causa que las direcciones de origen y destino deban ubicarse en la misma página de 2K.

3.2.7 Direccionamiento largo

Las instrucciones LCALL y LJMP son las únicas que utilizan el direccionamiento largo. Estas instrucciones de 3 bytes incluyen una dirección de destino de 16 bits, representada por los bytes 2 y 3 de la instrucción. (Consulte la figura 3-1g). La ventaja es que puede utilizarse todo el espacio

de código de 64K, pero la desventaja es que las instrucciones son de 3 bytes y dependen de su posición. La dependencia radicada en su posición es una desventaja porque el programa no es relocalizable, esto es, no puede ejecutarse en distintas direcciones de inicio. Por ejemplo, si un programa comienza en la dirección 2000H y aparece una instrucción tal como LJMP 2040H, el programa no puede cambiarse para que empiece en otra dirección, digamos en la dirección 4000H. De todas formas la instrucción LJMP saltaría a la dirección 2040H, que no sería la dirección correcta después de haber cambiado la dirección inicial del programa.

EJEMPLO En lenguaje de máquina, ¿cuáles son los bytes para la siguiente instrucción?

3.9

LJMP 8AF2H

Solución

02H, 8AH, F2H

Análisis

Tal como se muestra en el apéndice C, la instrucción LJMP tiene un tamaño de 3 bytes, de los cuales el primero contiene el código de operación (02H) y, en los bytes 2 y 3, se encuentra la dirección de destino de 16 bits. El byte superior de la dirección de destino (8AH) está contenido en el byte 2, mientras que el byte 3 contiene el byte inferior (F2H).

3.2.8 Direccionamiento indexado

El direccionamiento indexado utiliza un registro base (ya sea el contador de programa o el apuntador de datos) y un desplazamiento (el acumulador) para formar la dirección efectiva para una instrucción JMP o MOVC. (Consulte la figura 3-1h). Las tablas de salto o de búsqueda se crean fácilmente utilizando el direccionamiento indexado. El apéndice C presenta algunos ejemplos para las instrucciones MOVC A, @A + <registro base> y JMP @A + DPTR.

EJEMPLO ¿Cuál es el código de operación para la siguiente instrucción?

3.10

MOVC A, @A+DPTR

Solución

93H

Análisis

Para encontrar la respuesta sólo necesitamos consultar la instrucción MOVC en el apéndice C. La instrucción es de un solo byte, y el código de operación especifica tanto la operación como el modo de direccionamiento. La instrucción transfiere un byte de datos desde la memoria para código hacia el acumulador. Para encontrar la dirección en la memoria para código se suma el índice (el estado actual del acumulador) con el registro base (el apuntador de datos). El índice se pierde cuando la instrucción se termina de ejecutar, ya que el valor transferido de la memoria de código reemplaza al valor del índice.

3.3 TIPOS DE INSTRUCCIONES

Las instrucciones del 8051 se dividen en cinco grupos funcionales:

- Aritméticas
- Lógicas

- De transferencia de datos
- Booleanas
- De bifurcación de programa

El apéndice A presenta un diagrama de referencia que muestra a todas las instrucciones del 8051 agrupadas por función. Este diagrama le servirá como referencia útil y rápida una vez que usted se haya familiarizado con el conjunto de instrucciones. Examinaremos a continuación las instrucciones contenidas en cada grupo funcional del apéndice A.

3.3.1 Instrucciones aritméticas

Las instrucciones aritméticas están agrupadas en el apéndice A. La instrucción ADD A puede escribirse de distintas maneras, ya que hay cuatro posibles modos de direccionamiento:

ADD A, 7PH	(direcciónamiento directo)
ADD A, @R0	(direcciónamiento indirecto)
ADD A, R7	(direcciónamiento por registro)
ADD A, #35H	(direcciónamiento inmediato)

Todas las instrucciones aritméticas se ejecutan en un ciclo de máquina excepto INC DPTR (dos ciclos de máquina) y MUL AB y DIV AB (cuatro ciclos de máquina). (Observe que un ciclo de máquina dura 1 μ s si el 8051 opera mediante un reloj de 12 MHz).

EJEMPLO 3.11 El acumulador contiene los valores 63H, R3 contiene 23H, y la PSW contiene 00H. (a) En hexadecimal, ¿cuál es el contenido del acumulador y de la PSW luego de que se ejecuta la siguiente instrucción?

ADD A, R3

(b) En decimal, ¿cuál es el contenido del acumulador luego de que se ejecuta la siguiente instrucción?

Solución

(a) ACC = 86H, PSW = 05H. (b) Contenido decimal de ACC = ? (vea el análisis).

Análisis

Este ejemplo parece simple y directo a primera vista: dados dos valores, calculemos su suma. Sin embargo, debemos entender ciertos conceptos interesantes e importantes. Empezaremos por expresar los valores iniciales de ACC y R3 en notación decimal. Al utilizar el método común de conversión a decimal, tenemos que, A = 63H = 01100011B = 99₁₀ y R3 = 23H = 00100011B = 35₁₀. Sin embargo, existe un problema. Si asumimos que se utiliza la notación en complemento de dos, el número positivo más grande que podemos expresar en 8 bits es +127₁₀. Cuando utilizamos una notación sin signo, el valor más grande posible de 8 bits es +255₁₀ y, en este caso, el resultado final de 134₁₀ es perfectamente válido.

Es importante advertir que la CPU del 8051 no tiene ningún conocimiento especial sobre si los datos son binarios con signo, binarios sin signo, decimales codificados en binario, ASCII, etc. Sólo usted (el programador) puede saberlo con seguridad. El mecanismo para manipular diferentes formatos de datos se proporciona mediante los bits de estatus de la PSW. Para mostrar esto, a continuación detallamos la suma en binario.

11...11.
01100011 (ACC = 63H)

+00100011 (R3 = 23H)
10000110 (el resultado almacenado en ACC = 86H)

El resultado es $10000110_2 = 86H$. Observemos que han ocurrido acarreos de los bits 0, 1, 5 y 6. No hubo acarreos en los bits 2, 3, 4 y 7. El bit de acarreo C en la PSW no tomó el valor de 1 después de la suma porque no hubo acarreo del bit 7.

El apéndice C describe el bit de desbordamiento OV para la instrucción ADD como sigue: “el bit de OV se establece en 1 si hay un acarreo del bit 6 pero no del bit 7, o un acarreo del bit 7 pero no del bit 6; el bit OV se restablece en 0 en todos los demás casos”. Esta es la manera formal de decir que “el bit OV se establece en 1 si el resultado cae fuera del rango para números con signo de 8 bits”. En este ejemplo, existe un acarreo del bit 6 pero no del bit 7, y por lo tanto el bit OV es 1. Esto tiene sentido cuando los datos son números con signo, pues el rango permitido es de -128_{10} a $+127_{10}$ y el resultado de $99_{10} + 35_{10}$ está “fuera de rango”.

También es posible que, desde la perspectiva de la CPU del 8051, los datos sean decimales codificados en binario (sólo el programador puede estar seguro de esto) y, por lo tanto, pondrán a 1 o 0 el bit auxiliar de acarreo de la manera adecuada. Este bit de AC es 0 debido a que no ocurrió un acarreo del bit 3.

Por último, en la PSW, el bit P se pone a 1 o 0 para establecer una paridad par con el acumulador. El bit P es 1, puesto que el resultado en el ACC tiene tres bits iguales a uno, con lo cual ahora el resultado es un total de cuatro: un número par. En la PSW, el valor final es de $00000101B = 05H$. Sólo son 1 los bits de OV y P; los otros son 0 (consulte la tabla 2-3).

La segunda pregunta en este ejemplo es: “¿Cuál es el contenido, en hexadecimal, del acumulador y la PSW después de que se ejecuta la siguiente instrucción?”. Es aquí donde profundizamos en el importante concepto del significado o de la interpretación de los datos en que opera una CPU como la del 8051. No es posible contestar esta pregunta con toda seguridad ya que el problema original no indicó un formato o una representación para los datos originales, lo que explica el símbolo “?” en la solución. Sin embargo, existen por lo menos dos posibles respuestas. En primer lugar, si asumimos que los datos originales están en notación binaria sin signo, entonces el resultado caerá “dentro del rango” (ya que C = 0) y la respuesta será 134_{10} . En segundo lugar, suponiendo que los datos están en notación binaria con signo utilizando la notación de complemento de dos, entonces la respuesta correcta estará “fuera de rango” (porque OV = 1). Lo importante es que el significado o el esquema de representación no son, en efecto, una característica de la CPU, sino que se determinan por la manera en que el software manipula los datos.

EJEMPLO 3.12

Ilustre una secuencia de instrucciones para restar el contenido de R6 del contenido de R7 y dejar el resultado en R7.

Solución

```
MOV A, R7
CLR C
SUBB A, R6
MOV R7, A
```

Análisis

El acumulador mantiene uno de los valores para las operaciones tanto de suma como de resta. En consecuencia, la primera instrucción dada en la solución transfiere uno de los bytes al acumulador para prepararse para la operación. La segunda instrucción restablece en 0 la bandera de acarreo en la palabra de estado del programa. Esto es necesario porque la única forma de la instrucción de resta es SUBB (resta con préstamo). La operación efectúa una resta del byte fuente y del bit de acarreo del acumulador. Para las operaciones de resta, el bit de acarreo funciona como un bit

de “préstamo”. Si no se conoce el estado del bit de acarreo, éste debe ser puesto en 0 explícitamente utilizando la instrucción CLR C antes de ejecutar SUBB. La tercera instrucción ejecuta la operación de resta y deja el resultado en el acumulador. La cuarta instrucción transfiere el resultado a R7.

El 8051 cuenta con una poderosa capacidad de direccionamiento de su espacio de memoria interna. Cualquier ubicación puede ser incrementada o disminuida mediante el uso del direccionamiento directo, sin tener que acceder al acumulador. Por ejemplo, si la ubicación de memoria RAM interna 7FH contiene el valor 40H, entonces la instrucción

```
INC 7FH
```

incrementa este valor, dejando el nuevo valor de 41H en la ubicación 7FH.

EJEMPLO 3.13 Suponga que el 8051 no permite una instrucción para incrementar en forma directa una ubicación de memoria RAM interna. ¿Cómo realizaría esta operación?

Solución

```
MOV A,directo
INC A
MOV directo,A
```

Análisis

La primera instrucción transfiere un byte de datos desde la ubicación de memoria RAM interna hasta el acumulador. La segunda instrucción incrementa el valor leído (que ahora está en el acumulador), y la tercera instrucción escribe el resultado en la memoria RAM interna. Esta secuencia de instrucciones no sólo es más larga y más lenta que la instrucción individual equivalente (INC directo), sino que también se pierde el valor del acumulador.

Una de las instrucciones INC opera sobre el apuntador de datos de 16 bits. Como éste genera direcciones de 16 bits para la memoria externa, una característica útil sería poder incrementarlo mediante una sola operación. Desafortunadamente, no hay una instrucción para disminuir el apuntador de 16 bits, y se requiere de una secuencia de instrucciones como la siguiente:

```
DEC DPL ;DECREMENTA BYTE INFERIOR DE DPTR
MOV R7,DPL ;TRANSIERE A R7
CJNE R7,#0FFH,SALTA ;SI DPL = 0FFH
DEC DPH ;DECREMENTA BYTE SUPERIOR TAMBIEN
SALTA: (continúa)
```

Los bytes superior e inferior del DPTR deben disminuirse por separado; sin embargo, el byte superior (DPH) sólo se disminuye cuando ocurre un cambio de 00H a FFH en el byte inferior (DPL).

La instrucción MUL AB multiplica el acumulador por el dato localizado en el registro B, y transfiere el producto de 16 bits al registro B (byte superior) y al acumulador (byte inferior), ambos están concatenados. La instrucción DIV AB divide el acumulador entre el dato localizado

en el registro B, dejando el cociente de 8 bits en el acumulador y el residuo de 8 bits en el registro B. Por ejemplo, si A contiene el valor 25 (19H) y B el valor 6 (06H), la instrucción

DIV AB

divide el contenido de A entre el contenido de B. El acumulador A recibe el valor 4 y el acumulador B el valor 1. ($25 \div 6 = 4$ con un residuo de valor igual a 1).

- EJEMPLO 3.14** El acumulador contiene el valor 55H, el registro B contiene el valor 22H, y la palabra de estado del programa contiene 00H. ¿Cuál es el contenido de estos registros después de ejecutar la siguiente instrucción

MUL AB

Solución

ACC = 4AH, B = 0BH, PSW = 05H

Análisis

Usted podría calcular la respuesta utilizando una calculadora; de otra manera, comience por expresar los valores originales en decimal: ACC = 55H = 85_{10} y B = 22H = 34_{10} . El producto es $85_{10} \times 34_{10} = 2,890_{10} = 0B4AH$. El byte superior (0BH) se transfiere al registro B y el byte inferior (4AH) al acumulador. En la PSW, el bit P asume un valor de 1 para establecer una paridad par con el acumulador. La bandera de desbordamiento se establece en 1 debido a que el resultado es mayor a 255_{10} , lo cual resulta en el valor 05H en la PSW.

- EJEMPLO 3.15** El acumulador contiene el valor 1FH. ¿Cuál es el valor más grande que pudiera estar en el registro B pero que *no* causará el establecimiento en 1 del bit OV después de ejecutar la siguiente instrucción?

MUL AB

Solución

08H

Análisis

El acumulador contiene el valor 1FH = 31_{10} . La bandera de desbordamiento se establece en 1 después de una instrucción MUL si el producto es mayor a 255_{10} . El valor más grande que podemos multiplicar por 31_{10} sin que el resultado exceda 255_{10} es el valor $8_{10} = 08H$. (Advierta que $31_{10} \times 8_{10} = 248_{10}$, $31_{10} \times 9_{10} = 279_{10}$).

Para la aritmética BCD (decimal codificado en binario), las instrucciones ADD y ADDC deben ir seguidas de una operación DA A (ajuste decimal) para asegurar que el resultado se encuentre dentro del rango para BCD. Observe que la instrucción DA A no convierte un número binario en BCD, sino que produce un resultado significativo sólo durante el segundo paso efectuado en la suma de dos bytes BCD. Por ejemplo, si A contiene el valor BCD 59 (59H), entonces la secuencia de instrucciones

ADD A, #1
DA A

primero suma el valor 1 a A, dejando el resultado 5AH, y después ajusta el resultado al valor BCD correcto de 60 (60H). ($59 + 1 = 60$).

**EJEMPLO
3.16**

Ilustre cómo se sumarían dos números decimales en código binario de 4 dígitos. El primer número está en las ubicaciones de memoria interna 40H y 41H, y el segundo en las ubicaciones 42H y 43H. Los dígitos más significativos están ubicados en 40H y 42H. Coloque el resultado BCD en las ubicaciones 40H y 41H.

Solución

```

MOV    A, 43H
ADD    A, 41H
DA     A
MOV    41H, A
MOV    A, 42H
ADDC   A, 40H
DA     A
MOV    40H, A

```

Análisis

Este es un ejemplo de la “aritmética de múltiple precisión”, donde se requiere que la CPU realice operaciones aritméticas en datos que son más grandes que el tamaño natural de datos de la CPU. Aun cuando las instrucciones aritméticas del 8051 operan sobre bytes, es posible sumar o restar datos más grandes que, por ejemplo, 16 o 32 bits. Esto también se aplica a los números decimales codificados en binario. Para sumar dos números de 4 dígitos, se requieren dos sumas de bytes.

El truco en la aritmética de precisión múltiple es la propagación de acarreos de byte a byte. Los acarreos se propagan de manera natural de bit a bit dentro de un byte durante la suma, pero un acarreo de byte a byte (en caso de que ocurra) se almacena temporalmente en el bit C de la PSW. En el ejemplo, la primera instrucción es la instrucción genérica “ADD”, debido a que un acarreo no puede ocurrir en los bytes de orden inferior. Sin embargo, la instrucción “ADDC” se requiere para la segunda operación de suma, para incluir el acarreo que pudiera haber ocurrido del byte de orden inferior. Ambas instrucciones de suma van seguidas de una instrucción DA A para realizar los ajustes necesarios para valores decimales codificados en binario.

3.3.2 Instrucciones lógicas

Las instrucciones lógicas del 8051 (consulte el apéndice A) realizan operaciones booleanas (AND, OR, OR exclusivo y NOT) sobre bytes de datos, bit por bit. Si el acumulador contiene el valor 00110101B, la siguiente instrucción lógica AND

```
ANL A, #01010011B
```

deja el valor 00010001B en el acumulador. Esto se ilustra como sigue:

y	01010011	(dato inmediato)
	00110101	(valor original de A)
	00010001	(resultado en A)

La instrucción lógica AND puede tomar diferentes formas debido a que los modos de direccionamiento para las instrucciones lógicas son los mismos que para las instrucciones aritméticas:

ANL A, 55H	(ddireccionamiento directo)
ANL A, @R0	(ddireccionamiento indirecto)
ANL A, R6	(ddireccionamiento por registros)
ANL A, #33H	(ddireccionamiento inmediato)

Todas las instrucciones lógicas que utilizan al acumulador como uno de los operandos se ejecutan en un ciclo de máquina. Las otras instrucciones necesitan dos ciclos de máquina.

Las operaciones lógicas pueden realizarse en cualquier byte en el espacio de memoria interna para datos, sin tener que acceder al acumulador. La instrucción “XRL directo,#datos” ofrece una manera fácil y rápida de invertir los bits de los puertos, como por ejemplo

```
XRL P1, #0FFH
```

Esta instrucción realiza una operación de lectura-modificación-escritura. Se leen los ocho bits en el puerto 1; se realiza una operación OR exclusivo en cada uno de los bits leídos con cada bit correspondiente en los datos inmediatos. El resultado es que cada bit leído se complementa (en otras palabras, $A \oplus 1 = \overline{A}$), ya que todos los bits del dato inmediato tienen valor de 1. El resultado se escribe en el puerto 1.

Las instrucciones de rotación (RL A y RR A) desplazan una posición, a la izquierda o a la derecha respectivamente, todos los bits del acumulador. Para una rotación a la izquierda, el bit más significativo se desplaza a la posición del bit menos significativo. Para una rotación a la derecha, el bit menos significativo se desplaza a la posición del bit más significativo. Las variaciones de estas instrucciones, RLC A y RRC A, son rotaciones de 9 bits que utilizan el acumulador y la bandera de acarreo incluidos en la PSW. Por ejemplo, si la bandera de acarreo contiene el valor 1 y A contiene el valor 00H, la instrucción

```
RRC A
```

deja la bandera de acarreo en 0 y el registro A igual a 80H. La bandera de acarreo se rota hacia el bit ACC.7 y el bit ACC.0 rota hacia la bandera de acarreo.

La instrucción SWAP A intercambia los nibbles superior e inferior dentro del acumulador. Esta es una operación útil para manipular números en BCD. Por ejemplo, si el acumulador contiene un número binario que sabemos es menor a 100_{10} , éste se convierte rápidamente en BCD de la siguiente manera:

```
MOV B, #10
DIV AB
SWAP A
ADD A, B
```

Al dividir el número entre 10 en las primeras dos instrucciones, el de las décadas se deja en el nibble inferior del acumulador y el dígito de las unidades en el registro B. Las instrucciones SWAP y ADD transfieren el dígito de las décadas al nibble superior del acumulador, y el dígito de las unidades al nibble inferior.

EJEMPLO 3.17

Ilustre dos maneras de rotar el contenido del acumulador tres posiciones a la izquierda. Examine las ventajas y desventajas de cada método en términos de los requisitos de memoria y velocidad de ejecución.

Solución

- (a) RL A
 RL A
 RL A
 (b) SWAP A
 RR A

Análisis

La solución (a) es la más evidente, mientras que la solución (b) utiliza un truco. La instrucción SWAP A intercambia los nibbles de bajo y alto nivel en el acumulador y, como se describe en el apéndice C, esto equivale a una operación de rotación de 4 bits. Se utiliza una rotación final a la derecha para invalidar la cuarta rotación.

Aunque el efecto de ambas soluciones es el mismo, éstas son un tanto diferentes en términos de uso de memoria y velocidad de ejecución. Todas las instrucciones antes descritas son de 1 byte y utilizan un ciclo de máquina (consulte el apéndice C), así que la solución (a) utiliza tres bytes de memoria y tres ciclos de CPU para ejecutarse. La solución (b) sólo utiliza dos bytes de memoria y se ejecuta en dos ciclos de CPU. La diferencia puede ser minúscula, pero en el mundo de los sistemas de control embebidos, uno de los requisitos de diseño más comunes es el de aprovechar hasta “la última gota” de un programa.

EJEMPLO 3.18 Escriba una secuencia de instrucciones para invertir el orden de los bits en el acumulador. Se intercambiarán los bits 7 y 0, 6 y 1, etcétera.

Solución

```
MOV R7, #8
CICLO: RLC A
        XCH A, 0FOH
        RRC A
        XCH A, 0FOH
DJNZ R7, CICLO
        XCH A, 0FOH
```

Análisis

Algunas veces, a las peripecias requeridas en este ejemplo se les denomina “bailar con los bits”. La tarea puede parecer inútil, pero resulta curioso cómo a menudo surgen situaciones que requieren manipular la posición de los bits dentro de los bytes.

La solución es crear el nuevo valor en el registro B y desplazar un bit del acumulador de manera sucesiva al bit de acarreo, y luego desplazar el mismo bit al registro B. Para invertir el patrón de los bits, el primer desplazamiento es “a la izquierda” y el segundo “a la derecha”. El registro B y el acumulador se intercambian (XCH) después de cada rotación debido a que la instrucción de rotación sólo opera sobre el acumulador. Una instrucción XCH final posiciona el resultado correcto en el acumulador. Nota: El registro B está en la dirección directa 0F0H.

3.3.3 Instrucciones para transferencia de datos

3.3.3.1 RAM interna Las instrucciones que transfieren datos dentro de los espacios de memoria interna (consulte el apéndice A) se ejecutan ya sea en uno o en dos ciclos de máquina. El formato de la instrucción

```
MOV <destino>, <fuente>
```

permite la transferencia de los datos entre cualquier par de ubicaciones de memoria RAM interna o SFR sin tener que acceder al acumulador. Recuerde que sólo se tiene acceso a los 128 bytes

superiores de la memoria RAM para datos (8032/8052) mediante el direccionamiento indirecto, y sólo se accede a los SFR empleando el direccionamiento directo.

Una característica de la arquitectura de la familia MCS-51TM que difiere de la mayoría de los microprocesadores es que la pila está localizada en la memoria RAM incorporada en el chip y crece hacia arriba en la memoria, hacia direcciones de memoria más altas. La instrucción PUSH primero incrementa el apuntador de pila (SP) y luego copia el byte a la pila. Las instrucciones PUSH y POP utilizan el direccionamiento directo para identificar el byte que se va a almacenar o a recuperar, pero se accede a la pila mediante el direccionamiento indirecto utilizando el registro SP. Este direccionamiento indirecto significa que la pila puede utilizar los 128 bytes superiores de la memoria interna en el 8032/8052.

Los 128 bytes superiores de la memoria interna no están implementados en los dispositivos 8031/8051. Con estos dispositivos, si la SP se incrementa más allá del valor 7FH (127), los bytes almacenados por la instrucción PUSH se pierden y los recuperados por la instrucción POP son indeterminados.

EJEMPLO 3.19

El apuntador de la pila contiene el valor 07H, el acumulador A contiene 55H, y el acumulador B contiene 4AH. En la memoria RAM interna, ¿cuáles ubicaciones se alteran y cuáles son sus nuevos valores después de ejecutar las siguientes instrucciones?

```
PUSH    ACC
PUSH    0F0H
```

Solución

Dirección	Contenido
08H	55H
09H	4AH
81H (SP)	09H

Análisis

La primera instrucción almacena el acumulador en la pila. El apuntador de la pila se incrementa antes de la instrucción PUSH, así que el contenido de A (55H) se escribe a la memoria RAM interna en la ubicación 08H. La segunda instrucción almacena el acumulador B en la pila, el cual está ubicado en la dirección de memoria RAM interna F0H. Una vez más se incrementa el apuntador de pila antes de la instrucción PUSH, así que el contenido de B (4AH) se escribe a la memoria RAM interna en la dirección 09H. El apuntador de pila se incrementa dos veces como resultado de las dos instrucciones, así que su valor final es 09H.

Las instrucciones para la transferencia de datos incluyen una instrucción MOV de 16 bits para inicializar el apuntador de datos (DPTR) para acceder a tablas de búsqueda en la memoria de programa, o para accesos de 16 bits a la memoria externa para datos.

EJEMPLO 3.20

¿Cuál es la dirección del apuntador de datos en la memoria RAM interna?

Solución

DPH (el byte superior del apuntador de datos) está en la dirección 83H, y el byte inferior (DPL) en la dirección 82H.

Análisis

DPH (el byte superior del apuntador de datos) está en la dirección 83H, y el byte inferior (DPL) en la dirección 82H.

El formato de la instrucción

```
XCH A,<fuente>
```

hace que el acumulador y el byte direccionado intercambien sus datos. Una instrucción de intercambio de un solo “dígito” de la forma

```
XCHD A, @Ri
```

es similar, pero sólo se intercambian los nibbles de orden inferior. Por ejemplo, si A contiene el valor F3H, R1 contiene 40H, y la dirección de memoria RAM interna contiene 5BH, entonces la instrucción

```
XCHD A, @R1
```

deja el valor FBH en A, y la ubicación de memoria RAM interna 40H contendrá el valor 53H.

EJEMPLO 3.21 ¿Cuál es el contenido de los acumuladores A y B después de ejecutar las siguientes instrucciones?

```
MOV    0F0H, #12H
MOV    R0, #0F0H
MOV    A, #34H
XCH    A, 0F0H
XCHD   A, @R0
```

Solución

A = 14H, B = 32H

Análisis

Las primeras tres instrucciones establecen lo que ocurrirá en este ejemplo, terminando con el resultado de A = 34H, B = 12H, y R0 = F0H. Recuerde que el acumulador B está ubicado en la memoria RAM interna F0H. La cuarta instrucción intercambia los valores de A y B, dejando A = 12H y B = 34H. La quinta instrucción intercambia los dígitos de orden inferior de A y B, dejando A = 14H y B = 32H.

3.3.3.2 Memoria RAM externa Las instrucciones de transferencia de datos que transfieren datos entre la memoria interna y externa utilizan el direccionamiento indirecto. La dirección indirecta se especifica mediante el uso de una dirección de 1 byte (@Ri, donde Ri es R0 o R1 del banco de registros seleccionado) o una dirección de 2 bytes (@DPTR). La desventaja de utilizar direcciones de 16 bits es que se utilizan los ocho bits del puerto 2 como el byte superior del bus de direcciones. Esto deshabilita al puerto 2 como un puerto de E/S. Por otra parte, las direcciones de 8 bits permiten el acceso a pocos Kbytes de memoria RAM sin sacrificar a todo el puerto 2. (Consulte el capítulo 2, “Acceso a la memoria externa”).

Todas las instrucciones de transferencia de datos que operan en la memoria externa se ejecutan en dos ciclos de máquina y utilizan el acumulador, ya sea como el operando fuente o como el operando destino.

Los señales de sincronización de lectura y escritura de la memoria RAM externa (\overline{RD} y \overline{WR}) se activan sólo durante la ejecución de una instrucción MOVX. Estas señales están desactivadas (nivel alto) normalmente y, si la memoria externa para datos no se utiliza, se hallan disponibles como líneas de propósito general de E/S.

EJEMPLO 3.22 Ilustre una secuencia de instrucciones para leer el contenido de las ubicaciones de memoria RAM externa 10F4H y 10F5H y transferir los valores leídos a R6 y R7, respectivamente.

Solución

```
MOV    DPTR, #10F4H
MOVX   A, @DPTR
```

```

MOV    R6 ,A
INC    DPTR
MOVX   A ,@DPTR
MOV    R7 ,A

```

Análisis

La primera instrucción inicializa el apuntador de datos con la primera de las dos direcciones externas que serán leídas. La segunda instrucción lee un byte de la ubicación de memoria externa 10F4H y lo transfiere al acumulador. La tercera instrucción transfiere el byte leído al registro 6. Observe que la instrucción MOVX utiliza el acumulador, ya sea como fuente o como destino de los datos. La cuarta instrucción incrementa el apuntador de datos y lo deja apuntando a la segunda de las dos direcciones externas que se van a leer. La quinta instrucción lee un byte de la ubicación de memoria externa 10F5H y lo transfiere al acumulador. La sexta instrucción transfiere el byte leído al registro 7.

3.3.3.3 Tablas de búsqueda Hay dos instrucciones de transferencia de datos disponibles para leer tablas de búsqueda en la memoria para programas. Las tablas de búsqueda sólo pueden leerse, no actualizarse, ya que acceden a la memoria para programas. El nemónico es MOVC por su significado de “muestra una constante”, o “transfiere una constante”. MOVC utiliza el contador de programa o el apuntador de datos como registro base y el acumulador como desplazamiento.

La instrucción

```
MOVC  A ,@A+DPTR
```

puede alojar una tabla de 256 elementos, enumerados de 0 a 255. El número del elemento deseado se carga al acumulador y el apuntador de datos se inicializa al comienzo de la tabla. La instrucción

```
MOVC  A ,@A+PC
```

trabaja de igual manera, excepto que la dirección base es el contador de programa. Por lo general, se accede a la tabla mediante una subrutina. Primero, el número del elemento deseado se carga al acumulador y después se llama la subrutina. La secuencia de inicialización y de llamada se puede codificar como sigue:

```

MOV    A ,#NUMERO_DE_ELEMENTO
CALL   BUSQUEDA
.
.
.
BUSQUEDA:   INC   A
            MOVC  A ,@A+PC
            RET
TABLA:      DB    dato, dato, dato, dato, ...

```

La tabla sigue justo después de la instrucción RET en la memoria para programa. Necesitamos la instrucción INC, ya que el PC (contador de programa) apunta a la instrucción RET cuando se ejecuta la instrucción MOVC. Al incrementar el acumulador podremos sobrepasar la instrucción RET cuando se lleve a cabo la búsqueda en la tabla.

Observe, sin embargo, que aunque esperábamos contar con una tabla de 256 elementos cuando utilizamos esta técnica, en este caso únicamente podemos disponer de 255 elementos ya que perdemos un elemento por causa del efecto de la instrucción INC A. Considere el número de elemento 255 que se transfiere al acumulador. La instrucción INC A lo incrementa en 1, lo cual hace

que el valor 255 se recicle al valor 0 debido a que el acumulador tiene un tamaño de 8 bits. La siguiente instrucción, MOVC A,@A + PC, intentará cargar el acumulador con el valor de la instrucción RET, lo cual es inválido. Por lo tanto, los elementos válidos son sólo del elemento 0 al 254.

EJEMPLO 3.23 Escriba una subrutina llamada CUADRADO para calcular el cuadrado de un número entero que esté entre 0 y 9. Inicie la subrutina con el entero en A y regrese de la subrutina con el cuadrado del entero en A. Escriba dos versiones de la subrutina: (a) utilizando una tabla de búsqueda, y (b) sin utilizar una tabla de búsqueda. Después (c) ilustre una secuencia de llamada para convertir el número 6 en su cuadrado, 36.

Solución

a. Utilizando una tabla de búsqueda:

```
CUADRADO : INC A
           MOVC A, @A+PC
           RET
TABLA :    DB 0,1,4,9,16,25,36,49,64,81
```

b. Sin utilizar una tabla de búsqueda:

```
CUADRADO : PUSH OFOH
           MOV OFOH, A
           MUL AB
           POP OFOH
           RET
```

c. Secuencia de llamada:

```
MOV A, #6
CALL CUADRADO
```

Análisis

La solución en (a) y la secuencia de llamada (c) son implementaciones simples y directas de una tabla de búsqueda en la memoria de programas. Sin embargo, existe una interesante alternativa para este ejemplo, como se muestra en (b). Al copiar el contenido de A en el acumulador B (en la ubicación de memoria RAM interna F0H) y después multiplicar B por A utilizando la instrucción MUL AB, calculamos el cuadrado y lo dejamos en A. El acumulador B se almacena primero en la pila, debido a que se sustituirá en la segunda instrucción, luego lo recuperaremos antes de regresar de la subrutina. Tal vez las instrucciones PUSH/POP no sean necesarias, dependiendo del contexto; sin embargo, el diseño de subrutinas que presenten el menor número de posibles efectos secundarios es una buena práctica de programación.

Si comparamos las soluciones (a) y (b) encontraremos un sacrificio interesante. La solución (a) es de 13 bytes, incluyendo a la tabla de búsqueda, mientras que la solución (b) es de sólo ocho bytes. Sin embargo, la solución (a) se ejecuta en cinco ciclos de la CPU, mientras que la solución (b) requiere de 11 ciclos. El sacrificio es que la solución (a) es más rápida (lo cual es bueno) pero consume más memoria (lo cual es malo), comparado con la solución (b) que es más lenta (mal) pero consume menos memoria (bueno). Este sacrificio sería más pronunciado si la tabla de búsqueda fuera más grande, pues cada elemento añade un byte al tamaño de la rutina.

En muchas situaciones, la relación entre el índice para una tabla y los elementos de una tabla no es tan simple como en este ejemplo, y la única implementación viable es utilizar una tabla de búsqueda.

3.3.4 Instrucciones booleanas

El procesador del 8051 contiene un procesador booleano completo para efectuar operaciones sobre bits individuales. La memoria RAM interna contiene 128 bits direccionables, y el espacio para los SFR soporta hasta 128 bits direccionables más. Todas los bits de los puertos son direccionables individualmente, y cada bit puede tratarse como un puerto separado de un solo bit. Las instrucciones que acceden a estos bits no sólo son bifurcaciones condicionales, sino también un repertorio completo de instrucciones para transferir, ser puestos en 1 o 0, complementar, y ejecutar operaciones lógicas como OR y AND. Tales operaciones sobre bits (una de las características más poderosas de la familia de microcontroladores MCS-51TM) no son tan fáciles de obtener en otras arquitecturas con operaciones orientadas a bytes.

El apéndice A muestra las instrucciones booleanas disponibles. Todos los accesos a bits utilizan el direccionamiento directo con las direcciones de bit 00H-7FH en las 128 ubicaciones inferiores, y las direcciones de bit 80H-FFH en el espacio de los SFR. En las 128 ubicaciones inferiores en las direcciones de byte 20H-2FH, los accesos están enumerados de manera secuencial, desde el bit 0 de la dirección 20H (bit 00H) hasta el bit 7 de la dirección 2FH (bit 7FH).

Los bits pueden ser puestos en 1 o 0 utilizando una sola instrucción. El control mediante bits individuales es común para muchos dispositivos de E/S, incluyendo la salida a relevadores, motores, solenoides, LEDs de estado, timbres, alarmas, altavoces, o la entrada proveniente de una variedad de interruptores o indicadores de estado. Por ejemplo, si se conecta una alarma al bit 7 del puerto 1, podría activarse al establecer en 1 el bit de puerto

```
SETB P1.7
```

y podría desactivarse si se restablece en 0 el bit de puerto

```
CLR P1.7
```

El ensamblador hará la conversión necesaria del símbolo “P1.7” a la dirección de bit correcta, 97H.

Observe lo fácil que es transferir el valor de una bandera interna a una terminal de puerto:

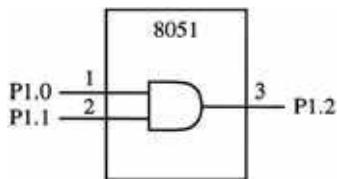
```
MOV C, BANDERA
MOV P1.0, C
```

En este ejemplo, la etiqueta BANDERA es el nombre de cualquier bit direccionable localizado en las 128 ubicaciones inferiores o en el espacio de los SFR. Una línea de E/S (el bit menos significativo del puerto 1, en este caso) es puesto en 1 o 0, dependiendo de si el bit de la bandera es 1 o 0.

El bit de acarreo en la palabra de estado del programa (PSW) se utiliza como el acumulador de un solo bit del procesador booleano. Las instrucciones de bit que hacen referencia al bit de acarreo como el término “C” están ensambladas como instrucciones específicas para el acarreo (por ejemplo, CLR C). El bit de acarreo también tiene una dirección directa ya que reside en el registro PSW, el cual es direccionable por bit. Los bits de la PSW, así como otros SFR direccionables por bit, tienen nemónicos predefinidos que el ensamblador acepta en lugar de la dirección de bit. El nemónico para la bandera de acarreo es “CY,” el cual se define como la dirección de bit 0D7H. Considere las siguientes dos instrucciones:

```
CLR C
CLR CY
```

Ambas tienen el mismo efecto; sin embargo, la primera instrucción es de un solo byte, mientras que la segunda es de 2 bytes. En la segunda instrucción, el segundo byte es la dirección directa del bit especificado: la bandera de acarreo.

**FIGURA 3-4**

Implementación simple de una operación lógica AND

Operaciones lógicas tales como AND, OR, NAND, NOR y NOT son fáciles de implementar para variables de un solo bit. Además, las variables pueden ser señales de entrada o de salida en los puertos de E/S del 8051. Por lo tanto, el estado de una terminal de E/S se puede leer o escribir en forma directa al acompañarse de una operación booleana. Suponga que deseamos calcular un AND lógico de las señales de entrada en el bit 0 y el bit 1 del puerto 1 y escribir el resultado al bit 2 del puerto 1. La figura 3-4 ilustra esta relación lógica.

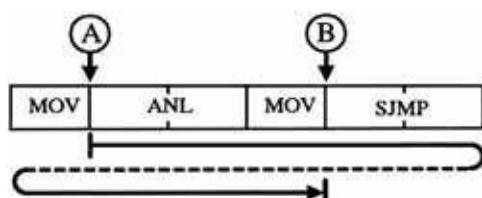
La siguiente secuencia de instrucciones crea esta operación lógica:

CICLO:	MOV C, P1.0	;n1 = 1 ciclo
	ANL C, P1.1	;n2 = 2 ciclos
	MOV P1.2, C	;n3 = 2 ciclos
	SJMP CICLO	;n4 = 2 ciclos

Los bits P1.0 y P1.1 se leen en forma continua y el AND lógico del resultado se presenta en forma continua en el bit P1.2.

Si en la figura 3-4 se implementa la operación lógica utilizando un circuito lógico electrónico común, tal como un 74AL508, el retraso de propagación de entrada a salida es del orden de 7 ns. Este es el tiempo que transcurre desde una transición de la señal de entrada hasta la aparición del nivel lógico correcto en la salida. ¿Cuál es el retraso de propagación en el peor caso para el circuito mostrado en la figura 3-4, si se implementa utilizando el software antes descrito? La figura 3-5 ayuda a responder esta pregunta. El peor caso es que el bit P1.0 cambie justo después de la primera instrucción, como se muestra en el punto “A” en la figura. El cambio no se identifica sino hasta el siguiente paso a través del ciclo. El estado de salida correcto aparece en el punto “B” en la figura. La línea continua de la figura 3-5 ilustra la secuencia de instrucciones para el peor caso. Se requieren 11 ciclos de CPU del punto “A” al punto “B”. Si operamos a 12 MHz, el retraso en la peor de las situaciones sería de 11 μ s. Resulta evidente que no hay comparación entre la velocidad de un microcontrolador y la de los circuitos lógicos electrónicos. La implementación efectuada en software para la figura 3-4 es cerca de 1 mil veces más lenta que una compuerta AND 74LS08!

Observe que las instrucciones booleanas incluyen las instrucciones ANL (AND lógico) y ORL (OR lógico), pero no la operación XRL (OR lógico exclusivo). Una operación XRL es fácil de implementar. Por ejemplo, suponga que se requiere calcular el OR exclusivo de dos bits,

**FIGURA 3-5**

Secuencia de instrucciones para el peor caso de retraso de propagación

BIT1 y BIT2, y el resultado se dejará en la bandera de acarreo. Las instrucciones se muestran a continuación.

```
MOV    C,BIT1
JNB    BIT2,SALTO
CPL    C
SALTO: (continua)
```

Primero, el BIT1 se transfiere a la bandera de acarreo. Si el BIT2 = 0, entonces C contendrá el resultado correcto; en otras palabras, $\text{BIT1} \oplus \text{BIT2} = \text{BIT1}$ si $\text{BIT2} = 0$. Si $\text{BIT2} = 1$, entonces C contendrá el complemento del resultado correcto. El complemento de C termina la operación.

3.3.4.1 Prueba de bits El código presentado en el ejemplo anterior utiliza la instrucción JNB, una de la serie de instrucciones para prueba de bits que ejecutan un salto si el bit direccionado es 1 (JC, JB, JBC) o si no es 1 (JNC, JNB). En el caso anterior, si $\text{BIT2} = 0$ ignoramos la instrucción CPL. JBC (salta si el bit es 1 y después restablece en 0 el bit) ejecuta el salto si el bit direccionado es 1 y también restablece en 0 el bit; por lo tanto, se puede probar y restablecer en 0 una bandera mediante una sola instrucción.

Todos los bits de la PSW pueden direccionarse en forma directa, así que el bit de paridad o las banderas de propósito general, por ejemplo, también están disponibles para cumplir las instrucciones de prueba de bits.

3.3.5 Instrucciones de bifurcación de programa

Como es evidente en el apéndice A, existe un gran número de instrucciones para controlar el flujo de los programas, incluyendo aquellas que llaman y regresan de subrutinas o se bifurcan en forma condicional o incondicional. Estas posibilidades aumentan su utilidad gracias a los tres modos de direccionamiento para las instrucciones de bifurcación de programa.

Existen tres variaciones de la instrucción JMP: SJMP, LJMP y AJMP (las cuales utilizan los direccionamientos relativo, largo y absoluto, respectivamente). El ensamblador de Intel (ASM51) permite el uso del nemónico genérico JMP si no le importa al programador qué variación es codificada. Los ensambladores de otras compañías tal vez no ofrezcan esta característica. La instrucción genérica JMP se ensambla como instrucción AJMP si el destino no contiene una referencia hacia delante y si está en la misma página de 2K que la instrucción siguiente a la instrucción AJMP. De otra manera, se ensambla como instrucción LJMP. La instrucción genérica CALL (vea abajo) funciona de igual manera.

La instrucción SJMP especifica la dirección de destino como un desplazamiento relativo, como se mostró en el análisis anterior sobre los modos de direccionamiento. La distancia del salto está limitada a -128 a +127 bytes relativos a la dirección que sigue de la instrucción SJMP, ya que ésta tiene un tamaño de dos bytes (un código de operación además de un desplazamiento relativo).

La instrucción LJMP especifica la dirección de destino como una constante de 16 bits. La dirección de destino puede encontrarse en cualquiera de los 64K de espacio de memoria de programas, ya que la instrucción es de tres bytes (un código de operación además de dos bytes de dirección).

La instrucción AJMP especifica la dirección de destino como una constante de 11 bits. Así como la instrucción SJMP, esta instrucción tiene dos bytes, pero su codificación es diferente. El código de operación contiene tres de los 11 bits de dirección, y el segundo byte almacena los ocho bits de orden inferior de la dirección de destino. Cuando se ejecuta la instrucción, estos 11 bits reemplazan los 11 bits de orden inferior en el PC (contador de programa), y los cinco bits de orden superior en el PC se mantienen iguales. El destino, por lo tanto, debe estar dentro del mismo bloque

de 2K que la instrucción siguiente a la instrucción AJMP. Existen 32 de estos bloques, ya que hay 64K de espacio de memoria para código, y cada uno comienza en un límite de direcciones de 2K (0000H, 0800H, 1000H, 1800H, etc., hasta llegar a F800H; consulte la figura 3-3).

En todos estos casos el programador especifica la dirección de destino al ensamblador de la manera usual: como una etiqueta o como una constante de 16 bits. El ensamblador colocará la dirección de destino en el formato correcto para cada instrucción. Si el formato requerido por la instrucción no soporta la distancia a la dirección de destino especificada, se proporcionará el mensaje “destino fuera de rango”.

3.3.5.1 Tablas de salto La instrucción @A + DPTR soporta saltos dependientes del caso para las tablas de salto. La dirección de destino se calcula en tiempo de ejecución mediante una suma del registro DPTR de 16 bits y el acumulador. Por lo general, el DPTR se carga con la dirección de inicio de una tabla de salto, y el acumulador actúa como índice. Por ejemplo, si se desean cinco “casos”, se carga un valor de 0 a 4 en el acumulador y se realiza un salto al caso correspondiente, como se muestra a continuación:

```
MOV    DPTR, #TABLA_DE_SALTO
MOV    A, #NUMERO_DE_INDICE
RL     A
JMP    @A+DPTR
```

La instrucción RL A convierte el número de índice (del 0 al 4) en un número par en el rango de 0 a 8, ya que cada elemento de la tabla de salto es una dirección de 2 bytes:

```
TABLA_DE_SALTO:    AJMP CASO0
                    AJMP CASO1
                    AJMP CASO2
                    AJMP CASO3
```

EJEMPLO 3.24 Suponga que la tabla de salto antes descrita comienza en la ubicación de memoria para código 8100H con la siguiente asignación de memoria:

<u>Dirección</u>	<u>Contenido</u>
8100	01
8101	B8
8102	01
8103	43
8104	41
8105	76
8106	E1
8107	F0

- ¿Cuáles son las direcciones inicial y final del bloque de 2K de memoria para código dentro de las cuales residen estas instrucciones?
- ¿En qué dirección comienzan los casos CASO0 a CASO3?

Solución

- 8000H a 87FFH
- CASO0 comienza en la dirección 80B8H
CASO1 comienza en la dirección 8043H

CASO2 comienza en la dirección 8276H
 CASO3 comienza en la dirección 87F0H

Análisis

Este ejemplo se relaciona más con el direccionamiento absoluto que con las tablas de salto. La tabla de salto consta de una serie de cuatro instrucciones ACALL. El destino de cada salto debe estar dentro de la misma página de 2K que las instrucciones ACALL, ya que esta instrucción utiliza el direccionamiento absoluto. Una página de 2K es un bloque de memoria para código con los mismos cinco bits superiores de dirección. Los cinco bits superiores en la dirección 8100H son 10000B. Los otros 11 bits de dirección para cada rutina de CASO constan de los tres bits superiores en el código de operación ACALL y del segundo byte de la instrucción ACALL. Considere la instrucción ACALL CASO3 ubicada en las direcciones 8106H (E1H) y 8107H (F0H). Los tres bits superiores del código de operación son 111B y el segundo byte de la instrucción es 11110000B. Si concatenamos estos tres patrones binarios en el orden dado, se produce el valor 10000 111 11110000B = 87F0H, que es la dirección de la rutina CASO3.

3.3.5.2 Subrutinas e interrupciones

Existen dos variaciones de la instrucción CALL: ACALL y LCALL, las cuales utilizan direccionamiento absoluto y largo, respectivamente. Así como en la instrucción JMP, el nemónico genérico CALL puede utilizarse con el ensamblador de Intel si al programador no le importa de qué manera es codificada la dirección. Cualquiera de las instrucciones mete el contenido del contador de programa en la pila y carga este contador con la dirección especificada en la instrucción. Observe que el contador de programa (PC) contendrá la dirección de la instrucción *que sigue* de la instrucción CALL cuando se almacena en la pila. Se mete primero en la pila el byte inferior de PC, y después el byte superior. Los bytes se sacan de la pila en orden inverso. Por ejemplo, si la instrucción LCALL está en las ubicaciones de memoria para código 1000H-1002H y el SP contiene el valor 20H, entonces la instrucción LCALL (a) almacena la dirección de regreso (1003H) en la pila interna, colocando el valor 03H en 21H y el 10H en 22H; (b) deja el valor 22H en el SP; y (c) salta a la subrutina después de cargar el PC con la dirección contenida en los bytes 2 y 3 de la instrucción.

EJEMPLO 3.25

La siguiente instrucción

LCALL COSENO

está en las direcciones de memoria para código 0204H a 0206H, y la subrutina COSENO comienza en la dirección de memoria para código 043AH. Suponga que el apuntador de pila contiene el valor 3AH justo antes de que esta instrucción se ejecute. ¿Qué ubicaciones en memoria RAM interna se alteran, y cuáles son sus nuevos valores después de ejecutar la instrucción LCALL?

Solución

<u>Dirección</u>	<u>Contenido</u>
3BH	07H
3CH	02H
81H (SP)	3CH

Análisis

La instrucción que sigue de esta instrucción LCALL está en la localidad de la memoria para código 0207H, ya que es una instrucción de tres bytes. Esta es la dirección a la que el programa debe regresar al final de la subrutina y que se almacena en la pila antes de la bifurcación a la subrutina. Debido a que el apuntador de la pila contiene un valor inicial de 3AH y, ya que éste se

incrementa antes de escribir en la pila, el byte inferior de la dirección de regreso (02H) se almacena en la ubicación de memoria interna 3BH, y el byte superior de la dirección de regreso (02H) se almacena en la ubicación de memoria interna 3CH. El SP se incrementa dos veces y contiene el valor 3CH después de ejecutar la instrucción LCALL, debido a que se almacenan dos bytes en la pila. Observe que el apuntador de pila es un registro con funciones especiales, ubicado en la dirección de memoria RAM interna 81H.

Las instrucciones LCALL y ACALL tienen las mismas restricciones con respecto a la dirección de destino que las instrucciones LJMP y AJMP que vimos antes.

Las subrutinas deben terminar con una instrucción RET, la cual regresa la ejecución del programa a la instrucción que sigue de la instrucción CALL. No hay nada de magia en la forma en que la instrucción RET regresa al programa principal. Simplemente “saca” los últimos dos bytes de la pila y los coloca en el contador de programa. Una de las reglas cardinales de la programación con subrutinas es que éstas siempre deben iniciar con una instrucción CALL y siempre deben terminar con una instrucción RET. Por lo general, cuando tratamos de saltar hacia dentro o hacia fuera de una subrutina de cualquier otra manera se destruye la pila y el programa termina su ejecución en forma abrupta.

La instrucción RETI se utiliza para regresar de una rutina de servicio de interrupción (ISR). La única diferencia entre las instrucciones RET y RETI es que RETI envía señales al sistema de control de interrupciones, las cuales indican que la interrupción en proceso ha terminado. Si no hay ninguna interrupción pendiente en el momento en que se ejecuta la instrucción RETI, entonces RETI es funcionalmente idéntica a RET. En el capítulo 6 veremos las interrupciones y la instrucción RETI con mayor detalle.

EJEMPLO 3.26 El apuntador de la pila contiene el valor 1FH antes de ejecutar la instrucción

RET

al final de una subrutina. ¿Cuál es el valor del apuntador de la pila luego de que se ejecuta esta instrucción?

Solución

1CH

Análisis

La dirección de retorno de la subrutina es de 16 bits o dos bytes. El propósito de la instrucción RET es recuperar dos bytes de la pila y colocarlos en el contador de programa, con lo cual permite al programa continuar su ejecución en la ubicación siguiente a las ocupadas por la instrucción de llamada de la subrutina. Sin importar cuál pueda ser esta dirección, el apuntador de la pila contiene el valor que tenía antes de ejecutar la instrucción RET, menos dos.

EJEMPLO 3.27 Aplicación de una operación XOR sobre bits

El conjunto de instrucciones del 8051 no incluye una instrucción para realizar la operación lógica XOR sobre los valores de dos bits. Escriba una subrutina XRB que se comporte como una instrucción para realizar la operación XOR sobre dos bits en la forma XRB C,P. Esto significa que los valores de los dos bits están almacenados en C y P, respectivamente, antes de la llamada a la subrutina, y que el resultado de la operación XOR se almacenará en C.

Solución

```

XRB:      MOV 20H, C      ;copia el primer bit, x
          ANL C, /P      ;C =  $\overline{xy}$ 
          MOV 21H, C      ;copia el resultado parcial,  $\overline{xy}$ 
          MOV C, P        ;almacena el segundo bit, y en C
          ANL C, /20H     ;C =  $\overline{y}x$ 
          ORL C, 21H      ;C =  $\overline{x}x + \overline{xy}$ 

```

Análisis

Existen tres maneras distintas de resolver este problema. En este ejemplo, utilizamos el hecho de que $x \oplus y = \bar{x}y + \bar{y}x$. En la sección de problemas, dejaremos los otros dos casos como ejercicio para el lector:

- Invierta cada bit de un byte y después realice un XOR sobre los bytes
 - Utilice JB y JNB.

3.3.5.3 Saltos condicionales El 8051 ofrece una gran variedad de instrucciones para saltos condicionales. Todas estas instrucciones especifican la dirección de destino mediante el direccionamiento relativo, y por lo tanto están limitadas a una distancia de salto de -128 a +127 bytes desde la instrucción que sigue de la instrucción de salto condicional. Sin embargo, debemos tener en cuenta que el usuario especifica la dirección de destino de igual manera que en los otros saltos, como una etiqueta o una constante de 16 bits. El ensamblador se encarga del resto.

No existe en la PSW una bandera Cero. Las instrucciones JZ y JNZ prueban los datos del acumulador para esa condición.

La instrucción DJNZ (disminuye y salta si no es cero) se utiliza para control de ciclos. Para ejecutar un ciclo N veces, cargamos un contador de un byte con el valor N y terminamos el ciclo con una instrucción DJNZ apuntando al inicio del ciclo, como se muestra a continuación para $N = 10$

```
        MOV    R7,#10
CICLO:   (inicio de ciclo)
          .
          .
          .
        (fin de ciclo)
DJNZ   R7,CICLO
        (continua)
```

La instrucción CJNE (compara y salta si no es igual) se utiliza también para control de ciclos. Se especifican dos bytes en el campo del operando de la instrucción, y el salto se ejecuta sólo cuando los dos bytes no son iguales. Si, por ejemplo, acabamos de leer un carácter al acumulador del puerto serial y deseamos saltar a una instrucción identificada por la etiqueta TERMINA si el carácter es igual a CONTROL-C (03H), entonces podemos utilizar las siguientes instrucciones:

CJNE A, #03H, SALTA
SJMP TERMINA
SALTA: (continua)

Debido a que el salto ocurre solamente si $A \neq \text{CONTROL-C}$, utilizamos un salto para ignorar la instrucción de salto que terminaría la ejecución del programa, excepto cuando leemos el código del carácter deseado.

Otra aplicación de esta instrucción es para las comparaciones “mayor que” y “menor que”. Los dos bytes ubicados en el campo del operando se toman como enteros sin signo. Si el segundo byte es menor al primero, se establece en 1 la bandera de acarreo. Si el primer byte es mayor o igual al segundo, se restablece en 0 la bandera de acarreo. Por ejemplo, si deseamos saltar a la etiqueta GRANDE cuando el valor del acumulador es mayor o igual a 20H, podemos utilizar las siguientes instrucciones:

```
CJNE A, #20H, $+3
JNC GRANDE
```

El destino del salto para la instrucción CJNE está especificado como “\$ + 3”. El signo de moneda (\$) es un símbolo especial del ensamblador para representar la dirección de la instrucción actual. Ya que la instrucción CJNE es una instrucción de 3 bytes, la representación “\$ + 3” es la dirección de la siguiente instrucción, JNC. En otras palabras, la instrucción CJNE continúa la ejecución en la instrucción JNC *sin importar* el resultado de la comparación. El único propósito de la comparación es poner en 1 o 0 la bandera de acarreo. La instrucción JNC decide si el salto ocurre o no. Éste es un ejemplo donde el enfoque utilizado por el 8051, en cuanto a una situación de programación común, es más torpe que el de la mayoría de los otros microprocesadores; sin embargo, y como veremos en el capítulo 7, el uso de macros permite la construcción y ejecución de secuencias de instrucciones poderosas, tales como las del ejemplo anterior, mediante el uso de un solo nemónico.

RESUMEN

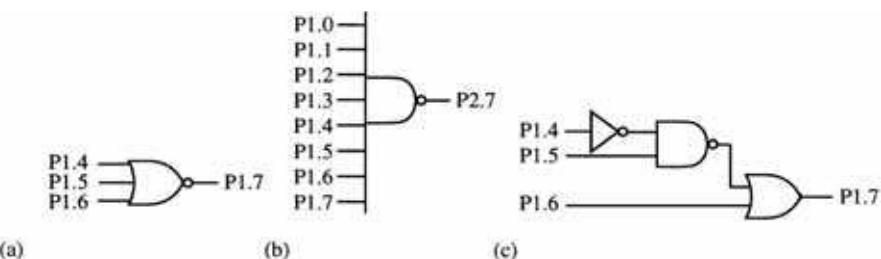
En este capítulo presentamos el conjunto de instrucciones del 8051. Sugerimos al lector la consulta del apéndice C para que obtenga más ejemplos de cómo utilizar este conjunto de instrucciones. Desde luego, nada es mejor que la práctica, así que el lector debería intentar probar tantos ejemplos de programación como le sea posible. En los siguientes tres capítulos veremos más ejemplos de programación en términos de la interacción con los periféricos incorporados al chip del 8051: los temporizadores, el puerto serial, y las interrupciones.

PROBLEMAS

- 3.1** ¿Cuáles son los bytes en hexadecimal para las siguientes instrucciones?
- INC DPTR
 - MOV A, #-2
 - MOVX @DPTR, A
 - CJNE A, #0DH, \$+3
 - PUSH ACC
 - SETB P2.2
- 3.2** ¿Cuáles son los bytes en hexadecimal para las siguientes instrucciones?
- MOV DPH, #84H
 - JNB ACC.0, \$
 - POP DPH
 - MOV A, #' = '

- e. XLR A, #'S'
 f. CLR C
- 3.3** ¿Qué instrucciones están representadas por los siguientes bytes en lenguaje de máquina?
- 7EH, 02H
 - C2H, 97H
 - 13H
 - F6H
 - 22H
 - 90H, 80H, 30H
- 3.4** ¿Qué instrucciones están representadas por los siguientes bytes en lenguaje de máquina?
- EFH
 - 12H, 80H, 50H
 - F5H, 8DH
 - 04H
 - 83H
 - 75H, 8AH, E7H
- 3.5** Enumere todas las instrucciones de 3 bytes del 8051 cuyo código de operación termina en 5H.
- 3.6** Enumere todas las instrucciones de 2 bytes del 8031 que comienzan con 2H.
- 3.7** Muestre cómo el contenido de la dirección interna 50H puede transferirse al acumulador mediante el direccionamiento indirecto.
- 3.8** Muestre dos maneras de transferir el contenido del acumulador a la dirección de memoria RAM interna 3FH.
- 3.9** ¿Cuál es el código de operación que no está definido en el 8051?
- 3.10** ¿Cuántos códigos de operación están definidos en el 8052?
- 3.11** La siguiente es una instrucción del 8051:
- ```
MOV 50H, #0FFH
```
- ¿Cuál es el código de operación para esta instrucción?
  - ¿Cuál es el tamaño en bytes de esta instrucción?
  - Explique el propósito de cada byte de esta instrucción.
  - ¿Cuánto ciclos de máquina se requieren para ejecutar esta instrucción?
  - Si un 8051 opera mediante un cristal de 16 MHz, ¿cuánto tarda en ejecutar esta instrucción?
- 3.12** La siguiente es una instrucción del 8051:
- ```
CJNE A, #'Q', ADELANTE
```
- ¿Cuál es el código de operación para esta instrucción?
 - ¿Cuál es el tamaño en bytes de esta instrucción?
 - Explique el propósito de cada byte de esta instrucción.
 - ¿Cuánto ciclos de máquina se requieren para ejecutar esta instrucción?
 - Si un 8051 opera mediante un cristal de 10 MHz, ¿cuánto tarda en ejecutar esta instrucción?
- 3.13** ¿Cuál es el desplazamiento relativo para la instrucción
- ```
SJMP ADELANTE
```
- si está ubicada en las direcciones 0400H y 0401H, y la etiqueta ADELANTE representa la instrucción en la dirección 041FH?

- 3.14** ¿Cuál es el desplazamiento relativo para la instrucción  
**SJMP ATRAS**  
 si está ubicada en las direcciones A050H y A051H, y la etiqueta ATRAS representa la instrucción en la dirección 9FE0H?
- 3.15** Suponga que la instrucción  
**AJMP ADELANTE**  
 está ubicada en la memoria para código 2FF0H y 2FF1H, y que la etiqueta ADELANTE corresponde a una instrucción en la dirección 2F96H. En lenguaje de máquina, ¿cuáles son los bytes para esta instrucción en hexadecimal?
- 3.16** Asuma que la instrucción  
**ACALL FACTORIAL**  
 está en las ubicaciones 06F4H y 06F5H de la memoria para código, y que la etiqueta FACTORIAL corresponde a una subrutina que comienza en la dirección 07ABH. En lenguaje de máquina, ¿cuáles son los bytes para esta instrucción en hexadecimal?
- 3.17** En cierto punto de un programa, deseamos saltar a la etiqueta SALIDA si el acumulador es igual al código ASCII para el retorno del carro. ¿Qué instrucción o instrucciones se utilizarían?
- 3.18** En cierto punto de un programa, deseamos saltar a la etiqueta SALIDA si el acumulador es igual al código ASCII para las letras 'Q' o 'q', o continuar si esto no ocurre. ¿Qué instrucción o instrucciones se utilizarían?
- 3.19** La instrucción  
**SJMP ATRAS**  
 está en las direcciones 0100H y 0101H de la memoria para código, y la etiqueta ATRAS corresponde a una instrucción en la dirección 00AEH. En lenguaje de máquina, ¿cuáles son los bytes para esta instrucción en hexadecimal?
- 3.20** La instrucción  
**CJNE R7, # 'Z', NO\_ES\_Z**  
 está en el rango de direcciones 022AH a 022CH de la memoria para código. En lenguaje de máquina, ¿cuáles son los bytes para esta instrucción?
- 3.21** ¿Qué ocurre al ejecutar la siguiente instrucción?  
**SETB 0D7H**  
 ¿Cuál es una mejor manera de realizar la misma operación? ¿Por qué?
- 3.22** ¿Cuál es la diferencia entre las siguientes dos instrucciones?  
**INC A**  
**INC ACC**
- 3.23** ¿Cuáles son los bytes en lenguaje de máquina para la instrucción  
**LJMP ADELANTE**  
 si la etiqueta ADELANTE representa a la instrucción en la dirección A0F6H?
- 3.24** Asuma que el acumulador A contiene el valor 5AH. ¿Cuál es el resultado en el acumulador A después de ejecutar la siguiente instrucción?  
**XRL A, #0FFH**
- 3.25** Asuma que el acumulador contiene el valor 29H. ¿Cuál es el contenido del acumulador después de ejecutar la siguiente instrucción?  
**ORL A, #47H**

**FIGURA 3-6**

Problemas de programación con compuertas lógicas. (a) NOR de 3 entradas. (b) NAND de 8 entradas. (c) Operación lógica de 3 compuertas

- 3.26** Asuma que la PSW contiene el valor 0C0H y el acumulador A el valor 50H justo antes de ejecutar la siguiente instrucción:

RLC A

¿Cuál es el contenido del acumulador A después de ejecutar la instrucción anterior?

- 3.27** Asuma que la PSW contiene el valor 78H y el acumulador el valor 81H. ¿Cuál es el contenido del acumulador después de ejecutar la siguiente instrucción?

RRC A

- 3.28** ¿Cuál es la secuencia de instrucciones que se puede utilizar para crear un pulso hacia nivel bajo de 5  $\mu$ s en el bit P1.7? Asuma que P1.7 está a nivel alto al inicio y que el 8051 opera mediante un cristal de 12 MHz.

- 3.29** Escriba un programa para crear una onda cuadrada de 83.3 kHz en P1.0. (Asuma una operación a 12 MHz).

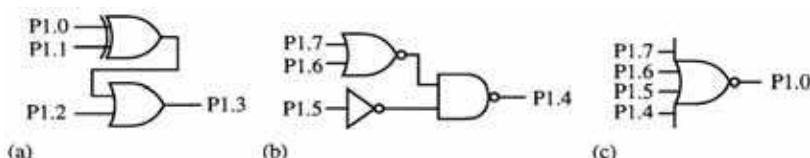
- 3.30** Escriba un programa para generar un pulso activo en alto de 4  $\mu$ s en P1.7 cada 200  $\mu$ s.

- 3.31** Escriba los programas necesarios para implementar las operaciones lógicas que se muestran en la figura 3-6. ¿Cuál es el retraso de propagación, en el peor de los casos, de una transición de entrada a una transición de salida para cada uno de estos programas? Asuma una operación a 12 MHz.

- 3.32** Escriba los programas necesarios para implementar las operaciones lógicas que se muestran en la figura 3-7. ¿Cuál es el retraso de propagación, en el peor de los casos, de una transición de entrada a una transición de salida para cada uno de estos programas? Asuma una operación a 12 MHz.

- 3.33** ¿Cuál es el contenido del acumulador A después de ejecutar la siguiente secuencia de instrucciones?

```
MOV A, #7FH
MOV 50H, #29H
```

**FIGURA 3-7**

Problemas de programación con compuertas lógicas. (a) Circuito lógico de 2 compuertas. (b) Circuito lógico de 3 compuertas. (c) NOR de 4 entradas

```
MOV R0, #50H
XCHD A, @R0
```

- 3.34** En lenguaje de máquina, ¿cuáles son los bytes requeridos para la siguiente instrucción?  
`SETB P2.6`
- 3.35** ¿Cuál es la secuencia de instrucciones que podemos utilizar para copiar la bandera 0 localizada en la PSW a la terminal de puerto P1.5?
- 3.36** ¿Bajo qué circunstancias convertirá el ensamblador de Intel (ASM51) una instrucción genérica JMP en una instrucción LJMP?
- 3.37** La memoria interna del 8051 se inicializa como se muestra a continuación, justo antes de la ejecución de una instrucción RET:

| Dirección interna | Contenido | SFRs | Contenido |
|-------------------|-----------|------|-----------|
| 0B                | 9A        | SP   | 0B        |
| 0A                | 78        | PC   | 0200      |
| 09                | 56        | A    | 55        |
| 08                | 34        |      |           |
| 07                | 12        |      |           |

¿Cuál es el contenido del PC después de ejecutar la instrucción RET?

- 3.38** Una subrutina del 8051 se muestra a continuación:
- ```
SUB:    MOV R0, #20H
        CICLO:   MOV @R0, #0
                  INC R0
                  CJNE R0, #80H, CICLO
                  RET
```
- ¿Qué hace esta subrutina?
 - ¿En cuántos ciclos de máquina se ejecuta cada instrucción?
 - ¿De qué tamaño en bytes es cada instrucción?
 - Convierta la subrutina a lenguaje de máquina.
 - ¿Cuánto tarda en ejecutarse la subrutina? (Asuma una operación a 12 MHz).
- 3.39** Un arreglo de interruptores DIP de 4 bits y un LED de 7 segmentos de ánodo común están conectados a un 8051 como se muestra en la figura 3-8. Escriba un programa que lea en forma continua un código de 4 bits del arreglo de interruptores DIP y actualice los segmen-

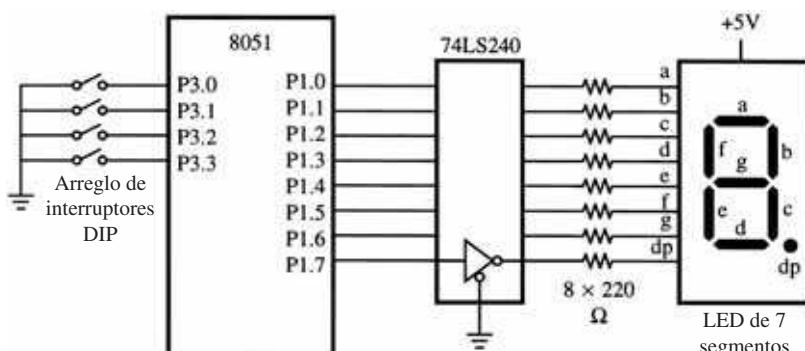


FIGURA 3-8

Interfase a un arreglo de interruptores DIP y a un LED de 7 segmentos

tos del LED para mostrar el carácter en el hexadecimal correspondiente. Por ejemplo, si leemos el código 1100B, deberá aparecer el carácter hexadecimal “C”; por lo tanto, los segmentos del *a* al *g*, respectivamente, deberán estar en el estado de ENCENDIDO, APAGADO, APAGADO, ENCENDIDO, ENCENDIDO y APAGADO. Observe que si establecemos en 1 la terminal de puerto de un 8051, cambiamos el segmento correspondiente a un estado de “ENCENDIDO”. (Consulte la figura 3-8).

- 3.40** ¿Qué tipo de instrucción de transferencia utilizaría para transferir un valor a:
- la memoria interna para datos?
 - la memoria interna para código?
 - la memoria externa para datos?
 - la memoria externa para código?
- 3.41** ¿Qué es una tabla de búsqueda? ¿Cuáles son las ventajas de su utilización?
- 3.42** ¿Cuál es la diferencia entre las siguientes dos instrucciones? Describa con todo detalle cómo es que funciona cada instrucción.

```
MOV A, @R0
MOV A, R0
```

- 3.43** Para el siguiente programa desarrollado en lenguaje ensamblador, encuentre las ubicaciones en memoria afectadas y su contenido final.

```
MOV R0, #10H
REP:    MOV @R0, #55H
        INC R0
        CJNE R0, #20H, REP
        MOV R1, #20H
CICLO:  MOV @R1, #AAH
        DEC R1
        CJNE R1, #5FH, CICLO
END
```

- 3.44** Para el siguiente programa desarrollado en lenguaje ensamblador, encuentre las ubicaciones en memoria afectadas y su contenido final.

```
MOV R0, #7FH
CICLO:  MOV @R0, #0FFH
        DEC R0
        CJNE R0, #20H, CICLO
        MOV R1, #30H
SIGUIENTE: MOV @R1, #00H
        INC R1
        CJNE R1, #5FH, SIGUIENTE
END
```

- 3.45** Suponga que las siguientes ubicaciones en memoria contienen los valores iniciales como sigue:

30H	55H
78H	00H
7FH	FFH

¿Cuál es el contenido de las ubicaciones en memoria 30H, 78H y 7FH después de la ejecución de cada una de las siguientes instrucciones? ¿Por qué? (Asuma que cada instrucción es independiente).

- a. CPL 7FH
 - b. CLR 78H
 - c. MOV 7FH, 78H
- 3.46** Escriba un programa en lenguaje ensamblador para sumar todos los números en la memoria RAM interna que se ubican en las direcciones localizadas dentro del rango de 30H a 6FH. Almacene el resultado en la ubicación en memoria RAM interna 70H. Escriba comentarios para cada línea de código.
- 3.47** La operación factorial (denotada por el símbolo '!') se encuentra con frecuencia en las operaciones matemáticas, especialmente en cálculos de probabilidad. Por ejemplo, $5! = 5 \times 4 \times 3 \times 2 \times 1$. Escriba un programa en lenguaje ensamblador apropiado para calcular la operación factorial de un número almacenado en la ubicación de memoria RAM 55H. Coloque el resultado de la operación en la ubicación de memoria RAM 77H.
- 3.48** El acumulador A contiene un número x de 4 bits (excluyendo al número 0). Escriba un programa que utilice una tabla de búsqueda para encontrar el resultado de $20 \log_{10} x$. Redondee sus cálculos a enteros.
- 3.49** Escriba un programa en lenguaje ensamblador utilizando tablas de búsqueda para calcular la función exponencial e^x o de un valor x en el acumulador. El resultado (redondeando al entero más cercano) debe regresarse en R1 (byte superior) y R0 (byte inferior). Por ejemplo, si el acumulador contiene el valor 2, su resultado deberá ser R1 = 0 y R0 = 7.
- 3.50** Escriba un programa en lenguaje ensamblador para realizar la multiplicación de dos números en R0 y R1. ¡NO UTILICE la instrucción MUL AB! Use otras instrucciones. El byte de orden superior del resultado deberá colocarse en R3 mientras que el byte de orden inferior deberá colocarse en R2.
- 3.51** ¿Qué son las subrutinas? Explique las ventajas de la utilización de subrutinas en sus programas desarrollados en lenguaje ensamblador.
- 3.52** El término *subrutinas anidadas* se refiere al uso de dos o más subrutinas, en donde cada subrutina llama a la otra.
 - a. Escriba una subrutina llamada POTENCIA que calcule el resultado de un número en el acumulador A elevado a la potencia de un número almacenado en el registro B. Por ejemplo, si $A = 2$ y $B = 3$, entonces su resultado deberá ser $2^3 = 8$. Almacene el resultado de 16 bits en el acumulador A (para el byte inferior) y el registro B (para el byte superior).
 - b. Después escriba una subrutina llamada CALCULA que determine el valor de $3^4 - 2^3$ y almacene el resultado de 16 bits en A (byte inferior) y B (byte superior). Su subrutina deberá llamar a la subrutina POTENCIA para realizar los cálculos de potencia (elevando un número a la potencia de otro).
- 3.53** Escriba una subrutina llamada SUMA que calcule la suma de dos números contenidos en el acumulador A y en el registro B, y que regrese el resultado al acumulador A. Escriba también otra subrutina llamada TOTAL que determine la suma de tres números en R1, R2 y R3. Llame a la subrutina SUMA desde la subrutina TOTAL para realizar las operaciones en lugar de utilizar la instrucción ADD. Almacene la suma total en R4.
- 3.54** El 8051 cuenta con una instrucción XCHD para intercambiar los nibbles de orden inferior de dos valores. Escriba una subrutina XCHH que intercambie los nibbles de orden superior de los dos valores en A y B.

- 3.55** Escriba una subrutina XRB para calcular el XOR de dos bits en C y P. Almacene el resultado en C. Escriba comentarios para cada línea de código.
- 3.56** La siguiente subrutina ADIVINAME puede utilizarse para realizar una operación muy útil que las instrucciones incorporadas en el 8051 no proporcionan.
- Escriba comentarios para cada línea de código.
 - Explique sobre lo que usted cree es la función de esta subrutina.

```
ADIVINAME: CLR    C
           RRC    A
           DJNZ   R0 ,ADIVINAME
           END
```

- 3.57** Escriba un programa en lenguaje ensamblador para realizar la división de dos números en R0 y R1. El cociente deberá ser almacenado en A y el resto en B. ¡NO UTILICE la instrucción DIV AB!
- 3.58** En el ejemplo 3.27 consideramos una subrutina XRB que se comporta tal y como una instrucción para realizar un XOR con dos bits de la forma XRB C,P. Esto significa que los dos bits están almacenados en C y P antes de la llamada a la subrutina, y que el resultado del XOR deberá almacenarse en C. Escriba la subrutina de nuevo para realizar un XOR con dos bits basándose en los siguientes métodos:
- Cambie cada bit a un byte y realice una operación de XOR sobre los bytes
 - Utilice las instrucciones JB y JNB
- 3.59** Las ubicaciones en memoria interna en el rango de 30H a 39H contienen los números del 0 al 9, respectivamente. Escriba las instrucciones necesarias en lenguaje ensamblador para invertir el orden en que los números están almacenados: 0 se almacena en 39H, 1 en 38H, etc.
(Sugerencia: utilice las instrucciones PUSH y POP).
- 3.60** Escriba las instrucciones necesarias en lenguaje ensamblador para sumar dos números de 16 bits.

Operación de los temporizadores

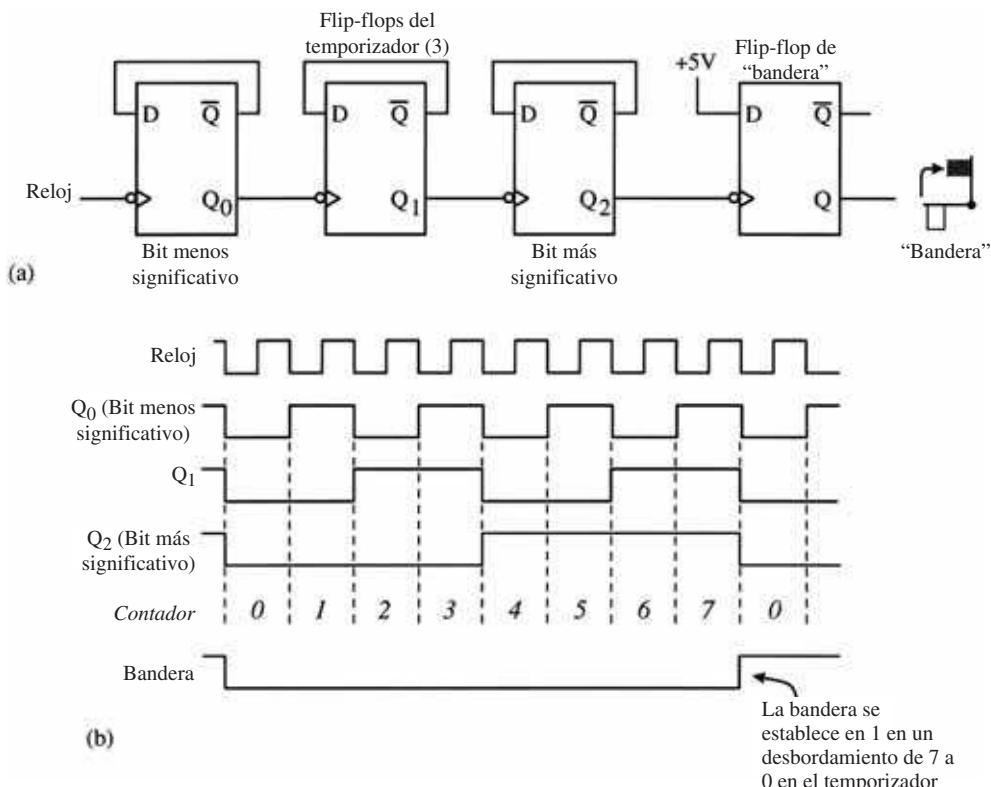
4.1 INTRODUCCIÓN

En este capítulo examinaremos los temporizadores incorporados en el chip del 8051. Empezaremos con una vista simplificada de los temporizadores y su utilización más común en microprocesadores o microcontroladores.

Un temporizador consta de una serie de flip-flops de división entre 2, los cuales reciben una señal de entrada como su fuente de reloj. El reloj se aplica al primer flip-flop, el cual divide la frecuencia del reloj entre 2. La salida del primer flip-flop se aplica a la entrada de reloj del segundo flip-flop, que también divide la frecuencia entre 2, y así sucesivamente. Un temporizador con un número n de etapas divide la frecuencia del reloj de entrada entre 2^n , ya que cada etapa sucesiva divide entre 2. La salida de la última etapa se aplica a la entrada de reloj de un flip-flop de desbordamiento del temporizador, o **flag**, cuyo estado puede ser verificado mediante el software y su establecimiento en 1 puede generar una interrupción. Podemos considerar al valor binario de las salidas de los flip-flops del temporizador como la “cuenta” del número de pulsos del reloj (o “eventos”) desde que el temporizador se inició. Por ejemplo, un temporizador de 16 bits contaría desde 0000H hasta FFFFH. La bandera de desbordamiento se establecería en 1 en el desbordamiento de FFFFH a 0000H del conteo.

La figura 4-1 ilustra una operación básica para un temporizador de 3 bits. Cada etapa se muestra como un flip-flop tipo D con disparo en flanco negativo, el cual opera en el modo de división entre 2 (en otras palabras, la salida \bar{Q} se conecta a la entrada D). El flip-flop de la bandera es un simple “latch” tipo D, que se activa mediante la última etapa en el temporizador. Como resulta evidente en el diagrama de sincronización de la figura 4-1b, la primera etapa Q_0 se dispara a la mitad de la frecuencia del reloj, la segunda etapa a un cuarto de la frecuencia del reloj, y así sucesivamente. La cuenta se muestra en notación decimal y es fácil de verificar si examinamos el estado de los tres flip-flops. Por ejemplo, la cuenta “4” ocurre cuando $Q_2 = 1$, $Q_1 = 0$, y $Q_0 = 0$ ($4_{10} = 100_2$).

Los temporizadores se utilizan en casi todas las aplicaciones orientadas al control, y los del 8051 no son la excepción. El 8051 tiene dos temporizadores de 16 bits, cada uno con cuatro modos de operación. Un tercer temporizador de 16 bits con tres modos de operación se ha añadido al 8052. Los temporizadores se utilizan para (a) la temporización de intervalos, (b) el

**FIGURA 4–1**

Un temporizador de 3 bits. (a) Diagrama esquemático. (b) Diagrama de temporización

conteo de eventos, o (c) la generación de la tasa de transmisión y recepción en baudios para el puerto serial incorporado. Cada temporizador es de 16 bits, por lo que la decimosexta o última etapa divide la frecuencia del reloj de entrada entre $2^{16} = 65,536$.

En las aplicaciones de temporización de intervalos, un temporizador se programa para causar un desbordamiento a intervalos regulares, lo cual establece en 1 la bandera de desbordamiento, misma que se utiliza para sincronizar el programa para que éste realice una acción tal como verificar el estado de las entradas o enviar datos a las salidas. Otras aplicaciones pueden utilizar la capacidad del temporizador de recibir pulsos a intervalos regulares para medir el tiempo transcurrido entre dos condiciones (por ejemplo, medidas de la anchura de un pulso).

El conteo de eventos se utiliza para determinar cuántas veces ocurre un evento, más que para medir el tiempo transcurrido entre varios eventos. Un “evento” es cualquier estímulo externo que provee una transición de 1 a 0 a una terminal en el circuito integrado del 8051. Los temporizadores también pueden proporcionar el reloj de la tasa de transmisión y recepción en baudios para el puerto serial interno del 8051.

Para acceder a los temporizadores del 8051 se utilizan seis registros de funciones especiales (SFR). (Consulte la tabla 4-1). Cinco SFR adicionales proporcionan acceso a un tercer temporizador en el 8052.

TABLA 4-1

Registros con funciones especiales de un temporizador

SFR del temporizador	Propósito	Dirección	Direccionable por bit
TCON	Control	88H	Sí
TMOD	Modo	89H	No
TL0	Byte inferior del temporizador 0	8AH	No
TL1	Byte inferior del temporizador 1	8BH	No
TH0	Byte superior del temporizador 0	8CH	No
TH1	Byte superior del temporizador 1	8DH	No
T2CON*	Control del temporizador 2	C8H	Sí
RCAP2L*	Captura del byte inferior del temp. 2	CAH	No
RCAP2H*	Captura del byte superior del temp 2	CBH	No
TL2*	Byte inferior del temporizador 2	CCH	No
TH2*	Byte superior del temporizador 2	CDH	No

*Sólo en el 8032/8052

4.2 REGISTRO DE MODO DEL TEMPORIZADOR (TMOD)

El registro TMOD contiene dos grupos de cuatro bits que establecen el modo de operación para los temporizadores 0 y 1. (Consulte las tablas 4-2 y 4-3).

Los bits de TMOD no son direccionables individualmente ni requieren serlo. Por lo general, el registro se carga una sola vez mediante el software al inicio de un programa para inicializar el modo del temporizador. Después de esto, el temporizador puede detenerse, iniciarse, etc., por acceso a otros SFR del temporizador.

4.3 REGISTRO DE CONTROL DEL TEMPORIZADOR (TCON)

El registro TCON contiene los bits de estado y de control para los temporizadores 0 y 1 (consulte la tabla 4-4). Los cuatro bits superiores en el TCON (TCON.4-TCON.7) se utilizan para encender o apagar los temporizadores (TR0, TR1), o para indicar el desbordamiento del temporizador (TF0, TF1). En los ejemplos de este capítulo haremos uso extenso de estos bits.

TABLA 4-2

Resumen del registro TMOD (modo del temporizador)

Bit	Nombre	Temporizador	Descripción
7	GATE	1	Bit de compuerta. Cuando es 1, el temporizador sólo funciona si la señal INTI está a nivel alto
6	C/T	1	Bit de selección de contador/temporizador 1 = contador de eventos 0 = temporizador de intervalos
5	M1	1	Bit de modo 1 (consulte la tabla 4-3)
4	M0	1	Bit de modo 0 (consulte la tabla 4-3)
3	GATE	0	Bit de compuerta del temporizador 0
2	C/T	0	Bit de selección de contador/temporizador del temporizador 0
1	M1	0	Bit M1 del temporizador 0
0	M0	0	Bit M0 del temporizador 0

TABLA 4-3
Modos del temporizador

M1	M0	Modo	Descripción
0	0	0	Modo de temporizador de 13 bits (modo del 8048)
0	1	1	Modo de temporizador de 16 bits
1	0	2	Modo de autorrecarga de 8 bits
1	1	3	Modo de temporizador dividido: Temporizador 0: TL0 es un temporizador de 8 bits controlado por los bits de modo del temporizador 0; TH0, lo mismo que TL0 excepto que está controlado por los bits de modo del temporizador 1 Temporizador 1: detenido

Los cuatro bits inferiores en TCON (TCON.0-TCON.3) no tienen nada que ver con los temporizadores. Éstos se utilizan para detectar e iniciar interrupciones externas. Hablaremos sobre estos bits en el capítulo 6, donde también veremos las interrupciones.

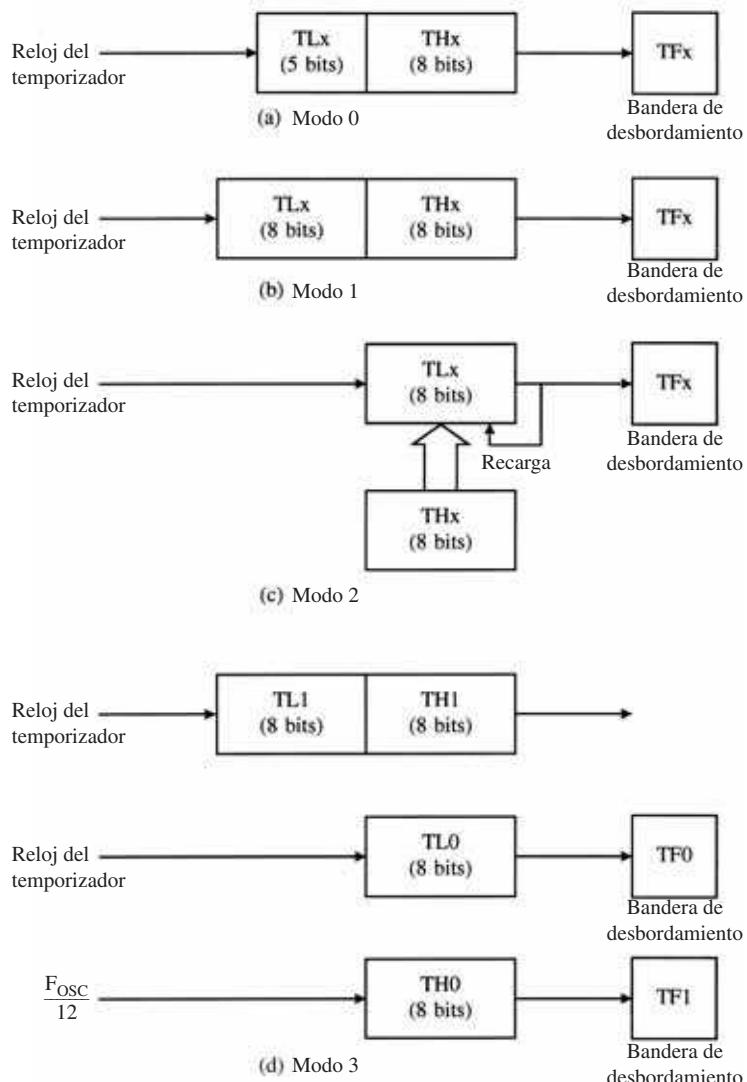
4.4 MODOS DEL TEMPORIZADOR Y LA BANDERA DE DESBORDAMIENTO

A continuación veremos cada uno de los temporizadores. Utilizamos la notación “x” para referirnos indistintamente a los temporizadores 0 o 1, ya que existen dos temporizadores en el 8051; por lo tanto, “THx” significa ya sea TH1 o TH0, dependiendo del temporizador.

La figura 4-2 muestra el arreglo de los registros del temporizador TLx y THx y las banderas de desbordamiento del temporizador TFx para cada modo.

TABLA 4-4
Resumen del registro TCON (control del temporizador)

Bit	Símbolo	Dirección de bit	Descripción
TCON.7	TF1	8FH	Bandera de desbordamiento del temporizador 1. Es puesto en 1 mediante el hardware cuando ocurre el desbordamiento; es puesto en 0 por el software o por el hardware cuando el procesador se vectoriza a la rutina de servicio de la interrupción correspondiente
TCON.6	TR1	8FH	Bit de control de ejecución del temporizador 1. Es puesto en 1/0 mediante el software para encender/apagar el temporizador
TCON.5	TF0	8DH	Bandera de desbordamiento del temporizador 0
TCON.4	TR0	8CH	Bit de control de ejecución del temporizador 0
TCON.3	IE1	8BH	Bandera de flanco de la interrupción externa 1. Es puesto en 1 mediante el hardware cuando se detecta un flanco descendente en $\overline{INT1}$; es puesto en 0 mediante el software o el hardware cuando la CPU se vectoriza a la rutina de servicio de la interrupción correspondiente
TCON.2	IT1	8AH	Bandera de tipo de la interrupción externa 1. Es puesto en 1/0 por el software para activar la interrupción externa por flanco descendente/nivel bajo
TCON.1	IE0	89H	Bandera de borde de la interrupción externa 0
TCON.0	IT0	88H	Bandera de tipo de la interrupción externa 0

**FIGURA 4-2**

Modos del temporizador. (a) Modo 0. (b) Modo 1. (c) Modo 2. (d) Modo 3

4.4.1 Modo de temporizador de 13 bits (Modo 0)

El modo 0 es un modo de temporizador de 13 bits que provee compatibilidad con el predecesor del 8051, el 8048. Éste no se utiliza generalmente en los nuevos diseños. (Consulte la figura 4-2a). El byte superior del temporizador (THx) está dispuesto en cascada con los cinco bits menos significativos del byte inferior del temporizador (TLx) para formar un temporizador de 13 bits. Los tres bits superiores del TLx no son utilizados.

4.4.2 Modo de temporizador de 16 bits (Modo 1)

El modo 1 es un modo de temporizador de 16 bits y es igual al modo 0, excepto que el temporizador opera como temporizador de 16 bits. El reloj se aplica a los registros superior e inferior combinados (TLx/THx). El contador contará hacia arriba mientras se reciban los pulsos del reloj: 0000H, 0001H, 0002H, etc. En la transición de FFFFH a 0000H del conteo ocurre un desbordamiento que establece en 1 la bandera de desbordamiento del temporizador. El temporizador sigue contando. La bandera de desbordamiento es el bit TFx en TCON, el cual se lee o escribe mediante el software. (Consulte la figura 4-2b).

El bit más significativo (MSB) del valor en los registros del temporizador es el bit 7 de THx, y el bit menos significativo (LSB) es el bit 0 de TLx. El LSB se dispara a una frecuencia de reloj de entrada dividida entre 2, mientras que el MSB se dispara a una frecuencia de reloj de entrada dividida entre 65,536 (en otras palabras, 2^{16}). Los registros del temporizador (TLx/THx) pueden leerse o escribirse mediante el software cuando sea necesario.

4.4.3 Modo de autorrecarga de 8 bits (Modo 2)

El modo 2 es un modo de autorrecarga de 8 bits. El byte inferior del temporizador (TLx) opera como temporizador de 8 bits mientras que el byte superior (THx) almacena un valor de recarga. Cuando la cuenta se desborda después de FFH, esto no sólo establece en 1 la bandera de desbordamiento del temporizador, también se carga el valor en THx a TLx; la cuenta sigue desde este valor hasta llegar al siguiente desbordamiento de FFH, y así sucesivamente. Este modo es muy útil ya que el desbordamiento del temporizador ocurre a intervalos periódicos específicos una vez que TMOD y THx se han inicializado. (Consulte la figura 4-2c). Si el TLx contiene el valor 4FH, por ejemplo, el temporizador cuenta continuamente desde el valor 4FH hasta el valor FFH.

4.4.4 Modo de temporizador dividido (Modo 3)

El modo 3 es un modo de temporizador dividido y es distinto para cada temporizador. El temporizador 0 en el modo 3 se divide en dos temporizadores de 8 bits. TL0 y TH0 actúan como temporizadores separados, en donde los desbordamientos establecen en 1 los bits TF0 y TF1, respectivamente.

En el modo 3 el temporizador 1 está detenido, pero puede iniciarse al cambiarlo a uno de los otros modos. La única limitación es que la bandera de desbordamiento del temporizador 1, TF1, no se ve afectada por los desbordamientos del temporizador 1, ya que TF1 está conectada a TH0.

En esencia, el modo 3 proporciona un temporizador de 8 bits adicional: el 8051 parece tener un tercer temporizador. Cuando el temporizador 0 está en el modo 3, el temporizador 1 puede encenderse y apagarse al habilitar o deshabilitar su modo 3. El temporizador 1 todavía puede ser utilizado por el puerto serial como un generador de la tasa de transmisión y recepción en baudios o emplearse en cualquier forma que no requiera de interrupciones (pues ya no estará conectado a TF1).

4.5 FUENTES DE RELOJ

La figura 4-2 no muestra cómo reciben los temporizadores la señal de reloj. Existen dos fuentes posibles de reloj, las cuales se seleccionan mediante la escritura al bit de contador/temporizador (C/T) en TMOD cuando se inicializa el temporizador. Una fuente de reloj se utiliza para la temporización de intervalos y la otra para conteo de eventos.

4.5.1 Temporización de intervalos

Si C/T = 0, se selecciona la operación continua del temporizador y éste recibe la señal de reloj del oscilador incorporado en el chip. Se agrega una etapa de división entre 12 para reducir la frecuencia del reloj a un valor razonable para la mayoría de las aplicaciones.

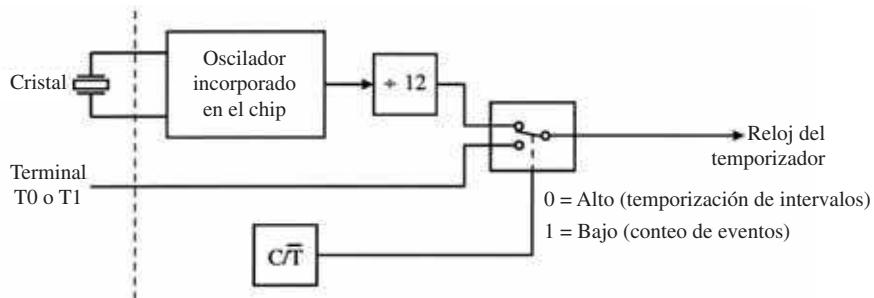


FIGURA 4-3
Fuente de reloj

El temporizador se utiliza para la **temporización de intervalos** cuando seleccionamos la operación continua del temporizador. Los registros del temporizador (TLx/THx) se incrementan a una velocidad equivalente a una doceava parte de la frecuencia del oscilador incorporado en el chip; por lo tanto, un cristal de 12 MHz daría una velocidad de reloj de 1 MHz. Los desbordamientos del temporizador ocurren después de un número fijo de pulsos de reloj, dependiendo del valor inicial cargado en los registros del temporizador, TLx/THx.

4.5.2 Conteo de eventos

El temporizador recibe la señal de reloj mediante una fuente externa si $C/\bar{T} = 1$. En la mayoría de las aplicaciones, esta fuente externa proporciona un pulso al temporizador cuando ocurre uno de estos “eventos”; el temporizador está realizando una operación de **conteo de eventos**. El número de eventos está determinado en el software mediante la lectura de los registros del temporizador TLx/THx, ya que en estos registros de 16 bits el valor se incrementa por cada evento.

La fuente de reloj externa está disponible mediante las funciones alternas de las terminales del puerto 3. El bit 4 del puerto 3 (P3.4) sirve como la entrada externa de reloj para el temporizador 0, y se conoce como “T0” en este contexto. P3.5, o “T1”, es la entrada de reloj para el temporizador 1. (Consulte la figura 4-3).

En aplicaciones de contador, los registros del temporizador se incrementan en respuesta a la transición de 1 a 0 de la entrada externa, Tx. La entrada externa se muestrea durante la etapa S5P2 de cada ciclo de máquina; por lo tanto, cuando la entrada muestra un nivel alto en un ciclo y un nivel bajo en el siguiente ciclo, la cuenta se incrementa. Este nuevo valor aparece en los registros del temporizador durante la etapa S3P1 del ciclo que sigue al ciclo en que se detectó la transición. La frecuencia externa máxima es de 500 kHz (si asumimos una operación a 12 MHz), ya que se necesitan dos ciclos de máquina $2 \mu s$ para reconocer una transición de 1 a 0.

4.6 INICIO, DETENCIÓN Y CONTROL DE LOS TEMPORIZADORES

La figura 4-2 ilustra las diferentes configuraciones de los registros del temporizador, TLx y THx, y las banderas de desbordamiento del temporizador, TFx. La figura 4-3 muestra los dos posibles modos de generación de la señal de reloj de los temporizadores. Demostraremos a continuación cómo iniciar, detener y controlar los temporizadores.

El método más simple para iniciar y detener los temporizadores es mediante el bit de control de arranque, TRx, en TCON. TRx se pone en 0 después de una reinicialización del sistema; por lo tanto, los temporizadores están deshabilitados (detenidos) por defecto. TRx se establece en 1 mediante el software para iniciar los temporizadores. (Consulte la figura 4-4).

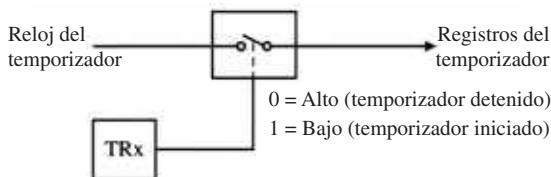


FIGURA 4–4
Inicio y detención de los temporizadores

Es fácil iniciar y detener los temporizadores mediante un programa, ya que TR_x está en el registro direccionable por bit TCON. Por ejemplo, el temporizador 0 se inicia mediante

SETB TR0

y se detiene con

CLR TR0

El ensamblador realizará la conversión simbólica necesaria de “TR0” a la dirección de bit correcta. SETB TR0 es exactamente la misma instrucción que SETB 8CH.

Otro método para controlar los temporizadores es a través del bit de GATE (compuerta) en TMOD y la entrada externa INT_x. Al establecer la señal GATE = 1, el temporizador puede controlarse mediante la señal INT_x. Esto es útil para obtener medidas de la anchura de un pulso, como veremos a continuación. Suponga que INT0 está a nivel bajo, pero cambia mediante un pulso a un nivel alto durante cierto período a medir. Inicialice el temporizador 0 en el modo 1, el modo de temporizador de 16 bits, con TL0/TH0 = 0000H, GATE = 1, y TR0 = 1. Cuando

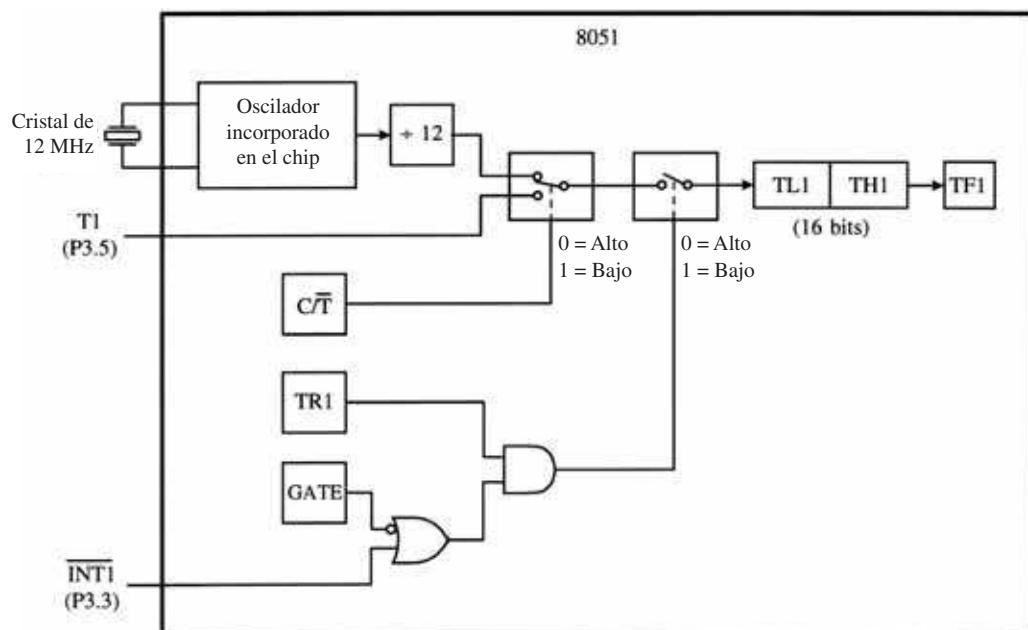


FIGURA 4–5

El temporizador 1 operando en el modo 1

INT0 cambia al nivel alto, el temporizador está en estado de “compuerta abierta” y recibe una señal de reloj a una tasa de 1 MHz. El temporizador está en estado de “compuerta cerrada” cuando la señal INT0 cambia al nivel bajo y la duración del pulso en microsegundos es la cuenta en TL0/TH0. (INT0 puede ser programada para generar una interrupción cuando regresa al nivel bajo).

La figura 4-5 ilustra al temporizador 1 operando en el modo 1 como un temporizador de 16 bits. El diagrama muestra, además de los registros del temporizador TL1/TH1 y la bandera de desbordamiento TF1, las posibilidades para la fuente de reloj y para iniciar, detener y controlar el temporizador.

- EJEMPLO 4.1** La figura 4-5 ilustra al temporizador 1 operando en el modo 1. Estudie la figura e identifique los registros del temporizador y los bits de control del 8051 mostrados. Construya una tabla de las direcciones de bit y de byte para cada registro y bit de control. Para los bits de control, identifique los registros con funciones especiales que los almacenan.

Solución

Registros del temporizador:

TH1 en la dirección de byte 8DH

TL1 en la dirección de byte 8BH

Bits de control/modo del temporizador:

TR1 en la dirección de bit 8EH (dentro de TCON)

TF1 en la dirección de bit 8FH (dentro de TCON)

C/T en el bit 6 en TMOD (dirección 88H)

GATE en el bit 7 en TMOD (dirección 88H)

Análisis

De los cuatro bits de control/modo mostrados, sólo TR1 y TF1 son direccionables directamente. A menudo, estos bits son puestos en 1 y 0 durante la ejecución para iniciar y detener al temporizador o para verificar su estado, según sea necesario. Por lo general, se escribe en los bits de C/T y GATE sólo una vez, al inicio de un programa para establecer el modo de operación del temporizador.

4.7 INICIALIZACIÓN Y ACCESO A LOS REGISTROS DEL TEMPORIZADOR

Por lo general, los temporizadores se inicializan una sola vez al comienzo de un programa para establecer el modo de operación correcto. Luego, dentro del cuerpo de un programa, los temporizadores se inician o detienen, los bits de bandera se prueban y son puesto en 0, los registros del temporizador se leen o actualizan, etc., según los requerimientos de la aplicación.

TMOD es el primer registro que se inicializa, ya que establece el modo de operación. Por ejemplo, la siguiente instrucción inicializa al temporizador 1 como un temporizador de 16 bits (modo 1) recibiendo la señal de reloj a partir del oscilador incorporado en el chip (temporización de intervalos):

```
MOV TMOD, #00010000B
```

El efecto de esta instrucción es que establece a M1 = 0 y M0 = 1 para el modo 1, deja la señal C/T = 0 y GATE = 0 para la generación de reloj interna, y restablecer a 0 los bits de modo del temporizador 0. (Consulte la tabla 4-2). Desde luego, el temporizador no empieza a funcionar sino hasta que su bit de control de arranque, TR1, es 1.

Cuando se requiere partir de una cuenta inicial, los registros del temporizador TL1/TH1 también deben inicializarse. Recuerde que los temporizadores cuentan hacia arriba y establecen en 1 la bandera de desbordamiento en una transición de FFFFH a 0000H; y, por lo tanto, un intervalo de 100 μ s puede obtenerse mediante la inicialización de TL1/TH1 a 100 conteos menos que 0000H. El valor correcto es –100 o FF9CH. Las siguientes instrucciones realizan esta operación:

```
MOV TL1, #9CH
MOV TH1, #0FFH
```

El temporizador se inicia mediante el establecimiento en 1 del bit de control de arranque, como se muestra enseguida:

```
SETB TR1
```

La bandera de desbordamiento se establece en 1 en forma automática 100 μ s después. El software puede mantenerse en un “ciclo de espera” durante 100 μ s utilizando una instrucción de bifurcación condicional que regresa a sí misma siempre que la bandera de desbordamiento no sea 1:

```
ESPERA:     JNB    TF1, ESPERA
```

Cuando el temporizador se desborda, es necesario detenerlo y restablecer a 0 la bandera de desbordamiento mediante el software:

```
CLR TR1
CLR TF1
```

4.7.1 Lectura de un temporizador “al instante”

En algunas aplicaciones es necesario leer el valor de los registros del temporizador “al instante”. Existe un problema potencial del que es posible salvaguardarse mediante el software. Ya que se requiere leer dos registros del temporizador, puede ocurrir un “error de fase” cuando el byte inferior se desborda hacia el byte superior, en medio de las dos operaciones de lectura. Podría leerse un valor que nunca existió. La solución es leer el byte superior primero, luego el byte inferior, y después el byte superior de nuevo. Si el byte superior ha cambiado, se repiten las dos operaciones de lectura. Las siguientes instrucciones leen el contenido de los registros del temporizador TL1/TH1 en los registros R6/R7, lo cual resuelve adecuadamente este problema.

```
DENUEVO: MOV A, TH1
          MOV R6, TL1
          CJNE A, TH1, DENUEVO
          MOV R7, A
```

4.8 INTERVALOS CORTOS, MEDIANOS Y LARGOS

¿Cuál es el rango de intervalos que pueden temporizarse? Examinemos este problema asumiendo que el 8051 opera mediante un cristal de 12 MHz. El oscilador incorporado en el chip se divide entre 12 y entonces los temporizadores reciben una señal de reloj con una frecuencia de 1 MHz.

El intervalo más corto posible está limitado no por la frecuencia del reloj del temporizador, sino por el software. Supuestamente, algo debe ocurrir a intervalos regulares, y es la duración de las instrucciones lo que limita esto a intervalos muy cortos. La instrucción más corta en el 8051 es un ciclo de máquina o un microsegundo. La tabla 4-5 resume las técnicas utilizadas para crear intervalos con diversas duraciones. (Se asume una operación mediante un cristal de 12 MHz).

TABLA 4-5

Técnicas de programación para intervalos medidos (operación a 12 MHz)

Intervalo máximo en microsegundos	Técnica
≈10	Ajuste del software
256	Temporizador de 8 bits con autorrecarga
65536	Temporizador de 16 bits
Sin límite	Temporizador de 16 bits además de ciclos en software

EJEMPLO**4.2****Generación de onda pulsante**

Escriba un programa para crear una forma de onda periódica en P1.0 con la frecuencia más alta posible. ¿Cuáles son la frecuencia y el ciclo de trabajo de la forma de onda?

Solución

```

8100      5      ORG    8100H
8100 D290  6      CICLO: SETB   P1.0 ;un ciclo de máquina
8102 C290  7      CLR    P1.0 ;un ciclo de máquina
8104 80FA  8      SJMP   CICLO ;dos ciclos de máquina
         9      END

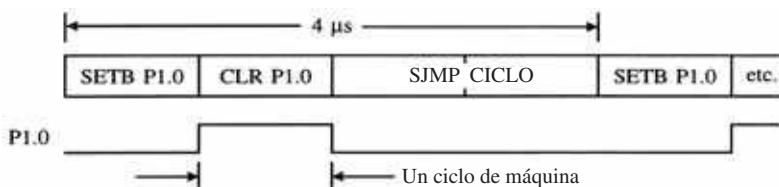
```

Análisis

Este programa crea una forma de onda pulsante en P1.0 con periodo de $4 \mu\text{s}$ y frecuencia de 250 kHz. En cada ciclo, la señal está a nivel alto durante $1 \mu\text{s}$ y a nivel bajo durante $3 \mu\text{s}$. Esto corresponde a un ciclo de trabajo de $1/4 = 0.25$ o 25% (consulte la figura 4-6).

En primera instancia, puede parecer que las instrucciones de la figura 4-6 no están en el lugar correcto, pero no es así. Por ejemplo, la instrucción SETB P1.0 no vuelve 1 el bit del puerto sino hasta el final de la instrucción, durante la etapa S6P1.

El periodo de la forma de onda puede incrementarse al introducir una instrucción NOP al ciclo. Cada instrucción NOP añade 1 ciclo de máquina o $1 \mu\text{s}$ al periodo de la forma de onda. Por ejemplo, si añadiésemos dos instrucciones NOP después de la instrucción SETB P1.0, caeríamos que la salida fuese una onda cuadrada (con ciclo de trabajo = 50%) con periodo de $6 \mu\text{s}$ y frecuencia de 166.7 kHz. Después de cierto punto, el ajuste del software resulta un tanto incómodo, y un temporizador es la mejor alternativa para crear retrasos de tiempo.

**FIGURA 4-6**

Forma de onda para el ejemplo 4.2

EJEMPLO Generación de una onda cuadrada

- 4.3** Escriba un programa para crear una onda cuadrada en P1.0 con la frecuencia más alta posible. ¿Cuál es la frecuencia y cuál el ciclo de trabajo de la forma de onda?

Solución

```

8100      5          ORG 8100H
8100 B290  6          CICLO: CPL P1.0 ;un ciclo de máquina
8102 80FC  7          SJMP LOOP ;dos ciclos de máquina
                           8          END

```

Análisis

El periodo es de $6 \mu s$: tiempo a nivel alto = tiempo a nivel bajo = $3 \mu s$. La frecuencia es de 166.67 kHz y el ciclo de trabajo es del 50%, así que ésta es una onda cuadrada a diferencia del ejemplo anterior, que no lo era.

Obtener intervalos de duración moderada es fácil utilizando el modo de autorrecarga de 8 bits, el modo 2. El intervalo más largo posible antes del desbordamiento es de $2^8 = 256 \mu s$, ya que el intervalo medido se establece mediante un conteo de 8 bits.

EJEMPLO Onda cuadrada de 10 kHz

- 4.4** Escriba un programa utilizando el temporizador 0 para crear una onda cuadrada de 10 kHz en P1.0.

Solución

```

8100      6          ORG 8100H
8100 758902 7          MOV TMOD,#02H ;modo de autorrecarga de 8 bits
8103 758CCE 8          MOV TH0,#-50 ;valor de recarga en TH0 de -50
8106 D28C   9          SETB TR0 ;inicia temporizador
8108 308DFD 10 CICLO: JNB TF0,CICLO ;espera al desbordamiento
810B C28D   11         CLR TF0 ;restablece en 0 la bandera de
                           desbordamiento del temporizador
810D B290   12         CPL P1.0 ;complementa bit de puerto
810F 80F7   13         SJMP CICLO ;repite
                           14         END

```

Análisis

El programa anterior crea una onda cuadrada en P1.0 con un tiempo a nivel alto de $50 \mu s$ y un tiempo a nivel bajo de $50 \mu s$. Podemos utilizar el modo 2 del temporizador, ya que el intervalo es menor que $256 \mu s$. Un desbordamiento cada $50 \mu s$ requiere un valor de recarga en TH0 de 50 conteos menos que 00H, o -50 .

El programa utiliza una instrucción de complemento a bit (CPL, línea 12) en lugar de SETB y CLR. Programamos un retraso de 1/2 del periodo deseado $50 \mu s$ entre cada operación de complemento utilizando el temporizador 0 en el modo de autorrecarga de 8 bits. El valor de recarga se especifica utilizando la notación decimal como el valor 250 (línea 8) en lugar de la notación hexadecimal. El ensamblador realiza la conversión necesaria. Observe que la bandera de desbordamiento del temporizador (TF0) se restablece en 0 en forma explícita en el software después de cada desbordamiento (línea 11).

Los intervalos medidos con duración más larga que $256 \mu s$ deben utilizar el modo de temporizador de 16 bits, modo 1. El retraso más largo es de $2^{16} = 65,536 \mu s$ o alrededor de 0.066

segundos. Lo inconveniente del modo 1 es que los registros del temporizador deben reinicializarse luego de cada desbordamiento, mientras que la recarga es automática en el modo 2.

EJEMPLO Onda cuadrada de 1 kHz

4.5

Escriba un programa utilizando el temporizador 0 para crear una onda cuadrada de 1 kHz en P1.0.

Solución

```

8100      6      ORG 8100H
8100 758901 7      MOV TMOD, #01H ;modo de temporizador de
                     16 bits
8103 75BCFE 8 CICLO: MOV TH0, #0FEH ;-500 (byte superior)
8106 758A0C 9      MOV TL0 #0CH ;-500 (byte inferior)
8109 D28C 10     SETB TR0 ;inicia temporizador
810B 308DFD 11 ESPERA: JNB TF0, ESPERA ;espera al desbordamiento
810E C28C 12      CLR TR0 ;detiene el temporizador
8110 C28D 13      CLR TF0 restablece en 0 la bandera de desbordamiento
                     del temporizador
8112 B290 14      CPL P1.0 ;complementa bit de puerto
8114 80ED 15      SJMP CICLO ;repite
                     16
                     END

```

Análisis

Una onda cuadrada de 1 kHz requiere de un tiempo a nivel alto de $500 \mu s$ y un tiempo a nivel bajo de $500 \mu s$. El modo 2 no puede utilizarse ya que el intervalo es más largo que $256 \mu s$. Se requiere del modo de temporizador de 16 bits, el modo 1. La diferencia principal en el software es que los registros del temporizador, TL0 y TH0, se reinician después de cada desbordamiento (líneas 8 y 9).

Existe una ligera discrepancia en la frecuencia de salida del programa anterior. Ello es resultado de las instrucciones adicionales agregadas después del desbordamiento del temporizador para reiniciararlo. El valor de recarga para TL0/TH0 debe ajustarse en cierta manera si se requiere 1 kHz exacto. Tales errores no ocurren en el modo de autorrecarga, ya que el temporizador nunca se detiene sino que se desborda a una velocidad constante establecida mediante el valor de recarga en TH0.

Un programa puede utilizar los dos temporizadores existentes en el 8051 para generar de manera simultánea dos formas de onda distintas en terminales de puerto separadas.

EJEMPLO Uso de dos temporizadores en forma simultánea

4.6

Escriba un programa para crear una onda cuadrada en P1.0 con frecuencia de 10 kHz y onda cuadrada en P2.0 con frecuencia de 1 kHz.

Solución

```

8100      6      ORG 8100H
8100 758912 7      MOV TMOD, #12H ;temporizador 1 en el modo 1,
                     ;temporizador 0 en el modo 2
8103 758CCE 8      MOV TH0, #-50 ;valor de recarga de -50
                     ;en TH0
8106 D28C 10     SETB TR0 ;inicia temporizador 0
8108 758DFE 12 CICLO: MOV TH1, #0FEH ;-500 (byte superior)

```

```

810B 758B0C 13      MOV TL1, #0CH      ; -500 (byte inferior)
810E D28E 14      SETB TR1       ; Inicia temporizador 1
8110 308D04 15    ESPERA: JNB TF0, SIGUIENTE ; ¿desbordamiento del
                   ; temporizador 0?
                   16
8113 C28D 17      CLR TF0        ; no: verifica temporizador 1
                   18
                   ; sí: restablece en 0
                   ; temporizador 0
                   ; bandera de
                   ; desbordamiento
8115 B290 19      CPL P1.0      ; dispara P1.0
8117 308FF6 20    SIGUIENTE: JNB TF1, WAIT ; ¿desbordamiento del
                   ; temporizador 1?
                   21
811A C28E 22      CLR TR1       ; no: verifica temporizador 0
811C C28F 23      CLR TF1       ; detiene temporizador 1
                   24
                   ; restablece en 0 temporizador 1
811E B2A0 25      CPL P2.0      ; bandera de desbordamiento
8120 80E6 26      SJMP CICLO    ; restablece en 0 temporizador 1
                   27
                   END

```

Análisis

En este caso, ambos temporizadores (0 y 1) se utilizan para generar al mismo tiempo dos ondas cuadradas en P1.0 y P2.0, respectivamente. El valor escrito en TMOD inicializa ambos temporizadores al mismo tiempo. La verificación del desbordamiento debe realizarse en secuencia, aun cuando los temporizadores operan en forma simultánea. Primero verificamos al temporizador 0, ya que su periodo es más pequeño, con el fin de evitar perder sus desbordamientos. Observe que el temporizador 0 está operando en el modo de autorrecarga, así que no hay necesidad de recargar la cuenta luego de cada desbordamiento. Mientras tanto, el temporizador 1 opera en el modo 1, así que su cuenta debe recargarse cada vez que ocurre un desbordamiento.

Podemos alcanzar intervalos mayores que 0.066 segundos si conectamos en cascada al temporizador 0 y al temporizador 1 mediante el software, pero esto ocupa a ambos temporizadores. Una manera más práctica de hacerlo es la de utilizar uno de los temporizadores en el modo de 16 bits con un ciclo en el software para contar los desbordamientos. La operación deseada se realiza cada n desbordamientos.

EJEMPLO**4.7****Interfaz para un timbre**

Un timbre está conectado a P1.7, y un interruptor sin rebotes está conectado a P1.6 (consulte la figura 4-7). Escriba un programa para leer el nivel lógico proporcionado por el interruptor y que active al timbre durante 1 segundo por cada transición de 1 a 0 detectada.

Solución

```

0064      6 CIEN      EQU      100      ; 100 × 10000 us=1 seg.
D8F0      7 CONTEO    EQU      -10000
8100      8           ORG      8100H
8100 758901 9      MOV      TMOD, #01H ; utiliza al temporizador
                                         ; 0 en el modo 1
8103 3096FD 10 CICLO: JNB      P1.6,LOOP ; espera la entrada 1
8106 2096FD 11 ESPERA: JB      P1.6,ESPERA ; espera la entrada 0
8109 D297 12          SETB     P1.7   ; enciende timbre

```

```

810B 128112 13      CALL   RETRASO ;espera 1 segundo
810E C297   14      CLR    P1.7    ;apaga timbre
8110 80F1   15      SJMP   CICLO
                    16 ;
8112 7F64   17 RETRASO:    MOV    R7,#CIEN
8114 758CD8 18 DENUEVO:   MOV    TH0,#CUENTA ALTA
8117 758AF0 19          MOV    TL0,#CUENTA BAJA
811A D28C   20          SETB   TR0
811C 308DFD 21 ESPERA2:  JNB    TF0,ESPERA2
811F C28D   22          CLR    TF0
8121 C28C   23          CLR    TR0
8123 DFEF   24          DJNZ   R7,DENUEVO
8125 22       25          RET
                    26          END

```

Análisis

El timbre ilustrado en la figura 4-7 es un transductor piezoelectrónico de cerámica que vibra cuando se estimula mediante un voltaje de CD (corriente directa). Un ejemplo típico es el AI-430 de Projects Unlimited, el cual genera un tono de alrededor de 3 kHz a 5 volts de CD. Un inversor se utiliza como amplificador de corriente, ya que el AI-430 utiliza 7 mA de corriente. Como se indica en las características de CD del 8051 en el apéndice E, las terminales del puerto 1 pueden absorber un máximo de 1.6 mA. El AI-430 cuesta unos cuantos dólares.

El ciclo principal en el software consta de seis instrucciones (líneas 10-15). En la línea 10, un ciclo de una sola instrucción se ejecuta para esperar por el cambio al nivel alto en la señal de entrada en P1.7. Entonces, se ejecuta otro ciclo de una sola instrucción (línea 11) para esperar por el cambio al nivel bajo en la señal de entrada. El timbre se activa por 1 segundo cuando esto ocurre. Esto se implementa en las siguientes tres instrucciones. En primer lugar, establecemos P1.7 en 1 para activar el timbre (línea 12); en segundo lugar, llamamos a una subrutina de retraso de 1 segundo (línea 13) y, en tercer lugar, restablecemos P1.7 en 0 para desactivar el timbre (línea 14). Entonces ejecutamos el ciclo principal de nuevo (línea 15).

La subrutina de retraso (líneas 17-25) utiliza la técnica identificada en la tabla 4-6 como “temporizador de 16 bits más ciclos de software”. En teoría, podemos crear retrasos de cualquier duración. En este ejemplo creamos un retraso de 1 segundo mediante un conteo de -10,000 en TH0/TL0 con un modo de temporizador de 16 bits. El efecto es que se crea un retraso de 10,000 μ s. Este retraso (líneas 18-23) se encajona dentro de un ciclo que se ejecuta 100 veces, utilizando a R7 como un contador. El efecto es un retraso de 1 segundo.

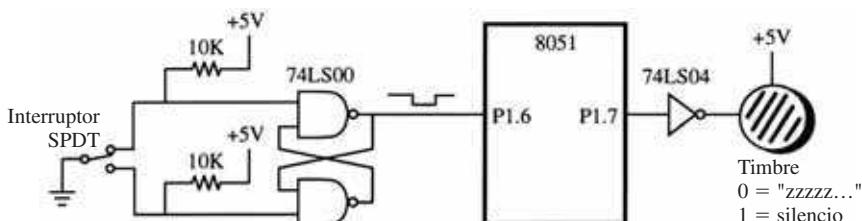
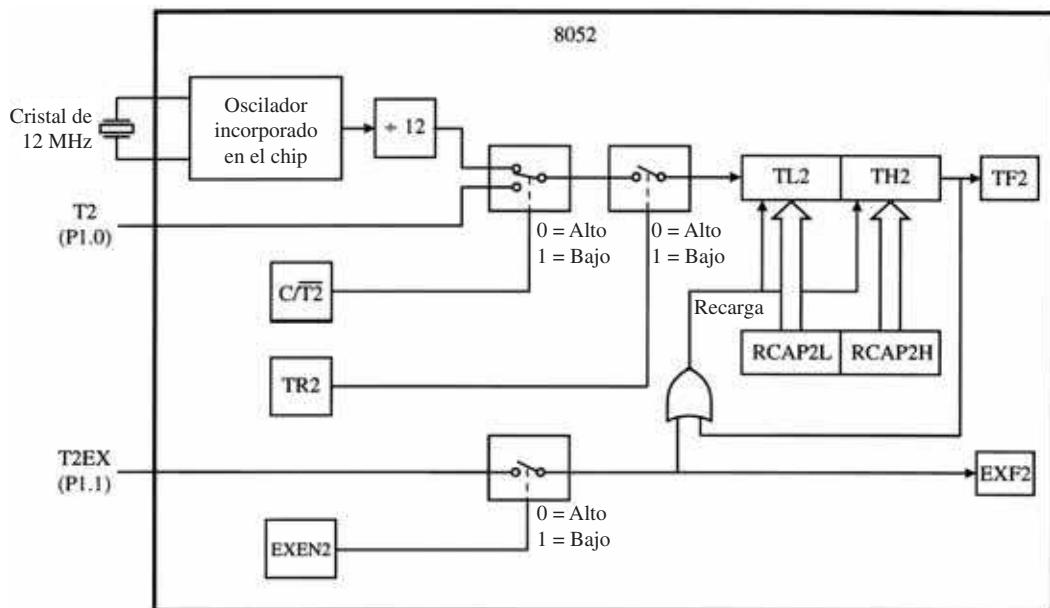


FIGURA 4-7
Ejemplo de un timbre

**FIGURA 4–8**

Temporizador 2 en el modo de autorrecarga de 16 bits

Existen dos situaciones que el ejemplo anterior no soluciona. En primer lugar, la transición no se detecta si la entrada se dispara durante el segundo en que el timbre está activado, ya que el software está ocupado en la rutina de retraso. En segundo lugar, las instrucciones JNB y JB pueden perder por completo la transición si la entrada se dispara demasiado rápido (en menos de un microsegundo). El problema 5 al final de este capítulo trata con la primera situación. La segunda situación sólo puede manejarse mediante el uso de una entrada de interrupción, para fijar una bandera de estado cuando ocurre una transición de 1 a 0. En el capítulo 6 veremos esto con más detalle.

4.9 GENERACIÓN DE FRECUENCIAS EXACTAS

Como vimos en la sección anterior, las frecuencias de salida de las ondas cuadradas que hemos generado hasta ahora tienen ligeros errores. Estos errores se deben al redondeo y a la pérdida de tiempo debido a la ejecución de las instrucciones.

4.9.1 Eliminación de los errores de redondeo

Los errores de redondeo tienen lugar cuando el periodo deseado de cierta forma de onda periódica no es un número entero, así que éste se redondea para poder representarlo en el 8051.

EJEMPLO

Errores de redondeo

4.8

Suponga que se debe generar una onda cuadrada de 3 kHz. ¿Cuál debería ser el valor de recarga del temporizador? Calcule el error de redondeo, si acaso existe, y por medio de esto determine la frecuencia del cristal bajo la cual no se producirían errores de redondeo.

Solución

Una frecuencia de 3 kHz indica que el periodo $333.33 \mu s$, así que el tiempo a nivel alto = tiempo a nivel bajo = $166.67 \mu s$. El valor de recarga del temporizador debe ser de 167 conteos antes del desbordamiento, o –167, ya que el 8051 sólo puede manejar valores de conteo enteros. Observe que se redondeó el conteo deseado a un valor entero. El periodo real es, por lo tanto, de $167 \times 2 \times 1 \mu s = 334 \mu s$, así que la frecuencia real es de 2.994 kHz. El error de redondeo es

$$\text{Error de redondeo} = \frac{|f_{\text{deseada}} - f_{\text{redondeada}}|}{f_{\text{deseada}}} \times 100\% \\ = \frac{3 \text{ kHz} - 2.994 \text{ kHz}}{3 \text{ kHz}} \times 100\% = 0.2\%.$$

Análisis

Los ejemplos anteriores utilizaron una frecuencia de cristal de 12 MHz, pero esto causa errores de redondeo. Nuestra meta es calcular la frecuencia del cristal de manera que podamos obtener una onda cuadrada con una frecuencia exacta de 3 kHz, asumiendo que utilizamos un conteo de 167. periodo deseado = $333.33 \mu s$ así que

$$\begin{aligned} \text{TIEMPO A NIVEL ALTO DESEADO} &= \text{TIEMPO A NIVEL BAJO DESEADO} \\ &= 166.67 \mu s \\ &= \text{TIEMPO DESEADO ENTRE CADA} \\ &\quad \text{DESBORDAMIENTO DEL TEMPORIZADOR} \\ &= \text{CUENTA EN TEMPORIZADOR} \times \\ &\quad \text{PERIODO DEL CICLO DE MAQUINA} \\ &= 167 \times \text{PERIODO DEL CICLO DE MAQUINA} \end{aligned}$$

Por lo tanto, PERIODO DEL CICLO DE MAQUINA = $166.67 \mu s / 167 =$

$$0.9980239 \mu s$$

y FRECUENCIA DE CICLO DE MAQUINA = $1 / 0.9980239 \mu s =$

$$1.00198 \text{ MHz}$$

así que FRECUENCIA DEL CRISTAL = $1.00198 \text{ MHz} \times 12 = 12.0238 \text{ MHz.}$

4.9.2 Compensación de la pérdida de tiempo debido a las instrucciones

Las instrucciones tardan cierto tiempo en ejecutarse. La instrucción más simple requiere un ciclo de máquina, mientras que instrucciones más complicadas requieren de 4 ciclos. Por lo tanto, las instrucciones también agregan un retraso de tiempo, y si preferimos frecuencias exactas tendríamos que ajustar nuestros conteos iniciales del temporizador para compensar la pérdida de tiempo causada por la ejecución de estas instrucciones. El ejemplo 4.9 considera una demostración bastante simple de esto.

EJEMPLO 4.9 Modifique el programa del ejemplo 4.5 para compensar el retraso ocasionado por la pérdida de tiempo debido a las instrucciones contenidas en el programa.

Solución

```
8100      1      ORG 8100H
8100 758901  2      MOV TMOD, #01H ;modo de temporizador de 16 bits
```

```

8103 758CFE 3 CICLO: MOV TH0, #0FEH ; -490 (byte superior)
8106 758A0E 4 MOV TL0, #0EH ; -490 (byte inferior)
8109 D28C 5 SETB TR0 ; inicia temporizador
810B 308DFD 6 ESPERA: JNB TF0, ESPERA ; espera el
                                         ; desbordamiento
810E C28C 7 CLR TR0 ; detiene el temporizador
8110 C28D 8 CLR TF0 ; restablece en 0 la
                         ; bandera de desbordamiento
                         ; del temporizador
8112 B290 9 CPL P1.0 ; complementa bit de puerto
8114 80ED 10 SJMP CICLO ; repite
11 END

```

Análisis

El programa es exactamente igual al del ejemplo 4.5. Sólo el valor de recarga es diferente. En este caso, elegimos un valor de -490 para compensar la pérdida de tiempo causada por las instrucciones. Analicemos por qué fue elegido este valor.

TABLA 4-6

Resumen del registro TCON (control del temporizador 2)

Bit	Símbolo	Dirección de bit	Descripción
T2CON.7	TF2	CFH	Bandera de desbordamiento del temporizador 2. (No se activa cuando TCLK o RCLK = 1)
T2CON.6	EXF2	CEH	Bandera externa del temporizador 2. Se establece en 1 cuando se causa una captura o recarga mediante la transición de 1 a 0 en T2EX Y EXEN2 = 1; cuando las interrupciones del temporizador están habilitadas, EXF2 = 1 causa que la CPU se vectorice a la rutina de servicio correspondiente; se restablece en 0 mediante el software
T2CON.5	RCLK	CDH	Reloj receptor del temporizador 2. Cuando es 1, el temporizador 2 provee la tasa de recepción en baudios de recepción de puerto serial, el temporizador 1 provee la tasa en baudios de transmisión
T2CON.4	TCLK	CCH	Reloj transmisor del temporizador 2. Cuando es 1, el temporizador 2 provee la tasa en baudios del transmisor; el temporizador 1 provee la tasa en baudios del receptor
T2CON.3	EXEN2	CBH	Habilitador externo del temporizador 2. Cuando es 1, se produce una captura o recarga en la transición de 1 a 0 de T2EX
T2CON.2	TR2	CAH	Bit de control de arranque del temporizador 2. Es puesto en 1/0 mediante el software para encender/apagar el temporizador
2T2CON.1	C/T2	C9H	Bit de selección de contador/temporizador de intervalos del temporizador 2. 1 = contador de eventos 0 = temporizador de intervalos
T2CON.0	CP/RL2C	C8H	Bandera de captura/recarga del temporizador 2. Cuando es 1, se produce una captura en una transición de 1 a 0 de EXEN2 = 1; cuando es 0, se produce una autorrecarga en un desbordamiento del temporizador o una transición en T2EX si EXEN2 = 1; este bit se ignora si RCLK o TCLK = 1

Nos concentraremos en el ciclo del software, el cual consta de dos instrucciones MOV, seguidas por SETB, JNB, dos instrucciones CLR, CLP y SJMP. Las instrucciones MOV, JNB y SJMP requieren 2 ciclos de máquina cada una, mientras que SETB, CLR y CPL sólo ocupan 1 ciclo cada una.

El valor inicial de P1.0 puede ser ALTO o BAJO. Podemos asumir que P1.0 está en BAJO al principio. Las primeras dos instrucciones MOV requieren 2 ciclos de máquina cada una, mientras que SETB requiere 1 ciclo. La siguiente instrucción es JNB, la cual determina la duración del ciclo y se ejecutará en forma continua hasta que el temporizador 0 se desborde. Cada ejecución de JNB requiere 2 ciclos de máquina.

P1.0 debe complementarse al estado opuesto justo en el momento en que el temporizador 0 se desborda. Sin embargo, en el programa anterior, P1.0 sólo se complementará después de la ejecución completa de la instrucción CPL, lo cual ocurre luego de 3 ciclos de máquina. Este retraso adicional es la pérdida de tiempo debido a las instrucciones. La figura 4-8 ilustra esto con mayor detalle, donde se indican los estados de P1.0 y TR0. Observe también que el tiempo en nivel alto o bajo real de la onda cuadrada en P1.0 se encuentra entre las dos siguientes instrucciones CPL. La figura 4-8 muestra que esta duración es de un ciclo completo del temporizador de $490 \mu s$, además de la pérdida de tiempo debido a la ejecución de las instrucciones CLR y CPL después del desbordamiento $\beta \mu s^2$ y a la ejecución de las instrucciones SJMP, MOV y SETB $\eta \mu s^2$ antes de que el temporizador se reinicie para el siguiente ciclo.

Por lo tanto, si tomamos todo esto en cuenta, el tiempo real en nivel alto o bajo es de $500 \mu s$, lo cual produce la frecuencia exacta deseada de 1 kHz para la onda cuadrada.

4.10 TEMPORIZADOR 2 DEL 8052

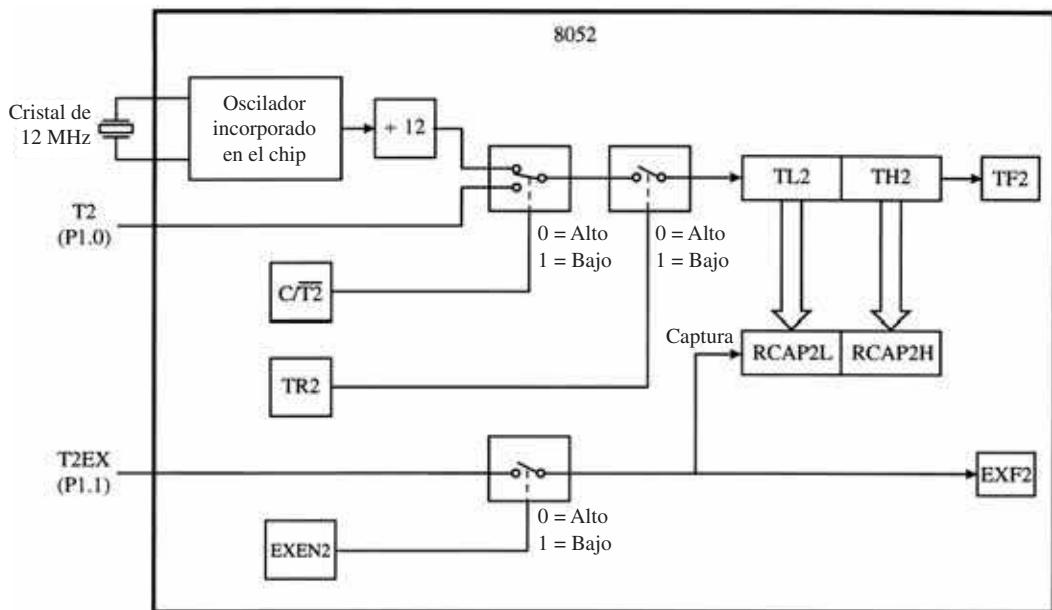
El tercer temporizador añadido al 8052 constituye una poderosa adición a los dos temporizadores que ya hemos estudiado. Como ilustramos antes en la tabla 4-1, se agregan cinco registros con funciones especiales adicionales para acomodar al temporizador 2. Éstos incluyen los registros del temporizador, TL2 y TH2, el registro de control del temporizador, T2CON, y los registros de captura, RCAP2L y RCAP2H.

El modo para el temporizador 2 se establece mediante su registro de control, T2CON. (Consulte la tabla 4-6). El temporizador 2, tal como los temporizadores 0 y 1, puede operar como temporizador de intervalos o como contador de eventos. La fuente de reloj puede ser proporcionada internamente por el oscilador incorporado en el chip, o en forma externa mediante T2, la cual es la función alterna del bit 0 del puerto 1 (P1.0) en el 8052. El bit C/T2 en T2CON selecciona entre un reloj interno o uno externo, tal como lo hacen los bits C/T en el TCON de los temporizadores 0 y 1. Existen tres modos de operación sin importar qué fuente de reloj se utiliza: autorrecarga, captura y generador de tasa en baudios.

4.10.1 Modo de autorrecarga

El bit de captura/recarga en T2CON selecciona uno de los dos primeros modos. Cuando CP/RL2 = 0, el temporizador 2 se encuentra en el modo de autorrecarga, donde TL2/TH2 actúan como los registros del temporizador, y RCAP2L y RCAP2H almacenan el valor de recarga. A diferencia del modo de recarga para los temporizadores 0 y 1, el temporizador 2 siempre es un temporizador de 16 bits, incluso en el modo de autorrecarga.

La recarga se produce en una transición de FFFFH a 0000H en TL2/TH2 y establece en 1 la bandera del temporizador, TF2. Esta condición está determinada por el software o se programa para generar una interrupción. De cualquier manera, debemos restablecer a 0 TF2 mediante el software antes de activarla de nuevo.

**FIGURA 4–9**

Temporizador 2 en el modo de captura de 16 bits

Podemos ocasionar también una recarga en la transición de 1 a 0 de la señal aplicada a la terminal T2EX mediante la puesta en 1 de EXEN2 en T2CON, lo cual es la función alterna de la terminal P1.1 en el 8052. Una transición de 1 a 0 en T2EX también pone en 1 una nueva bandera de bit en el temporizador 2, EXF2. Así como con TF2, el estado de EXF2 se prueba mediante el software o genera una interrupción. EXF2 debe restablecerse a 0 mediante el software. La figura 4-8 muestra al temporizador 2 en el modo de autorrecarga.

4.10.2 Modo de captura

Cuando $CP/RL2 = 1$, se selecciona el modo de captura. El temporizador opera como un temporizador de 16 bits, y establece en 1 el bit TF2 cuando ocurre una transición de FFFFH a 0000H del valor en TL2/TH2. El estado de TF2 se prueba mediante el software o genera una interrupción.

Para habilitar la característica de captura, el bit EXEN2 en T2CON debe estar en 1. La transición de 1 a 0 en T2EX (P1.1) “captura” el valor en los registros del temporizador TL2/TH2 al sincronizarlos en los registros RCAP2L y RCAP2H, cuando EXEN2 = 1. La bandera EXF2 en T2CON también se establece en 1 y, como vimos antes, se prueba mediante el software o genera una interrupción. La figura 4-9 muestra el temporizador 2 en el modo de captura.

4.11 GENERACIÓN DE TASA EN BAUDIOS

Otro de los usos de los temporizadores es proporcionar el reloj de la tasa de transmisión y recepción en baudios para el puerto serial incorporado en el chip. Esto se genera mediante el temporizador 1 en el 8051, o con el temporizador 1 y/o el temporizador 2 en el 8052. En el capítulo 5 hablaremos sobre la generación de la tasa en baudios.

RESUMEN

En este capítulo presentamos los temporizadores 8051 y 8052. Las soluciones en software para los ejemplos aquí presentados cuentan con una característica común, pero que también es un tanto limitante. En las soluciones, el software consume todo el tiempo de ejecución de la CPU. Los programas se ejecutan en ciclos de espera, esperando un desbordamiento del temporizador. Esto es aceptable para propósitos de aprendizaje, pero en las aplicaciones prácticas orientadas al control, donde se utilizan microcontroladores, la CPU debe realizar otras tareas y responder a eventos externos, tales como cuando un operador utiliza el teclado para cambiar un parámetro. En el capítulo que trata sobre las interrupciones, demostraremos cómo es que utilizamos los temporizadores en un ambiente “controlado por interrupciones”. Las banderas de desbordamiento del temporizador no se prueban en un ciclo de software, sino que generan una interrupción. Otro programa interrumpe temporalmente al programa principal mientras se realiza alguna acción que afecta la interrupción del temporizador (tal vez el complemento de un bit de puerto). Podemos crear la ilusión de que se realizan varias tareas en forma simultánea mediante las interrupciones.

PROBLEMAS

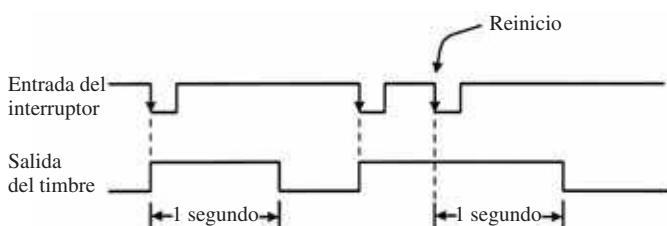
- 4.1 Escriba un programa en el 8051 para crear una onda cuadrada en P1.5 con frecuencia de 100 kHz. (Sugerencia: no utilice los temporizadores).
- 4.2 ¿Cuál es el efecto de la siguiente instrucción?

```
SETB 8EH
```
- 4.3 ¿Cuál es el efecto de la siguiente instrucción?

```
MOV TMOD, #11010101B
```
- 4.4 Considere el programa de tres instrucciones mostrado en el ejemplo 4-2. ¿Cuál es la frecuencia y cuál el ciclo de trabajo de la forma de onda creada en P1.0 para un 8051 que opera a 16 MHz?
- 4.5 Escriba de nuevo la solución al ejemplo 4.7, pero esta vez incluya un modo de “reinicio”. Si ocurre una transición de 1 a 0 mientras el timbre está activado, reinicie el ciclo de temporización para continuar activando al timbre durante un segundo más. La figura 4-10 ilustra esto.
- 4.6 Escriba un programa en el 8051 para generar una onda cuadrada de 12 kHz en P1.2 utilizando el temporizador 0.
- 4.7 Diseñe una aplicación de “torniquete” utilizando el temporizador 1 para determinar cuándo es que el número de personas que van entrando a un parque de juegos llega a 10,000. Asuma que (a) un sensor del torniquete está conectado a T1 y genera un pulso cada vez que el torniquete rota, y (b) una luz conectada a P1.7 está encendida cuando P1.7 = 1, y apagada si no es así. Cuente los “eventos” en T1 y encienda la luz en P1.7 cuando la persona número 10,000 entra al parque. (Consulte la figura 4-11).

FIGURA 4-10

Temporización para el ejemplo
del timbre modificado



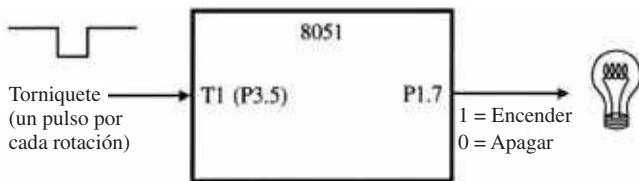


FIGURA 4-11
Problema del torniquete

- 4.8** El estándar internacional de afinación para los instrumentos musicales es de “LA arriba del DO medio” a una frecuencia de 440 Hz. Escriba un programa en el 8051 para generar esta frecuencia de afinación y activar un tono de 440 Hz en un altavoz conectado a P1.1. (Consulte la figura 4-12). Existe un ligero error en la frecuencia de salida, debido al redondeo de los valores almacenados en TL1/TH1. ¿Cuál es la frecuencia de salida exacta y cuál el porcentaje de error? ¿Cuál es el valor del cristal que proporcionaría exactamente 400 Hz con el programa que usted acaba de escribir?
- 4.9** Escriba un programa en el 8051 para generar una señal de 500 Hz en P1.0 utilizando el temporizador 0. La forma de onda deberá tener un ciclo de trabajo del 30% $\frac{\text{Tiempo a nivel alto}}{\text{Tiempo a nivel alto} + \text{periodo}}$
- 4.10** El circuito mostrado en la figura 4-13 proveerá una señal extremadamente precisa de 60 Hz a T2 mediante la utilización del secundario de un transformador de fuente de poder. Inicialice el temporizador 2 de tal forma que la señal recibida en T2 constituya la señal de reloj y se desborde una vez por segundo. En cada desbordamiento deberá actualizar un valor de la hora del día, almacenado en la memoria interna del 8052 en las ubicaciones 50H (horas), 51H (minutos), y 52H (segundos). Consulte el capítulo 6 para ver más ejemplos y problemas sobre temporizadores.
- 4.11**
 - Escriba un programa para generar una onda de pulso rectangular (con ciclo de trabajo del 30%) en P1.0, y una forma de onda cuadrada en P2.0.
 - Después de esto determine (i) el error de redondeo, y (ii) el error debido a la pérdida de tiempo causada por la ejecución de las instrucciones para ambas formas de onda.
- 4.12** Escriba un programa para generar una forma de onda de 2.5 kHz en P1.0 con ciclo de trabajo del 20%. Muestre su proceso y escriba comentarios para cada línea del código.
- 4.13** Considere un oscilador de cristal externo con una frecuencia de 16 MHz en lugar de los 12 MHz estándar. Escriba un programa para generar una forma de onda periódica en P1.0 con la frecuencia más alta posible. Después calcule la frecuencia y el ciclo de trabajo de dicha forma de onda.
- 4.14** Suponga que desea utilizar el temporizador del 8051 para medir la duración de 1 hora. ¿Cuál es el modo del temporizador que utilizaría? ¿Por qué?
- 4.15** Suponga que debe generar una forma de onda cuadrada en P1.0 con frecuencia de 21 kHz y ciclo de trabajo del 10%. ¿Cuál es el modo de temporizador que utilizaría? ¿Por qué?
- 4.16** ¿Cuál es el valor más alto hasta el que puede contar un temporizador del 8051? ¿Qué efecto tendría en su conteo si el temporizador del 8051 se utilizara como un contador de eventos? ¿Por qué?

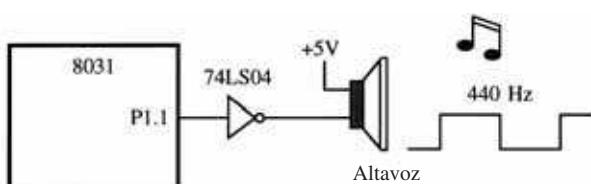
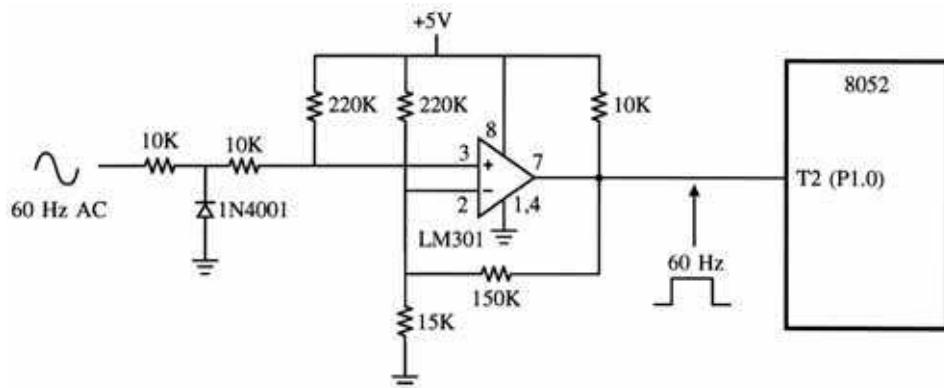


FIGURA 4-12
Interfaz para altavoz

**FIGURA 4-13**

Base de tiempo de 60 Hz

- 4.17** ¿A qué nos referimos con el término temporizador de 16 bits? Explique su respuesta.
- 4.18** a. Escriba un programa para generar una forma de onda de 3 kHz en P2.0 con ciclo de trabajo del 30%. Muestre su proceso y escriba comentarios para cada línea del código. Efectúe los ajustes necesarios para resolver el retraso debido a la pérdida de tiempo causada por la ejecución de las instrucciones.
- b. ¿Existe algún error por truncamiento en el programa del inciso anterior? Si existe, calcule el porcentaje de error. Si no, ¿por qué afirma usted que no existe ningún error?

<http://www.ingeelectronico.blogspot.com>

Operación del puerto serial

5.1 INTRODUCCIÓN

El 8051 incluye un puerto serial incorporado al chip, el cual puede operar en diversos modos sobre un extenso rango de frecuencias. La función esencial del puerto serial es realizar una conversión de paralelo a serial para los datos de salida, y una conversión de serial a paralelo para los datos de entrada.

El acceso al puerto serial mediante el hardware se realiza a través de las terminales TXD y RXD, las cuales presentamos en el capítulo 2. Estas terminales constituyen las funciones alternas para dos bits del puerto 3, P3.1 en la terminal 11 (TXD) y P3.0 en la terminal 10 (RXD).

El puerto serial cuenta con una operación **full duplex** (transmisión y recepción simultáneas) y con **búfer de recepción**, lo cual permite recibir un carácter que se almacena en un búfer mientras se recibe un segundo carácter. Los datos no se pierden si la CPU lee el primer carácter antes de que el segundo se haya recibido por completo.

La frecuencia de operación del puerto serial, o **tasa de transmisión y recepción en baudios**, puede ser fija (derivada del oscilador incorporado al chip del 8051) o variable. El temporizador 1 proporciona el reloj de la tasa de transmisión y recepción en baudios cuando se utiliza una tasa en baudios variable, y debe ser programado en forma adecuada. (Podemos programar el temporizador 2 en el 8032/8052 para proveer el reloj de la tasa en baudios.)

Dos registros de función especial, el registro de búfer del puerto serial (SBUF) y el registro de control del puerto serial (SCON), disponen el acceso mediante software al puerto serial.

5.2 COMUNICACIÓN SERIAL

Antes de continuar el análisis sobre la operación del puerto serial del 8051, trataremos primero algunos conceptos básicos de la comunicación serial. Este tipo de comunicación involucra la transmisión de bits de datos a través de una sola línea de comunicación. Los datos se transmiten bit por bit, ya sea en formato síncrono o asíncrono. La comunicación serial **síncrona** transmite un bloque completo de caracteres sincronizados mediante un reloj de referencia, mientras que la comunicación serial **asíncrona** transmite de manera aleatoria un carácter en cualquier momento,

independientemente de cualquier reloj. Por ejemplo, la transmisión que se realiza hacia la computadora cada que se oprime una tecla en el teclado es una comunicación asíncrona, pues la velocidad a la que se oprimen las teclas no es fija y la transmisión puede ocurrir en cualquier instante. Por otra parte, para que exista una comunicación síncrona entre dos computadoras, ambas deberán estar sincronizadas al mismo reloj de referencia durante la transmisión.

5.3 REGISTRO DE BÚFER DEL PUERTO SERIAL (SBUF)

El registro de búfer del puerto serial (SBUF) localizado en la dirección 99H constituye, en realidad, dos memorias intermedias. Al escribir en el SBUF se cargan los datos a transmitir y al leer del SBUF se accede a los datos recibidos. Éstos son dos registros separados y distintos, el registro de sólo escritura para transmisión y el de sólo lectura para recepción. (Consulte la figura 5-1).

Observe en la figura 5-1 que se utiliza un registro de desplazamiento de serial a paralelo para sincronizar los datos recibidos antes de que se transfieran al registro de sólo lectura para recepción. El registro de desplazamiento es el elemento clave para utilizar el búfer de recepción. Los 8 bits de los datos entrantes únicamente se transferirán al registro de sólo lectura para recepción cuando se hayan recibido todos. Esto asegura que mientras se estén recibiendo los datos entrantes, los datos recibidos previamente se conserven todavía intactos en el registro de sólo lectura para recepción.

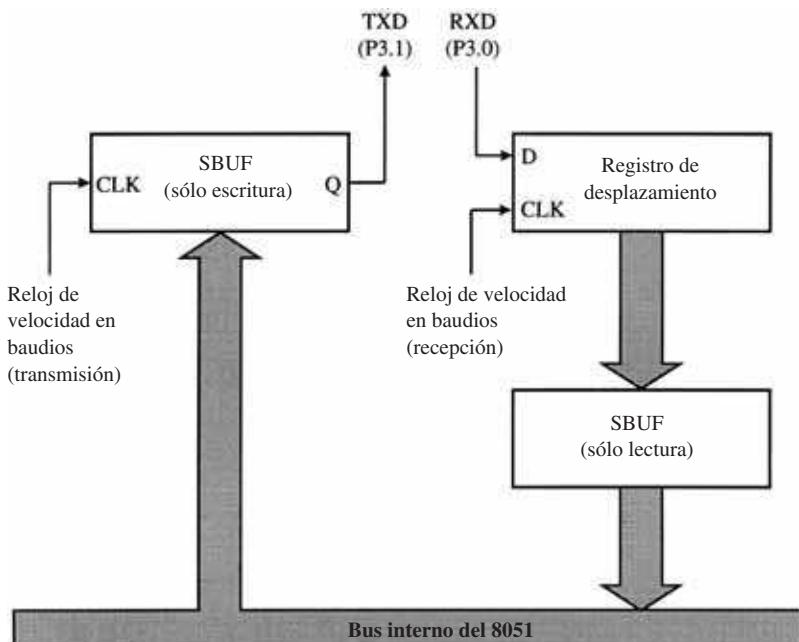


FIGURA 5-1

Diagrama de bloques del puerto serial

5.4 REGISTRO DE CONTROL DEL PUERTO SERIAL (SCON)

El registro de control del puerto serial (SCON) ubicado en la dirección 98H es un registro direccional por bit que contiene los bits de estado y los bits de control. Los bits de estado indican el fin de una transmisión o recepción de un carácter y se prueban en el software o se programan para producir una interrupción. Por otra parte, el modo de operación del puerto serial del 8051 se establece mediante la escritura a los bits de control. (Consulte las tablas 5-1 y 5-2).

Antes de utilizar el puerto serial, inicializamos el SCON para operar en el modo correcto, y así sucesivamente. Por ejemplo, la siguiente instrucción

```
MOV SCON, #01010010B
```

inicializa el puerto serial en el modo 1 ($SM0/SM1 = 0/1$), habilita al receptor ($REN = 1$), y establece en 1 la bandera de interrupción de transmisión ($T1 = 1$) para indicar que el transmisor está listo para operar.

5.5 MODOS DE OPERACIÓN

El puerto serial del 8051 tiene cuatro modos de operación, los cuales se seleccionan en el SCON mediante la escritura de unos o ceros a los bits SM0 y SM1. Tres de estos modos habilitan las comunicaciones asíncronas, en donde cada carácter recibido o transmitido está enmarcado por un bit de inicio y uno de detención. Los lectores que estén familiarizados con la operación de un puerto serial RS232C común en una microcomputadora, encontrarán que estos modos les son conocidos. En el cuarto modo, el puerto serial opera como un simple registro de desplazamiento. A continuación sintetizaremos cada uno de estos modos.

5.5.1 Registro de desplazamiento de 8 bits (Modo 0)

El modo 0, el cual se selecciona al escribir ceros en los bits SM1 y SM0 del SCON, configura el puerto serial en el modo de registro de desplazamiento de 8 bits. Los datos seriales entran y salen a través de RXD, y TXD envía la salida del reloj de desplazamiento. En este modo se transmiten o reciben ocho bits, con el bit menos significativo (LSB) primero.

TABLA 5-1

Resumen del registro SCON (control del puerto serial)

Bit	Símbolo	Dirección	Descripción
SCON.7	SM0	9FH	Bit 0 del modo del puerto serial (consulte la tabla 5-2)
SCON.6	SM1	9EH	Bit 1 del modo del puerto serial (consulte la tabla 5-2)
SCON.5	SM2	9DH	Bit 2 del modo del puerto serial. Habilita la comunicación entre múltiples procesadores en los modos 2 y 3; R1 no se activará si el noveno bit es 0
SCON.4	REN	9CH	Habilitación del receptor. Debe ser puesto en 1 para recibir caracteres
SCON.3	TB8	9BH	Bit de transmisión 8. El noveno bit se transmite en los modos 2 y 3; es puesto en 1/0 mediante el software
SCON.2	RB8	9AH	Bit 8 de recepción. El noveno bit recibido
SCON.1	TI	99H	Bandera de interrupción de transmisión. Se establece en 1 al final de la transmisión de un carácter; se restablece en 0 mediante el software
SCON.0	RI	98H	Bandera de interrupción de recepción. Se establece en 1 al final de la recepción de un carácter; se restablece en 0 mediante el software

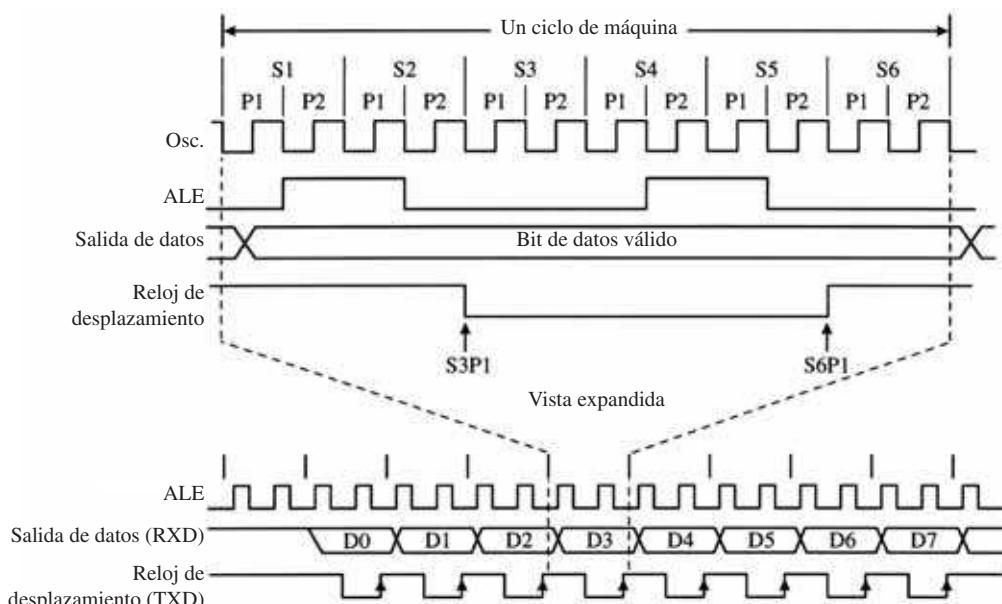
TABLA 5–2
Modos del puerto serial

SM0	SM1	Modo	Descripción	Velocidad en baudios
0	0	0	Registro de desplazamiento	Fija (frecuencia del oscilador $\div 12$)
0	1	1	UART de 8 bits	Variable (establecida por un temporizador)
1	0	2	UART de 9 bits	Fija (frecuencia del oscilador $\div 12$ o $\div 64$)
1	1	3	UART de 9 bits	Variable (establecida por un temporizador)

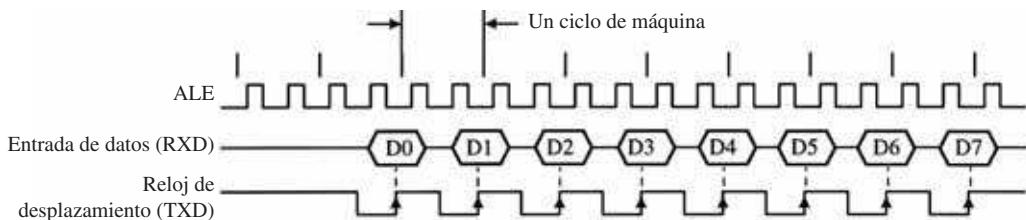
La velocidad en baudios está fija a un doceavo de la frecuencia del oscilador incorporado al chip. El uso de los términos “RXD” y “TXD” en este modo es confuso. La línea RXD se utiliza tanto para datos de entrada como para datos de salida, y la línea TXD sirve como el reloj.

La transmisión se inicia mediante una instrucción que escribe datos al SBUF. Los datos se desplazan hacia fuera de la línea RXD (P3.0) mediante pulsos de reloj enviados a través de la línea TXD (P3.1). Cada bit transmitido es válido en la terminal RXD sólo durante un ciclo de máquina. Durante cada ciclo de máquina, la señal del reloj cambia a bajo nivel en S3P1 y regresa al nivel alto en S6P1. La figura 5–2 muestra la sincronización para los datos de salida.

La recepción se inicia cuando el bit de habilitación del receptor (REN) es igual a 1 y el bit de interrupción de recepción (RI) es igual a 0. La regla general es que la señal REN se activa al inicio de un programa para inicializar el puerto serial, y entonces la señal RI se limpia para comenzar una operación de entrada de datos. Los pulsos de reloj se escriben en la línea TXD cuando RI se limpia, lo cual comienza el siguiente ciclo de máquina, y los datos se sincronizan en la línea RXD. Resulta evidente que los circuitos adjuntos son los encargados de proporcionar los datos

**FIGURA 5–2**

Sincronización de la transmisión para el modo 0 del puerto serial

**FIGURA 5-3**

Sincronización de la recepción para el modo 0 del puerto serial

en la línea RXD, sincronizados mediante la señal de reloj en TXD. La sincronización de los datos hacia el puerto serial ocurre en el flanco positivo de TXD. (Consulte la figura 5-3). Observe que en este modo de operación, las transferencias de datos entre el puerto serial del 8051 y los circuitos adjuntos se realizan a través de una comunicación síncrona, en la cual ambos participantes se sincronizan a la señal de reloj en TXD. En las secciones posteriores descubriremos que los otros modos de operación del puerto serial operan a través de una comunicación asíncrona.

Una posible aplicación del modo de registro de desplazamiento es la de expandir la capacidad de salida del 8051. Podemos conectar un circuito integrado de registro de desplazamiento serial a paralelo a las líneas de TXD y RXD en el 8051 para proporcionar ocho líneas adicionales de salida. (Consulte la figura 5-4). Para aumentar la expansión, con el primer registro podemos conectar en cascada registros de desplazamiento adicionales.

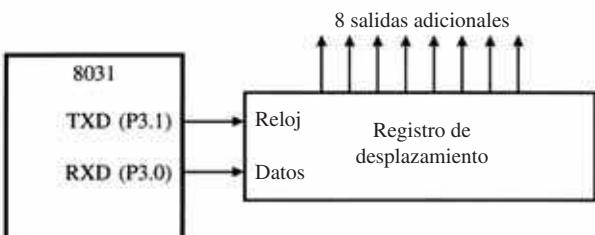
5.5.2 UART de 8 bits con velocidad en baudios variable (Modo 1)

El modo 1 del puerto serial del 8051 le permite operar como un UART de 8 bits con velocidad en baudios variable. Un “receptor/transmisor asíncrono universal” (UART, por sus siglas en inglés), es un dispositivo que recibe y transmite datos seriales y donde cada carácter va precedido por un bit de inicio (bajo) y seguido por un bit de detención. A veces se añade un bit de paridad entre el último bit de los datos y el bit de detención. La operación esencial de un UART es la conversión paralelo a serial de los datos de salida y la conversión serial a paralelo de los datos de entrada.

En el modo 1 se transmiten 10 bits en TXD o se reciben en RXD. Estos bits constan de un bit de inicio (siempre 0), ocho bits de datos (el bit menos significativo primero), y un bit de detención (siempre 1). El bit de detención se transfiere a RB8 en SCON para operar una recepción. La velocidad en baudios del 8051 se establece mediante la velocidad de desbordamiento del temporizador 1; la velocidad en baudios del 8052 se establece mediante la velocidad de desbordamiento del temporizador 1 o 2 o por una combinación de ambos (uno para transmisión, otro para recepción).

FIGURA 5-4

Modo de registro de desplazamiento del puerto serial



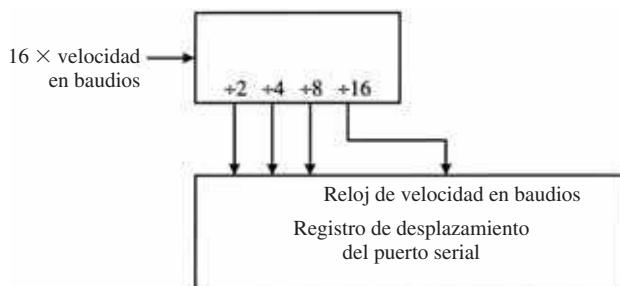


FIGURA 5–5
Sincronización del puerto serial

La sincronización de los registros de desplazamiento del puerto serial en los modos 1, 2 y 3 se establece mediante un contador de división entre 16 de 4 bits, cuya salida es el reloj de velocidad en baudios. (Consulte la figura 5–5). La entrada a este contador se selecciona mediante el software, como veremos después.

La transmisión se inicia al escribir al SBUF, pero no empieza sino hasta el siguiente desbordamiento del contador de división entre 16, el cual sirve como fuente de la velocidad en baudios del puerto serial. Los datos desplazados se envían como salida a la línea TXD comenzando con el bit de inicio, seguido por los ocho bits de datos, y entonces el bit de detención. El periodo para cada bit es el recíproco de la velocidad en baudios programada en el temporizador. La bandera de interrupción de transmisión (TI) se activa tan pronto como aparece el bit de detención en TXD. (Consulte la figura 5–6).

La recepción se inicia mediante una transición de 1 a 0 en RXD. El contador de división entre 16 se reinicia de inmediato para alinear el conteo con el flujo de los bits entrantes (el siguiente bit llega en el siguiente desbordamiento de la división entre 16, y así sucesivamente). El flujo de bits entrantes se muestrea a la mitad de los 16 conteos.

El receptor incluye una “detección de bit de inicio falso”, ya que requiere de ocho conteos en el estado 0 después de la primera transición de 1 a 0. Si esto no ocurre, se asume que el receptor se disparó por causa de ruido en lugar de por un carácter válido. El receptor se reinicializa y regresa al estado de reposo, esperando la siguiente transición de 1 a 0.

La recepción de caracteres continúa si asumimos que se detectó un bit de inicio válido. El bit de inicio ignorado y ocho bits de datos se aplican mediante un pulso de reloj al registro de desplazamiento del puerto serial. Cuando los ocho bits se han sincronizado, ocurre lo siguiente:

1. El noveno bit (el bit de detención) se sincroniza a RB8 en SCON,
2. se carga el SBUF con los ocho bits de datos, y
3. se activa la bandera de interrupción de recepción (RI).

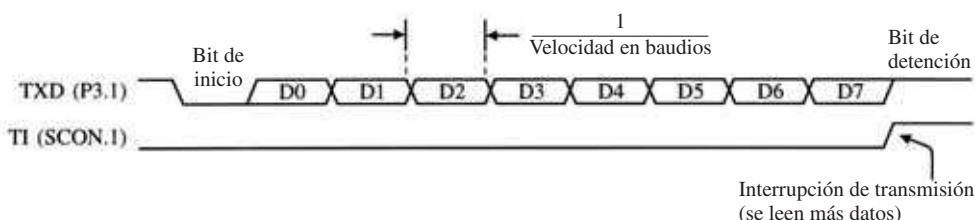


FIGURA 5–6

Activación de la bandera del puerto serial T1

Sin embargo, lo anterior ocurre sólo cuando existen las siguientes condiciones:

1. y
2. SM2 = 1 y el bit de detención recibido = 1, o SM2 = 0.

El requisito de RI = 0 asegura que el software haya leído el carácter anterior (y que haya restablecido RI en 0). La segunda condición parece complicada, pero se aplica sólo en el modo de comunicaciones entre múltiples procesadores (consulte la siguiente sección). Esto implica que “no debemos activar RI en el modo de comunicaciones entre múltiples procesadores cuando el noveno bit es igual a 0”.

5.5.3 UART de 9 bits con velocidad en baudios fija (Modo 2)

El puerto serial opera en modo 2 como un UART de 9 bits, con velocidad en baudios fija cuando SM1 = 1 y SM0 = 0. Se transmiten o reciben once bits: un bit de inicio, ocho bits de datos, un noveno bit de datos programable, y un bit de detención. El noveno bit es igual al contenido de TB8 en SCON (quizás un bit de paridad) durante la transmisión. El noveno bit recibido se transfiere a RB8 durante la recepción. La velocidad en baudios en el modo 2 es de 1/32 o de 1/64 de la frecuencia del oscilador incorporado al chip. (Consulte la sección 5.9 Velocidades en baudios del puerto serial).

5.5.4 UART de 9 bits con velocidad en baudios variable (Modo 3)

El modo 3, un UART de 9 bits con velocidad en baudios variable, es igual que el modo 2, excepto que la velocidad en baudios es programable y la proporciona el temporizador. De hecho, los modos 1, 2 y 3 son muy similares. Sus diferencias se centran en las velocidades en baudios (fija en el modo 2, variable en los modos 1 y 3) y en la cantidad de bits de datos (ocho en el modo 1, nueve en los modos 2 y 3).

5.6 ESCOLLOS ACERCA DE LA COMUNICACIÓN SERIAL TIPO FULL DUPLEX

El puerto serial del 8051 permite una operación full duplex, lo cual significa que tanto la transmisión como la recepción de caracteres pueden realizarse en forma simultánea. Sin embargo, ello implica algunas dificultades.

La primera dificultad se relaciona con las conexiones físicas. La comunicación full duplex debe utilizar dos líneas separadas: una para transmisión y otra para recepción; de otra manera ambas señales colisionarían. En el modo 0 de operación del puerto serial se utiliza sólo una línea (la de RXD) tanto para transmisión como para recepción, y en consecuencia no se puede alcanzar la comunicación full duplex. De hecho, como establecimos en la sección que trata sobre los modos de operación del puerto serial, el modo 0 provee sólo una operación half duplex. Por otra parte, los otros tres modos utilizan dos líneas separadas, TCD y RXD para transmisión y recepción, respectivamente, de modo que sí permiten una operación full duplex.

En segundo lugar, debemos considerar la sincronización. Una pregunta común de un estudiante cuando se entera de que el puerto serial puede operar en modo full duplex es: “Supongamos que el puerto serial está transmitiendo un carácter a un dispositivo serial, así que tanto el 8051 como el dispositivo serial están sincronizados. ¿Qué pasaría cuando en medio de todo esto se detectara un carácter entrante en la línea RXD?”. Esto plantea una pregunta interesante. Para

responderla, es necesario entender la estructura interna del puerto serial. El puerto serial del 8051 consta de dos registros SBUF físicamente separados, como indica la figura 5-1. El registro de sólo escritura para transmisión está sincronizado mediante un reloj de velocidad en baudios de transmisión, mientras que el registro de sólo lectura para recepción está sincronizado mediante un reloj de velocidad en baudios de recepción. Ambos relojes están basados en una velocidad de desbordamiento del temporizador 1, como hemos visto (consulte la sección 5.5.2 Modo 1). No obstante, y a pesar de esto, ambos relojes de velocidad en baudios son separados e independientes.

La transmisión se inicia al escribir en el SBUF, pero sólo empieza cuando se detecta el siguiente desbordamiento del reloj de velocidad en baudios de transmisión. Por otra parte, al detectar cualquier carácter entrante, la transición de 1 a 0 que denota al bit de inicio reiniciará el reloj de velocidad en baudios de recepción inmediatamente, de manera que el 8051 esté sincronizado con el dispositivo conectado al puerto serial.

Hacia el final de este capítulo se incluye un ejemplo que demuestra la operación full duplex del puerto serial.

5.7 INICIALIZACIÓN Y ACCESO A LOS REGISTROS DEL PUERTO SERIAL

5.7.1 Habilitación del receptor

El bit de habilitación del receptor (REN) en SCON debe activarse mediante el software para habilitar la recepción de caracteres. Por lo general esto se hace al inicio de un programa, cuando se inicializan el puerto serial, los temporizadores, etc. Puede efectuarse en dos formas. La instrucción

```
SETB REN
```

activa el REN en forma explícita, o la instrucción

```
MOV SCON, #xxx1xxxxB
```

establece en 1 el REN y restablece en 1 o 0 los otros bits en SCON, según se requiera. (Las x deben ser unos o ceros para establecer el modo de operación).

5.7.2 El noveno bit de datos

El noveno bit de datos transmitido en los modos 2 y 3 debe cargarse en TB8 mediante el software. El nuevo bit de datos recibido se transfiere a RB8. El software puede o no requerir un noveno bit de datos, dependiendo de las especificaciones del dispositivo serial con el cual se establecen las comunicaciones. (El noveno bit de datos también cumple una función importante en las comunicaciones entre múltiples procesadores. Consulte la siguiente sección).

5.7.3 Agregar un bit de paridad

A menudo, el noveno bit se utiliza para agregar un bit de paridad a un carácter. Como vimos en el capítulo 2, en la palabra de estado del programa (PSW), el bit P se activa o limpia en cada ciclo de máquina para establecer una paridad par con los ocho bits incluidos en el acumulador. Si, por ejemplo, las comunicaciones requieren de ocho bits de datos además de una paridad par, podemos utilizar las siguientes instrucciones para transmitir los ocho bits en el acumulador con una paridad par agregada en el noveno bit:

```
MOV C, P ; ALMACENA BIT DE PARIDAD PAR EN TB8
MOV TB8, C ; ESTE SE CONVIERTE EN EL NOVENO BIT
MOV SBUF, A ; TRANSFIERE 8 BITS DEL ACUMULADO A SBUF
```

Si requerimos de una paridad impar, podemos modificar las instrucciones como sigue:

```
MOV C, P           ; ALMACENA BIT DE PARIDAD PAR EN BANDERA C
CPL C             ; CONVIERTA A UNA PARIDAD IMPAR
MOV TB8,C
MOV SBUF,A
```

Desde luego, el uso de la paridad no está limitado a los modos 2 y 3. En el modo 1, los ocho bits de datos transmitidos pueden constar de siete bits de datos, además de un bit de paridad. Podemos utilizar las siguientes instrucciones para transmitir un código ASCII de 7 bits con paridad par en el bit 8:

```
CLR ACC.7          ; ASEGURA QUE EL BIT MAS SIGNIFICATIVO
                    ; ESTA LIMPIO
                    ; PARIDAD PAR ESTA EN P
MOV C, P            ; COPIA A C
MOV ACC.7,C          ; ALMACENA PARIDAD PAR AL BIT MAS
                    ; SIGNIFICATIVO
MOV SBUF,A          ; ENVIA CARACTER
                    ; 7 BITS DE DATOS ADEMÁS DE UNA PARIDAD PAR
```

5.7.4 Banderas de interrupción

En el SCON, las banderas de interrupción de recepción y transmisión (RI y TI) cumplen una función importante en las comunicaciones seriales del 8051. Ambos bits se activan mediante el hardware, pero deben ser puestos en 0 mediante el software.

La bandera RI se activa por lo común al final de la recepción de un carácter, e indica que el “búfer de recepción está lleno”. Esta condición se prueba en el software o se programa para generar una interrupción. (En el capítulo 6 veremos las interrupciones). Si el software desea recibir un carácter de entrada del dispositivo conectado al puerto serial (tal vez una terminal de visualización de video), debe esperar a que la bandera RI se establezca en 1, y luego restablecerla en 0 y leer el carácter del SBUF. Esto se muestra a continuación:

```
ESPERA:    JNB    RI,ESPERA ; VERIFICA RI HASTA QUE SEA 1
            CLR    RI        ; PONE EN 0 RI
            MOV    A,SBUF     ; LEE CARACTER
```

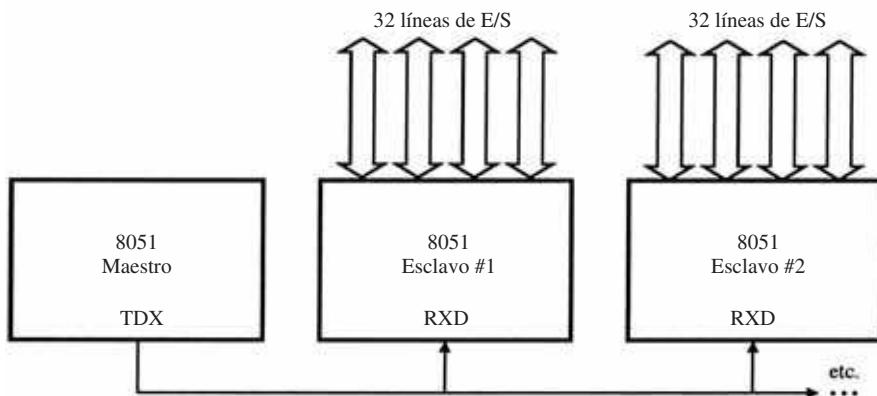
TI se activa al final de la transmisión de un carácter e indica que el “búfer de transmisión está vacío”. Si el software desea enviar un carácter al dispositivo conectado al puerto serial, tiene que verificar primero si el puerto serial está listo. En otras palabras, si antes envió un carácter, debe esperar hasta que termine esa transmisión para poder enviar el siguiente carácter. Las siguientes instrucciones transmiten el carácter en el acumulador:

```
ESPERA:    JNB    TI,ESPERA ; VERIFICA RI HASTA QUE SEA 1
            CLR    TI        ; PONE EN 0 RI
            MOV    SBUF,A     ; LEE CARACTER
```

Estas secuencias de instrucciones para recibir y transmitir a menudo forman parte de subrutinas estándar para efectuar la entrada y salida de caracteres. Los ejemplos 5.2 y 5.3 describen esto con mayor detalle.

5.8 COMUNICACIONES ENTRE MÚLTIPLES PROCESADORES

Los modos 2 y 3 cuentan con una prevención especial para posibilitar las comunicaciones entre múltiples procesadores. En estos modos se reciben nueve bits de datos y el noveno bit se almacena en

**FIGURA 5-7**

Comunicación entre múltiples procesadores

RB8. El puerto se puede programar para que la interrupción del puerto serial se active sólo si RB8 = 1, cuando se recibe el bit de detención. Esta característica se habilita al activar el bit SM2 en el SCON. Un ejemplo de esta aplicación tiene lugar en un ambiente de conexión de redes, donde se utilizan múltiples 8051 en un arreglo tipo maestro/esclavo, como indica la figura 5-7.

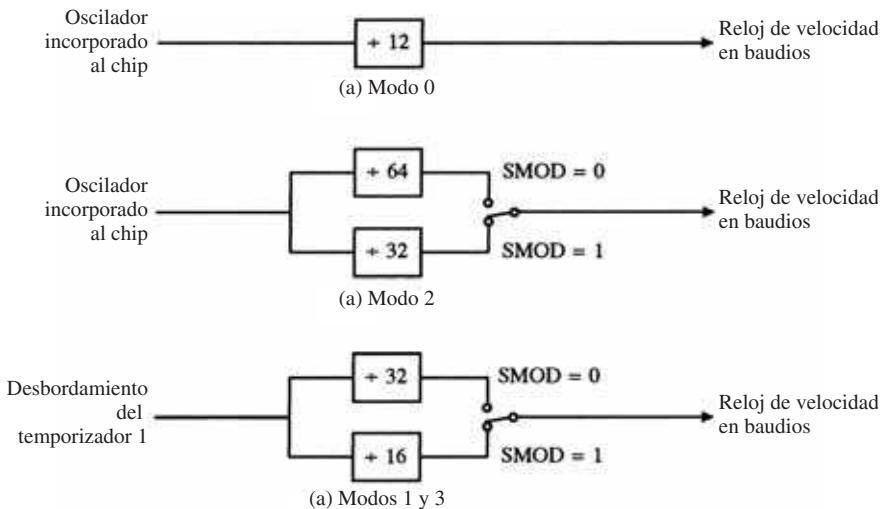
Cuando el procesador maestro desea transmitir un bloque de datos a uno de sus esclavos, envía primero un byte de dirección que identifica al esclavo destino. Un byte de dirección difiere de un byte de datos en cuanto a que el noveno bit es igual a 1 en el byte de dirección, e igual a 0 en el byte de datos. No obstante, con un byte de dirección se interrumpen todos los esclavos para que cada uno pueda examinar el byte recibido y verificar si es el esclavo destino. El esclavo direccionado puede poner en 0 su bit SM2 y prepararse para recibir los bytes de datos que siguen. Los esclavos no direccionaldos dejan en 1 sus bits SM2 y continúan su operación normal e ignoran los bytes de datos entrantes. Los esclavos serán interrumpidos de nuevo cuando el procesador maestro envíe el siguiente byte de dirección. Podemos pensar en casos especiales donde un esclavo puede también transmitir al maestro, una vez que el vínculo entre maestro/esclavo ha sido establecido. Lo importante es no utilizar el noveno bit luego de establecer un vínculo (debido a la posibilidad de seleccionar accidentalmente a otros esclavos).

El bit SM2 no tiene efecto alguno en el modo 0, y en el modo 1 se puede utilizar para verificar la validez del bit de detención. En la recepción en el modo 1, si SM2 = 1, la interrupción de recepción no se activará a menos que se reciba un bit de detención válido.

5.9 VELOCIDADES EN BAUDIOS DEL PUERTO SERIAL

En la tabla 5-2, resulta evidente que la velocidad en baudios está fija en los modos 0 y 3. En el modo 0, la velocidad en baudios siempre es igual a la frecuencia del oscilador incorporado al chip, dividida entre 12. A menudo, un cristal controla el oscilador incorporado al 8051, pero también se puede utilizar otra fuente de sincronización. (Consulte el capítulo 2). Si asumimos una frecuencia nominal del oscilador de 12 MHz, entonces la velocidad en baudios del modo 0 es de 1 MHz. (Consulte la figura 5-8a).

De manera predeterminada, después de un reinicio del sistema, la velocidad en baudios del modo 2 es igual a la frecuencia del oscilador dividida entre 64. La velocidad en baudios también se ve afectada por un bit en el registro de control de poder, PCON. El bit 7 del PCON es el bit del SMOD. Activar el SMOD tiene el efecto de duplicar la velocidad en baudios en los modos 1, 2 y 3.

**FIGURA 5-8**

Fuentes de sincronización del puerto serial: (a) Modo 0, (b) Modo 2, (c) Modos 1 y 3

En el modo 2, la velocidad en baudios puede duplicarse desde un valor predeterminado de 1/64 de la frecuencia del oscilador (SMOD = 0), hasta 1/32 de la frecuencia del oscilador (SMOD = 1). (Consulte la figura 5-8b).

Debido a que el PCON no es direccionable por bit, la activación del SMOD sin alterar los otros bits del PCON requiere una operación de “lectura-modificación-escritura”. Las siguientes instrucciones activan al SMOD:

```

MOV    A, PCON           ; OBTIENE VALOR ACTUAL DE PCON
SETB   ACC.7             ; ACTIVA BIT 7 (SMOD)
MOV    PCON, A            ; ESCRIBE VALOR AL PCON

```

Las velocidades en baudios del 8051 en los modos 1 y 3 se determinan con base en la velocidad de desbordamiento del temporizador 1. Debido a que el temporizador opera a una frecuencia relativamente alta, el desbordamiento se divide también entre 32 (16 si SMOD = 1) antes de proporcionar el reloj de velocidad en baudios al puerto serial. La velocidad en baudios del 8052 en los modos 1 y 3 está determinada ya sea por la velocidad de desbordamiento del temporizador 1 o por la del temporizador 2, o por ambas a la vez.

5.9.1 Uso del temporizador 1 como el reloj de velocidad en baudios

Si por el momento consideramos sólo al 8051, la técnica más común para generar la velocidad en baudios es la de inicializar al TMOD en el modo de autorrecarga de 8 bits (modo 2 del temporizador) y colocar el valor de recarga correcto en TH1 para proporcionar la velocidad de desbordamiento para la velocidad en baudios. El TMOD se inicializa como sigue:

```
MOV    TMOD, #0010xxxxB
```

Las x se sustituyen por unos o ceros, según lo requiera el temporizador 0.

Ésta no es la única posibilidad. Podemos alcanzar velocidades en baudios muy bajas usando el modo de 16 bits, o el modo 2 del temporizador, con TMOD = 0001xxxxB. Sin embargo, esto ocasiona una ligera pérdida de tiempo y recursos en el software, ya que los registros TH1/TL1 deben reinicializarse después de cada desbordamiento. Esto se realizaría en una rutina de servicio de interrupciones. Otra opción es la de sincronizar al temporizador 1 en forma externa usando la terminal T1 (P3.5). Independientemente de esto, la velocidad en baudios es la velocidad de desbordamiento del temporizador 1 dividida entre 32 (o entre 16, si SMOD = 1).

Por ejemplo, la fórmula para determinar la velocidad en baudios en los modos 1 y 3 es

$$\text{VELOCIDAD EN BAUDIOS} = \frac{\text{VELOCIDAD DE DESBORDAMIENTO DEL TEMPORIZADOR 1}}{32}$$

Por ejemplo, una operación de 1200 baudios requiere una velocidad de desbordamiento que se calcula de la siguiente forma:

$$1200 = \frac{\text{VELOCIDAD DE DESBORDAMIENTO DEL TEMPORIZADOR 1}}{32}$$

$$\text{VELOCIDAD DE DESBORDAMIENTO DEL TEMPORIZADOR 1} = 38.4 \text{ kHz}$$

Si un cristal de 12 MHz controla al oscilador incorporado al chip, el temporizador 1 está sincronizado a una velocidad de 1 MHz o 1000 kHz. Como el temporizador debe desbordarse a una velocidad de 38.4 kHz y está sincronizado a 1000 kHz, se requiere un desbordamiento cada $1000 \div 38.4 = 26.04$ pulsos de reloj. (Se redondea a 26). Y debido a que el temporizador cuenta hacia arriba y se desborda en la transición de FFH a 00H del conteo, se requiere un valor de recarga para TH1 de 26 conteos menos que 0. El valor correcto, por lo tanto, es de -26. La manera más fácil de colocar este valor de recarga en TH1 es con

```
MOV TH1, #-26
```

El ensamblador realizará la conversión necesaria. En este caso, el valor de -26 se convierte a 0E6H; por lo tanto, la instrucción previa es idéntica a

```
MOV TH1, #0E6H
```

Existe un ligero error en la velocidad en baudios resultante debido al redondeo. Por lo general, podemos tolerar un 5% de error al utilizar comunicaciones asíncronas (inicio/detención). Es posible obtener velocidades en baudios exactas si utilizamos un cristal de 11.059 MHz. La tabla 5-3 presenta un resumen de los valores de recarga para TH1, para las velocidades en baudios más comunes, con un cristal de 12.000 MHz o uno de 11.059 MHz.

TABLA 5-3

Resumen de las velocidades en baudios

Velocidad en baudios	Frecuencia del cristal	SMOD	Valor de recarga del TH1	Velocidad en baudios real	Error
9600	12.000 MHz	1	-7 (F9H)	8923	7%
2400	12.000 MHz	0	-13 (F3H)	2404	0.16%
1200	12.000 MHz	0	-26 (E6H)	1202	0.16%
19200	11.059 MHz	1	-3 (FDH)	19200	0
9600	11.059 MHz	0	-3 (FDH)	9600	0
2400	11.059 MHz	0	-12 (F4H)	2400	0
1200	11.059 MHz	0	-24 (E8H)	1200	0

EJEMPLO Inicialización del puerto serial**5.1**

Escriba una secuencia de instrucciones para inicializar el puerto serial de manera que opere como un UART de 8 bits a 2400 baudios. Utilice el temporizador 1 para proporcionar el reloj de velocidad en baudios.

Solución

Para este ejemplo, debemos inicializar cuatro registros: SMOD, TMOD, TCON y TH1. Los valores requeridos se resumen a continuación.

	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
SCON :	0	1	0	1	0	0	1	0
	GTE	C/T	M1	M0	GTE	C/T	M1	M0
TMOD :	0	0	1	0	0	0	0	0
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
TCON :	0	1	0	0	0	0	0	0
TH1 :	1	1	1	1	0	0	1	1

Cuando establecemos que $SM0/SM1 = 0/1$, configuramos el puerto serial en el modo de UART de 8 bits. $REN = 1$ habilita el puerto serial para la recepción de caracteres. Al activar $TI = 1$ se permite la transmisión del primer carácter al indicar que el búfer de transmisión está vacío. Para el TMOD, si establecemos que $M1/M0 = 1/0$, el temporizador 1 se configura en el modo de auto-recarga de 8 bits. Al activar $TR1 = 1$ en TCON se activa el temporizador 1. Los otros bits se muestran como ceros, debido a que controlan características o modos no utilizados en este ejemplo.

Análisis

El valor requerido de TH1 es aquel que proporciona desbordamientos a una velocidad de $2400 \times 32 = 76.8$ kHz. Si asumimos que el 8051 está sincronizado mediante un cristal de 12 MHz, el temporizador 1 está sincronizado a una velocidad de 1 MHz o 1000 kHz, y el número de pulsos de reloj para cada desbordamiento es de $1000 \div 76.8 = 13.02$. (Se redondea a 13). El valor de recarga es de -13 o 0F3H.

La secuencia de instrucciones para la inicialización se muestra enseguida.

Solución para el ejemplo 5.1 Ejemplo del puerto serial del 8051 (inicialización del puerto serial)

```

8100      5          ORG  8100H
8100 759852 6  INICIA: MOV  SCON, #52H ;puerto serial,
                  modo 1
8103 758920 7          MOV  TMOD, #20H ;temporizador 1, modo 2
8106 758DF3 8          MOV  TH1, #-13 ;conteo de recarga para
                  2400 baudios
8109 D28E   9          SETB TR1        ;activa temporizador 1
10          END

```

EJEMPLO Subrutina de carácter de salida**5.2**

Escriba una subrutina llamada CARSAL para transmitir un código ASCII de 7 bits, localizado en el acumulador, hacia el puerto serial del 8051, con paridad impar que se agrega a través del octavo bit. Regrese de la subrutina dejando al acumulador intacto, es decir, conteniendo el mismo valor que tenía antes de la llamada a la subrutina.

Solución

Este ejemplo, así como el siguiente, ilustra dos de las subrutinas más comunes que se utilizan en los sistemas de microcomputadoras con una terminal RS232 conectada: carácter de salida (CARSAL) y carácter de entrada (CARENT).

```

8100      5          ORG  8100H
8100  A2D0  6  CARSAL:  MOV  C,P      ;transfiere bit de paridad a
                        ;la bandera C
8102  B3      7          CPL  C      ;cambia a paridad impar
8103  92E7  8          MOV  ACC.7,C  ;suma al código del carácter
8105  3099FD 9  DENUEVO: JNB  TI,DENUEVO ;¿Tx vacío? no:verifica de
                                         ;nuevo
8108  C299  10         CLR  TI      ;sí: restablece bandera en 0 y
810A  F599  11         MOV  SBUF,A  ;envía el carácter
810C  C2E7  12         CLR  ACC.7  ;remueve el bit de paridad y
810E  22      13         RET           ; regresa
                                         14         END

```

Análisis

Las primeras tres instrucciones transfieren una paridad impar al bit 7 del acumulador. Debido a que el bit P en la PSW establece una paridad par con el acumulador, el bit se complementa antes de transferirse a ACC.7. La instrucción JNB crea un “ciclo de espera”, en el cual se prueba repetidamente la bandera de interrupción de transmisión (TI) hasta que se establece en 1. Cuando TI es 1 (debido a que la transmisión del carácter anterior ha terminado), es puesto en 0 y entonces el carácter ubicado en el acumulador se escribe en el búfer del puerto serial (SBUF). La transmisión comienza en el siguiente desbordamiento del contador de división entre 16, que sincroniza al puerto serial. (Consulte la figura 5-5). Por último, ACC.7 se restablece en 0 para que el valor de regreso sea el mismo que el código de 7 bits transferido a la subrutina.

La subrutina CARSAL es un bloque de construcción y tiene poca utilidad por sí misma. A un “nivel más alto”, esta subrutina se llama para transmitir un solo carácter o una cadena de caracteres. Por ejemplo, las siguientes instrucciones transmiten el código ASCII para la letra “Z” al dispositivo serial conectado al puerto serial del 8051:

```

MOV  A, #'Z'
CALL CARSAL
(continua)

```

Como extensión natural de esta idea, el problema 1 incluido al final de este capítulo utiliza la subrutina CARSAL como bloque de construcción en una subrutina CADSAL (cadena de salida), la cual transmite una secuencia de códigos ASCII (terminada por un byte NULO, 00H) al dispositivo serial conectado al puerto serial del 8051.

EJEMPLO Subrutina de carácter de entrada**5.3**

Escriba una subrutina llamada CARENT para que utilice un carácter del puerto serial del 8051 como entrada y regrese el código ASCII de 7 bits incluido en el acumulador. Espere una paridad impar en el octavo bit recibido y establezca en 1 la bandera de acarreo si hay un error de paridad.

Solución

```

8100      5      ORG 8100H
8100 3098FD 6  CARENT: JNB RI,$ ;espera al carácter
8103 C298   7      CLR RI    ;limpia la bandera
8105 E599   8      MOV A,SBUF ;lee carácter y lo pone en A
8107 A2D0   9      MOV C,P   ;para una paridad impar en A,
                           ;P debe activarse
8109 B3     11     CPL C    ;complementando en forma
                           ;correcta
                           ;indica si hubo error
810A C2E7   12     CLR ACC.7 ;remueve paridad
810C 22     13     RET
810C 22     14     END
                           ;indica si hubo error
                           ;remueve paridad

```

Análisis

Esta subrutina comienza esperando a que la bandera de interrupción de recepción (RI) se establezca en 1, lo cual indica que un carácter está esperando para leerse del SBUF. Cuando RI es 1, la instrucción JNB continúa la ejecución en la siguiente instrucción. RI es puesto en 0, se lee el código en SBUF y se coloca en el acumulador. El bit P en la PSW establece una paridad par con el acumulador, así que debe activarse cuando el acumulador, por sí mismo, contiene una paridad impar correcta en el bit 7. Al transferir el bit P a la bandera de acarreo y complementarla, CY = 0 si no ocurrió algún error. Por otra parte, si el acumulador contiene un error de paridad, entonces CY = 1, lo cual indica correctamente que existe un “error de paridad”. Finalmente, ACC.7 se pone en 0 para asegurar que sólo se regresa un código de 7 bits al programa que llamó a la subrutina.

EJEMPLO**5.4****Operación full duplex**

Escriba un programa que transmita caracteres en forma continua desde un búfer de transmisión (memoria RAM interna en las direcciones 30H a 4FH). Si detecta caracteres entrantes en el puerto serial, almacénelos en el búfer de recepción empezando en la ubicación de memoria RAM interna 50H. Asuma que el puerto serial del 8051 está inicializado en el modo 1.

Solución

Este ejemplo muestra cómo podemos utilizar al puerto serial en operación full duplex, es decir, que la transmisión y la recepción se realizan al mismo tiempo.

```

8100      1      ORG 8100H
8100 7830  2      MOV R0, #30H ;apuntador para el búfer de tx
8102 7950  3      MOV R1, #50H ;apuntador para el búfer de rx
8104 209819 4  CICLO:JB RI, RECIBE ;¿se recibió el carácter?
                           ; sí: procesa el carácter
                           ; no: continúa verificación
                           ; sí: procesa el carácter
                           ; no: continúa verificación
8107 209902 6      JB TI, TX ;¿se transmitió el
                           ; carácter anterior?
                           ; sí: procesa el carácter
                           ; no: continúa verificación
810A 80F8   9      SJMP CICLO ;no: continúa verificación
810C E6   10 TX:  MOV A, @R0 ;obtén carácter de buffer
                           ; de tx
                           ; de tx

```

810D A2D0	12	MOV C, P	;transfiere bit de paridad a C
810F B3	13	CPL C	;cambia a paridad impar
8110 92E7	14	MOV ACC.7, C	;agrega al código de carácter
8112 C299	15	CLR TI	;pone en 0 la bandera de transmisión
8114 F599	16	MOV SBUF, A	;Envía carácter
8116 C2E7	17	CLR ACC.7	;remueve bit de paridad
8118 08	18	INC R0	;apunta al siguiente carácter en el búfer
	19		
8119 B850E8	20	CJNE R0, #50H, CICLO	;¿fin del búfer?
	21		; no: continúa
811C 7830	22	MOV R0, #30H	; sí: recicla
811E 80E4	23	SJMP CICLO	;continúa verificación
	24		
8120 C298	25	RX: CLR RI	;pone en 0 bandera de recepción
8122 E599	26	MOV A, SBUF	;lee carácter a A
8124 A2D0	27	MOV C, P	;P debe activarse para paridad impar en A
	28		
8126 B3	29	CPL C	;complemento
	30		; correcto indica
	31		; "error"
8127 C2E7	32	CLR ACC.7	;remueve bit de paridad
8129 F7	33	MOV @R1, A	;almacena el carácter recibido en el búfer
	34		
812A 09	35	INC R1	;apunta a la siguiente ubicación en el búfer
	36		
812B 80D7	37	SJMP CICLO	;continúa verificación
	38	END	

Análisis

Este programa utiliza primero los dos registros R0 y R1 para apuntar a los búferes de transmisión y recepción, respectivamente. Después verifica si se ha recibido un carácter o si se ha enviado el primer carácter a transmitir. Observe que la recepción se procesa primero. Esto es porque la recepción resulta más crítica ya que el 8051 depende de un dispositivo externo para el carácter entrante, el cual debe procesarse y almacenarse tan pronto como sea posible, de otra manera los siguientes caracteres entrantes podrían sobreescribirlo. Recuerde que el puerto serial puede almacenar cuando mucho un carácter en su búfer mientras está recibiendo un segundo carácter. Si se ha recibido completamente un carácter entrante, se lee del SBUF y se verifica la paridad antes de que se almacene en el búfer de recepción cuya ubicación vacía es apuntada por R1. R1 se incrementa para apuntar a la siguiente ubicación disponible y el programa vuelve a verificar. Cuando la transmisión previa termina, el programa obtiene primero el carácter a transmitir del búfer de transmisión. R0 se utiliza como el apuntador al siguiente carácter en el búfer de transmisión. La paridad impar se transfiere al bit 7 del código antes de que éste se envíe al SBUF para su transmisión. Entonces R1 se incrementa para apuntar al siguiente carácter. El programa también verifica si se ha llegado al fin del búfer de transmisión, y si esto ocurre el programa se reciclará al inicio del búfer. Finalmente, el programa regresa a verificar si ocurren recepciones o transmisiones adicionales.

EJEMPLO**5.5****Compensación de los errores de redondeo**

Calcule el error de redondeo para la velocidad en baudios del puerto serial analizado en el ejemplo 5.1. Si suponemos que vamos a utilizar el mismo valor de recarga para el temporizador 1 de -13 o $0F3H$, calcule el valor de la frecuencia del cristal para obtener más velocidad en baudios exacta de 2400 baudios.

Solución

El error de redondeo para la velocidad en baudios es:

$$\begin{aligned}\text{Error de redondeo} &= \frac{|\text{baudios}_{\text{deseados}} - \text{baudios}_{\text{redondeados}}|}{\text{baudios}_{\text{deseados}}} \times 100\% \\ &= \frac{|2400 - 2403.8|}{2400} \times 100\% \\ &= 0.158\% \\ f_{\text{deseada}} &= \frac{\text{baudios}_{\text{deseados}}}{\text{baudios}_{\text{redondeados}}} \times f_{\text{redondeada}} \\ &= \frac{2400}{2403.8} \times 12 \text{ MHz} \\ &= 11.98 \text{ MHz}\end{aligned}$$

Análisis

La velocidad en baudios deseada es de 2400 baudios. El valor de recarga del temporizador 1 redondeado es de -13 para $13 \mu s$, lo cual genera una velocidad de desbordamiento de 76.9 kHz, que a su vez genera una velocidad en baudios de $76.9 \text{ kHz} \div 32 = 2403.8$ baudios.

Al calcular la frecuencia del cristal deseada para producir una velocidad en baudios exacta de 2400 baudios, podemos utilizar la técnica de relación, como se muestra en la solución anterior. Verifiquemos si f_{deseada} genera una velocidad en baudios exacta de 2400. Una frecuencia del cristal de 11.98 MHz significa que el temporizador 1 se incrementa a una velocidad de $11.98 \text{ MHz}/12 = 0.998 \text{ MHz}$, así que la duración de un conteo es de $1/0.998 \text{ MHz} = 1.002 \mu s$. El valor de recarga del temporizador 1 es de -13 , así que se desborda cada $13 \times 1.002 \mu s = 1.3026 \mu s$, por lo tanto la velocidad de desbordamiento es de $1/1.3026 \mu s = 76.77 \text{ kHz}$. Esto genera una velocidad en baudios de $76.77 \text{ kHz} \div 32 = 2399$ baudios ≈ 2400 baudios.

Una frecuencia de cristal que se utiliza más a menudo es la de 11.059 MHz, en vez de 11.98MHz. En este caso, el valor de recarga del temporizador 1 tendría que ser -12 en vez de -13 . Para comprobar que resulte en una velocidad de baudios exacta de 2400, hay que considerar que el temporizador 1 se incrementa a una velocidad de $11.059 \text{ MHz}/12 = 0.9216 \text{ MHz}$, por lo que la duración de una cuenta es de $1/0.9216 \text{ MHz} = 1.085 \mu s$. El valor de recarga para el temporizador 1 es de -12 , por lo que se desborda cada $12 \times 1.085 \mu s = 13.02 \mu s$, por lo que la velocidad de desbordamiento es $1/13.02 \mu s = 76.8 \text{ kHz}$. Esto produce una velocidad en baudios de $76.8 \text{ kHz} \div 32 = 2400$ baudios.

Es evidente que la frecuencia del cristal necesaria para eliminar los errores de redondeo en la velocidad de baudios varía dependiendo del valor de recarga del temporizador 1 que se utilice.

RESUMEN

En este capítulo presentamos los detalles principales que se requieren para programar el puerto serial del 8051. Hemos mencionado superficialmente el uso de interrupciones en

éste y en el último capítulo. De hecho, las aplicaciones más avanzadas en que se utilizan los temporizadores o el puerto serial del 8051 a menudo requieren operaciones de entrada/salida sincronizadas mediante interrupciones. Este es el tema del siguiente capítulo.

PROBLEMAS

Los siguientes problemas son típicos de las rutinas de software para la interfaz de terminales (u otros dispositivos seriales) con una microcomputadora. Asuma que el puerto serial del 8051 se inicializa en el modo de UART de 8 bits, y que la velocidad en baudios se genera mediante el temporizador 1.

- 5.1** Escriba una subrutina llamada CADSAL, la cual debe enviar una cadena de códigos en ASCII que termina en un carácter nulo al dispositivo (tal vez una terminal de visualización de video) conectado al puerto serial del 8051. Asuma que la cadena de códigos en ASCII está en la memoria externa para código y que el programa que llama a la subrutina transfiere la dirección de la cadena al apuntador de datos antes de llamar a CADSAL. Una cadena que termina en un carácter nulo es una serie de bytes en ASCII que termina con un valor de byte igual a 00H.
- 5.2** Escriba una subrutina llamada ENLINEA que envíe como entrada una línea de códigos en ASCII de un dispositivo conectado al puerto serial del 8051 y la transfiera a la memoria interna para datos, comenzando en la dirección 50H. Asuma que la línea termina con un código de retorno de carro. Transfiera el código de retorno de carro en el búfer de la línea junto con los otros códigos, y entonces termine el búfer de la línea con un byte nulo (00H).
- 5.3** Escriba un programa que envíe en forma continua el alfabeto (en minúsculas) a un dispositivo conectado al puerto serial del 8051. Utilice la subrutina CARSAL escrita anteriormente.
- 5.4** Asuma la existencia de la subrutina CARSAL y escriba un programa que envíe en forma continua el conjunto de códigos ASCII que pueden visualizarse (códigos 20H a 7EH) al dispositivo conectado al puerto serial del 8051.
- 5.5** Modifique la solución al problema anterior para suspender y resumir la salida a la pantalla, utilizando los códigos XOFF (apagado) y XON (encendido) que se introducirán mediante el teclado. Todos los demás códigos recibidos deberán ignorarse. (Nota: XOFF = CONTROL-S = 13H, XON = CONTROL-Q = 11H).
- 5.6** Asuma la existencia de las subrutinas CARENT y CARSAL y escriba un programa que reciba caracteres del teclado y los repita en pantalla, convirtiendo los caracteres de minúsculas a mayúsculas.
- 5.7** Asuma la existencia de las subrutinas CARENT y CARSAL y escriba un programa que reciba caracteres del dispositivo conectado al puerto serial del 8051 y los repita en pantalla sustituyendo cualquier carácter de control por un punto (.) (códigos en ASCII del 00H al 1FH y 7FH).
- 5.8** Asuma la existencia de la subrutina CARSAL, y escriba un programa que limpie la pantalla en la terminal de visualización de video conectada al puerto serial del 8051 y que después envíe su nombre a la terminal de video 10 veces en 10 líneas separadas. La función para borrar la pantalla en las terminales de video se realiza mediante la transmisión de las secuencias de escape CONTROL-Z en muchas de las terminales o <ESC>[2J en terminales que soportan ANSI (American National Standards Institute). Utilice cualquiera de estos métodos en su solución.
- 5.9** La figura 5-4 ilustra una técnica para expandir la capacidad de salida del 8051. Asuma dicha configuración y escriba un programa que inicialice el puerto serial del 8051 en el modo

- de desplazamiento de registros, y que después haga un mapa en las ocho salidas adicionales del contenido de la memoria interna en la ubicación 20H, 10 veces por segundo.
- 5.10** ¿Qué tipo de comunicación serial soporta el puerto serial del 8051? ¿Simple, half-duplex o full-duplex? ¿Por qué? Explique los componentes localizados dentro del puerto serial y cómo es que soportan este tipo de comunicación serial.
- 5.11** Explique cómo se puede expandir la capacidad de entrada/salida (E/S) del 8051 mediante el uso del puerto serial. ¿Cuál es la desventaja de dicho método?
- 5.12** Escriba una subrutina llamada CARSAL9 para transmitir un código de 9 bits ($C_8C_7C_6C_5C_4C_3C_2C_1C_0$) a través del puerto serial del 8051. Observe que C_8 es el bit menos significativo del registro B mientras que $C_7C_6C_5C_4C_3C_2C_1C_0$ están en el acumulador. Regrese de la subrutina con todos los registros intactos.
- 5.13** Escriba una subrutina llamada CARENT8 para recibir un carácter en ASCII extendido de 8 bits del puerto serial y regresar con un código de 8 bits en el acumulador. Espere una paridad impar en el noveno bit recibido y establezca en 1 la bandera de acarreo si ocurre un error de paridad.
- 5.14** Escriba una subrutina llamada CARSAL8 para transmitir el código en ASCII extendido de 8 bits en el acumulador a través del puerto serial del 8051, añadiendo una paridad impar en el noveno bit. Regrese de la subrutina con el acumulador intacto.
- 5.15**
- Escriba una secuencia de instrucciones para inicializar el puerto serial de modo que opere como un UART de 9 bits a 9600 baudios. Utilice al temporizador 1 para generar la velocidad en baudios, y asuma que la frecuencia del cristal oscilador, $f_{osc} = 12 \text{ MHz}$.
 - ¿La velocidad resultante es de 9600 baudios exactos? Si no es así, ¿cuál es el porcentaje de error (% error)? Con esto en mente, calcule el valor de f_{osc} tal que podamos alcanzar una velocidad en baudios exactamente igual a 9600 baudios.

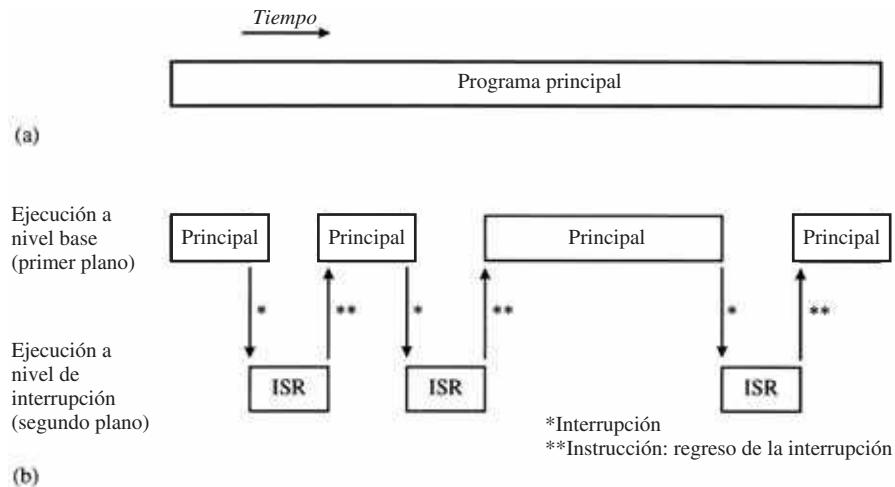
Las interrupciones

6.1 INTRODUCCIÓN

Una **interrupción** es la ocurrencia de una condición (o evento) que ocasiona la suspensión temporal de un programa mientras que otro programa se encarga de servir a dicha condición. Las interrupciones cumplen una función importante en el diseño y la implementación de las aplicaciones con microcontroladores. Las interrupciones permiten que un sistema pueda responder a un evento en forma asíncrona y se encargue del evento mientras se ejecuta otro programa. Un **sistema controlado mediante interrupciones** nos da la falsa percepción de que está realizando muchas cosas en forma simultánea. Por supuesto que la CPU no puede ejecutar más de una instrucción al mismo tiempo; pero sí puede suspender temporalmente la ejecución de un programa, ejecutar otro, y después regresar al primer programa. En cierta manera, esto es como una subrutina. La CPU ejecuta otro programa (la subrutina) y después regresa al programa original. La diferencia está en que, en un sistema controlado mediante interrupciones, la interrupción es la respuesta a un “evento” que ocurre de manera asíncrona con el programa principal. En otras palabras, no sabemos cuándo se interrumpirá el programa principal.

Una **rutina de servicio de interrupción (ISR)**, también conocida como **manejador de interrupciones**, es el programa que se encarga de efectuar una interrupción. La ISR se ejecuta en respuesta a la interrupción y, por lo general, realiza una operación de entrada o de salida sobre un dispositivo. El programa principal suspende su ejecución temporalmente y se bifurca a la ISR cuando ocurre una interrupción; la ISR se ejecuta, realiza la operación, y termina con una instrucción de “regreso de la interrupción”; el programa principal continúa desde donde se quedó. Es algo muy común referirnos a que el programa principal se ejecuta a **nivel base** y que las ISR se ejecutan a **nivel de interrupción**. También utilizamos los conceptos de **primer plano** (nivel base) y **segundo plano** (nivel de interrupción). La figura 6-1 ilustra este breve resumen de las interrupciones y muestra (a) la ejecución de un programa sin interrupciones, (b) la ejecución a nivel base con interrupciones ocasionales y las ISR ejecutándose a nivel de interrupción.

Un ejemplo típico de las interrupciones es el de la entrada manual mediante el uso de un teclado. Considere una aplicación para un horno de microondas. El programa principal (primer plano) podrá controlar un elemento de energía de microondas para cocinar; sin embargo, mientras

**FIGURA 6–1**

Ejecución de un programa con y sin interrupciones. (a) Sin interrupciones. (b) Con interrupciones

cocinamos, el sistema debe poder responder a alguna entrada manual que tenga lugar en la puerta del horno, tal como una petición para reducir o incrementar el tiempo de cocción. Cuando el usuario oprime una tecla, se genera una interrupción (tal vez una señal que cambia de nivel alto a nivel bajo) y el programa principal se interrumpe. La ISR toma el control en el segundo plano, lee el o los códigos del teclado, cambia las condiciones de cocción en forma adecuada, y termina al pasar el control de regreso al programa principal. El programa principal continúa su ejecución desde donde se quedó. Lo importante a destacar en este ejemplo es que la entrada manual ocurre “de manera asíncrona”; es decir, a intervalos que el software que se ejecuta en el sistema no puede predecir o controlar. Esto es una interrupción.

6.2 ORGANIZACIÓN DE LAS INTERRUPCIONES EN EL 8051

Existen cinco fuentes de interrupción en el 8051: dos interrupciones externas, dos de temporizador, y una interrupción del puerto serial. El 8052 agrega una sexta fuente de interrupción mediante el temporizador adicional. Todas las interrupciones se deshabilitan después de una reinicialización del sistema y se habilitan en forma individual mediante el software.

Si ocurren dos o más interrupciones de manera simultánea, o si una interrupción ocurre al mismo tiempo que se le está dando servicio a otra interrupción, existe la secuencia de sondeo y un esquema de prioridades de dos niveles para planificar interrupciones. La secuencia de sondeo es fija, pero la prioridad de interrupción es programable.

Empecemos por examinar las maneras de habilitar y deshabilitar las interrupciones.

6.2.1 Habilitar y deshabilitar interrupciones

Cada una de las fuentes de interrupción se habilita o deshabilita en forma individual mediante el registro de función especial IE (habilitador de interrupción), el cual es direccionable por bit y está

TABLA 6-1

Resumen del registro IE (habilitación de interrupciones)

Bit	Símbolo	Dirección de bit	Descripción (1 = Habilita, 0 = Deshabilita)
IE.7	EA	AFH	Habilitación/deshabilitación global
IE.6	—	AEH	Indefinido
IE.5	ET2	ADH	Habilita interrupción del temporizador 2 (8052)
IE.4	ES	ACH	Habilita interrupción del puerto serial
IE.3	ET1	ABH	Habilita interrupción del temporizador 1
IE.2	EX1	AAH	Habilita interrupción externa 1
IE.1	ET0	A9H	Habilita interrupción del temporizador 0
IE.0	EX0	A8H	Habilita interrupción externa 0

ubicado en la dirección 0A8H. Así como existen bits individuales de habilitación para cada fuente de interrupción, hay un bit de habilitación/deshabilitación global que se desactiva para deshabilitar todas las interrupciones o se activa para habilitarlas. (Consulte la tabla 6-1).

Se deben activar dos bits para habilitar cualquier interrupción: el bit individual de habilitación y el bit de habilitación global. Por ejemplo, las interrupciones del temporizador 1 se habilitan como sigue:

```
SETB    ET1      ; HABILITA INTERRUPCION DE TEMPORIZADOR 1
SETB    EA       ; ACTIVA EL BIT DE HABILITACION GLOBAL
```

También podemos codificar esto como

```
MOV    IE,#10001000B
```

A pesar de que estas dos soluciones tienen exactamente el mismo efecto después de una reinicialización del sistema, el efecto es diferente si se escribe en IE “al instante”, en medio de la ejecución de un programa. La primera solución no tiene ningún efecto en los otros cinco bits del registro IE, mientras que la segunda borra de manera explícita los otros bits. Podemos inicializar el IE con una instrucción de “transferencia de byte” al inicio de un programa (es decir, después de la inicialización o reinicialización del sistema), pero para habilitar y deshabilitar interrupciones durante el tiempo de ejecución de un programa debemos utilizar instrucciones de “activar bit” y “borrar bit” para evitar efectos secundarios en los otros bits del registro IE.

6.2.2 Prioridad de las interrupciones

Cada fuente de interrupción se programa de manera individual con uno de dos niveles de prioridad mediante el registro de función especial direccionable por bit IP (prioridad de interrupción) en la dirección 0B8H. (Consulte la tabla 6-2).

El registro IP se borra después de una reinicialización del sistema para que todas las interrupciones empiecen en el nivel de prioridad más bajo de manera predeterminada. El concepto de “prioridades” permite que una rutina de servicio de interrupciones se interrumpe mediante una interrupción si esta nueva interrupción es de una prioridad más alta que la interrupción a la que se está dando servicio. Esto es simple y directo en el 8051, ya que sólo existen dos niveles de prioridad. Si una ISR de baja prioridad se está ejecutando cuando ocurre una interrupción de alta prioridad, esta ISR se interrumpe. Una ISR de alta prioridad no puede ser interrumpida.

TABLA 6-2

Resumen del registro IP (prioridad de interrupción)

Bit	Símbolo	Dirección de bit	Descripción (1 = alto nivel, 0 = bajo nivel)
IP.7	—	—	Indefinido
IP.6	—	—	Indefinido
IP.5	PT2	0BDH	Prioridad de interrupción del temporizador 2 (8052)
IP.4	PS	0BCH	Prioridad de interrupción del puerto serial
IP.3	PT1	0BBH	Prioridad de interrupción del temporizador 1
IP.2	PX1	0BAH	Prioridad de interrupción externa 1
IP.1	PT0	0B9H	Prioridad de interrupción del temporizador 0
IP.0	PX0	0B8H	Prioridad de interrupción externa 0

El programa principal, que se ejecuta a nivel base y no está asociado con ninguna interrupción, siempre puede interrumpirse sin importar la prioridad de la interrupción. Si ocurren dos interrupciones de diferente prioridad simultáneamente, se le dará servicio a la interrupción de más alta prioridad primero.

6.2.3 Secuencia de sondeo

Si ocurren dos interrupciones de la misma prioridad simultáneamente, una secuencia de sondeo fija determina a cuál se le dará servicio primero. La secuencia de sondeo es: externa 0, temporizador 0, externa 1, temporizador 1, puerto serial, temporizador 2.

La figura 6-2 ilustra las cinco fuentes de interrupción, el mecanismo de habilitación individual y global, la secuencia de sondeo, y los niveles de prioridad. El estado de todas las fuentes de interrupción está disponible mediante sus bits de bandera respectivos en los SFR. Por supuesto que si cualquier interrupción está deshabilitada, entonces no ocurre ninguna interrupción, pero aún así el software puede probar la bandera de interrupción. Los ejemplos del temporizador y del puerto serial presentados en los dos capítulos anteriores utilizaron las banderas de interrupción en forma extensa, sin aplicar interrupciones.

Una interrupción del puerto serial es resultado de un OR lógico de una interrupción de recepción (RI) o una interrupción de transmisión (TI). De la misma forma, las interrupciones del temporizador 2 se generan mediante un desbordamiento del temporizador (TF2) o mediante la bandera de entrada externa (EXF2). La tabla 6-3 presenta un resumen de los bits de bandera que generan interrupciones.

TABLA 6-3

Bits de bandera de interrupción

Interrupción	Bandera	Registro de función especial y posición de bit
Externa 0	IE0	TCON.1
Externa 1	IE1	TCON.3
Temporizador 1	TF1	TCON.7
Temporizador 0	TF0	TCON.5
Puerto serial	T1	SCON.1
Puerto serial	RI	SCON.0
Temporizador 2	TF2	T2CON.7 (8052)
Temporizador 2	EXF2	T2CON.6 (8052)

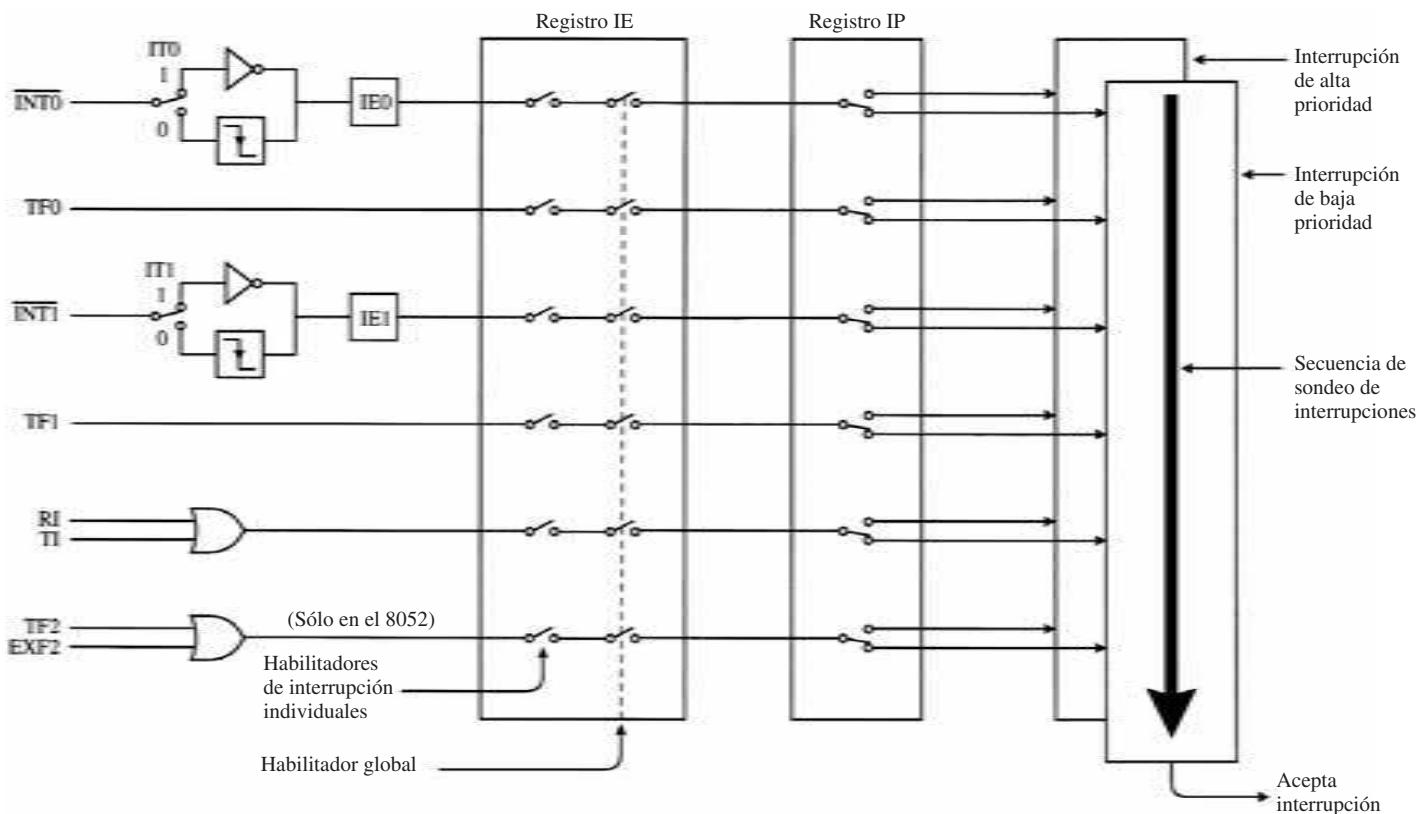


FIGURA 6–2

Información general sobre la estructura de interrupciones del 8051

6.3 PROCESAMIENTO DE INTERRUPCIONES

El programa principal se interrumpe cuando ocurre una interrupción y ésta es aceptada por la CPU. Esto causa las siguientes reacciones:

- La instrucción actual completa su ejecución.
- El contador de programa (PC) se almacena en la pila.
- El estado de la interrupción actual se almacena internamente.
- Las interrupciones se bloquean al nivel de interrupción.
- El contador de programa se carga con la dirección de vector de la rutina de servicio de interrupción (ISR).
- La ISR se ejecuta.

La ISR se ejecuta y toma una acción en respuesta a la interrupción. La ISR termina con una instrucción RETI (regresa de una interrupción). Esto obtiene el valor anterior del contador de programa de la pila y recupera el estado previo a la interrupción. El programa principal continúa su ejecución desde donde se quedó.

6.3.1 Vectores de interrupción

Cuando se acepta una interrupción, el valor cargado en el contador de programa es conocido como **vector de interrupción**. Este valor es la dirección del inicio de la ISR para la fuente que causó la interrupción. Los vectores de interrupción se listan en la tabla 6-4.

El vector de reinicialización del sistema (RST en la dirección 0000H) se incluye en esta tabla ya que, en cierto sentido, es como una interrupción: dicha reinicialización interrumpe el programa principal y carga al contador de programa con un nuevo valor.

Cuando “vectorizamos una interrupción”, el hardware borra de manera automática la bandera que causó la interrupción. Las excepciones a esto son RI y TI para las interrupciones del puerto serial, y TF2 y EXF2 para las interrupciones del temporizador 2. No es práctico que la CPU limpie la bandera de interrupción debido a que existen dos fuentes posibles para cada una de estas interrupciones. Estos bits deben probarse en la ISR para determinar la fuente de interrupción, y después borrar mediante el software la bandera que causó la interrupción. A menudo ocurre una bifurcación hacia la acción apropiada, dependiendo de la fuente de interrupción.

Debido a que los vectores de interrupción se encuentran en la parte inferior de la memoria para código, la primera instrucción del programa principal es a menudo un salto por encima de esta área de memoria, tal como LJMP 0030H.

TABLA 6-4
Vectores de interrupción

Interrupción	Bandera	Dirección de vector
Reinicialización del sistema	RST	0000H
Externa 0	IE0	0003H
Temporizador 0	TF0	000BH
Externa 1	IE1	0013H
Temporizador 1	TF1	001BH
Puerto serial	RI o TI	0023H
Temporizador 2	TF2 o EXF2	002BH

6.4 DISEÑO DE PROGRAMAS MEDIANTE EL USO DE INTERRUPCIONES

Los ejemplos dados en los capítulos 3 y 4 no utilizaron interrupciones, pero sí hicieron uso extenso de “ciclos de espera” para probar las banderas de desbordamiento del temporizador (TF0, TF1 o TF2) o las banderas de transmisión y recepción del puerto serial (TI o RI). El problema de esta solución es que el valioso tiempo de ejecución de la CPU se consume por completo esperando a que las banderas se activen. Esto no es apropiado para las aplicaciones orientadas al control, en donde un microcontrolador debe interactuar con muchos dispositivos de entrada y salida al mismo tiempo.

En esta sección desarrollaremos ejemplos para demostrar métodos prácticos de implementación del software en aplicaciones orientadas al control. El ingrediente principal es la interrupción. Aunque los ejemplos no son necesariamente más grandes, sí resultan más complejos, y reconociendo esto procederemos paso a paso. Sugerimos al lector que siga los ejemplos lentamente y examine el software de modo meticuloso. Algunos de los errores más difíciles de depurar en los diseños de sistemas involucran interrupciones. Debemos comprender todos los detalles de manera íntegra.

Los ejemplos serán completos e independientes debido a que estamos utilizando interrupciones. Cada programa inicia en la dirección 0000H y asumimos que inicia su ejecución después de una reinicialización del sistema. La idea es que estos programas se conviertan en aplicaciones completas que residan en memoria ROM o EPROM.

La estructura sugerida para un programa independiente que utilice interrupciones se muestra a continuación.

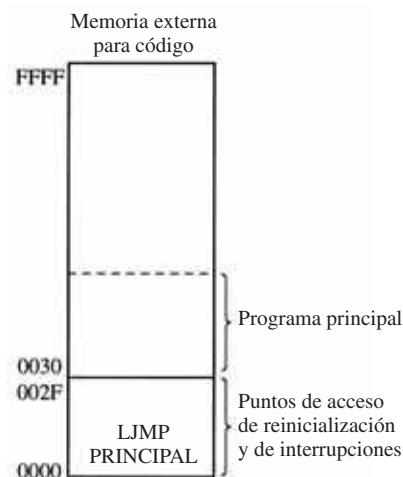
La primera instrucción salta a la dirección 0030H, justo encima de las ubicaciones de vector donde empiezan las ISR, como se lista en la tabla 6-4. La figura 6-3 muestra que el programa principal inicia en la dirección 0030H.

6.4.1 Rutinas de servicio de interrupción pequeñas

Las rutinas de servicio de interrupción deben iniciar cerca de la parte inferior de la memoria para código en las direcciones mostradas en la tabla 6-4. Aunque sólo existen ocho bytes entre cada punto de acceso de interrupción, a menudo esto es suficiente memoria para realizar la operación deseada y regresar de la ISR al programa principal.

Podemos aplicar la siguiente estructura si sólo utilizamos una fuente de interrupción, por ejemplo el temporizador 0:

```
ORG      0000H      ; REINICIALIZACION  
LJMP    PRINCIPAL  
ORG      000BH      ; PUNTO DE ACCESO DEL  
                ; TEMPORIZADOR 0
```

**FIGURA 6–3**

Organización de la memoria cuando se utilizan interrupciones

```

TOISR:      .           ; ISR DEL TEMPORIZADOR 0 INICIA
.
.
RETI        ; REGRESA AL PROGRAMA PRINCIPAL
PRINCIPAL:  .
.
.
.
```

Si utilizamos más interrupciones, debemos tener cuidado para asegurarnos de que iniciarán en la ubicación correcta (consulte la tabla 6-4) y que la ejecución no se desbordará a la siguiente ISR. En el ejemplo anterior, el programa principal comienza inmediatamente después de la instrucción RETI debido a que sólo utiliza una interrupción.

6.4.2 Rutinas de servicio de interrupción grandes

Si una ISR es mayor de ocho bytes, será necesario transferirla a otra ubicación en la memoria para código o puede traspasar el punto de acceso para la siguiente interrupción. Por lo general, la ISR comienza con un salto a otra área de la memoria para código donde la ISR puede desarrollarse. Podemos utilizar la siguiente estructura si, por el momento, consideramos sólo al temporizador 0:

```

ORG      000H          ; PUNTO DE ENTRADA DE REINICIALIZACION
LJMP    PRINCIPAL
ORG      000BH         ; PUNTO DE ENTRADA DEL TEMPORIZADOR 0
LJMP    TOISR
ORG      0030H         ; POR ENCIMA DE LOS VECTORES DE
                      ; INTERRUPCION

PRINCIPAL: .
.
.

TOISR:   .
.
.

RETI     ; ISR DEL TEMPORIZADOR 0
.
.
; REGRESO AL PROGRAMA PRINCIPAL
```

Para simplificar esto, al principio nuestros programas sólo realizan una cosa a la vez. El programa principal o de primer plano inicializa el temporizador, el puerto serial, y los registros de

interrupción como sea necesario, y no hace nada después. El trabajo se realiza completamente en la ISR. El programa principal consta de la siguiente instrucción después de las instrucciones de inicialización:

AQUI : SJMP AQUI

Cuando ocurre una interrupción, el programa principal se interrumpe en forma temporal mientras la ISR se ejecuta. La instrucción RETI al final de la ISR regresa el control al programa principal, y continúa inactiva. Esto no es tan difícil de comprender como podría pensarse. De hecho, en las aplicaciones orientadas al control, la mayor parte del trabajo se realiza en las rutinas de interrupción.

6.5 INTERRUPCIONES DEL TEMPORIZADOR

Las interrupciones del temporizador ocurren cuando la bandera de desbordamiento del temporizador, TFx, se activa después de un desbordamiento de los registros del temporizador, THx/TLx. La bandera de desbordamiento del temporizador, TFx, se borra de manera automática mediante el hardware cuando el 8051 procede a dar servicio a la interrupción. Por lo tanto, con las interrupciones habilitadas, no es necesario borrar de manera explícita a TFx en el software como hicimos en el capítulo 4, donde utilizamos temporizadores sin interrupciones.

EJEMPLO

6.1

Onda cuadrada mediante el uso de interrupciones del temporizador

Escriba un programa que utilice al temporizador 0 e interrupciones para crear una onda cuadrada de 10 kHz en P1.0.

Solución

0000	5	ORG	0	;punto de acceso de reinicialización
0000 020030	6	LJMP	PRINCIPAL	;salta encima de los vectores de interrupción
000B	7	ORG	000BH	;Vector de interrupción del temporizador 0
000B B290	8	T0ISR:	CPL	P1.0 ;dispara bit de puerto
000D 32	9		RETI	
0030	10	ORG	0030H	;Programa de acceso del programa principal
0030 758902	11	PRINCIPAL:	MOV	TMOD, #02H ;temporizador 0, modo 2
0033 758CCE	12		MOV	TH0, #-50 ;Retraso de 50 µs
0036 D28C	13		SETB	TR0 ;inicia temporizador
0038 75A882	14		MOV	IE, #82H ;habilita interrupción del temporizador 0
003B 80FE	15		SJMP	\$;haz nada

Análisis

Con las interrupciones del temporizador habilitadas, la activación de la bandera del temporizador, TFx, luego de un desbordamiento de los registros del temporizador, THx/TLx, es el evento que genera la interrupción. Este ejemplo aparece en el capítulo 4 sin utilizar interrupciones. La mayor parte de este programa es igual a dicho ejemplo, a excepción de que ahora está organizado mediante la estructura para las interrupciones.

Esta solución es un programa completo. Podemos quemarlo en una memoria EPROM e instalarlo en una computadora de una sola tarjeta basada en el 8051 para que se ejecute. El contador de programa se carga con el valor 0000H inmediatamente después de una reinicialización. La primera instrucción ejecutada es LJMP PRINCIPAL, la cual se bifurca encima de la ISR del

temporizador hacia la dirección 0030H en la memoria para código. Las siguientes tres instrucciones (líneas 11-13) inicializan el temporizador 0 en el modo de autorrecarga de 8 bits, con desbordamientos cada $50\ \mu s$. En la línea 14, la instrucción MOV habilita las interrupciones del temporizador 0, de manera que cada desbordamiento del temporizador genera una interrupción. Desde luego, el primer desbordamiento no ocurrirá sino hasta después de $50\ \mu s$, así que el programa principal continúa hacia el ciclo “haz nada”. Cada $50\ \mu s$ ocurre una interrupción; el programa principal se interrumpe y la ISR del temporizador 0 se ejecuta. La ISR simplemente complementa al bit de puerto (línea 8) y regresa al programa principal (línea 9), donde el ciclo haz nada se ejecuta durante $50\ \mu s$ más.

Observe que la bandera del temporizador, TF0, no se borra en forma explícita mediante el software. Si las interrupciones están habilitadas, el hardware borra automáticamente la TF0 cuando la CPU da servicio a la interrupción.

Debemos estar conscientes de cómo es que la pila está operando, ya que la solución es un programa “completo”. La dirección de regreso de la ISR es la ubicación de la instrucción SJMP. Esta dirección se almacena en la pila interna del 8051 antes de la vectorización a la interrupción y se saca de la pila cuando se ejecuta la instrucción RETI (línea 9). El apuntador de pila se reinitializa al valor 07H de manera predeterminada ya que ésta no se inicializó. La operación de almacenamiento deja la dirección de regreso en las ubicaciones de memoria RAM interna 08H (byte superior del contador de programa) y 09H (byte inferior del contador de programa).

EJEMPLO

6.2

Dos ondas cuadradas mediante el uso de interrupciones

Escriba un programa que utilice interrupciones para crear ondas cuadradas de 7 kHz y 500 Hz al mismo tiempo en P1.7 y P1.6.

Solución

La figura 6-4 muestra la configuración del hardware con los tiempos de las formas de onda deseadas.

Esta combinación de salidas sería en extremo difícil de generar en un sistema no controlado mediante interrupciones. El temporizador 0 provee la sincronización para la señal de 7 kHz y opera en el modo 2, como en el ejemplo anterior; y el temporizador 1 provee la sincronización para la señal de 500 Hz y opera en el modo 1, el modo de temporizador de 16 bits. No podemos utilizar el modo 2, debido a que una señal de 500 Hz requiere de un tiempo a nivel alto de 1 ms y

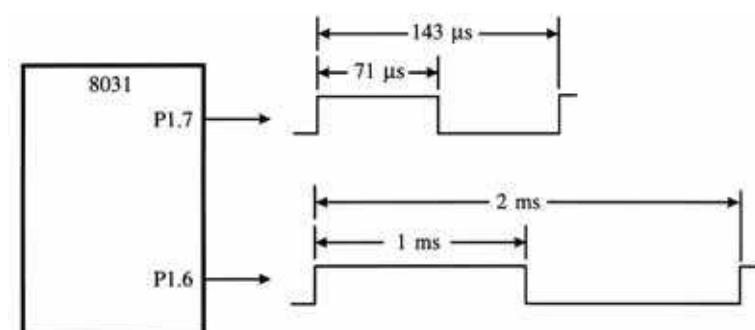


FIGURA 6-4

Ejemplo de formas de onda

un tiempo a nivel bajo de 1 ms. (Recuerde que el intervalo medido máximo en el modo 2 es de 256 ms cuando el 8051 opera a 12 MHz). El programa se describe a continuación:

```

0000      5          ORG    0
0000 020030 6          LJMP   PRINCIPAL
000B      7          ORG    000BH      ;Dirección de vector del
                                         temporizador 0
000B 02003F 8          LJMP   TOISR
001B      9          ORG    001BH      ;Dirección de vector del
                                         temporizador 1
001B 020042 10         LJMP   T1ISR
0030      11         ORG    0030H
0030 758912 12 PRINCIPAL: MOV    TMOD, #12H      ;Temporizador 1 = modo 1
                                         ;Temporizador 0 = modo 2
                                         ;7 kHz utilizando el temporizador 0
0033 758CB9 14         MOV    TH0, #-71
0036 D28C   15         SETB   TR0
0038 D28F   16         SETB   TF1      ;fuerza la interrupción del
                                         temporizador 1
003A 75A88A 17         MOV    IE, #8AH      ;habilita ambas interrupciones de
                                         temporizador
003D 80FE   18         SJMP   $
                                         ;
003F B297   20 TOISR:  CPL    P1.7
0041 32     21         RETI
0042 C28E   22 T1ISR:  CLR    TR1
0044 758DFC 23         MOV    TH1, #HIGH(-1000) ;1 ms de tiempo en nivel alto y
0047 758B18 24         MOV    TL1, #LOW(-1000) ;tiempo en nivel bajo
004A D28E   25         SETB   TR1
004C B296   26         CPL    P1.6
004E 32     27         RETI
                                         28         END

```

Análisis

Esta estructura también es para un programa completo que puede instalarse en una memoria EPROM o ROM, en un producto basado en el 8051. El programa principal y las ISR se localizan encima de las ubicaciones de los vectores para la reinicialización del sistema y las interrupciones. Ambas formas de onda son creadas mediante las instrucciones “CPL bit”; sin embargo, los intervalos medidos necesitan de una solución diferente para cada una.

Debido a que los registros TL1/TH1 deben recargarse luego de cada desbordamiento (es decir, después de cada interrupción), la ISR del temporizador 1 (a) detiene el temporizador, (b) recarga TL1/TH1, (c) inicia el temporizador, y entonces (d) complementa al bit de puerto. Observe también que TL1/TH1 *no* se inicializan al comienzo del programa principal, a diferencia de TH0. Debido a que TL1/TH1 deben reinicializarse después de cada desbordamiento, TF1 se activa en el programa principal mediante el software para “forzar” una interrupción inicial tan pronto como las interrupciones se hayan habilitado. Esto inicia en efecto la forma de onda de 500 Hz.

La ISR del temporizador 0, como en el ejemplo anterior, sólo complementa el bit de puerto y regresa al programa principal. La instrucción SJMP \$ se utiliza en el programa principal en la forma abreviada AQUI: SJMP AQUI. Las dos formas son funcionalmente equivalentes. (Consulte la sección “Símbolos especiales del ensamblador” en el capítulo 7).

6.6 INTERRUPCIONES DEL PUERTO SERIAL

Las interrupciones del puerto serial ocurren cuando se activa ya sea la bandera de interrupción de transmisión (TI) o la bandera de interrupción de recepción (RI). Una interrupción de transmisión ocurre cuando la transmisión del carácter escrito previamente al SBUF ha terminado. Una interrupción de recepción ocurre cuando un carácter se ha recibido por completo y está esperando en el SBUF para ser leído.

Las interrupciones del puerto serial son un poco distintas de las del temporizador. La bandera que genera una interrupción del puerto serial no se borra mediante el hardware cuando la CPU se vectoriza hacia la interrupción. La razón para esto es que existen dos fuentes para una interrupción del puerto serial, TI o RI. Debemos determinar la interrupción en la ISR y borrar mediante el software la bandera que generó la interrupción. Recuerde que en las interrupciones del temporizador el hardware borra la bandera que generó la interrupción cuando el procesador se vectoriza hacia la ISR.

EJEMPLO

6.3

Operación de salida de caracteres mediante el uso de interrupciones

Escriba un programa que utilice interrupciones para transmitir en forma continua un conjunto de códigos ASCII (excluyendo los códigos de control) hacia una terminal conectada al puerto serial del 8051.

Solución

Existen 128 códigos de 7 bits en la tabla de ASCII. (Consulte el apéndice F). De estos 128 códigos, 95 son de gráficos (20H a 7EH) y 33 de control (00H a 1FH, y 7FH). El programa que mostramos a continuación es independiente y se puede ejecutar desde una memoria EPROM o ROM justo después de una reinicialización del sistema.

```

0000      5          ORG    0
0000 020030  6          LJMP   PRINCIPAL
0023      7          ORG    0023H ;entrada a la interrupción del puerto
                                ;serial
0023 020042  8          LJMP   REIPS
0030      9          ORG    0030H
0030 758920 10 PRINCIPAL: MOV    TMOD,#20H ;Temporizador 1, modo 2
0033 758DE6 11          MOV    TH1,#-26 ;valor de recarga para 1200 baudios
0036 D28E 12          SETB   TR1 ;inicia el temporizador
0038 759842 13          MOV    SCON,#42H ;modo 1, activa TI para forzar primera
                                ;interrupción; envía primer carácter.
003B 7420 14          MOV    A,#20H ;envía ASCII al primer espacio
003D 75A890 15          MOV    IE,90H ;habilita interrupción del puerto serial
0040 80FE 16          SJMP   $      ;haz nada
0040 80FE 17          SJMP   $      ;
0040 80FE 18          SJMP   $      ;
0042 B47F02 19 REIPS: CJNE   A,#7FH,SKIP ;si termina conjunto en ASCII,
0045 7420 20          MOV    A,#20H ; reinicializa a un ESPACIO
0047 F599 21 IGNORA:  MOV    SBUF,A ;envía carácter al puerto serial
0049 04 22          INC    A      ;incrementa código en ASCII
004A C299 23          CLR    TI     ;borra bandera de interrupción
004C 32 24          RETI   END
004C 32 25          END

```

Análisis

Después de saltar a la etiqueta PRINCIPAL en la dirección de código 0030H, las primeras tres instrucciones inicializan el temporizador 1 para que genere una velocidad en baudios de 1200 baudios hacia el puerto serial (líneas 10-12). La instrucción MOV SCON,#42H inicializa el puerto serial al modo 1 (UART de 8 bits) y activa la bandera TI para forzar una interrupción tan

pronto como las interrupciones hayan sido habilitadas. Después, el primer código de gráficos en ASCII (20H) se carga al registro A y se habilitan las interrupciones del puerto serial. Finalmente, el cuerpo principal del programa entra a un ciclo donde hace nada (SJMP \$).

La rutina de servicio a la interrupción del puerto serial realiza todo el trabajo una vez que el programa principal ha establecido las condiciones iniciales. Las primeras dos instrucciones verifican el acumulador, y si el código ASCII ha llegado a 7FH (es decir, el último código transmitido fue 7EH), las instrucciones reinicializan el acumulador al valor 20H (líneas 19-20). Después, el código ASCII se envía al búfer del puerto serial (MOV SBUF,A), el código se incrementa (INC A), la bandera de interrupción de transmisión se borra (CLR TI), y la ISR termina (RETI). El control regresa al programa principal, y la instrucción SJMP \$ se ejecuta hasta que TI se activa al final de la siguiente transmisión de un carácter.

Si comparamos la velocidad de la CPU con la velocidad de transmisión de caracteres, veremos que la instrucción SJMP \$ se ejecuta durante un gran porcentaje del tiempo destinado para este programa. ¿Cuál es este porcentaje? A 1200 baudios, cada bit transmitido se tarda $1/1200 = 0.833$ ms. Ocho bits de datos más un bit de inicio-y-detención, por lo tanto, tardarían 8.33 ms o 8333 μ s. Encontramos el peor caso de tiempo de ejecución para la rutina REIPS mediante la suma del número de ciclos necesarios para cumplir cada instrucción y multiplicándola por 1 μ s (si asumimos una operación a 12 MHz). Esto resulta en un valor de 8 μ s. En consecuencia, de los 8333 μ s necesarios para cada transmisión de carácter, sólo 8 μ s pueden ser utilizados por la rutina de servicio de interrupción. La instrucción SJMP \$ se ejecuta durante un $8325 \div 8333 \times 100 = 99.90\%$ del tiempo. Debido a que estamos utilizando interrupciones, podemos reemplazar la instrucción SJMP \$ con otras instrucciones que realizan otras tareas requeridas por la aplicación. Sólo ocurrirán interrupciones cada 8.33 ms, y los caracteres seguirán transmitiéndose a través del puerto serial como lo hicieron en el programa anterior.

6.7 INTERRUPCIONES EXTERNAS

Las interrupciones externas ocurren como resultado de un borde de bajo nivel o negativo en las terminales INT0 o INT1 del 8051. Éstas son las funciones alternas de los bits P3.2 (terminal 12) y P3.3 (terminal 13) del puerto 3, respectivamente.

Las banderas que generan estas interrupciones son los bits IE0 e IE1 en el registro TCON. Cuando se genera una interrupción externa, la bandera que la generó se borra mediante el hardware cuando se vectoriza a la ISR sólo si la interrupción fue activada mediante una transición. Si la interrupción fue activada debido a un bajo nivel, la fuente externa de petición, y no el hardware, controla el nivel de la bandera de petición.

La opción de utilizar interrupciones activadas por un bajo nivel en lugar de interrupciones activadas por un borde negativo es programable mediante los bits IT0 e IT1 en el TCON. Por ejemplo, si IT1 = 0, la interrupción externa I se dispara por un bajo nivel detectado en la terminal INT1. Si IT1 = 1, la interrupción externa 1 se dispara por un cambio de borde. En este modo, si muestras sucesivas de la terminal INT1 presentan nivel alto en un ciclo y nivel bajo en el ciclo siguiente, la bandera de petición de interrupción IE1 en TCON se activa. Después de esto, el bit de bandera IE1 pide la interrupción.

Debido a que se obtienen muestras de las terminales de interrupción externa en cada ciclo de máquina, una entrada debe mantenerse durante, por lo menos, 12 períodos del oscilador para asegurar un muestreo apropiado. Si la interrupción externa se activa mediante una transición, la fuente externa debe mantener la terminal de petición a nivel alto durante, por lo menos, un ciclo y entonces mantenerla a nivel bajo cuando menos un ciclo más para asegurar la detección de una transición. Los bits IE0 e IE1 se borran automáticamente cuando la CPU se vectoriza hacia la interrupción.

Si la interrupción externa se activa mediante un cambio de nivel, la fuente externa debe mantener activa la petición hasta que se genere la interrupción requerida. Después de esto, la fuente externa debe desactivar la petición antes de que se termine la rutina de servicio de interrupción o se genere otra interrupción. A menudo, la acción tomada en la ISR causa que la fuente de petición regrese la señal de interrupción a un estado inactivo

EJEMPLO Controlador de un horno

6.4

Con base en el 8051, diseñe un controlador que utilice interrupciones de tal forma que mantenga un edificio a $20^{\circ}\text{C} \pm 1^{\circ}\text{C}$.

Solución

Vamos a suponer la siguiente interfaz para este ejemplo. El solenoide de encendido y apagado se conecta a P1.7 de tal manera que

P1.7 = 1 cuando el solenoide está activado (hornos encendidos)
 P1.7 = 0 cuando el solenoide está desactivado (hornos apagados)

Dos sensores de temperatura están conectados a INT0 y INT1 y proporcionan señales de CALIENTE y FRÍO respectivamente, de tal manera que

CALIENTE = 0 if $T > 21^{\circ}\text{C}$

FRÍO = 0 if $T < 19^{\circ}\text{C}$

El programa deberá encender el horno cuando $T < 19^{\circ}\text{C}$ y apagarlo cuando $T > 21^{\circ}\text{C}$. La figura 6-5 muestra la configuración del hardware y el diagrama de sincronización.

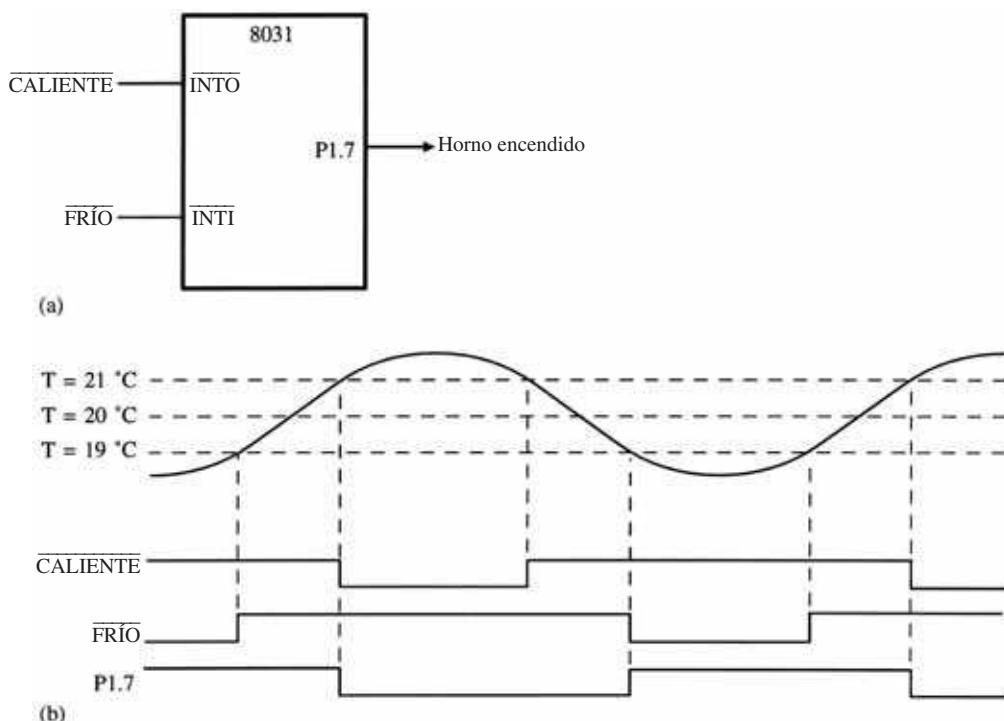
```

0000      5          ORG    0
0000  020030  6          LJMP   PRINCIPAL
                    7          ;vector de EXT 0 en 0003H
0003  C297    8  EX0ISR: CLR    P1.7      apaga el horno
0005  32      9          RETI
0013      10         ORG    0013H
0013  D297    11 EX1ISR: SETB   P1.7      ;enciende el horno
0015  32      12         RETI
0030      13         ORG    30H
0030  75A885  14 PRINCIPAL: MOV     IE,#85H    ;habilita interrupciones externas
0033  D288    15          SETB   IT0       disparado mediante un borde negativo
0035  D28A    16          SETB   IT1
0037  D297    17          SETB   P1.7      ;enciende el horno
0039  20B202  18          JB     P3.2,IGNORA ;si T > 21 grados,
003C  C297    19          CLR    P1.7      ; apaga el horno
003E  80FE    20 SKIP:    SJMP   $          ;haz nada
                    21          END

```

Análisis

En el programa principal, las primeras tres instrucciones (líneas 14-16) habilitan las interrupciones externas y hacen que INT0 y INT1 se disparen por flanco negativo. Debido a que no conocemos el estado actual de las entradas de CALIENTE (P3.3) y FRÍO (P3.3) se requiere que las siguientes tres instrucciones (líneas 17-19) enciendan o apaguen el horno, según sea necesario. Primero, el horno se enciende (SETB P1.7), y entonces se obtiene una muestra de la

**FIGURA 6–5**

Ejemplo del horno. (a) Conexiones del hardware. (b) Sincronización

entrada de CALIENTE (JB P3.2,IGNORA). Si CALIENTE está a nivel alto, entonces $T < 21^{\circ}\text{C}$, y por lo tanto ignoramos la siguiente instrucción y dejamos al horno encendido. Si, por el contrario, CALIENTE está a nivel bajo, entonces $T > 21^{\circ}\text{C}$. En este caso no realizamos el salto. La siguiente instrucción apaga al horno (CLR P1.7) antes de entrar al ciclo haz nada.

Una vez que todo se configura de manera correcta en el programa principal, no hay mucho más que hacer. Cada vez que la temperatura suba por encima de los 21°C o caiga por debajo de los 19°C , se genera una interrupción. Las ISR simplemente se encargan de encender (SETB P1.7) o apagar (CLR P1.7) al horno, conforme sea necesario, y regresan al programa principal.

Observe que no necesitamos de la instrucción ORG 0003H justo antes de la etiqueta EX0ISR. Debido a que la instrucción LJMP PRINCIPAL es de tres bytes, es casi seguro que la etiqueta EX0ISR empezará en la dirección 0003H, el punto de acceso correcto para las interrupciones externas 0.

EJEMPLO Sistema de detección de intrusos

6.5

Diseñe un sistema de detección de intrusos que utilice interrupciones de tal forma que se genere un tono de 400 Hz durante un segundo (utilizando un altavoz conectado a P1.7) siempre que un sensor de puerta conectado a INT0 realice una transición de nivel alto a nivel bajo.

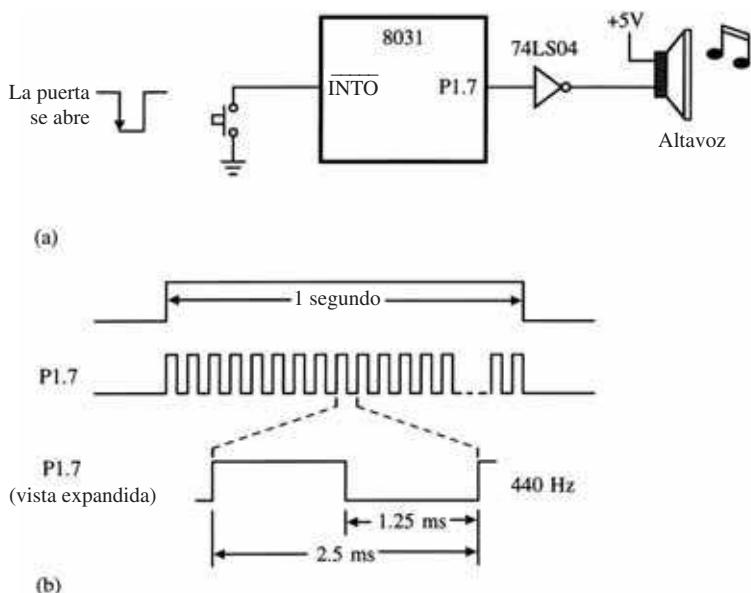


FIGURA 6–6

Interfaz para un altavoz mediante el uso de interrupciones: (a) Conexiones del hardware. (b) Sincronización

Solución

La solución para este ejemplo utiliza tres interrupciones: externa 0 (sensor de puerta), temporizador 0 (1 segundo de tiempo límite), y el temporizador 1 (tono de 400 Hz). La figura 6-6 muestra la configuración del hardware y los tiempos de sincronización.

```

0045 C28C    25  TOISR:    CLR   TR0      ;detiene el temporizador
0047 DF07    26          DJNZ  R7, IGNORA ;si no es la 20a vez, termina
0049 C2A9    27          CLR   ET0      ;si es la 20a, deshabilita el tono
004B C2AB    28          CLR   ET1      ;se autodeshabilita
004D 020058  29          LJMP  TERMINA
0050 758C3C  30  IGNORA:  MOV   TH0,#HIGH(-50000) ;retraso de 0.05 seg
0053 758AB0  31          MOV   TL0,#LOW(-5000)
0056 D28C    32          SETB  TR0
0058 32      33  TERMINA: RETI
                    34  ;
0059 C28E    35  T1ISR:    CLR   TR1
005B 758DFB  36          MOV   TH1,#HIGH(-1250) ;conteo para 400 Hz
005E 758B1E  37          MOV   TL1,#LOW(-1250)
0061 B297    38          CPL   P1.7      ;¡música maestro!
0063 D28E    39          SETB  TR1
0065 32      40          RETI
                    41          END

```

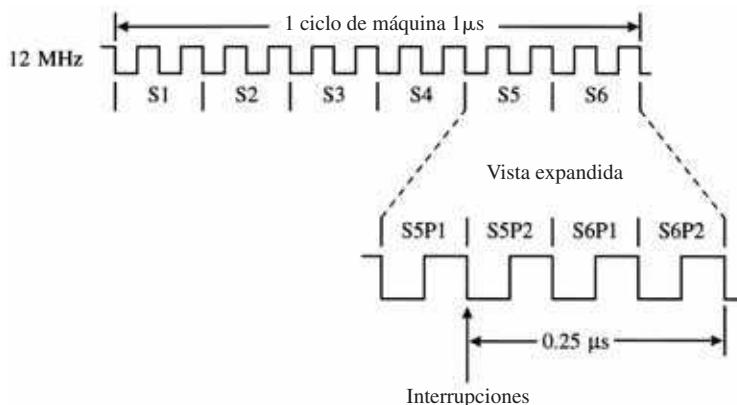
Análisis

Este es nuestro más largo programa presentado hasta ahora. Las cinco distintas secciones son las ubicaciones de los vectores de interrupción, el programa principal, y las tres rutinas de servicio de interrupciones. Todas las ubicaciones de los vectores contienen instrucciones LJMP para sus rutinas respectivas. El programa principal, iniciando en la dirección para código 0030H, contiene sólo cuatro instrucciones. La instrucción SETB IT0 configura la entrada de la interrupción de sensor de la puerta como una interrupción disparada por un borde negativo. La instrucción MOV TMOD,#11H configura ambos temporizadores en el modo 1, el modo de temporizador de 16 bits. Solamente la interrupción externa 0 se habilita al principio (MOV IE,#81H), así que se requiere una condición de “puerta abierta” antes de aceptar cualquier interrupción. Finalmente, la instrucción SJMP \$ coloca al programa en el ciclo de haz nada.

Cuando se detecta una condición de puerta abierta (mediante una transición de nivel alto a nivel bajo de $\overline{\text{INT0}}$), se genera una interrupción externa 0 y la rutina EX0ISR empieza a transferir la constante con valor 20 a R7 (vea más adelante) y activa las banderas de desbordamiento para ambos temporizadores, de manera que se force la generación de interrupciones de los temporizadores.

Sin embargo, las interrupciones de los temporizadores sólo ocurrirán si sus bits correspondientes están habilitados en el registro IE. Las siguientes dos instrucciones (SETB ET0 y SETB ET1) habilitan las interrupciones de los temporizadores. Por último, la rutina EX0ISR termina con una instrucción RETI hacia el programa principal.

El temporizador 0 crea un límite de tiempo de 1 segundo, y el temporizador 1 genera el tono de 400 Hz. Después que la rutina EX0ISR regresa al programa principal, las interrupciones de los temporizadores se generan de inmediato (y se aceptan después de una ejecución de SJMP \$). Debido a la secuencia de sondeo fija (consulte la figura 6-2), primero se le da servicio a la interrupción del temporizador 0. Un límite de tiempo de 1 segundo se genera mediante la programación de 20 repeticiones de un tiempo límite de 50,000 μs . R7 sirve como el contador. Diecinueve de las 20 veces, la rutina TOISR opera como sigue. Primero, se detiene el temporizador 0 y R7 disminuye. Después, TH0/TL0 se recargan con el valor de -50,000, se inicia el temporizador de nuevo y se termina la interrupción. En la 20^a interrupción del temporizador 0, R7 se disminuye a 0 (ha transcurrido 1 segundo). Ambas interrupciones de los temporizadores se deshabilitan (CLR, ET0, CLR ET1) y termina la interrupción. No se generan más interrupciones de los temporizadores sino hasta que se detecta la siguiente condición de “puerta abierta”.

**FIGURA 6-7**

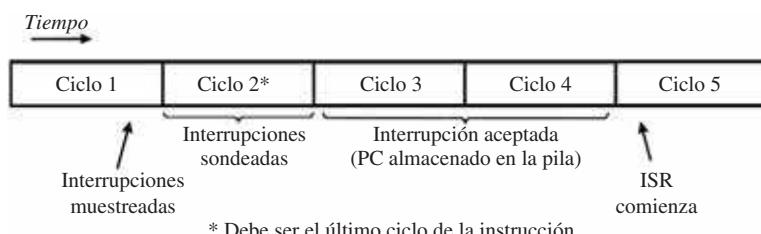
Muestreo de las interrupciones en S5P2

El tono de 400 Hz se programa mediante las interrupciones del temporizador 1, una frecuencia de 400 Hz requiere de un periodo de $1/400 = 2,500 \mu\text{s}$ o $1,250 \mu\text{s}$ de tiempo en nivel alto y $1,250 \mu\text{s}$ de tiempo en nivel bajo. Cada ISR del temporizador 1 simplemente transfiere un valor de -1250 a los registros TH1/TL1, complementa el bit de puerto que controla al altavoz, y entonces termina.

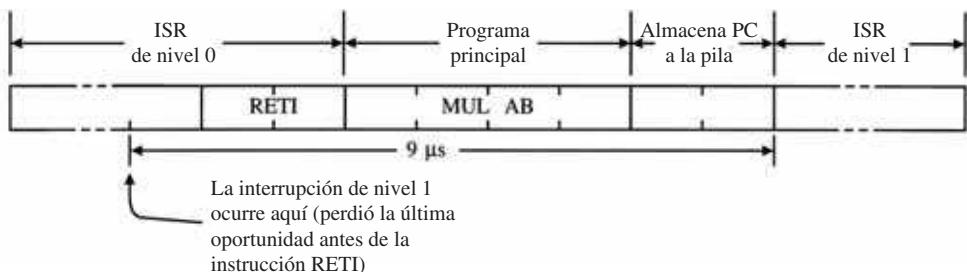
6.8 SINCRONIZACIÓN DE INTERRUPCIONES

Las interrupciones se muestrean y fijan en la etapa S5P2 de cada ciclo de máquina. (Consulte la figura 6-7). Estas interrupciones se sondan en el siguiente ciclo de máquina y, cuando existe una condición de interrupción, se aceptan si (a) no existe otra interrupción de una prioridad igual o más alta en proceso, (b) el ciclo de sondeo es el último ciclo en una instrucción, y (c) la instrucción en proceso no es una RETI ni algún acceso a IE o IP. El procesador almacena el contador de programa en la pila durante los siguientes dos ciclos, y carga el PC con la dirección del vector de interrupción. La ISR comienza.

La condición de que la instrucción en proceso no sea una instrucción RETI asegura que, por lo menos, una instrucción se ejecutará después de cada rutina de servicio de interrupción. La figura 6-8 muestra la sincronización.

**FIGURA 6-8**

Sondeo de interrupciones

**FIGURA 6–9**

Latencia de interrupción

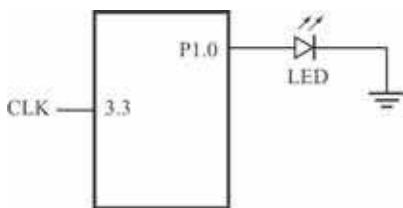
El tiempo entre la ocurrencia de una condición de interrupción y el comienzo de la ISR se conoce como **latencia de interrupción**. Este tiempo es crítico en muchas de las aplicaciones de control. Con un cristal de 12 MHz, la latencia de interrupción puede ser tan corta como $3.25 \mu s$ en el 8051. Un sistema basado en el 8051 que utilice una interrupción de alta prioridad siempre tendrá una latencia de interrupción, en el peor caso, de $9.25 \mu s$ (si asumimos que la interrupción de alta prioridad siempre está habilitada). Esto ocurre si existe una condición justo antes de la instrucción RETI de una ISR de nivel 0 que está seguida por una instrucción de multiplicación (consulte la figura 6-9).

RESUMEN

En este capítulo presentamos los detalles principales requeridos para embarcarnos en el diseño de sistemas controlados mediante interrupciones con el microcontrolador 8051. Sugerimos al lector que comience la programación con interrupciones en pasos incrementales. Los ejemplos incluidos en este capítulo sirven como un buen primer contacto con las interrupciones del 8051.

A menudo, las computadoras de una sola tarjeta basadas en el 8051 contienen un programa de monitoreo en la memoria EPROM que reside en la parte inferior de la memoria para código. Si el programa de monitoreo no utiliza interrupciones, es probable que las ubicaciones de los vectores contengan instrucciones de LJMP hacia un área en la memoria RAM para código donde se cargan las aplicaciones del usuario para su ejecución y depuración. La literatura del fabricante proporcionará las direcciones que los programadores pueden utilizar como puntos de acceso a las rutinas de servicio de las interrupciones. Por otro lado, los usuarios también pueden “examinar” las ubicaciones de los vectores de interrupción utilizando los comandos del programa de monitoreo para examinar ubicaciones de memoria para código. Por ejemplo, el contenido de la dirección de memoria para código 0003H contendrá el código de operación de la primera instrucción que se ejecute para una interrupción externa 0. Si este código de operación es el de la instrucción LJMP (22H; consulte el apéndice B), entonces las siguientes dos direcciones (0004H y 0005H) contendrán la dirección de la ISR, y así sucesivamente.

De otra manera, los usuarios también pueden desarrollar aplicaciones completamente independientes que utilicen interrupciones, como ilustramos en los ejemplos. Los bytes del objeto pueden quemarse en la memoria EPROM e instalarse en el sistema destino en la dirección para código 0000H. Cuando el sistema se inicia o reinicializa, la aplicación comienza a ejecutarse sin necesidad de un programa de monitoreo para cargar y ejecutar la aplicación.

**FIGURA 6-10**

Interfaz para un LED mediante el uso de interrupciones

PROBLEMAS

- 6.1** Modifique el ejemplo 6.1 para deshabilitar las interrupciones y terminar si cualquier tecla es presionada en la terminal.
- 6.2** Genere una onda cuadrada de 1 kHz en P1.7, mediante el uso de interrupciones.
- 6.3** Genere una onda de pulso de 7 kHz con ciclo de trabajo de 30% en P1.6, utilice interrupciones.
- 6.4** Combine los ejemplos 6.1 y 6.3 (incluidos en este capítulo) en un solo programa.
- 6.5** Modifique el ejemplo 6.3 para enviar un carácter por segundo. (Sugerencia: utilice un temporizador y envíe el carácter a través de la rutina ISR del temporizador).
- 6.6** Modifique el ejemplo 6.5 para que incluya un modo de “reinicialización”. Si ocurre una transición de nivel alto a nivel bajo mientras el altavoz está activado, reinicialice el ciclo de sincronización para que el altavoz continúe activado durante un segundo más. La figura 6-10 ilustra esto.
- 6.7** Suponga que una interrupción externa está conectada a INT0 y ocurre una interrupción del temporizador 0 al mismo tiempo. ¿A cuál interrupción se le deberá dar servicio primero? ¿Por qué?
- 6.8** De manera predeterminada, cuando una interrupción del puerto serial y una interrupción externa conectada a INT1 ocurren al mismo tiempo, el 8051 le dará servicio a la interrupción del puerto serial sólo después que le haya dado servicio a la interrupción externa. ¿Cómo configuraría usted al 8051 para que primero dé servicio a la interrupción del puerto serial cuando ambas interrupciones ocurren al mismo tiempo?
- 6.9** Los registros IE e IP están inicializados con los siguientes valores:
IE = 10001111
IP = 00001110
Suponga que una interrupción del temporizador 0, una interrupción del puerto serial, y una interrupción externa INT1 ocurren al mismo tiempo. ¿A cuál deberá darse servicio primero? ¿Por qué?
- 6.10** ¿Cuál es la diferencia entre las ISR pequeñas y grandes?
- 6.11** Consulte la figura 6-10, suponga que la terminal 3.3 está conectada a una señal de reloj, CLK, con frecuencia de 1 kHz. Escriba un programa en lenguaje ensamblador con las interrupciones habilitadas para que envíe un nivel ALTO a P1.0 (encienda el LED) durante cerca de $250 \mu\text{s}$ cada vez que se detecte una transición con tendencia negativa (NGT) en la señal de reloj en la terminal 3.3.

Programación en lenguaje ensamblador

7.1 INTRODUCCIÓN

En este capítulo presentaremos la programación en lenguaje ensamblador para el microcontrolador 8051. El lenguaje ensamblador es un lenguaje computacional ubicado entre el lenguaje máquina y un lenguaje de alto nivel. Los lenguajes de alto nivel típicos, como Pascal o C, utilizan palabras e instrucciones que son fáciles de entender para los humanos, aunque todavía están muy lejos de parecerse a un lenguaje “natural”. El lenguaje máquina es el lenguaje binario de las computadoras. Un programa en lenguaje máquina es una serie de bytes binarios que representan instrucciones que la computadora puede ejecutar.

El lenguaje ensamblador sustituye a los códigos binarios del lenguaje máquina con “mnemónicos” fáciles de recordar que facilitan la programación. Por ejemplo, una instrucción de suma que en lenguaje máquina puede estar representada por el código “10110011”, también es posible representarla en lenguaje ensamblador mediante el mnemónico “ADD”. Es evidente entonces que la programación con mnemónicos tiene mayor preferencia que la programación con códigos binarios.

Por supuesto, ésta no es la historia completa. Las instrucciones operan con datos, y la ubicación de los datos está especificada por varios “modos de direccionamiento” incrustados en el código binario de la instrucción codificada en lenguaje máquina. Así que pueden existir diversas variaciones de la instrucción ADD, dependiendo de qué se esté sumando. Las reglas para especificar estas variaciones son fundamentales para desarrollar el tema de la programación en lenguaje ensamblador.

Una computadora no puede ejecutar un programa codificado en lenguaje ensamblador. Una vez escrito el programa, debe pasar por una traducción o conversión a lenguaje máquina. En el ejemplo anterior, el mnemónico “ADD” se debe traducir al código binario “10110011”. Esta traducción o conversión puede involucrar uno o más pasos antes de que se produzca un programa en lenguaje máquina ejecutable, dependiendo de la complejidad del ambiente de programación. Como mínimo, requerimos un programa llamado “ensamblador” para traducir los mnemónicos de las instrucciones a sus códigos binarios del lenguaje máquina. Un paso adicional puede requerir de un “enlazador” para combinar porciones de programas desarrollados en archivos por separado y establecer la dirección en memoria donde el programa puede ejecutarse. Empecemos con unas cuantas definiciones.

Un **programa en lenguaje ensamblador** es un programa que se escribe utilizando etiquetas, mnemónicos, etc., y en donde cada instrucción corresponde a una instrucción máquina. Los programas en lenguaje ensamblador, a menudo llamados código fuente o código simbólico, no pueden ser ejecutados por una computadora.

Un **programa en lenguaje máquina** es un programa que contiene códigos binarios que representan instrucciones para una computadora. Los programas en lenguaje máquina, a menudo llamados código objeto, sí pueden ser ejecutados por una computadora.

Un **ensamblador** es un programa que traduce o convierte un programa escrito en lenguaje ensamblador a un programa codificado en lenguaje máquina. El programa en lenguaje máquina (código objeto) puede existir en formas “absoluta” o “relocalizable”. En el segundo caso, se requiere de un “enlazamiento” para establecer la dirección absoluta para su ejecución.

Un **enlazador** es un programa que combina programas objeto relocalizables (módulos) para producir un programa objeto absoluto que pueda ser ejecutado por una computadora. Algunas veces, al enlazador se le denomina “enlazador/localizador” para reflejar sus funciones separadas de combinar módulos relocalizables (enlazamiento) y establecer la dirección para ejecutarlos (localización).

Un **segmento** es una unidad de memoria para código o para datos. Un segmento puede ser relocalizable o absoluto. Un segmento relocalizable tiene nombre, tipo y otros atributos que le permiten al enlazador poder combinarlo con otros segmentos parciales, si es necesario, y localizarlo correctamente. Un segmento absoluto no tiene nombre ni puede combinarse con otros segmentos.

Un **módulo** contiene uno o más segmentos o segmentos parciales. Un módulo tiene un nombre asignado por el usuario. Las definiciones del módulo determinan el alcance de los símbolos locales. Un archivo objeto contiene uno o más módulos. En muchas circunstancias, podemos considerar a un módulo como un “archivo”.

Un **programa** consta de un solo módulo absoluto, combinando todos los segmentos absolutos y relocalizables provenientes de todos los módulos de entrada. Un programa contiene únicamente los códigos binarios para instrucciones (con direcciones y constantes de datos) que son entendidos por una computadora.

7.2 OPERACIÓN DE UN ENSAMBLADOR

Existen muchos programas ensambladores y otros de soporte disponibles para facilitar el desarrollo de aplicaciones para el microcontrolador 8051. El ensamblador original de Intel para la familia MCS-51™, el ASM51™, ya no está disponible comercialmente. Sin embargo, estableció el estándar bajo el que los otros ensambladores son comparados. En este capítulo, nos enfocaremos sobre la programación en lenguaje ensamblador, tomando en cuenta las características más comunes del ASM51. Aunque muchas características son estandarizadas, algunas pueden no estar implementadas en los ensambladores de otras compañías.

El ASM51 es un poderoso ensamblador que tiene todos los adornos posibles. Está disponible en los sistemas de desarrollo de Intel y en la familia de microcomputadoras de IBM PC. Debido a que estas computadoras “anfitrionas” contienen una CPU distinta al 8051, el ASM51 se conoce como **ensamblador cruzado**. Un programa fuente para el 8051 puede escribirse en la computadora anfitriona (utilizando cualquier editor de texto) y ser ensamblado a un archivo objeto y a un archivo de listado (utilizando el ASM51), pero podría no ser ejecutado. La CPU del sistema anfitrión no es un 8051 y, por lo tanto, no entiende las instrucciones binarias incluidas en el archivo objeto. En una computadora anfitriona, la ejecución requiere ya sea de emulación de hardware o de simulación en software de la CPU destino. En el capítulo 10 analizamos emulación de hardware, simulación en software, descarga, y otras técnicas de desarrollo.

El ASM51 se invoca desde el indicador de comandos mediante

```
ASM51 archivo_fuente [controles_del_ensamblador]
```

El archivo fuente es ensamblado y cualquier control de ensamblador especificado surte efecto. (Los controles del ensamblador, que son opcionales, los examinamos más adelante en este capítulo). El ensamblador recibe como entrada un archivo fuente (por ejemplo, PROGRAMA.SRC) y genera un archivo objeto (PROGRAMA.OBJ) y un archivo de listado (PROGRAMA.LST) como salida. Esto se ilustra en la figura 7-1.

Debido a que la mayoría de los ensambladores examinan al programa fuente dos veces al realizar la traducción o conversión a lenguaje máquina, se describen como **ensambladores de dos pasos**. El ensamblador utiliza un **contador de ubicación** como la dirección de las instrucciones y los valores para las etiquetas. A continuación describimos la acción ejecutada en cada paso.

7.2.1 Paso uno

Durante el primer paso, el archivo fuente es examinado línea por línea y se construye una **tabla de símbolos**. El contador de ubicación tiene el valor 0 predeterminado o se establece mediante la directiva ORG (establece origen). El contador de ubicación se incrementa en un valor igual al tamaño de cada instrucción conforme se examina el archivo. Directivas de datos definidas (DB o DW) incrementan al contador de ubicación en un valor igual al número de bytes definidos. Las directivas de memoria reservada (DS) incrementan al contador de ubicación en un valor igual al número de bytes reservados.

Cada vez que se encuentra una etiqueta al comienzo de una línea, se le coloca en la tabla de símbolos además del valor actual del contador de ubicación. Los símbolos que son definidos utilizando directivas de igualdad (EQU) se colocan en la tabla de símbolos además del valor “igualado”. La tabla de símbolos se almacena y luego es utilizada durante el paso dos.

7.2.2 Paso dos

Durante el paso dos, los archivos objeto y de listado son creados. Los mnemónicos se convierten a códigos de operación y se colocan en los archivos de salida. Los operandos se evalúan y colocan después de los códigos de operación de instrucciones. Cuando aparecen símbolos en el campo de operandos, sus valores se obtienen de la tabla de símbolos (creada durante el paso uno) y se utilizan en el cálculo de las direcciones o los datos correctos para las instrucciones.

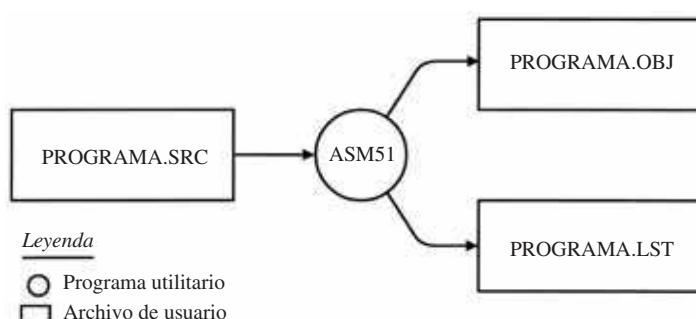


FIGURA 7-1
Ensamblado de un programa fuente

```

ASM(archivo_entrada) /* ensambla el programa fuente en archivo_entrada */
BEGIN
    ***** PASADA 1: GENERAR LA TABLA DE SÍMBOLOS *****/
    lc = 0;
    nemonico = null;
    abrir_archivo_entrada;
    WHILE (nemonico != fin) DO BEGIN
        obtener_linea();
        explorar_linea(); /* obtiene etiqueta/símbolo y nemónico */
        IF (etiqueta) THEN
            introducir_en_tabla_simbolos(etiqueta, lc);
        CASE nemonico OF BEGIN
            null, comentario, fin: ; /* no hace nada */
            ORG: lc = valor_operando;
            EQU: introducir_en_tabla_simbolos(símbolo, valor_operando);
            DB: WHILE (obtuvo_operando) DO incrementar_lc;
            DS: lc = lc + valor_operando;
            instrucion_byte_1: lc = lc + 1;
            instrucion_byte_2: lc = lc + 2;
            instrucion_byte_3: lc = lc + 3;
        END
    END
    ***** PASADA 2: CREAR EL PROGRAMA OBJETO *****/
    rebobinar_apuntador_archivo_entrada;
    lc = 0;
    nemonico = null;
    abrir_archivo_salida;
    WHILE (nemonico != fin) DO BEGIN
        obtener_linea();
        explorar_linea(); /* determina código op. nemónico y valor(es) operando(s) */
        /* Nota: Los símbolos utilizados en el campo de operando se buscan en la tabla */
        /* de símbolos que se crea durante la primera pasada */
        CASE nemonico OF BEGIN
            null, comentario, EQU, END: ; /* no hace nada */
            ORG: lc = valor_operando;
            DB: WHILE (operando) BEGIN
                colocar_en_archivo_objeto(operando);
                lc = lc + 1;
            END
            DS: lc = lc + operando;
            instrucion_byte_1: colocar_en_archivo_objeto(cod_inst);
            instrucion_byte_2: colocar_en_archivo_objeto(cod_inst);
            colocar_en_archivo_objeto(operando);
            instrucion_byte_1: colocar_en_archivo_objeto(cod_inst);
            colocar_en_archivo_objeto(byte superior operando);
            colocar_en_archivo_objeto(byte inferior operando);
            lc = lc + tamano_de_instrucion;
        END
    END
    cerrar_archivo_entrada;
    cerrar_archivo_salida;
END

```

FIGURA 7-2

Descripción en seudocódigo de un operador de dos pasos

Debido a que se realizan dos pasos, el programa fuente puede utilizar “referencias hacia delante”, es decir, aplicar un símbolo antes de que éste se defina. Esto puede ocurrir, por ejemplo, cuando se bifurca hacia delante en un programa.

El archivo objeto, si es absoluto, contiene solamente los bytes binarios (00H-FFH) del programa en lenguaje máquina. Un archivo objeto relocalizable también contendrá una tabla de símbolos y otra información requerida para enlazamiento y localización. El archivo de listado contiene los códigos de texto en ASCII (20H-7EH) tanto para el programa fuente como para los bytes hexadecimales empleados en el programa escrito en lenguaje máquina.

Una buena demostración de la diferencia entre un archivo objeto y un archivo de listado es presentarlos en la pantalla de la computadora anfitriona (utilizando, por ejemplo, el comando TYPE en los sistemas con MS-DOS). El archivo de listado muestra claramente en cada línea de salida una dirección, un código de operación, y tal vez datos, seguidos de una instrucción de programa del archivo fuente. El archivo de listado se presenta adecuadamente debido a que sólo contiene códigos de texto en ASCII. Sin embargo, presentar el archivo objeto resulta problemático. La salida aparecerá como “basura”, ya que el archivo objeto contiene códigos binarios de un programa en lenguaje máquina del 8051 en lugar de códigos de texto en ASCII.

La figura 7-2 presenta la descripción de un ensamblador de dos pasos escrito en un seudolenguaje computacional (similar a Pascal o C) para mejorar su legibilidad.

7.3 FORMATO DE UN PROGRAMA EN LENGUAJE ENSAMBLADOR

Los programas desarrollados en lenguaje ensamblador contienen lo siguiente:

- Instrucciones máquina
- Directivas del ensamblador
- Controles del ensamblador
- Comentarios

Las instrucciones máquina son los mnemónicos familiares de las instrucciones ejecutables (por ejemplo, ANL). Las directivas del ensamblador son instrucciones para el programa ensamblador que definen la estructura del programa, los símbolos, datos, constantes, y así sucesivamente (por ejemplo, ORG). Los controles del ensamblador establecen los modos del ensamblador y dirigen su flujo (por ejemplo, \$TITLE). Los comentarios mejoran la legibilidad de los programas ya que explican el propósito y la operación de las secuencias de instrucciones.

Aquellas líneas que contengan instrucciones máquina o directivas del ensamblador deben escribirse siguiendo reglas específicas entendibles por el ensamblador. Cada línea está dividida en “campos” separados por un carácter de espacio o de tabulación. El formato general para cada línea es como sigue:

```
[etiqueta:] mnemónico [operando] [,operando] [...] [;comentario]
```

Sólo el campo del mnemónico es obligatorio. Muchos ensambladores requieren que el campo de la etiqueta, si está presente, empiece a la izquierda en la columna 1, y que los campos siguientes estén separados por un carácter de espacio o de tabulación. Con el ASM51, el campo de la etiqueta no tiene que comenzar en la columna 1 y el campo del mnemónico no necesita estar en la misma línea que el de la etiqueta. El campo del operando debe, sin embargo, comenzar en la misma línea que el campo del mnemónico. A continuación describimos los campos.

7.3.1 Campo de etiqueta

Una etiqueta representa la dirección de la instrucción (o datos) que le siguen. Cuando se bifurca esta instrucción, la etiqueta se utiliza en el campo de operando de la instrucción de bifurcación o de salto (por ejemplo, SJMP IGNORA).

Mientras que el término “etiqueta” siempre representa una dirección, el término “símbolo” es más general. Las etiquetas son un tipo de símbolo y se identifican por el requerimiento de que deben terminar con dos puntos (:). Se asignan valores o atributos a los símbolos, utilizando directivas tales como EQU, SEGMENT, BIT, DATA, etc. Los símbolos pueden ser direcciones, constantes de datos, nombres de segmentos, u otros elementos formulados por el programador. Los símbolos no terminan con dos puntos. En el ejemplo siguiente, PAR es un símbolo y COMIENZO es una etiqueta (la cual es un tipo de símbolo).

```

PAR      EQU    500          ; "PAR" ES UN SIMBOLO QUE
                           ; REPRESENTA AL VALOR 500
INICIO:   MOV    A, #0FFH     ; "INICIO" ES UNA ETIQUETA QUE
                           ; REPRESENTA LA DIRECCION DE
                           ; LA INSTRUCCION MOV

```

Un símbolo (o una etiqueta) debe comenzar con una letra, un signo de interrogación, o un signo de subrayado (_); seguido de letras, dígitos, “?” o “_”; y puede contener un máximo de 31 caracteres.¹ Los símbolos pueden utilizar caracteres en mayúsculas o minúsculas, pero se tratan igual. No se pueden utilizar palabras reservadas (mnemónicos, operadores, símbolos predefinidos ni directivas).

7.3.2 Campo de mnemónico

Los mnemónicos de instrucciones o las directivas del ensamblador pertenecen al campo de mnemónico, el cual sigue al campo de etiqueta. Algunos ejemplos de mnemónicos de instrucciones son ADD, MOV, DIV o INC. Ejemplos de directivas del ensamblador son ORG, EQU o DB. Describimos las directivas del ensamblador más adelante en este capítulo.

7.3.3 Campo de operando

El campo de operando sigue al campo de mnemónico. Este campo contiene la dirección o los datos utilizados por la instrucción. Se puede utilizar una etiqueta para representar la dirección de los datos o un símbolo para representar una constante de datos. Las posibilidades para el campo de operando son en gran parte dependientes de la operación. Algunas operaciones no tienen operandos (por ejemplo, la instrucción RET), mientras que otras permiten múltiples operandos separados por comas. De hecho, las posibilidades para el campo de operando son numerosas, y las examinaremos con mayor detalle. Pero primero describiremos el campo de comentario.

7.3.4 Campo de comentario

Los comentarios para clarificar el programa pertenecen al campo de comentario ubicado al final de cada línea. Los comentarios deben empezar con punto y coma (;). Líneas enteras pueden

¹Recordamos al lector que las reglas especificadas en este capítulo se aplican al ASM51 de Intel. Otros ensambladores pueden tener distintos requerimientos.

ser líneas de comentarios si se comienzan con punto y coma. Subrutinas y secciones grandes de un programa generalmente inician con un bloque de comentarios, el cual consta de varias líneas que explican las propiedades generales de la sección de software que sigue.

7.3.5 Símbolos especiales del ensamblador

Los símbolos especiales del ensamblador se utilizan en modos de direccionamiento específicos para los registros. Los registros incluyen a A, R0 a R7, DPTR, PC, C y AB. Además, se puede utilizar el signo de moneda (\$) para referirse al valor actual en el contador de ubicación. Continuamos con algunos ejemplos.

```
SETB C
INC DPTR
JNB TI, $
```

La última instrucción hace uso efectivo del contador de ubicación del ASM51 para evitar la utilización de una etiqueta. También puede escribirse como:

```
AQUI : JNB TI, AQUI
```

7.3.6 Dirección indirecta

Para algunas instrucciones, el campo de operando puede especificar un registro que contiene la dirección de los datos. El signo de arroba (@) indica direccionamiento indirecto y sólo puede utilizarse con R0, R1, el DPTR o el PC, dependiendo de la instrucción. Por ejemplo,

```
ADD A, @R0
MOVC A, @A+PC
```

La primera instrucción obtiene un byte de datos de la memoria RAM interna en la dirección especificada por R0. La segunda instrucción obtiene un byte de datos de la memoria externa para código en la dirección formada por la suma de los contenidos del acumulador al contador de programa. Observe que el valor del contador de programa, cuando se lleva a cabo la suma, es la dirección de la instrucción que sigue de MOVC. Para las dos instrucciones anteriores, el valor obtenido se coloca en el acumulador.

7.3.7 Datos inmediatos

Las instrucciones que utilizan direccionamiento inmediato proporcionan datos en el campo de operando que forman parte de la instrucción. Los datos inmediatos están precedidos por un signo de número (#). Por ejemplo,

```
CONSTANTE EQU 100
MOV A, #0FEH
ORL 40H, #CONSTANTE
```

Todas las operaciones de datos inmediatos (a excepción de MOV DPTR,#datos) requieren de ocho bits de datos. Los datos inmediatos se evalúan como una constante de 16 bits, y el byte inferior es el que se utiliza. Todos los bits incluidos en el byte superior deben ser iguales (00H o FFH), de lo

contrario se generará el mensaje de error “el valor no cabe en un byte”. Por ejemplo, las siguientes instrucciones son sintácticamente correctas:

```
MOV A, #0FF00H
MOV A, #00FFH
```

Pero las siguientes dos instrucciones generan mensajes de error:

```
MOV A, #0FE00H
MOV A, #01FFH
```

Si se utiliza la notación decimal con signo, también se pueden usar las constantes desde -256 hasta +255. Por ejemplo, las siguientes dos instrucciones son equivalentes (y sintácticamente correctas):

```
MOV A, #-256
MOV A, #0FF00H
```

Ambas instrucciones colocan el valor 00H en el acumulador A.

7.3.8 Dirección de datos

Muchas instrucciones acceden a ubicaciones en memoria utilizando el direccionamiento directo, y requieren de una dirección de memoria para datos incorporada en el chip (00H a 7FH) o una dirección de SFR (80H a 0FFH) en el campo de operando. Se pueden utilizar símbolos predefinidos para las direcciones de SFR. Por ejemplo,

```
MOV A, 45H
MOV A, SBUF ; IGUAL AS MOV A, 99H
```

7.3.9 Dirección de bit

Una de las características más poderosas del 8051 es la capacidad de acceder a bits individuales sin que sea necesario realizar operaciones de máscara sobre bytes. Las instrucciones que acceden a ubicaciones direccionables por bit deben proporcionar la dirección de bit en la memoria interna para datos (00H a 7FH) o una dirección de bit en los SFR (80H a 0FFH).

Existen tres maneras de especificar una dirección de bit en una instrucción: (a) en forma explícita, al proporcionar la dirección, (b) utilizando el **operador de punto** localizado entre la dirección de byte y la posición del bit, y (c) mediante un símbolo predefinido del ensamblador. A continuación presentamos algunos ejemplos.

```
SETB 0E7H ; DIRECCION DE BIT EXPLICITA
SETB ACC.7 ; OPERADOR DE PUNTO (IGUAL QUE LA ANTERIOR)
JNB TI,$ ; "TI" ES UN SIMBOLO PRE-DEFINIDO
JNB 99H,$ ; (IGUAL QUE LA ANTERIOR)
```

7.3.10 Dirección de código

Una dirección de código se utiliza en el campo de operando para las instrucciones de salto, incluyendo saltos relativos (SJMP y saltos condicionales), llamadas y saltos absolutos (ACALL, AJMP), y llamadas y saltos largos (LJMP, LCALL).

La dirección de código a menudo se da en forma de etiqueta. Por ejemplo,

```
AQUI : .
.
.
SJMP AQUI
```

El ASM51 determinará la dirección de código correcta y agregará a la instrucción el desplazamiento con signo de 8 bits correcto, la dirección de página de 11 bits, o la dirección larga de 16 bits, como sea adecuado.

7.3.11 Saltos genéricos y llamadas

El ASM51 permite que los programadores utilicen un mnemónico de JMP o CALL genérico. “JMP” se puede utilizar en lugar de SJMP, AJMP o LJMP; y “CALL” puede ir en lugar de ACALL o LCALL. El ensamblador convierte el mnemónico genérico en una instrucción “real” cuando sigue algunas reglas simples. El mnemónico genérico se convierte a la forma corta (sólo para JMP) si no se utilizan referencias hacia delante y el destino del salto está dentro de –128 ubicaciones, o a la forma absoluta si no se utilizan referencias hacia delante y la instrucción que sigue a JMP o a CALL se ubica en el mismo bloque de 2K que la instrucción de destino. Si no se pueden utilizar las formas cortas o absolutas, la conversión será de la forma larga.

La conversión no es necesariamente la mejor alternativa de programación. Por ejemplo, si ocurre una bifurcación hacia unas cuantas instrucciones adelante, la instrucción JMP genérica siempre se convertirá a LJMP aunque una SJMP sea una mejor alternativa. Consideremos la secuencia de instrucciones ensambladas que muestra la figura 7-3 utilizando tres saltos genéricos. El primer salto (línea 3) se ensambla como SJMP porque el destino está antes del salto (en otras palabras, no existen referencias hacia delante) y el desplazamiento es menor que –128. La directiva ORG en la línea 4 crea un espacio de 200 ubicaciones entre la etiqueta INICIO y el segundo salto, así que la línea 5 se convierte a AJMP porque el desplazamiento es demasiado grande para una SJMP. Observe también que la dirección que sigue del segundo salto (12FEH) y la dirección de INICIO (1234H) se encuentran dentro de la misma página de 2K, la cual, para esta secuencia de instrucciones, está limitada entre 1000H y 17FFH. Este criterio se debe cumplir para lograr un direccionamiento absoluto. El tercer salto se ensambla a LJMP porque el destino (FIN) aún no está definido cuando se ensambla el salto (en otras palabras, se está utilizando una referencia

LOC	OBJ	LINEA	FUENTE			
1234		1		ORG	1234H	
1234 04		2	INICIO:	INC	A	
1235 80FD		3		JMP	INICIO	; SE ENSAMBLA COMO SJMP
12FC		4		ORG	INICIO + 200	
12FC 4134		5		JMP	INICIO	; SE ENSAMBLA COMO AJMP
12FE 021301		6		JMP	FIN	; SE ENSAMBLA COMO LJMP
1301 04		7	FIN:	INC	A	
		8		END		

FIGURA 7–3

Uso del mnemónico genérico JMP

hacia delante). El lector puede verificar que la conversión ocurre como se ha descrito si examina el campo de objeto para cada instrucción de salto. Verifique los códigos presentados en hexadecimal con los encontrados en el apéndice C para SJMP, AJMP y LJMP.

7.4 EVALUACIÓN DE EXPRESIÓN AL TIEMPO DE ENSAMBLADO

Los valores y las constantes en el campo de operando pueden expresarse de tres maneras: (a) en forma explícita (por ejemplo, 0EFFH), (b) con un símbolo predefinido (por ejemplo, ACC) o (c) mediante una expresión (por ejemplo, 2 + 3). El uso de expresiones proporciona una técnica poderosa para hacer más legibles y flexibles los programas escritos en lenguaje ensamblador. Cuando se utiliza una expresión, el ensamblador calcula un valor y lo agrega a la instrucción.

Todos los cálculos de expresiones se realizan utilizando una aritmética de 16 bits; sin embargo, se agregan ya sea 8 o 16 bits a la instrucción conforme se necesite. Por ejemplo, las siguientes dos instrucciones son iguales:

```
MOV    DPTR, #04FFH + 3
MOV    DPTR, #0502H      ; SE UTILIZA EL RESULTADO ENTERO DE 16 BITS
```

No obstante, cuando se utiliza la misma expresión en una instrucción “MOV A,#data”, el ASM51 genera un error de “el valor no cabe en un byte”. A continuación presentamos una visión general de las reglas para evaluar expresiones.

7.4.1 Bases de números

La base para las constantes numéricas está indicada en la forma usual para los microprocesadores Intel. Las constantes deben estar seguidas de una “B” para números binarios, “O” o “Q” para octales, “D” o nada para decimales, o “H” para hexadecimales. Por ejemplo, las siguientes instrucciones son iguales:

```
MOV  A, #15
MOV  A, #1111B
MOV  A, #0FH
MOV  A, #17Q
MOV  A, #15D
```

Observemos que el primer carácter de las constantes decimales debe ser un dígito para diferenciarlas de las etiquetas (por ejemplo, “0A5H” y no “A5H”).

7.4.2 Cadenas de caracteres

Cadenas que utilicen uno o dos caracteres pueden utilizarse como operandos en expresiones. Los códigos en ASCII se convierten a su equivalente binario por el ensamblador. Las constantes de caracteres están enmarcadas por comillas sencillas (''). A continuación presentamos algunos ejemplos:

```
CJNEA, #'Q' , DENUEVO
SUBB A, #'0'          ; CONVIERTA UN DIGITO EN ASCII A UN
                      ; DIGITO
                      ; DIGITO EN BINARIO
MOV  DPTR, #'AB'
MOV  DPTR, #4142H      ; IGUAL QUE LA ANTERIOR
```

7.4.3 Operadores aritméticos

Los operadores aritméticos son

+	suma
-	resta
*	multiplicación
/	división
MOD	módulo (resto después de una división)

Por ejemplo, las siguientes dos instrucciones son iguales:

```
MOV A, 10 + 10H
MOV A, #1AH
```

Las siguientes dos instrucciones también son iguales:

```
MOV A, #25 MOD 7
MOV A, #4
```

Ya que el operador MOD puede confundirse con un símbolo, debe estar separado de sus operandos cuando menos por un carácter de espacio o de tabulación, o los operandos deben ir entre paréntesis. Lo mismo se aplica para los otros operadores compuestos por letras.

7.4.4 Operadores lógicos

Los operadores lógicos son

OR	OR lógico
AND	AND lógico
XOR	OR exclusivo lógico
NOT	NOT (complemento) lógico

La operación se aplica a los bits correspondientes en cada operando. El operador debe estar separado de los operandos por un carácter de espacio o de tabulación. Por ejemplo, las siguientes dos instrucciones son iguales:

```
MOV A, #'9' AND 0FH
MOV A, #9
```

El operador NOT emplea sólo un operando. Las siguientes tres instrucciones MOV son iguales:

TRES	EQU	3
MENOS_TRES	EQU	-3
	MOV	A, #(NOT TRES) + 1
	MOV	A, #MENOS_TRES
	MOV	A, #11111101B

7.4.5 Operadores especiales

Los operadores especiales son

SHR	desplazamiento a la derecha
SHL	desplazamiento a la izquierda
HIGH	byte superior
LOW	byte inferior
()	evalúa primero

Por ejemplo, las siguientes dos instrucciones son iguales:

```
MOV A, #8 SHL 1
MOV A, #10H
```

Las siguientes dos instrucciones también son iguales:

```
MOV A, #HIGH 1234H
MOV A, #12H
```

7.4.6 Operadores relacionales

Cuando se utiliza un operador relacional entre dos operandos, el resultado siempre es o falso (0000H) o verdadero (FFFFH). Los operadores son

EQ	=	igual a
NE	<>	no es igual a
LT	<	menor que
LE	<=	menor o igual que
GT	>	mayor que
GE	>=	mayor o igual que

Observe que para cada operador se aceptan dos formas (por ejemplo, “EQ” o “=”). En los siguientes ejemplos, todas las pruebas relacionales son “verdaderas”:

```
MOV A, #5 = 5
MOV A, #5 NE 4
MOV A, #'X' LT 'Z'
MOV A, #'X' >= 'X'
MOV A, #\$ > 0
MOV A, #100 GE 50
```

Por lo tanto, las instrucciones ensambladas son iguales a

```
MOV A, #0FFH
```

Aunque las expresiones se evalúan a resultados de 16 bits (en otras palabras, 0FFFFH), en los ejemplos anteriores sólo se utilizan los 8 bits de orden inferior ya que la instrucción es una operación de transferencia de un byte. El resultado no se considera demasiado grande en este caso, porque como números con signo, el valor de 16 bits FFFFH y el valor de 8 bits FFH son iguales (-1).

7.4.7 Ejemplos de expresiones

Los siguientes son ejemplos de expresiones y de los valores que resultan:

Expresión	Resultado
'B' - 'A'	0001H
8/3	0002H
155 MOD 2	0001H
4 * 4	0010H
8 AND 7	0000H
NOT 1	FFFEH
'A' SHL 8	4100H
LOW 65535	0FFFH
(8 + 1) * 2	0012H
5 EQ 4	0000H
'A' LT 'B'	FFFFH
3 <= 3	FFFFH

A continuación presentamos un ejemplo práctico que ilustra una operación común utilizada para la inicialización de un temporizador: colocar el valor de -500 en los registros TH1 y TL1 del temporizador 1. Una buena solución es utilizar los operadores HIGH y LOW como sigue

```
VALOR    EQU   -500
MOV    TH1, #HIGH VALOR
MOV    TL1, #LOW VALOR
```

El ensamblador convierte el valor -500 a su correspondiente valor de 16 bits (FE0CH); después los operadores HIGH y LOW extraen adecuadamente los bytes superior (FEH) e inferior (0CH) para cada instrucción MOV.

7.4.8 Precedencia de los operadores

La precedencia de los operadores de expresión desde el más alto hasta el más bajo es:

```
( )
HIGH LOW
* / MOD SHL SHR
+ -
EQ NE LT LE GT GE = <> < <= > >=
NOT
AND
OR XOR
```

Cuando se utilizan operadores con la misma precedencia, se evalúan de izquierda a derecha. Ejemplos:

Expresión	Valor
HIGH('A' SHL 8)	0041H
HIGH 'A' SHL 8	0000H
NOT 'A'-1	FFBFH
'A' OR 'A' SHL8	4141H

7.5 DIRECTIVAS DEL ENSAMBLADOR

Las directivas del ensamblador son instrucciones enviadas al programa ensamblador. No son instrucciones en lenguaje ensamblador ejecutables por el microprocesador destino. Sin embargo, se colocan en el campo de mnemónico del programa. A excepción de DB y DW, no tienen ningún efecto directo en el contenido de la memoria.

El ASM51 proporciona varias categorías de directivas:

- Control del estado del ensamblador (ORG, END, USING)
- Definición de símbolos (SEGMENT, EQU, SET, DATA, IDATA, XDATA, BIT, CODE)
- Inicialización/reservación de almacenamiento (DS, DBIT, DB, DW)
- Enlazamiento de programas (PUBLIC, EXTRN, NAME)
- Selección de segmentos (RSEG, CSEG, DSEG, ISEG, ESEG, XSEG)

A continuación presentamos cada una de las directivas del ensamblador, ordenadas por categoría.

7.5.1 Control del estado del ensamblador

7.5.1.1 ORG (establece origen) El formato de la directiva ORG (establece origen) es

ORG expresión

La directiva ORG altera el contador de ubicación para establecer un nuevo origen del programa para las instrucciones que le siguen. Una etiqueta no está permitida. A continuación presentamos dos ejemplos.

```
ORG    100H           ;ESTABLECE CONTADOR DE UBICACION
                  A 100H
ORG    ($ + 1000H) AND 0F000H ;ESTABLECE ORIGEN EN LOS
                  SIGUIENTES LIMITES DE 4K
```

La directiva ORG se puede utilizar en cualquier tipo de segmento. Si el segmento actual es absoluto, el valor será una dirección absoluta en el segmento actual. Si un segmento relocalizable está activo, el valor de la expresión ORG se trata como un desplazamiento desde la dirección base de la instancia actual del segmento.

7.5.1.2 End El formato de la directiva END es

END

La directiva END debe ser la última instrucción en el archivo fuente. No se permite ninguna etiqueta y el ensamblador no procesa nada después de la instrucción END.

7.5.1.3 Using El formato de la directiva USING es

USING expresión

Esta directiva informa al ASM51 del banco de registros activo. Usos subsecuentes de las direcciones de registro simbólicas predefinidas AR0 a AR7 las convertirán en la dirección directa adecuada para el banco de registro activo. Consideremos la siguiente secuencia:

```
USING      3
PUSH      AR7
```

```
USING      1
PUSH       AR7
```

La primera instrucción PUSH se ensambla a PUSH 1FH (R7 en el banco 3), mientras que la segunda se ensambla a PUSH 0FH (R7 en el banco 1).

Observe que la directiva USING de hecho no cambia entre bancos de registros; solamente informa al ASM51 del banco activo. La única manera de cambiar entre bancos de registros es mediante la ejecución de instrucciones en el 8051. Podemos ilustrar esto si modificamos el ejemplo anterior como sigue:

```
MOV      PSW, #00011000B ; SELECCIONA BANCO DE REGISTROS 3
USING    3
PUSH     AR7             ; ENSAMBLA A PUSH 1FH
MOV      PSW, #00001000B ; SELECCIONA BANCO DE REGISTROS 1
USING    1
PUSH     AR7             ; ENSAMBLA A PUSH 0FH
```

7.5.2 Definición de símbolos

Las directivas de definición de símbolos crean símbolos que representan segmentos, registros, números y direcciones. Ninguna de estas directivas puede estar precedida por una etiqueta. Los símbolos definidos por estas directivas no deben estar previamente definidos y no se pueden redefinir de ninguna manera. La directiva SET es la única excepción. Enseguida describimos las directivas de definición de símbolos.

7.5.2.1 Segmento El formato de la directiva SEGMENT se muestra a continuación.

```
símbolo   SEGMENT tipo_de_segmento
```

El símbolo es el nombre de un segmento relocalizable. En el uso de segmentos, el ASM51 es más complejo que los ensambladores convencionales, los cuales generalmente soportan sólo tipos de segmento de “código” y de “datos”. Sin embargo, el ASM51 define tipos de segmento adicionales para acomodar los diversos espacios de memoria en el 8051. Los siguientes son los tipos de segmento (espacios en memoria) definidos en el 8051:

- CODE (el segmento de código)
- XDATA (el espacio externo para datos)
- DATA (el espacio interno para datos accesible mediante direccionamiento directo, 00H-7FH)
- IDATA (el espacio entero interno para datos accesible mediante direccionamiento indirecto, 00H-7FH, 00H-FFH en el 8052)
- BIT (el espacio para bits; ubicaciones de byte traslapadas 20H-2FH del espacio interno para datos)

Por ejemplo, la instrucción

```
EPROM   SEGMENT   CODE
```

declara al símbolo EPROM como un SEGMENT de tipo CODE. Advierta que esta instrucción simplemente declara lo que es EPROM. Para comenzar a usar este segmento, utilizamos la directiva RSEG (más adelante).

7.5.2.2 EQU (Equate, o igualdad)

El formato de la directiva EQU es

Símbolo EQU expresión

La directiva EQU asigna un valor numérico al nombre del símbolo especificado. El símbolo debe ser un nombre de símbolo válido, y la expresión debe respetar las reglas antes descritas.

Los siguientes son ejemplos de la directiva EQU:

```

N27      EQU 27          ;ASIGNA EL VALOR 27 A N27
AQUI     EQU $           ;ASIGNA EL VALOR DEL CONTADOR
        ;DE UBICACION A "AQUI"
CR       EQU 0DH         ;ASIGNA EL VALOR 0DH A CR
                    ;(RETORNO DE CARRO)
MENSAJE: DB 'Este es un mensaje'
LONGITUD  EQU $ - MENSAJE ;"LONGITUD" ES IGUAL A LA
                           LONGITUD DEL MENSAJE

```

7.5.2.3 Otras directivas de definición de símbolos La directiva SET es similar a la directiva EQU, salvo que el símbolo puede redefinirse después utilizando otra directiva SET.

Las directivas DATA, IDATA, XDATA, BIT y CODE asignan las direcciones del tipo de segmento correspondiente a un símbolo. Estas directivas no son esenciales. Podemos lograr un efecto similar mediante la directiva EQU; no obstante, si las utilizamos, provocan una poderosa verificación de tipos por el ASM51. Consideremos las siguientes dos directivas y cuatro instrucciones:

```

BANDERA1  EQU 05H
BANDERA2  BIT 05H
SETB BANDERA1
SETB BANDERA2
MOV BANDERA1, #0
MOV BANDERA2, #0

```

El uso de BANDERA2 en la última instrucción de esta secuencia generará un mensaje de error de “dirección de segmento de datos esperada” del ASM51. Debido a que BANDERA2 está definida como una dirección de bit (utilizando la directiva BIT), se puede usar en una instrucción para activar un bit, pero no es posible utilizarla en una instrucción para transferir un byte. Esto, por lo tanto, causa el error. Aunque BANDERA1 representa el mismo valor (05H), fue definida utilizando EQU y no tiene asociado un espacio en memoria. Esta no es una ventaja de EQU, sino una desventaja. Al definir adecuadamente los símbolos de direcciones para utilizarlos en un espacio en memoria específico (mediante las directivas BIT, DATA, XDATA, etc.), el programador toma ventaja de la poderosa verificación de tipos del ASM51 y evita errores de programación debidos al mal uso de los símbolos.

7.5.3 Inicialización/reservación de almacenamiento

Las directivas de inicialización y reservación de almacenamiento inicializan y reservan espacio, ya sea en unidades de palabras, en bytes o en bits. El espacio reservado empieza en la ubicación indicada por el valor actual del contador de ubicación en el segmento activo. Estas directivas pueden estar precedidas por una etiqueta. A continuación describimos las directivas de inicialización/reservación de almacenamiento.

7.5.3.1 DS (definición de almacenamiento) El formato para la directiva DS (definición de almacenamiento) es

[etiqueta:] DS expresión

La directiva DS reserva espacio en unidades de bytes. Puede utilizarse en cualquier tipo de segmento excepto en el de BIT. La expresión debe ser una expresión válida al tiempo de ensamblado sin referencias hacia delante ni referencias no relocalizables o externas. Cuando se encuentra una instrucción DS en un programa, el contador de ubicación del segmento actual se incrementa mediante el valor de la expresión. La suma del contador de ubicación y de la expresión especificada no debe exceder las limitaciones del espacio de direcciones actual.

Las siguientes instrucciones crean un búfer de 40 bytes en el segmento de datos interno:

```
DSEG AT 30H      ; COLOCA EN SEGMENTO DE DATOS
                  ; (ABSOLUTO, INTERNO)
LONGITUD EQU 40
BUFER:   DS    LONGITUD ; 40 BYTES RESERVADOS
```

La etiqueta BUFER representa la dirección de la primera ubicación de memoria reservada. Para este ejemplo, el búfer empieza en la dirección 30H porque se especifica “AT 30H” con DSEG. (Consulte la sección 7.5.5.2 Selección de segmentos absolutos). Este búfer puede limpiarse utilizando la siguiente secuencia de instrucciones:

```
MOV  R7 , #LONGITUD
MOV  R0 , #BUFER
CICLO: MOV  @R0 , #0
        DJNZ R7 , CICLO
                  (continúa)
```

Podemos utilizar las siguientes directivas para crear un búfer de 1000 bytes en memoria RAM externa, comenzando en 4000H:

```
XINICIO EQU 4000H
XLONGITUD EQU 1000
XSEG      AT XINICIO
XBUFER:  DS XLONGITUD
```

Podemos limpiar este búfer con la siguiente secuencia de instrucciones:

```
MOV  DPTR , #XBUFER
CICLO: CLR  A
        MOVX @DPTR , A
        INC  DPTR
        MOV  A , DPL
        CJNE A , #LOW (XBUFER + XLONGITUD + 1) , CICLO
        MOV  A , DPH
        CJNE A , #HIGH (XBUFER + XLONGITUD + 1) , CICLO
                  (continúa)
```

Éste es un ejemplo excelente de un poderoso uso de los operadores y de las expresiones en tiempo de ensamblado del ASM51. Debido a que no existe una instrucción para comparar el apuntador de datos con un valor inmediato, la operación se debe implementar mediante las instrucciones disponibles. Se requieren dos comparaciones, una para cada uno de los bytes superior e inferior del DPTR. Además, la instrucción de compara-y-salta-si-no-es-igual funciona sólo con el acumulador o con un registro, así que los bytes del apuntador de datos deben ser transferidos al acumulador antes de la instrucción CJNE. El ciclo termina sólo cuando el apuntador de datos ha

llegado a XBUFER + XLONGITUD + 1. (Necesitamos la expresión “+ 1” ya que el apuntador de datos se incrementa después de la última instrucción MOVX).

7.5.3.2 DBIT

El formato de la directiva DBIT (definición de bit) es,

[etiqueta:] DBIT expresión

La directiva DBIT reserva espacio en unidades de bit. Puede utilizarse únicamente en un segmento de tipo BIT. La expresión debe ser válida al tiempo de ensamblador sin referencias hacia delante. Cuando se encuentra la instrucción DBIT en un programa, el contador de ubicación del segmento actual (BIT) se incrementa por el valor de la expresión. Advierta que en un segmento BIT, la unidad básica del contador de ubicación está en bits en lugar de en bytes. Las siguientes directivas crean tres banderas en un segmento de bit absoluto:

BSEG	;	SEGMENTO DE BIT (ABSOLUTO)
TECBANDERA:	DBIT 1	; ESTATUS DEL TECLADO
IMPBANDERA:	DBIT 1	; ESTATUS DE IMPRESORA
DISCOBANDERA:	DBIT 1	; ESTATUS DE DISCO

Debido a que la dirección no está especificada con BSEG en el ejemplo anterior, la dirección de las banderas definidas por DBIT puede determinarse (si lo desea) si examinamos la tabla de símbolos en los archivos .LST o .M51. (Consulte las figuras 7-1 y 7-6). Si las definiciones anteriores fueron las primeras de BSEG utilizadas, entonces TECBANDERA estará en la dirección de bit 00H (bit 0 de la dirección de byte 20H; consulte la figura 2-6). Si otros bits fueron previamente definidos utilizando BSEG, entonces las definiciones anteriores seguirían del último bit definido. (Consulte la sección 7.5.5.2 Selección de segmentos absolutos).

7.5.3.3 DB (definición de bytes)

El formato de la directiva DB (definición de bytes) es

[etiqueta:] DB expresión [,expresión] [. . .]

La directiva DB inicializa la memoria para código con valores de bytes. Un segmento tipo CODE debe estar activo ya que la directiva se utiliza para colocar constantes de datos en la memoria para código. La lista de expresiones es una serie de uno o más valores de byte (cada uno de los cuales puede ser una expresión) separadas por comas.

La directiva DB permite implementar cadenas de caracteres (entre comillas sencillas) más largas que dos caracteres siempre y cuando no sean parte de una expresión. Cada carácter incluido en la cadena se convierte a su correspondiente código en ASCII. Si se utiliza una etiqueta, se le asigna la dirección del primer byte. Por ejemplo, las siguientes instrucciones

CSEG AT 0100H			
CUADRADOS:	DB 0,1,4,9,16,25	;	CUADRADOS DE LOS NUMEROS 0-5
MENSAJE:	DB 'Login:',0	;	CADENA DE CARACTERES TERMINADA CON UN CARACTER NULO

resultan, cuando se ensamblan, en las siguientes asignaciones de memoria en hexadecimal para la memoria externa para código:

Dirección	Contenido
0100	00
0101	01

0102	04
0103	09
0104	10
0105	19
0106	4C
0107	6F
0108	67
0109	69
010A	6E
010B	3A
010C	00

7.5.3.4 DW (definición de palabra) El formato de la directiva DW (definición de palabra) es

```
[etiqueta:] DW expresión [,expresión] [.. .]
```

La directiva DW es la misma que la directiva DB, salvo que se asignan dos ubicaciones en memoria (16 bits) para cada elemento de datos. Por ejemplo, las instrucciones

```
CSEG AT 200H
DW $, 'A', 1234H, 2, 'BC'
```

resultan en las siguientes asignaciones de memoria en hexadecimal:

Dirección	Contenido
0200	02
0201	00
0202	00
0203	41
0204	12
0205	34
0206	00
0207	02
0208	42
0209	43

7.5.4 Enlazamiento de programas

Las directivas de enlazamiento de programas permiten que los módulos ensamblador (archivos) se comuniquen por separado unos con otros mediante las referencias intermodulares y la nomenclatura de módulos. Podemos considerar a un “módulo” como un “archivo” en el siguiente análisis. (De hecho, un módulo puede abarcar más de un archivo).

7.5.4.1 Public El formato de la directiva PUBLIC (símbolo público)

```
PUBLIC símbolo [,símbolo] [.. .]
```

La directiva PUBLIC permite conocer y utilizar la lista de símbolos especificados fuera del módulo ensamblador actual. Un símbolo declarado como PUBLIC debe estar definido en el

módulo actual. Declarándolo PUBLIC se permite que otro módulo haga referencia a él. Por ejemplo,

```
PUBLIC CARENT, CARSAL, LINEAENT, CADSAL
```

7.5.4.2 Extn

El formato de la directiva EXTRN (símbolo externo) es

```
EXTRN tipo_de_segmento(símbolo [, símbolo] [. . .], . . .)
```

La directiva EXTRN presenta una lista de los símbolos a los que se hará referencia en el módulo actual y que están definidos en otros módulos. La lista de los símbolos externos debe tener un tipo de segmento asociado con cada símbolo incluido en la lista. (Los tipos de segmento son CODE, XDATA, DATA, IDATA, BIT y NUMBER. NUMBER es un símbolo sin tipo definido por EQU). El tipo de segmento indica la manera en que se puede utilizar un símbolo. La información es importante al tiempo de enlazamiento para asegurar que los símbolos se están utilizando adecuadamente en los diferentes módulos.

Las directivas PUBLIC y EXTRN trabajan juntas. Consideremos los dos archivos mostrados en la figura 7-4, PRINCIPAL.SRC y MENSAJES.SRC. Las subrutinas HOLA y ADIOS están definidas en el módulo MENSAJES, pero se pueden volver disponibles para otros módulos utilizando la directiva PUBLIC. Las subrutinas son llamadas en el módulo PRINCIPAL aunque no están definidas en él. La directiva EXTRN declara que estos símbolos están definidos en otro módulo.

PRINCIPAL.SRC

EXTRN	CODE (HOLA,ADIOS)

CALL	HOLA

CALL	ADIOS

END	

MENSAJES.SRC

PUBLIC	(HOLA,ADIOS)

HOLA	(subrutina inicia)

RET	
ADIOS	(subrutina inicia)

RET	

END	

FIGURA 7-4

Uso de las directivas EXTRN y PUBLIC del ensamblador

Ni PRINCIPAL.SRC ni MENSAJES.SRC son programas completos; deben ensamblarse por separado y enlazarse juntos para formar un programa ejecutable. Las referencias externas se resuelven durante el enlazamiento con las direcciones correctas agregadas para establecer el destino de las instrucciones CALL.

7.5.4.3 Name

El formato de la directiva NAME es

```
NAME nombre_del_módulo
```

Todas las reglas usuales para los nombres de los símbolos se aplican a los nombres de los módulos. Si no se proporciona un nombre, el módulo toma el nombre del archivo (sin el especificador de unidad o subdirectorio y sin una extensión). Un programa contendrá un módulo para cada archivo si no existe la directiva NAME. El concepto de “módulos”, por lo tanto, es un tanto ineficaz, al menos para problemas de programación relativamente pequeños. Aún así, los programas de tamaño moderado (abarcando, por ejemplo, varios archivos completos con segmentos relocalizables) no necesitan usar la directiva NAME ni poner alguna atención especial al concepto de “módulos”. Por ello mencionamos en la definición que un módulo puede considerarse como un “archivo” para simplificar el aprendizaje del ASM51. Sin embargo, para programas bastante grandes (varios miles de líneas de código o más), tiene sentido particionar el problema en módulos, donde, por ejemplo, cada módulo abarca varios archivos que contienen rutinas con un propósito común.

7.5.5 Directivas de selección de segmentos

Cuando el ensamblador encuentra una directiva de selección de segmentos, desvía el código o los datos siguientes al segmento seleccionado hasta que se seleccione otro segmento mediante una directiva de selección de segmentos. La directiva puede seleccionar un segmento relocalizable previamente definido o crear y seleccionar de manera alterna segmentos absolutos.

7.5.5.1 RSEG (segmento relocalizable)

El formato de la directiva RSEG (segmento relocalizable) es

```
RSEG nombre_del_segmento
```

donde “nombre_del_segmento” es el nombre de un segmento relocalizable previamente definido con la directiva SEGMENT. RSEG es una directiva de “selección de segmento” que desvía el código o los datos subsecuentes al segmento con nombre hasta que se encuentra otra directiva de selección de segmentos.

7.5.5.2 Selección de segmentos absolutos

RSEG selecciona un segmento relocalizable. Un segmento “absoluto”, por otra parte, se selecciona mediante una de las siguientes directivas:

```
CSEG (AT dirección)
DSEG (AT dirección)
ISEG (AT dirección)
BSEG (AT dirección)
XSEG (AT dirección)
```

Estas directivas seleccionan un segmento absoluto dentro de los espacios de direcciones para código, datos internos, datos internos indirectos, de bit o de datos externos, respectivamente. Al proporcionar una dirección absoluta (mediante la expresión “AT dirección”), el ensamblador termina el último segmento de direcciones absolutas, si existe uno, del tipo de segmento especificado y crea un segmento absoluto nuevo iniciando en esa dirección. Si la dirección absoluta no está especificada, se continúa con el último segmento absoluto del tipo especificado. Cuando no se seleccione previamente un segmento absoluto de este tipo y la dirección absoluta esté omitida, se creará un nuevo segmento iniciando en la ubicación 0. Las referencias hacia delante no están permitidas y las direcciones de inicio deben ser absolutas.

Cada segmento tiene su propio contador de ubicación, y al principio siempre es igual a 0. El segmento predeterminado es un segmento de código absoluto; por lo tanto, el estado inicial del ensamblador es la ubicación 0000H en el segmento de código absoluto. Cuando se selecciona otro segmento por primera vez, el contador de ubicación del segmento anterior retiene el último valor activo. Al volver a seleccionar el segmento anterior, el contador de ubicación continúa desde el último valor activo. La directiva ORG puede utilizarse para cambiar el contador de ubicación dentro del segmento seleccionado. La figura 7-5 muestra ejemplos de la definición e inicio de segmentos relocalizables y absolutos.

Las primeras dos líneas de la figura 7-5 declaran los símbolos ENCHIP y EPROM como segmentos de tipo DATA (memoria RAM interna para datos) y CODE, respectivamente. La línea 4 inicia un segmento de bit absoluto comenzando en la dirección de bit 70H (bit 0 de la dirección de byte 2EH; consulte la figura 2-6). A continuación, se crean las etiquetas BANDERA1 y BANDERA2 correspondientes a las ubicaciones direccionables por bit de 70H y 71H. En la línea 8, RSEG inicia el segmento relocalizable ENCHIP de memoria RAM interna para datos. Las etiquetas TOTAL y CONTEO corresponden a ubicaciones de byte. SUMA16 es una etiqueta que corresponde a una ubicación de palabra (dos bytes). La siguiente ocurrencia de RSEG en la línea 13 inicia el segmento relocalizable EPROM de memoria para código. La etiqueta INICIO es la

LOC	OBJ	LÍNEA	FUENTE	
		1	ENCHIP	SEGMENT
		2	EPROM	SEGMENT
		3		
----		4	BSEG	AT 70H ;inicia segmento de bit absoluto
0070		5	BANDERA1:	DBIT 1
0071		6	BANDERA2:	DBIT 2
		7		
----		8	RSEG	ENCHIP ;inicia segmento de datos relocalizable
0000		9	TOTAL:	DS 1
0001		10	CONTEO:	DS 1
0002		11	SUMA16:	DS 2
12				
----		13	RSEG	EPROM ;inicia segmento de código relocalizable
0000 750000 F		14	INICIO:	MOV TOTAL,#0
		15		(programa continúa)
		16		END

FIGURA 7-5

Definición e inicio de segmentos absolutos y relocalizables

dirección de la primera instrucción en esta instancia del segmento EPROM. Observemos que no es posible determinar la dirección de las etiquetas TOTAL, CONTEO, SUMA16, e INICIO de la figura 7-5. Como estas etiquetas ocurren en segmentos relocalizables, el archivo objeto debe procesarse mediante el enlazador/localizador (consulte la sección 7.7 Operación del enlazador) con las direcciones iniciales especificadas para los segmentos ENCHIP y EPROM. El archivo de listado .M51 creado por el enlazador/localizador presenta las direcciones absolutas para estas etiquetas. BANDERA1 y BANDERA2, sin embargo, siempre corresponderán a las direcciones de bit 70H y 71H porque están definidas en un segmento BIT absoluto.

7.6 CONTROLES DEL ENSAMBLADOR

Los controles del ensamblador establecen el formato de los archivos objeto y de listado mediante la regulación de las acciones del ASM51. En su mayoría, los controles del ensamblador afectan la presentación del archivo de listado sin tener ningún efecto en el programa mismo. Pueden agregarse a la línea de invocación cuando se ensambla un programa o sumarse al archivo fuente. Los controles del ensamblador que aparecen en el archivo fuente deben estar precedidos por un signo de moneda (\$) e iniciar en la columna 1.

Existen dos categorías de controles del ensamblador: primarios y generales. Los controles primarios pueden agregarse a la línea de invocación o al inicio del programa fuente. Sólo otros controles primarios pueden preceder a un control primario. Los controles generales se pueden agregar en cualquier parte del programa fuente. La figura 7-6 muestra los controles del ensamblador que el ASM51 soporta.

7.7 OPERACIÓN DEL ENLAZADOR

Cuando desarrollamos programas de aplicación a gran escala, es común dividir las tareas en subprogramas o módulos que contienen secciones de código (a menudo subrutinas) que pueden escribirse en forma separada del programa completo. El término “programación modular” se refiere a esta estrategia de programación. Generalmente, los módulos son relocalizables, lo cual significa que no están destinados a ubicarse en algún espacio de direcciones para código o para datos específico. Se necesita un programa enlazador y localizador para combinar los módulos en un módulo de objetos absoluto que pueda ser ejecutado.

El RL51 de Intel es un enlazador/localizador típico. Procesa una serie de módulos de objetos relocalizables como entrada y crea un programa en lenguaje máquina ejecutable (PROGRAMA, tal vez) y un archivo de listado que contiene un mapa de memoria y una tabla de símbolos (PROGRAMA.M51). Esto se ilustra en la figura 7-7.

Conforme los módulos relocalizables se combinan, todos los valores para los símbolos externos con valores agregados al archivo de salida son resueltos. El enlazador se invoca del indicador de comandos del sistema con

```
RL51 lista_de_entradas [archivo_de_salida] [controles_de_ubicación]
```

La lista_de_entradas es una lista de módulos de objetos relocalizables (archivos) separados por comas. El archivo_de_salida es el nombre del módulo de objetos absoluto de salida. Cuando no se proporciona un nombre, se utiliza el nombre del primer archivo de entrada sin sufijo predefinido. Los controles_de_ubicación establecen las direcciones de inicio para los segmentos con nombre.

Por ejemplo, supongamos que se van a combinar tres módulos o archivos (PRINCIPAL.OBJ, MENSAJES.OBJ, y SUBRUTINAS.OBJ) en un programa ejecutable (EJEMPLO), y que cada uno de estos módulos contiene dos segmentos relocalizables, uno llamado EPROM

NOMBRE	PRIMARIO/ GENERAL	POR DEFECTO	ABREVIATURA	SIGNIFICADO
DATE (fecha)	P	DATE()	DA	Colocar una cadena en el encabezado (máximo 9 caracteres)
DEBUG	P	NODEBUG	DB	Escribir información de símbolos de depuración al archivo objeto
NODEBUG	P	NODEBUG	NODB	No escribir información de símbolos al archivo objeto
EJECT	G	no aplica	EJ	Para continuar el listado en la página siguiente
ERRORPRINT(archivo)	P	NOERRORPRINT	EP	Indica que un archivo recibirá mensajes de error además del archivo de listado (la consola predeterminada)
NOERRORPRINT	P	NOERRORPRINT	NOEP	Indica que los mensajes de error sólo se presentarán en el archivo de listado
GEN	G	GENONLY	GO	Listar sólo la fuente completamente expandida como si todas las líneas generadas por una llamada a una macro existieran ya en el archivo fuente
GENONLY	G	GENONLY	NOGE	Listar sólo el texto fuente original en el archivo de listado
INCLUDE(archivo)	G	no aplica	IC	Indica que un archivo se incluirá como parte del programa
LIST	G	LIST	LI	Imprimir líneas subsecuentes del código fuente en el archivo de listado
NOLIST	G	LIST	NOLI	No imprimir líneas subsecuentes del código fuente en el archivo de listado
MACRO (porcentaje_de _memoria)	P	MACRO(50)	MR	Evaluar y expandir todas las llamadas a macros. Separa un porcentaje de la memoria libre para el procesamiento de macros
NOMACRO	P	MACRO(50)	NOMR	No evaluar llamadas a macros
MOD51	P	MOD51	MO	Reconocer los registros con funciones especiales predefinidos específicos para el 8051
NOMOD(51)	P	MOD51	NOMO	No reconocer los registros con funciones especiales predefinidos específicos para el 8051

FIGURA 7–6

Controles del ensamblador soportados por el ASM51

NOMBRE	PRIMARIO/ GENERAL	POR DEFECTO	ABREVIATURA	SIGNIFICADO
OBJECT (archivo)	P	OBJECT (fuente.OBJ)	OJ	Indica que el archivo recibirá código objeto
NOOBJECT	P	OBJECT (fuente.OBJ)	NOOJ	Indica que no se creará ningún archivo objeto
PAGING	P	PAGING	PI	Indica que el archivo de listado se separará en páginas y cada página tendrá un encabezado
NOPAGING	P	PAGING	NOPI	Indica que el archivo de listado no contendrá saltos de página
PAGELENGTH(N)	P	PAGELENGT(60)	PL	Establece el máximo número de líneas en cada página del archivo de listado (rango = 10 a 65,536)
PAGEWIDTH(N)	P	PAGEWIDTH(120)	PW	Establece el máximo número de caracteres permitidos en cada línea del archivo de listado (rango = 72 a 132)
PRINT(archivo)	P	PRINT(fuente.LST)	PR	Indica que el archivo recibirá el listado fuente
NOPRINT	P	PRINT(fuente.LST)	NOPR	Indica que no se creará un archivo de listado
SAVE	G	no aplica	SA	Almacena los actuales atributos de control de la pila SAVE
RESTORE	G	no aplica	RS	Recupera los atributos de control de la pila SAVE
REGISTERBANK(rb,...)	P	REGISTERBANK(0)	RB	Indica que uno o más bancos se utilizan en el módulo del programa
NOREGISTERBANK	P	REGISTERBANK(0)	NORB	Indica que no se utilizan bancos de registro
SYMBOLS	P	SYMBOLS	SB	Crea una tabla formateada de todos los símbolos utilizados en el programa
NOSYMBOLS	P	SYMBOLS	NOSB	Indica que no se creará una tabla de símbolos
TITLE(cadena)	G	TITLE()	TT	Coloca una cadena en los subsecuentes encabezados de página (máximo 60 caracteres)
WORKFILES(directorio)	P	igual que la fuente	WF	Indica un directorio alternativo para archivos de trabajo temporales
XREF	P	NOXREF	XR	Crea un listado de referencia que lista todos los símbolos utilizados en un programa
NOXREF	P	NOXREF	NOXR	Indica que no se creará una lista de referencia

FIGURA 7–6

(continuación). Controles del ensamblador soportados por el ASM51

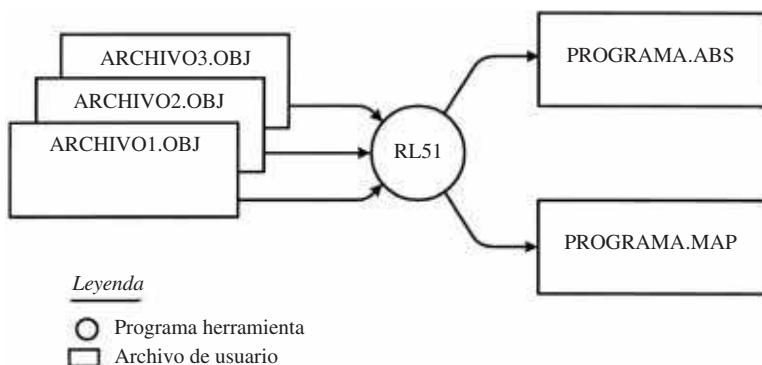


FIGURA 7-7
Operación del enlazador

de tipo CODE y el otro llamado ENCHIP de tipo DATA. Además, supongamos que el segmento de código se ejecutará en la dirección 4000H y que el segmento de datos iniciará en la dirección 30H (en memoria RAM interna). Podemos invocar al enlazador como sigue:

```
RL51 PRINCIPAL.OBJ, MENSAJES.OBJ, SUBRUTINAS.OBJ TO EJEMPLO & CODE
(EPROM(4000H)) DATA(ENCHIP(30H))
```

Observe que “&” se utiliza como el carácter de continuación de línea.

Si el programa inicia en la etiqueta INICIO, y esta es la primera instrucción en el módulo PRINCIPAL, entonces la ejecución comienza en la dirección 4000H. Si el módulo PRINCIPAL no fue enlazado primero, o si la etiqueta INICIO no está al comienzo del módulo PRINCIPAL, entonces el punto de entrada del programa se puede determinar examinando la tabla de símbolos en el archivo de listado EJEMPLO.M51 creado por RL51. De manera predeterminada, EJEMPLO.M51 contendrá sólo el mapa de enlace. Si deseamos una tabla de símbolos, cada programa fuente debe haber utilizado el control SDEBUG. (Consulte la figura 7-6).

7.8 EJEMPLO CON NOTAS: ENLACE DE SEGMENTOS Y MODULOS RELOCALIZABLES

Reunimos muchos de los conceptos que acabamos de presentar en un ejemplo con notas de un programa simple para el 8051. El código fuente está dividido a lo largo de dos archivos y utiliza símbolos declarados como EXTRN o PUBLIC para permitir la comunicación entre archivos. Cada archivo es un módulo —uno se llama PRINCIPAL y el otro SUBRUTINAS—. El programa utiliza un segmento de código relocalizable llamado EPROM y un segmento de datos interno relocalizable llamado ENCHIP. Trabajar con múltiples archivos, módulos y segmentos es esencial en proyectos de programación grandes. Un cuidadoso examen del ejemplo que sigue reforzará estos conceptos fundamentales y preparará al lector para embarcarse en diseños prácticos basados en el 8051.

Nuestro ejemplo es un programa simple de entrada/salida que utiliza el puerto serial del 8051 y teclado y pantalla de una terminal de video. El programa realiza lo siguiente:

- Inicializa el puerto serial (una sola vez)
- Despliega en pantalla la petición de “Escriba un comando.”

- Recibe una línea del teclado como entrada y despliega cada carácter en la pantalla al tiempo de recibirla
- Despliega en pantalla la línea completa
- Se repite

La figura 7-8 muestra (a) el archivo de listado (PANTALLA.LST) para el primer archivo fuente, (b) el archivo de listado (ES.LST) para el segundo archivo fuente, y (c) el archivo de listado (EJEMPLO.M51) creado por el enlazador/localizador.

7.8.1 PANTALLA.LST

La figura 7-8a muestra el contenido del archivo PANTALLA.LST creado por el ASM51 cuando el archivo fuente (PANTALLA.SRC) fue ensamblado. Las primeras líneas incluidas en el archivo de listado proporcionan información general sobre el ambiente de programación. Entre otras cosas, la línea de invocación se restablece en una forma expandida que muestra el directorio de los archivos. Observemos el uso del control del ensamblador EP (por ERRORPRINT) en la línea de invocación. Esto causa que se envíen mensajes de error a la consola y al archivo de listado. (Consulte la figura 7-6).

El archivo fuente original se muestra bajo la columna llamada SOURCE, justo a la derecha de la columna LINE. Es evidente que el archivo PANTALLA.SRC contiene 22 líneas. Las líneas 1 a 4 contienen controles del ensamblador. (Consulte la figura 7-6). \$DEBUG en la línea 1 instruye al ASM51 para que coloque una tabla de símbolos en el archivo objeto, PANTALLA.OBJ. Esto es necesario para emulación de hardware o para que el enlazador/localizador pueda crear una tabla de símbolos en su archivo de listado. \$TITLE define una cadena para que se coloque al inicio de cada página en el archivo de listado. \$PAGEWIDTH especifica la máxima anchura de cada línea en el archivo de listado, \$NOPAGING previene que los saltos de página (alimentación de forma) se agreguen al archivo de listado. La mayor parte de los controles del ensamblador afectan la presentación del archivo de listado de salida. Se puede llegar a la salida para impresión deseada con el método de ensayo y error.

La directiva del ensamblador NAME en la línea 6 define el archivo actual como parte del módulo PRINCIPAL. Para este ejemplo, no se utiliza nada más que esta instancia del módulo PRINCIPAL; sin embargo, proyectos grandes pueden incluir otros archivos definidos también como parte del módulo PRINCIPAL. El lector puede ayudarse a entender el resto de este ejemplo si sustituye el término “módulo” por “archivo”.

Las líneas 7 y 8 identifican los símbolos utilizados en el módulo actual, pero que están definidos en alguna otra parte. Sin estas directivas EXTRN, el ASM51 generaría el mensaje “símbolo indefinido” en cada línea del programa fuente donde se utiliza uno de estos símbolos. El “tipo de segmento” también debe estar definido para cada símbolo para asegurar su uso apropiado. Todos los símbolos externos incluidos en este ejemplo son del tipo CODE.

A continuación presentamos la definición de los símbolos. La línea 10 define el símbolo CR como el código en ASCII del retorno de carro 0DH. La línea 11 define el símbolo EPROM como un segmento de tipo CODE. Recuerde que la directiva SEGMENT define únicamente lo que el símbolo es —nada más, nada menos.

La directiva RSEG ubicada en la línea 13 inicia el segmento relocalizable llamado EPROM. Las instrucciones subsecuentes, las definiciones de constantes de datos, etc., se colocarán en el segmento de código EPROM.

El programa comienza en la línea 14, en la etiqueta PRINCIPAL. La primera instrucción es una llamada a la subrutina INICIA, la cual inicializa el puerto serial del 8051. El código ensamblado bajo la columna OBJ contiene el código de operación correcto (12H para LCALL); sin embargo, los bytes 2 y 3 de la instrucción (la dirección de la subrutina) aparecen como 0000H seguidos de la letra “F”. El enlazador/localizador debe arreglar (“fix”) esto cuando los

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
 OBJECT MODULE PLACED IN PANTALLA.OBJ
 ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE PANTALLA.SRC EP

LOC	OBJ	LINE	SOURCE	
		1	\$DEBUG	
		2	\$TITLE(*** EJEMPLO CON NOTAS (MODULO PRINCIPAL) ***)	
		3	\$PAGewidth(98)	
		4	\$NOPAGING	
		5		
		6	NAME PRINCIPAL ;EL NOMBRE DEL MODULO ES "PRINCIPAL"	
		7	EXTRN CODE(INIC,CADSAL) ;DECLARACION DE SIMBOLOS EXTERNOS	
		8	EXTRN CODE(LINEAENT,LINEASAL)	
		9		
000D		10	CR EQU 0DH ;CODIGO DE RETORNO DE CARRO	
		11	EPROM SEGMENT CODE ;DEFINICION DEL SIMBOLo "EPROM"	
		12		
----		13	RSEG EPROM ;INICIA SEGMENTO DE CODIGO	
0000 120000 F		14	PRINCIPAL: CALL INIC ;INICIALIZA PUERTO SERIAL	
0003 900000 F		15	CICLO: MOV DPTR,#INTERROGA ;ENVIA INTERROGACION	
0006 120000 F		16	CALL CADSAL	
0009 120000 F		17	CALL LINEAENT ;RECIBE UNA LINEA DE COMANDOS Y	
000C 120000 F		18	CALL LINEASAL ; PRESENTALO	
000F 80F2		19	JMP CICLO ;SE REPITE	
		20		
0011 0D		21	INTERROGA: DB CR,'Escriba un comando: ',0	
0012 456E7465				
0016 72206120				
001A 636F6D6D				
001E 6106E643A				
0022 20				
0023 00				
		22	END	

SYMBOL TABLE LISTING

----- ----- -----

N A M E	T Y P E	V A L U E	A T T R I B U T E S
CR	NUMB	000DH	A
EPROM.	C SEG	0024H	REL=UNIT
INIC	C ADDR	---	EXT
LINEAENT	C ADDR	---	EXT
CICLO.	C ADDR	0003H	R SEG=EPROM
PRINCIPAL. . . .	C ADDR	0000H	R SEG=EPROM
LINEASAL	C ADDR	---	EXT
CADSAL	C ADDR	---	EXT
INTERROGA. . . .	C ADDR	0011H	R SEG=EPROM

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

FIGURA 7-8a

Ejemplo con notas: enlace de segmentos y módulos relocalizables. (a) PANTALLA.LST. (b) ES.LST. (c) EJEMPLO.M51.

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2

OBJECT MODULE PLACED IN ES.OBJ

ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE ES.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$DEBUG
		2	\$TITLE(** EJEMPLO CON NOTAS (MODULO DE SUBRUTINAS) **)
		3	\$PAGEWIDTH(98)
		4	\$NOPAGING
		5	
		6	NAME SUBRUTINAS ;NOMBRE DEL MODULO
		7	PUBLIC INIC,CARSAL,CARENT ;DECLARA SIMBOLOS PUBLICOS
		8	PUBLIC LINEAENT,LINEASAL,CADSAL
		9	
		10	;*****
		11	; DEFINE SIMBOLOS
		12	;*****
000D		13	CR EQU 0DH ;RETORNO DE CARRO
0028		14	LONGITUD EQU 40 ;BUFER DE 40 CARACTERES
		15	EPROM SEGMENT CODE ;"EPROM" ES UN SEGMENTO DE CODIGO
		16	ENCHIP SEGMENT DATA ;"ENCHIP" ES UN SEGMENTO DE DATOS
		17	
----		18	RSEG EPROM ;EMPIEZA SEGMENTO REUBICABLE DE CODIGO
		19	
		20	;*****
		21	; INICIALIZA EL PUERTO SERIAL
		22	;*****
0000 759852		23	INIC: MOV SCONM#52H ;MODO UART 8 BITS
0003 758920		24	MOV TMOD,#20H ;TEMP 1 SUMINISTRA RELOJ DE VEL. EN BAUDIOS
0006 758DF3		25	MOV TH1,#-13 ;2400 BAUDIOS
0009 D28E		26	SETB TR1 ;INICIA TEMPORIZADOR
000B 22		27	RET
		28	
		29	;*****
		30	; IMPRIME CARACTER EN ACC (NOTA: VDT DEBE CONVERTIR CR EN CR/LF)
		31	;*****
000C 3099FD		32	CARSAL: JNB TI,\$;ESPERA QUE BUFER TRANSM. SE VACIE
000F C299		33	CLR TI ;CUANDO SE VACIE, BORRA BANDERA Y
0011 F599		34	MOV SBUF,A ;ENVIA CARACTER
0013 22		35	RET
		36	
		37	;*****
		38	; INTRODUCE CARACTER EN ACC
		39	;*****
0014 3098FD		40	CARENT: JNB RI,\$;ESPERA QUE BUFER RECEP. SE LLENE
0017 C298		41	CLR RI ;CUANDO LLEGA CAR., BORRA BANDERA E
0019 E599		42	MOV A,SBUF ;INTRODUCE CARACTER EN ACC
001B 22		43	RET
		44	
		45	;*****
		46	; IMPRIME CADENA CON TERMINACION NULA
		47	;*****
001C E4		48	CADSAL: CLR A ;DPTR APUNTA A CADENA DE CARACTERES
001D 93		49	MOVC A,@A+DPTR ;OBTIENE CARACTER
001E 6006		50	JZ SALIDA ;SI BYTE NULO, TERMINA
0020 120000 F		51	CALL CARSAL ;EN CASO CONTRARIO, LO ENVIA
0023 A3		52	INC DPTR ;APUNTA AL SIGUIENTE CARACTER
0024 80F6		53	JMP CADSAL ;Y LO ENVIA TAMBien
0026 22		54	EXIT: RET

FIGURA 7-8b*continuación*

```

55
56 ;*****
57 ; INTRODUCE CARACTERES EN BUFER *
58 ;*****
0027 7800 F 59 LINEAENT: MOV R0, #BUFER ;USA R0 COMO APUNTADOR A BUFER
0029 120000 F 60 DENUEVO: CALL CARENT ;OBTIENE UN CARACTER
002C 120000 F 61 CALL CARSAL ;LO PRESENTA
002F F6 62 MOV @R0, A ;LO COLOCA EN BUFER
0030 08 63 INC R0 ;INCREMENTA APUNTADOR A BUFER
0031 B40DF5 64 CJNE A, #CR, DENUEVO ;SI NO ES CR, OBTIENE OTRO CAR.
0034 7600 65 MOV @R0, #0 ;COLOCA BYTE NULO AL FINAL
0036 22 66 RET
67
68 ;*****
69 ; IMPRIME CONTENIDO DE BUFER *
70 ;*****
0037 7800 F 71 LINEASAL: MOV R0, #BUFER ;USA R0 COMO APUNTADOR A BUFER
0039 E6 72 DENUEVO2: MOV A, @R0 ;OBTIENE CARACTER DE BUFER
003A 6006 73 JZ SALIDA2 ;SI BYTE NULO, TERMINA
003C 120000 F 74 CALL CARSAL ;EN CASO CONTRARIO, LO ENVIA
003F 08 75 INC R0 ;APUNTA AL SIG. CAR. EN BUFER
0040 80F7 76 JMP DENUEVO2 ;Y LO ENVIA TAMBIEN
0042 22 77 SALIDA2: RET
78
79 ;*****
80 ; CREA UN BUFER EN LA RAM EN CHIP *
81 ;*****
---- 82 RSEG ENCHIP ;EMPIEZA SEG. REUBIC. DE DATOS
0000 83 BUFER: DS LONGITUD ;ASIGNA RAM INTERNA COMO BUFER
84 END

```

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
DENUEVO . . .	C ADDR	0029H	R SEG=EPROM
DENUEVO2. . .	C ADDR	0039H	R SEG=EPROM
BUFER	D ADDR	0000H	R SEG=ENCHIP
CR	NUMB	000DH	A
EPROM	C SEG	0043H	REL=UNIT
SALIDA. . . .	C ADDR	0026H	R SEG=EPROM
SALIDA2. . . .	C ADDR	0042H	R SEG=EPROM
CARENT.	C ADDR	0014H	R PUB SEG=EPROM
INIC.	C ADDR	0000H	R PUB SEG=EPROM
LINEAENT. . . .	C ADDR	0027H	R PUB SEG=EPROM
LONGITUD. . . .	NUMB	0028H	A
ENCHIP.	D SEG	0028H	REL=UNIT
CARSAL.	C ADDR	000CH	R PUB SEG=EPROM
LINEASAL. . . .	C ADDR	0037H	R PUB SEG=EPROM
CADSAL.	C ADDR	001CH	R PUB SEG=EPROM
RI.	B ADDR	0098H.0	A
SBUF.	D ADDR	0099H	A
SCON.	D ADDR	0098H	A
SUBRUTINAS	-----	-----	-----
TH1	D ADDR	008DH	A
TI.	B ADDR	0098H.1	A
TMOD.	D ADDR	0089H	A
TR1	B ADDR	0088H.6	A

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

FIGURA 7-8c
continuación

DATE : 03/17/91
 DOS 3.31 (038-N) MCS-51 RELOCATOR AND LINKER V3.0, INVOKED BY:
 C:\ASM51\RL51.EXE PANTALLA.OBJ,ES.OBJ TO EJEMPLO CODE (EPROM(8000H))DATA(ENCHIP(30H
 >>))

INPUT MODULES INCLUDED
 PANTALLA.OBJ(PRINCIPAL)
 ES.OBJ(SUBRUTINAS)

LINK MAP FOR EJEMPLO(PRINCIPAL)

TYPE	BASE	LENGTH	RELOCATION	SEGMENT NAME
---	---	-----	-----	-----
REG	0000H	0008H		"REG BANK 0"
	0008H	0028H		*** GAP ***
DATA	0030H	0028H	UNIT	ENCHIP
	0000H	8000H		*** GAP ***
CODE	8000H	0067H	UNIT	EPROM

SYMBOL TABLE FOR EJEMPLO(PRINCIPAL)

VALUE	TYPE	NAME
----	---	----
-----	MODULE	PRINCIPAL
N:000DH	SYMBOL	CR
C:8000H	SEGMENT	EPROM
C:8003H	SYMBOL	CICLO
C:8000H	SYMBOL	PRINCIPAL
C:8011H	SYMBOL	INTERROGA
-----	ENDMOD	PRINCIPAL
-----	MODULE	SUBRUTINAS
C:804DH	SYMBOL	DENUEVO
C:805DH	SYMBOL	DENUEVO2
D:0030H	SYMBOL	BUFER
N:000DH	SYMBOL	CR
C:8000H	SEGMENT	EPROM
C:804AH	SYMBOL	SALIDA
C:8066H	SYMBOL	SALIDA2
C:8038H	PUBLIC	CARENT
C:8024H	PUBLIC	INIC
C:8048H	PUBLIC	LINEAENT
N:0028H	SYMBOL	LONGITUD
D:0030H	SEGMENT	ENCHIP
C:8030H	PUBLIC	CARSAL
C:805BH	PUBLIC	LINEASAL
C:8040H	PUBLIC	CADSAL
B:0098H	SYMBOL	RI
D:0099H	SYMBOL	SBUF
D:0098H	SYMBOL	SCON
D:008DH	SYMBOL	TH1
B:0098H.1	SYMBOL	T1
D:0089H	SYMBOL	TMOD
B:0088H.6	SYMBOL	TR1
-----	ENDMOD	SUBRUTINAS

FIGURA 7-8d

continuación

módulos del programa se enlazan y las direcciones para los segmentos relocalizables se establecen. Observe que la dirección bajo la columna LOC también aparece como 0000H. Debido a que el segmento EPROM es relocalizable, no podemos conocer el tiempo de ensamblado donde iniciará el segmento. Todos los segmentos relocalizables mostrarán 0000H como la dirección inicial del archivo de listado.

El resto de las instrucciones del programa está en las líneas 15 a 19. Un mensaje de interrogación es enviado a la terminal de video cargando al DPTR con la dirección inicial de la interrogación y llamando a la subrutina CADSAL. Ya que las subrutinas CADSAL, LINEAENT y LINEASAL no están definidas en PANTALLA.SRC, sólo podemos intentar adivinar sobre su operación a partir de su nombre y de las líneas de comentarios.

La interrogación es una cadena ASCII terminada por un elemento nulo, y se coloca en el segmento de código EPROM utilizando la directiva DB (definición de byte) en la línea 21. Debido a que los bytes de la interrogación son constantes (en otras palabras, no cambian), es correcto colocarlos en la memoria para código (a pesar de que son bytes de datos). La interrogación comienza con un retorno de carro para asegurar que se presentará en una nueva línea. (En este ejemplo, asumimos que la terminal de video convierte CR a CR/LF).

Todos los símbolos y las etiquetas en PANTALLA.SRC aparecen en la tabla de símbolos al final de PANTALLA.LST. Debido a que el segmento EPROM es relocalizable y las subrutinas son externas, la columna VALUE no es de mucho valor. Sin embargo, el valor del símbolo EPROM representa la longitud del segmento, el cual es de 24H o 36 bytes en este caso.

7.8.2 ES.LST

La figura 7-8b muestra el contenido del archivo ES.LST —el archivo que contiene las subrutinas de entrada/salida—. Este módulo es llamado SUBRUTINAS en la línea 6. Las líneas 7 y 8 declaran todos los nombres de las subrutinas como símbolos PUBLIC. Ello permite que estos símbolos estén disponibles para otros módulos. Observe que todas las subrutinas se hacen públicas aunque sólo cuatro fueron utilizadas en el módulo PRINCIPAL. Tal vez, conforme el programa crezca, se agregarán otros módulos que puedan utilizar estas subrutinas. Por lo tanto, todas se hacen públicas.

Las líneas 13 a 16 definen varios símbolos. De nuevo, EPROM se utiliza como el nombre del segmento de código. Otro segmento se utiliza en este módulo. ENCHIP se define en la línea 17 como un segmento de datos interno.

Cada subrutina se escribe por turno, comenzando en la línea 20. El bloque de comentarios que inicia cada una de las subrutinas es deliberadamente breve en este ejemplo; sin embargo, a menudo se da una descripción más detallada de la subrutina. Resulta muy útil proporcionar, por ejemplo, las condiciones de entrada y salida para cada subrutina.

Después de la última subrutina, se crea un búfer en memoria RAM interna utilizando el segmento ENCHIP. El segmento se inicia con RSEG (línea 82) y el búfer se crea utilizando la directiva DS (definición de almacenamiento) en la línea 83. El tamaño del búfer se asigna al símbolo LONGITUD “igualado” en la parte superior del programa (línea 14) a 40. La ubicación de la definición del símbolo LONGITUD en el archivo fuente y de la instancia del segmento ENCHIP es meramente una cuestión de gusto. Ambos pueden ubicarse también justo antes o después de la subrutina LINEAENT, en donde se utilizan.

Así como el segmento EPROM, el segmento ENCHIP tiene una dirección inicial de 0000H bajo la columna LOC en la línea 83. De nuevo, la ubicación actual del segmento ENCHIP no será determinada hasta el tiempo de enlazamiento (vea líneas abajo). La letra “F” aparece en numerosas ubicaciones en ES.LST. Cada línea identificada así contiene una instrucción que utiliza un símbolo cuyo valor no puede determinarse al tiempo de ensamblado. Los ceros colocados

en el archivo objeto en estas ubicaciones pueden ser reemplazados con valores “absolutos” por el enlazador/localizador.

7.8.3 EJEMPLO.M51

La figura 7-8c muestra el contenido del archivo EJEMPLO.M51 creado por el programa enlazador/localizador RL51. La línea de invocación se repite cerca del inicio del archivo EJEMPLO.M51 y se debe examinar cuidadosamente. La presentamos de nuevo (sin el directorio completo):

```
RL51 PANTALLA.OBJ,ES.OBJ TO EJEMPLO CODE (EPROM (8000H)) &
DATA (ENCHIP (30H))
```

Después del comando se listan los módulos de objetos separados por comas en el orden a ser enlazados. Luego de la lista de entrada, se especifica el control opcional TO EJEMPLO, el cual proporciona el nombre para el módulo de objetos absoluto creado por el RL51. Si el nombre se omite, se utiliza el del primer archivo incluido en la lista de entrada (sin la extensión). El archivo de listado, en este ejemplo, toma automáticamente el nombre de EJEMPLO.M51. Finalmente, los controles de ubicación CODE y DATA especifican los nombres de los segmentos del tipo asociado y la dirección absoluta en que empezará el segmento. En este ejemplo el segmento de código EPROM comienza en la dirección 8000H y el segmento de datos ENCHIP en la dirección de byte 30H de la memoria RAM interna del 8051.

Después de repetirse la línea de invocación, el archivo EJEMPLO.M51 contiene una lista de los módulos de entrada incluidos por el RL51. En este ejemplo sólo se listan dos archivos (PANTALLA.OBJ y ES.OBJ) y dos módulos (PRINCIPAL y SUBRUTINAS). Si la directiva NAME no se ha utilizado en los archivos fuente, los nombres de los módulos serán los mismos que los de los archivos.

El mapa de enlace aparece a continuación. Se identifica a ambos segmentos de ENCHIP y EPROM y se proporciona la dirección de inicio y el tamaño (en hexadecimal) de cada segmento. ENCHIP es identificado como un segmento de datos comenzando en la dirección 30H y de tamaño igual a 28H (40) bytes. EPROM está identificado como un segmento de código comenzando en la dirección 8000H y con tamaño de 67H (103) bytes.

Por último, el archivo EJEMPLO.M51 contiene una tabla de símbolos. Se listan todos los símbolos (incluyendo las etiquetas) utilizados en el programa, ordenados con base en los módulos. Todos los valores son “absolutos”. Recordemos que la tabla de símbolos en el archivo .M51 sólo puede ser creada si colocamos el control del ensamblador \$DEBUG al inicio de cada archivo fuente. La dirección de la subrutina INIC (la cual señalamos anteriormente que estaba ausente en el archivo PANTALLA.LST) se identifica como 8024H en el módulo SUBRUTINAS. Esta dirección se sustituye por la dirección de código en cualquier módulo de objetos utilizando la instrucción CALL INIC, como vimos anteriormente en el módulo PRINCIPAL. Es importante saber el valor absoluto de las etiquetas cuando depuramos un programa. A menudo podemos hacer una actualización cuando encontramos errores si modificamos los bytes del programa y lo volvemos a ejecutar. Si la actualización corrige el error, hacemos el cambio adecuado en el programa fuente.

7.9 MACROS

Regresamos al ASM51 para estudiar el tema final de este capítulo. La facilidad de procesamiento de macros (MPL) del ASM51 es una facilidad de “reemplazo de cadenas”. Las macros permiten definir secciones de código utilizadas con frecuencia una sola vez mediante un simple mnemónico

y utilizarlas en cualquier parte del programa insertando el mnemónico. La programación utilizando macros es una poderosa extensión de las técnicas que hemos descrito hasta ahora. Las macros se pueden definir en cualquier parte de un programa fuente y ser utilizadas subsecuentemente como cualquier otra instrucción. La sintaxis para la definición de una macro es

```
%*DEFINE (patrón_de_llamada) (cuerpo_de_macro)
```

Una vez definido, el patrón es como un mnemónico: puede utilizarse como cualquier otra instrucción del lenguaje ensamblador si lo colocamos en el campo de mnemónico de un programa. Las macros se construyen a partir de instrucciones “reales” precediéndolas con un signo de porcentaje, “%”. Al ensamblar el programa fuente, el patrón de llamada sustituye a todo lo que haya dentro del cuerpo de la macro, carácter por carácter. El aspecto místico de las macros en cierta manera no tiene base. Las macros proveen una manera simple de reemplazar patrones de instrucciones complicados con mnemónicos rudimentarios y fáciles de recordar. La sustitución se hace, y lo repetimos, carácter por carácter —nada más, nada menos.

Por ejemplo, la siguiente definición de una macro aparece al inicio de un archivo fuente,

```
%*DEFINE (PUSH_DPTR)
          (PUSH DPH
           PUSH DPL
          )
```

entonces la instrucción

```
%PUSH_DPTR
```

aparecerá en el archivo .LST como

```
PUSH DPH
PUSH DPL
```

El ejemplo anterior es una macro típica. Debido a que las instrucciones de pila del 8051 operan sólo en direcciones directas, el apuntador de datos requiere de dos instrucciones PUSH para almacenarla. Podemos crear una macro similar para recuperar (POP) el apuntador de datos.

Existen varias ventajas distintivas cuando utilizamos macros:

- Un programa fuente que utiliza macros es más legible, ya que el mnemónico de la macro es a menudo más indicativo de la operación a realizar que las instrucciones de lenguaje ensamblador equivalentes.
- El programa fuente es más corto y requiere de menos mecanografía.
- Utilizando macros, se reduce el número de errores de programación.
- Utilizando macros, se libera al programador de lidiar con detalles de bajo nivel.

Los últimos dos puntos están relacionados. Una vez que se ha escrito y depurado una macro, se puede usar ampliamente sin preocupaciones de haber cometido errores. En el ejemplo PUSH_DPTR anterior, si se utilizan las instrucciones PUSH y POP en lugar de las macros para almacenar y recuperar, el programador puede inadvertidamente revertir el orden del almacenamiento o de la recuperación. (¿Fue el byte superior o el inferior el que se almacenó primero?) Esto causará un error de programación. Si utilizamos macros, sin embargo, trabajamos en los detalles sólo una vez —cuando escribimos la macro— y la macro se utiliza ampliamente después, sin preocuparnos de los errores.

Ya que el reemplazo es de carácter por carácter, la definición de una macro se debe construir cuidadosamente con retornos de carro, tabulaciones, etc., para asegurar la alineación

apropiada de las instrucciones de la macro con el resto del programa escrito en lenguaje ensamblador. Requerimos de un método de tanteo.

Existen características avanzadas de la facilidad de procesamiento de macros del ASM51 que permiten la transferencia de parámetros, etiquetas locales, operaciones de repetición, control del flujo de ensamblado, y así sucesivamente. A continuación hablaremos sobre esto.

7.9.1 Transferencia de parámetros

Una macro con parámetros transferidos del programa principal tiene el siguiente formato modificado:

```
%*DEFINE (nombre_de_macro (lista_de_parámetros)) (cuerpo_de_macro)
```

Por ejemplo, si definimos la siguiente macro,

```
%*DEFINE (CMPA# (VALUE))
          (CJNE A, #VALUE, $ + 3
           )
```

entonces la llamada a la macro

```
%CMPA# (20H)
```

se expandirá a la siguiente instrucción en el archivo .LST:

```
CJNE A, #20H, $ + 3
```

Aunque el 8051 no tiene una instrucción de “compara el acumulador”, podemos crear una fácilmente con la instrucción CJNE y la expresión “\$ + 3” (la siguiente instrucción) como destino para el salto condicional. El mnemónico CMPA# puede ser más fácil de recordar para muchos programadores. Además, el uso de la macro libera al programador de recordar detalles de notación, tales como “\$ + 3”.

Desarrollemos otro ejemplo. Sería buena idea que el 8051 tuviera instrucciones tales como

```
SALTA SI EL ACUMULADOR ES MAYOR A X
SALTA SI EL ACUMULADOR ES MAYOR O IGUAL A X
SALTA SI EL ACUMULADOR ES MENOR A X
SALTA SI EL ACUMULADOR ES MENOR O IGUAL A X
```

pero no cuenta con ellas. Estas operaciones pueden crearse utilizando CJNE seguida de JC o JNC, pero los detalles son un poco más complicados. Supongamos, por ejemplo, que deseamos saltar a la etiqueta MAYOR_A si el acumulador contiene un código en ASCII mayor que “Z” (5AH). La siguiente secuencia de instrucciones funcionaría:

```
CJNE A, #5BH, $+3
JNC MAYOR_A
```

La instrucción CJNE resta 5BH (en otras palabras, “Z” + 1) del contenido de A y activa o limpia la bandera de acarreo adecuadamente. CJNE deja C = 1 para los valores en el acumulador de 00H hasta, e incluyendo, 5AH. (Nota: 5AH – 5BH < 0, y por lo tanto C = 1; pero 5BH – 5BH = 0, en consecuencia C = 0.) Si saltamos a MAYOR_A en la condición de “no hay acarreo”, saltaremos correctamente para los valores del acumulador 5BH, 5CH, 5DH, y así

sucesivamente, hasta llegar a FFH. Una vez resueltos estos detalles, se pueden simplificar mediante la invención de un mnemónico apropiado, definiendo una macro y utilizándola en lugar de la secuencia de instrucciones correspondiente. He aquí la definición para una macro de “salta si es mayor que”:

```
%*DEFINE (JGT (VALUE, LABEL))
          (CJNE A, #VALUE+1, $+3 ;JGT
           JNC    %LABEL
           )
```

Para verificar que el acumulador contiene un código ASCII mayor a “Z”, como vimos, podemos llamar a la macro como sigue

```
%JGT ('Z', MAYOR_A)
```

El ASM51 expandirá esto a

```
CJNE A, #5BH, $+3      ;JGT
JNC    MAYOR_A
```

La macro JGT es un ejemplo excelente del uso pertinente y poderoso de las macros. Al emplear macros, el programador se beneficia con la utilización de mnemónicos significativos y evita detalles sucios y potencialmente llenos de errores de programación.

7.9.2 Etiquetas locales

Podemos utilizar etiquetas locales dentro de una macro aplicando el siguiente formato:

```
%*DEFINE (nombre_de_macro [(lista_de_parámetros)])
          [LOCAL lista_de_etiquetas_locales] (cuerpo_de_macro)
```

Por ejemplo, la siguiente definición de una macro

```
%*DEFINE (DEC_DPTR) LOCAL SKIP
          (DEC DPL                  ;DISMINUYE APUNTADOR DE DATOS
           MOV A,DPL
           CJNE A,#0FFH,%IGNORA
           DEC DPH
%IGNORA:      )
```

sería llamada como

```
%DEC_DPTR
```

y el ASM51 la expandiría a

```
DEC   DPL                  ;DECREMENTA APUNTADOR DE DATOS
MOV   A,DPL
CJNE A, #0FFH, IGNORA00
DEC   DPH
IGNORA00:
```

Observe que una etiqueta local generalmente no causará conflicto con la misma etiqueta utilizada en algún otro lugar del programa fuente, ya que el ASM51 agrega un código numérico a la etiqueta local cuando la macro se expande. Además, el siguiente uso de la misma etiqueta local recibe el siguiente código numérico, y así sucesivamente.

La macro anterior tiene un “efecto secundario” potencial. El acumulador se utiliza como lugar de almacenamiento temporal para DPL. Si la macro fuese utilizada dentro de una sección de código que emplea A para otro propósito, se perdería el valor en A. Este efecto secundario probablemente representa un error en el programa. La definición de la macro puede salvaguardarse contra esto cuando se almacena A en la pila. Presentamos a continuación una definición alternativa para la macro DEC_DPTR.

```
%*DEFINE (DEC_DPTR) LOCAL IGNORA
    (PUSHACC
        DEC DPL           ;DISMINUYE APUNTADOR DE DATOS
        MOV A,DPL
        CJNE A,#0FFH,% IGNORA
        DEC DPH
    % IGNORA:   POP ACC
    )
```

7.9.3 Operaciones de repetición

Esta es una de varias macros incorporadas (predefinidas). El formato es

```
%REPEAT (expresión) (texto)
```

Por ejemplo, para llenar un bloque de memoria con 100 instrucciones NOP,

```
%REPEAT (100)
(NOP
)
```

7.9.4 Operaciones de flujo de control

La definición de una macro para el flujo de control del ASM51 proporciona el ensamblado condicional de secciones de código. El formato es

```
%IF(expresión) THEN (texto_balanceado)
[ELSE (texto_balanceado) ]FI
```

Por ejemplo,

```
INTERNO EQU 1      ;1 = CONTROLADORES DE E/S SERIAL DEL 8051
;0 = CONTROLADORES DE E/S SERIAL DEL 8251
.
.
%IF (INTERNO) THEN
(CARENT: .          ;CONTROLADORES DEL 8051
.
.
CARSAL: .          .
.
```

```

) ELSE
(CARENT : . ; CONTROLADORES DEL 8051
.
.
CARSAL : .
.
)

```

En este ejemplo, se le da el valor 1 al símbolo INTERNO para seleccionar las subrutinas de E/S para el puerto serial del 8051, o el valor 0 para seleccionar las subrutinas de E/S para un UART externo, el 8251 en este caso. La macro IF causa que el ASM51 ensamble un conjunto de controladores e ignore el otro. En alguna otra parte del programa, las subrutinas CARENT y CARSAL se utilizan sin consideración para la configuración particular del hardware. La subrutina correcta se ejecuta siempre y cuando el programa sea ensamblado con el valor correcto para INTERNO.

RESUMEN

En este capítulo presentamos conceptos importantes de la programación en lenguaje ensamblador del 8051, incluyendo formatos de programas estándar y el uso de las directivas del ensamblador y las macros para ayudar a que el programa sea más legible, compacto y ordenado.

Conforme un programa aumenta su tamaño, podemos preguntarnos si existen otras maneras de escribir programas en el 8051 de modo más estructurado y parecido a nuestro lenguaje cotidiano. En el siguiente capítulo analizaremos tal alternativa, la cual constituye el lenguaje C del 8051.

PROBLEMAS

- 7.1** Vuelva a escribir las siguientes instrucciones expresando al operando en binario.

```

MOV A, #255
MOV A, #11Q
MOV A, #1AH
MOV A, #'A'

```

- 7.2** Vuelva a escribir las siguientes instrucciones expresando al operando en hexadecimal.

```

MOV A, #255
MOV A, #-3
MOV A, #'z'
MOV A, #'33Q
MOV A, #'$'
MOV A, #64

```

- 7.3** ¿Qué está mal en la codificación de la siguiente instrucción?

```
ORL 80H, #FOH
```

- 7.4** Identifique el error en los siguientes símbolos.

```
?byte.bit
@ADIOS
1A_BANDERA
MI_PROGRAMA
```

- 7.5** Vuelva a escribir las siguientes instrucciones con la expresión evaluada como una constante hexadecimal de 16 bits.

```
MOV DPTR, # '0' EQ 48
MOV DPTR, # HIGH'AB'
MOV DPTR, #-1
MOV DPTR, #NOT (257 MOD 256)
```

- 7.6** Vuelva a escribir cada una de las siguientes instrucciones especificando la ubicación de la fuente como una dirección hexadecimal, en lugar de especificarla en su forma simbólica.

```
MOV A, PSW
MOV A, P0
MOV A, DPH
MOV A, TLO
MOV A, IP
MOV A, TMOD
```

- 7.7** Vuelva a escribir cada una de las siguientes instrucciones especificando la ubicación de la fuente en forma simbólica, en lugar de especificarla como una dirección hexadecimal.

```
MOV A, 0B0H
MOV A, 99H
MOV A, 82H
MOV A, 8BH
MOV A, 0A8H
MOV A, 0D0H
```

- 7.8** ¿Cuáles son los “tipos de segmento” definidos por el ASM51 para el 8051, y qué espacios de memoria representan?

- 7.9** ¿Cómo podemos definir y seleccionar un segmento relocalizable en la memoria externa para datos y crear un búfer de 100 bytes? (Use el nombre “FUERADECHIP” para el segmento y “XBUFER” para el búfer).

- 7.10** Cierta aplicación requiere de cinco bits de estatus (BANDERA1 a BANDERA5). ¿Cómo podemos definir un búfer de 5 bits en el segmento BIT absoluto comenzando en la dirección de bit 0811? ¿En qué dirección de byte residirán estos bits?

- 7.11** ¿Cuáles son dos buenas razones para hacer uso generoso de la directiva EQU en los programas escritos en lenguaje ensamblador?

- 7.12** ¿Cuál es la diferencia entre las directivas DB y DW?

- 7.13** ¿Cuáles son las asignaciones de memoria para las siguientes directivas del ensamblador?

```
ORG 0FH
DW $ SHL 4
DB 65535
DW '0'
```

- 7.14** ¿Cuál directiva se utiliza para seleccionar un segmento de código absoluto?
- 7.15** Un archivo llamado “ASCII” contiene 33 directivas de igualdad, una para cada código de control:

```

NULO    EQU  00H      ; BYTE NULO
IDE     EQU  01H      ; INICIO DE ENCABEZADO
.
.
.
SU      EQU  7FH      ; SEPARADOR DE UNIDAD
BOR     EQU  7FH      ; BORRAR

```

¿Cómo se pueden hacer conocidas estas definiciones en otro archivo —un programa fuente— sin tener que agregar las directivas de igualdad a ese archivo?

- 7.16** Para que la impresión de un archivo de listado se vea bien, sugerimos comenzar cada subrutina al inicio de una página. ¿Cómo podemos lograr esto?
- 7.17** Escriba la definición para una macro que pueda ser utilizada para llenar un bloque de memoria externa para datos con una constante. Transfiera a la macro la dirección inicial, el tamaño y la constante como parámetros.
- 7.18** Escriba la definición para las siguientes macros:

JGE – salta a ETIQUETA si el acumulador es mayor o igual a VALOR
 JLT – salta a ETIQUETA si el acumulador es menor a VALOR
 JLE – salta a ETIQUETA si el acumulador es menor o igual a VALOR
 JOR – salta a ETIQUETA si el acumulador está fuera del rango INFERIOR y SUPERIOR

- 7.19** Escriba la definición para una macro llamada CJNE_DPTR que saltará a ETIQUETA si el apuntador de datos no contiene VALOR. Defina la macro en forma tal que permanezca intacto el contenido de todos los registros y de las ubicaciones de memoria.

Programación en C del 8051

8.1 INTRODUCCIÓN

A lo largo de los capítulos anteriores hemos analizado cómo comunicarnos con el 8051 mediante el uso del lenguaje ensamblador. De hecho, existe otra manera de establecer comunicación con el 8051. Esto se conoce como lenguaje C del 8051, el cual es una elección que se prefiere a menudo cuando la complejidad de un programa aumenta en forma considerable. En este capítulo introducimos el lenguaje C del 8051 como alternativa para la programación en lenguaje ensamblador. Como programador, usted decidirá cuál lenguaje prefiere utilizar. Los factores que por lo común influyen en tal decisión son: la velocidad deseada, el tamaño del código, y la facilidad de programación. Ya que en este capítulo intentamos presentar los conceptos básicos sobre la programación en C del 8051, asumiremos que el lector ya está familiarizado con la programación en C convencional, lo cual es cierto la mayor parte de las veces debido a la popularidad y el amplio uso del lenguaje C.

8.2 VENTAJAS Y DESVENTAJAS DEL LENGUAJE C DEL 8051

Las ventajas de la programación en C del 8051, comparado con el lenguaje ensamblador, son:

- Ofrece todos los beneficios de los lenguajes de alto nivel y de programación estructurada tales como el lenguaje C, incluyendo la facilidad de codificación de subrutinas y otros conceptos que veremos con más detalle en la sección 9.2
- A menudo libera al programador de lidiar con los detalles de hardware que el compilador maneja a favor del programador
- Es más fácil de codificar, especialmente en programas largos y complejos
- Produce códigos fuente de programas más legibles

No obstante, el lenguaje C del 8051 también sufre de las siguientes desventajas debido a su similitud con el lenguaje C convencional:

- Posee las desventajas de los lenguajes de alto nivel y de programación estructurada, tal como se menciona en la sección 9.2
- A menudo genera códigos máquina extensos
- El programador tiene menor control y poca habilidad para interactuar directamente con el hardware

Para hacer una comparación entre el lenguaje C del 8051 y el lenguaje ensamblador, considere la solución al ejemplo 4.5, el cual se escribe a continuación en lenguaje C del 8051:

```
sbit bit_puerto = P1^0; /* Utiliza la variable bit_puerto para
                           referirnos a P1.0 */

main( )
{
    TMOD = 1;
    while (1)
    {
        TH0 = 0xFE;
        TL0 = 0xC;
        TR0 = 1;
        while (TF0 != 1);
        TR0 = 0;
        TF0 = 0;
        bit_puerto = !(P1^0);
    }
}
```

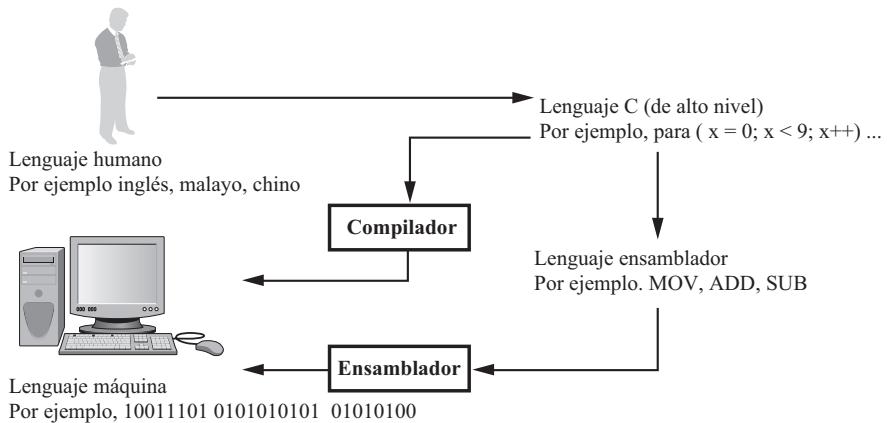
Observe que ambas soluciones para el ejemplo 4.5, tanto en lenguaje ensamblador como en lenguaje C, requieren casi del mismo número de líneas de código. Sin embargo, la diferencia reside en la legibilidad de estos programas. La versión en C se asemeja más al lenguaje humano que el lenguaje ensamblador y, por lo tanto, es más legible. Esto a menudo facilita los esfuerzos del programador para escribir programas aún más complejos. La versión en lenguaje ensamblador está más relacionada con el código máquina y, aunque es menos legible, generalmente resulta en un código máquina más compacto. Así, en este ejemplo, el código máquina resultante de la versión en lenguaje ensamblador es de 83 bytes mientras que la versión en lenguaje C requiere de 149 bytes, ¡lo cual representa un incremento del 79.5 por ciento!

La alternativa del programador de utilizar ya sea el lenguaje de alto nivel C o el lenguaje ensamblador para comunicarse con el 8051, que utiliza el lenguaje máquina, presenta un interesante dilema, como se muestra en la figura 8-1.

8.3 COMPILADORES PARA C DEL 8051

Como podemos observar en la figura 8-1, se necesita un compilador para convertir los programas escritos en lenguaje C del 8051 al lenguaje máquina, tal y como necesitamos de un ensamblador en el caso de programas escritos en lenguaje ensamblador. En esencia, un compilador actúa igual que un ensamblador, excepto que es más complejo pues la diferencia entre el lenguaje C y el lenguaje máquina es más grande que la diferencia entre lenguaje ensamblador y lenguaje máquina. Es por ello que un compilador enfrenta una tarea más grande para sobrelevar dicha diferencia.

En la actualidad existen varios compiladores para C del 8051, los cuales ofrecen casi las mismas funciones. Hemos compilado y probado todos nuestros ejemplos y programas utilizando el

**FIGURA 8-1**

Conversión entre lenguajes humano, de alto nivel, ensamblador y máquina

IDE μ Vision 2 de Keil Software,¹ un ambiente integrado para desarrollar programas en el 8051, que incluye su propio compilador multiplataforma para C, llamado C51 (consulte el apéndice H para obtener una guía sobre la utilización del IDE μ Vision 2). Un compilador multiplataforma es un compilador que, por lo general, se ejecuta sobre una plataforma tal como las computadoras personales compatibles con IBM, pero que es utilizado para compilar programas en códigos que pueden ser ejecutados en otras plataformas, tales como el 8051.

8.4 TIPOS DE DATOS

El lenguaje C del 8051 es bastante similar al lenguaje C convencional, excepto que se le han hecho diversas extensiones y adaptaciones para volverlo más apropiado para el ambiente de programación del 8051. La primera preocupación del programador en C del 8051 tiene que ver con los tipos de datos. Recuerde que un tipo de datos es algo que utilizamos para almacenar datos. Los lectores estarán familiarizados con los tipos de datos básicos para C, tales como int, char y float, los cuales se utilizan para crear variables que almacenan enteros, caracteres o números de punto flotante. El lenguaje C del 8051 soporta todos los tipos de datos básicos formulados para C, además de algunos adicionales destinados a ser utilizados específicamente con el 8051.

La tabla 8-1 presenta una lista de los tipos de datos más utilizados en el lenguaje C del 8051. Los que aparecen en negritas son extensiones específicas para el 8051. El tipo de datos **bit** puede utilizarse para declarar variables que residen en las ubicaciones direccionables por bit del 8051 (principalmente las ubicaciones de byte 20H a 2FH o las de bit 00H a 7FH). Es evidente que estas variables bit sólo pueden almacenar valores de bit (0 o 1). Por ejemplo, la siguiente instrucción en C:

```
bit bandera = 0;
```

declara una variable **bit** llamada bandera y la inicializa con 0.

El tipo de datos **sbit** es, en cierta manera, similar al tipo de datos **bit**, salvo que por lo general se utiliza para declarar variables de 1 bit que residen en los registros de función especial (SFR). Por ejemplo:

```
sbit P = 0xD0;
```

¹Keil Software, Inc., 1501 10th Street, Suite 110, Plano, TX 75074. Sitio web: <http://www.keil.com>

TABLA 8-1

Tipos de datos utilizados en el lenguaje C del 8051

Tipo de datos	Bits	Bytes	Rango de valores
bit	1		0 a 1
signed char	8	1	-128 a +127
unsigned char	8	1	0 a 255
enum	16	2	-32768 a +32767
signed short	16	2	-32768 a +32767
unsigned short	16	2	0 a 65535
signed int	16	2	-32768 a +32767
unsigned int	16	2	0 a 65535
signed long	32	4	-2,147,483,648 a +2,147,483,647
unsigned long	32	4	0 a 4,294,967,295
float	32	4	$\pm 1.175494E-38$ a $\pm 3.402823E+38$
sbit	1		0 a 1
sfr	8	1	0 a 255
sfr16	16	2	0 a 65535

declara la variable **sbit** P y especifica que se refiere a la dirección de bit D0H, la cual es en realidad el bit menos significativo del SFR de la PSW. Observe la diferencia en el uso del operador de asignación ('='). En el contexto de las declaraciones **sbit**, esto indica en qué dirección reside la variable, mientras que en las declaraciones **bit**, el operador se utiliza para especificar el valor inicial de la variable **bit**.

Podríamos utilizar, además de la asignación directa de la dirección de bit a una variable **sbit**, una variable **sfr** previamente definida como dirección base, y asignar nuestra variable **sbit** para referirnos a cierto bit localizado dentro de esa variable **sfr**. Por ejemplo:

```
sfr PSW = 0xD0;
sbit P = PSW^0;
```

Esto declara una variable **sfr** llamada PSW, la cual se refiere a la dirección de byte D0H y después la utiliza como la dirección base para hacer referencia a su bit menos significativo (bit 0). Esta variable se asigna después a la variable **sbit**, P. Es con este propósito que utilizamos el símbolo (^), para especificar la posición de bit 0 de la PSW.

Una tercera alternativa utiliza una dirección de byte constante como dirección base dentro de la cual se hace referencia a cierto bit. Para ilustrar esto, podemos reemplazar las dos instrucciones anteriores con la instrucción siguiente:

```
sbit P = 0xD0 ^ 0;
```

Por otra parte, el tipo de datos **sfr** se utiliza para declarar variables de byte (8 bits) asociadas con los SFR. La instrucción:

```
sfr IE = 0xA8;
```

declara una variable **sfr** llamada IE que reside en la dirección de byte A8H. Recuerde que esta dirección es donde se ubica el SFR de habilitación de interrupciones (IE); por lo tanto, el tipo de datos **sfr** es sólo un medio por el cual se nos permite asignar nombres a los SFR para que sean más fáciles de recordar.

El tipo de datos **sfr16** es muy similar al tipo de datos **sfr**, a excepción de que mientras el tipo de datos **sfr** se utiliza para SFR de 8 bits, **sfr16** se aplica a SFR de 16 bits. Por ejemplo, la siguiente instrucción:

```
sfr16 DPTR = 0x82;
```

declara una variable de 16 bits llamada DPTR, cuya dirección de byte inferior está ubicada en 82H. Si analizamos la arquitectura del 8051, encontraremos que ésta es la dirección del SFR DPL, así que, una vez más, el tipo de datos **sfr16** ayuda a referirnos más fácilmente a los SFR por su nombre en lugar de emplear su dirección. Sólo queda mencionar una cosa. Cuando declaremos variables **sbit**, **sfr**, o **sfr16**, debemos recordar hacerlo fuera del bloque principal “main”, de lo contrario se producirá un error.

No obstante, es un hecho que todos los SFR del 8051, incluyendo los bits individuales de bandera, estado y control en los SFR direccionables por bit, ya se han declarado en un archivo de cabecera llamado **reg51.h**, el cual se incluye en la mayoría de los compiladores para C del 8051. La figura 8-2 presenta el contenido del archivo reg51.h. Mediante el archivo reg51.h podemos referirnos, por ejemplo, al registro de habilitación de interrupciones simplemente con el término IE, en lugar de tener que especificar la dirección A8H, y al apuntador de datos como DPTR en lugar de 82H. Todo esto hace que los programas en C del 8051 sean más legibles y manejables para los humanos.

```
/*-----
REG51.H
Header file for generic 80C51 and 80C31 microcontroller.
Copyright (c) 1988-2001 Keil Elektronik GmbH and Keil Software,
Inc.
All rights reserved.
-----*/
/* BYTE Register */
sfr P0      = 0x80;
sfr P1      = 0x90;
sfr P2      = 0xA0;
sfr P3      = 0xB0;
sfr PSW     = 0xD0;
sfr ACC     = 0xE0;
sfr B       = 0xF0;
sfr SP      = 0x81;
sfr DPL     = 0x82;
sfr DPH     = 0x83;
sfr PCON    = 0x87;
sfr TCON    = 0x88;
sfr TMOD   = 0x89;
sfr TL0     = 0x8A;
sfr TL1     = 0x8B;
sfr TH0     = 0x8C;
sfr TH1     = 0x8D;
sfr IE      = 0xA8;
sfr IP      = 0xB8;
sfr SCON    = 0x98;
sfr SBUF   = 0x99;
```

FIGURA 8-2a

Listado del archivo reg51.h

```

/* BIT Register */
/* PSW */
sbit CY    = 0xD7;
sbit AC    = 0xD6;
sbit F0    = 0xD5;
sbit RS1   = 0xD4;
sbit RS0   = 0xD3;
sbit OV    = 0xD2;
sbit P     = 0xD0;
/* TCON */
sbit TF1   = 0x8F;
sbit TR1   = 0x8E;
sbit TF0   = 0x8D;
sbit TR0   = 0x8C;
sbit IE1   = 0x8B;
sbit IT1   = 0x8A;
sbit IE0   = 0x89;
sbit IT0   = 0x88;

/* IE */
sbit EA    = 0xAF;
sbit ES    = 0xAC;
sbit ET1   = 0xAB;
sbit EX1   = 0xAA;
sbit ET0   = 0xA9;
sbit EX0   = 0xA8;

/* IP */
sbit PS    = 0xBC;
sbit PT1   = 0xBB;
sbit PX1   = 0xBA;
sbit PT0   = 0xB9;
sbit PX0   = 0xB8;

/* P3 */
sbit RD    = 0xB7;
sbit WR    = 0xB6;
sbit T1    = 0xB5;
sbit T0    = 0xB4;
sbit INT1  = 0xB3;
sbit INT0  = 0xB2;
sbit TXD   = 0xB1;
sbit RXD   = 0xB0;

/* SCON */
sbit SM0   = 0x9F;
sbit SM1   = 0x9E;
sbit SM2   = 0x9D;
sbit REN   = 0x9C;
sbit TB8   = 0x9B;
sbit RB8   = 0x9A;
sbit TI    = 0x99;
sbit RI    = 0x98;

```

FIGURA 8-2b*continuación*

8.5 TIPOS Y MODELOS DE MEMORIA

El 8051 cuenta con varios tipos de espacios de memoria, incluyendo las memorias interna y externa para código y datos. Es razonable, por lo tanto, pensar en qué tipo de memoria residirán las variables que declaremos. Para esto, existen varios especificadores de tipos de memoria disponibles, como se muestra en la tabla 8-2.

El primer especificador de tipo de memoria dado en la tabla 8-2 es **code** (código). Code se utiliza para especificar que la variable residirá en la memoria para código, la cual tiene un rango de hasta 64 Kbytes. Por ejemplo:

```
char code mensaje_error[] = "Ha ocurrido un error";
```

declara un arreglo de caracteres llamado **mensaje_error** que reside en la memoria para código.

Si usted desea colocar una variable en la memoria para datos, le bastará con utilizar alguno de los cinco restantes especificadores de memoria para datos de la tabla 8-2. Aunque la elección queda por completo en sus manos, recuerde que cada tipo de memoria para datos afecta la velocidad de acceso y el tamaño disponible de la memoria para datos. Por ejemplo, considere las siguientes declaraciones:

```
signed int data num1;
bit bdata numbit;
unsigned int xdata num2;
```

La primera instrucción crea una variable entera con signo llamada **num1**, la cual reside en la memoria interna para datos (**data**, en las direcciones 00H a 7FH). La siguiente línea declara una variable bit **numbit**, que residirá en las ubicaciones de memoria direccionables por bit (direcciones de byte 20H a 2FH), también es conocida como variable **bdata**. Al final, la última línea declara una variable entera sin signo llamada **num2** residente en la memoria externa para datos, **xdata**. El tener una variable ubicada en la memoria interna para datos con direccionamiento directo acelera la velocidad de acceso en forma considerable; por lo tanto, para los programas en donde el tiempo es crucial, las variables deben ser del tipo **data**. Para el 8052 y otras variantes con memoria interna para datos de hasta 256 bytes, podemos utilizar el especificador **idata**. No obstante, observe que **idata** es más lento que **data**, pues debe usar direccionamiento indirecto. Por otra parte, si usted quiere que sus variables residan en la memoria externa, tiene la alternativa de declararlas como **pdata** o **xdata**. Una variable declarada como **pdata** reside en los primeros 256 bytes (una página) de la memoria externa, y si requerimos más espacio de almacenamiento, debemos utilizar el tipo de datos **xdata**, ya que éste permite el acceso de hasta 64 Kbytes de memoria externa para datos.

Pero, ¿qué tal si cuando declaramos una variable nos olvidamos de especificar explícitamente en qué tipo de memoria deberá residir, o si deseamos que todas las variables tengan asignado un tipo de memoria predeterminado, sin tener que especificar una por una? En este caso, podemos

TABLA 8-2

Tipos de memoria utilizados en el lenguaje C del 8051

Tipo de memoria	Descripción (tamaño)
code	Memoria para código (64 Kbytes)
data	Memoria interna para datos con direccionamiento directo (128 bytes)
idata	Memoria interna para datos con direccionamiento indirecto (256 bytes)
bdata	Memoria interna para datos direccionable por bit (16 bytes)
xdata	Memoria externa para datos (64 Kbytes)
pdata	Memoria externa para datos paginada (256 bytes)

TABLA 8-3

Modelos de memoria utilizados en el lenguaje C del 8051

Modelo de memoria	Descripción
Pequeño (small)	Las variables se declaran de modo predeterminado en la memoria interna para datos (data)
Compacto (compact)	Las variables se declaran de modo predeterminado en los primeros 256 bytes de la memoria externa para datos (pdata)
Grande (large)	Las variables se declaran de modo predeterminado en la memoria externa para datos (xdata)

hacer uso de los **modelos de memoria**. La tabla 8-3 presenta una lista de los diversos modelos de memoria disponibles

Con el propósito de que resida en cierto modelo de memoria, un programa se selecciona en forma explícita mediante la directiva `#pragma` de C. De otra manera, el modelo de memoria predeterminado es el pequeño (**small**). Recomendamos que los programas utilicen el modelo de memoria pequeño, ello permite el acceso más rápido posible al hacer que las variables residan en la memoria interna para datos de manera predeterminada.

En el modelo de memoria compacto (**compact**), todas las variables residen de manera predeterminada en la primera página de memoria externa para datos; mientras que en el modelo de memoria grande (**large**), todas las variables residen de modo predeterminado en todo el rango de memoria externa para datos, hasta los 64Kbytes.

8.6 ARREGLOS

Para mejorar la legibilidad, a menudo necesitamos agrupar un conjunto de variables utilizadas para almacenar datos del mismo tipo. Por ejemplo, la tabla ASCII para los dígitos decimales sería tal como lo muestra la tabla 8-4. Podríamos utilizar un **arreglo** para almacenar dicha tabla en un programa escrito en C del 8051. Un arreglo es un grupo de variables del mismo tipo de datos, donde cada variable puede ser ingresada mediante el nombre del arreglo junto con el índice apropiado.

El arreglo para almacenar la tabla ASCII de decimales es:

```
int tabla[10] =
{0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38, 0x39};
```

TABLA 8-4

Tabla ASCII para dígitos decimales

Dígito decimal	Código ASCII en hexadecimal
0	30H
1	31H
2	32H
3	33H
4	34H
5	35H
6	36H
7	37H
8	38H
9	39H

Observe que todos los elementos del arreglo están separados por comas. Utilizamos un índice iniciando desde 0 para acceder a un elemento individual. Por ejemplo, `tabla[0]` se refiere al primer elemento mientras que `tabla[9]` se refiere al último elemento incluido en esta tabla ASCII.

8.7 ESTRUCTURAS

Algunas veces es conveniente agrupar variables de diferentes tipos de datos pero que están relacionadas de alguna manera. Por ejemplo, nombre, edad y fecha de nacimiento de una persona estarían almacenados en diferentes tipos de variables, pero todas se refieren a detalles personales de un individuo. En este caso podemos declarar una **estructura**. Una estructura es un grupo de variables relacionadas que pueden ser de diferentes tipos de datos. Es posible declarar una estructura tal mediante:

```
struct persona {
    char nombre[10];
    int edad;
    long fecha_de_nacimiento;
};
```

Una vez declarada una estructura, se puede utilizar como especificador de tipo de datos para crear variables de estructura que contendrán el nombre, la edad y la fecha de nacimiento del miembro. Por ejemplo:

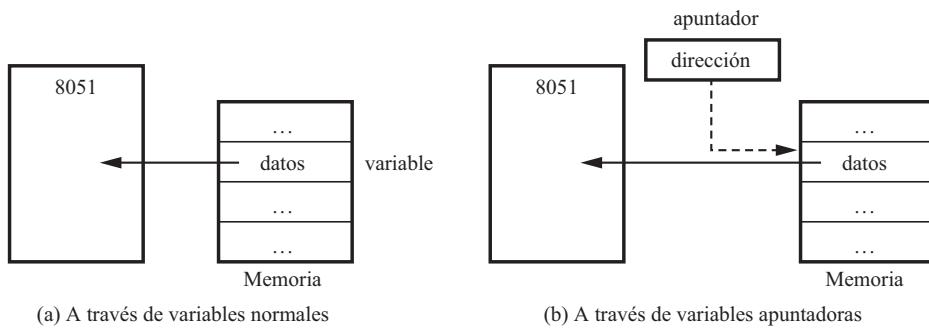
```
struct persona grace = {"Grace", 22, 01311980};
```

crea una variable de estructura `grace` para almacenar el nombre, la edad y la fecha de nacimiento de una persona llamada Grace. Para acceder a miembros específicos incluidos dentro de la variable de estructura `persona`, utilizamos el nombre de la variable seguido del operador punto (.) y el nombre del miembro. Por lo tanto, `grace.nombre`, `grace.edad`, `grace.fecha_de_nacimiento` se refieren a nombre, edad y fecha de nacimiento de Grace, respectivamente.

8.8 APUNTADORES

Cuando programamos el 8051 en lenguaje ensamblador, a menudo utilizamos registros tales como R0, R1 y DPTR para almacenar las direcciones de ciertos datos en cierta ubicación de memoria. Utilizamos el direccionamiento indirecto cuando accedemos a los datos a través de estos registros. En este caso, podemos afirmar que utilizamos R0, R1 o DPTR para apuntar a los datos, y por lo tanto son esencialmente apuntadores.

De igual forma, en el lenguaje C podemos acceder indirectamente a los datos a través de variables que son apuntadores y se definen en forma especial. Muchos alumnos que estudian el lenguaje C tratan de mantenerse alejados de los apuntadores. De hecho, los apuntadores son tan sólo un tipo especial de variables, pero mientras que las variables normales se utilizan para almacenar datos en forma directa, las variables apuntadoras almacenan las direcciones de los datos. Sólo recuerde que al utilizar ya sea variables normales o variables apuntadoras, al final podrá acceder a los datos de cualquier manera. La diferencia está en si se obtienen los datos desde donde están almacenados, como en el caso de las variables normales, o si primero se consulta un directorio para obtener su ubicación antes de tomarlos desde donde están, como sucede para las variables apuntadoras. La figura 8-3 presenta un contraste entre cómo se accede a los datos a través de las variables normales y las variables apuntadoras

**FIGURA 8-3**

Diferentes maneras de acceder a los datos. (a) A través de variables normales. (b) A través de variables apuntadoras

La declaración de un apuntador sigue el siguiente formato:

```
tipo_de_datos * nombre_del_apuntador;
donde
    tipo_de_datos      se refiere a qué tipo de datos está apuntando el apuntador
    *                  denota que ésta es una variable apuntadora
    nombre_del_
    apuntador          es el nombre del apuntador
```

Por ejemplo, en las siguientes declaraciones:

```
int * numPtr;
int num;
numPtr = &num;
```

primero se declara una variable apuntadora llamada numPtr que se utiliza para apuntar a los datos de tipo int. La segunda declaración establece una variable normal que utilizamos para comparar. La tercera línea asigna la dirección de la variable num al apuntador numPtr. Podemos obtener la dirección de cualquier variable mediante el *operador de dirección*, &, como lo hacemos en este ejemplo. Recuerde que una vez asignado, el apuntador numPtr contiene la dirección de la variable num, y no el valor de sus datos.

El ejemplo anterior también puede reescribirse de tal forma que el apuntador se inicialice directamente con una dirección cuando se declara por primera vez:

```
int num;
int * numPtr = &num;
```

Para poder ilustrar la diferencia que hay entre las variables normales y las variables apuntadoras, consideremos lo siguiente, el cual no es un programa completo escrito en C sino simplemente un fragmento para ejemplificar este punto:

```
int num = 7;
int * numPtr = &num;
printf("%d\n", num);
```

```
printf("%d\n", numPtr);
printf("%d\n", &num);
printf("%d\n", *numPtr);
```

La primera línea declara una variable normal, num, la cual se inicializa para contener el dato 7. Después se declara una variable apuntadora, numPtr, inicializada para apuntar a la dirección de num. Las siguientes cuatro líneas utilizan la función printf() para que los datos se impriman en alguna terminal de video conectada al puerto serial. La primera de estas líneas muestra el contenido de la variable num, que en este caso es el valor 7. La siguiente línea exhibe el contenido del apuntador numPtr, el cual es un número extraño que representa la dirección de la variable num. La tercera de estas líneas también presenta la dirección de la variable num, pues el operador de dirección es utilizado para obtener la dirección de la variable num. La última línea señala el valor real de los datos al que el apuntador numPtr está apuntando, el cual es 7. El símbolo de * se conoce como *operador de indirección*, y al utilizarlo junto con un apuntador podemos obtener en forma indirecta los datos cuya dirección está apuntada por el apuntador. Por lo tanto, la salida en la terminal de video mostraría:

```
7
13452 (o algún otro número extraño)
13452 (o algún otro número extraño)
7
```

Después que hayamos cubierto la sección 8.10, que muestra cómo escribir programas completos en C, usted podrá verificar la salida por su cuenta.

8.8.1 El tipo de memoria de un apuntador

Recuerde que los apuntadores también son variables, por lo cual surge la pregunta sobre dónde tienen que almacenarse. Cuando declaramos apuntadores, podemos especificar además diferentes tipos de áreas de memoria en donde deben ubicarse, por ejemplo:

```
int * xdata numPtr = &num;
```

Esto es igual a los ejemplos anteriores. Declaramos un apuntador numPtr, el cual apunta a datos de tipo int almacenados en la variable num. La diferencia está en el uso del especificador de tipo de memoria **xdata** después del símbolo de *. Esto especifica que el apuntador numPtr debe residir en la memoria externa para datos (**xdata**), y así estipulamos que el *tipo de memoria del apuntador* es **xdata**.

8.8.2 Apuntadores con tipo

Podemos profundizar más en la declaración de apuntadores. Considere el siguiente ejemplo:

```
int data * xdata numPtr = &num;
```

Esta instrucción declara el mismo apuntador numPtr para que resida en la memoria externa para datos (**xdata**), y este apuntador apunta a datos de tipo int, los cuales están almacenados en la variable num en la memoria interna para datos (**data**). El especificador de tipo de memoria **data**, incluido antes del símbolo de *, especifica el *tipo de memoria para datos* mientras que el especificador de tipo de memoria **xdata**, escrito después de *, establece el tipo de memoria del apuntador.

TABLA 8–5

Tipos de memoria para datos almacenados en el primer byte de los apuntadores sin tipo

Valor	Tipo de memoria para datos
1	idata
2	xdata
3	pdata
4	data/bdata
5	code

Las declaraciones de apuntadores donde los tipos de memoria se especifican en forma explícita se conocen como *apuntadores con tipo*. Los apuntadores con tipo tienen la propiedad que especificamos en nuestro código donde deben residir los datos apuntados por los apuntadores. El tamaño de los apuntadores con tipo depende del tipo de memoria para datos y puede ser de uno o dos bytes.

8.8.3 Apuntadores sin tipo

Los *apuntadores sin tipo* son aquellos para los cuales no especificamos en forma explícita el tipo de memoria para datos cuando los declaramos, y son apuntadores genéricos que pueden apuntar a los datos que residen en cualquier tipo de memoria. Los apuntadores sin tipo tienen la ventaja de que pueden utilizarse para apuntar a cualquier dato, independientemente de la clase de memoria en que esté almacenado. Todos los apuntadores sin tipo constan de 3 bytes, y por lo tanto son más grandes que los apuntadores con tipo. Los apuntadores sin tipo también son generalmente más lentos, pues el tipo de memoria no está determinado o no se conoce sino hasta que el programa compilado se ejecuta. El primer byte de los apuntadores sin tipo se refiere al tipo de memoria para datos, el cual es simplemente un número de acuerdo con la tabla 8-5. Los bytes segundo y tercero son, respectivamente, los bytes de alto y bajo nivel de la dirección a la cual se está apuntando.

Un apuntador sin tipo se declara de igual manera que en el lenguaje C normal, donde:

```
int * xdata numPtr = &num;
```

no especifica en forma explícita el tipo de memoria de los datos a los cuales apunta el apuntador. En este caso, estamos utilizando apuntadores sin tipo.

8.9 FUNCIONES

Al estudiar la programación en lenguaje ensamblador del 8051, aprendimos acerca de las ventajas de utilizar subrutinas para agrupar instrucciones comunes y frecuentemente utilizadas. El mismo concepto aparece en el lenguaje C del 8051, pero en lugar de llamarlas subrutinas las denominamos **funciones**. Así como en el lenguaje C convencional, una función debe estar declarada y definida. La definición de una función incluye una lista del número y tipo de las entradas, y el tipo de la salida (tipo del retorno), además de una descripción del contenido interno o de lo que se va a hacer dentro de esa función.

El formato de la definición de una función típica se muestra enseguida:

```
tipo_del_retorno nombre_de_función(argumentos) [memory] [reentrant]
[interrupt] [using]
{
    .
    .
}
```

donde²

<code>tipo_de_retorno</code>	se refiere al tipo de datos del valor de retorno (salida)
<code>nombre_de_función</code>	es cualquier nombre que se le quiera dar a la función
<code>argumentos</code>	es la lista del tipo y número de valores de entrada (argumentos)
<code>memory</code>	se refiere a un modelo de memoria explícito (pequeño, compacto o grande)
<code>reentrant</code>	se refiere a si la función es reentrante (recursiva) o no
<code>interrupt</code>	indica que la función es en realidad una rutina de servicio de interrupción
<code>using</code>	especifica en forma explícita qué banco de registros utilizar

Considere un ejemplo típico, en donde una función calcula la suma de dos números:

```
int sum(int a, int b)
{
    return a + b;
}
```

Esta función se llama `suma` y toma dos argumentos, ambos de tipo `int`. El tipo del retorno es también `int`, ello significa que la salida (el valor de retorno) será un `int`. Dentro del cuerpo de la función, delimitado por los corchetes, observamos que el valor de retorno es básicamente la suma de los dos argumentos. En este ejemplo hemos omitido la especificación explícita de las opciones `memory`, `reentrant`, `interrupt`, y `using`. Esto significa que los argumentos transferidos a la función utilizarían el modelo de memoria pequeño predeterminado, y por lo tanto estarían almacenados en la memoria interna para datos. Esta función es también predeterminada no recursiva y una función normal, no una ISR. De igual manera, la función utiliza el banco de registros 0 predeterminado.

8.9.1 Transferencia de parámetros

Los parámetros se transfieren hacia y desde las funciones en el lenguaje C del 8051, y se utilizan como argumentos de una función (entradas). Sin embargo, los detalles técnicos sobre dónde y cómo se almacenan estos parámetros son transparentes para el programador, quien no tiene que preocuparse acerca de estos tecnicismos. Aún así, en esta sección trataremos brevemente sobre cómo es que los parámetros se transfieren; entonces podremos apreciar cómo el compilador se encarga de hacerlo en nuestro favor. Los parámetros se transfieren mediante los registros o a través de la memoria en el lenguaje C del 8051. La transferencia de parámetros mediante el uso de registros es más rápida y es la manera predeterminada de hacer las cosas. La tabla 8-6 describe con mayor detalle los registros utilizados y su propósito.

TABLA 8-6

Registros utilizados en la transferencia de parámetros

Número del Argumento	Apuntador a Char/1 byte	Apuntador a INT/2 bytes	Long/Float	Apuntador genérico
1	R7	R6 & R7	R4-R7	R1-R3
2	R5	R4 & R5	R4-R7	
3	R3	R2 & R3		

²Las instrucciones delimitadas por los corchetes indican que son opcionales.

TABLA 8-7

Registros utilizados para regresar valores de las funciones

Tipo de valor de retorno	Registro	Descripción
bit		Bandera de acarreo (C)
Apuntador a char/unsigned char/1 byte	R7	
Apuntador a int/unsigned int/2 bytes	R6 & R7	MSB en R6, LSB en R7
long/unsigned long	R4–R7	MSB en R4, LSB en R7
float	R4–R7	Formato de 32 bits de IEEE
Apuntador genérico	R1–R3	Tipo de memoria en R3, MSB en R2, LSB en R1

Debido a que sólo existen ocho registros en el 8051, pueden presentarse situaciones en que no podamos contar con suficientes registros para efectuar la transferencia de parámetros. Cuando esto ocurre, el resto de los parámetros puede transferirse a través de ubicaciones fijas en memoria. Utilizamos la directiva de control NOREGPARAMS para especificar que todos los parámetros se transferirán a través de la memoria; y con la directiva de control REGPARAMS especificamos lo inverso.

8.9.2 Valores de retorno

A diferencia de los parámetros, que pueden transferirse ya sea mediante los registros o por ubicaciones de memoria, los valores de salida deben regresarse desde las funciones a través de los registros. La tabla 8-7 muestra los registros utilizados para regresar diferentes tipos de valores desde las funciones.

8.10 ALGUNOS EJEMPLOS EN LENGUAJE C DEL 8051

Hemos presentado brevemente los conceptos básicos de la programación en C del 8051. En esta sección analizaremos diversos ejemplos sobre cómo escribir dicha programación. Todos los programas del 8051 desarrollados en C, y que veremos de aquí en adelante, se compilaron y probaron en el software *μVision2* de Keil, el cual es un ambiente integrado de desarrollo (IDE) para el 8051 que incluye edición, compilación y depuración de código fuente.³

8.10.1 El primer programa

Como es costumbre en cualquier curso introductorio de lenguajes de programación de alto nivel y estructurados, el primer ejemplo que presentaremos es un programa simple que muestra el mensaje “Hello world”,

```
#include <REG51.H>          /* Declaraciones de los SFR */
#include <stdio.h>           /* Declaraciones para funciones de
                                E/S (como printf) */

main ()
{
    SCON = 0x52;             /* puerto serial, modo 1 */
    TMOD = 0x20;              /* temporizador 1, modo 2 */
```

³Puede encontrar una versión gratuita para evaluación en <http://www.keil.com>.

```

TH1 = -13;           /* conteo de recarga para 2400
                      baudios */
TR1 = 1;            /* inicia temporizador 1 */

while (1)           /* repite infinitamente */
{
    printf ("Hello World\n"); /* Muestra "Hello world" */
}
}

```

El programa anterior muestra en forma continua el mensaje “Hello world” al dispositivo conectado al puerto serial, el cual está conectado predeterminadamente a una ventana serial simulada. Consulte el apéndice H o la documentación de Keil en la Ayuda de μ Vision 2 para obtener más detalles sobre cómo interactuar con el IDE μ Vision2.

8.10.2 Temporizadores

En el capítulo 4 analizamos cómo interactuar con los temporizadores incorporados al 8051. A continuación presentamos las soluciones en C para algunos ejemplos del capítulo 4, como base de comparación entre la programación en lenguaje ensamblador y en lenguaje C del 8051.

EJEMPLO Generación de onda de pulso

8.1

Escriba un programa que genere una forma de onda periódica en P1.0 con la frecuencia más alta posible.

Solución

```

#include <REG51.H>      /* Declaraciones de los SFR */
sbit bitpuerto = P1^0; /* Utilizamos la variable bitpuerto para
                        referirnos a P1.0 */

main ()
{
    while (1)           /* repite infinitamente */
    {
        bitpuerto = 1;   /* activa P1.0 */
        bitpuerto = 0;   /* limpia P1.0 */
    }
}

```

Análisis

Este programa es en realidad la versión en C para el ejemplo 4.2. Aquí debemos activar y limpiar repetidas veces el puerto P1.0. Para ello sólo hay que escribir un 1 y un 0 en el puerto. Podemos referirnos al bit menos significativo del puerto P1 declarando una variable sbit y utilizando en el programa, debido a que P1 ya está declarado en el archivo de cabecera REG51.H.

Sólo hay que considerar un detalle cuando utilizamos instrucciones para generar retrasos de sincronización y formas de onda. En lenguaje ensamblador, podemos llevar a cabo esto con facilidad tomando nota de cuántos ciclos de máquina requiere cierta instrucción para ejecutarse. Sin embargo, en C no es posible determinar directamente qué tanto tardarían en ejecutarse una

instrucción, un ciclo o ciertas funciones. Todo esto depende del compilador para C específico que se utilice, pues diferentes compiladores utilizan diversos niveles de optimización y, por lo tanto, generan distintas combinaciones de instrucciones en lenguaje ensamblador y, en consecuencia, código máquina distinto.

La mejor manera de resolver esto es examinar las instrucciones en lenguaje ensamblador específicas que genera el compilador para cierta instrucción en C y determinar cuántos ciclos de máquina requerirían. Usted puede desensamblar en el IDE de μ Vision2 su programa en C para convertirlo en lenguaje ensamblador; para ello sólo tiene que observar los resultados en la ventana de desensamblaje. Consulte el apéndice H o la documentación de ayuda de Keil para obtener más información.

EJEMPLO Onda cuadrada de 10 kHz

8.2

Escriba un programa que utilice el temporizador 0 para generar una onda cuadrada de 10 kHz en P1.0.

Solución

```
#include <REG51.H>
sbit bitpuerto = P1^0; /* Declaraciones de los SFR */
/* Utilizamos la variable
bitpuerto para referirnos a
P1.0 */

main ()
{
    TMOD = 2; /* modo de autorrecarga de 8 bits */
    TH0 = -50; /* valor de recarga de -50 en TH0
    TR0 = 1; /* inicia temporizador 0 */

    while (1) /* repite infinitamente */
    {
        while (TF0 != 1); /* espera un desbordamiento */
        TF0 = 0; /* borra bandera de desbordamiento
        del temporizador */
        bitpuerto = !(bitpuerto); /* dispara P1.0 */
    }
}
```

Análisis

Esta es realmente la versión en C para el ejemplo 4.4. Para un ciclo infinito se utiliza la instrucción `while`, cuya condición es el valor constante 1. En lenguaje C, 1 denota el valor TRUE (verdadero) y 0 denota el valor FALSE (FALSO). Por lo tanto, en este caso la condición siempre sería verdadera, ello ocasiona que la instrucción `while` se repita en forma indefinida.

EJEMPLO Onda cuadrada de 1 kHz

8.3

Escriba un programa que utilice el temporizador 0 para generar una onda cuadrada de 1 kHz en P1.0.

Solución

```
#include <REG51.H>
sbit bitpuerto = P1^0; /* Declaraciones de los SFR */
/* Utilizamos la variable bitpuerto
para referirnos a P1.0 */
```

```

main ()
{
    TMOD = 1;                      /* modo de temporizador de 16 bits */
    while (1)                      /* repite infinitamente */
    {
        TH0 = 0xFE;                 /* -500 (byte superior)
        TL0 = 0x0C;                 /* -500 (byte inferior)
        TR0 = 1;                     /* inicia temporizador 0 */
        while (TF0 != 1);           /* espera un desbordamiento */
        TR0 = 0;                     /* detiene temporizador 0 */
        TF0 = 0;                     /* borra bandera de desbordamiento
                                         del temporizador */
        bitpuerto = !(bitpuerto);   /* dispara P1.0 */
    }
}

```

Análisis

Esta es la versión en C del ejemplo 4.5 y es bastante similar al ejemplo anterior.

EJEMPLO**8.4 Interfaz a un altavoz**

Un altavoz está conectado a P1.7, y un interruptor sin rebotes está conectado a P1.6. Escriba un programa que lea el nivel lógico proporcionado por el interruptor y active el altavoz durante 1 segundo por cada transición de 1 a 0 que se detecte.

Solución

```

#include <REG51.H>          /* Declaraciones de los SFR */

int cien = 100;

sbit bitentrada = P1 ^ 6;    /* Utilizamos la variable bitentrada para
                             referirnos a P1.6
sbit bitsalida = P1 ^ 7;    /* Utilizamos la variable bitsalida para
                             referirnos a P1.7 */
unsigned char R7;            /* utilizamos una variable de 8 bits para
                             representar R7 */

void retraso(void);          /* Prototipo de la función */

main ()
{
    TMOD = 1;                  /* utilizamos al temporizador 0 en el modo
                                 1 */
    while (1)                  /* repite infinitamente */
    {
        while (bitentrada!=1); /* espera por la entrada de valor 1*/
        while (bitentrada==1); /* espera por la entrada de valor 0*/
        bitsalida = 1;          /* enciende altavoz */
        retraso();               /* espera por un segundo */
        bitsalida = 0;          /* apaga altavoz */
    }
}

```

```

void retraso(void)
{
    R7 = cien;
    do
    {
        TH0 = 0xD8;           /* -10000 (byte superior) */
        TL0 = 0xF0;           /* -10000 (byte inferior) */
        TR0 = 1;
        while (TF0 != 1);
        TF0 = 0;
        TR0 = 0;
        R7 -= 1;
    }
    while (R7!=0);
}

```

Análisis

Este ejemplo utiliza una función llamada `retraso()` junto con la función principal (main). Debido a que la función retraso está definida después de la función principal, requerimos agregar un prototipo de función de una sola línea antes de la función principal para permitir que el compilador sepa de la existencia de dicha función. El archivo de cabecera REG51.H tampoco define a R7, así que la definimos aquí antes de utilizarla en la función `retraso()`. Hacemos esto para mantener correspondencia directa con la solución de ejemplo en lenguaje ensamblador encontrada para este problema en el ejemplo 4.7. En realidad, cuando programamos en C, en lugar de utilizar los registros de propósito general R0 a R7, podríamos simplemente declarar variables. Esto se debe también a que el compilador utiliza los registros R0 a R7 para transferir parámetros hacia y desde funciones.

8.10.3 El puerto serial

En esta sección daremos soluciones ejemplo para los ejemplos analizados previamente en el capítulo 5 y que están relacionados con el puerto serial del 8051.

EJEMPLO

8.5

Inicialización del puerto serial

Escriba una secuencia de instrucciones para inicializar al puerto serial de tal forma que opere como un UART de 8 bits a 2400 baudios. Utilice el temporizador 1 para proporcionar el reloj para la velocidad en baudios.

Solución

```

#include <REG51.H> /* Declaraciones de los SFR */

main ()
{
    SCON = 0x52;           /* puerto serial, modo 1 */
    TMOD = 0x20;           /* temporizador 1, modo 2 */
    TH1 = -13;              /* conteo de recarga para 2400 baudios */
    TR1 = 1;                /* inicia temporizador 1 */
}

```

EJEMPLO**8.6****Subrutina de carácter de salida**

Escriba una función llamada CARSAL para transmitir el código en ASCII de 7 bits en el acumulador a través del puerto serial del 8051, añadiendo una paridad par como el octavo bit. Regrese de la función con el acumulador intacto.

Solución

```
#include <REG51.H>      /* Declaraciones de los SFR */

sbit AccMSB = ACC ^ 7; /* Utilizamos la variable AccMSB para
referirnos a ACC.7 */

void CARSAL(void)
{
    CY = P;           /* coloca bit de paridad en bandera C */
    CY = !CY;         /* cambia a paridad impar */
    AccMSB = CY;     /* agrega al código de carácter */
    while (TI != 1); /* ¿Tx vacío? no: verifica de nuevo */
    TI = 0;           /* sí: borra bandera y */
    SBUF = ACC;       /* envía carácter */
    AccMSB = 0;       /* remueve bit de paridad */
}
```

Análisis

En este programa, declaramos una variable sbit llamada MSBAcc para referirnos al bit más significativo del acumulador, ACC, ya que se debe acceder a este bit.

EJEMPLO**8.7****Subrutina de carácter de entrada**

Escriba una función llamada CARENT para recibir como entrada un carácter del puerto serial del 8051 y regresar el código en ASCII de 7 bits en el acumulador. Espere una paridad impar en el octavo bit recibido y active la bandera de acarreo si existe un error de paridad.

Solución

```
#include <REG51.H>      /* Declaraciones de los SFR */

sbit AccMSB = ACC ^ 7; /* Utilizamos la variable AccMSB para
referirnos a ACC.7 */

void INCHAR(void)
{
    while (RI != 1);   /* espera por un carácter */
    RI = 0;            /* borra bandera */
    ACC = SBUF;        /* lee el carácter y lo coloca en el
acumulador */
    CY = P;            /* P debe activarse para una paridad impar
en el acumulador */
    CY = !CY;          /* complemento correcto indica si hubo
"error" */
    AccMSB = 0;         /* remueve paridad */
}
```

8.10.4 Interrupciones

Gracias a nuestro estudio sobre las funciones efectuado en la sección 8.9, sabemos que las rutinas de servicio de interrupciones (ISR) están escritas en una manera muy parecida a las funciones normales, con la excepción de que utilizamos la instrucción “interrupt” en la definición de la función. Ahora demostraremos cómo resolver los ejemplos del capítulo 6 acerca de interrupciones mediante programas escritos en lenguaje C del 8051.

EJEMPLO
8.8
Onda cuadrada utilizando interrupciones del temporizador

Escriba un programa que utilice el temporizador 0 e interrupciones para generar una onda cuadrada de 10 kHz en P1.0.

Solución

```
#include <REG51.H>           /* Declaraciones de los SFR */

sbit bitpuerto = P1 ^ 0;    /* Utilizamos variable bitpuerto para
                           referirnos a P1.0 */

main ()
{
    TMOD = 0x02;           /* temporizador 0, modo 2 */
    TH0 = -50;             /* retraso de 50 µs */
    TR0 = 1;               /* inicia temporizador */
    IE = 0x82;             /* habilita interrupción del temporizador
                           0 */
    while(1);              /* repite infinitamente */
}

void T0ISR(void) interrupt 1
{
    portbit = !bitpuerto;   /* dispara bit de puerto P1.0 */
}
```

Análisis

Este programa introduce el uso de una función de interrupción, un tipo especial de función que es automática siempre que ocurre la interrupción correspondiente. Observe que la instrucción de interrupción (interrupt) va seguida de un número de interrupción; en este caso 1, el cual se refiere a una interrupción del temporizador/contador 0. La tabla 8-8 presenta una lista de los diversos números de interrupciones, tipos y direcciones de vector.

EJEMPLO
8.9
Dos ondas cuadradas que utilizan interrupciones del temporizador

Escriba un programa que utilice interrupciones para generar, en forma simultánea, ondas cuadradas de 7 kHz y de 500 Hz en P1.7 y P1.6.

Solución

```
#include <REG51.H>           /* Declaraciones de los SFR */

sbit puertosiete = P1 ^ 7; /* Utilizamos la variable puertosiete
                           para referirnos a P1.7 */
sbit puertoseis = P1 ^ 6;  /* Utilizamos la variable puertoseis
                           para referirnos a P1.6 */

main ()
{
```

TABLA 8-8

Interrupciones y números de interrupción estándar del 8051

Número de interrupción	Descripción	Dirección vectorial
0	Dirección vectorial	0003H
1	Temporizador/ Contador 0	000BH
2	INT 1 externa	0013H
3	Temporizador/ Contador 1	0018H
4	Puerto serial	0023H

```

TMOD = 0x12;           /* temporizador 1, modo 1;
                           temporizador 0, modo 2 */
TH0 = -71;             /* 7kHz utilizando el temporizador 0 */
TR0 = 1;               /* inicia temporizador */
TF1 = 1;               /* fuerza interrupción del temporizador
                           1 */
IE = 0x8A;             /* habilita ambas interrupciones de
                           temporizador */
while(1);              /* repite infinitamente */

void T0ISR(void) interrupt 1
{
puertosiete = !puertosiete; /* dispara bit de puerto P1.7 */
}

void T1ISR(void) interrupt 3
{
TR1 = 0;
TH1 = 0xFC;             /* 1ms a alto nivel y */
TL1 = 0x18;             /*           a bajo nivel */
TR1 = 1;
puertoseis = !puertoseis; /* dispara bit de puerto P1.6 */
}

```

Análisis

Utilizamos dos funciones de interrupción en este ejemplo para dar servicio a las interrupciones de los temporizadores 0 y 1. Como se muestra en la tabla 8-8, los números de interrupción correspondientes son 1 y 3, respectivamente.

EJEMPLO**8.10****Salida de carácter mediante el uso de interrupciones**

Escriba un programa que utilice interrupciones para transmitir en forma continua el conjunto de códigos en ASCII (excluyendo los códigos de control) hacia una terminal conectada al puerto serial del 8051.

Solución

```

#include <REG51.H> /* Declaraciones de los SFR */

main ()
{
TMOD = 0x20;           /* temporizador 1, modo 2 */
TH1 = -26;              /* valor de recarga para 12000 baudios */

```

```

TR1 = 1;          /* inicia temporizador */
SCON = 0x42;      /* modo 1, activa TI para forzar primera
                   interrupción */
ACC = 0x20;       /* envía un espacio ASCII primero */
IE = 0x90;        /* habilita interrupción del puerto serial */
while(1);         /* repite infinitamente */
}

void PSISR(void) interrupt 4
{
if (ACC == 0x7F) /* si terminó conjunto ASCII */
    ACC = 0x20;  /* reinicializa a un espacio */
SBUF = ACC;      /* envía carácter al puerto serial */
ACC = ACC + 1;   /* incrementa código en ASCII */
TI = 0;           /* borra bandera de interrupción */
}

```

Análisis

Este programa utiliza una función de interrupción para dar servicio a una petición de interrupción del puerto serial, donde utilizamos el número de interrupción 4. Observe también que, aunque no existe restricción sobre qué nombre utilizar para una función de interrupción, la convención es emplear nombres explícitos tales como PSISR y T0ISR para referirnos a estas funciones de interrupción.

EJEMPLO Controlador de un horno

8.11 Con base en el 8051, diseñe un controlador que utilice interrupciones de tal forma que mantenga un edificio a $20^{\circ}\text{C} \pm 1^{\circ}\text{C}$.

Solución

```

#include <REG51.H>           /* Declaraciones de los SFR */

sbit bitsalida = P1 ^ 7;     /* Utilizamos variable bitsalida para
                           referirnos a P1.7 */
sbit bitcliente = P3 ^ 2;    /* Utilizamos variable bitcliente
                           para referirnos a P3.2 */

main ()
{
IE = 0x85;                  /* habilita interrupciones externas */
IT0 = 1;                     /* se dispara mediante un borde
                           negativo */

IT1 = 1;
bitsalida = 1;                /* enciende horno */
if (bitcliente != 1)          /* si T > 21 grados, */
    bitsalida = 0;            /* apaga horno */
while(1);                    /* repite infinitamente */

void EX0ISR(void) interrupt 0
{
bitsalida = 0;                /* apaga horno */
}

```

```

void EX1ISR(void) interrupt 2
{
    bitsalida = 1;           /* enciende horno */
}

```

Análisis

De nuevo, como ilustra este ejemplo, el escribir funciones de interrupción para dar servicio a interrupciones externas es similar a escribir funciones para interrupciones de temporizador y puerto serial. La diferencia está en los números de interrupción que estamos utilizando, los cuales pueden obtenerse a partir de la tabla 8-8. Observe también que las funciones de interrupción se nombraron en forma explícita.

EJEMPLO Sistema de detección de intrusos

8.12

Diseñe un sistema de detección de intrusos que utilice interrupciones de tal forma que se genere un tono de 400 Hz durante un segundo (utilizando un altavoz conectado a P1.7) siempre que un sensor de puerta conectado a $\overline{\text{INT0}}$ realice una transición de nivel alto a nivel bajo.

Solución

```

#include <REG51.H>          /* Declaraciones de los SFR */

sbit bitsalida = P1 ^ 7; /* utilizamos la variable bitsalida para
                         referirnos a P1.7 */
unsigned char R7;        /* utilizamos la variable de 8 bits para
                         representar a R7 */

main ()
{
    IT0 = 1;             /* se activa por un borde negativo */
    TMOD = 0x11;         /* modo de temporizador de 16 bits */
    IE = 0x81;           /* habilita sólo EXT 0 */
    while(1);            /* repite infinitamente */
}

void EX0ISR(void) interrupt 0
{
    R7 = 20;              /* 20 x 5000 µs = 1 segundo */
    TF0 = 1;               /* fuerza interrupción del temporizador 0 */
    TF1 = 1;               /* fuerza interrupción del temporizador 1 */
    ET0 = 1;               /* inicia tono durante 1 segundo */
    ET1 = 1;               /* habilita interrupciones de temporizador */
                           /* */
                           /* las interrupciones de temporizador harán
                           el trabajo */
}

void T0ISR(void) interrupt 1
{
    TR0 = 0;                /* detiene temporizador */
    R7 = R7 - 1;             /* disminuye R7 */
    if (R7 == 0)              /* si 20a vez, */
    {
        ET0 = 0;                /* se deshabilita a sí misma */
        ET1 = 0;
    }
}

```

```

else
{
    TH0 = 0x3C;           /* retraso de 0.05 seg */
    TL0 = 0xB0;
    TR0 = 1;
}
}

void T1ISR(void) interrupt 3
{
TR1 = 0;
TH1 = 0xFB;           /* conteo para 400Hz */
TL1 = 0x1E;
bitsalida = !bitsalida; /* ¡Música maestro! */
TR1 = 1;
}

```

Análisis

Esta es una combinación de varias funciones de interrupción que estudiamos antes.

RESUMEN

En este capítulo presentamos los conceptos de la programación del 8051 en lenguaje C, por contraste con los capítulos anteriores donde tratamos la programación del 8051 en lenguaje ensamblador. No obstante, ambos lenguajes hacen posible la escritura de programas, especialmente de aquellos que son complejos, en una manera organizada y estructurada. Este será el tema del siguiente capítulo.

PROBLEMAS

- 8.1** ¿Cómo difiere el lenguaje C del 8051 del lenguaje C convencional en términos de los tipos de datos?
- 8.2** Suponga que desea declarar una variable para almacenar un valor de bit en la dirección de bit física 20H. Describa cómo podría llevarlo a cabo en lenguaje C para el 8051.
- 8.3** ¿Para qué se utiliza el símbolo (`\`) en términos de ubicaciones direccionables por bit?
- 8.4** ¿De cuántas maneras distintas podemos declarar variables `sbit`? Explique cada una de estas maneras mediante un ejemplo.
- 8.5** ¿Cuál es la diferencia entre una variable `SFR` y una variable `unsigned char`?
- 8.6** Explique la diferencia que hay entre los términos “tipos de memoria” y “modelos de memoria”.
- 8.7** Explique la diferencia que hay entre una función y una función de interrupción.
- 8.8** Explique, mediante un ejemplo, lo que usted entiende por un apuntador.
- 8.9** Suponga que usted quiere hacer que la mayoría de sus variables de programa resida en la memoria RAM externa, mientras que ciertas variables en donde el tiempo es crucial deberán residir en los primeros 128 bytes de la memoria RAM interna. Describa brevemente cómo llevaría a cabo esto.

- 8.10** ¿Por qué no es recomendable utilizar al acumulador, ACC, cuando programamos en lenguaje C?
- 8.11** Escriba un programa en C que tenga los números del 1 al 10 en memoria para código y que copie los números impares (respectivamente pares) a la memoria externa para datos cuando P1.0 está activado (respectivamente borrado).
- 8.12** Escriba un programa en C del 8051 para transferir en forma continua una cadena de caracteres en ASCII, almacenada en la memoria interna para datos localizada en la ubicación 30H, a la memoria externa para datos localizada en la ubicación 1234H.
- 8.13**
- Muestre cómo crearía una tabla de búsqueda en la memoria para código para almacenar los primeros 10 valores de la función $f(x) = e^{x+2}$ correspondientes a $x = \{1, 2, \dots, 10\}$.
 - Después declare un apuntador que está almacenado en una ubicación de memoria interna RAM con acceso indirecto, y úselo para obtener el valor de $f(2)$ e imprimirlo en la pantalla.

<http://www.ingeelectronico.blogspot.com>

Estructura y diseño de programas

9.1 INTRODUCCIÓN

¿Qué hace a un programa mejor que otros? Más allá de razones simples del tipo: “porque funciona”, la respuesta es compleja y depende de muchos factores, tales como los requerimientos de su mantenimiento, el lenguaje de la computadora, la calidad de la documentación, el tiempo de desarrollo, qué tan largo es el programa, el tiempo de ejecución, la confiabilidad, la seguridad, y detalles por el estilo. En este capítulo presentamos las características de los buenos programas y algunas técnicas para desarrollarlos. Empecemos con una introducción a las técnicas de la programación estructurada.

La **programación estructurada** es una técnica para organizar y codificar programas que reduce la complejidad, mejora la claridad, y facilita la depuración y modificación de dichos programas. La idea de lograr programas apropiadamente estructurados se enfatiza en la mayoría de las tareas de programación, que también veremos aquí. Podemos apreciar la potencialidad de este enfoque al considerar la siguiente declaración: todos los programas pueden escribirse empleando solamente tres estructuras. Esto parece demasiado bueno para ser real, pero no es así. Las “instrucciones”, los “ciclos” y las “opciones” constituyen un conjunto íntegro de estructuras, y podemos codificar todos los programas utilizando sólo estas tres estructuras. El control del programa se transfiere a través de las estructuras sin bifurcaciones incondicionales hacia otras estructuras. Cada estructura tiene un punto de entrada y uno de salida. Por lo general, un programa estructurado posee cierta jerarquía de subrutinas, cada subrutina cuenta con un solo punto de entrada y un solo punto de salida.¹

El propósito de este capítulo es introducir al lector a la programación estructurada en la forma como se aplica a la programación en lenguaje ensamblador y en lenguaje C del 8051. Esto resulta muy útil especialmente en el contexto de la programación en lenguaje ensamblador, pues aunque los lenguajes de alto nivel (tales como Pascal o C) promueven la programación estructurada a través de sus instrucciones (WHILE, FOR, etc.) y convenciones de notación (sangría), el lenguaje ensamblador no cuenta con tales propiedades inherentes. No obstante, la programación en lenguaje ensamblador puede beneficiarse enormemente al utilizar las técnicas

¹Los programas están compuestos por funciones o procedimientos desarrollados en lenguajes de alto nivel.

estructuradas. Por otra parte, el lenguaje C del 8051 hereda todas las técnicas estructuradas, ya que es una extensión del lenguaje C. Así, conforme desarrollemos técnicas estructuradas para el lenguaje ensamblador en este capítulo, presentaremos también las propiedades estructuradas del lenguaje C del 8051 como una comparación con el lenguaje ensamblador.

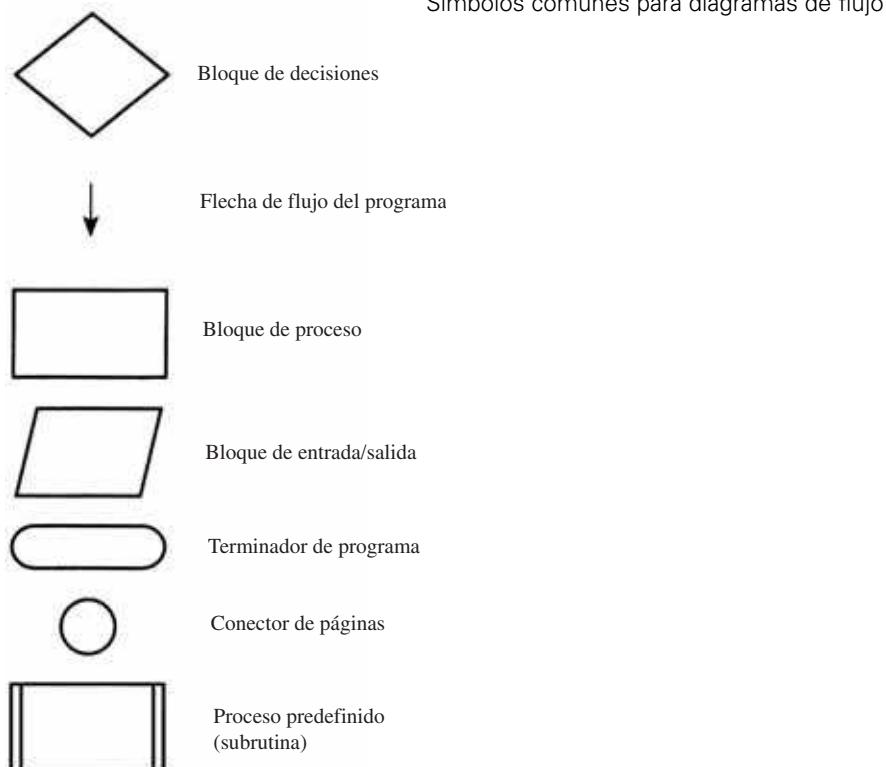
Para avanzar hacia nuestra meta (producir buenos programas en lenguaje ensamblador), resolveremos los problemas de ejemplo utilizando cuatro métodos:

- Diagramas de flujo
- Seudocódigo
- Lenguaje ensamblador
- Lenguaje C del 8051

Nuestro objetivo principal es, por supuesto, la solución de problemas de programación en lenguaje ensamblador; sin embargo, los diagramas de flujo y el seudocódigo son herramientas útiles para las etapas iniciales. Ambas herramientas “visuales” facilitan la formulación y el entendimiento del problema. Permiten que la problemática pueda describirse en términos de “lo que debe hacerse” en lugar de “cómo hacerlo”. La solución puede expresarse a menudo mediante los diagramas de flujo o el seudocódigo en términos independientes del sistema, sin considerar lo complejo del conjunto de instrucciones de la máquina destino.

Es raro utilizar seudocódigo al mismo tiempo que diagramas de flujo. La alternativa preferida es, en primer lugar, cuestión de estilo personal. La figura 9-1 muestra los símbolos más comunes empleados en los diagramas de flujo.

FIGURA 9-1
Símbolos comunes para diagramas de flujo



El seudocódigo es tal como su nombre sugiere: “casi” un lenguaje de computadora. La idea se ha venido utilizando de modo informal como una manera conveniente de elaborar un esquema de soluciones para problemas de programación. El seudocódigo, como lo aplicamos aquí, imita la sintaxis de Pascal o C en la notación de su estructura, pero al mismo tiempo alienta la utilización del lenguaje natural para describir acciones. Por lo tanto, instrucciones tales como

[obtener un carácter del teclado]

puede aparecer en seudocódigo. El beneficio de utilizar seudocódigo está en la aplicación estricta de estructura combinada con lenguaje informal. Por lo tanto, podremos encontrar

```
IF [condición es verdadera]
    THEN [ejecuta instrucción 1]
    ELSE BEGIN
        [ejecuta instrucción 2]
        [ejecuta instrucción 3]
    END
```

o

```
IF [la temperatura es menor a 20 grados centígrados]
    THEN [vestir una chamarra]
    ELSE BEGIN
        [vestir una camiseta de manga corta]
        [traer gafas antisolares]
    END
```

El uso de palabras clave, sangría y orden es esencial para emplear efectivamente el seudocódigo. Nuestra meta es demostrar con claridad la solución a un problema de programación utilizando diagramas de flujo y/o seudocódigo, de tal forma que la traducción al lenguaje ensamblador sea más fácil que la codificación directa en este lenguaje. El producto final también es más fácil de leer, depurar y mantener. Definiremos al seudocódigo con más formalidad en una sección posterior.

9.2 VENTAJAS Y DESVENTAJAS DE LA PROGRAMACIÓN ESTRUCTURADA

Las ventajas de adoptar un enfoque estructurado para realizar la programación son numerosas. Entre éstas se incluyen las siguientes:

- La secuencia de operaciones es fácil de seguir, lo cual facilita la depuración.
- Existe un número finito de estructuras con terminología estandarizada.
- Las estructuras se prestan a la fácil construcción de subrutinas.
- El conjunto de estructuras es completo; en otras palabras, todos los programas pueden escribirse utilizando tres estructuras.
- Las estructuras se documentan automáticamente y, por lo tanto, son fáciles de leer.
- Las estructuras resultan fáciles de describir en diagramas de flujo, diagramas de sintaxis, seudocódigo, etcétera.
- La programación estructurada resulta en mayor productividad del programador —los programas pueden escribirse más rápidamente.

Sin embargo, también existen algunos sacrificios. La programación estructurada tiene desventajas tales como:

- Sólo pocos lenguajes de alto nivel (Pascal, C, PL/M) aceptan las estructuras en forma directa; otros requieren una etapa extra de traducción.
- Los programas estructurados pueden ejecutarse con más lentitud y requerir de más memoria que su programa equivalente no estructurado.
- Algunos problemas (una minoría) son más difíciles de resolver utilizando sólo las tres estructuras en lugar del método de fuerza bruta tipo “espagueti”.
- Las estructuras anidadas pueden ser difíciles de seguir.

9.3 LAS TRES ESTRUCTURAS

Todos los problemas de programación pueden resolverse mediante el uso de tres estructuras:

- Instrucciones
- Ciclos
- Opciones

Lo completo de estas estructuras parece un poco increíble; pero, con ayuda de las estructuras anidadas (estructuras dentro de otras estructuras), podemos demostrar fácilmente que es posible resolver cualquier problema de programación utilizando sólo tres estructuras. Examinemos cada una con detalle.

9.3.1 Instrucciones

Las instrucciones proporcionan el mecanismo básico necesario para hacer algo. Entre las posibilidades se incluyen la simple asignación de un valor a una variable, tal como

```
[conteo = 0]
```

o la llamada a una subrutina, tal como

```
IMPRIME_CADENA("Seleccione una opción:")
```

Podemos utilizar un grupo de instrucciones, o **bloque de instrucciones**, en cualquier lugar donde utilicemos una sola instrucción. Esto se realiza en pseudocódigo mediante la agrupación de las instrucciones entre las palabras claves BEGIN y END como sigue:

```
BEGIN
    [instrucción 1]
    [instrucción 2]
    [instrucción 3]
END
```

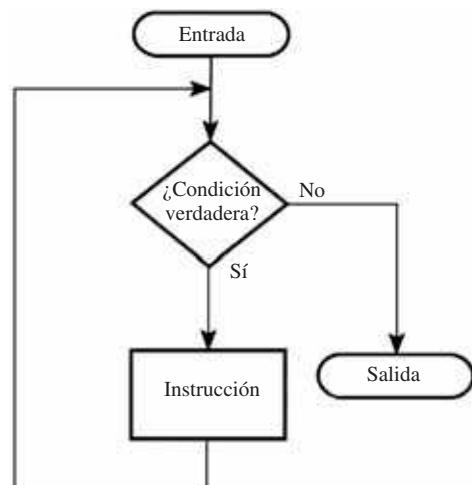
Observe que se aplicó sangría a las instrucciones incluidas dentro del bloque de instrucciones, en comparación con las palabras clave BEGIN y END. Esta es una característica importante de la programación estructurada.

9.3.2 La estructura de ciclo

La segunda estructura básica es la de “ciclo”, utilizada para realizar una operación repetidas veces. La suma de una serie de números o la búsqueda de un valor en una lista son dos ejemplos

FIGURA 9–2

Diagrama de flujo para la estructura WHILE/DO



de problemas de programación que requieren ciclos. También utilizamos el término “iteración” en este contexto. Aunque existen diversas formas de ciclos posibles, sólo dos son necesarias: WHILE/DO y REPEAT/UNTIL.

9.3.2.1 La instrucción WHILE/DO La instrucción WHILE/DO proporciona el modo más fácil para implementar ciclos. Se le llama “instrucción” porque se le trata como a una instrucción —como una sola unidad con un solo punto de entrada (el comienzo) y un solo punto de salida (el final). El formato en seudocódigo es:

```
WHILE [condición] DO
    [instrucción]
```

La “condición” es una “expresión relacional” que se evalúa a “verdadero” o “falso”. La instrucción (o el bloque de instrucciones) que sigue a la palabra clave “DO” se ejecuta si la condición es verdadera, y después se reevalúa dicha condición. Esto se repite hasta que la expresión relacional resulta en una respuesta de falso; ello causa que la “instrucción” se ignore y la ejecución del programa continúa en la siguiente instrucción. La figura 9-2 muestra el diagrama de flujo para la estructura WHILE/DO.

EJEMPLO Estructura WHILE/DO

9.1

Ilustre una estructura WHILE/DO tal que ejecute una instrucción mientras la bandera de acarreo del 8051 esté activada.

Solución

Seudocódigo:

```
WHILE [c == 1] DO
    [instrucción]
```

(Nota: El doble signo de igual se utiliza para distinguir al operador relacional, que efectúa una prueba de igualdad, del operador de asignación, el cual es un solo signo de igual [consulte la sección 9.4 Sintaxis del seudocódigo].)

Diagrama de flujo:

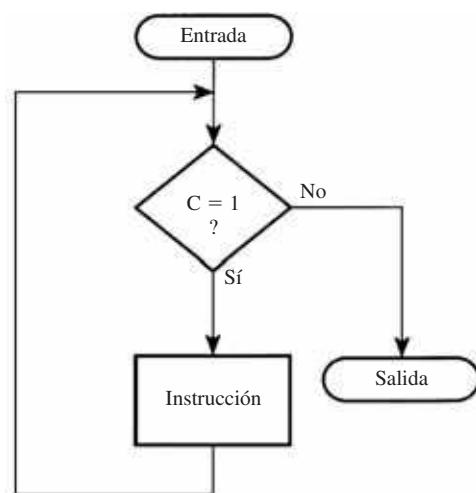


FIGURA 9–3

Diagrama de flujo para el ejemplo 9.1

Lenguaje ensamblador:

ENTRADA:	JNC Salida
INSTRUCCION:	(instrucción)
	JMP Entrada
SALIDA:	(continúa)

Lenguaje C del 8051:

```

while (c == 1)
    {instrucción;}
  
```

Como una regla general, las acciones incluidas dentro del bloque de instrucciones deben afectar a por lo menos una variable que se localice en la expresión relacional; de otra manera aparecerá un error en forma de ciclo infinito.

EJEMPLO Subrutina SUMA

9.2

En el 8051, escriba una subrutina llamada SUMA para calcular la suma de una serie de números. Los parámetros transferidos a la subrutina incluyen el tamaño de la serie en R7 y la dirección inicial de la serie en R0. (Asuma que la serie está ubicada en la memoria interna del 8051). Regrese con la suma en el acumulador.

Solución

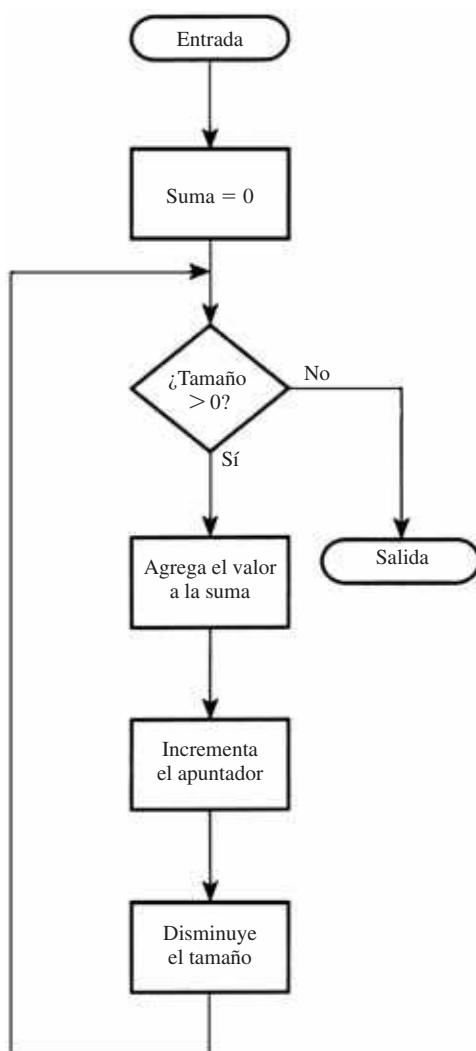
Seudocódigo:

```

[suma = 0]
WHILE [tamaño > 0] DO BEGIN
    [suma = suma + @apuntador]
    [incrementa el apuntador]
    [disminuye el tamaño]
END
  
```

Diagrama de flujo:

FIGURA 9–4
Diagrama de flujo para el
ejemplo 9.2



Lenguaje ensamblador: (muy estructurado; 13 bytes)

```

SUMA:      CLR      A
CICLO:     CJNE    R7, #0, INSTRUCCION
           JMP     SALIDA
INSTRUCCION: ADD    A, @R0
              INC    R0
              DEC    R7
              JMP    CICLO
SALIDA:    RET
  
```

(ligeramente estructurado; 9 bytes)

```

SUMA:      CLR      A
INC       R7
MAS:      DJNZ    R7, IGNORA
          RET
IGNORA:   ADD    A, @R0
          INC    R0
          SJMP   MAS
  
```

Observe que aquí la solución ligeramente estructurada es más pequeña (y más rápida) que la solución muy estructurada. Sin duda, los programadores con experiencia codificarán ejemplos simples como éste de manera intuitiva, y seguirán una solución ligeramente estructurada. Los principiantes, sin embargo, claramente se benefician al solucionar problemas en seudocódigo primero para después avanzar al lenguaje ensamblador al tiempo que siguen la estructura del seudocódigo.

Lenguaje C del 8051:

```

void suma(int * inicio, int tamaño)
{
    int resultado = 0, i = 0;
    while (tamaño > 0)
    {
        resultado = resultado + inicio[i];
        i++;
        tamaño--;
    }
}
  
```

EJEMPLO Estructura WHILE/DO

9.3

Ilustre una estructura WHILE/DO utilizando la siguiente condición compuesta: que el acumulador no sea igual al retorno de carro (0DH) y R7 no sea igual a 0.

Solución

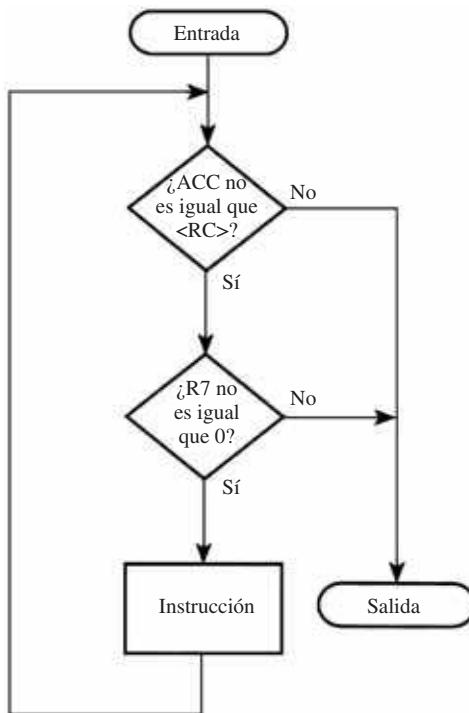
Seudocódigo:

```

WHILE [ACC!=CR AND R7 !=0] DO
[instrucción]
  
```

Diagrama de flujo:

FIGURA 9-5
Diagrama de flujo para el
ejemplo 9.3



Lenguaje ensamblador:

```

ENTRADA:      CJNE A, #0DH, IGNORA
              JMP SALIDA
IGNORA:       CJNE R7, #0, INSTRUCCION
              JMP SALIDA
INSTRUCCION:   una o más instrucciones)
.
.
.
JMP ENTRADA
SALIDA:        (continúa)
  
```

Lenguaje en C del 8051:

```

while( (ACC != C) && (R7 != 0) )
{instrucción;}
  
```

9.3.2.2 La instrucción REPEAT/UNTIL La instrucción REPEAT/UNTIL es muy similar a WHILE/DO, y resulta útil cuando la “instrucción de repetición” debe llevarse a cabo por lo menos una vez. Las instrucciones WHILE/DO prueban la condición primero; entonces, la instrucción podría no ejecutarse en su totalidad.

Seudocódigo:

```
REPEAT [instrucción]
    UNTIL [condición]
```

Diagrama de flujo:

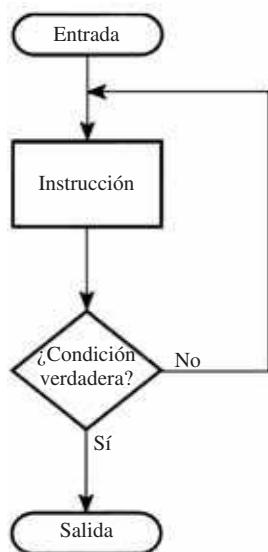


FIGURA 9–6

Diagrama de flujo para la estructura REPEAT/UNTIL

EJEMPLO Subrutina de búsqueda

9.4

Escriba una subrutina en el 8051 para buscar en una cadena terminada por el carácter nulo, a la cual apunta R0, y determine si la letra “Z” se encuentra en la cadena. Devuelva ACC = Z en caso afirmativo o ACC = 0 en cualquier otro caso.

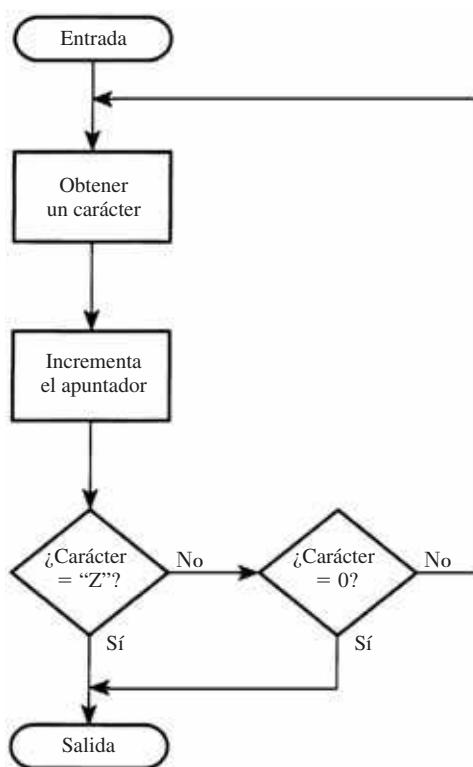
Solución

Seudocódigo:

```
REPEAT
    [ACC = @apuntador]
    [incrementa apuntador]
UNTIL [ACC == 'Z' o ACC == 0]
```

Diagrama de flujo:

FIGURA 9-7
Diagrama de flujo para el ejemplo 9.4



Lenguaje ensamblador:

INSTRUCCION:	MOV A, @R0
	INC R0
	JZ SALIDA
	CJNE A, #'Z', INSTRUCCION
SALIDA:	RET

Lenguaje en C del 8051:

```

char instrucción(char * inicio)
{
    char a;
    do
    {
        a = *inicio;
        inicio++;
    }
}
  
```

```

while ( (a != 'z') || (a != 0) );
return a;
}

```

9.3.3 La estructura opción

La tercera estructura básica es la de “opción”: la “bifurcación en el camino” del programador. Los dos arreglos más comunes son la instrucción IF/THEN/ELSE y la instrucción CASE.

9.3.3.1 La instrucción IF/THEN/ELSE La instrucción IF/THEN/ELSE se utiliza cuando se debe seleccionar una de dos instrucciones (o bloques de instrucciones), dependiendo de una condición. La parte ELSE de la instrucción es opcional.
Seudocódigo:

```

IF [condición]
    THEN [instrucción 1]
    ELSE [instrucción 2]

```

Diagrama de flujo:

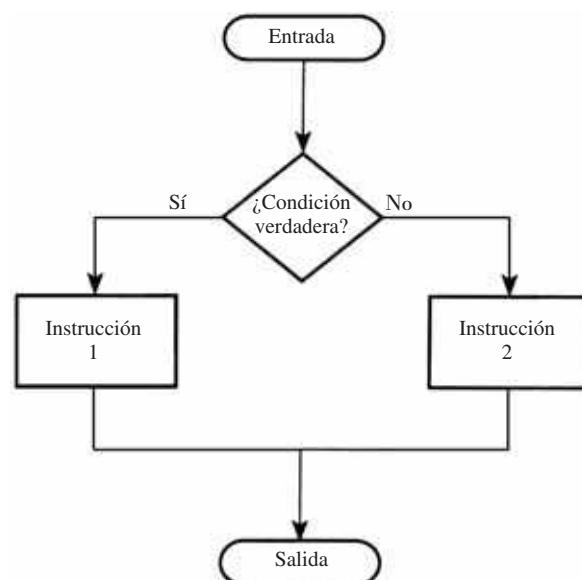


FIGURA 9–8
Diagrama de flujo para la estructura IF/THEN/ELSE

EJEMPLO

9.5

Prueba de carácter

Escriba una secuencia de instrucciones para enviar como entrada y probar un carácter a través del puerto serial. Si el carácter es un ASCII presentable (en otras palabras, situado en el rango 20H a 7EH), entonces habrá que presentarlo como tal; en cualquier otro caso, habrá que desplegar un punto (.).

Solución

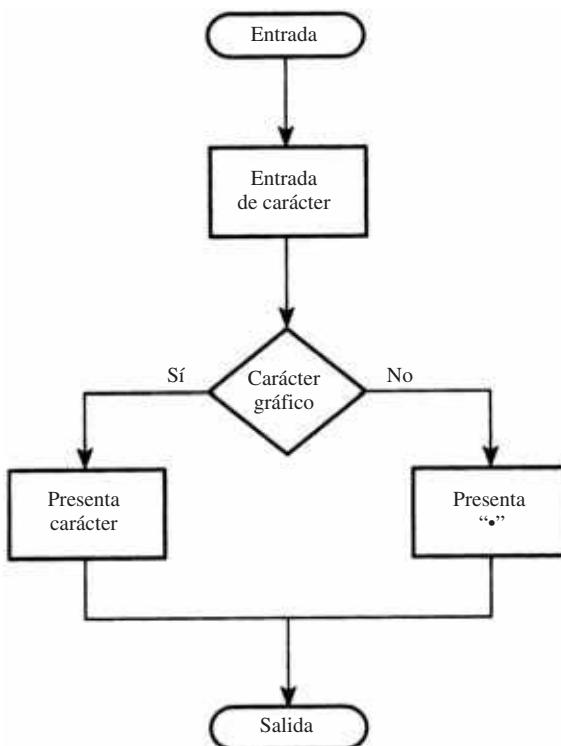
Seudocódigo:

```
[entrada de carácter]
IF [carácter == gráfico]
    THEN [presenta carácter]
    ELSE [presenta '.']
```

Diagrama de flujo:

FIGURA 9-9

Diagrama de flujo para el ejemplo 9.5



Lenguaje ensamblador: (muy estructurado; 14 bytes)

ENTRADA :	ACALL	CARENT
	ACALL	ESGRAF
	JNC	INSTRUCCION2
INSTRUCCION1 :	ACALL	CARSAL
	JMP	SALIDA
STMENT2 :	MOV	A, #' .'
	ACALL	CARSAL
SALIDA :	(continúa)	

(ligeramente estructurado; 10 bytes)

	ACALL	CARENT
	ACALL	ESGRAF
	JC	IGNORA
	MOV	A, #'.'
IGNORA:	ACALL	CARSAL
	(continúa)	

Lenguaje en C del 8051:

```
while (1)
{
    char a;
    a = carent ();
    if (esgraf(a))
        carsal(a);
    else
        carsal('.');

}
```

Modifique la estructura para repetir infinitamente:

```
WHILE [1] DO BEGIN
    [entrada de carácter]
    IF [carácter == gráfico]
        THEN [presenta carácter]
    ELSE [presenta '.']
END
```

9.3.3.2 La instrucción CASE La instrucción CASE es una variación útil de la instrucción IF/THEN/ELSE. Se utiliza cuando tenemos que seleccionar una instrucción de entre muchas instrucciones, según lo determine cierto valor.

Seudocódigo

```
CASE [expresión] OF
    0:[instrucción 0]
    1:[instrucción 1]
    2:[instrucción 2]
    .
    .
    .
    n:[instrucción n]
    [instrucción predeterminada]
END_CASE
```

Diagrama de flujo:

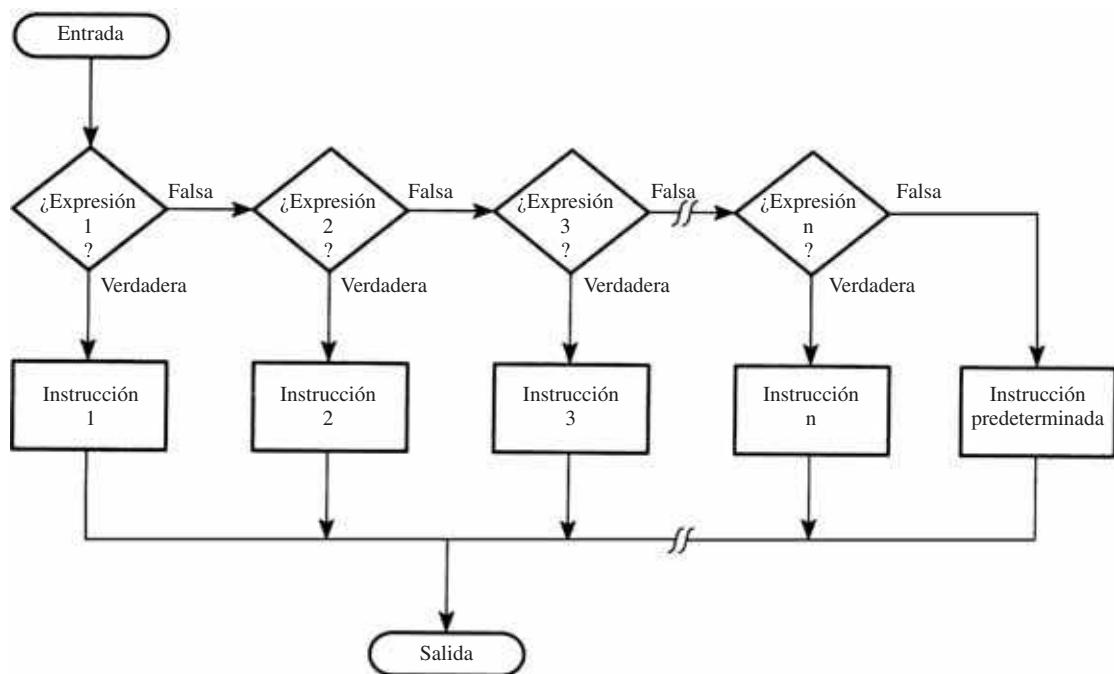


FIGURA 9–10

Diagrama de flujo para la estructura CASE

EJEMPLO Respuesta del usuario

9.6

Un programa controlado mediante un menú requiere de una respuesta del usuario de 0, 1, 2 o 3 para seleccionar una de cuatro acciones posibles. Escriba una secuencia de instrucciones para obtener un carácter de entrada del teclado y saltar a ACCION0, ACCION1, ACCION2 o ACCION3, dependiendo de la respuesta del usuario. Omita la verificación de errores.

Solución

Seudocódigo:

```

[obtener un carácter]
CASE [carácter] OF
  '0': [instrucción 0]
  '1': [instrucción 1]
  '2': [instrucción 2]
  '3': [instrucción 3]
END_CASE
  
```

Diagrama de flujo:

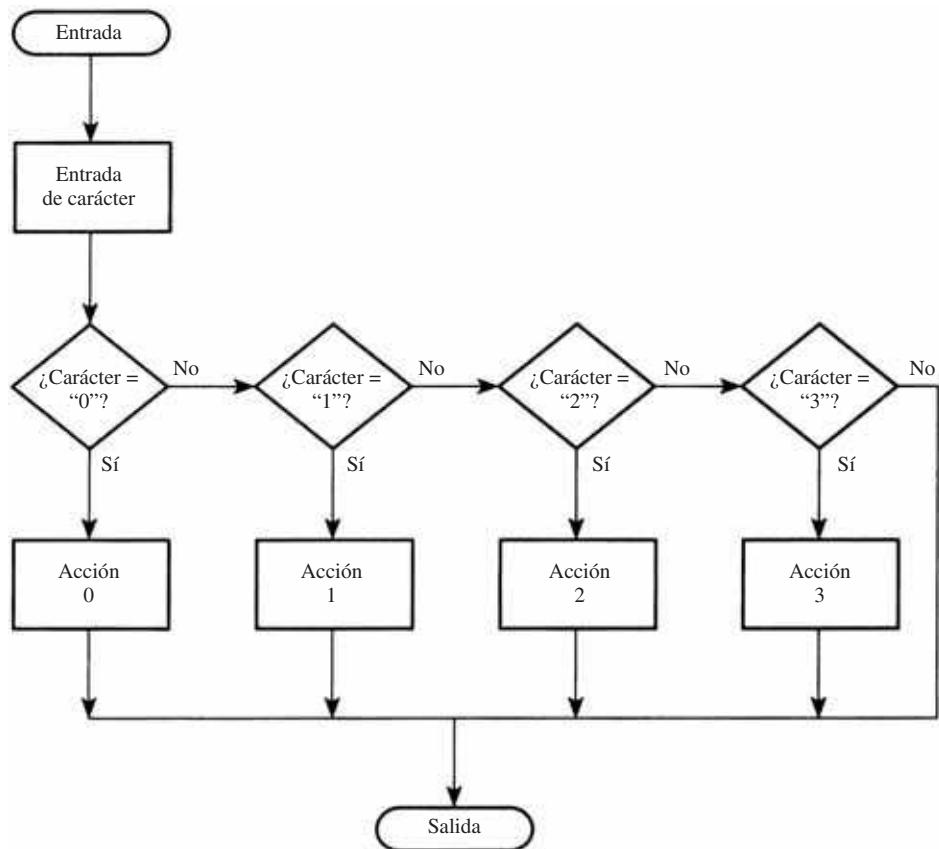


FIGURA 9-11

Diagrama de flujo para el ejemplo 9.6

Lenguaje ensamblador: (muy estructurado)

```

CALL CARENT
CJNE A, #'0', IGNORA1
ACCION0: .
.
.
JMP SALIDA
IGNORA1:   CJNEA, #'1', IGNORA2
ACCION1: .
.
.
JMP SALIDA
  
```

```

IGNORA2:      CJNE A, #'2', IGNORA3
ACCION2:      .
.
.
IGNORA3:      CJNEA, #'3', SALIDA
ACCION3:      .
.
.
SALIDA:       (continúa)

```

(ligeramente estructurado)

```

CALL CARENT          ;REDUCE a 2 BITS
ANL   A, #3          ;DESPLAZAMIENTO DE PALABRA
RL    A
MOV   DPTR, #TABLA
JMP   @A+DPTR
TABLA:   AJMP ACCION0
AJMP ACCION1
AJMP ACCION2
ACCION3:  .
.
.
JMP   SALIDA
ACCION0:  .
.
.
JMP   SALIDA
ACCION1:  .
.
.
JMP   SALIDA
ACCION2:  .
.
.
SALIDA:    (continúa)

```

Lenguaje C del 8051:

```

char a;
a = carent();
switch(a)
{
    case '0':
        instrucción 0;
        break;

    case '1':
        instrucción 1;
        break;
}

```

```

        case '2':
            instrucción 2;
            break;

        case '3':
            instrucción 3;
    }
}

```

9.3.3.3 La instrucción GOTO Siempre podremos evitar las instrucciones GOTO si utilizamos las estructuras aquí presentadas. Algunas veces una instrucción GOTO proporciona un método fácil para terminar una estructura cuando ocurren errores; sin embargo, extreme las precauciones. Las instrucciones GOTO a menudo se convierten en instrucciones de salto incondicionales cuando el programa está codificado en lenguaje ensamblador. Por ejemplo, se producirá un problema si entramos a una subrutina de la manera común (mediante la instrucción CALL para llamar a la subrutina), pero salimos empleando una instrucción de salto en lugar de una instrucción de regreso de la subrutina. La dirección de regreso quedará en la pila y, en un momento dado, ocurrirá un desbordamiento de la pila.

9.4 SINTAXIS DEL SEUDOCÓDIGO

Debido a que el seudocódigo es parecido a un lenguaje de alto nivel como Pascal o C, vale la pena definirlo un poco más formalmente para que, por ejemplo, un programador pueda escribir un programa en seudocódigo tal que otro programador lo pueda convertir a lenguaje ensamblador.

Debemos reconocer también que el seudocódigo no siempre es la mejor manera de diseñar programas. Aunque ofrece la ventaja de una fácil construcción en un procesador de palabras (con las modificaciones apropiadas), el seudocódigo sufre de la desventaja que es común a otros lenguajes de programación: los programas en seudocódigo se escriben línea por línea, así que las operaciones en paralelo no se hacen evidentes de inmediato. Con los diagramas de flujo, por otra parte, podemos colocar las operaciones en paralelo físicamente adjuntas con las otras operaciones, lo cual mejora el modelo conceptual (consulte la figura 9-10).

Antes de presentar una sintaxis formal, le haremos las siguientes sugerencias para aumentar el poder de resolución de problemas de programación utilizando seudocódigo:

- Utilice un lenguaje descriptivo para las instrucciones.
- Evite depender de ciertas máquinas o sistemas en las instrucciones.
- Encierre las condiciones o instrucciones entre corchetes: [].
- Comience todas las subrutinas con sus nombres seguidos por un conjunto de paréntesis: (). Los parámetros transferidos a las subrutinas se escriben (por nombre o por valor) dentro de los paréntesis.
- Termine todas las subrutinas con un RETURN seguido de paréntesis. Los valores de regreso se escriben dentro de los paréntesis.

Ejemplos de subrutinas:

CARENT()	CARSAL(char)	STRLEN(apuntador)
[instrucción]	[instrucción]	[instrucción]
...
RETURN(char)	RETURN()	RETURN(tamaño)

- Utilice el texto en minúsculas, excepto en palabras reservadas y nombres de las subrutinas.
- Todas las instrucciones deben tener sangría en relación con los puntos de entrada y de salida de la estructura. Cuando se inicia una estructura de CICLO o de OPCION, las instrucciones que están dentro de la estructura aparecen en el siguiente nivel de sangría.
- Utilice el signo de arroba (@) para codificar el direccionamiento indirecto.

A continuación presentamos una sintaxis sugerida para el seudocódigo:

Palabras reservadas:

```
BEGIN      END
REPEAT    UNTIL
WHILE     DO
IF        THEN          ELSE
CASE      OF
RETURN
```

Operadores aritméticos:

```
+  suma
-  resta
*  multiplicación
/  división
%  módulo (resto después de una división)
```

Operadores relacionales: (resultados de verdadero o falso)

```
==  verdadero si los valores son iguales uno al otro
!=  verdadero si los valores no son iguales
<  verdadero si el primer valor es menor al segundo
<=  verdadero si el primer valor <= al segundo valor
>  verdadero si el primer valor > al segundo valor
>=  verdadero si el primer valor >= al segundo valor
&&  verdadero si ambos valores son verdaderos
||  verdadero si cualquier valor es verdadero
```

Operadores lógicos a nivel de bit:

```
&  AND lógico
|  OR lógico
^  OR exclusivo lógico
-  NOT lógico (complemento a unos)
>>  desplazamiento lógico a la derecha
<<  desplazamiento lógico a la izquierda
```

Operadores de asignación:

```
=  igualarlo al valor
op =  forma corta para la operación de asignación, en donde
       "op" es uno de + - * / % << >> & ^ |
       por ejemplo: j += 4 es equivalente a j = j + 4
```

Operación de precedencia:

()

Dirección indirecta:

@

Precedencia de los operadores:

()				
~	@			
*	/	%		
+	-			
<<	>>			
<	<=	>	>=	
==	!=			
&				
^				
&&				
=	+=	-=	*=	etc.

Nota 1. No confunda los operadores relacionales con los operadores lógicos a nivel de bit. Los operadores lógicos a nivel de bit, por lo general, se utilizan en instrucciones de asignación tales como

[nibble_inferior = byte & 0FH]

mientras que los operadores relacionales por lo común se utilizan en expresiones condicionales del tipo

IF [carácter != 'Q' && carácter != 0DH] THEN . . .

Nota 2. No confunda al operador relacional “==” con el operador de asignación “=”. Por ejemplo, la expresión booleana

j == 9

es verdadera o falsa dependiendo de si j es igual al valor 9 o no, mientras que la instrucción de asignación

j = 9

asigna el valor 9 a la variable j.

Estructuras:

Instrucción:

[haz algo]

Bloque de instrucciones:

BEGIN
[instrucción]

```

    [instrucción]
    .
    .
    END

```

WHILE/DO:

```

WHILE [condición] DO
    [instrucción]

```

REPEAT/UNTIL:

```

REPEAT
    [instrucción]
UNTIL [condición]

```

IF/THEN/ELSE:

```

IF [condición]
    THEN [instrucción 1]
    (ELSE [instrucción 2])

```

CASE/OF:

```

CASE [expresión] OF
    1:[instrucción 1]
    2:[instrucción 2]
    3:[instrucción 3]
    .
    .
    .
    n:[instrucción n]
    [instrucción predeterminada]
END_CASE

```

9.5 ESTILO DE PROGRAMACIÓN EN LENGUAJE ENSAMBLADOR

Es fundamental adoptar un estilo claro y consistente en la programación escrita en lenguaje ensamblador. Esto resulta particularmente importante cuando estamos trabajando como parte de un equipo, ya que los miembros individuales deben ser capaces de leer y entender los programas de los demás integrantes.

Las soluciones que hemos presentado hasta ahora en lenguaje ensamblador para los problemas planteados han sido deliberadamente un bosquejo. Para tareas de programación grandes, sin embargo, requerimos de un enfoque más crítico. Ofrecemos las siguientes sugerencias para ayudarle a mejorar el estilo de programación en lenguaje ensamblador.

9.5.1 Etiquetas

Utilice etiquetas que describan el destino que representan. Por ejemplo, cuando emplee una bifurcación hacia atrás en forma repetitiva para realizar una operación, utilice una etiqueta del tipo “CICLO”, “ATRAS”, “MAS”, etc. Cuando ignore ciertas instrucciones del programa, utilice

etiquetas como “IGNORA” o “ADELANTE”. Al verificar en forma repetitiva un bit de estado, utilice una etiqueta de “ESPERE” o “DENUEVO”.

Las elección de las etiquetas es un tanto restringida ya sea que utilicemos un simple ensamblador residente en memoria o un ensamblador absoluto. Estos ensambladores tratan al programa completo como una unidad, lo cual limita el uso de etiquetas comunes. Existen varias técnicas para resolver este problema. Las etiquetas comunes pueden enumerarse secuencialmente, tal como IGNORA1, IGNORA2, IGNORA3, etc.; o, dentro de las subrutinas, tal vez todas las etiquetas puedan utilizar el nombre de la subrutina seguido de un número, digamos, ENVIA, ENVIA2, ENVIA3, etc. Existe una evidente pérdida de claridad aquí, ya que las etiquetas ENVIA2 y ENVIA3 no reflejan las acciones de ignorar o de ciclo que se están llevando a cabo.

Ensambladores más sofisticados, tales como el ASM51, permiten que cada subrutina (o un grupo común de subrutinas) exista como un archivo separado que se ensambla independientemente del programa principal. El programa principal también se ensambla por sí solo y se combina con las subrutinas utilizando a un programa enlazador y localizador el cual, entre otras cosas, resuelve las referencias externas entre los archivos. Este tipo de ensamblador, a menudo llamado ensamblador “relocalizable”, permite que la misma etiqueta aparezca en diferentes archivos.

9.5.2 Comentarios

No podemos enfatizar demasiado el uso de comentarios, especialmente en la programación en lenguaje ensamblador, la cual es inherentemente abstracta. Todas las líneas de código, a excepción de aquellas con acciones claramente evidentes, deben incluir un comentario.

Podemos hacer comentarios efectivos para las instrucciones de salto condicionales, utilizando una pregunta similar a la incluida en el diagrama de flujo para una operación similar. Las respuestas de “sí” y “no” a la pregunta deben aparecer en los comentarios en las líneas que representan las acciones de “saltar” y “no saltar”. Por ejemplo, en la subrutina LINEAENT siguiente, advierta el estilo de los comentarios utilizados para hacer una prueba mediante el código de retorno de carro <CR>.

```
; ****
;
;LINEAENT    ENTRADA DE LINEA DE CARACTERES
;           LA LINEA DEBE TERMINAR CON <CR>
;           MAXIMA LONGITUD DE 31 CARACTERES INCLUYE <CR>
;
;ENTRADA:    NO HAY CONDICIONES
;SALIDA:     CODIGOS EN ASCII EN MEMORIA RAM INTERNA PARA DATOS
;           0 ALMACENADO EN EL FIN DE LINEA
;USA        CARENT, CARSAL
;
; ****
;
LINEAENT:    PUSH 00H          ;ALMACENA R0 EN LA PILA
              PUSH 07H          ;ALMACENA R7 EN LA PILA
              PUSH ACC           ;ALMACENA EL ACUMULADOR EN LA
                           ;PILA
              MOV   R0 ,#60H       ;ESTABLECE BUFER EN 60H
              MOV   R7 ,#31         ;MAXIMA LONGITUD DE LINEA
INSTRUCCIÓN: ACALLINCHAR      ;ENTRADA DE CARACTER
              ACALLOUTCHR        ;PRESENTA A LA CONSOLA
```

```

MOV  @R0,A          ;ALMACENA EN BUFER
INC  R0             ;INCREMENTA APUNTADOR DE BUFER
DEC  R7             ;DISMINUYE CONTADOR DE LONGITUD
CJNE A,#0DH, IGNORA ;¿CARACTER = <CR>?
SJMP SALIDA         ;SI: SALIDA
IGNORA: CJNE R7,#0, INSTRUCCIÓN ;NO: OBTIENE OTRO CARACTER
SALIDA: MOV @R0,#0   ;RECUPERA REGISTROS DE
              POP ACC           ;LA PILA
              POP 07H
              POP 00H
              RET

```

9.5.3 Bloques de comentarios

Las líneas de comentarios son esenciales al inicio de cada subrutina. Las subrutinas deben ser de propósito general y estar bien documentadas, debido a que llevan a cabo tareas bien definidas que por lo general se requieren durante la ejecución de un programa. Cada subrutina va precedida por un **bloque de comentarios**, el cual es una serie de líneas de comentarios que establecen en forma explícita:

- El nombre de la subrutina
- La operación realizada
- Las condiciones de entrada
- Las condiciones de salida
- El nombre de otras subrutinas utilizadas (si las hay)
- El nombre de los registros afectados por la subrutina (si los hay)

La subrutina LINEAENT anterior es un buen ejemplo de una subrutina bien documentada.

9.5.4 Almacenamiento de registros en la pila

Conforme las aplicaciones aumentan en tamaño y complejidad, a menudo escribimos nuevas subrutinas que agregan y utilizan otras subrutinas existentes. Por lo tanto, las subrutinas llaman a otras subrutinas, las cuales llaman a otras subrutinas, y así sucesivamente. Estas son las llamadas “subrutinas anidadas”. No hay peligro alguno en utilizar las subrutinas anidadas, siempre y cuando la pila tenga espacio suficiente para almacenar las direcciones de regreso. Esto no constituye un problema, ya que es raro anidar más allá de cierto número de niveles.

Sin embargo, existe un problema potencial en el uso de los registros dentro de las subrutinas. Esto es, al tiempo que la jerarquía de subrutinas aumenta, se vuelve más y más difícil estar al tanto de cuáles registros son afectados por las subrutinas. Una sólida práctica de programación es, en consecuencia, la de almacenar los registros que son alterados por una subrutina en la pila y recuperarlos al final de la subrutina. Observe que la subrutina LINEAENT que presentamos antes almacena y recupera R0, R7, y al acumulador utilizando la pila. Cuando LINEAENT regresa de la subrutina que hizo la llamada, estos registros contienen el mismo valor que tenían cuando LINEAENT fue llamada.

9.5.5 El uso de igualdades

La definición de constantes mediante instrucciones de igualdad permite que los programas sean más fáciles de leer y mantener. Las igualdades aparecen al comienzo de un programa para definir

constantes, tales como el retorno de carro (<CR>) y el salto de línea (<LF>) o las direcciones de registros dentro de circuitos integrados periféricos como los de STATUS o CONTROL.

La constante puede utilizarse a lo largo del programa sustituyendo el símbolo igualado por el valor. El valor se sustituye por el símbolo cuando el programa se ensambla. El uso generoso de las igualdades permite que un programa sea más fácil de mantener y lo vuelve más legible. Sólo debemos cambiar una línea cuando tenemos que cambiar una constante, y es la línea donde el símbolo se está igualando. El nuevo valor se sustituye en forma automática dondequiera que el símbolo se esté utilizando cuando el programa se vuelve a ensamblar.

9.5.6 El uso de subrutinas

Conforme un programa aumenta en tamaño, es importante “dividir y conquistar”; esto es, subdividir operaciones grandes y complejas en operaciones pequeñas y simples. Estas operaciones pequeñas y simples se programan como subrutinas. Las subrutinas son jerárquicas ya que subrutinas simples pueden ser utilizadas por subrutinas más complejas, y así sucesivamente.

Un diagrama de flujo hace referencia a una subrutina mediante el bloque de “proceso predefinido”. (Consulte la figura 9-1). El uso de este símbolo indica que otro diagrama de flujo incluido en alguna otra parte describe los detalles de la operación.

Las subrutinas se construyen en seudocódigo como secciones completas de código, comenzando con sus nombres y los paréntesis. Dentro de estos paréntesis se escriben los nombres o valores de los parámetros transferidos a la subrutina (si los hay). Cada subrutina termina con la palabra clave RETURN seguida de paréntesis, los cuales contienen el nombre o valor de los parámetros regresados por la subrutina (si los hay).

Tal vez los ejemplos más simples de la jerarquía de las subrutinas son los de cadena de salida (CADSAL) y carácter de salida (CARSAL). La subrutina CADSAL (una rutina de alto nivel) llama a la subrutina CARSAL (una rutina de bajo nivel).

A continuación mostramos los diagramas de flujo, el seudocódigo, y las soluciones en lenguaje ensamblador y lenguaje C del 8051.

Seudocódigo:

```
CARSAL (carácter)
    [coloca paridad impar en el bit 7]
    REPEAT [prueba búfer de transmisión]
        UNTIL [búfer vacío]
            [limpia bandera vacía de búfer de transmisión]
            [transfiere carácter al búfer de transmisión]
            [limpia bit de paridad]
    RETURN ()
    OUTSTR (apuntador)
        WHILE [(carácter = @apuntador) !=0] BEGIN
            CARSAL (carácter)
            [incrementa apuntador]
        END
    RETURN ()
```

Lenguaje ensamblador:

CARSAL:	MOV	C, P	; COLOCA BIT DE PARIDAD EN BANDERA C
	CPL	C	; CAMBIA A PARIDAD IMPAR
	MOV	ACC.7,C	; AGREGA A CARACTER
DENUEVO:	JNB	TI,DENUEVO	; TX VACIO?

```

CLR    TI          ;SI: LIMPIA BANDERA Y
MOV    SBUF,A      ; ENVIA CARACTER
CLR    ACC.7       ;REMUEVE BIT DE PARIDAD Y
RET
CADSAL: MOV   A,@DPTR   ;OBTEN CARACTER
        JZ    SALIDA    ;SI 0, TERMINAMOS
        CALL  CARSAL    ;SI NO LO ENVIAMOS
        INC   DPTR      ;INCREMENTA APUNTADOR
        SJMP CADSAL    ; Y OBTEN SIGUIENTE CARACTER
SALIDA: RET

```

Diagrama de flujo:

FIGURA 9-12

Diagrama de flujo para la subrutina
CARSAL

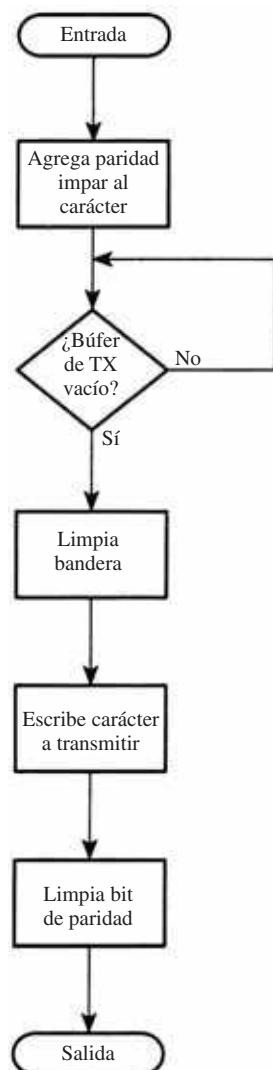
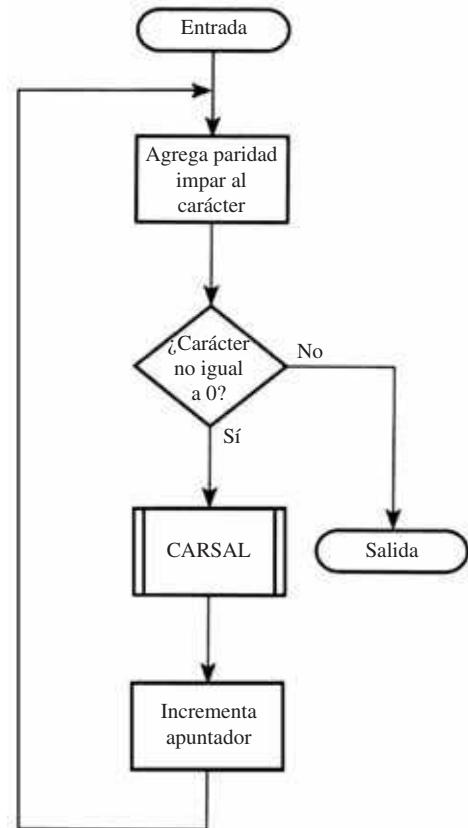


FIGURA 9–13

Diagrama de flujo para la subrutina CADSAL



Lenguaje C del 8051:

```

sbit BMSAcum = ACC ^ 7;
void carsal (char a)
{
    ACC = a;
    CY = P;
    CY = !CY;
    BMSAcum = CY;
    while (TI != 1);
    TI = 0;
    SBUF = ACC;
    BMSAcum = 0;
}
void cadsal(char * msg)
{
    while (*mensaje != '\0')
        carsal(*s++);
}
  
```

9.5.7 Organización de programas

Aunque los programas a menudo se escriben por partes (en otras palabras, las subrutinas se escriben en forma separada del programa principal), todos deben ser consistentes en su organización final. En general, las secciones de un programa se ordenan como sigue:

- Igualdades
- Instrucciones de inicialización
- Cuerpo principal del programa
- Subrutinas
- Definiciones de constantes de datos (DB y DW)
- Ubicaciones de datos en RAM utilizando la directiva DS

Todas estas secciones se llaman “segmento de código” salvo las ubicaciones de datos en RAM, conocidas como el “segmento de datos”. Los segmentos de código y de datos tradicionalmente van separados debido a que el código por lo general está destinado a ubicarse en memoria ROM o EPROM, mientras que los datos en RAM siempre están destinados a ubicarse en la memoria RAM. Observe que las constantes de datos y las cadenas definidas mediante las directivas DB o DW son parte del segmento de código (no del segmento de datos), ya que estos datos son constantes que no cambian y, por lo tanto, forman parte del programa.

9.6 ESTILO DE PROGRAMACIÓN EN LENGUAJE C DEL 8051

Cuando programamos en lenguaje C del 8051, resulta aún más necesario ser claro y consistente. Esto es porque los programadores a menudo escriben en C cuando sus programas en el 8051 son grandes y complejos, e involucran a varios programadores que trabajan en el proyecto al mismo tiempo. Debemos observar lo siguiente cuando programamos en C del 8051.

9.6.1 Comentarios

En C, los comentarios van enmarcados por los símbolos /* y */. Por ejemplo, el programa con comentarios “Hola programador” se muestra a continuación:

```
main ()
{
    SCON = 0x52;          /* puerto serial, modo 1 */
    TMOD = 0x20;          /* temporizador 1, modo 2 */
    TH1  = -13;           /* Conteo de recarga para 2400 baudios */
    TR1  = 1;              /* inicia temporizador 1 */

    while (1)             /* repite infinitamente */
    {
        printf ("Hello World\n"); /* Muestra "Hello World" */
    }
}
```

9.6.2 El uso de definiciones

Usted tiene que definir constantes con alias para que su programa sea más legible y facilite cambiar la constante en un futuro con sólo cambiar una línea de código. Por ejemplo:

```
#define PI 3.1415927
#define MAX 10

int areaCirculo(int radio)
{
    return (PI * radius * radio);
}

main()
{
    int i;
    for (i = 0; i < MAX; i++)
        areaCirculo(i);
}
```

El programa de ejemplo anterior define a `PI` como la constante $\pi = 3.1415927$, mientras que `MAX` se define como el número constante 10. `PI` se utiliza para calcular el área de un círculo, mientras que `MAX` es el máximo número de diferentes áreas de círculo que serán calculadas. Si, por ejemplo, deseamos calcular el área de 20 círculos diferentes, entonces el valor de `MAX` puede cambiarse con tan sólo cambiar la línea de definición a `#define MAX 20`.

9.6.3 El uso de funciones

Los programas en C, que son grandes y complejos, generalmente son escritos por varios programadores en forma simultánea. Esto es posible mediante el enfoque de la programación modular, en donde se dividen secciones del programa en módulos y funciones. La división del programa en funciones permite mayor modularidad y vuelve al programa más claro y fácil de seguir. Hemos adoptado este enfoque en los programas de ejemplo de interfaz y diseño que veremos en el capítulo 12.

9.6.4 El uso de arreglos y apuntadores

Cuando deseamos almacenar una secuencia de ciertos datos relacionados, es buena idea utilizar arreglos y tener apuntadores para apuntar a los arreglos. Recuerde que los apuntadores almacenan ubicaciones en memoria. Por lo tanto, la dirección del primer elemento de un arreglo puede asignarse a un apuntador. Con esto podemos acceder a cada elemento del arreglo con tan sólo añadir un desplazamiento apropiado al valor del apuntador y hacer una desreferencia. Como ejemplo, las tablas de búsqueda pueden ser implementadas en forma óptima mediante el uso de arreglos

Los arreglos también resultan muy útiles cuando almacenamos cadenas. Por ejemplo, el mensaje “Este es un mensaje de bienvenida” puede almacenarse en memoria como un arreglo con tan sólo utilizar la línea: `char * MENSAJE = {"Este es un mensaje de bienvenida."}`, la cual utilizaría un arreglo de caracteres para almacenar la cadena.

9.6.5 Organización de los programas

Con el propósito de estandarizar la distribución y la organización, hemos adoptado la siguiente organización de programas cuando los escribimos en lenguaje C del 8051:

- "Includes" (inclusión de cabeceras)
- Definiciones de constantes
- Definiciones de variables
- Prototipos de funciones
- Función principal
- Definiciones de funciones

Las definiciones de constantes son líneas tales como `#define PI 3.1415927`, mientras que las definiciones de variables son del tipo `char A`. Los prototipos de funciones son instrucciones de una sola línea, digamos `void HTOA(void)`, la cuales consisten en el nombre de la función, el tipo del regreso, y la lista de parámetros. Las definiciones de funciones son, en tanto, las definiciones completas de las operaciones realizadas en las funciones, por ejemplo:

```
void HTOA(void)
{
    A = A & 0xF;
    if (A>=0xA)
        A = A + 7;
    A = A + '0';
}
```

RESUMEN

En este capítulo presentamos las técnicas de programación estructurada que se desarrollan mediante diagramas de flujo y seudocódigo. Ofrecemos sugerencias para aumentar la legibilidad del diseño y de la presentación de los programas. En el siguiente capítulo estudiaremos algunas herramientas y técnicas muy útiles para desarrollar programas.

PROBLEMAS

- 9.1** Ilustre una estructura WHILE/DO tal que se ejecute un bloque de instrucciones mientras el acumulador sea menor o igual a 7EH.
- 9.2** Ilustre una estructura WHILE/DO utilizando la siguiente condición compleja: (acumulador mayor a 0 y R7 mayor a 0) o la bandera de acarreo igual a 1. Trate los valores incluidos en el acumulador y en R7 como enteros sin signo.
- 9.3** Escriba una subrutina para encontrar el lugar del 1 más significativo en el acumulador. Regresa con el "lugar" en R7. Por ejemplo, si ACC = 00010000B, regrese con R7 = 4.
- 9.4** Escriba una subrutina llamada LINEAENT para obtener como entrada una línea de caracteres de la consola y colocarla en la memoria interna, empezando en la ubicación 60H. La longitud máxima de una línea es de 31 caracteres, incluyendo al retorno de carro. Coloque un 0 al final de la línea.

Herramientas y técnicas para el desarrollo de programas

10.1 INTRODUCCIÓN

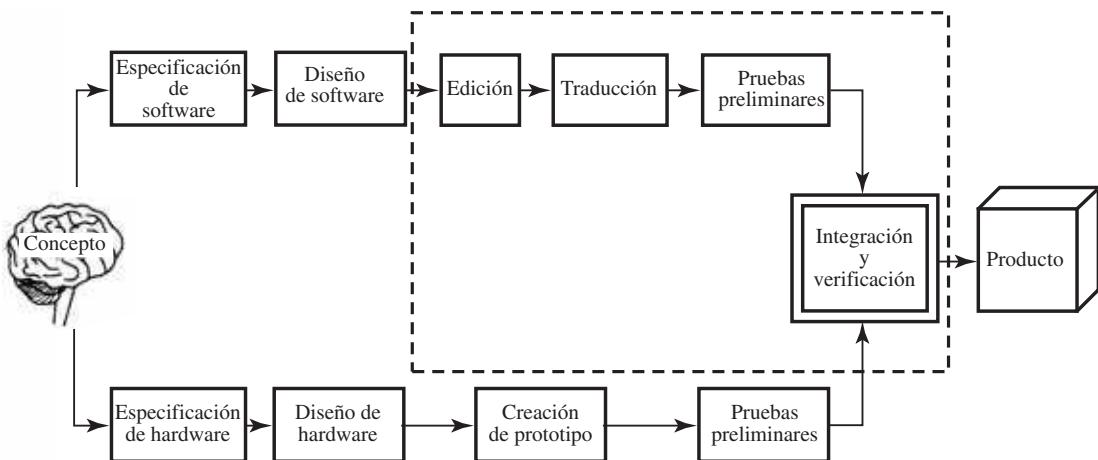
En este capítulo describiremos el proceso de desarrollo de productos basados en microcontroladores o microprocesadores, para lo cual seguiremos una serie de pasos y utilizaremos una diversidad de herramientas. En el transcurso que va del concepto al producto, se involucran muchos pasos y se utilizan numerosas herramientas. A continuación le presentaremos los pasos y las herramientas más comunes que se encuentran en escenarios típicos de diseño donde se emplea el microcontrolador 8051.

El diseño es una actividad altamente creativa, y en reconocimiento de ello declaramos que se requiere cierta independencia sustancial, ya sea que diseñemos individualmente o como integrantes de equipos de desarrollo. Sin embargo, es posible que tal autonomía sea difícil de alcanzar en proyectos bastante grandes o donde la seguridad resulta crítica. Admitimos que en tales ambientes el manejo del proceso y la validación de los resultados deben satisfacer un orden más elevado. En este capítulo trabajaremos en el desarrollo de productos de escala relativamente pequeña, tales como controladores para hornos de microondas, paneles de instrumentos para automóvil, periféricos para computadora, máquinas de escribir electrónicas, y equipo de audio de alta fidelidad.

Presentaremos y trabajaremos con los pasos requeridos y las herramientas y técnicas disponibles, también daremos los ejemplos apropiados. Es importante desarrollar la comprensión de los pasos, pero no sugerimos adherirse estrictamente a su secuencia. Sentimos que forzar un proceso de desarrollo a través de actividades ordenadas y aisladas a menudo resulta extenuante y quizás sea equivocado. Más adelante en este capítulo presentaremos un escenario de desarrollo tipo todo incluido, en donde conocemos los recursos y los utilizamos siguiendo siempre el instinto del diseñador. Empecemos con el examen de los pasos necesarios en el ciclo de desarrollo.

10.2 EL CICLO DE DESARROLLO

El proceso de pasar del concepto al producto a menudo se muestra en un diagrama de flujo conocido como **ciclo de desarrollo**, parecido al que se muestra en la figura 10-1. El lector podrá observar que no hay nada particularmente “cíclico” en los pasos mostrados. De hecho, la figura presenta un

**FIGURA 10–1**

El ciclo de desarrollo

escenario ideal e imposible que no tiene “separaciones”. Claro que existen problemas. Necesitamos de la **depuración** (encontrar y corregir problemas) en cada paso del ciclo de desarrollo para introducir correcciones y volver a realizar una actividad previa. Dependiendo de la severidad del error, la corrección puede resultar trivial o, en caso extremo, obligar a que el diseñador regrese a la etapa de concepto. Por lo tanto, existe una conexión implicada en la figura 10-1 desde la salida de cualquier paso incluido en el ciclo de desarrollo hasta cualquier otro paso.

Los pasos localizados a lo largo de la parte superior de la figura 10-1 corresponden al desarrollo de software, mientras que los de la parte inferior corresponden al desarrollo de hardware. Las dos partes se juntan en un paso crítico y complicado llamado “integración y verificación”, el cual lleva a la aceptación del diseño como un “producto”. Los diversos pasos subsiguientes a la aceptación del diseño no se muestran en la figura. Éstos incluyen, por ejemplo, manufactura, pruebas, distribución y comercialización. En la figura 10-1, la línea punteada abarca los pasos de mayor importancia a tratar en este capítulo (y en todo el libro). Más tarde trabajaremos con mayor detalle sobre estos pasos. Pero primero, empecemos por examinar los pasos a seguir en el desarrollo de software.

10.2.1 Desarrollo de software

En esta sección estudiaremos los pasos incluidos en la parte superior de la figura 10-1, comenzando con la especificación del software de aplicación.

Especificación de software. La especificación de software es la tarea de establecer explícitamente lo que el software va a hacer. Esto se puede llevar a cabo en varias formas. A nivel superficial, las especificaciones pueden tratar primero sobre la interfaz para el usuario; esto es, cómo interactuará el usuario con el sistema y lo controlará. (¿Qué efectos resultarán y se observarán para cada acción tomada?). Si se utilizan interruptores, perillas o indicadores auditivos o visuales en el hardware prototípico, entonces debemos establecer el propósito explícito y la operación para cada uno de tales dispositivos.

Los científicos computacionales han definido métodos formales para especificar los requerimientos de software; sin embargo, a menudo no se utilizan en el diseño de aplicaciones basadas en microcontroladores, las cuales son pequeñas en comparación con el software de aplicación destinado a las supercomputadoras.

Las especificaciones de software pueden tratar también sobre detalles de operación del sistema situados por debajo del nivel del usuario. Por ejemplo, un controlador para fotocopiadora puede monitorear las condiciones internas necesarias para realizar una operación normal o segura, tales como temperatura, corriente, voltaje o movimiento del papel. Estas condiciones son, en gran parte, independientes de la interfaz para el usuario, pero aún así el software debe tratarlas.

Las especificaciones pueden estar distribuidas en módulos con base en cada función del sistema, siendo las condiciones de entrada y salida muy bien definidas para permitir la comunicación entre los módulos. Las técnicas descritas en los capítulos anteriores para la documentación de subrutinas son un primer paso razonable a seguir en la especificación de software.

Los sistemas controlados por interrupciones requieren de una cuidadosa planeación y tienen características únicas que deben considerarse en la etapa de especificación. Las actividades con requisitos, en donde el tiempo no es crítico, pueden colocarse en el ciclo de primer plano o en una secuencia cíclica (round-robin) para manejar las interrupciones sincronizadas. Las actividades donde el tiempo es crítico generan interrupciones de alta prioridad que toman el control del sistema para su manejo inmediato; por ello, las especificaciones de software pueden enfatizar el tiempo de ejecución necesario en dichos sistemas. ¿Qué tanto tarda cada subrutina o rutina de servicio de interrupciones (ISR) en ejecutarse? ¿Qué tan frecuente se ejecuta cada ISR? Las ISR que se ejecutan de manera asíncrona (en respuesta a un evento) pueden tomar el control del sistema en cualquier momento; así que tal vez sea necesario bloquearlas en ciertas instancias o interrumpirlas en otras. Las especificaciones de software para dichos sistemas deben considerar los niveles de prioridad, las secuencias de sondeo, y la posibilidad de una reasignación dinámica de los niveles de prioridad o de las secuencias de sondeo dentro de las ISR.

Diseño de software. El diseño de software es una tarea en donde los diseñadores se involucrarán sin mucha planeación. Existen dos técnicas comunes para efectuar el diseño de software antes de la codificación: diagramas de flujo y seudocódigo. Éstos fueron el tema del capítulo 9.

Edición y traducción. La edición y la traducción de software ocurren, por lo menos al principio, en un ciclo restringido. Los errores detectados por el ensamblador se corrigen rápidamente por medio de la edición del archivo fuente, y ensamblándolo de nuevo. Debido a que el ensamblador no tiene idea del propósito del programa y verifica únicamente errores “gramaticales” (por ejemplo, si faltan comas, si hay instrucciones indefinidas), los errores detectados de esta forma son **errores de sintaxis**. Éstos se conocen también como **errores en tiempo de ensamblado**.

Pruebas preliminares. Un **error en tiempo de ejecución** no aparecerá sino hasta que el programa se ejecute mediante un simulador, o en el sistema destino. Este tipo de error puede resultar elusivo, y puede requerir de una cuidadosa observación de la actividad de la CPU en cada etapa del programa. Un **depurador** es un programa del sistema que ejecuta un programa de usuario con el propósito de encontrar errores al tiempo de ejecución. El depurador incluye características tales como la de poder ejecutar un programa hasta llegar a cierta dirección (un **punto de interrupción**), la **ejecución paso a paso** a través de instrucciones mientras se presentan los registros de la CPU, los bits de estado o los puertos de entrada/salida.

10.2.2 Desarrollo de hardware

En su mayor parte, este libro no enfatiza el desarrollo de hardware. Debido a que el 8051 es un dispositivo altamente integrado, nos hemos enfocado en el aprendizaje de su arquitectura interna y la utilización de los recursos incorporados al chip a través de software. Los ejemplos presentados hasta ahora han utilizado sólo interfaces simples a componentes externos.

Especificación de hardware. La especificación de hardware involucra la asignación de datos cuantitativos a las funciones del sistema. Por ejemplo, un proyecto para un brazo automatizado debe especificarse en términos de articulaciones, alcance, velocidad, precisión, momento de torsión, requerimientos de energía, y así sucesivamente. A menudo se requiere que los diseñadores proporcionen una hoja de especificaciones análoga a las que acompañan a un amplificador de audio o a una reproductora de video. Otras especificaciones de hardware incluyen tamaño físico y peso, velocidad de la CPU, cantidad y tipo de memoria, asignaciones de los mapas de memoria, puertos de E/S, características opcionales, etcétera.

Diseño de hardware. El método convencional para efectuar el diseño de hardware, utilización de un lápiz y una plantilla lógica, todavía se aplica ampliamente, pero puede mejorarse mediante uso del software para diseño asistido por computadora (CAD). Aunque las herramientas CAD son más empleadas en las disciplinas mecánicas o ingeniería civil, algunas están orientadas específicamente hacia la ingeniería electrónica. Los dos ejemplos más comunes son las herramientas disponibles para trazar diagramas esquemáticos y diseñar tarjetas de circuito impresas (PCB). Aunque estos programas tienen una larga curva de aprendizaje, los resultados son impresionantes. Algunos programas para trazado de diagramas esquemáticos producen archivos que los programas PCB pueden leer para generar el diseño de un circuito.

Construcción del prototipo. Existe un patéticamente escaso número de atajos para efectuar las labores de construcción de prototipos. Ya sea que se trate de construir una simple interfaz a un bus o a un conector de puerto en una computadora de una sola tarjeta (SBC), o del alambrado entero de una tarjeta controladora, las técnicas para la construcción de prototipos sólo pueden desarrollarse con una gran cantidad de práctica. Compañías grandes con presupuestos enormes pueden proceder directamente al formato de tarjetas de circuito impresas, inclusive para la primera iteración del diseño de hardware. Sin embargo, los proyectos llevados a cabo por compañías pequeñas, estudiantes o aficionados son quizás más propensos a utilizar el método común de alambrado para los prototipos.

Pruebas preliminares. La primera prueba del hardware se realiza sin ningún software de aplicación. Una prueba paso por paso es importante: no hay razón para medir la señal de sincronización utilizando un osciloscopio antes de haber verificado los voltajes presentes en la fuente de poder.

- Verificaciones visuales
- Verificaciones de continuidad
- Mediciones de corriente directa
- Mediciones de corriente alterna

Las verificaciones visuales y de continuidad deben ocurrir antes de suministrar energía a la tarjeta. Las verificaciones de continuidad mediante el uso de un ohmmetro deben realizarse desde la parte de los circuitos integrados del prototipo, de terminal a terminal. De esta manera, las conexiones de la terminal de IC al enchufe y de la terminal del enchufe a los alambres se verifican al mismo tiempo. Los voltajes de corriente directa deben verificarse en toda la tarjeta utilizando un voltímetro. Por último, las mediciones de corriente alterna se realizan con los circuitos integrados instalados para verificar señales de sincronización, y así sucesivamente.

Después de verificar conexiones, voltajes y señales de sincronización, la depuración se convierte en una actividad pragmática: ¿El prototipo está funcionando como se planeó? Si no, una acción para corregirlo podría regresar al diseñador a la construcción, el diseño o la especificación del hardware.

Si el diseño es el de un sistema completo con una CPU, un solo error de alambrado puede evitar que la CPU complete su secuencia de reinicialización: la primera instrucción después de la reinicialización tal vez nunca se execute. Un truco eficaz para realizar la depuración es el de controlar la línea de reinicialización de la CPU con una onda cuadrada de baja frecuencia (\approx kHz)

y observar (mediante un osciloscopio o un analizador lógico) la actividad de los buses inmediatamente después de la reinicialización.

Las pruebas funcionales de la tarjeta pueden requerir de software de aplicación o un programa de monitoreo para “trabajar” a la tarjeta a través de sus movimientos. En esta etapa es cuando el software debe auxiliar para completar el ciclo de desarrollo.

10.3 INTEGRACIÓN Y VERIFICACIÓN

La etapa más difícil en el ciclo de desarrollo ocurre cuando el hardware se encuentra con el software. Algunos errores muy sutiles que evadieron la simulación (si ésta se llevó a cabo) emergen durante la ejecución en tiempo real. El problema se complica aún más porque se necesita todo un conjunto de recursos: hardware tal como un sistema de desarrollo de PC, un sistema destino, fuente de poder, cables y equipo de prueba; y software como un programa de monitoreo, sistema operativo, un programa de emulación de terminal, y así sucesivamente.

Vamos a trabajar más sobre el paso de integración y verificación, pero primero ampliaremos el área incluida dentro de la línea punteada en la figura 10-1. (Consulte la figura 10-2).

La figura 10-2 muestra programas y herramientas de desarrollo dentro de círculos, archivos de usuario dentro de cuadrados, y “ambientes de desarrollo” dentro de cuadrados con doble línea. El uso de un editor para crear un archivo fuente es simple y directo. El paso de traducción (de la figura 10-1) se muestra en dos etapas. Un ensamblador (por ejemplo, el ASM51) convierte un archivo fuente en un archivo de objeto, y el enlazador/localizador (por ejemplo, el RL51) combina uno o más archivos de objeto relocalizables en un solo archivo de objeto absoluto para que sea ejecutado dentro de un sistema destino o un simulador. El ensamblador y el enlazador/localizador también crean archivos de listado.

Los sufijos más comunes para identificar los archivos se muestran entre paréntesis para cada tipo de archivo. A pesar de que con frecuencia se puede proporcionar cualquier nombre de archivo y de sufijo como un argumento, los ensambladores varían en su selección de sufijos pre-determinados.

Si el programa se escribió originalmente en un solo archivo siguiendo un formato absoluto, entonces el enlazado y la localización no son necesarios. En este caso, la alternativa incluida en la figura 10-2 muestra al ensamblador generando un archivo de objeto absoluto.

Es posible (aunque no lo enfatizamos en este libro) que también se utilicen lenguajes de alto nivel, tales como C o PL/M, en lugar de, o además de, el lenguaje ensamblador. La traducción o la conversión requieren de un **compilador cruzado** para generar módulos de objetos relocalizables para el enlazado y la localización.

También puede participar un **programa bibliotecario**, tal como el LIB51 de Intel. Módulos de objetos relocalizables que son de propósito general y útiles para muchos proyectos (muy probablemente subrutinas) pueden almacenarse en “bibliotecas”. El RL51 recibe el nombre de una biblioteca como un argumento y busca en la biblioteca el código (subrutinas) correspondiente a los símbolos externos declarados previamente y que no se han resuelto hasta ese punto en el enlazamiento/localización.

10.3.1 Simulación en software

La figura 10-2 muestra cinco ambientes de ejecución. Las pruebas preliminares (consulte la figura 10-1) proceden aún en ausencia de un sistema destino. Esto se muestra en la figura 10-2 como simulación en software. Un **simulador** es un programa que se ejecuta en el sistema de desarrollo e imita la arquitectura de la máquina de destino. Un simulador del 8051, por ejemplo, contendría un registro ficticio (o “simulado”) para cada uno de los registros con funciones especiales y ubicaciones en memoria ficticias correspondientes a los espacios de memoria interna y externa del

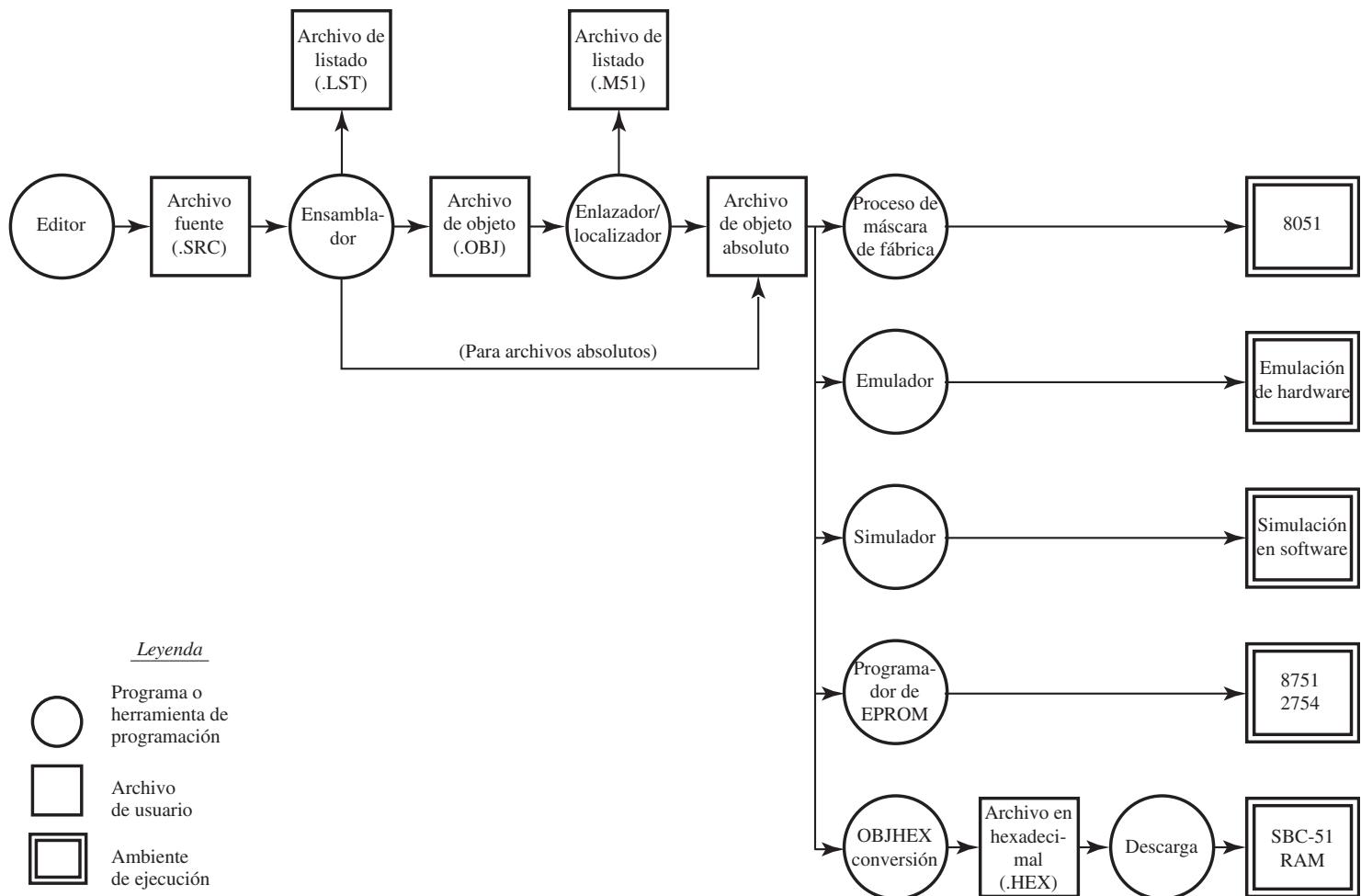


FIGURA 10-2
Detalle de los pasos involucrados en el ciclo de desarrollo

8051. Los programas se ejecutan en el modo de simulación, y el avance es presentado en la pantalla del sistema de desarrollo. Los simuladores son útiles para las pruebas iniciales; sin embargo, las porciones del programa de aplicación que manipulan directamente al hardware deben integrarse con el sistema destino para efectuar sus pruebas.

10.3.2 Emulación de hardware

Es posible establecer una conexión directa entre el sistema de desarrollo y el sistema destino a través de un **emulador de hardware** (o **emulador en circuito**). El emulador contiene un procesador que sustituye al procesador integrado al sistema destino. Sin embargo, el procesador del emulador está bajo control directo del sistema de desarrollo. Esto permite que el software se ejecute en el ambiente del sistema destino sin dejar el sistema de desarrollo. Existen comandos disponibles para ejecutar el software paso a paso, para ejecutar hasta llegar a un punto de interrupción (o la enésima ocurrencia de un punto de interrupción), y así sucesivamente. Además, la ejecución ocurre a la máxima velocidad, así que pueden salir a relucir los errores de programación dependientes del tiempo que pudieran haber eludido la depuración realizada durante una simulación.

La mayor desventaja de los emuladores de hardware es su costo. Las unidades con una PC como anfitrión se venden entre 2000 y 7000 dólares estadounidenses, lo cual está fuera del presupuesto de muchos aficionados y lleva hasta el límite los presupuestos de la mayoría de los institutos o universidades (si equipan todo un laboratorio, por ejemplo). Sin embargo, las compañías que apoyan los ambientes de desarrollo profesionales no lo pensarán dos veces para invertir en emuladores de hardware. El beneficio subyacente en la aceleración del proceso de desarrollo de un producto fácilmente justifica el costo.

10.3.3 Ejecución desde la RAM

Un escenario simple y efectivo para efectuar la prueba del software en el sistema destino sí es posible, aunque no se disponga de un emulador de hardware. Si el sistema destino contiene memoria RAM externa configurada para traslapar el espacio externo para código (utilizando el método que estudiamos en el capítulo 2; consulte la sección 2.6.4 Apuntador de datos), entonces el programa objeto absoluto puede transferirse, o “descargarse”, desde el sistema de desarrollo hacia el sistema destino y ejecutarse en el sistema destino.

Formato hexadecimal de Intel. Como se muestra en la figura 10-2, se requiere de una etapa extra de traducción para convertir el archivo de objeto absoluto al formato en ASCII estándar para su transmisión. Ya que los archivos de objeto contienen códigos binarios, éstos no pueden presentarse en pantalla o imprimirse. Esta debilidad se puede disminuir mediante la partición de cada byte binario en dos nibbles para después convertir cada nibble en su carácter en ASCII hexadecimal correspondiente. Por ejemplo, el byte 1AH no puede transmitirse a una impresora porque representa un carácter de control en ASCII en lugar de un carácter gráfico. Sin embargo, los bytes 31H y 41H sí pueden transmitirse a una impresora porque corresponden a códigos en ASCII gráficos o presentables. De hecho, estos dos bytes se imprimirán como “1A”. (Consulte el apéndice F.)

Un estándar para almacenamiento de programas en lenguaje máquina es mediante un formato presentable o imprimible conocido como “formato hexadecimal de Intel”. Un archivo hexadecimal de Intel es una serie de líneas o “registros hexadecimales” que contienen los siguientes campos:

Campo	Bytes	Descripción
Marca del registro	1	“:” indica inicio del registro
Longitud del registro	2	cantidad de bytes de datos en el registro

Dirección de carga	4	dirección de inicio para bytes de datos
Tipo de registro	2	00 = registro de datos; 01 = fin de registro
Bytes de datos	0-16	datos
Suma de validación	2	suma de todos los bytes en el registro + suma de validación = 0

Estos campos se muestran en el archivo hexadecimal de Intel ilustrado en la figura 10-3. Existen programas de conversión disponibles que reciben un programa objeto absoluto como entrada, convierten los bytes que vienen en lenguaje máquina al formato hexadecimal de Intel, y generan un archivo hexadecimal como salida. La herramienta de conversión de Intel se llama OH.

10.3.4 Ejecución desde una EPROM

Una vez que se haya obtenido un grado de rendimiento satisfactorio mediante la ejecución en memoria RAM (o por emulación en circuito), el software se quema en una EPROM e instala en el sistema como **firmware**. La figura 10-2 identifica dos tipos de EPROM como ejemplos. El 8751 es la versión EPROM del 8051, y la 2764 es una memoria EPROM común y de propósito general que se utiliza en muchos productos basados en microprocesadores o microcontroladores. Los sistemas diseñados utilizando un 8751 se benefician en cuanto a que los puertos 0 y 2 están disponibles para uso de E/S, en lugar de funcionar como buses de direcciones y de datos. Sin embargo, los EPROM 8751 son relativamente más baratos, en comparación con las memorias 2764 (\$30 dólares en lugar de \$5, por ejemplo).

10.3.5 El proceso de máscara de fábrica

Si un diseño final está destinado a la producción en masa, entonces una alternativa de bajo costo a una EPROM es la de un dispositivo de ROM con máscara de fábrica, tal como el 8051. Un 8051 es funcionalmente idéntico a un 8751; sin embargo, la memoria para código no se puede cambiar en el 8051. Los datos son grabados de modo permanente durante el ciclo de manufactura del circuito integrado utilizando una “máscara”, la cual es esencialmente un plato fotográfico que pasa o enmascara (en otras palabras, bloquea) luz durante una etapa de su manufactura. En el 8051, las

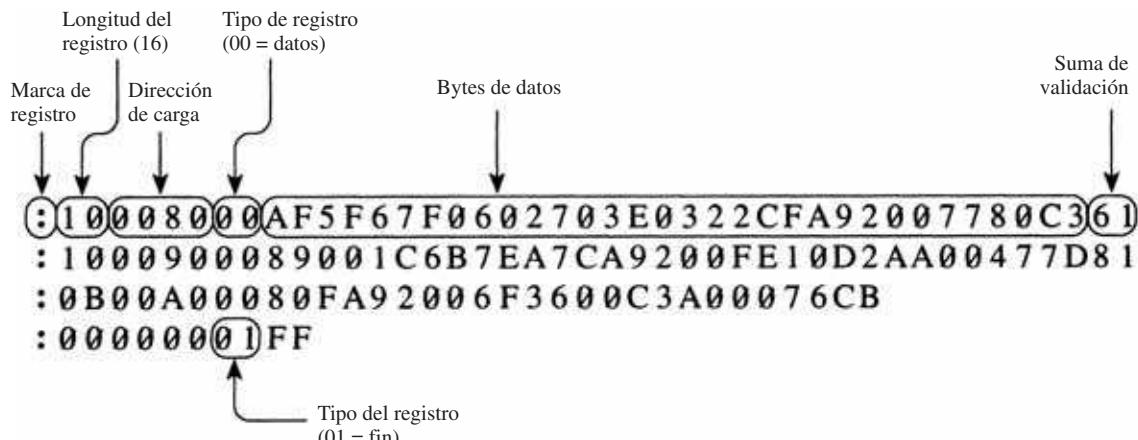


FIGURA 10-3

Formato hexadecimal de Intel

conexiones a las células de memoria se efectúan o bloquean, entonces cada célula se programa a un 1 o a un 0.

La alternativa para utilizar un 8751 en lugar de un 8051 es primordialmente económica. Un dispositivo enmascarado en fábrica es mucho más barato que el dispositivo EPROM; sin embargo, existe un gran precio inicial para producir la máscara e iniciar un ciclo de manufactura propietario. Se puede llegar a un punto de intercambio para determinar la posibilidad de cada planteamiento. Por ejemplo, si los dispositivos 8751 se venden a \$25 y los dispositivos 8051 a \$5 más un precio inicial de \$5,000, entonces el punto en que ambos planteamientos resultan iguales es

$$\begin{aligned} 25n &= 5n + 5000 \\ 20n &= 5000 \\ n &= 250 \text{ unidades} \end{aligned}$$

Una corrida de producción de 250 o más unidades justificaría el uso del 8051 sobre el 8751.

La situación se complica aún más cuando comparamos diseños, por ejemplo, utilizando un 8051 en lugar de un 8031 + 2764. En el segundo caso, la alternativa del 8031 + 2764 es mucho más barata que un 8751 con EPROM incorporada en el chip, así que el punto de igualdad ocurre con cantidades mayores. Si un 8031 + 2764 se vende en, digamos, \$7, entonces el punto de igualdad es

$$\begin{aligned} 7n &= 5n + 5000 \\ 2n &= 5000 \\ n &= 2500 \text{ unidades} \end{aligned}$$

Una corrida de producción de 100 unidades no justificaría el uso del 8051, o al menos así parece. El uso de una memoria EPROM externa significa que los puertos 0 y 2 no están disponibles para E/S. Esto puede ser un punto crítico que evite la solución con el 8031 + 2764. Aunque la pérdida de E/S incorporada en el chip no sea importante, aparecen otros factores. La solución con el 8031 + 2764 requiere de dos circuitos integrados en lugar de uno. Esto complica la manufactura, las pruebas, el mantenimiento, la confiabilidad, la compra, y una variedad de otras dimensiones aparentemente inocuas, pero aún así reales, del diseño de productos. Además, el diseño con el 8031 + 2764 será físicamente más grande que el obtenido con el 8051. Si el producto final necesita un factor de forma pequeño, entonces quizás tengamos que utilizar el 8051, independientemente del costo adicional.

10.4 COMANDOS Y AMBIENTES

En esta sección consideraremos el ambiente de desarrollo en su totalidad. Presentamos la noción de que en cualquier momento el diseñador trabaja dentro de un “ambiente” donde los comandos realizan la mayor parte de las tareas. El ambiente central es el sistema operativo en el sistema anfitrión, el cual muy probablemente sea MS-DOS ejecutándose en un miembro de la familia PC de microcomputadoras. Como sugiere la figura 10-4, algunos comandos regresan a MS-DOS al final de su ejecución, mientras que otros evocan un ambiente nuevo.

Invocación de comandos. Los comandos pueden ser **residentes** (por ejemplo, DIR) o **transitorios** (FORMAT, DISKCOPY). Un comando residente está en memoria todo el tiempo, listo para su ejecución (por ejemplo, DIR). Un comando transitorio es un archivo en disco ejecutable que se carga en memoria para su ejecución (por ejemplo, FORMAT).

Los programas de aplicación son similares a los comandos transitorios en el sentido de que existen como un archivo en disco ejecutable y se invocan desde el indicador de comandos de MS-DOS. Sin embargo, hay aún más posibilidades. Los comandos o aplicaciones pueden invocarse como parte de un archivo de proceso en lotes, por medio de una tecla de función, o desde una interfaz al usuario controlada por menús que actúan como presentación para el MS-DOS.

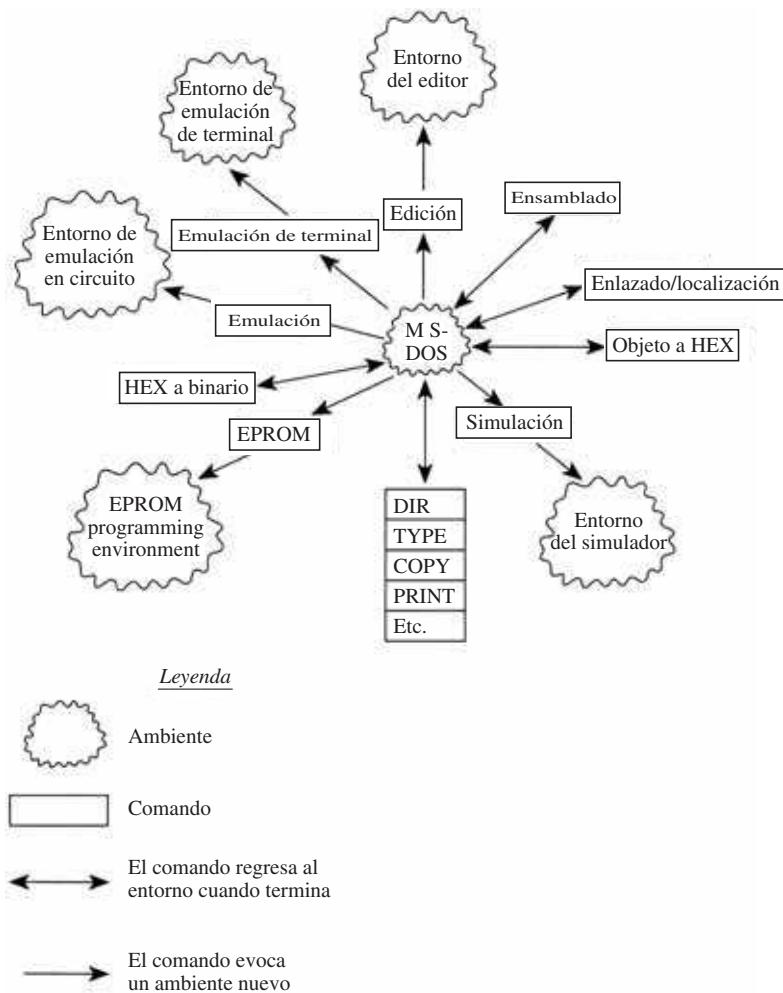


FIGURA 10-4
El ambiente de desarrollo

Existen aún más posibilidades cuando se requieren argumentos para los comandos. Aunque los argumentos se envían por lo general en la línea de invocación después del comando, algunos comandos tienen valores predeterminados para los argumentos o los piden al usuario. Desafortunadamente, no existe un mecanismo estándar, tal como el “cuadro de diálogo” utilizado en la interfaz de *Macintosh*, para obtener la información adicional requerida para ejecutar un comando o una aplicación.

Algunas aplicaciones, como los editores, “toman el control” del sistema y llevan al usuario a un nuevo ambiente para efectuar actividades subsecuentes.

Entornos. Como resulta evidente en la figura 10-4, algunas herramientas de software tales como el simulador, el emulador de circuito o el programador de EPROM evocan su propio entorno. Aprender las características de cada uno lleva tiempo, debido a la gran variedad de técnicas

existentes para dirigir las actividades del ambiente: teclas de cursor, teclas de funciones, mandos de primera letra, menús resaltados, directorios predeterminados, y así sucesivamente. Con frecuencia es posible intercambiar los entornos mientras se les deja activos. Por ejemplo, los emuladores de terminales y los editores a menudo permiten cambiar un momento al DOS para ejecutar comandos. El comando EXIT en DOS lleva de inmediato al usuario de vuelta al entorno suspendido.

Metodología. La investigación en inteligencia artificial y ciencia cognitiva ha descubierto que modelar la “solución a los problemas” efectuada por los humanos es difícil de capturar. Los humanos parecen enfocarse hacia los elementos de una situación en paralelo, es decir, midiendo las acciones posibles y procediendo por intuición al mismo tiempo. La metodología aquí sugerida reconoce esta cualidad humana. Los pasos incluidos en el ciclo de desarrollo y las herramientas y técnicas ofrecidas por el entorno de desarrollo deben entenderse con claridad, pero el proceso global debe dar soporte a la suficiente libertad de acción.

La operación básica de los comandos es “traducir”, “observar” o “evocar” (un entorno nuevo). Los resultados de la traducción deben poder ser observados para verificar los resultados. Podemos tomar la actitud de no creer en la consecuencia de cualquier traducción (ensamblado, programación de EPROM, etc.) y verificar todo mediante la observación de resultados. Las herramientas de observación son comandos tales como DIR (¿Se han creado los archivos de salida que esperábamos?), TYPE (¿Qué es lo que contiene el archivo de salida?), EDIT, PRINT, y así sucesivamente.

RESUMEN

En este capítulo presentamos las herramientas y técnicas disponibles para efectuar el diseño de productos basados en un microcontrolador. Sin embargo, no hay sustituto para la experiencia. El éxito en el diseño requiere de una considerable intuición, recurso invaluable que no puede entregarse en un libro. La principal técnica empleada por los diseñadores para convertir ideas en productos reales sigue siendo la búsqueda de soluciones por tanteo.

PROBLEMAS

- 10.1** Si los EPROM 8751 se venden a \$30 sea cual fuere la cantidad comprada, y un 8051 programado con máscara se vende a \$3 más un precio inicial de \$10,000, ¿cuántas unidades se necesitan para justificar el uso del dispositivo 8051? ¿Cuál es el ahorro para las ventas pronosticadas de 3000 unidades del producto final si se utiliza el 8051 en lugar del 8751?
- 10.2** A continuación se presenta un programa para el 8051 en el formato hexadecimal de Intel.

```
:100800007589117F007E0575A88AD28FD28D80FEF2
:10081000C28C758C3C758AB0DE087E050FBF09025C
:100820007F00D28C32048322FB90FC0CFC7AFCAD5E
:0A083000FD0AFD5CFDA6FDC8FDC831
:00000001FF
```

- ¿Cuál es la dirección inicial del programa?
- ¿Cuál es el tamaño del programa?
- ¿Cuál es la última dirección del programa?

- 10.3** La siguiente es una sola línea de un archivo hexadecimal de Intel con un error en la suma de validación. La suma de validación incorrecta aparece en los últimos dos caracteres como “00”. ¿Cuál es la suma de validación correcta?

:100800007589117C007F0575A8FFD28FD28D80FE00

- 10.4** El contenido de un archivo hexadecimal de Intel se muestra a continuación.

:090100007820765508B880FA2237

:00000001FF

Vuelva a crear el programa fuente original que representa este archivo.

Ejemplos de diseño e interfaces

11.1 INTRODUCCIÓN

En este capítulo combinaremos muchas de las características del hardware y del software que conforman el 8051 a través de diversos ejemplos de diseño e interfaces. El primer ejemplo es una computadora de una sola tarjeta basada en el 8051 (la SBC-51), la cual resulta muy útil para aprender acerca del 8051 o para desarrollar productos basados en este dispositivo. La SBC-51 utiliza un sólido programa de monitoreo que ofrece comandos básicos para la operación del sistema y la interacción con el usuario. El apéndice G describe con todo detalle el programa de monitoreo (MON51).

Los ejemplos de interfaces son avanzados si los comparamos con aquellos presentados en capítulos anteriores. Cada ejemplo incluye un diagrama esquemático del hardware, una descripción del objetivo de diseño, un listado del programa que lleva a cabo el objetivo de diseño, y una descripción general de la operación del hardware y del software. Los listados de software contienen muchos comentarios; le sugerimos consultarlos para obtener detalles específicos.

11.2 LA SBC-51

Diversas compañías ofrecen computadoras de una sola tarjeta basada en el 8051 similar a la descrita en esta sección. Sin embargo, el diseño básico de una computadora de una sola tarjeta basada en el 8051 no varía sustancialmente entre los diversos productos ofrecidos. Debido a que muchas características están “incorporadas al chip”, el diseño de una computadora de una sola tarjeta basada en el 8051 es simple y directo. En general, sólo se requiere de las conexiones básicas a la memoria externa y de la interfaz a la computadora anfitriona.

También se requiere de un programa de monitoreo en la memoria EPROM. Para comenzar, se necesitan únicamente los requerimientos más básicos del sistema, tales como el análisis y el cambio de las ubicaciones en memoria o la descarga de programas de aplicación desde una computadora anfitriona. La SBC-51 que describimos a continuación trabaja junto con un programa de monitoreo simple que proporciona estas funciones básicas.

La figura 11-1 contiene el diagrama esquemático elaborado para la SBC-51. El diseño completo incluye sólo 10 circuitos integrados, pero aún así es lo suficientemente poderoso y flexible como para soportar el desarrollo de productos sofisticados basados en el 8051. La parte central de la operación de una SBC-51 es el programa de monitoreo residente en memoria EPROM y que se comunica con una terminal de video (VDT) conectada al 8051. El apéndice G describe con detalle el programa de monitoreo.

La SBC-51 incluye, además de las características estándar del 80C31, una memoria EPROM externa de 16K bytes, 8.25K bytes de memoria RAM externa, un temporizador de 14 bits adicional, y 22 líneas de entrada/salida adicionales. La configuración mostrada en la figura 11-1 incluye los siguientes componentes y partes:

- 10 circuitos integrados
- 15 capacitores
- 2 resistencias
- 1 cristal
- 1 interruptor de presión de botón
- 3 conectores
- 13 puentes para configuración

Debido a que se utiliza la memoria externa, los puertos 0 y 2 no están disponibles para uso de entrada/salida. Aunque los puertos 1 y 3 son utilizados parcialmente para características especiales, algunas de sus líneas pueden utilizarse con propósitos de entrada/salida, dependiendo de la configuración.

La fuente de sincronización del 80C31 es un cristal de 12 MHz conectado de la manera usual. (Consulte la figura 2-2). La línea de RST (reinicialización) está controlada por una red R-C para reinicialización al encender y por un interruptor de presión de botón para reiniciar manualmente. El puerto 0 funciona como bus de datos (D0 a D7) y como byte inferior del bus de direcciones (A0 a A7), de acuerdo con lo que estudiamos antes. (Consulte la sección 2.7 Memoria externa). El latch octal 74HC373 está sincronizado mediante la señal ALE para mantener el byte inferior del bus de direcciones durante un ciclo de memoria. Debido a que el 80C31 no incluye memoria ROM incorporada en el chip, la ejecución ocurre desde la memoria EPROM externa y, por lo tanto, la señal EA (acceso externo) está conectada a tierra mediante el puente para configuración X2.

La conexión a la computadora anfitriona o a la VDT utiliza la interfaz serial RS232C. El conector DB25S está cableado como un DTE (equipo terminal de datos) con la señal de transmisión de datos (TXD) en la terminal 2, recepción de datos (RXD) en la terminal 3, y tierra en la terminal 7. Un controlador de línea 1488 RS232 está conectado a la señal de TXD y un receptor de línea 1489 RS232 a la señal de RXD. La conexión predeterminada para el 80C31 es a través de los puentes X9 y X10, con P3.1 como TXD y P3.0 como RXD. De modo alterno, usando los puentes X11 y X12, las funciones de TXD y RXD pueden proporcionarse mediante software utilizando P1.7 y P1.6.

El programa de monitoreo lee las líneas 3, 4 y 5 del puerto 1 en el tiempo de reinicialización para evocar características especiales. Sin embargo, después de una reinicialización, estas líneas quedan disponibles para E/S de propósito general. Si se utiliza la interfaz a la impresora, las líneas 0, 1 y 2 del puerto 1 son las señales de intercambio; en caso contrario, estas líneas están disponibles para E/S de propósito general.

El 74H4C138 decodifica los tres bits superiores en el bus de direcciones (A15 a A13) y genera ocho líneas de selección, una para cada bloque de 8K de memoria. Estas líneas se llaman $\overline{S8K0}$ (“selecciona bloque de 8K 0”) hasta $\overline{S8K7}$. Son cuatro los circuitos integrados que seleccionan dichas líneas: dos memorias EPROM 2764, una memoria RAM 6264, y un 8155 que es un TEMPORIZADOR/RAM/ES.

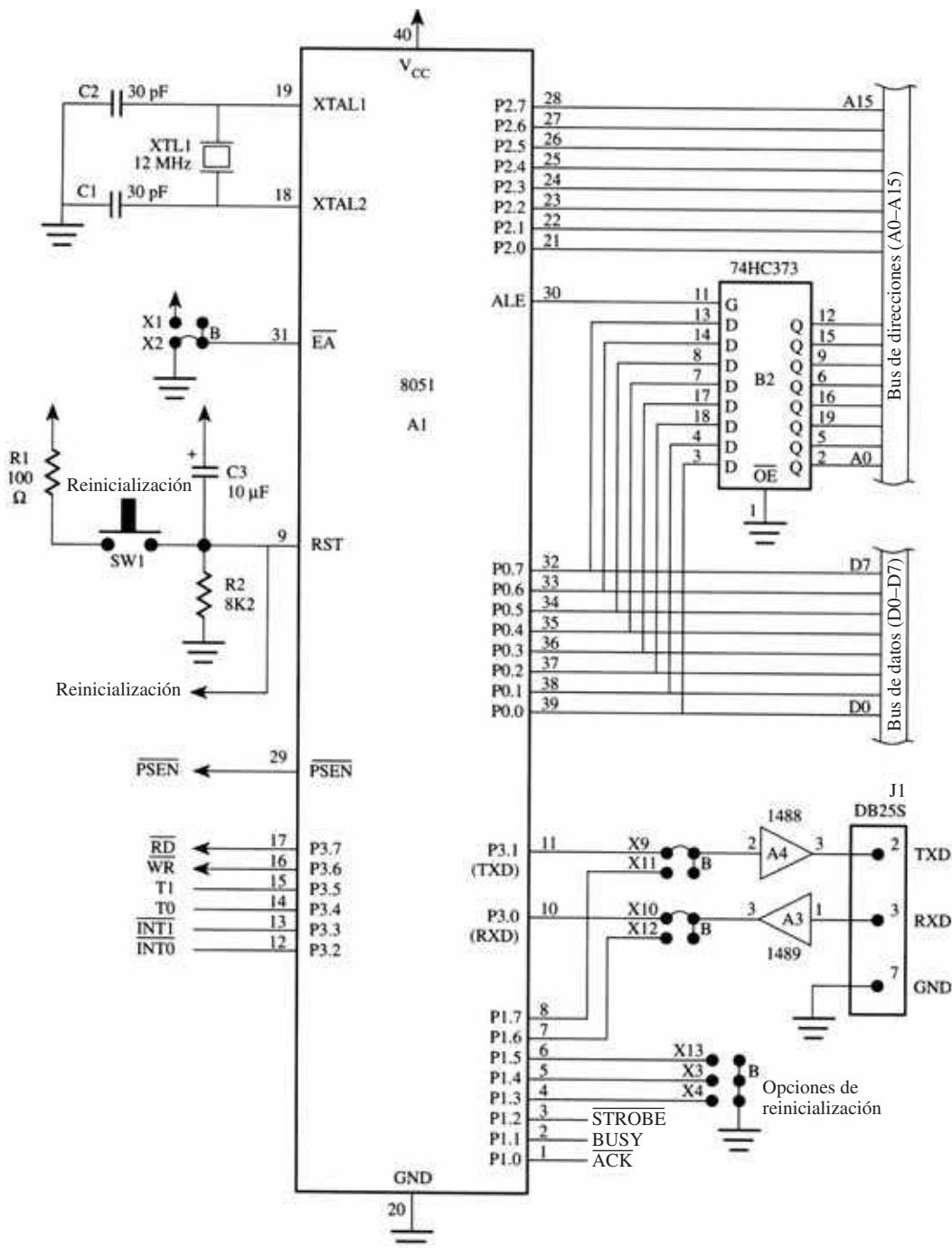
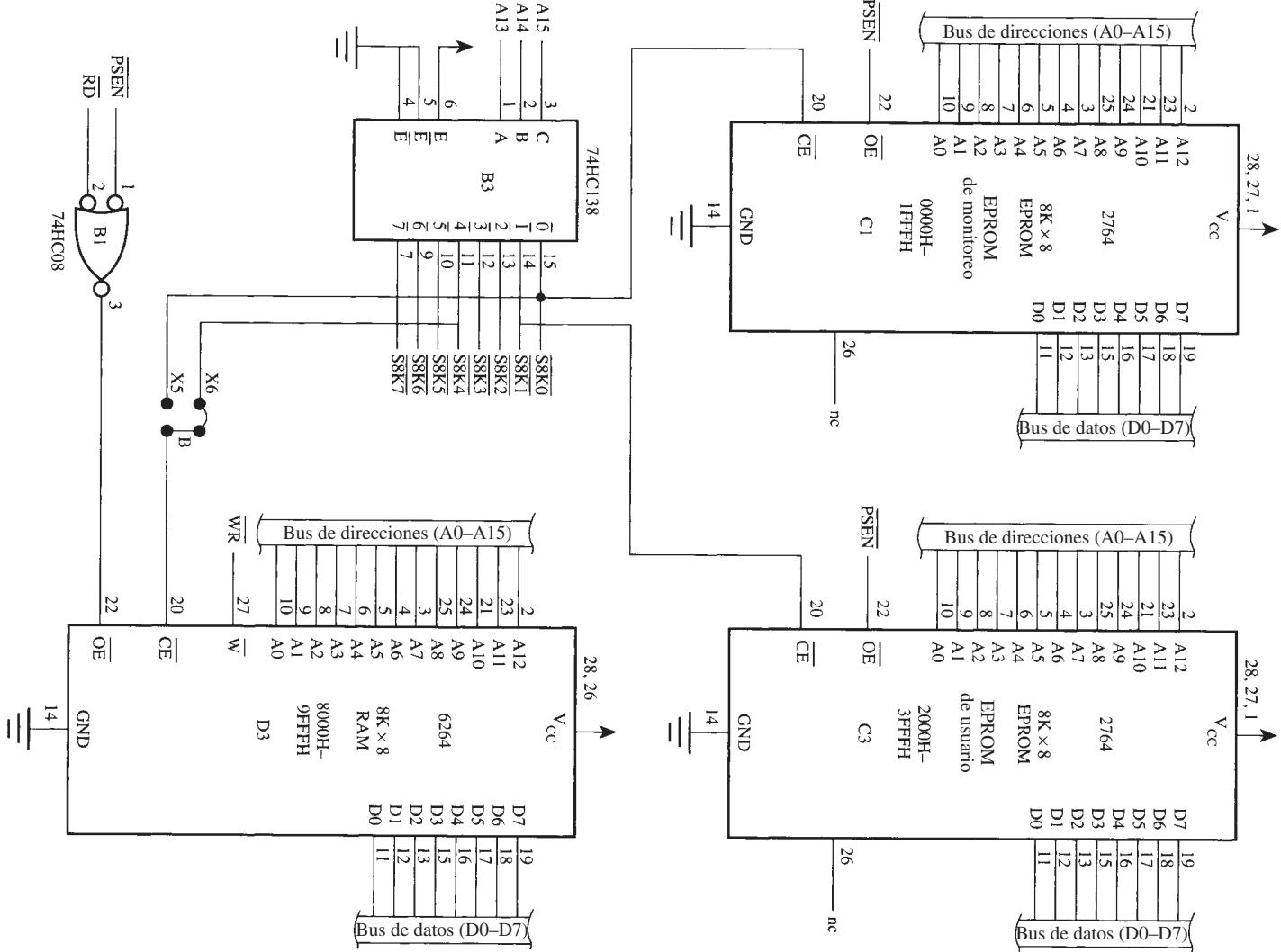


FIGURA 11-1a

Computadora de una sola tarjeta basada en el 8051 –la SBC-51. (a) Procesador e interfaz al puerto serial; (b) Decodificación de direcciones, RAM y EPROM; (c) El 8155 y las conexiones de alimentación

FIGURA 11-1b
continuación



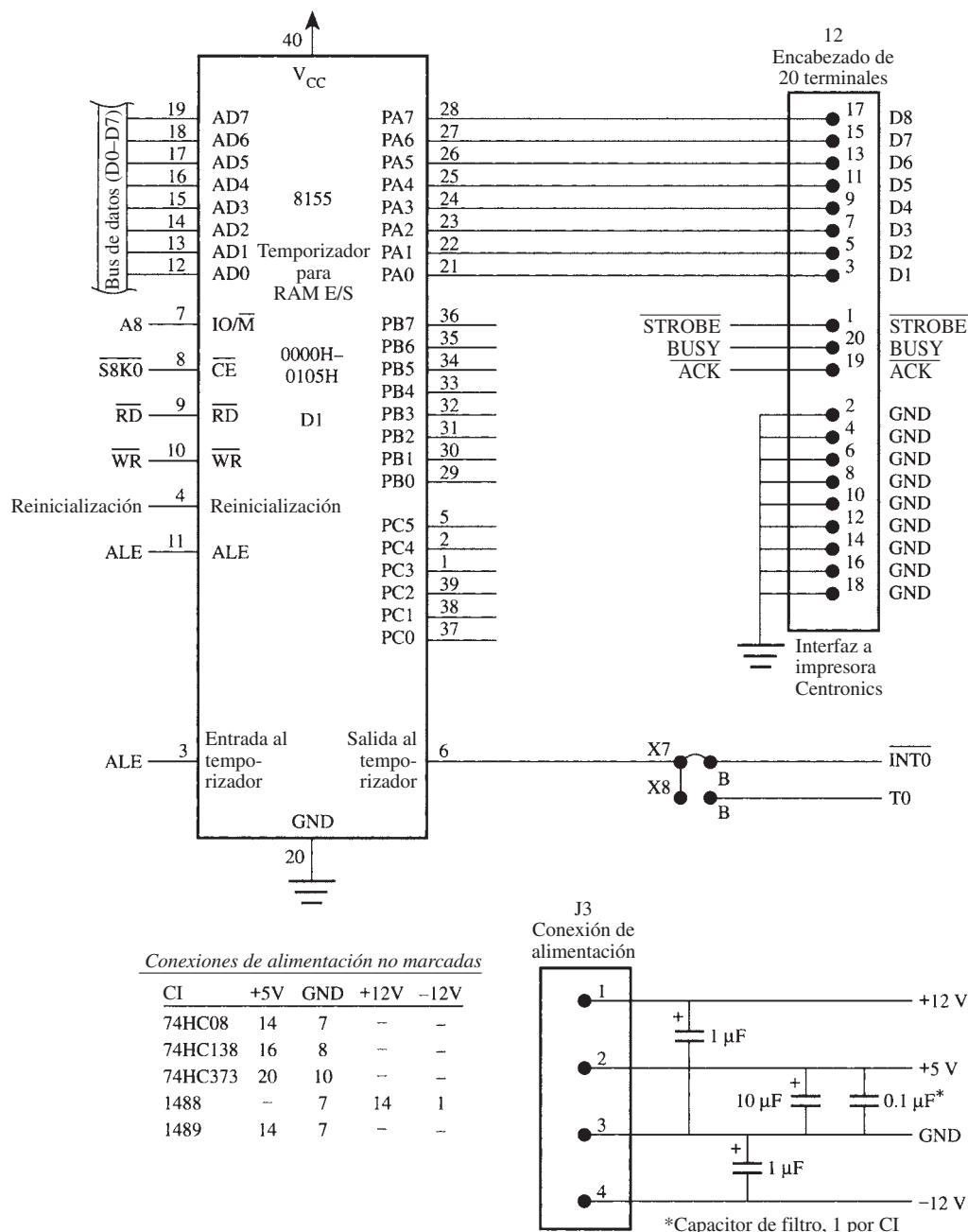


FIGURA 11-1c
continuación

La figura 11-1 muestra dos memorias EPROM 2764 de 8K por 8. La primera memoria (llamada “EPROM DE MONITOREO”) se selecciona mediante $\overline{S8K0}$ y reside en el espacio externo para código de las direcciones 0000H a 1FFFH. Debido a que la SBC-51 comenzará a ejecutarse desde la dirección 0000H inmediatamente después de una reinicialización del sistema, el programa de monitoreo debe residir en este circuito integrado. La segunda memoria 2764 se llama “EPROM DEL USUARIO” y se selecciona mediante $\overline{S8K1}$ para ser ejecutada en las direcciones 2000H a 3FFFH. Este circuito integrado tiene como destino las aplicaciones del usuario y no se necesita para la operación básica del sistema. Observemos que ambas memorias EPROM se seleccionan sólo si la señal \overline{CE} (habilitación de chip; terminal 20) está activa (o a nivel bajo) y \overline{OE} también está activa (o a nivel bajo). \overline{OE} está controlada por la línea \overline{PSEN} del 80C31; por ello la selección está en el espacio externo para código, como lo esperamos.

El circuito integrado 6264 de memoria RAM de 8K por 8 se selecciona mediante la señal $\overline{S8K4}$ (si el puente X6 está instalado, como se muestra), así que reside en las direcciones 8000H a 9FFFH. La memoria RAM es seleccionada para ocupar tanto el espacio externo para datos como el espacio externo para código utilizando el método descrito anteriormente. (Consulte la sección 2.7.4 Traslape para los espacios externos para código y datos). Esta ocupación dual permite que los programas de usuario se carguen (o escriban) a la memoria RAM como “memoria para datos” y entonces se ejecuten como “memoria para código”

El temporizador para RAM/ES 8155 es un circuito integrado de interfaz periférica que se agregó para demostrar la capacidad de expansión de la SBC-51. Es fácil añadir otros circuitos integrados de interfaz periférica de manera similar. El 8155 se selecciona a través de $\overline{S8K0}$, ello lo ubica en la parte inferior de la memoria. No hay conflicto alguno con el monitoreo de memoria EPROM (que también reside en la parte inferior de la memoria, pero en el espacio externo para código) debido a que el 8155 se selecciona además para realizar operaciones de lectura o escritura mediante las señales \overline{RD} y \overline{WR} .

El 8155 posee las siguientes características:

- 256 bytes de memoria RAM
- 22 Líneas de entrada/salida
- Temporizador de 15 bits

La línea de dirección A8 se conecta a la línea IO/\overline{M} (terminal 7) del 8155 y selecciona la memoria RAM cuando está a nivel bajo y las líneas de E/S o el temporizador cuando está a nivel alto. El acceso a las líneas de E/S y al temporizador se realiza mediante seis direcciones, así que el rango total de direcciones del 8155 es 0000H a 0105H (256 + 6 direcciones); las cuales se resumen a continuación.

Dirección	Propósito
0000H	primera dirección de RAM
...	Otras direcciones de RAM
00FFH	última dirección de RAM
0100H	Registro de intervalo/comando
0101H	Puerto A
0102H	Puerto B
0103H	Puerto C
0104H	8 bits inferiores del conteo del temporizador
0105H	6 bits superiores del conteo del temporizador y 2 bits del modo del temporizador

Aunque debemos consultar la hoja de datos del fabricante para conocer los detalles de operación del 8155, la configuración de los puertos de E/S es bastante sencilla. De manera predeterminada,

todas las líneas de puerto son entradas después de una reinicialización del sistema; por lo tanto, no se requiere de ninguna operación de “inicialización” para leer dispositivos de entrada conectados al 8155. Para leer el puerto A al acumulador, por ejemplo, utilizamos la siguiente secuencia de instrucciones:

```
MOV DPTR,#0101H      ;DPTR apunta al puerto A del 8155
MOVX A,@DPTR         ;lee puerto A al acumulador
```

Para programar los puertos A y B como salidas, debemos escribir números uno a los bits 0 y 1 del registro de comandos, respectivamente. Por ejemplo, para configurar al puerto B como puerto de salida y dejar los puertos A y C como entrada, utilizamos la siguiente secuencia de instrucciones:

```
MOV DPTR,#0100H      ;registro de comandos del 8155
MOV A,#00000010B    ;Puerto B = salida
MOVX @DPTR,A         ;inicializa al 8155
```

El puerto C se configura como una salida escribiendo números uno a los bits 2 y 3 del registro de comandos. Todos los tres puertos se configurarán como salida con lo siguiente:

```
MOV DPTR,#0100H      ;registro de comandos del 8155
MOV A,#00001111B    ;todos los puertos = salida
MOVX @DPTR,A         ;inicializa al 8155
```

El puerto A del 8155 se muestra conectado a un encabezado de 20 terminales llamado “Interfaz a impresora Centronics”. Esta interfaz es sólo para fines de demostración. El programa MON51 incluye una subrutina PCHAR (imprime carácter) y dirige la salida a la VDT y a una impresora en paralelo si se envía una señal CONTROL-Z desde el teclado. (Consulte el apéndice G). Por supuesto que el puerto A puede utilizarse para otros propósitos, si así lo desea el usuario.

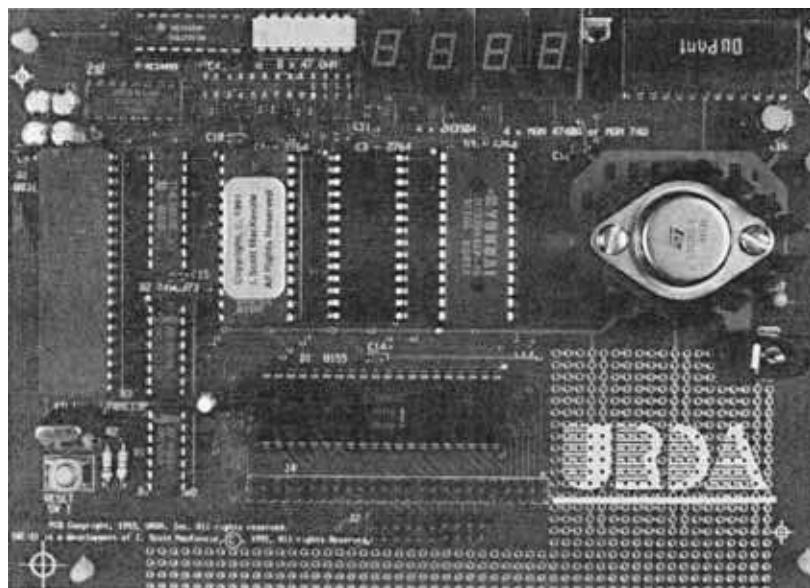
La figura 11-1 también muestra las conexiones a la fuente de poder. Los capacitores de filtro son particularmente importantes para la fuente de +5 voltios para evitar problemas ocasionados por los efectos de inducción cuando los dispositivos digitales se intercambian. Si la SBC-51 está construida en una tarjeta para prototipos (por ejemplo, mediante la “envoltura de alambre”), los capacitores deben considerarse imprescindibles. Coloque un capacitor electrolítico de $10 \mu\text{F}$ donde la alimentación de poder entra a la tarjeta para prototipos, y capacitores cerámicos de $0.01 \mu\text{F}$ al lado de cada zócalo para cada circuito integrado, conectados entre la terminal de +5 voltios y la terminal de tierra.

Debido a que la SBC-51 es pequeña y económica, resulta sencillo construir un prototipo y obtener experiencia práctica a través del programa de monitoreo y de los ejemplos de interfaces presentados en este capítulo. La técnica de “envoltura de alambres” es el método más práctico de construcción. La SBC-51 también está disponible ya ensamblada y probada en una tarjeta de circuito impreso (consulte la figura 11-2).

Esto concluye nuestra descripción de la SBC-51. Las siguientes secciones contienen ejemplos de las interfaces a dispositivos periféricos que se han desarrollado para conectarse a la SBC-51 (o a una computadora similar de una sola tarjeta basada en el 8051).

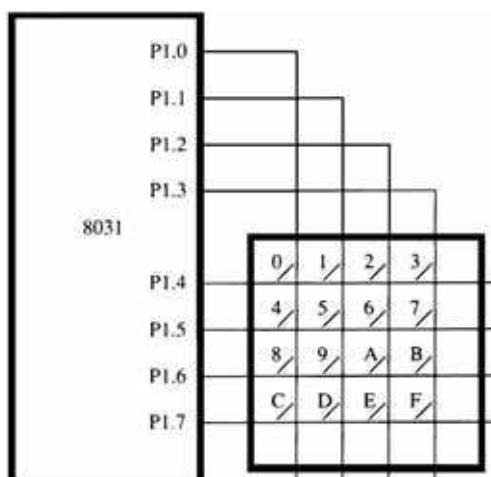
11.3 INTERFAZ A UN TECLADO NUMÉRICO HEXADECIMAL

Las interfaces a teclados numéricos son comunes para los diseños basados en microcontroladores. La entrada a través del teclado numérico y una salida a LED son una alternativa económica

**FIGURA 11–2**

La versión de tarjeta de circuito impreso para la SBC-51 (cortesía de URDA, Inc.)

para una interfaz al usuario y a menudo resultan adecuados para aplicaciones complejas. Ejemplos de esto incluyen la interfaz al usuario de los hornos de microondas o las máquinas bancarias automatizadas. La figura 11-3 muestra una interfaz entre el puerto 1 y un teclado numérico hexadecimal. El teclado numérico contiene 16 teclas distribuidas en cuatro renglones y cuatro columnas. Las líneas de los renglones están conectadas a los bits 4-7 del puerto 1 y las líneas de las columnas a los bits 0 a 3 del puerto 1.

**FIGURA 11–3**

Interfaz a un teclado numérico hexadecimal

EJEMPLO **Objetivo del diseño****11.1**

Escriba un programa que lea en forma continua caracteres hexadecimales desde el teclado numérico y que presente el correspondiente código en ASCII en la consola.

A simple vista, este ejemplo parece demasiado simple. El software se puede dividir en los siguientes pasos:

1. Obtener un carácter hexadecimal del teclado numérico.
2. Convertir el código en hexadecimal a ASCII.
3. Enviar el código en ASCII a la VDT.
4. Ir al paso 1.

De hecho, la solución en software mostrada en la figura 11-4 sigue este patrón exacto (consulte las líneas 16 a 19). Por supuesto que el trabajo se realiza en las subrutinas. Observe que los pasos 2 y 3 se implementan mediante la llamada a subrutinas en MON51. Desde luego, el código pudo haberse extraído del programa MON51 y ser colocado en el listado de la figura 11-4, pero esto sería un gasto innútil. En lugar de eso, definimos los puntos de entrada de MON51 a estas subrutinas al inicio del listado (en las líneas 12 y 13) utilizando los símbolos H_A_A y CARSAL, y entonces llamamos a las subrutinas en el ciclo de programa PRINCIPAL de la manera usual. Los puntos de entrada para las subrutinas de MON51 pueden encontrarse en la tabla de símbolos creada por el RL51 cuando se enlazó y localizó el MON51. En el apéndice G podemos encontrar, por ejemplo, los puntos de entrada para H_A_A y CARSAL.

El verdadero reto de este ejemplo es el de escribir las subrutinas HEX_ENT y OBSENTECLA. OBSENTECLA realiza el trabajo de examinar las líneas de renglón y de columna del teclado numérico para determinar si el usuario oprimió una tecla. Si no se oprimió una tecla, la subrutina regresa con C = 0; en caso contrario, la subrutina regresa con C = 1 y con el código en hexadecimal para la tecla en los bits 0 a 3 del acumulador.

HEX_ENT realiza una estabilización mediante el software. Debido a que el teclado numérico es una serie de interruptores mecánicos, cuando los contactos se cierran y abren generan un rebote —el rápido pero breve hacer y deshacer de los contactos de los interruptores—. La estabilización se lleva a cabo llamando la subrutina OBSENTHex repetidamente hasta que 50 llamadas consecutivas regresan con C = 1. Cualquier llamada a OBSENTHex que regrese con C = 0 se interpreta como ruido (en otras palabras, rebote) y el contador se reinicializa. Cuando se detecta un cierre legítimo de una tecla, HEX_ENT espera por 50 llamadas consecutivas a OBSENTHex que regresen con C = 0. Esto asegura que la tecla se ha dejado de oprimir antes de la siguiente llamada a OBSENTHex.

El software mostrado en la figura 11-4 funciona, pero en sí no es elegante. Debido a que no usamos interrupciones, la utilidad del programa dentro de una aplicación más grande es limitada. Una mejora más razonable, por lo tanto, es el rediseño del software utilizando interrupciones. El siguiente ejemplo ilustra una interfaz controlada por interrupciones.

11.4 INTERFAZ CON VARIOS LEDS DE 7 SEGMENTOS

Al final del capítulo 3 se presentó una interfaz a un visualizador LED de 7 segmentos como un problema. (Consulte la figura 3-8). Desafortunadamente, la interfaz utilizaba siete líneas en el puerto 1, así que representa una distribución pobre de los recursos incorporados en el chip del 8051. En esta sección, demostramos una interfaz a cuatro LEDs de 7 segmentos utilizando sólo tres de las líneas de E/S del 8051. Esto, desde luego, es un diseño bastante mejorado, particularmente si requerimos conectar múltiples segmentos.

```

LOC OBJ LINE SOURCE
1 $DEBUG
2 $NOPAGING
3 $NOSYMBOLS
4 ;ARCHIVO: TECLADONUMERICO.SRC
5 ;*****EJEMPLO DE INTERFAZ A TECLADO NUMERICO*****
6 ;
7 ;
8 ; Este programa lee caracteres en hexadecimal de un teclado numérico
9 ; conectado al puerto 1 y presenta las teclas oprimidas en la consola
10 ;
11 ;*****Subrutinas del MON51 (V12)*****
033C 12 H_A A EQU 033CH ;Subrutinas del MON51 (V12)
01DE 13 CARSAL EQU 01DEH
14
8000 15 ORG 8000H
8000 16 PRINCIPAL: CALL HEX_ENT ;obtener código del teclado numérico
8003 17 CALL H_A A ;convertir a ASCII
8006 18 CALL CARSAL ;presentar en consola
8009 19 SJMP PRINCIPAL ;repetir
20
21 ;*****HEX_ENT - recibe código en hexadecimal como entrada del teclado numérico*****
22 ; con estabilización para la presión y liberación de una tecla
23 ; (50 operaciones repetidas para cada una)
24 ;*****Subrutinas del MON51 (V12)*****
25
800B 7B32 26 HEX_ENT: MOV R3,#50 ;conteo de estabilización
800D 128022 27 ATRAS: CALL OBTEN_TECLA ;¿tecla presionada?
8010 50F9 28 JNC HEX_ENT ;no: verifica de nuevo
8012 DBF9 29 DJNZ R3,ATRAS ;sí: repetir 50 veces
8014 C0E0 30 PUSH ACC ;almacena código en hex
8016 7B32 31 ATRAS2: MOV R3,#50 ;espera por liberación de tecla
8018 128022 32 ATRAS3: CALL OBTEN_TECLA ;¿tecla presionada?
801B 40F9 33 JC ATRAS2 ;sí: continúa verificando
801D DBF9 34 DJNZ R3,ATRAS3 ;no: repetir 50 veces
801F D0E0 35 POP ACC ;recupera código en hex y
8021 22 36 RET ; regresa
37
38 ;*****OBTEN_TECLA - obtener estatus del teclado numérico*****
39 ; - regresar con C = 0 si no se presionó una tecla
40 ; - regresar con C = 1 y código en hexadecimal en
41 ; ACC si se presionó una tecla
42 ;*****Subrutinas del MON51 (V12)*****
43
8022 74FE 44 OBTEN_TECLA: MOV A,#0FEH ;inicia con columna 0
8024 7E04 45 MOV R6,#4 ;utiliza R6 como contador
8026 F590 46 PRUEBA: MOV P1,A ;activar línea de columna
8028 FF 47 MOV R7,A ;almacenar ACC
8029 E590 48 MOV A,P1 ;lectura del puerto 0
802B 54F0 49 ANL A,#0FOH ;aislar líneas de renglón
802D B4F007 50 CJNE A,#0FOH,TECLA_PRESION ;¿línea de renglón activada?
8030 EF 51 MOV A,R7 ;no: continúa a la siguiente
8031 23 52 RL A ; línea de columna
8032 DEF2 53 DJNZ R6,PRUEBA
8034 C3 54 CLR C ;ninguna tecla presionada
8035 8015 55 SJMP TERMINAR ;regresar con C = 0
8037 FF 56 TECLA_PRESION: MOV R7,A ;almacenar en R6
8038 7404 57 MOV A,#4 ;preparar para cálculos
803A C3 58 CLR C ;peso de columna
803B 9E 59 SUBB A,R6 ;4 - R6 = peso
803C FE 60 MOV R6,A ;almacenar en R6
803D EF 61 MOV A,R7 ;recuperar código de escaneo
803E C4 62 SWAP A ;colocar en nibble inferior
803F 7D04 63 MOV P5,#4 ;utilizar P5 como contador

```

FIGURA 11-4a

Software para la interfaz al teclado numérico

```

8041 13    64  DENUEVO:      RRC      A          ;rotación hasta llegar a 0
8042 5006   65  JNC      LISTO      ;listo cuando C = 0
8044 0E     66  INC       R6         ;agregar 4 hasta que esté activo
8045 0E     67  INC       R6         ; renglón encontrado
8046 0E     68  INC       R6
8047 0E     69  INC       R6
8048 DDF7   70  DJNZ     R5, DENUEVO
804A D3    71  LISTO:      SETB     C          ;C = 1 (tecla presionada)
804B EE    72  MOV       A,R6      ;código en A (jal fin!)
804C 22   73  TERMINAR:   RET
804D 22   74  END

```

FIGURA 11-4b*continuación*

La parte central de este diseño es el decodificador/controlador Motorola MC14499 de 7 segmentos, el cual incluye mucho del circuito necesario para controlar cuatro visualizadores. Los únicos componentes adicionales son un capacitor de sincronización de $0.015 \mu\text{F}$, siete resistencias limitadoras de corriente de 47Ω , y cuatro transistores 2N3904. La figura 11-5 muestra las conexiones entre el 80C51, el MC14499, y los cuatro LED de 7 segmentos.

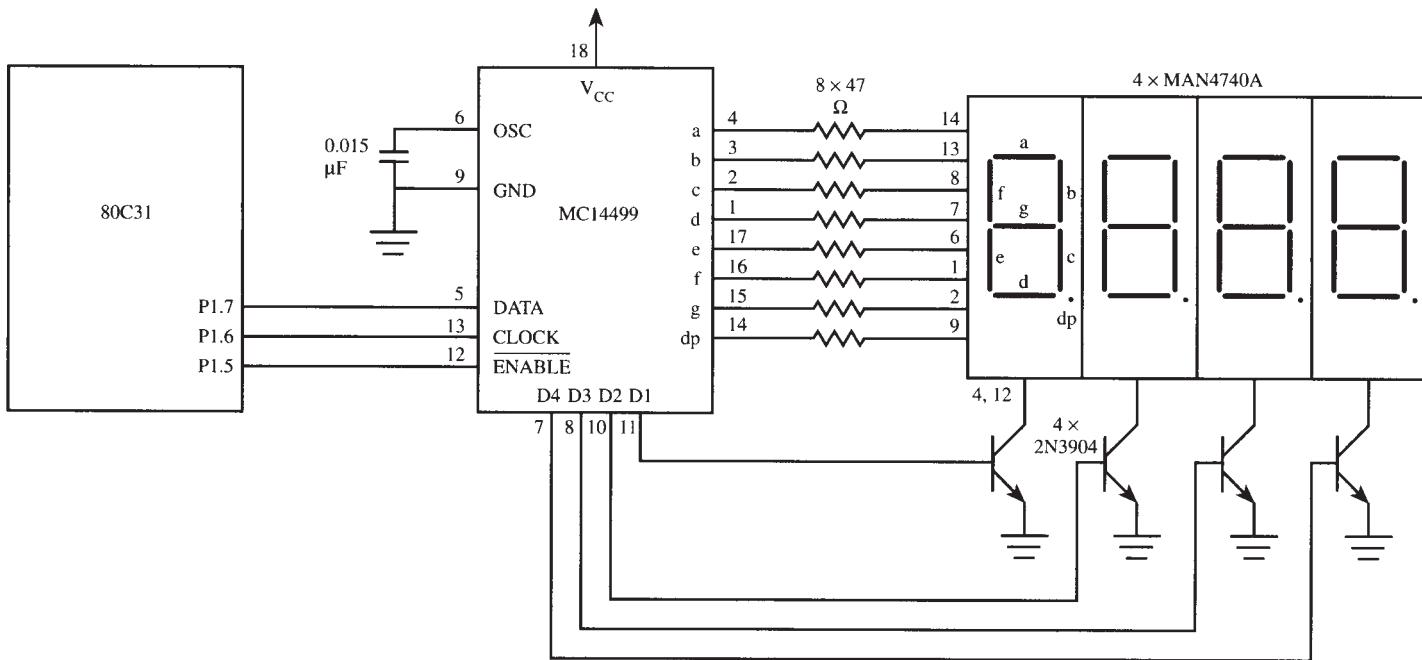
EJEMPLO**11.2****Objetivo del diseño**

Suponga que hay dígitos en BCD almacenados en las ubicaciones de memoria RAM interna 70H y 71H. Copie los dígitos en BCD al visualizador LED 10 veces por segundo, utilice interrupciones.

La figura 11-6 muestra el software necesario para alcanzar el objetivo anterior. El listado ilustra cierta cantidad de conceptos que ya hemos visto. Los detalles a bajo nivel sobre cómo enviar datos al MC14499 se encuentran en las subrutinas ACTUALIZACION y SALIDA8. A un nivel superior, este ejemplo ilustra el diseño de una aplicación controlada por interrupciones con una cantidad significativa de actividad en los planos primero y segundo (a diferencia de los ejemplos del capítulo 6 que operaban sólo en el segundo plano). Las interrupciones elaboradas para este capítulo coexisten con el programa MON51, el cual no utiliza interrupciones por sí mismo. El programa de monitoreo se ejecuta en el primer plano mientras que el de la figura 11-6 se ejecuta al nivel de interrupciones en el segundo plano. Cuando el programa inicia (por ejemplo, al ejecutar el comando GO8000 del MON51; consulte el apéndice G), las condiciones se inicializan para procesar la necesaria actualización iniciada por interrupciones de los visualizadores LED, entonces el control regresa rápidamente al programa de monitoreo. Los comandos de monitoreo pueden ejecutarse de la manera usual; mientras tanto, las interrupciones ocurren en el segundo plano. Si, por ejemplo, el comando de monitoreo SET se utiliza para cambiar las ubicaciones en memoria RAM interna 70H y 71H, veremos los cambios inmediatamente (dentro de 0.1 s) en los visualizadores LED de 7 segmentos.

Observemos la estructura general del programa. Las siguientes secciones aparecen en orden:

- Controles del ensamblador (líneas 1 a 3)
- Bloque de comentarios (líneas 4 a 30)
- Definición de símbolos (líneas 31 a 38)
- Declaración de definiciones de almacenamiento (líneas 40 a 42)
- Tabla de saltos para los puntos de entrada del programa y de las interrupciones (líneas 44 a 51)
- Sección principal (PRINCIPAL; líneas 56 a 69)

**FIGURA 11–5**

Interfaz al MC14499 y cuatro LEDs de 7 segmentos

LOC	OBJ	LINE	SOURCE
		1	\$DEBUG
		2	\$NOPAGING
		3	\$NOSYMBOLS
		4	;ARCHIVO: MC14499.SRC
		5	;*****
		6	; EJEMPLO DE INTERFAZ MC14499 *
		7	;
		8	; Este programa actualiza un visualizador de 4 dígitos 10 veces *
		9	; por segundo utilizando interrupciones. Los dígitos son LEDs de *
		10	; 7 segmentos controlados por un decodificador/controlador MC14499 *
		11	; conectado a P1.5 (-ENABLE), P1.6 (CLOCK) y P1.7 (DATA IN). *
		12	; Las interrupciones son generadas por la línea TIMER OUT del *
		13	; 8155 conectado a -INT0. La señal TIMER OUT oscila a 500 Hz y *
		14	; genera una interrupción en cada transición de 1 a 0. Utilizamos *
		15	; un contador de interrupciones para actualizar el visualizado *
		16	; cada 50 interrupciones para una frecuencia de actualización *
		17	; de 10 Hz.
		18	;
		19	; El ejemplo ilustra el concepto de primero/segundo plano para los *
		20	; sistemas controlados por interrupciones. Una vez que se inicializa *
		21	; el 8155 y las interrupciones Externa 0 están habilitadas,
		22	; el programa regresa al programa de monitoreo. El MON51 por sí *
		23	; mismo no utiliza interrupciones; sin embargo, se ejecuta como es *
		24	; costumbre en el primer plano mientras que las interrupciones se *
		25	; llevan a cabo en el segundo plano. Si el comando SI (establece *
		26	; memoria interna) del MON51 se utiliza para cambiar las ubicaciones *
		27	; de DIGITOS o DIGITOS+1, entonces el valor escrito se observará *
		28	; inmediatamente (dentro de 0.1 s) en el visualizador LED.
		29	;
		30	;*****
00BC		31	MON51 CODE 00BCH ;entrada a MON51 (V12)
0100		32	X8155 XDATA 0100H ;dirección del 8155
0104		33	TEMPORIZADOR XDATA X8155 + 4 ;registros del temporizador
0FA0		34	CONTEO EQU 4000 ;genera interrupción cada 2000 us
0040		35	MODO EQU 0100000B ;bits del modo del temporizador
0097		36	DIN BIT P1.7 ;líneas de interfaz al MC14499
0096		37	CLOCK BIT P1.6
0095		38	ENABLE BIT P1.5
		39	
----		40	DSEG AT 70H ;segmento interno absoluto
0070		41	DIGITOS: DS 2 ;(no hay conflicto con MON51)
0072		42	I CONTEO: DS 1
		43	
----		44	CSEG AT 8000H
8000 028015		45	LJMP PRINCIPAL ;punto de entrada al programa
8003 028031		46	LJMP EX0RSI ;interrupción del 8155
8006 02805D		47	LJMP T0RSI ;interrupción del temporizador 0
8009 02805D		48	LJMP EX1RSI ;interrupción externa 1
800C 02805D		49	LJMP T1RSI ;interrupción del temporizador 1
800F 02805D		50	LJMP PSRSI ;interrupción del puerto serial
8012 02805D		51	LJMP T2RSI ;interrupción del temporizador 2
		52	
		53	;*****
		54	; PROGRAMA PRINCIPAL INICIA (INICIALIZA 8155 Y HABILITA INTERRUP.) *
		55	;*****
8015 900104		56	PRINCIPAL: MOV DPTR,#TEMPORIZADOR;inicializa el temporizador 8155
8018 74A0		57	MOV A,#LOW(CONTEO)
801A F0		58	MOVX @DPTR,A
801B A3		59	INC DPTR ,inicializa registro alto
801C 744F		60	MOV A,#HIGH(CONTEO) OR MODO
801E F0		61	MOVX @DPTR,A
801F 900100		62	MOV DPTR,#X8155 ;registro de comandos del 8155
8022 74C0		63	MOV A,#0COH ;comando para iniciar temporizador

FIGURA 11–6a

Software para la interfaz al MC14499

```

8024 F0      64          MOVX @DPTR,A      ;onda cuadrada de 500 Hz
8025 757232 65          MOV  ICONEO,#50   ;inicializa contador de interrupciones
8028 D2AF    66          SETB EA        ;habilita interrupciones
802A D2A8    67          SETB EX0       ;habilita interrupción externa 0
802C D288    68          SETB IT0       ;disparado por borde negativo
802E 0200BC  69          LJMP MON51     ;regresa al MON51
802F          70
8030          71          ****
8031 D57205  74          EX0RSI:    DJNZ ICONEO,SALIDA ;a la 50a interrupción,
8032 757232  75          MOV  ICONEO,#50   ; reinicializa contador y
8033 113A    76          ACALL ACTUALIZACION ; actualiza visualizador LED
8034 32      77          SALIDA:    RETI
8035          78
8036          79          ****
8037          80          ; ACTUALIZA VISUALIZADOR LED DE 4 DIGITOS (EJECUCION = 84 us)
8038          81          ;
8039          82          ; ENTRADA: Cuatro dígitos en BCD en ubicaciones en memoria interna
8040          83          ;           DIGITOS y DIGITOS+1 (dígito más significativo en el
8041          84          ;           nibble superior de DIGITOS)
8042          85          ; SALIDA: visualizador de MC14499 actualizado
8043          86          ; UTILIZA: P1.5, P1.6, P1.7
8044          87          ; Todas las ubicaciones en memoria y registros intactos
8045          88          ****
8046 C0E0    89          ACTUALIZACION: PUSH ACC      ;almacena acumulador en la pila
8047 C295    90          CLR HABILITA   ;prepara al MC14499
8048 E570    91          MOV A,DIGITOS  ;obtener primeros dos dígitos
8049 114B    92          ACALL SALIDA8   ;enviar dos dígitos
8050 E571    93          MOV A,DIGITOS + 1 ;obtener segundo byte
8051 114B    94          ACALL SALIDA8   ;enviar los últimos dos dígitos
8052 D295    95          SETB HABILITA   ;deshabilita MC14499
8053 D0E0    96          POP ACC       ;recupera ACC de la pila
8054 22      97          RET
8055          98
8056          99          ****
8057 7F08    100         ; ENVIA 8 BITS EN EL ACUMULADOR AL MC14499 (MSB PRIMERO)
8058          101         ****
8059          102         USING 0       ;asuma que banco de registros 0 está habilitado
8060 C007    103         SALIDA8:    PUSH AR7      ;almacena R7 en la pila
8061 7F08    104         MOV  R7,#8     ;utiliza R7 como el contador de bit
8062 33      105         DENUEVO:   RLC  A        ;coloca bit en bandera C
8063 9297    106         MOV  DIN,C     ;envía bandera C al MC14499
8064 C296    107         CLR  CLOCK    ;pulso bajo de 3 us en la línea de sincronización
8065 00      108         NOP
8066 00      109         NOP
8067 D296    110         SETB CLOCK    ;(anchura mínima del pulso es
8068 DFF5    111         DJNZ R7,DENUEVO ;de 2 us)
8069 D007    112         POP AR7      ;repetir hasta que todos los 8 bits se hayan enviado
8070 22      113         RET
8071          114
8072          115         ****
8073          116         ; INTERRUP. NO UTILIZADAS (ERROR;REGRESAR AL PROGRAMA DE MONITOREO)
8074          117         ****
8075          118         T0RSI:
8076          119         EX1RSI:
8077          120         T1RSI:
8078          121         PSRSI:
8079          122         T2RSI:    CLR EA      ;apaga interrupciones y
8080 0200BC  123         LJMP MON51     ; regresa al MON51
8081          124         END

```

FIGURA 11-6b

continuación

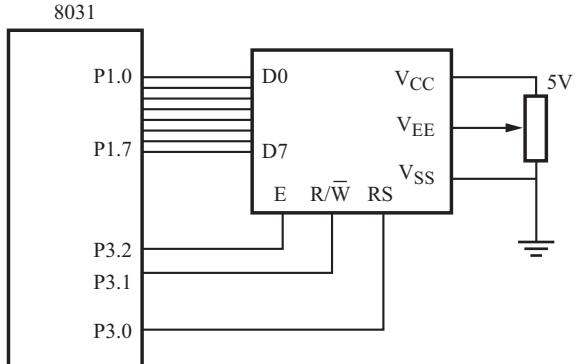
- Rutina de servicio de interrupciones externa (EXARSI; líneas 74 a 77)
- Subrutina de actualización de visualizador LED (ACTUALIZACION; líneas 89 a 97)
- Subrutina de byte de salida (SALIDA8; líneas 103 a 113)
- Código para manipular interrupciones no implementadas (líneas 118 a 123)

El programa está escrito para ser ejecutado en la dirección 8000H del circuito integrado 6264 RAM de la SBC-51. Debido a que las interrupciones se vectorizan a través de las ubicaciones localizadas en la parte inferior de la memoria, el programa de monitoreo incluye una tabla de saltos que sirve para redireccionar las interrupciones hacia las direcciones comenzando en la dirección 8000H. (Consulte el apéndice G). El punto de entrada del programa es por conveniencia 8000H; sin embargo, una instrucción LJMP (línea 45; consulte la figura 11-6) transfiere el control a la etiqueta PRINCIPAL. Todas las instrucciones de inicialización están contenidas en las líneas 56 a 68. La sección PRINCIPAL termina saltando de regreso al programa de monitoreo.

11.5 INTERFAZ CON PANTALLAS DE CRISTAL LÍQUIDO (LCDs)

En la sección anterior vimos cómo utilizar los LED de 7 segmentos para propósitos de visualización. La pantalla de 7 segmentos es suficiente para presentar números y caracteres simples, pero algunos caracteres más complejos requieren emplear alternativas tales como la pantalla de cristal líquido (LCD). Una aplicación muy popular de las LCD se encuentra en las calculadoras científicas. En esta sección mostraremos cómo hacer una interfaz a una LCD simple que consta de dos líneas de 16 caracteres, donde cada carácter está formado por una matriz de puntos de 5×7 . La mayoría de los LCD son compatibles con el estándar HD44780 de Hitachi. La figura 11-7 muestra las conexiones simples y directas que hay entre el 8051 y una LCD compatible con el HD44780.

FIGURA 11-7
Interfaz a una LCD



EJEMPLO Objetivo del diseño**11.3**

Suponga que hay caracteres en ASCII almacenados en las ubicaciones de memoria RAM interna 30H a 7FH. Escriba un programa para presentar en forma continua estos caracteres en la LCD, 16 caracteres a la vez.

La figura 11-8 muestra el listado del software necesario para llevar esto a cabo. Como ilustramos anteriormente en la figura 11-7, la LCD tiene tres líneas de control: selección de registro (RS), lectura/escritura (R/W), y habilitación (E). Cuando RS = 0, se envía una palabra de comando a la LCD, mientras que si RS = 1, se enviará una palabra de datos. R/W = 1 significa

```

1      $DEBUG
2      $NOSYMBOLS
3      $NOPAGING
4      ;ARCHIVO: LCD.SRC
5      ;*****                                                 *
6      ;          EJEMPLO DE INTERFAZ A UNA LCD             *
7      ;
8      ; Este programa presenta continuamente en la LCD los   *
9      ; caracteres en ASCII almacenados en las ubicaciones   *
10     ; de memoria RAM interna 30H a 7FH.                   *
11     ;*****                                                 *
12
00B0    13           RS      EQU      P3.0
00B1    14           RW      EQU      P3.1
00B2    15           E       EQU      P3.2
0090    16           DBUS    EQU      P1
REG     17           PTR     EQU      R0
REG     18           CONTEO  EQU      R1
19
8000    20           ORG     8000H
8000 1115 21 PRINCIPAL: ACALL INIT      ; inicializa LCD
8002 7916 22           MOV CONTEO, #16 ; inicializa contador de
                                         ; caracteres
8004 7830 23 INICIO:   MOV PTR, #30H   ; inicializa apuntador
8006 E6   24 SIGUIENTE: MOV A, @PTR
8007 113A 25           ACALL PRESENTA ; presenta en LCD
8009 08   26           INC PTR    ; apunta a la siguiente
                                         ; ubicación
800A D904 27           DJNZ CONTEO, PRUEBA ; ¿es fin de línea?
800C 7916 28           MOV CONTEO, #16 ; sí: reinicializa el conteo
800E 1127 29           ACALL NUEVO   ; y actualiza LCD
8010 B880F3 30 PRUEBA:  CJNE PTR, #80H, SIGUIENTE ; ¿última ubicación?
                                         ; no: continúa en SIGUIENTE
8013 80EF 31           SJMP INICIO ; sí: regresa al INICIO
32
33
34     ;*****                                                 *
35     ; Inicialización de la LCD                         *
36     ;*****                                                 *
37
8015 7438 37 INIC:    MOV A, #38H    ; 2 líneas, matriz
                                         ; de 5 × 7
8017 113D 38           ACALL ESPERA ; espera a que LCD esté libre

```

FIGURA 11-8a

Software para la interfaz a una LCD

```

8019 C2B0      39      CLR RS           ;prepara comando para salida
801B 114B      40      ACALL SALIDA    ;envíalo
801D 740E      41
801F 113D      42      MOV A, #0EH       ;LCD encendida, cursor encendido
801F 113D      43      ACALL ESPERA    ;espera a que LCD esté libre
8021 C2B0      44      CLR RS           ;prepara comando para salida
8023 114B      45      ACALL SALIDA    ;envíalo
8025 7401      46
8027 113D      47      NUEVO:        MOV A, #01H       ;limpia LCD
8027 113D      48      ACALL ESPERA    ;espera a que LCD esté libre
8029 C2B0      49      CLR RS           ;prepara comando para salida
802B 114B      50      ACALL SALIDA    ;envíalo
802D 7480      51
802F 113D      52      MOV A, #80H       ;cursor: línea 1, pos. 1
802F 113D      53      ACALL ESPERA    ;espera a que LCD esté libre
8031 C2B0      54      CLR RS           ;prepara comando para salida
8033 114B      55      ACALL SALIDA    ;envíalo
8035 22        56
8035 22        57      RET
8035 22        58
8035 22        59      ;*****
8035 22        60      ; Presenta datos en LCD
8035 22        61      ;*****
8035 22        62
8036 113D      63      PRESENTA: ACALL ESPERA ;espera a que LCD esté libre
8038 D2B0      64      SETB RS          ;prepara dato para salida
803A 114B      65      ACALL SALIDA    ;envíalo
803C 22        66      RET
803C 22        67
803C 22        68      ;*****
803C 22        69      ; Espera a que LCD esté libre
803C 22        70      ;*****
803D C2B0      71      ESPERA:   CLR RS           ;comando
803F D2B1      72      SETB RW          ;lectura
8041 D297      73      SETB DBUS.7     ;DB7 = entrada
8043 D2B2      74      SETB E            ;transición de 1 a 0 para
8045 C2B2      75      CLR E             ; habilitar LCD
8047 2097F3    76      JB DBUS.7, ESPERA
804A 22        77      RET
804A 22        78
804A 22        79      ;*****
804A 22        80      ; Salida a LCD
804A 22        81      ;*****
804B F590      82      SALIDA:  MOV DBUS, A
804D C2B1      83      CLR RW
804F D2B2      84      SETB E
8051 C2B2      85      CLR E
8053 22        86      RET
8053 22        87      END

```

FIGURA 11–8b
continuación

TABLA 11-1
Códigos de comandos del LCD

Código	Descripción
1	Limpia pantalla de visualización
2	Regresa a posición inicial
4	Desplaza cursor a la izquierda
5	Desplaza visualizador a la derecha
6	Desplaza cursor a la derecha
7	Desplaza visualizador a la izquierda
8	Visualizador apagado, cursor apagado
A	Visualizador apagado, cursor encendido
C	Visualizador encendido, cursor apagado
E	Visualizador encendido, cursor con destello
F	Visualizador encendido, cursor sin destello
10	Desplaza cursor a la izquierda
14	Desplaza cursor a la derecha
18	Desplaza todo el visualizador a la izquierda
1C	Desplaza todo el visualizador a la derecha
80	Forzar el cursor al inicio de la primera línea
C0	Forzar el cursor al inicio de la segunda línea
38	Visualizador de 2 líneas, matriz de 5×7

que el 8051 está leyendo de la LCD, $R/\bar{W} = 0$ significa que el 8051 está escribiendo a la LCD. Por último, la señal E debe activarse a nivel alto y después llevarse a nivel bajo para enviar la señal a la LCD de que se requiere su atención para procesar el comando o dato que está disponible en sus terminales de datos. La tabla 11-1 muestra los códigos de los comandos que se pueden enviar a la LCD.

El software llama primero a una subrutina INIC para inicializar los modos de visualización y la configuración antes de proceder a presentar una secuencia de 16 caracteres en ASCII en la LCD un carácter a la vez. Observemos que antes de enviar cualquier dato a la LCD, primero se llama a una subrutina de ESPERA para esperar mientras la LCD esté ocupada. Esto se puede determinar desde la terminal 7 del bus de datos de la LCD, D7. El software llamará a la subrutina SALIDA sólo cuando la LCD ya no esté ocupada para enviar un byte de datos o de comando al bus de datos de la LCD.

11.6 INTERFAZ CON UN ALTAZOZ

La figura 11-9 muestra una interfaz entre un 8031 y un altavoz. Los altavoces pequeños, como los instalados en computadoras personales o juguetes para niños, pueden controlarse mediante una compuerta lógica, como se muestra. Un lado de la bobina del altavoz está conectado a +5 voltios

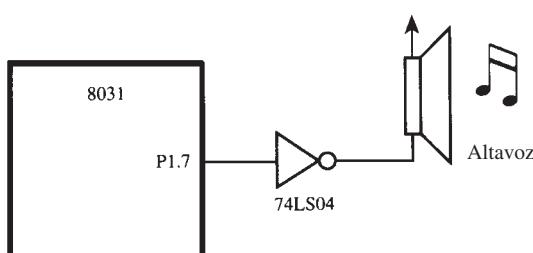


FIGURA 11-9
Interfaz a un altavoz

y el otro a la salida de un inversor lógico 74LS04. Requerimos del inversor debido a que tiene más alta capacidad de control que las líneas de puerto en el 8031.

EJEMPLO	Objetivo del diseño
11.4	Escriba un programa controlado por interrupciones para que reproduzca de manera continua una escala musical de LA mayor.

Las melodías son fáciles de generar desde un 8051 utilizando una interfaz a un altavoz simple. Comencemos con algo de teoría musical. La frecuencia necesaria para cada nota ubicada en una escala musical de LA mayor está escrita en el bloque de comentarios al inicio del listado de software mostrado en la figura 11-10 (líneas 14 a 21). La primera frecuencia es de 440 Hz (llamada “LA arriba de DO intermedio”), y es la frecuencia de referencia internacional para instrumentos musicales utilizando una escala templada igualmente (por ejemplo, el piano). La frecuencia de todas las demás notas puede determinarse al multiplicar esta frecuencia por $2^{n/12}$, donde n es el número de pasos (o “semitones”) necesarios para llegar a la nota que estamos calculando. El ejemplo más fácil es A', una octava, o 12 pasos, arriba de LA, la cual tiene una frecuencia de $440 \times 2^{12/12} = 880$ Hz. Esta es la última nota en nuestra escala musical. (Consulte la figura 11-10, línea 21). Con referencia a la nota inferior (o “raíz”) en cualquier escala mayor en pasos es 2, 4, 5, 7, 9, 11 y 12. Por ejemplo, la nota “MI” de la figura 11-10 (línea 18) está siete pasos por arriba de la raíz; por lo tanto, su frecuencia es de $440 \times 2^{7/12} = 659.26$ Hz.

Para crear una escala musical, requerimos dos sincronizaciones: la sincronización de una nota a la siguiente y la sincronización para disparar el bit de puerto que controla el altavoz. Estas dos sincronizaciones son bastante diferentes. Para reproducir la melodía a una velocidad de cuatro notas/segundo, por ejemplo, requerimos de un vencimiento de sincronización (o interrupción) cada 250 ms. Para crear la frecuencia de la primera nota en la escala, necesitamos de un vencimiento de sincronización cada 1.136 ms. (Consulte la figura 11-10, línea 14).

El software ilustrado en la figura 11-10 inicializa ambos temporizadores al modo de temporizador de 16 bits (línea 43), y utiliza las interrupciones del temporizador 0 para efectuar los cambios en las notas y las interrupciones del temporizador 1 para la frecuencia de las notas. Los valores de recarga para las frecuencias de las notas se leen desde una tabla de búsqueda (líneas 90 a 104). Consulte el listado de la figura 11-10 para obtener más detalles.

11.7 INTERFAZ CON RAM NO VOLÁTIL

Las memorias RAM no volátiles (NVRAMs) son memorias de semiconductores que mantienen su contenido aun en ausencia de energía. Las NVRAM incorporan tanto las células de RAM estática como células de ROM eléctricamente borrable y programable (EEPROM). Cada bit de la RAM estática se superpone con un poco de la EEPROM. Podemos transferir datos de ida y vuelta entre las dos memorias.

Las NVRAM ocupan un nicho importante en las aplicaciones basadas en microprocesadores y microcontroladores. Se utilizan para almacenar datos de inicialización o parámetros que el usuario cambia ocasionalmente pero que deben retenerse aunque se pierda la capacidad.

Como ejemplo, muchos de los diseños de VDT evitan el uso de interruptores DIP (los cuales son propensos a fallas) y utilizan NVRAMs para almacenar información de inicialización tal como la velocidad en baudios, la paridad encendido/apagado, la paridad impar/par, y así sucesivamente. Cada vez que la VDT se enciende, estos parámetros se recuperan de la NVRAM y el sistema se inicializa adecuadamente. Cuando el usuario cambia un parámetro (mediante el teclado), los nuevos valores se almacenan en la NVRAM.

Los módems con la característica de automarcado a menudo almacenan números telefónicos en la memoria interna. Estos números telefónicos, por lo general, se guardan en una NVRAM para que puedan retenerse incluso durante alguna eventual falla de energía. Se pueden almacenar

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$nopaging
		3	\$nosymbols
		4	;ARCHIVO: ESCALA.SRC
		5	;*****
		6	; EJEMPLO DE INTERFAZ A UN ALTAVOZ
		7	;
		8	; Este programa toca una escala musical de LA mayor utilizando un altavoz
		9	; controlado por un inversor a través de P1.7
		10	;*****
		11	;
		12	; Nota Frecuencia (Hz) Periodo (us) Periodo/2 (us)
		13	----- ----- ----- -----
		14	; LA 440.00 2273 1136
		15	; SI 493.88 2025 1012
		16	; DO# 554.37 1804 902
		17	; RE 587.33 1703 851
		18	; MI 659.26 1517 758
		19	; FA# 739.99 1351 676
		20	; SOL# 830.61 1204 602
		21	; LA' 880.00 1136 568
		22	;*****
00BC		23	MONITOR CODE 00BCH ;punto de entrada al MON51 (V12)
3CBO		24	CONTEO EQU -50000 ;0.05 segundos por interrupción
0005		25	REPETIR EQU 5 ;5 x 0.05 = 0.25 segundos/nota
		26	;
		27	;*****
		28	; Nota: X3 no está instalado en SBC-51, por lo tanto las interrupciones son
		29	; dirigidas a la siguiente tabla de saltos comenzando en 8000H
		30	;
		31	;*****
8000		32	ORG 8000H ;puntos de entrada a RAM para...
8000 028015		33	LJMP PRINCIPAL ; programa principal
8003 02806B		34	LJMP EXTORSI ; interrupción externa 0
8006 028025		35	LJMP TORSI ; interrupción del temporizador 0
8009 02806B		36	LJMP EXT1RSI ; interrupción externa 1
800C 02803A		37	LJMP T1RSI ; interrupción del temporizador 1
800F 02806B		38	LJMP PSRSI ; interrupción del puerto serial
8012 02806B		39	LJMP T2RSI ; interrupción del temporizador 2
		40	;
		41	;*****
		42	; PROGRAMA PRINCIPAL INICIA
		43	;*****
8015 758911		44	PRINCIPAL: MOV TMOD,#11H ;ambos temporizadores en modo de 16 bits
8018 7F00		45	MOV R7,#0 ;utiliza R7 como contador de nota
801A 7E05		46	MOV R6,#REPETIR ;utiliza R6 como contador de interrupción
801C 75A88A		47	MOV IE,#8AH ;interrupciones del temporizador 0 y 1 encendidas
801F D28F		48	SETB TF1 ;forzar interrupción del temporizador 1
8021 D28D		49	SETB TF0 ;forzar interrupción del temporizador 0
8023 80FE		50	SJMP \$;ZzZzZzZz tiempo para una siesta
		51	;
		52	;*****
		53	; RUTINA DE SERVICIO DE INTERRUP. DEL TEMP. 0 (CADA 0.05 SEG.)
		54	;*****
8025 C28C		55	TORSI: CLR TR0 ;detener temporizador
8027 758C3C		56	MOV TL0,#HIGH (CONTEO) ;recarga
802A 758AB0		57	MOV TL0,#LOW (CONTEO)
802D DB08		58	DJNZ R6,SALIDA ;si no es 5a int, salir
802F 7E05		59	MOV R6,#REPETIR ;si 5a, reinicializar
8031 0F		60	INC R7 ;incrementa nota
8032 BF0C02		61	CJNE R7,#LONGITUD,SALIDA ;¿más allá de la última nota?
8035 7F00		62	MOV R7,#0 ;sí: reinicializar, A=440 Hz
8037 D28C		63	SALIDA: SETB TR0 ;no: inicia temporizador, vuelve
8039 32		64	RETI ; a ZzZzZzZz

FIGURA 11–10a

Software para la interfaz a un altavoz

```

65
66 ;*****
67 ; RUTINA DE SERVICIO DE INTERRUPCIÓN TEMPORIZADOR 1 (TONO DE NOTA) *
68 ;
69 ; Nota: Las frecuencias de salida son un poco inexactas debido a la longitud *
70 ; de esta ISR. Los valores de recarga del temporizador necesitan ajustarse. *
71 ;
72 ;*****
803A B297 73 T1RSI: CPL P1.7 ;¡música maestro!
803C C28E 74 CLR TR1 ;detiene el temporizador
803E EF 75 MOV A,R7 ;obtiene contador de notas
803F 23 76 RL A ;multiplica (2 bytes/nota)
8040 128050 77 CALL OBTENBYTE ;obtiene byte superior del conteo
8043 F58D 78 MOV TH1,A ;coloca en registro sup. del temporizador
8045 EF 79 MOV A,R7 ;obtiene contador de notas otra vez
8046 23 80 RL A ;se alinea en límites de palabra
8047 04 81 INC A ;pasamos del byte superior (al fin!)
8048 128050 82 CALL OBTENBYTE ;obtiene byte inferior del conteo
804B F58B 83 MOV TL1,A ;coloca en reg. inferior del temporizador
804D D28E 84 SETB TR1 ;inicia el temporizador
804F 32 85 RETI ;tiempo para un descanso
86
87 ;*****
88 ; OBTIENE BYTE DE TABLA DE BUSQUEDA DE NOTAS EN ESCALA DE LA MAYOR *
89 ;*****
8050 04 90 OBTENBYTE: INC A ;subrutina de búsqueda en tabla
8051 83 91 MOVC A, @A+PC
8052 22 92 RET
8053 FB90 93 TABLA: DW -1136 ;LA
8055 FB90 94 DW -1136 ;LA (toca de nuevo; media nota)
8057 FC0C 95 DW -1012 ;SI (cuarta nota, etc.)
8059 FC7A 96 DW -902 ;DO# - tercera mayor
805B FCAD 97 DW -851 ;RE
805D FD0A 98 DW -758 ;MI - quinta perfecta
805F FD5C 99 DW -676 ;FA#
8061 FDA6 100 DW -602 ;SOL#
8063 FDC8 101 DW -568 ;LA'
8065 FDC8 102 DW -568 ;LA' (toca 4 veces; nota completa)
8067 FDC8 103 DW -568
8069 FDC8 104 DW -568
000C 105 LONGITUD: EQU ($ - TABLA) / 2 ;LONGITUD = # de notas
106
107 ;*****
108 ; INTERRUPCIONES NO USADAS - REGRESA AL PROGRAMA MONITOR (ERROR) *
109 ;*****
110 EXT0RSI:
111 EXT1RSI:
112 PSRSI:
806B C2AF 113 T2RSI: CLR EA ;apaga interrupciones y
806D 0200BC 114 LJMP MONITOR ; regresa a MON51
115 END

```

FIGURA 11–10b
continuación

256 Bit

Commercial
IndustrialX2444
X2444I

16 x 16 Bit

Nonvolatile Static RAM**FEATURES**

- Ideal for use with Single Chip Microcomputers
 - Static Timing
 - Minimum I/O Interface
 - Serial Port Compatible (COPSTM, 8051)
 - Easily Interfaces to Microcontroller Ports
 - Minimum Support Circuits
- Software and Hardware Control of Nonvolatile Functions
 - Maximum Store Protection
- TTL Compatible
- 16 x 16 Organization
- Low Power Dissipation
 - Active Current: 15 mA Typical
 - Store Current: 8 mA Typical
 - Standby Current: 6 mA Typical
 - Sleep Current: 5 mA Typical
- 8 Pin Mini-DIP Package

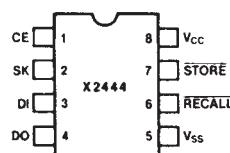
DESCRIPTION

The Xicor X2444 is a serial 256 bit NOVRAM featuring a static RAM configured 16 x 16, overlaid bit for bit with a nonvolatile E²PROM array. The X2444 is fabricated with the same reliable N-channel floating gate MOS technology used in all Xicor 5V nonvolatile memories.

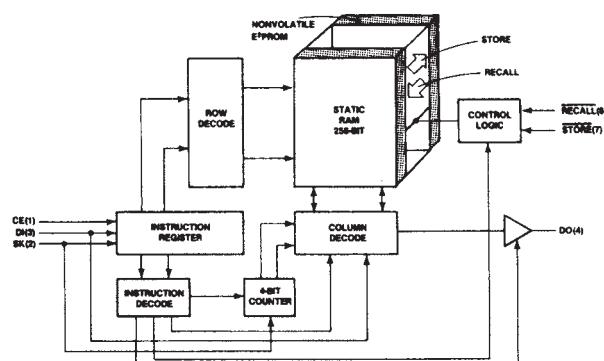
The Xicor NOVRAM design allows data to be transferred between the two memory arrays by means of software commands or external hardware inputs. A store operation (RAM data to E²PROM) is completed in 10 ms or less and a recall operation (E²PROM data to RAM) is completed in 2.5 μ s or less.

Xicor NOVRAMs are designed for unlimited write operations to RAM, either from the host or recalls from E²PROM and a minimum 100,000 store operations. Data retention is specified to be greater than 100 years.

COPSTM is a trademark of National Semiconductor Corp.

PIN CONFIGURATION

0042-1

FUNCTIONAL DIAGRAM

0042-2

PIN NAMES

CE	Chip Enable
SK	Serial Clock
DI	Serial Data In
DO	Serial Data Out
RECALL	Recall
STORE	Store
Vcc	+5V
Vss	Ground

FIGURA 11-11

Portada para la hoja de datos de la RAM no volátil X244

diez números telefónicos con siete dígitos cada uno en 35 bytes (mediante la codificación de cada dígito en notación BCD).

La NVRAM utilizada para este ejemplo de una interfaz es la X2444 fabricada por Xicor,¹ una compañía que se especializa en NVRAMs y EEPROMs. La X2444 contiene 256 bits de RAM estática superpuesta con 256 bits de EEPROM. Los datos pueden transferirse entre las dos memorias, ya sea mediante instrucciones enviadas desde el procesador a través de la interfaz serial o por el disparo de las entradas externas STORE y RECALL. Los datos no volátiles se retienen en la EEPROM, mientras que en la RAM se acceden y actualizan datos independientes. La primera página de la hoja de datos de la X2444 presenta un resumen de sus características, y se reproduce en la figura 11-11.

En este ejemplo de una interfaz no utilizamos las líneas de STORE y RECALL. Los diferentes modos de operación se seleccionan enviando instrucciones seriales de la X2444 a través de las terminales de puerto del 8051.

La figura 11-12 muestra la interfaz al 8051. Sólo se utilizan tres líneas:

- P1.0—SK (reloj serial)
- P1.1—CE (habilitador de chip)
- P1.2—DI/DO (entrada/salida de datos)

Las instrucciones se envían a la X2444 elevando CE a nivel alto y sincronizando entonces un código de operación de 8 bits a la X2444 a través de las líneas de SK y DI/DO. Necesitamos los siguientes códigos de operación para este ejemplo:

Instrucción	Código de operación	Operación
RCL	85H	Recupera datos de EEPROM a RAM
WREN	84H	Activa latch de habilitación de escritura
STORE	81H	Almacena datos de RAM a EEPROM
WRITE	1AAAA011B	Escribe datos a la dirección AAAA en RAM
READ	1AAAA111B	Lee datos de la dirección AAAA en RAM

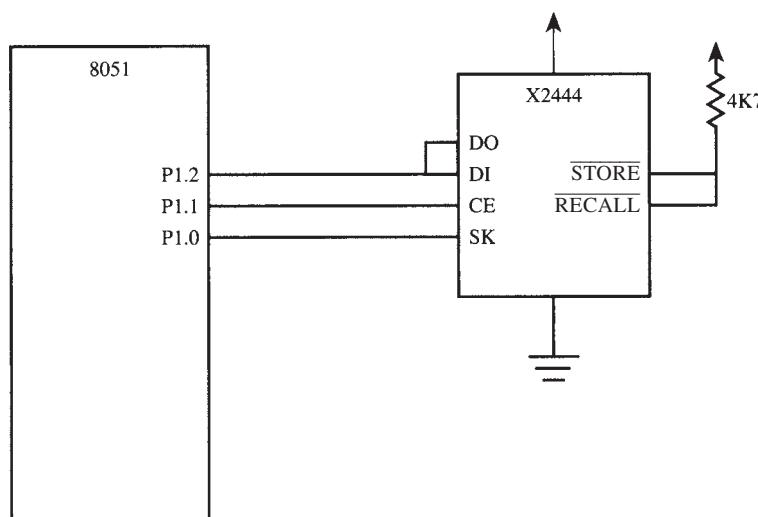


FIGURA 11-12

Interfaz a la RAM no volátil X2444

¹XICOR, Inc., 851 Buckeye Court, Milpitas, CA 95035.

EJEMPLO Objetivo del diseño**11.5**

Escriba los siguientes dos programas. El primero, llamado ALMACENA, copia el contenido de las ubicaciones internas 60H a 7FH del 8051 a la memoria X2444 EEPROM. El segundo programa, denominado RECUPERA, lee los datos previamente almacenados desde la X2144 EEPROM y los recupera a las ubicaciones 60H a 7FH.

Estos son dos programas distintos. Por lo general, el programa ALMACENA se ejecuta cada vez que se cambia la información no volátil (por ejemplo, si un usuario altera un parámetro de configuración). El programa RECUPERA se ejecuta cada vez que el sistema se enciende o reinicializa. Para este ejemplo, la información no volátil está almacenada en las ubicaciones internas 60H a 7FH del 8051 (quizás para que un programa de control ejecutado en firmware pueda tener acceso a ella). La figura 11-13 muestra el listado del software.

Las operaciones para almacenar y recuperar datos involucran los siguientes pasos:

Escribe datos a la X2444

1. Ejecuta instrucción RCL (recuerda).
2. Ejecuta instrucción WREN (activa latch de habilitación de escritura).
3. Escribe los datos a la RAM X2444.
4. Ejecuta la instrucción STO (almacena RAM en EEPROM).
5. Ejecuta instrucción SLEEP.

Lee datos de la X2444

1. Ejecuta instrucción RCL (recuerda).
2. Lee datos de la RAM X2444.
3. Ejecuta instrucción SLEEP.

Como un ejemplo de lo que los controladores en software deben hacer, la figura 11-14 muestra el diagrama de sincronización para enviar la instrucción RCL a la X2444. En realidad, varios de los bits “no importan” (como están especificados en la hoja de datos); sin embargo, se muestran como ceros en la figura.

En cuanto a las instrucciones de ESCRITURA y LECTURA de datos, la sincronización es un poco diferente. Esto es, el código de operación de 8 bits va seguido inmediatamente por 16 bits de datos y la señal de habilitación de chip se mantiene a nivel alto para todos los 24 bits. Para la instrucción de lectura, los ocho bits (el código de operación) son escritos a la X2444, y entonces se leen 16 bits de datos de la X2444. Utilizamos subrutinas separadas para leer ocho bits (L_BYT; líneas 106 a 112) y escribir ocho bits (E_BYT; líneas 117 a 123). Consulte el listado del software para conocer detalles específicos.

11.8 EXPANSIÓN DE ENTRADAS/SALIDAS

El 8051 tiene cuatro puertos de entrada/salida: puerto 0 a puerto 3. Si se utiliza la memoria externa, entonces todo o parte de los puertos 0 y 2 se tomará como líneas de datos y/o direcciones, reduciendo así el número de líneas de E/S disponibles para uso de E/S de propósito general. En esta sección examinaremos dos maneras simples de incrementar el número de líneas de E/S. Esto se llama expansión de las E/S.

11.8.1 Uso de registros de desplazamiento

El ejemplo siguiente ilustra una manera simple de incrementar el número de líneas de entradas en el 8051. Se utilizan tres líneas de puerto como interfaz a múltiples registros de desplazamiento de entrada en paralelo y salida serial 74HC165 (en este ejemplo, 2). (Consulte la figura 11-15).

LOC	OBJ	LINE	SOURCE	
		1	\$DEBUG	
		2	\$NOPAGING	
		3	\$NOSYMBOLS	
		4	;ARCHIVO: NVRAM.SRC	
		5	;*****	*
		6	;	*
		7	;	*
		8	; A continuación se muestran dos subrutinas, donde una ALMACENA o RECUPERA datos	*
		9	; entre una memoria RAM no-volátil X2444 y 32 bytes de la memoria RAM interna	*
		10	; del 8051.	*
		11	;*****	*
0085		12	RECUERDA EQU 85H	;instrucción recuerda de la X2444
0084		13	ESCRIBE EQU 84H	;instrucción de habilitación de escritura de la X2444
0081		14	GUARDA EQU 81H	;instrucción guarda de la X2444
0082		15	DUERME EQU 82H	;instrucción duerme de la X2444
0083		16	E_DATOS EQU 83H	;instrucción para escritura de datos de la X2444
0087		17	L_DATOS EQU 87H	;instrucción para lectura de datos de la X2444
00BC		18	MON51 EQU 00BCH	;punto de entrada al MON51 (V12)
0020		19	LONGITUD EQU 32	;32 bytes almacenados/recuperados
0092		20	DIN BIT P1.2	;líneas de interfaz a la X2444
0091		21	ENABLE BIT P1.1	
0090		22	CLOCK BIT P1.0	
		23		
----		24	DSEG AT 60H	
0060		25	NVRAM: DS LONGITUD	;60H - 7FH almacenados/recuperados
		26		
----		27	CSEG AT 8000H	
8000	110A	28	EX2444: ACALL ALMACENA	;8000H punto de entrada para escritura
8002	0200BC	29	LJMP MON51	
8005	1149	30	LX2444: ACALL RECUPERA	;8005H punto de entrada para lectura
8007	0200BC	31	LJMP MON51	
		32		
		33	;*****	*
		34	; ALMACENA UBICACIONES EN RAM 60H - 7FH DEL 8051 A LA X2444 NVRAM	*
		35	;*****	*
800A	7860	36	ALMACENA: MOV R0,#NVRAM	;R0 -> ubicaciones a almacenar
800C	C291	37	CLR ENABLE	;deshabilita la X2444
800E	7485	38	MOV A,#RECUERDA	;instrucción recuerda
8010	D291	39	SETB ENABLE	
8012	1184	40	ACALL E_BYT	
8014	C291	41	CLR ENABLE	
8016	7484	42	MOV A,#ESCRIBE	;se prepara habilitación de escritura
8018	D291	43	SETB ENABLE	;X2444 a ser escrita
801A	1184	44	ACALL E_BYT	
801C	C291	45	CLR ENABLE	
801E	7F00	46	MOV R7,#0	;R7 = dirección de la X2444
8020	EF	47	DENUEVO: MOV A,R7	;coloca dirección en ACC
8021	23	48	RL A	;coloca en bits 3,4,5,6
8022	23	49	RL A	
8023	23	50	RL A	
8024	4483	51	ORL A,#E_DATOS	;construye instrucción de escritura
8026	D291	52	SETB ENABLE	
8028	1183	53	ACALL E_BYT	
802A	7D02	54	MOV R5,#2	
802C	E6	55	CICLO: MOV A,@R0	;obtener datos del 8051
802D	08	56	INC R0	;apunta al siguiente byte
802E	1184	57	ACALL E_BYT	;envía byte a la X2444
8030	DDPA	58	DJNZ R5,CICLO	;repetir (envía 2o byte)
8032	C291	59	CLR ENABLE	
8034	0F	60	INC R7	;incrementa dirección de la X2444
8035	BF10E8	61	CJNE R7,#16,DENUEVO	;si no hemos terminado, hacerlo de nuevo
8038	7481	62	MOV A,#GUARDA	;si terminamos, copia a EEPROM
803A	D291	63	SETB ENABLE	
8803C	1184	64	ACALL E_BYT	

FIGURA 11-13a

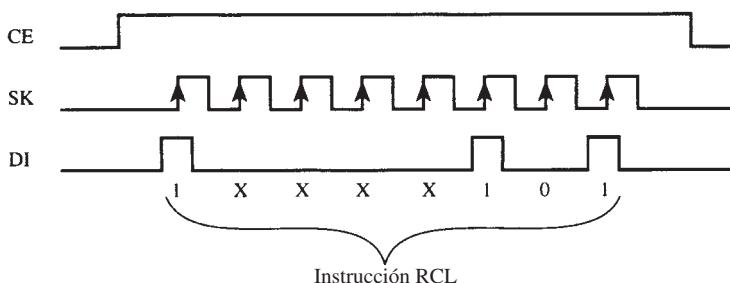
Software para la interfaz a una X2444

```

803E C291    65      CLR    ENABLE
8040 7482    66      MOV    A,#DUEERME      ;lleva a la X2444 a dormir
8042 D291    67      SETB   ENABLE
8044 1184    68      ACALL  E_BYTEx
8046 C291    69      CLR    ENABLE
8048 22      70      RET    ;LISTO!
71
72 ;*****
73 ; RECUPERA UBICACIONES EN RAM 60H - 7FH DEL 8051 A LA X2444 NVRAM *
74 ;*****
8049 7860    75      RECUPERA: MOV   R0,#NVRAM
804B C291    76      CLR    ENABLE
804D 7485    77      MOV    A,#RECUERDA    ;instrucción recuerda
804F D291    78      SETB   ENABLE
8051 1184    79      ACALL  E_BYTEx
8053 C291    80      CLR    ENABLE
8055 7F00    81      MOV    R7,#0       ;R7 = dirección de la X2444
8057 EF      82      DENUEVO2: MOV   A,R7      ;coloca dirección en ACC
8058 23      83      RL     A          ;construye instrucción de lectura
8059 23      84      RL     A
805A 23      85      RL     A
805B 4487    86      ORL    A,#L_DATOS
805D D291    87      SETB   ENABLE
805F 1184    88      ACALL  E_BYTEx      ;envía instrucción de lectura
8061 7D02    89      MOV    R5,#2       ; (+ dirección)
8063 1178    90      CICLO2: ACALL L_BYTEx      ;lee byte de datos
8065 F6      91      MOV    @R0,A      ;coloca en RAM del 8051
8066 08      92      INC    R0          ;apunta a la siguiente ubicación
8067 DDFA    93      DJNZ   R5,CICLO2
8069 C291    94      CLR    ENABLE
806B 0F      95      INC    R7          ;incrementa dirección de la X2444
806C BF10E8    96     CJNE   R7,#16,DENUEVO2 ;repetir hasta el último
806F 7482    97      MOV    A,#DUEERME      ;lleva a la X2444 a dormir
8071 D291    98      SETB   ENABLE
8073 1184    99      ACALL  E_BYTEx
8075 C291    100     CLR    ENABLE
8077 22      101     RET    ;LISTO!
102
103 ;*****
104 ; LEE UN BYTE DE DATOS DE LA X2444 *
105 ;*****
8078 7E08    106     L_BYTEx: MOV   R6,#8       ;utiliza R6 como contador de bits
807A A292    107     DENUEVO3: MOV   C,DIN      ;coloca bit de datos de la X2444 en C
807C 33      108     RLC    A          ;construye byte en el acumulador
807D D290    109     SETB   CLOCK      ;dispara línea del reloj (1 us)
807F C290    110     CLR    CLOCK
8081 DEF7    111     DJNZ   R6,DENUEVO3    ;si no es el último bit, hazlo de nuevo
8083 22      112     RET
113
114 ;*****
115 ; ESCRIBE UN BYTE DE DATOS A LA X2444 *
116 ;*****
8084 7E08    117     E_BYTEx: MOV   R6,#8       ;utiliza R6 como contador de bits
8086 33      118     DENUEVO4: RLC   A          ;coloca en C el bit a escribir
8087 9292    119     MOV    DIN,C      ;coloca en línea DATA IN de la X2444
8089 D290    120     SETB   CLOCK      ;sincroniza bit a la X2444
808B D290    121     CLR    CLOCK
808D DEF7    122     DJNZ   R6,DENUEVO4    ;si no es el último bit, hazlo de nuevo
808F 22      123     RET
124     END

```

FIGURA 11-13b
continuación

**FIGURA 11-14**

Sincronización para la instrucción recuerda (recall) de la X2444

Las entradas adicionales se muestran periódicamente pulsando la línea SHIFT/LOAD a nivel bajo. Los datos son leídos entonces al 8051 a través de la línea DATA IN y enviando pulsos a la línea CLOCK. En la línea del reloj, cada pulso desplaza los datos (“hacia abajo”, como se muestra en la figura 11-15), así que la siguiente lectura de DATA IN lee el siguiente bit, y así sucesivamente.

EJEMPLO**11.6****Objetivo del diseño**

Escriba una subrutina que copie el estado de las 16 entradas que ilustra la figura 11-15 a las ubicaciones en memoria RAM interna 25H y 26H.

La figura 11-16 muestra el software necesario para llevar esto a cabo. Observe que el ciclo del programa principal consta de llamadas a dos subrutinas: OBTEN_BYTES y PRESENTA_RESULTADOS (líneas 34 y 35). Incluimos la segunda subrutina con el propósito de ilustrar una técnica útil para efectuar la depuración cuando los recursos son limitados. La subrutina PRESENTA_RESULTADOS (líneas 72 a 83) lee los datos de las ubicaciones internas 25H y 26H y envía cada nibble a la consola como un carácter en hexadecimal. Esto proporciona una simple interfaz visual para verificar si el programa y la interfaz están funcionando. Conforme las líneas se disparan entre nivel alto y nivel bajo, los cambios aparecen inmediatamente en la consola (si la interfaz y el programa están funcionando en forma adecuada).

La subrutina OBTEN_BYTES (líneas 44 a 58) tarda $112 \mu s$ en ejecutarse cuando se utilizan dos 74HC165 y el sistema opera mediante un cristal de 12 MHz. Si las entradas se muestrenan, por ejemplo, 20 veces por segundo, OBTEN_BYTES consumiría $112 \div 50,000 = 0.2\%$ del tiempo de ejecución de la CPU. Esto tendría un impacto mínimo en los recursos del sistema; sin embargo, al incrementar el número de líneas de entrada y/o la velocidad de muestreo, el rendimiento general del sistema puede empezar a verse afectado. Consulte el listado del software para conocer más detalles.

11.8.2 Uso del 8255

El 8255 es un circuito integrado de interfaz periférica programable (PPI) que puede utilizarse para expandir las E/S del 8051. El 8255 es un chip de 40 terminales con 3 puertos de 8 bits conocidos como puerto A, puerto B y puerto C. También incluye dos señales de control de entrada activadas a nivel bajo, \overline{RD} y \overline{WR} , las cuales se utilizan para conectarse a las líneas correspondientes del 8051. Por otra parte, D0 a D7 son las terminales de datos utilizadas para conectarse al bus de datos del 8051.

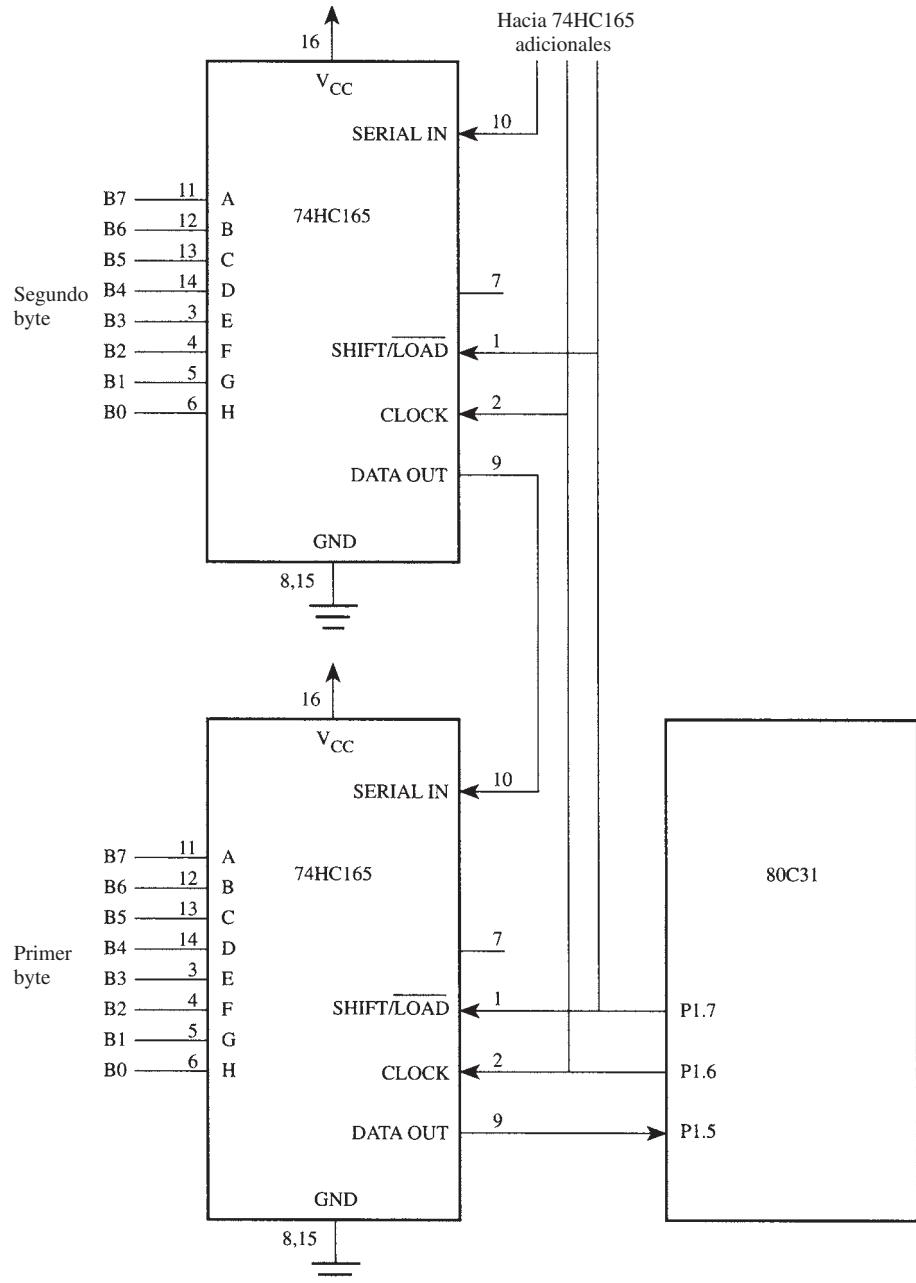


FIGURA 11–15
Interfaz a dos 74HC165

```

1 $DEBUG
2 $NOSYMBOLS
3 $NOPAGING
4 ;ARCHIVO: HC165.SRC
5 ;***** *****
6 ; EJEMPLO DE INTERFAZ A UN 74HC165
7 ;
8 ; La subrutina OBTEM_BYTES a continuación lee múltiples (en este caso dos)
9 ; registros de desplazamiento de entrada en paralelo y salida serial 74HC165
10 ; conectados a P1.7 (SHIFT/LOAD), P1.6 (CLOCK) y P1.5 (DATA OUT).
11 ; Los bytes leídos se colocan en las ubicaciones direccionables
12 ; por bit comenzando en la dirección de byte BUFER.
13 ;
14 ;*****
000D 15 CR EQU 0DH
0002 16 CONTEO EQU 2 ;número de 74HC165s
0097 17 SHIFT BIT P1.7 ;entrada SHIFT/LOAD del 74HC165
18 ; l = desplazamiento, 0 = carga
0096 19 CLOCK BIT P1.6 ;entrada de CLOCK del 74HC165
0095 20 DOUT BIT P1.5 ;salida de DATA OUT del 74HC165
0282 21 CADSAL CODE 0282H ;subrutinas en MON51 (V12)
028D 22 SALIDA2HEX CODE 028DH ;enviar byte como salida como dos caracteres en hex
01DE 23 CARSLA CODE 01DEH
24
8000 25 ORG 8000H ;empieza segmento de código en 8000H
8000 D296 26 SETB CLOCK ;activa líneas de interfaz inicialmente en
8002 D297 27 SETB SHIFT ; caso que no se haya hecho ya
8004 D295 28 SETB DOUT ;DOUT debe activarse (entrada)
29
30 ;*****
31 ; CICLO PRINCIPAL (PEQUEÑO PARA ESTE EJEMPLO)
32 ;*****
8006 128029 33 CALL ENVIA_MENSAJE_HOLA ;mensaje
8009 128011 34 REPETIR: CALL OBTEM_BYTES ;lee los 74HC165s
800C 128050 35 CALL PRESENTA_RESULTADOS ;presenta resultados
800F 80F8 36 JMP REPEAT ;ciclo
37
38 ;*****
39 ; OBTENER BYTES DE LOS 74HC165s Y COLOCARLOS EN RAM INTERNA
40 ;
41 ; Tiempo de ejecución = 112 microsegundos (@ 12 MHz).
42 ; Tiempo de ejecución para N 74HC165s = 6 + (N x 53) us
43 ;*****
44 OBTEM_BYTES:
8011 7E02 45 MOV R6,#CONTEO ;utiliza R6 como contador de bytes
8013 7825 46 MOV R0,#BUFER ;utiliza R0 como apuntador a búfer
8015 C297 47 CLR SHIFT ;carga a 74HC165s mediante
8017 D297 48 SETB SHIFT ; pulsos bajos a SHIFT/LOAD
8019 7F08 49 DENUEVO: MOV R7,#8 ;utiliza R7 como contador de bits
801B A295 50 CICLO: MOV C,DOUT ;obtener un bit (colocarlo en C)
801D 13 51 RRC A ;colocarlo en ACC.0 (bit menos significativo primero)
801E C296 52 CLR CLOCK ;enviar pulso a línea de CLOCK (desplaza
8020 D296 53 SETB CLOCK ; bits hacia DATA OUT)
8022 DFF7 54 DJNZ R7,CICLO ;si no es 8o desplazamiento, repetir
8024 F6 55 MOV @R0,A ;si 8o desplazamiento, colocarlo en búfer
8025 08 56 INC R0 ;incrementa apuntador a búfer
8026 DEF1 57 DJNZ R6,DENUEVO ;obtener dos bytes
8028 22 58 RET
59
60 ;*****
61 ; ENVIAR MENSAJE DE HOLA A LA CONSOLA (AYUDA PARA DEPURACION)
62 ;*****
63 ENVIA_MENSAJE_HOLA:
8029 908030 64 MOV DPTR,#MENSAJE ;apunta a mensaje de hola

```

FIGURA 11–16a

Software para la interfaz a un 74HC165

```

802C 120282      65          CALL       CADSAL           ;envíalo a la consola
802F 22           66          RET
8030 2A2A2A20    67  MENSAJE:   DB        '*** PRUEBA DE INTERFAZ A 74HC165 ***', CR, 0
8034 54455354
8038 20373448
803C 43313635
8040 20494E54
8044 45524641
8048 43452C2A
804C 2A2A
804E 0D
804F 00

68
69  ;*****
70  ; PRESENTA RESULTADOS EN CONSOLA (AYUDA PARA DEPURACION) *
71  ;*****
72  PRESENTA_RESULTADOS:          ;presenta bytes
8050 7825          73          MOV        R0,#BUFER        ;R0 apunta a bytes
8052 7E02          74          MOV        R6,#CONTEO        ;R6 es # de bytes leídos
8054 E6           75  CICLO2:    MOV        A,@R0           ;obtener byte
8055 08           76          INC        R0             ;incrementa apuntador
8056 12028D        77          CALL       SALIDA2HEX      ;salida como 2 caracteres en hex
8059 7420          78          MOV        A,#' '
805B 1201DE        79          CALL       CARSAL          ;separar bytes
805E DEF4          80          DJNZ      R6,CICLO2       ;repetir para cada byte
8060 740D          81          MOV        A,#CR           ;empieza una nueva línea
8062 1201DE        82          CALL       CARSAL          ;enviar CR (LF también)
8065 22           83          RET
84
85  ;*****
86  ; CREAR BUFER EN RAM INTERNA DIRECCIONABLE POR BIT *
87  ;*****
---- 88          DSEG      AT 25H           ;segmento de datos en chip en
0025 89  BUFER:     DS        CONTEO          ; en espacio direccionable por bit
90          END

```

FIGURA 11-16b*continuación*

Para seleccionar puertos específicos dentro del 8255 se utilizan dos terminales de control de entrada, A0 y A1. (Consulte la tabla 11-2). Sin embargo, observemos que si tanto A0 como A1 están activados, esto no selecciona a un puerto sino al registro de control. El registro de control es un registro de 8 bits (consulte la tabla 11-3) incorporado dentro del 8255 y que se utiliza para especificar el modo de operación de los tres puertos. Esto es parecido a los registros TMOD y SCON del 8051, los cuales son utilizados para establecer los modos de operación de los temporizadores y el puerto serial, respectivamente.

Los puertos del 8255 pueden dirigirse para operar en cualquiera de cuatro modos, pero aquí sólo utilizaremos el más simple, el modo 0 (E/S básicas). Este modo configura los puertos para proporcionar operaciones simples de entrada y salida similares a los puertos normales 0 y 3 del 8051. Otros modos más complejos permiten llevar a cabo el protocolo de intercambio, el cual básicamente permite que dos dispositivos (en este caso el 8051 y el dispositivo de E/S) se comuniquen entre ellos con mayor capacidad mediante una serie de señales de protocolo de intercambio.

TABLA 11-2

Selección de puertos del 8255

A1	A0	Selección
0	0	Puerto A
0	1	Puerto B
1	0	Puerto C
1	1	Registro de control

TABLA 11-3

Resumen del registro de control del 8255

Bit	Grupo	Descripción
D7	A	1 = modo de E/S 0 = modo de BSR
D6	A	Bit 1 de selección de modo
D5	A	Bit 0 de selección de modo 00 = modo 0 01 = modo 1 10 = modo 2 11 = modo 3
D4	A	Puerto A 1 = entrada 0 = salida
D3	A	Puerto C (PC7 a PC4 superior) 1 = entrada 0 = salida
D2	B	Bit de selección de modo 0 = modo 0 1 = modo 1
D1	B	Puerto B 1 = entrada 0 = salida
D0	B	Puerto C (PC3 a PC0 inferior) 1 = entrada 0 = salida

EJEMPLO**11.7****Objetivo del diseño**

Escriba un programa que lea el estatus de los ocho interruptores mostrados en la figura 11-17 y encienda el LED correspondiente a cada interruptor cerrado.

En la figura 11-17, observe que el 8255 está conectado al 8051 como si fuera un dispositivo de memoria externa. Al recurso de conectar dispositivos de E/S (en este caso el 8255) como memoria se le conoce como *asignación de memoria*. Esto habilita al 8051 para que dirija las direcciones y el registro de control del 8255 como ubicaciones en memoria externa. Los dos bits menos significativos del bus de direcciones, A1 y A0, se han conectado directamente a las entradas A1 y A0 del 8255; por lo tanto, éstas se utilizarán para seleccionar un puerto específico del 8255 o el registro de control. Por otra parte, la línea de dirección A8 se ha conectado a la entrada de selección de chip (CS) del 8255, así que una dirección de la forma:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
x	x	x	x	x	x	x	1	x	x	x	x	x	x	A1	A0

seleccionaría el 8255, donde los bits A1 y A0 seleccionan un puerto específico o el registro de control incorporado dentro del 8255. Asumiendo que todas las equis tienen el valor de 0, entonces las direcciones 0100H, 0101H, y 0102H seleccionarían los puertos A, B y C, respectivamente, mientras que la dirección 0103H seleccionaría el registro de control.

Este es un ejemplo de interfaz simple y la figura 11-18 muestra el software para llevarla a cabo. El primer paso es seleccionar el registro de control para poder establecer el modo de

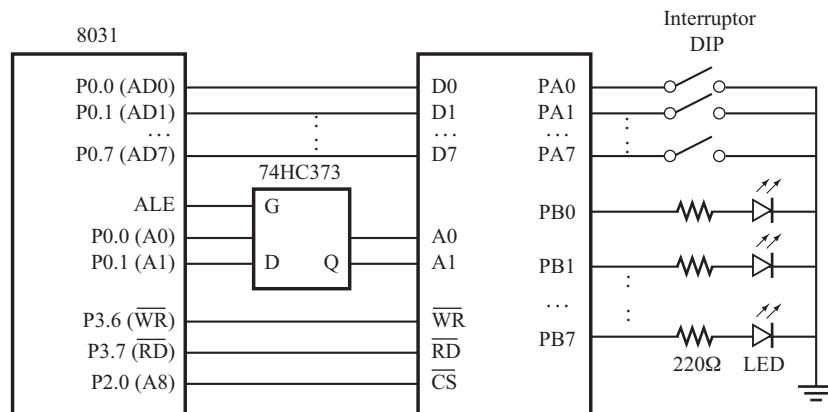


FIGURA 11-17
Interfaz a un 8255

```

1      $DEBUG
2      $NOSYMBOLS
3      $NOPAGING
4      ;ARCHIVO: 8255.SRC
5      ****
6      ;          EJEMPLO DE INTERFAZ PPI AL 8255      *
7      ;
8      ; Este programa utiliza la PPI al 8255 para conectarse      *
9      ; a 8 interruptores y a sus correspondientes 8 LEDs.      *
10     ; Mediante el 8255 proporciona 8 terminales de E/S      *
11     ; adicionales para uso de E/S.      *
12     ;*****
13
8000    14      ORG 8000H
8000 7490   15      MOV A, #1001000B ;Puerto A de entrada, puerto B de
                      ;salida
8002 900103  16      MOV DPTR, #0103H ;apunta al registro de control
8005 F0      17      MOVX @DPTR, A ;envía palabra de control
8006 900100  18 CICLO: MOV DPTR, #0100H ;apunta al puerto A
8009 E0      19      MOVX A, @DPTR ;lee interruptores
800A F4      20      CPL A ;complemento
800B 900101  21      MOV DPTR, #0101H ;apunta al puerto B
800E F0      22      MOVX @DPTR, A ;enciende LEDs
800F 80F5    23      SJMP CICLO ;ciclo
24          END

```

FIGURA 11-18
Software para la interfaz al 8255

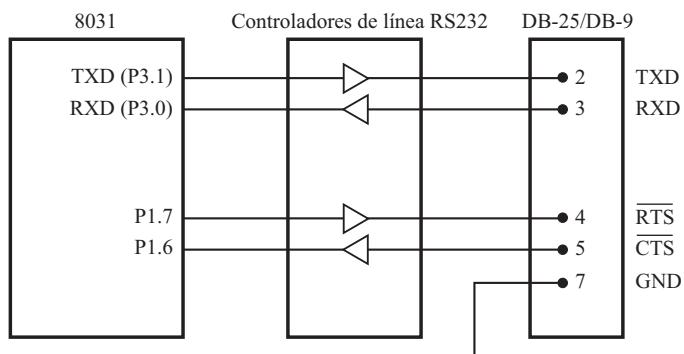
operación como el modo 0. Esto es seguido por el envío de la palabra de control adecuada para inicializar el puerto A como entrada y el puerto B como salida. Observe que los interruptores conectados al puerto A corresponden directamente a los LED conectados al puerto B. Por ejemplo, un interruptor cerrado en la terminal 0 del puerto A daría un nivel bajo, lo cual indicaría que debemos encender el LED correspondiente en la terminal 0 del puerto B enviando un nivel alto a esa terminal. Por lo tanto, seleccionamos primero el puerto A y sus contenidos se leen al acumulador, indicando el estatus actual de los interruptores. Entonces, todo lo que necesitamos hacer es complementar el contenido del acumulador, seleccionar el puerto B, y enviar el contenido del acumulador fuera del acumulador.

11.9 INTERFAZ SERIAL RS232 (EIA-232)

Hemos aprendido que el 8051 consta de un puerto serial incorporado para tener una interfaz a dispositivos de E/S seriales. De hecho, también podemos conectar el 8051 al puerto serial de una computadora personal (PC). El puerto serial de la PC sigue el estándar de la interfaz serial RS232 o EIA-232, y por esta razón se puede utilizar un cable RS232 normal para conectar una PC al 8051. El cable RS232 termina en ambos extremos con un conector (llamado DB-25) de 25 terminales. Sin embargo, debido a que no todas estas terminales se utilizan en la mayoría de las aplicaciones de una PC, también existe una versión diferente del conector llamada DB-9 que sólo tiene las nueve terminales más comúnmente utilizadas. Las tres terminales más importantes, ya sea para un DB-25 o un DB-9, son las de recepción de datos (RXD), transmisión de datos (TXD), y tierra (GND). La interfaz serial RS232 también permite habilitar el protocolo de intercambio, donde para establecer un canal de comunicación, un dispositivo inicia al enviar la señal de petición para enviar (RTS) a otro dispositivo y espera a que la señal correspondiente conceda permiso para enviar el regreso (CTS). Cuando se recibe la señal CTS, los dos dispositivos pueden enviar mensajes uno al otro.

Al conectar el 8051 a la interfaz serial RS232, una de las mayores preocupaciones es la diferencia que pueda haber entre los niveles de voltaje de estos dos dispositivos. El 8051 utiliza los niveles de voltaje de TTL donde 5V indican un nivel alto mientras que 0V indican un nivel bajo. Por otra parte, para la RS232, el nivel alto está definido entre +3V a +15V, mientras que un nivel bajo está entre -5V a -15V. Debido a esta diferencia, las conexiones entre el 8051 y la RS232 deben realizarse a través de controladores de línea. Los controladores de línea funcionan básicamente para efectuar la conversión entre dos diferentes niveles de voltaje, de tal manera que un nivel alto o bajo que el 8051 pueda descifrar también significa un nivel alto o bajo en la RS232 y viceversa. La figura 11-19 muestra cómo se conecta el 8051 a la interfaz serial RS232 mediante controladores de línea RS232 tales como el 1488/1489.

FIGURA 11-19
Interfaz a la RS232



EJEMPLO Objetivo del diseño**11.8**

El 8051 está conectado a una PC a través de la interfaz serial RS232. Escriba un programa que reciba como entrada números decimales de la PC conectada al puerto serial del 8051 y envíe el correspondiente código en ASCII a la pantalla.

La figura 11-20 muestra el software necesario para llevar esto a cabo. Primero se llama a la subrutina FAZ para inicializar el puerto serial y realizar el protocolo de intercambio con la interfaz serial antes de transmitir un mensaje de inicio. Con este propósito, el puerto serial se configura en el modo 1, UART de 8 bits, y la velocidad en baudios correspondiente a 9600 baudios. La señal RTS se verifica y el 8051 espera a recibir la señal CTS de aceptación de la interfaz serial. Un mensaje inicial sólo se enviará a través del puerto serial cuando se detecte la señal CTS.

```

1    $DEBUG
2    $NOSYMBOLS
3    $NOPAGING
4    ;ARCHIVO: SERIAL.SRC
5    ;*****
6    ;          EJEMPLO DE INTERFAZ SERIAL RS232      *
7    ;
8    ; Este programa obtiene un número decimal de la PC a través   *
9    ; de la interfaz serial RS232 y envía como salida el           *
10   ; correspondiente código en ASCII a la pantalla.            *
11   ;
12  ;*****
13
020E 14  CARENT    EQU     020EH
01DE 15  CARSAL    EQU     01DEH
0097 16  RTS       EQU     P1.7
0096 17  CTS       EQU     P1.6
18
8000 19  ORG 8000H
8000 12800F 20  CALL FAZ      ;inicialización y protocolo de
                           ;intercambio
8003 908043 21  MOV DPTR, #ASC ;apunta a tabla ASCII
8006 12020E 22  CICLO:      CALL CARENT    ;obtener número decimal
8009 93     23  MOVC A, @A+DPTR ;convertir a ASCII
800A 1201DE 24  CALL CARSAL    ;salida en ASCII
800D 80F7     25  SJMP CICLO    ;ciclo
26
27  ;*****
28  ; Subrutina para inicialización y protocolo de intercambio  *
29  ;*****
30
800F 758920 31  FAZ:        MOV TMOD, #20H ;temporizador 1, modo 2

```

FIGURA 11-20a

Software para la interfaz RS232

```

8012 758D98    32      MOV TH1, #98H ;conteo de recarga para 9600 baudios
8015 759852    33      MOV SCON, #52H ;puerto serial, modo 1
8018 90802C    34      MOV DPTR, #MSJ ;apuntador a mensaje inicial
801B D28E      35      SETB TR1  ;inicia temporizador 1
                         36
801D C297      37      CLR RTS   ;verifica RTS (protocolo de
                           ;intercambio)
801F 2096FD    38      JB CTS, $  ;espera por CTS
                         39
8022 93        40      SALIDA: MOVC A, @A+DPTR ;obtener caracteres
8023 1201DE    41      CALL CARSAL  ;enviar a salida
8026 6003      42      JZ ALTO   ;si es el final del mensaje, alto
8028 A3        43      INC DPTR   ;si no, obtener el siguiente carácter
8029 80F7      44      SJMP SALIDA ;ciclo
802B 22        46      ALTO:    RET
                         47
                         ****
49 ; Mensaje inicial y códigos en ASCII para los números *
50 ; ****
51
52 ;mensaje inicial
53 MSG:      DB 'POR FAVOR TECLEE UN NUMERO = ', 00H
802C 504C4541
8030 53452045
8034 4E544552
8038 204E554D
803C 42455220
8040 3D20
8042 00
                         54
55 ;Códigos en ASCII para números decimales
8043 30        56 ASC:    DB 30H ;0
8044 31        57 DB 31H ;1
8045 32        58 DB 32H ;2
8046 33        59 DB 33H ;3
8047 34        60 DB 34H ;4
8048 35        61 DB 35H ;5
8049 36        62 DB 36H ;6
804A 37        63 DB 37H ;7
804B 38        64 DB 38H ;8
804C 39        65 DB 39H ;9
                         66 END

```

FIGURA 11–20b*continuación*

El programa llama entonces a CARENT para esperar por números decimales de la interfaz serial. Una vez que se haya detectado un número a su llegada, el correspondiente código en ASCII se obtiene a partir de una tabla de búsqueda y más tarde se envía a través del puerto serial llamando a la subrutina CARSAL.

TABLA 11-4

Terminales de la interfaz paralela Centronics DB-25

Terminal	Descripción
1	STROBE
2	DATA 0
3	DATA 1
4	DATA 2
5	DATA 3
6	DATA 4
7	DATA 5
8	DATA 6
9	DATA 7
10	<u>ACK</u> (Aceptación)
11	BUSY
12	PAPER END
13	SLCT (Selección)
14	<u>AUTOFEED</u>
15	<u>ERROR</u>
16	<u>INITIALIZE PRINTER</u>
17	SLCTIN (Entrada de selección)
18–25	GND (Tierra)

11.10 INTERFAZ PARALELA CENTRONICS

La interfaz para impresoras más común es la interfaz paralela Centronics, la cual se ha convertido en el estándar de facto para la interfaz paralela, tal y como actúa su contraparte RS232 en la sección anterior. La interfaz paralela Centronics especifica 36 señales, y puede terminar en un conector Centronics de 36 terminales o incluso en un conector DB-25, como es más común en la mayoría de las computadoras personales en la actualidad. (Consulte la tabla 11-4). En esta sección estudiaremos cómo conectar un 8051 a una impresora a través de una interfaz paralela Centronics DB-25.

La interfaz paralela Centronics consta de ocho terminales de datos para cargar un byte de datos e imprimirla. Cuando un byte de datos se ha puesto disponible en estas terminales de datos, el 8051 envía una señal STROBE que permite al 8051 informar a la impresora que hay datos disponibles en sus terminales de datos. Si la impresora no está ocupada, enviará una señal ACK correspondiente para indicar que la petición de impresión se ha procesado con éxito. De otra manera, se enviará una señal BUSY en su lugar.

EJEMPLO**11.9****Objetivo del diseño**

Escriba un programa para interrogar a la impresora y después enviar un mensaje de prueba e imprimir en forma continua.

La figura 11-21 muestra la conexión de un 8051 a una impresora a través de la interfaz paralela Centronics. La figura 11-22 presenta el software correspondiente. El programa activa primero la señal STROBE a la impresora y después verifica el estatus de la impresora para asegurar que no está ocupada (BUSY = 0), que todavía hay papel (FIN DE PAPEL = 0), que ha sido seleccionada (SLCT = 1), y que no hay ningún error (ERROR = 1). Si alguna de las condiciones no se cumple, se genera un mensaje de error y el programa termina. De otra manera,

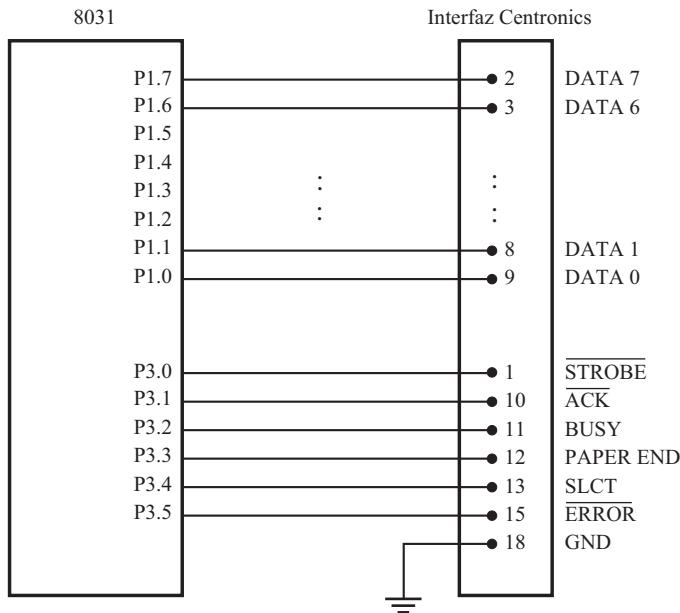


FIGURA 11-21
Interfaz a la interfaz paralela Centronics

```

1   $DEBUG
2   $NOSYMBOLS
3   $NOPAGING
4   ;ARCHIVO: PARALELA.SRC
5   ;*****EJEMPLO DE INTERFAZ PARALELA CENTRONICS*****
6   ;
7   ;
8   ; Este programa envía continuamente un mensaje de
9   ; prueba a la impresora.
10  ;*****
11
0090 12 DAT      EQU     P1
00B0 13 STR      EQU     P3.0
00B1 14 ACK      EQU     P3.1
00B2 15 BUSY    EQU     P3.2
003C 16 MASCARA  EQU     00111100B
0030 17 OK       EQU     00110000B
18
8000 19          ORG 8000H
8000 90801C 20 INICIO: MOV DPTR, #MSJ ;apunta al mensaje de prueba
8003 75B03C 21 CICLO:  MOV P3, #MASCARA ;activa STROBE y
                           ; P3.1 a P3.5 = entrada
22

```

FIGURA 11-22a
Software para la interfaz paralela Centronics

```

8006 E5B0      23     MOV A, P3          ;lee estatus de la impresora
8008 543C      24     ANL A, #MASCARA   ;sólo P3.1 a P3.5
800A B4300D    25     CJNE A, #OK, ERR  ;¿algún error?
800D E4        26     ENVIAR: CLR A    ;no: alistarse para enviar
800E 20B1FD    27     JB ACK, $       ;antes de enviar, espera por ACK
8011 93        28     MOVC A, @A+DPTR ;obtener carácter en mensaje de
                           prueba
8012 F590      29     MOV DAT, A      ;enviar carácter a la impresora
8014 A3        30     INC DPTR        ;apunta al siguiente carácter
8015 B400EB    31     CJNE A, #00H, CICLO ;¿fin del mensaje? Ciclo
8018 80E6      32     SJMP INICIO    ;fin de mensaje, regresa al inicio
                           33
801A 801F      34     ERR:     SJMP ALTO   ;error de impresora, alto
                           35
                           36 ;*****Mensaje de prueba para la impresora*****
                           37 ; Mensaje de prueba para la impresora *
                           38 ;*****Mensaje de prueba para la impresora*****
                           39
                           40 ;mensaje de prueba
801C 54484953  41     MSJ:     DB 'ESTA ES UNA PRUEBA PARA LA IMPRESORA', 00H
8020 20495320
8024 41205445
8028 53542046
802C 4F522054
8030 48452050
8034 52494E54
8038 4552
803A 00
803B 00        42     ALTO:    NOP
                           43     END

```

FIGURA 11–22b*continuación*

el 8051 espera por la señal de $\overline{\text{ACK}}$ de la impresora antes de proceder. Una vez que la señal de $\overline{\text{ACK}}$ se activa, el carácter actual se obtiene del mensaje de prueba y se envía a las terminales de datos de la impresora a través de P1.

11.11 SALIDA ANALÓGICA

Una interfaz al mundo real a menudo requiere generar o detectar condiciones analógicas. La generación y el control de una señal de salida analógica de un microcontrolador es algo fácil de obtener. Este ejemplo de diseño utiliza dos resistencias, dos capacitores, un potenciómetro, un amplificador operacional LM301, y un convertidor de digital a analógico (DAC) de 8 bits MC1408L8. Ambos circuitos integrados son baratos y se consiguen fácilmente. Las ocho entradas de datos al DAC están controladas desde el puerto 1 en el 8031 (consulte la figura 11–23). Después de construir el circuito y conectarlo a la SBC-51, deberá ser probado utilizando comandos de monitoreo. Mida el voltaje de salida en la terminal 6 del LM301 (V_o) cuando se escriben

diferentes valores al puerto 1 y se ajusta el potenciómetro de 1K. La salida debe variar desde 0 volts ($P1 = 00H$) hasta aproximadamente 10 volts ($P1 = FFH$).

Una vez que el circuito esté operando correctamente, estaremos listos para divertirnos con el software de interfaz. El programa de prueba típico es un generador de forma de onda diente de sierra que envía un valor al DAC, incrementa el valor, lo envía de nuevo, y así sucesivamente (vea la pregunta 3 al final de este capítulo). Sin embargo, nos adentraremos en un diseño mucho más ambicioso —un generador de onda sinusoidal controlado digitalmente.

EJEMPLO Objetivo del diseño

11.10

Escriba un programa para generar una onda sinusoidal utilizando la interfaz al DAC mostrada en la figura 11-23. Aplique una constante llamada PASO para establecer la frecuencia de la onda sinusoidal. Diseñe el programa para que esté controlado por interrupciones con una velocidad de actualización de 10 kHz.

Debido a que la capacidad de cálculos numéricos del 8031 es bastante limitada, el único enfoque razonable para este problema es el de utilizar una tabla de búsqueda. Necesitamos una tabla con valores de 8 bits que correspondan a un periodo de una onda sinusoidal. Los valores deben comenzar cerca de 127, incrementarse a 255, disminuir desde 127 hasta 0, y volver a subir hasta 127, siguiendo el patrón de una onda sinusoidal.

Una reproducción razonable de una onda sinusoidal requiere de una tabla relativamente grande; así surge la problemática de cómo generar la tabla. Los métodos manuales no son prácticos. El enfoque más fácil es escribir un programa en algún otro lenguaje de alto nivel para crear

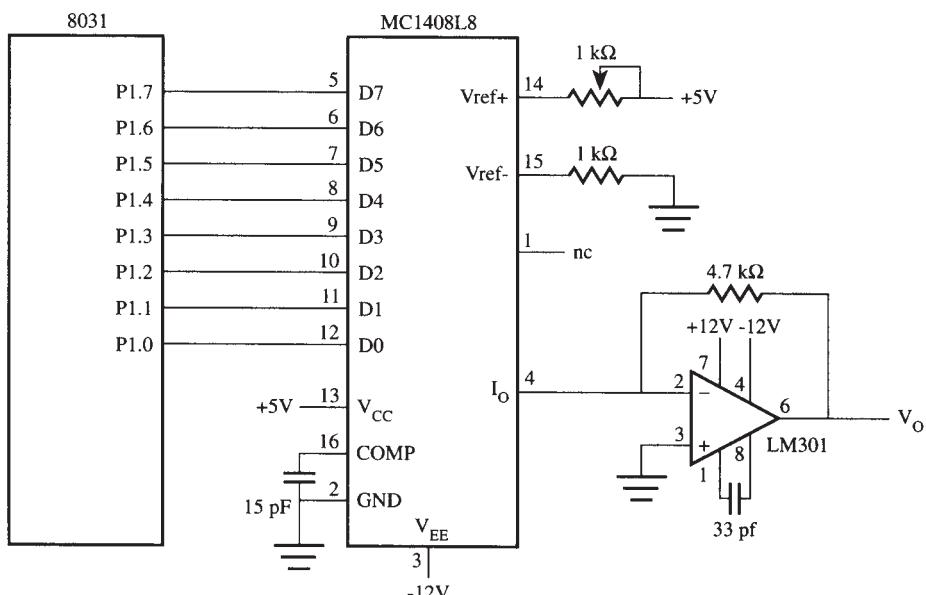


FIGURA 11-23
Interfaz a un MC1408L8

```

/*********************  

/* tabla51.c - programa para generar una tabla de onda sinusoidal */  

/*  

/* La tabla consiste de 1024 registros entre 0 y 255. Cada */  

/* registro está precedido por " DB " para incluirse en un */  

/* programa fuente del 8051. La tabla se escribe al archivo */  

/* seno51.src. */  

/*********************  

#include <stdio.h>  

#include <math.h>

#define PI    3.1415927
#define MAX   1024
#define BYTE  255

main()
{
    FILE *fp, *fopen();
    double x, y;

    fp = fopen("seno51.src", "w");
    for(x = 0; x < MAX; ++x) {
        y = ((sin((x / MAX) * (2 * PI)) + 1) / 2 * BYTE;
        fprintf(fp, " DB %3d\n", (int)y);
    }
}

```

FIGURA 11-24

Software para generar una tabla de onda sinusoidal

la tabla y almacenar los registros en un archivo. La tabla se importa a nuestro programa fuente del 8031, y así podremos trabajar. La figura 11-24 es un sencillo programa escrito en C y llamado tabla51.c, el cual realizará el trabajo por nosotros. El programa genera una tabla de onda sinusoidal de 1024 registros con los valores restringidos entre 0 y 255. La salida se escribe a un archivo de salida llamado seno51.src. Cada registro va precedido por “DB” para lograr compatibilidad con el código fuente del 8031.

La figura 11-25 muestra el programa de onda sinusoidal del 8031. El ciclo principal (líneas 36 a 40) lleva a cabo tres tareas: inicializa el temporizador 0 a que se interrumpa cada 100 μ s, habilita las interrupciones, y se mantiene en un ciclo infinito. La rutina de servicio de interrupción del temporizador 0 (líneas 41 a 51) realiza todo el trabajo. Cada 100 μ s se lee un valor de la tabla de búsqueda utilizando el DPTR y después se escribe al puerto 1. Se utiliza una constante llamada PASO como el incremento a través de la tabla. PASO está definido en la línea 26 como un byte en memoria RAM interna. Debe inicializarse utilizando un comando de monitoreo. Dentro de cada ISR, PASO se agrega a DPTR para obtener la dirección de la siguiente muestra. La tabla recibe un comando ORG en 8400H (línea 69) para que empiece en límites de 1K pares. Si el DPTR se incrementa más allá de 87FFH (el final de la tabla), se ajusta para reciclarse hasta llegar al inicio de la tabla. Debido a que la tabla es bastante grande, utilizamos una directiva de ensamblador \$NOLIST luego de los primeros cinco registros (línea 77) para apagar la salida al archivo de listado. En la línea 1092 (no mostrada) se utilizó una directiva \$LIST para encender la entrada una vez más durante los últimos cinco registros. La frecuencia

FIGURA 11-25a

Programa de onda sinusoidal del 8031

```

65      ; son de 8 bits cada uno (0 a 255) para salida a un DAC de 8 bits.* 
66      ; La tabla fue generada de un programa en C y copiada a este      *
67      ; programa del 8051.                                         *
68      ;*****                                                 *****

8400    69          ORG      8400H
8400 7F    70      TABLA:   DB       127
8401 80    71          DB       128
8402 81    72          DB       129
8403 81    73          DB       129
8404 82    74          DB       130
8404 82    75      ; Listado deshabilitado después de los primeros cinco registros
8404 82    76      ;-----
8404 82    77 +1    $NOLIST
8404 82    1093   ; Listado habilitado para los últimos cinco registros
87FB 7B    1094   DB       123
87FC 7C    1095   DB       124
87FD 7D    1096   DB       125
87FE 7D    1097   DB       125
87FF 7E    1098   DB       126
87FF 7E    1099   END

```

FIGURA 11–25b*continuación*

de la onda sinusoidal está controlada mediante tres parámetros: PASO, el tamaño de la tabla, y el periodo de interrupción del temporizador, como explicamos en las líneas 16 a 20 del listado.

11.12 ENTRADA ANALÓGICA

Nuestro siguiente ejemplo de diseño es un canal de entrada analógica. El circuito ilustrado en la figura 11-26 utiliza una resistencia, un capacitor, un potenciómetro de ajuste pequeño, y un convertidor de analógico a digital (ADC) ADC0804. El ADC0804 es un ADC económico (National Semiconductor Corp.) que convierte un voltaje de entrada a una palabra digital de 8 bits en cerca de $100\ \mu s$.

El ADC0804 está controlado por una entrada de escritura (\overline{WR}) y una salida de interrupción (\overline{INTR}). Un pulso a nivel bajo enviado a \overline{WR} inicia una conversión. Cuando la conversión se completa (100 ms después), el ADC0804 verifica la señal \overline{INTR} , convirtiéndola a nivel bajo. \overline{INTR} se lleva a nivel alto en la siguiente transición de 1 a 0 de \overline{WR} , lo cual inicia la siguiente conversión. \overline{INTR} y \overline{WR} se conectan a las líneas P1.1 y P1.0 del 8031, respectivamente. En este ejemplo, utilizamos el puerto A del 8155 para efectuar la transferencia de datos, como se muestra en la figura.

El ADC0804 opera mediante un reloj interno creado por la red RC que conecta las terminales 19 y 4. El voltaje de entrada analógico es una señal diferencial aplicada a las entradas $Vin(+)$ y $Vin(-)$ en las terminales 6 y 7. En este ejemplo, $Vin(-)$ es tierra y $Vin(+)$ está controlada desde la toma central del potenciómetro de ajuste pequeño. $Vin(+)$ variará entre el rango de 0 a +5 volts, controlado por el potenciómetro de ajuste pequeño. Consulte la hoja de datos para obtener una descripción detallada de la operación del ADC0804.

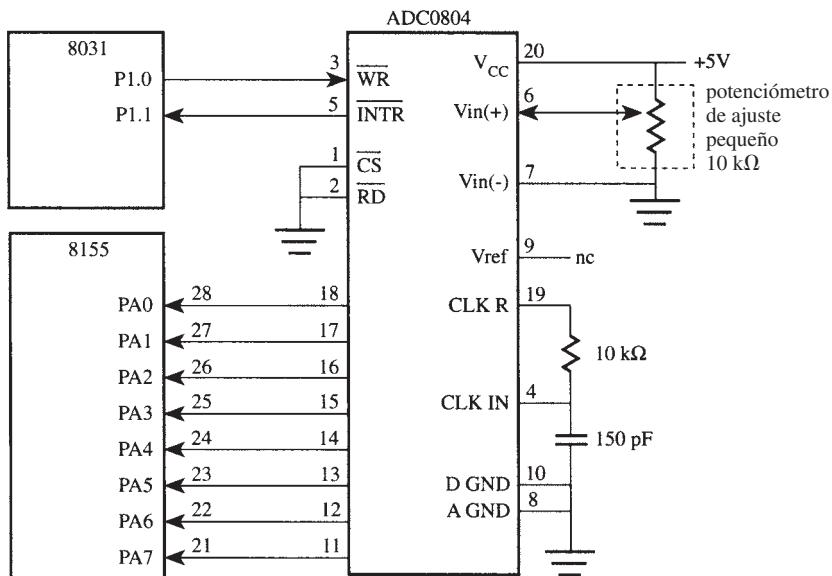


FIGURA 11–26
Interfaz al ADC ADC0804

EJEMPLO

11.11

Objetivo del diseño

Escriba un programa para detectar en forma continua el voltaje presente en la toma central del potenciómetro de ajuste pequeño (una vez convertido por el ADC0804). Presente el resultado en la consola como un byte en ASCII.

El programa que ilustra la figura 11-27 lleva a cabo el objetivo antes descrito. No requerimos de una secuencia de inicialización debido a que los puertos del 8155 son entradas predeterminadas luego de una reinicialización. El puerto A está en la dirección 0101H en memoria externa y es fácil de leer mediante una instrucción MOVX. Al limpiar y activar P1.0 (líneas 34 y 35) se inicia una conversión, la entrada WR del ADC0804. Después el programa se mantiene en un ciclo de espera para que el ADC0804 termine la conversión y la validación de INTR, en P1.1 (línea 36). Los datos se leen en las líneas 37 y 38 y se envían a la consola utilizando la subrutina SALIDA2HX del MON51 (línea 39). Mientras el programa está en ejecución se presenta un byte en la consola, el cual variará de 00H a FFH conforme el potenciómetro se ajuste.

El programa de la figura 11-27 es una primera aproximación del potencial de las entradas analógicas. Es posible reemplazar el potenciómetro con otras entradas analógicas. La medición de temperatura se lleva a cabo utilizando un termistor —dispositivo con una resistencia que varía con la temperatura. También es posible recibir voz como entrada utilizando un micrófono. El periodo de conversión del ADC0804 de 100 μ s se traduce a una frecuencia de muestreo de 10 kHz. Esto es suficiente para capturar señales de hasta 5 kHz de ancho de banda, lo cual equivale aproximadamente a una línea telefónica de grado vocal. Se requieren circuitos adicionales para aumentar las señales de bajo nivel proporcionadas por los micrófonos típicos al rango de 0 a 5 volts que el ADC0804 espera. Adicionalmente, requerimos de un circuito de muestreo y almacenamiento para mantener un voltaje constante a lo largo de cada conversión. Dejaremos que el lector explore estas posibilidades.

```

1   $debug
2   $nopaging
3   $nosymbols
4   ;ARCHIVO: ADC.SRC
5   ;*****EJEMPLO DE INTERFAZ DE ADC0804*****
6   ; Este programa lee datos de entrada analógicos de un ADC0804 con
7   ; interfaz al puerto A del 8155. El resultado se reporta en la
8   ; consola como un byte hexadecimal.
9   ; Se llevan a cabo los siguientes pasos:
10  ;
11  ;      1. Enviar mensaje a la consola
12  ;      2. Disparar línea -WR del ADC0804 (P1.0) para
13  ;          iniciar la conversión.
14  ;      3. Esperar a que la línea -INTR del ADC0804 (P1.1)
15  ;          llegue a nivel bajo indicando "conversión completa"
16  ;      4. Leer datos del puerto A del 8155
17  ;      5. Enviar datos como salida a la consola en hexadecimal.
18  ;      6. Ir al paso 2
19  ;
20  ;
21  ;*****PUERTO A: ADC0804*****
22 PUERTOA EQU    0101H      ;Puerto A del 8155
23 CR     EQU    0DH       ;ASCII para retorno de carro
24 LF     EQU    0AH       ;ASCII para salto de línea
25 ESC    EQU    1BH       ;ASCII para tecla de escape
26 SALIDA2HX EQU    028DH    ;Subrutina del MON51
27 CADSAL  EQU    0282H    ;Subrutina del MON51
28 WRITE   BIT    P1.0      ;línea -WR del ADC0804
29 INTR    BIT    P1.1      ;línea -INTR del ADC0804
30
31         ORG    8000H
32 ADC:    MOV    DPTR,#MENSAJE ;enviar mensaje
33         CALL   CADSAL
34 LOOP:   CLR    WRITE      ; disparar línea -WR
35         SETB   WRITE
36         JB    INTR,$      ;espera por -INTR = 0
37         MOV    DPTR,#PUERTOA ;inicializar DPTR --> puerto A
38         MOVX  A,@DPTR      ;leer datos del ADC0804
39         CALL   SALIDA2HX  ;enviar datos a la consola
40         MOV    DPTR,#IZQ2   ;regresar cursor 2 lugares a la izq.
41         CALL   CADSAL
42         SJMP  CICLO      ;repetir
43
44 MENSAJE: DB     '*** PRUEBA DEL ADC0804 ***',CR,0
45 IZQ2:   DB     ESC,'[2D',0      ;Secuencia de escape para la VT100
46
47         END

```

FIGURA 11-27

Software para la interfaz a un ADC0804

11.13 INTERFAZ A SENSORES

En la sección del capítulo 6 sobre interrupciones externas presentamos un ejemplo del controlador de un horno que hace uso de un sensor simple de temperatura. En esta sección examinaremos un sensor de temperatura más complejo, el termómetro y termostato digital DS1620, fabricado por Maxim.²

El DS1620 es un circuito integrado de 8 terminales con tres salidas para alarmas termales, T_{HIGH} , T_{LOW} , y T_{COM} . T_{HIGH} llega a nivel alto si la temperatura medida es mayor o igual que la temperatura alta definida por el usuario, TH . Por otra parte, T_{LOW} llega a nivel alto si la temperatura es menor o igual a la temperatura baja definida por el usuario, TL . Por el contrario, T_{COM} llega a nivel alto cuando la temperatura excede TH y sólo regresa un nivel bajo cuando la temperatura es menor que TL .

Para usar el DS1620 junto con el 8051, el DS1620 debe configurarse para comunicaciones con 3 conexiones. Este término resulta del hecho de que la comunicación entre los dos se lleva a cabo mediante tres conexiones: la terminal de entrada/salida de datos (DQ), la terminal de entrada de reloj (CLK/CONV), y la terminal de entrada de reinicialización (RST). La comunicación se inicia al activar la señal RST, y conforme se envían las señales de sincronización al DS1620 a través de la terminal RST, los datos se sincronizan a la entrada o la salida mediante la terminal DQ, enviando el bit menos significativo primero.

La figura 11-28 muestra las conexiones establecidas entre el 8051 y el DS1620. La interacción con el DS1620 requiere que el 8051 le envíe comandos (consulte la tabla 11-5).

FIGURA 11-28
Interfaz al DS1620

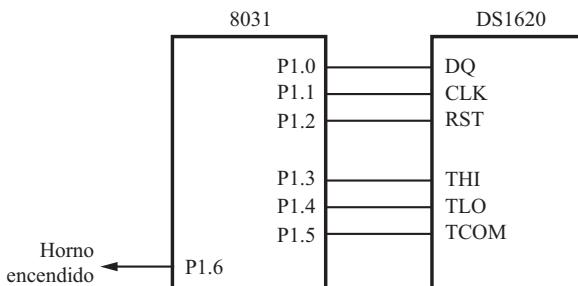


TABLA 11-5
Conjunto de comandos del DS1620

Código	Instrucción	Descripción
AAH	Lectura de temperatura	Lee la última temperatura convertida del registro de temperatura
A0H	Contador de lectura	Lee el valor del conteo restante del contador
A9H	Lectura de inclinación	Lee el valor del acumulador de inclinación
EEH	Inicio de convertidor	Inicia la conversión de temperatura
22H	Paro de conversión	Detiene la conversión de temperatura
01H	Escribe TH	Escribe el valor de TH al registro TH
02H	Escribe TL	Escribe el valor de TL al registro TL
A1H	Lee TH	Lee el valor de TH del registro TH
A2H	Lee TL	Lee el valor de TL del registro TL
0CH	Escribe configuración	Escribe los datos de configuración al registro de configuración
ACH	Lee configuración	Lee los datos de configuración del registro de configuración

²Maxim Integrated Products, Inc., 120 San Gabriel Drive, Sunnyvale, CA 94086.

EJEMPLO **Objetivo del diseño****11.12**

Escriba un programa que utilice un sensor de temperatura para mantener la temperatura ambiental a 20°C; encender el horno si la temperatura disminuye por debajo de 17°C, y apagar el horno cuando la temperatura excede los 23°C.

La figura 11-29 muestra el software necesario para llevar esto a cabo. El horno se apaga primero y el DS1620 se inicializa. Esto requiere enviar el comando ‘escribe configuración’ a la terminal DQ, seguido de un byte del código de configuración para el registro de configuración/estatus (consulte la tabla 11-6). En nuestro ejemplo, estableceremos el código de operación en forma tal que el bit de la CPU sea 1, lo cual selecciona al DS1620 para efectuar una operación de 3 conexiones con el 8051. Además, el DS16290 se selecciona para conversión continua de temperatura (detección) al limpiar el bit de “1shot” a 0. Advierta que tanto el envío del byte de comando como la configuración del código de operación se realizan mediante la subrutina ENVIAR, la cual los envía bit por bit a la terminal DQ con el bit menos significativo primero. Antes de enviar

```

1      $DEBUG
2      $NOSYMBOLS
3      $NOPAGING
4      ;ARCHIVO: SENSOR.SRC
5      ;*****                                 *
6      ;     EJEMPLO DE INTERFAZ A UN SENSOR DE TEMPERATURA   *
7      ;                                                 *
8      ; Este programa utiliza un sensor de temperatura para   *
9      ; monitorear la temperatura ambiental. Cuando la tem-   *
10     ; peratura excede los 23 grados, el horno se apaga.    *
11     ; Si la temperatura baja más allá de 17 grados, el   *
12     ; horno se enciende.                                     *
13     ;*****                                 *
14
0090    15      DQ      EQU      P1.0
0091    16      CLK     EQU      P1.1
0092    17      RST     EQU      P1.2
18
0093    19      THI     EQU      P1.3
0094    20      TLO     EQU      P1.4
0095    21      TCOM    EQU      P1.5
22
0096    23      HORNO   EQU      P1.6
24
8000    25      ORG 8000H
8000 C296  26      CLR HORNO          ;apagar horno
27
8002 D292  28      CONF: SETB RST      ;iniciar transferencia
8004 740C   29      MOV A, #0CH        ;escribir configuración
8006 128045 30      CALL ENVIAR       ;enviar al DS1620
8009 740A   31      MOV A, #00001010B   ;CPU = 1; 1-shot=0

```

FIGURA 11-29a

Software para la interfaz a un DS1620

```

800B 128045 32      CALL ENVIAR      ;enviar al DS1620
800E C292  33      CLR RST        ;detener transferencia
800F          34
8010 D292  35      SETB RST        ;iniciar transferencia
8012 7401  36      MOV A, #01H      ;escribir a TH
8014 128045 37     CALL ENVIAR      ;enviar al DS1620
8017 7430  38      MOV A, #48       ;TH = 48 x 0.5 = 24 grados C
8019 128045 39     CALL ENVIAR      ;enviar al DS1620
801C C292  40      CLR RST        ;detener transferencia
801D          41
801E D292  42      SETB RST        ;iniciar transferencia
8020 7402  43      MOV A, #02H      ;escribir a TL
8022 128045 44     CALL ENVIAR      ;enviar al DS1620
8025 7420  45      MOV A, #32       ;TH = 32 x 0.5 = 16 deg C
8027 128045 46     CALL ENVIAR      ;enviar al DS1620
802A C292  47      CLR RST        ;detener transferencia
802B          48
802C D292  49      CONV:         SETB RST        ;iniciar transferencia
802E 74EE  50      MOV A, #0EEH      ;iniciar detección de temperatura
8030 128045 51     CALL ENVIAR      ;enviar al DS1620
8033 C292  52      CLR RST        ;detener transferencia
8034          53
8035 209305 54      SENS:         JB THI, APAGAR   ;si T >= 24 grados, apagar
8038 209406 55      JB TLO, ENCENDER  ;si T <= 16 grados, encender
803B 80F8  56      SJMP SENS      ;ciclo
803C          57
803D C296  58      APAGAR:        CLR HORNO      ;apagar horno
803F 80F4  59      SJMP SENS      ;continuar detectando
8040          60
8041 D296  61      ENCENDER:     SETB HORNO      ;encender horno
8043 80F0  62      SJMP SENS      ;continuar detectando
8044          63      ;*****                                 ****
8045 7808  64      ENVIAR:        MOV R0, #08      ;utiliza R0 como contador
8047 C291  65      SIGUIENTE:    CLR CLK       ;inicia ciclo de
8048          66      RRC A         reloj/sincronización
8049 13   71      MOV DQ, C      ;rotar A a C, bit menos
804A 9290  72      SETB CLK      significativo primero
804B D291  73      DJNZ R0, SIGUIENTE ;enviar bit a DQ
804C          74      RET           ;completar el ciclo de
804D          75      END           reloj/sincronización
804E D8F7  76      ;procesar el siguiente bit
8050 22   77      ;continuar

```

FIGURA 11-29b
continuación

TABLA 11-6

Registro de configuración/estatus del DS1620

Bit	Nombre	Descripción
7	DONE	Bit de fin de conversión 1 = listo 0 = progresando
6	THF	Bandera de temperatura alta. 1 = temperatura \geq TH
5	TLF	Bandera de temperatura baja 1 = temperatura \leq TL
4	NVB	Bandera de memoria no volátil ocupada. 1 = escritura a memoria en proceso 0 = memoria desocupada
3	1	—
2	0	—
1	CPU	Bit de uso de la CPU. 0 = modo autónomo (no se requiere CPU) 1 = modo de comunicaciones de 3 conexiones
0	1SHOT	Modo de un solo disparo. 1 = una conversión de temperatura cuando se recibe el comando de ‘iniciar conversión’ 0 = conversión continua de temperatura

cualquier comando al DS1620, la transferencia debe iniciarse llevando la señal RST a nivel alto. La transferencia se detiene una vez que se ha enviado el comando al limpiar RST.

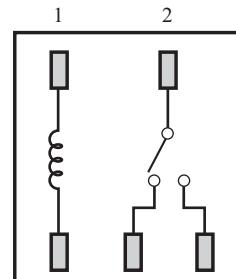
Después de un comando ‘escribir configuración’, se activan tanto la temperatura alta, TH, como la temperatura baja, TL, definidas por el usuario. A continuación, se envía un comando de ‘iniciar conversión’ para dirigir al DS1620 a que empiece a detectar la temperatura. Es entonces cuando el 8051 espera en un ciclo infinito a recibir el estatus de las salidas T_{HIGH} y T_{LOW} y enciende o apaga el horno en la forma apropiada.

11.14 INTERFAZ A RELEVADORES

Un relevador es un interruptor cuyos contactos se abren o cierran debido a un campo magnético producido cuando se le aplica electricidad. Los relevadores pueden considerarse, esencialmente,

FIGURA 11-30

Conexiones internas para el G6RN



como interruptores magnetomecánicos, y son muy útiles en aplicaciones donde se desea controlar eléctricamente la apertura y el cierre de interruptores.

Para el ejemplo presentado en esta sección utilizaremos el relevador unipolar de dos direcciones (SPDT) modelo G6RN de OMRON,³ cuyas conexiones internas se muestran en la figura 11-30.

EJEMPLO

11.13 Objetivo del diseño

Considere el sistema simple de semáforos peatonales ilustrado en la figura 11-31. Diseñe tal sistema de semáforos utilizando un 8051 y un relevador para controlarlo.

La figura 11-32 muestra el diagrama esquemático con las conexiones entre el 8051, el relevador, los botones peatonales, y los LED de los semáforos. La figura 11-33 muestra el listado

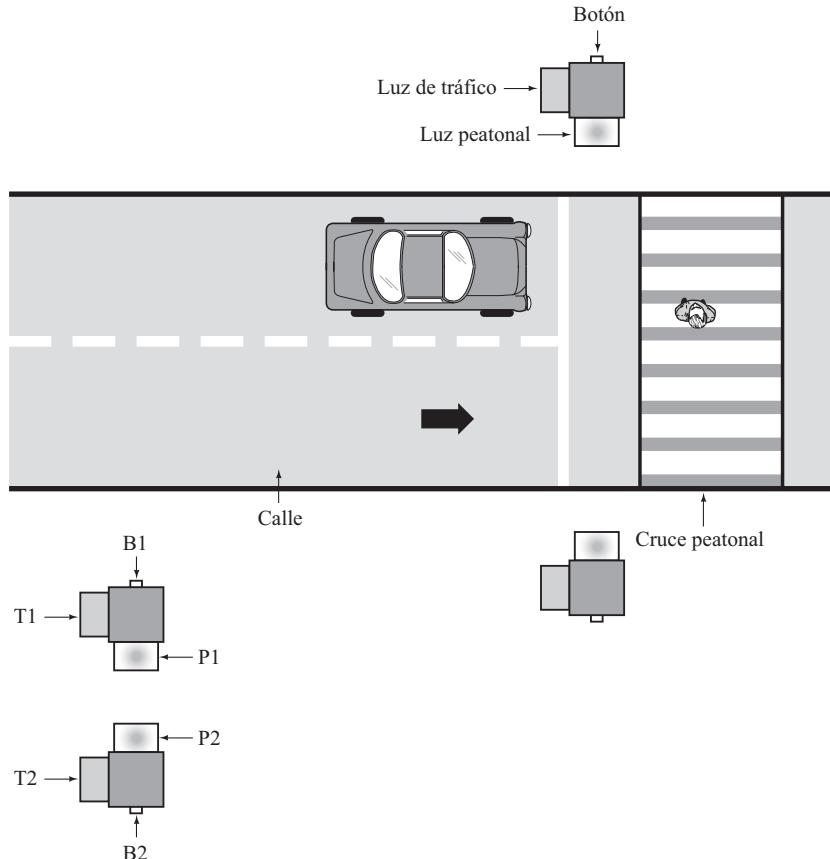
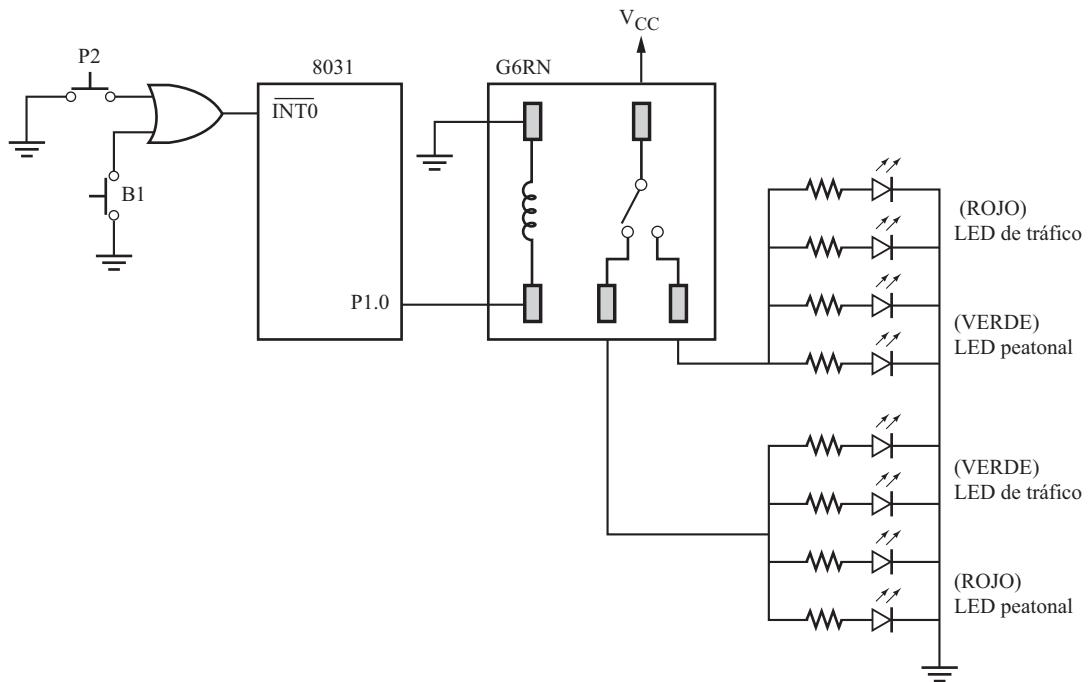


FIGURA 11-31

Sistema de semáforos peatonales⁴

³OMRON Corporation, 28º piso, Cristal Tower Bldg., 1-2-27, Shiromi, Chuo-ku, Osaka 540 Japón.

⁴Gracias a Joseph, Ken y Tonny por llevar a cabo este proyecto y por generar los gráficos de la figura 11-31.

**FIGURA 11–32**

Interfaz a un G6RN: sistema de semáforos peatonales

del software correspondiente. El programa principal habilita las interrupciones relevantes y espera a que un peatón oprima un botón (B1 o B2, dependiendo del lado de la acera en que se encuentre). Al principio, el puerto 1.0 está a nivel BAJO, y el polo instalado dentro del relevador causa que la luz de tráfico VERDE y la luz peatonal ROJA estén encendidas. Al oprimir un botón se genera una interrupción INT0 al 8051, el cual llama a la rutina de servicio de interrupción correspondiente EX0RSI, que es la mayor parte del programa. El puerto 1.0 se activa dentro de

```

1      $DEBUG
2      $NOSYMBOLS
3      $NOPAGING
4      ;ARCHIVO: SEMAFOROS.SRC
5      ;*****SEMAFOROS*****SEMAFOROS*****
6      ; EJEMPLO DE INTERFAZ A UN RELEVADOR:          *
7      ;                                              SEMAFORO PEATONAL   *
8      ;
9      ; Este programa interactúa con un relevador para  *
10     ; controlar algunos LEDs ROJOS y VERDES en un      *
11     ; sistema de semáforos peatonales.                 *
12     ;

```

FIGURA 11–33a

Software para la interfaz a un G6RN

```

13      ; Al principio, la luz de tráfico VERDE y la luz pea- *
14      ; tonal ROJA están encendidas. Si se oprime un botón *
15      ; de cruce peatonal, se genera una transición de ni-*
16      ; vel alto a nivel bajo en INT0. Esto envía una señal *
17      ; para encender la luz de tráfico ROJA y la luz pea-*
18      ; tonal VERDE, lo cual es seguido por un retraso de *
19      ; 10 segundos antes de que la luz de tráfico VERDE y *
20      ; la luz peatonal ROJA se enciendan de nuevo.          *
21      ;*****                                                 *
22
000A    23      DIEZ      EQU 10
0064    24      CIEN      EQU 100 ;10 x 100 x 10000 us = 10 segundos
D8F0    25      CONTEO    EQU -10000
26
0000    27      ORG 0
0000 02000B 28      LJMP PRINCIPAL
29                  ;vector EXT 0 a 0003H
0003 D290    30      EX0RSI:   SETB P1.0  ;luz de tráfico = ROJA, y
31                  ; luz peatonal = VERDE
0005 120017  32      CALL RETRASO ;esperar 10 segundos
0008 C290    33      CLR P1.0   ;luz de tráfico = VERDE,
34                  ; luz peatonal = ROJA
000A 32      35      RETI
36
000B 75A881  37      PRINCIPAL: MOV IE, #81H ;habilita interrupción
38                  ;externa 0
000E D288    39      SETB IT0    ;disparada por borde
39                  ;negativo
0010 758901  40      MOV TMOD, #01 ;temporizador 0 en modo 1
0013 C290    41      CLR P1.0   ;luz de tráfico = VERDE,
41                  ; luz peatonal = ROJA
0015 80FE    42      SJMP $     ;esperar para siempre
43
44      ;*****                                                 *
45      ; Subrutina para retraso de 10 segundos           *
46      ;*****                                                 *
47
0017 7E0A    48      RETRASO:  MOV R6, #DIEZ
0019 7F64    49      DENUEVO1: MOV R7, #CIEN
001B 758CD8  50      DENUEVO2: MOV TH0, #HIGH CONTEO
001E 758AF0  51      MOV TL0, #LOW CONTEO
0021 D28C    52      SETB TR0
0023 308DFD  53      ESPERA:  JNB TF0, $
0026 C28D    54      CLR TF0
0028 C28C    55      CLR TR0
002A DFEF    56      DJNZ R7, DENUEVO2
002C DEEB    57      DJNZ R6, DENUEVO1
002E 22      58      RET
59
60      END

```

FIGURA 11–33b
continuación

esta ISR y causa que el polo del relevador cambie a su otro contacto, lo cual enciende la luz de tráfico ROJA y la luz peatonal VERDE. A continuación ocurre un retraso de 10 segundos para permitir que los peatones crucen la calle antes de que todo regrese a la normalidad, y la luz de tráfico VERDE y la luz peatonal ROJA se enciendan de nuevo.

11.15 INTERFAZ A MOTOR PASO A PASO

Diversos dispositivos, tales como una impresora de matriz de puntos y una unidad de disco flexible, utilizan un tipo especial de motor llamado motor paso a paso. A diferencia de los motores convencionales de corriente directa, un motor paso a paso se mueve en incrementos o pasos precisos. Esto permite su utilización en aplicaciones que requieran del posicionamiento exacto de partes mecánicas.

Ripasemos brevemente la operación básica de un motor paso a paso. Un motor paso a paso tiene una parte móvil llamada **rotor**, el cual se fabrica por lo general a partir de un material magnético permanente. El rotor está rodeado por el **estator**, un electroimán elaborado con dos rebobinados alrededor de algunos polos conductores. La figura 11-34 muestra estos elementos con mayor detalle.

Observe que los dos rebobinados tienen una toma central, COM, la cual está conectada generalmente a +5V. Consideremos ahora uno de estos rebobinados, el mostrado con una orientación vertical en la figura 11-34(a), con sus dos puntas denotadas como A y B. Este rebobinado vertical estaría colocado alrededor de los polos estatores de hacia arriba y hacia abajo como se muestra en la figura 11-34(b). Las dos puntas A y B se energizarían alternadamente y causarían el flujo de corriente a través de un estator, lo cual convertiría al polo estator en un electroimán para atraer el imán del rotor colocado en el centro y moverlo en la dirección deseada. La figura 11-34(b) también muestra que el estator A ha sido energizado para atraer el imán del rotor hacia él en la dirección indicada. De la misma forma, las dos puntas del rebobinado horizontal, C y D, estarían bobinadas alrededor de los polos estatores hacia la izquierda y hacia la derecha, y éstos también se energizarían alternadamente.

La figura 11-35 muestra una ilustración adicional de la interacción entre el rotor y el estator. Supongamos que se energiza primero al estator A, y por lo tanto se le convierte en un electroimán para atraer el rotor como se muestra en la figura 11-35(a). A continuación energizamos

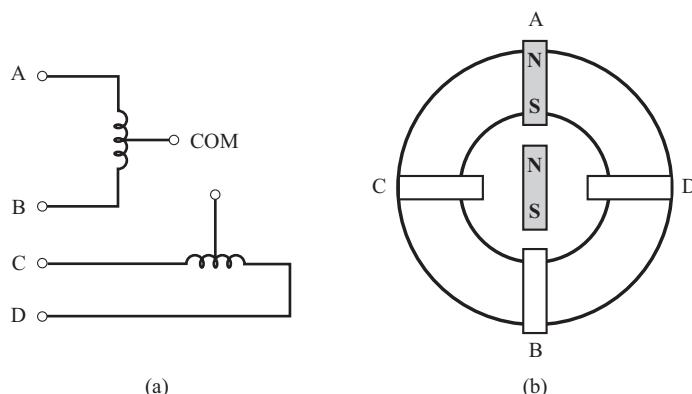
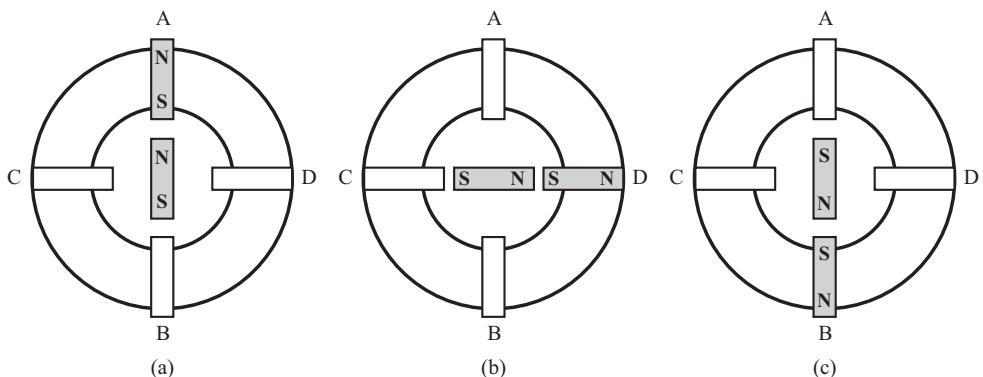


FIGURA 11-34

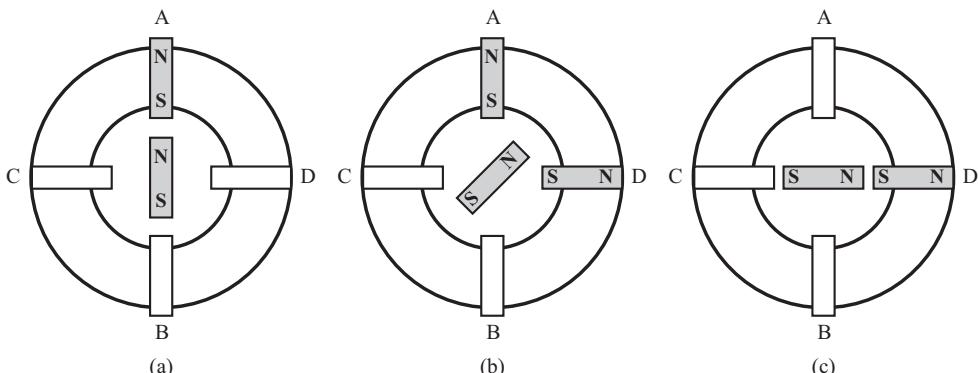
Configuración de los rebobinados de los estatores en un motor paso a paso

**FIGURA 11-35**

Rotación del rotor de un motor paso a paso

al estator D, que atrae entonces al rotor como indica la figura 11-35(b). Enseguida energizamos al estator B y de nuevo atrae al rotor como en la figura 11-35(c). Finalmente, energizamos al estator C y atrae en forma similar al rotor. Al energizar los estatores en la secuencia A, D, B, C causamos que el rotor gire en el sentido de las manecillas del reloj para completar una revolución, también se dice que el rotor ha completado una rotación de 360° en el sentido de las manecillas del reloj. Por el contrario, si energizáramos los estatores en la secuencia opuesta, el rotor efectuaría una rotación en sentido contrario al del reloj.

Observemos que, al operar el rotor de esta manera, se mueve a cuatro posiciones únicas conforme realiza una rotación completa paso a paso, donde cada paso involucra una rotación de 90°. Un motor de este tipo es conocido como un **motor paso a paso de 4 pasos**. Podemos utilizar **motores paso a paso de 8 pasos** para obtener resoluciones más finas, en donde un paso realiza una rotación de 45°. Esto se lleva a cabo, por ejemplo, energizando los polos estatores no uno a la vez sino alternadamente dos estatores adjuntos al mismo tiempo. Por ejemplo, para rotar el motor paso a paso en el sentido de las manecillas del reloj, podemos energizar los polos estatores de acuerdo con la secuencia A, AD, D, DB, B, BC, C, CA. La figura 11-36 ilustra los primeros tres pasos.

**FIGURA 11-36**

Rotación de medio paso de un motor paso a paso

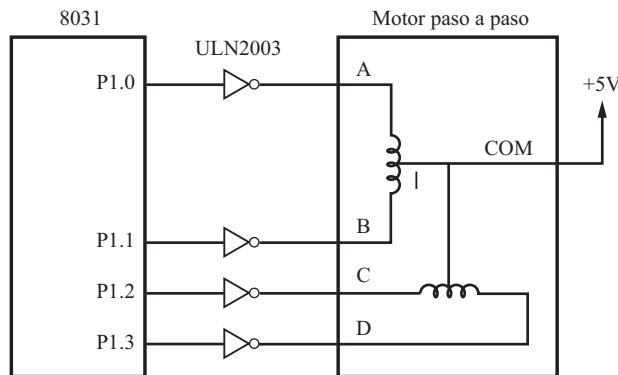


FIGURA 11-37
Interfaz al motor paso a paso

Debemos llevar a cabo la conexión del 8051 a los rebobinados del motor paso a paso a través de un controlador de motor paso a paso, ya que la corriente de las terminales de puerto del 8051 no son suficientes para controlar los rebobinados. La figura 11-37 muestra cómo el controlador de motor paso a paso ULN2003 realiza una interfaz del 8051 al motor paso a paso.

Es evidente que para energizar un polo estator con las conexiones establecidas en la figura 11-37 necesitamos escribir un ‘1’ alto a la terminal de puerto correspondiente conectada al polo, ello permite el flujo de corriente a través de los rebobinados del estator para convertirlo en un electroimán. Las tablas 11-7 y 11-8 muestran, respectivamente, la secuencia de pasos necesarios para habilitar motores paso a paso tanto de 4 como de 8 pasos.

TABLA 11-7

Secuencia de 4 pasos para lograr una rotación completa en el sentido de las manecillas del reloj

Paso	Rebobinado A	Rebobinado B	Rebobinado C	Rebobinado D
1	1	0	0	0
2	0	0	0	1
3	0	1	0	0
4	0	0	1	0

TABLA 11-8

Secuencia de 8 medios pasos para lograr una rotación en el sentido de las manecillas del reloj

Paso	Rebobinado A	Rebobinado B	Rebobinado C	Rebobinado D
1	1	0	0	0
2	1	0	0	1
3	0	0	0	1
4	0	1	0	1
5	0	1	0	0
6	0	1	1	0
7	0	0	1	0
8	1	0	1	0

EJEMPLO **Objetivo del diseño****11.14**

Escriba un programa para rotar inicialmente un motor paso a paso en el sentido de las manecillas del reloj. Si un interruptor conectado a la señal INT0 efectúa una transición de nivel alto a nivel bajo, el programa debe cambiar la dirección de la rotación, la cual en este caso iría en sentido contrario al de las manecillas del reloj.

La figura 11-38 muestra el software necesario para llevar esto a cabo. Debido a que utilizamos la interrupción externa 0 (INT0) para detectar la transición de nivel alto a nivel bajo, incluimos una ISR relevante para la señal INT0 como parte del programa, en tal forma que el

```

1  $DEBUG
2  $NOSYMBOLS
3  $NOPAGING
4  ;ARCHIVO: PASOAPASO.SRC
5  ;*****EJEMPLO DE INTERFAZ A UN MOTOR PASO A PASO*****
6  ;
7  ;
8  ; Este programa utiliza la secuencia de 8 pasos para girar ini-
9  ; cialmente al motor paso a paso en el sentido de las manecillas *
10 ; del reloj. Si un interruptor conectado a INT0 efectúa una tran-
11 ; sición de nivel alto a nivel bajo, cambiamos la rotación en   *
12 ; sentido contrario. *
13 ;*****EJEMPLO DE INTERFAZ A UN MOTOR PASO A PASO*****
14
0009 15 PASO1      EQU    1001B ;secuencia de 8 pasos para
0008 16 PASO2      EQU    1000B ; una rotación en el sentido de las
                                ; manecillas del reloj
000C 17 PASO3      EQU    1100B
0004 18 PASO4      EQU    0100B
0006 19 PASO5      EQU    0110B
0002 20 PASO6      EQU    0010B
0003 21 PASO7      EQU    0011B
0001 22 PASO8      EQU    0001B
0000 23 D          EQU    0      ;bit de dirección
                                ; en dirección del bit 0
0064 25 CIEN       EQU    100  ;100 x 10000 us = 1 segundo
D8F0 26 CONTEO    EQU    -10000
27
0000 28           ORG 0
0000 020006 29 LJMP PRINCIPAL
                                ;vector EXT 0 a 0003H
0003 B200 31 EXORSI:  CPL D      ;complementar el bit de dirección
0005 32           RETI
33
0006 75A881 34 PRINCIPAL: MOV IE, #81H ;habilita INT0
0009 D288 35     SETB IT0   ;disparada por borde negativo
000B 758901 36    MOV TMOD, #01 ;temporizador 0 en modo 1
000E C200 37     CLR D      ;inicializar D = 0
0010 120020 38    CALL SR    ;inicialmente: rotación en el sentido de
                                ; las manecillas del reloj
0013 200005 39 REPETIR:  JB D, IR_CSR ;¿en sentido contrario al de las
                                ; manecillas del reloj?
0016 120020 40 IR_SR :   CALL SR    ;no: en el sentido de las manecillas
                                ; del reloj

```

FIGURA 11-38a

Software para la interfaz a un motor paso a paso

```

0019 80F8   41      SJMP REPETIR
001B 120035  42      IR_CSR:    CALL CSR      ;sí: en sentido contrario al de las
                                ;manecillas del reloj
001E 80F3   43      SJMP REPETIR ;repetir para siempre
                                44
                                45 ;*****
                                46 ; Subrutina para rotación en el sentido de las manecillas *
                                ; del reloj (SR)
                                47 ;*****
                                48
0020 90005F  49      SR:       MOV DPTR, #SEQ ;apuntar a la tabla
0023 E4      50      CLR A      ;apuntar al primer paso
0024 C0E0   51      SIGUIENTE: PUSH ACC ;respaldar índice en A
0026 93      52      MOVC A,
0027 540F   53      ANL A, #0FH ;enmascarar los 4 bits superiores
0029 F590   54      MOV P1, A ;enviar al motor paso a paso
002B 12004B  55      CALL RETRASO ;esperar 1 segundo
002E D0E0   56      POP ACC   ;recuperar índice a A
0030 04      57      INC A      ;apuntar al siguiente paso
0031 B408F0  58      CJNE A, #8, SIGUIENTE ;asegurar 8 pasos
0034 22      59      RET
0046 14      73      DEC A      ;apuntar al siguiente paso
0047 B4FFF0  74      CJNE A, #0FFH, SIGUIENTE2 ;asegurar 8 pasos
004A 22      75      RET
                                76
                                77 ;*****
                                78 ; Subrutina para retraso de 1 segundo *
                                79 ;*****
                                80
004B 7F64   81      RETRASO:  MOV R7, #CIEN
004D 758CD8  82      DENUEVO:  MOV TH0, #HIGH CONTEO
0050 758AF0  83      MOV TL0, #LOW CONTEO
0053 D28C   84      SETB TR0
0055 308DFD  85      ESPERA:  JNB TF0, $
0058 C28D   86      CLR TF0
005A C28C   87      CLR TR0
005C DFEF   88      DJNZ R7, DENUEVO
005E 22      89      RET
                                90
                                91 ;*****
                                92 ; Patrón de secuencia de 8 pasos para lograr una rotación *
                                93 ; en el sentido de las manecillas del reloj. *
                                94 ;*****
                                95
005F 09      96      SEC:     DB PASO1   ;tabla de búsqueda con
0060 08      97      DB PASO2   ; el patrón de secuencia
0061 0C      98      DB PASO3   ; para rotación en el sentido de las
                                ;manecillas del reloj
0062 04      99      DB PASO4
0063 06     100     DB PASO5
0064 02     101     DB PASO6
0065 03     102     DB PASO7
0066 01     103     DB PASO8
                                104
                                105     END

```

FIGURA 11–38b
continuación

bit de dirección, D, se complemente a cada instancia de una transición de nivel alto a nivel bajo en INT0. Existen también dos funciones, SR y CSR, para hacer que el motor pase a paso rote en contra o en el mismo sentido que las manecillas del reloj, respectivamente, además de una subrutina que genera un retraso de 1 segundo.

Al principio, el programa principal dirige al motor paso a paso para que gire en el sentido de las manecillas del reloj. Despues de completar un ciclo de rotación, verificamos la dirección actual al referirnos a D, y llamaremos ya sea a SR o a CSR para continuar la rotación del motor paso a paso en la dirección indicada por D.

RESUMEN

Con lo estudiado en este capítulo concluye nuestro examen de ejemplos de interfaces. Los diseños aquí presentados ilustran muchos de los conceptos requeridos para implementar interfaces sofisticadas utilizando un microcontrolador 8051.

No obstante, no hay ningún sustituto para la experiencia práctica. Podemos entender mejor los ejemplos de este capítulo, y los que presentamos en capítulos anteriores, a través del proceso de ensayo y error. La mejor manera de desarrollar los conceptos de este libro es implementando los ejemplos en un sistema real. El texto proporciona la base para que los estudiantes puedan explorar más a fondo las posibilidades de la utilización de un microcontrolador tal como el 8051 en diseños con el mínimo número de componentes.

PROBLEMAS

- 11.1** Utilice la interfaz a un altavoz ilustrada en la figura 11-9 para reproducir repetidas veces la melodía que se muestra en la figura 11-39.
- 11.2** Reconfigure la interfaz al 74HC165 ilustrada en la figura 11-15 y modifique el software de la figura 11-16 para utilizar el puerto serial del 8051 (en modo 0) para las líneas de reloj/sincronización y de datos.
- 11.3** Si utilizamos el siguiente programa con el circuito de salida de digital a analógica mostrado en la figura 11-23, lo que resulta es una onda diente de sierra. (Asuma una operación a 12 MHz.)

PASO	EQU	1
PRINCIPAL	MOV	P1,A
ADD		A, #PASO
SJMP		PRINCIPAL



FIGURA 11-39

Melodía para el problema 11.1

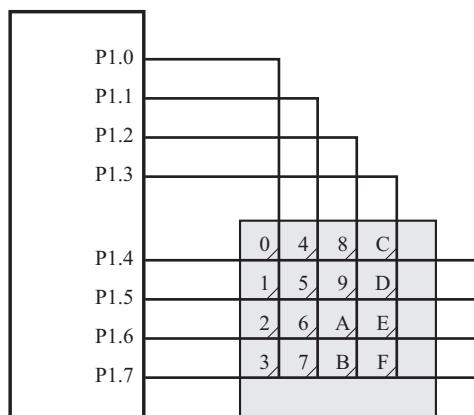
- a. ¿Cuál es la frecuencia de la onda diente de sierra?
 - b. ¿Cuál es el valor para PASO que alcanzaría una frecuencia de salida de 10 kHz (aproximadamente)?
 - c. Derive la ecuación para la frecuencia, dado PASO.
- 11.4** Varios de los programas ilustrados en este capítulo utilizaron subrutinas del MON51. Si un programa va a utilizar la subrutina ISDIG (para verificar si un byte es un dígito en ASCII), ¿cuál es la dirección de la MON51 a la que el símbolo ESDIG debe igualarse?
- 11.5** Escriba un programa para crear una onda cuadrada de 1 kHz en P1.7 utilizando el temporizador 8155 y las interrupciones externas 0. (Sugerencia: consulte la hoja de datos de un 8155.)
- 11.6** Suponga que ampliamos la interfaz al 74HC165 mostrada en la figura 11-15 a 6 circuitos integrados, proporcionando 48 líneas de entrada adicionales.
- a. ¿Cómo debemos modificar el programa de la figura 11-16 para leer las 48 líneas de entrada?
 - b. ¿Dónde serán almacenados los datos en la memoria RAM interna del 8031?
 - c. ¿Cuál es la nueva duración de la subrutina OBTEN_BYTES ilustrada en la figura 11-16?
 - d. Si colocamos OBTEN_BYTES en una rutina de servicio de interrupciones que se ejecuta cada segundo, ¿cuál es el porcentaje del tiempo de ejecución de la CPU que se llevaría la lectura de las 48 entradas generadas por los 74HC165 adicionales?
- 11.7** En la figura 11-25, la constante PASO fue definida como un byte de datos internos en la ubicación 50H utilizando la siguiente directiva del ensamblador:
- ```
PASO DATA 50H
```
- Esta es la manera correcta de definir PASO; sin embargo, lo siguiente funcionaría también:
- ```
PASO    EQU      50H
```
- En el segundo caso, el ASM51 no realiza ninguna verificación de tipos cuando el programa se ensambla. Proporcione el ejemplo de un uso incorrecto de la etiqueta PASO que *no generaría* un error al tiempo de ensamblado si PASO se definiera con EQU, pero que *sí generaría* un error si PASO se definiera adecuadamente utilizando DATA.
- 11.8** Suponga que un teclado numérico en hexadecimal de 16 teclas está conectado a un 8051 como se muestra en la figura 11-40. Las columnas están enumeradas de izquierda a derecha, empezando por el 0, mientras que los renglones lo están de arriba hacia abajo, también empiezan en 0. Suponga que al detectar una tecla oprimida llamamos a la subrutina TECLADO, y que R6 contiene la información de la columna y los 4 bits superiores de P1 contienen la información del renglón, donde

R6 = 3	Columna 0
R6 = 2	Columna 1
R6 = 3	Columna 2
R6 = 0	Columna 3
P1.4 = 0	Renglón 0
P1.5 = 0	Renglón 1
P1.6 = 0	Renglón 2
P1.7 = 0	Renglón 3

Escriba la subrutina TECLA que utiliza los valores en R6 y P1 para enviar como salida el valor de la tecla oprimida. Muestre sus pasos.

FIGURA 11-40

Teclado numérico en hexadecimal



- 11.9** Modifique el software de la LCD mostrada en la figura 11-8 para presentar continuamente el mensaje: “Bienvenido a la experiencia del microcontrolador 8051. Esperamos que disfrute su aventura de lectura”.
- 11.10** Hemos visto cómo utilizar el 8255 en el modo 0. Investigue sobre los modos más complejos del 8255 y entonces escriba las instrucciones en lenguaje ensamblador para inicializar los puertos del 8255 en cada uno de esos modos.
- 11.11** Efectúe una comparación entre la interfaz serial y la interfaz paralela. ¿Cuál opina usted que es la diferencia más significativa? Y con esto, ¿por qué cree que la interfaz serial es más comúnmente utilizada en teclados, ratones y módems mientras que la interfaz paralela se utiliza en las impresoras?
- 11.12** Escriba el seudocódigo correspondiente para el software de semáforo peatonal mostrado en la figura 11-33.
- 11.13** Investigue los tipos y las aplicaciones de los motores paso a paso. Seleccione tres de esas aplicaciones, y proporcione una explicación específica sobre la razón para utilizar un motor paso a paso en cada aplicación.

Ejemplos de diseño e interfaces en C

12.1 INTRODUCCIÓN

En el capítulo anterior presentamos varios ejemplos de diseño e interfaz en los que se involucra al 8051, con los correspondientes programas de interfaz escritos en lenguaje ensamblador. En este capítulo trataremos esos problemas de interfaz mediante la escritura de programas similares de demostración de conceptos en lenguaje C del 8051. Ello nos proporcionará un panorama más claro en cuanto a la diferencia, y las similitudes, que hay entre los dos lenguajes que utilizamos para interactuar con el 8051.

12.2 INTERFAZ DE TECLADO NUMÉRICO HEXADECIMAL

En el capítulo anterior hablamos sobre la interfaz de teclado numérico hexadecimal, donde el objetivo de diseño era escribir un programa para leer caracteres en forma continua del teclado de 4×4 , y enviar el correspondiente código ASCII a la consola. En el ejemplo 12.1 se muestra un programa similar en C para el 8051.

EJEMPLO Interfaz de teclado numérico hexadecimal

12.1 Vuelva a escribir en lenguaje C el software para la interfaz de teclado numérico hexadecimal mostrado en la figura 11-4.

Solución

```
#include <reg51.h>
#include <stdio.h>

unsigned char R3, R5, R6, R7, tempA;
unsigned char bdata A;           /* representa al ACC */
sbit aMSB = A ^ 7;             /* la variable aMSB hace
                                referencia a A.7 */
```

```

sbit aLSB = A ^ 0;           /* la variable aLSB hace referencia
                             a A.0 */
bit PY;                      /* representa bit de paridad */

void HEX_ENT(void);
void HTOA(void);
void CARSAL(void);          /* tomado del ejemplo 8-6 y
                             modificado */

bit OBTEN_TECLA(void);
void RL_A(void);
bit RRC_A(bit);
void PARIDAD(void);

main( )
{
while (1)                  /* se repite infinitamente */
{
    HEX_ENT();             /* obtiene codigo del teclado
                             numerico */
    HTOA();                 /* convierte en ASCII */
    CARSAL();                /* envia a la consola */
}
}

void HEX_ENT(void)
{
R3 = 50;                    /* cuenta para eliminar rebotes */
while (R3 != 0)
{
    if (OBTEN_TECLA() == 0) /* se oprimio la tecla? */
        R3 = 50;            /* no: comprueba otra vez */
    else
        R3--;               /* si: repite 50 veces */
}

tempA = A;                  /* guarda codigo hexadecimal */
R3 = 50;                    /* espera a que se suelte la tecla
                             */
while (R3 != 0)
{
    if (OBTEN_TECLA() == 1) /* se oprimio la tecla? */
        R3 = 50;            /* si: sigue comprobando */
    else
        R3--;               /* no: repite 50 veces */
}
A = tempA;                  /* recupera codigo hexadecimal y
                             regresa */
}

bit OBTEN_TECLA(void)
{
bit C = 0;                  /* valor de retorno predeterminado:
                             no se oprimio tecla */

```

```

A = 0xFE;                                /* empieza con columna 0 */
for (R6 = 4; R6 > 0; R6--)
    /* comprueba las 4 columnas */
    /*
    {
        P1 = A;
        A = P1;
        tempA = A&(0xF0);
        if (tempA == 0xF0)
            RL_A();
        else
            {
                R7 = A;
                A = 4;
                C = 0;
                A = A - R6;
                R6 = A;
                A = R7;
                A = ( R7 << 4 | R7 >> 4 );
                for (R5 = 4; R5 > 0; R5--)
                    /*
                    {
                        C = RRC_A(C);
                        if (C == 0)
                            break;
                        else
                            R6 = R6 + 4;
                    }
                    C = 1;
                    A = R6;
                    return C;
                }
            }
        return C;
    }

void RL_A(void)
{
    bit tempBit;
    tempBit = aMSB;
    A = A << 1;
    aLSB = tempBit;
}

bit RRC_A(bit C)
{
    bit tempBit;
    tempBit = C;
}

```

```

C = aLSB;
A = A >> 1;
aMSB = tempBit;
return C;
}

void HTOA(void)
{
A = A & 0xF;
/* asegura que nibble superior se
borre */

if (A >= 0xA)
    A = A + 7;
/* 'A' a 'F'? */
/* si: suma adicional */
A = A + '0';
/* no: convierte directamente */
}

void CARSAL(void)
{
PARIDAD();
/* obtiene paridad par de A y
coloca en PY */

PY = !PY;
/* cambia a paridad impar */
aMSB = PY;
/* suma al código del carácter */
while (TI != 1);
/* Tx empty? no: comprueba otra vez
*/
TI = 0;
/* si: borra bandera y */
SBUF = A;
/* envía carácter */
aMSB = 0;
/* quita bit de paridad */
}

void PARIDAD(void)
{
int i;
PY = 0;
/* inicializa paridad en 0 */
for (i = 0; i < 8; i++)
    PY ^= (A >> i) & 1;
}

```

Análisis

Recuerde que los registros de R0 a R7 se utilizan para almacenar parámetros en funciones de C. Por esta razón, a menudo no es aconsejable usar estos registros como almacenamiento temporal cuando se programa el 8051 en C. En vez de ello, simplemente declare cualquier variable de 8 bits para almacenar. Observe también que en el ejemplo anterior denominamos de manera intencional a las variables con los nombres de registro R3 a R7, para que sea más fácil relacionar nuestro programa en C con su contraparte en lenguaje ensamblador de la figura 11-4.

Por la misma razón, el acumulador (ACC) se utiliza en la mayoría de las operaciones y, en consecuencia, a menudo se sobreescribe con datos, por lo que no debe usarlo en sus programas en C. En el ejemplo anterior utilizamos una variable temporal direccionable por bit, llamada A, para representar al acumulador.

Como no utilizamos ACC sino una variable A para representarlo, la paridad de A no se actualiza de manera automática en la bandera de paridad, P. En vez de ello, tenemos que calcularla por nuestra cuenta. Por lo tanto, modificamos ligeramente CARSAL() para incluir la llamada de una función PARIDAD() que calcule la paridad de A y la almacene en un bit, PY.

En nuestra solución también incluimos dos funciones, `RL_A()` y `RRC_A()`, las cuales en esencia funcionan de la misma forma que las instrucciones en lenguaje máquina `RL A` y `RRC A`. Fue necesario hacer esto en C porque en este lenguaje no hay instrucciones o funciones que soporten los desplazamientos cíclicos. En C únicamente se soportan las operaciones de desplazamiento (denotadas mediante `<<`).

12.3 INTERFAZ PARA VARIOS LEDS DE 7 SEGMENTOS

En el capítulo 11 presentamos un programa escrito en lenguaje ensamblador para conectar el 8051 con varios LEDs de 7 segmentos. Básicamente, el programa debía copiar dígitos en BCD almacenados en las ubicaciones 70H a 71H de la RAM interna, y enviarlos a la pantalla de LEDs 10 veces por segundo, usando interrupciones. A continuación presentamos el bosquejo de un posible programa escrito en C. Dejamos al lector los detalles de ponerlo a punto y adaptarlo de manera acorde a sus preferencias individuales.

EJEMPLO Interfaz para varios LEDs de 7 segmentos

12.2

Vuelva a escribir en lenguaje C el software desarrollado para la interfaz de varios LEDs de 7 segmentos en la figura 11-6.

Solución

```
#include <reg51.h>
#include <stdio.h>

unsigned char bdata A;           /* representa el ACC */
int xdata * idata DPTR;         /* DPTR en idata, apunta a
                                  xdata */
sbit DIN = P1^7;                /* lineas de interfaz MC14499 */
sbit CLOCK = P1^6;
sbit ENABLE = P1^5;
sbit aLSB = A^0;
sbit aMSB = A^7;
int xdata X8155 = 0x100;         /* direccion del 8155 */
int xdata TEMPORIZADOR = X8155 + 4; /* registros del temporizador */
int cuenta = 4000;              /* interrupciones @ 2000 µs */
unsigned char modo = 0x40;        /* bits de modo del
                                  temporizador */

int digitos , icuenta;
unsigned char tempA;

void ACTUALIZAR(void);
void SAL8(void);
bit RLC_A(bit);

main( )
{
    DPTR = TEMPORIZADOR;          /* inicializa temporizador 8155
                                  */
    A = 0xA0;                     /* byte inferior de cuenta para
                                  onda cuadrada de 500Hz */
    DPTR [TEMPORIZADOR] = A;
    TEMPORIZADOR++;
}
```

```

A = 0xF;                                /* byte superior de cuenta */
DPTR[TEMPORIZADOR] = A;
A = 0xC0;                                /* inicia comando de temporizador */
DPTR[X8155] = A;                         /* envia a registro de comandos del
                                            8155 */
icuenta = 50;                            /* inicializa contador int. */
EA = 1;                                  /* habilita interrupciones */
EX0 = 1;                                  /* habilita interrupcion 0 externa */
IT0 = 1;                                  /* disparado por flanco negativo */
while(1);                               /* no hace nada */
}

void EX0ISR(void) interrupt 0
{
if (--icuenta == 0)                      /* en 50a interrupcion */
{
icuenta = 50;                            /* reinicia contador y */
ACTUALIZAR();                           /* actualiza pantalla de LED */
}
}

void ACTUALIZAR(void)
{
tempA = A;                                /* guarda A */
ENABLE = 0;                               /* prepara MC14499 */
A = digitos ;                            /* obtiene los primeros dos digitos */
SAL8();                                   /* envia dos digitos */
A = digitos + 1;                          /* obtiene el segundo byte */
SAL8();                                   /* envia los ultimos dos digitos */
ENABLE = 1;                               /* deshabilita MC14499 */
A = tempA;                               /* restaura A */
}

void SAL8(void) using 0
{
for (tempA = 8; tempA > 0; tempA--)
{
{
CY = RLC_A(CY);                         /* coloca bit en bandera C */
DIN = CY;                                /* lo envia a MC14499 */
CLOCK = 0;                               /* pulso bajo de 3µs en linea de
                                            reloj */
CLOCK = 1;
}
}

bit RLC_A(bit C)
{
bit tempBit;

tempBit = C;                             /* respalda C */
C = aMSB;                                /* desplaza A.7 hacia C */
A = A << 1;                             /* desplaza A a la izquierda */
aLSB = tempBit;                          /* desplaza C hacia A.0 */
return C;
}
}

```

Análisis

En el 8051, el DPTR se utiliza como apuntador para apuntar a ubicaciones externas de memoria. Con el propósito de relacionar directamente nuestro programa en C con su contraparte en lenguaje ensamblador, utilizamos la declaración `int xdata * idata DPTR` para indicar que el DPTR es un apuntador de 16 bits almacenado en la memoria idata (en donde se encuentran los SFR) que apunta a una ubicación en la memoria xdata. En lenguaje ensamblador, los elementos incorporados en las tablas de búsqueda se obtienen mediante la instrucción `MOVX @DPTR, A` o `MOVC @DPTR, A`, en la cual el DPTR guarda la dirección base (inicial) de la tabla, mientras que A es el índice para un elemento específico. La situación es similar en C. Puede utilizarse un apuntador, que podemos llamar DPTR, para apuntar al primer elemento de un arreglo (el equivalente en C de una tabla de búsqueda). Después, la instrucción `DPTR [A]` haría referencia a un elemento específico localizado en el arreglo, donde A es el índice.

12.4 INTERFAZ PARA LAS PANTALLAS DE CRISTAL LÍQUIDO (LCD)

La sección 11.5 presentó un ejemplo de interfaz que implica el uso de una LCD. El programa que se escribió extraía en forma continua caracteres ASCII de las ubicaciones 30H a 7FH de la RAM interna y las mostraba en la LCD, 16 caracteres a la vez. En el ejemplo 12.3 se muestra el mismo programa, reescrito en C.

EJEMPLO**12.3 Interfaz para LCDs**

Vuelva a escribir en lenguaje C el software desarrollado para la interfaz de LCD en la figura 11-8.

Solución

```
#include <reg51.h>
#include <stdio.h>

sbit RS = P3^0;
sbit RW = P3^1;
sbit E = P3^2;
sbit ocupado = P1^7;
unsigned char bdata A; /* representa el ACC */
unsigned char * ptr;
unsigned char cuenta;
void INIC(void);
void NUEVO(void);
void MUESTRA(void);
void ESPERA(void);
void SAL(void);

main( )
{
INIC(); /* inicializa la LCD */
cuenta = 16; /* inicializa cuenta de caracteres */
while(1)
{
    for (ptr = 0x30; ptr < 0x80; ptr++, cuenta--)
    {
        /* Código para enviar datos a la LCD */
    }
}
}
```

```

        A = *ptr;           /* obtiene siguiente caracter */
        MUESTRA();          /* muestra en la LCD */
        if (cuenta==0)
        {
            cuenta = 16;    /* si es fin de linea, reinicializa
                                cuenta */
            NUEVO();          /* y actualiza la LCD */
        }
    }

void INIC(void)
{
    A = 0x38;           /* matriz de 5 × 7, 2 lineas */
    ESPERA();            /* espera que la LCD este libre */
    RS = 0;              /* envia un comando de salida */
    SAL();                /* lo envia de salida */
    A = 0x0E;            /* LCD encendida, cursor encendido */
    ESPERA();            /* espera que la LCD este libre */
    RS = 0;              /* envia un comando de salida */
    SAL();                /* lo envia de salida */
    NUEVO();              /* actualiza pantalla LCD */
}

void NUEVO(void)
{
    A = 0x01;           /* borra la LCD */
    ESPERA();            /* espera que LCD este libre */
    RS = 0;              /* envia un comando de salida */
    SAL();                /* lo envia de salida */
    A = 0x80;            /* cursor: linea 1, posicion 1 */
    ESPERA();            /* espera que la LCD este libre */
    RS = 0;              /* envia un comando de salida */
    SAL();                /* lo envia de salida */
}

void MUESTRA(void)
{
    ESPERA();            /* espera que la LCD este libre */
    RS = 1;              /* envia un dato de salida */
    SAL();                /* lo envia de salida */
}

void ESPERA(void)
{
do
{
    RS = 0;              /* comando */
    RW = 1;              /* lee */
    ocupado = 1;          /* convierte bit ocupado = entrada*/
}

```

```

        E = 1;           /* transición de 1 a 0 para */
        E = 0;           /* habilitar la LCD */
    }
while (ocupado);          /* si esta ocupada, espera */
}

void SAL(void)
{
P1 = A;                   /* prepara salida a la LCD */
RW = 0;                   /* escribe */
E = 1;                   /* transición de 1 a 0 para */
E = 0;                   /* habilitar la LCD */
}

```

Análisis

En este ejemplo, necesitamos acceder a ciertas ubicaciones consecutivas de la RAM interna. En lenguaje ensamblador, esto se haría mediante direccionamiento indirecto. En C, usamos los apuntadores, y como las ubicaciones a las que se apunta deben estar en la RAM interna nuestro apuntador, por decir `ptr`, debería definirse con la instrucción `unsigned char * ptr`. Sin embargo, aquí utilizamos `unsigned char` porque las ubicaciones de memoria contienen valores de 8 bits. El resto del programa es simple.

12.5 INTERFAZ PARA UN ALTAVOZ

En la sección 11.6 del capítulo anterior presentamos un programa en lenguaje ensamblador controlado por interrupciones para reproducir una escala musical en LA mayor. El siguiente ejemplo 12.4 muestra el correspondiente programa en C.

EJEMPLO **Interfaz para un altavoz**

12.4 Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un altavoz en la figura 11-10.

Solución

```

#include <reg51.h>
#include <stdio.h>

#define LONGITUD 12

sbit bitsalida = P1^7;
int recarga;           /* utilizamos esto para almacenar
                           temporalmente el valor de */
int codigo * PC;       /* recarga para la nota actual */
int REPETIR = 5;        /* PC apunta a la TABLA */
                        /* valor de recarga = -50000 causa
                           0.05 segundos por */
                        /* interrupcion. Lo hacemos 5 veces
                           para obtener 5 × 0.05 */
                        /* = 0.25 segundos por nota */
int nconteo, tconteo;  /* contador de notas y contador de
                           interrupciones */

```

```

int i, j;
/* tabla de busqueda de notas en la
   escala de LA mayor */
int codigo TABLA[LONGITUD] = {-1136, -1136, -1012, -902, -851,
                               758, -676,
                               -602, -568, -568, -568};

void OBTENVAL(void);

main()
{
    TMOD = 0x11;                      /* ambos temporizadores en modo de
                                         16 bits */
    nconteo = 0;                      /* inicializa contador de notas a 0 */
    tconteo = REPETIR;                /* inicializa contador de
                                         interrupciones a 5 */
    IE = 0x8A;                        /* habilitar interrupciones del
                                         temporizador 0 y 1 */
    TF1 = 1;                          /* forzar interrupcion del
                                         temporizador 1 */
    TF0 = 1;                          /* forzar interrupcion del
                                         temporizador 0 */
    while(1);                         /* ZzZzZzZz tiempo para una siesta */
}

void T0RSI(void) interrupt 1
{
    TR0 = 0;                          /* detener temporizador */
    TH0 = 0x3C;                       /* SUPERIOR(-50000) */
    TL0 = 0xB0;                       /* INFERIOR(-50000) */
    if (--tconteo == 0)
    {
        tconteo = REPETIR;           /* si es 5a interrupcion,
                                         reinicializar */
        nconteo++;                  /* incrementar nota */
        if (nconteo == LONGITUD) /* si estamos mas alla de la ultima
                                   nota... */
            nconteo = 0;             /* reinicializar, A = 440Hz */
    }
    TR0 = 1;                          /* inicia temporizador, vuelve a
                                         dormir */
}

void T1RSI(void) interrupt 3
{
    bitsalida = !bitsalida;          /* musica maestro! */
    TR1 = 0;                          /* detener temporizador */
    recarga = nconteo;               /* obtener conteo de notas */
    GETBYTE();                        /* obtener los dos bytes del valor
                                         de recarga */
    TH1 = recarga >> 8;             /* colocar byte superior en registro
                                         superior del temporizador */
    TL1 = recarga & 0xFF;            /* colocar byte inferior en registro
                                         inferior del temporizador */
    TR1 = 1;                          /* inicia temporizador */
}

```

```

void OBTENVAL(void)          /* funcion de busqueda en la tabla */
{
    PC = TABLA;             /* apunta a TABLA */
    recarga = PC[recarga];   /* lee de la TABLA al valor de
                                recarga */
}

```

Análisis

La solución en lenguaje ensamblador mostrada en la figura 11-10 utiliza una tabla de búsqueda incorporada en la memoria para código. Para realizar lo mismo en C, utilizamos un arreglo que está almacenado en la memoria para código. En la solución presentada líneas arriba, llamamos TABLA a este arreglo. La función OBTENVAL () realiza la tarea de obtener elementos del arreglo utilizando un apuntador PC para acceder a ellos. Observemos que, a diferencia de la solución escrita en lenguaje ensamblador en la figura 11-10, aquí no tenemos que leer dos veces de la tabla. Esto es debido a que en C podemos definir tablas de búsqueda con elementos de 16 bits para almacenar cada valor de recarga como un elemento dentro del arreglo, como hicimos en el ejemplo anterior. Con esto sólo tenemos que leer una vez de la tabla para obtener el valor de recarga de 16 bits. Después lo desplazamos 8 bits a la derecha para obtener su byte superior, y simplemente enmascaramos los 8 bits superiores (realizando una operación AND con 0 sobre ellos) para obtener su byte inferior.

12.6 INTERFAZ PARA UNA RAM NO VOLÁTIL

En la sección 11.7 mostramos cómo una memoria RAM no volátil (NVRAM) retiene su contenido aun en ausencia de energía. También presentamos un ejemplo de interfaz para copiar el contenido del 8051 a la NVRAM y después leer los valores almacenados. El ejemplo 12.5 presenta el programa escrito en C.

EJEMPLO Interfaz para una NVRAM

12.5 Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a una NVRAM en la figura 11-13.

Solución

```

#include <reg51.h>
#include <reg51.h>
#include <stdio.h>

#define RECUPERACION 0x85      /* instruccion de recuperacion de la X2444 */
#define ESCRITURA 0x84         /* instruccion de habilitacion de escritura
                                de la X2444 */
#define ALMACENAMIENTO 0x81    /* instruccion de almacenamiento de la
                                X2444 */
#define DESCANSO 0x82          /* instruccion de descanso de la X2444 */
#define E_DATOS 0x83           /* instruccion de escritura de datos de la
                                X2444 */
#define L_DATOS 0x87           /* instruccion de lectura de datos de la
                                X2444 */
#define LONGITUD 32            /* 32 bytes almacenados/recuperados */

unsigned char * R0;           /* utilizado para apuntar a las ubicaciones
                                en NVRAM */

unsigned char R5, R6, R7;
unsigned char bdata A;        /* representa al acumulador */
sbit aMSB = A ^ 7;           /* variable aMSB para referirnos a A.7 */

```

```

sbit aLSB = A ^ 0;      /* variable aLSB para referirnos a A.0 */
sbit DIN = P1^2;        /* lineas de interfaz de la X2444 */
sbit ENABLE = P1^1;
sbit CLOCK = P1^0;
bit C;
unsigned char NVRAM[LONGITUD] _at_ 0x60;

void ALMACENA(void);
void RECUPERA(void);
void L_BYTEx(void);
void E_BYTEx(void);
void RL_A(void);
bit RLC_A(bit);

main()
{
while(1)                /* se repite infinitamente */
{
    ALMACENA();          /* copiar de las ubicaciones internas 60H a
                           7FH del 8051 */
    /*      a la EPROM de la X2444      */
    RECUPERA();           /* leer datos almacenados previamente de la
                           EPROM de la X2444 a */
    /*      las ubicaciones 60H a 7FH      */
}
}

void ALMACENA(void)
{
R0 = NVRAM;              /* R0 para apuntar a las ubicaciones en
                           donde vamos a almacenar */
ENABLE = 0;                /* deshabilita X2444 */
A = RECUPERA;             /* instruccion de recuperacion */
ENABLE = 1;
E_BYTEx();
ENABLE = 0;
A = WRITE;                /* habilitacion de escritura prepara la
                           X2444 para escribir en ella */
ENABLE = 1;
W_BYTEx();
ENABLE = 0;
for (R7 = 0; R7 < 16; R7++)/* R7 = direccion de la X2444 */
{
    A = R7;                  /* colocar direccion en A */
    RL_A();                  /* colocar en bits 3, 4, 5, 6 */
    RL_A();
    RL_A();
    A = A|W_DATA;            /* construye instruccion de escritura */
    ENABLE = 1;
    E_BYTEx();
    for (R5 = 2; R5 > 0; R5--)
    {

```

```

        A = *R0;          /* obtener datos del 8051 */
        R0++;
        /* apuntar al siguiente byte */
        E_BYT();
    }
}

ENABLE = 0;
}
A = ALMACENAMIENTO;      /* si terminamos, copiar a la EPROM */
ENABLE = 1;
E_BYT();
ENABLE = 0;
A = DESCANSO;           /* llevar a la X2444 a dormir */
ENABLE = 1;
E_BYT();
ENABLE = 0;
}

void RECUPERA(void)
{
R0 = NVRAM;
ENABLE = 0;
A = RECUPERACION;       /* instruccion de recuperacion */
ENABLE = 1;
E_BYT();
ENABLE = 0;
for (R7 = 0; R7 < 16; R7++)/* R7 = direccion de la X2444 */
{
    A = R7;           /* colocar direccion en A */
    RL_A();           /* construye instruccion de lectura */
    RL_A();
    RL_A();
    A = A|R_DATA;
    ENABLE = 1;
    E_BYT();           /* enviar instruccion de lectura */
    for (R5 = 2; R5 > 0; R5--)
    {
        L_BYT();           /* leer byte de datos */
        *R0 = A;           /* colocar en RAM del 8051 */
        R0++;              /* apuntar a la siguiente ubicacion */
    }
    ENABLE = 0;
}
A = DESCANSO;           /* llevar a la X2444 a dormir */
ENABLE = 1;
E_BYT();
ENABLE = 0;
}

void L_BYT(void)
{
for (R6 = 8; R6 > 0; R6--) /* utiliza R6 como contador de bit */

```

```

{
C = DIN;           /* colocar bit de datos de la X2444 en
                    C */
C = RLC_A(C);    /* construye byte en el Acumulador */
CLOCK = 1;        /* activar linea de reloj (1 us) */
CLOCK = 0;
}
}

void E_BYTE(void)
{
for (R6 = 8; R6 > 0; R6--) /* utiliza R6 como contador de bit */
{
C = RLC_A(C);          /* colocar en C el bit a escribir */
DIN = C;                /* colocar en la linea DATA IN de la
                           X2444 */
CLOCK = 1;              /* bit de reloj en la X2444 */
CLOCK = 0;
}
}

void RL_A(void)
{
bit tempBit;

tempBit = aMSB;          /* respaldar MSB de A */
A = A << 1;             /* desplazar A a la izquierda */
aLSB = tempBit;          /* desplazar LSB hacia MSB */
}

bit RLC_A(bit C)
{
bit tempBit;

tempBit = C;              /* respalda C */
C = aMSB;                /* desplazar ACC.7 hacia C */
A = A << 1;              /* desplazar ACC hacia la izquierda */
aLSB = tempBit;           /* desplazar C hacia ACC.0 */
return C;
}

```

Análisis

El programa requiere que se asigneen 32 bytes en las ubicaciones de memoria interna para datos desde 60h hasta 7FH. Esto se lleva a cabo declarando un arreglo, NVRAM de 32 elementos de byte, y especificando que debe iniciar en la dirección absoluta 60H. El programa principal continuamente realiza el almacenamiento de los bytes de datos de las ubicaciones internas para datos 60H a 7FH a la NVRAM, con lo cual se recuperan los datos previamente almacenados de la NVRAM a las ubicaciones internas para datos 60H a 7FH. Los elementos se leen del arreglo NVRAM en forma indirecta por medio de un apuntador, R0. Los datos se envían en serie hacia y desde la NVRAM, y utilizamos las funciones E_BYTE() y L_BYTE(), respectivamente, para cumplir dicho propósito.

12.7 AMPLIACIÓN DE ENTRADAS/SALIDAS

En el capítulo anterior mostramos dos maneras de expandir las E/S en lenguaje ensamblador. Los siguientes dos ejemplos muestran cómo podemos realizar lo mismo en C. En especial, el ejemplo 12.6 muestra el método de ampliación de E/S mediante registros de desplazamiento, mientras que el ejemplo 12.7 muestra cómo expandir las E/S utilizando el 8255.

EJEMPLO Interfaz para registros de desplazamiento

12.6

Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un registro de desplazamiento en la figura 11-16.

Solución

```
#include <reg51.h>
#define CONTEO 2                                /* cantidad de registros de desplazamiento */
unsigned char bdata * R0;
unsigned char R6, R7;
unsigned char xdata * idata DPTR;    /* DPTR en datos, apunta a xdata */
unsigned char bdata A;                /* representa el ACC */
sbit aMSB = A^7;
sbit aLSB = A^0;
sbit SHIFT = P1^7;                  /* entrada a señal SHIFT/LOAD: 1 = desplazamiento, 0 = carga */
sbit CLOCK = P1^6;                  /* entrada a señal CLOCK */
sbit DOUT = P1^5;                   /* salida de DATA OUT */
bit C;
bit PY;                            /* representa bit de PARIDAD */

char * MENSAJE = {"*** PRUEBA DE INTERFAZ A UN 74HC165 ***\n"};
unsigned char bdata BUFER[CONTEO]; /* bufer para almacenar los bytes leidos */
void OBTEN_BYTES(void);
void ENVIA_MENSAJE_HOLA(void);
void PRESENTA_RESULTADOS(void);
bit RRC_A(bit C);
void CADSAL(void);
void CARSAL(void);
void SALIDA2HEX(void);
void INTERCAMBIA_A(void);
void PARIDAD(void);
void H_A_A(void);
main()
{
CLOCK = 1;                          /* activa lineas de interfaz inicialmente en ...
... */
SHIFT = 1;                          /* ... caso de que no sea asi */
DOUT = 1;                           /* DOUT debe activarse (entrada) */
ENVIA_MENSAJE_HOLA();
while(1)
{
OBTEN_BYTES();                      /* leer registros de desplazamiento */
}
```

```

        DISPLAY_RESULTS();           /* presenta resultados */
    }
}

void OBTEN_BYTES(void)
{
for (R6 = CONTEO; R6 > 0; R6--) /* utiliza R6 como contador de bytes */
{
    R0 = 0x25;                  /* utiliza R0 como apuntador al bufer en
                                bdata */
    SHIFT = 0;                  /* cargar a los registros de desplazamiento
                                enviando pulsos ... */
    SHIFT = 1;                  /* ... a la señal SHIFT/LOAD a nivel bajo */
    for (R7 = 8; R7 > 0; R7--) /* utiliza R7 como contador de bits */
    {
        C = DOUT;                /* obtener un bit (colocarlo en C) */
        C = RRC_A(C);            /* colocarlo en A.0 (LSB primero) */
        CLOCK = 0;                /* enviar pulso a la linea de CLOCK (desplaza
                                bits hacia ... */
        CLOCK = 1;                /* ... DATA OUT */
    }
    *R0 = A;                    /* si es el 8o desplazamiento, colocalo en el
                                bufer */
    R0++;                      /* incrementa el apuntador al bufer */
}
}

void ENVIA_MENSAJE_HOLA(void)
{
DPTR = MENSAJE;                /* apunta al mensaje de hola */
CADSAL();                      /* envialo a la consola */
}

void PRESENTA_RESULTADOS(void)
{
R0 = 0x25;                      /* R0 apunta a los bytes */
for (R6 = CONTEO; R6 > 0; R6--) /* utiliza R6 como contador de bytes */
{
    {
        A = *R0;                /* obten byte */
        R0++;                   /* incrementa el apuntador */
        SALIDA2HEX();            /* enviarlo como caracter de 2 hex como
                                salida */
        A = ' ';
        CARSAL();                /* separa los bytes */
    }
    A = '\n';                   /* repite para cada byte */
    CARSAL();                /* comenzar una linea nueva*/
}

bit RRC_A(bit C)
{
bit tempBit;

tempBit = C;                    /* respaldar C */
C = aLSB;                      /* desplazar A.0 hacia C */
A = A >> 1;                   /* desplazar A hacia la derecha */
}

```

```

aMSB = tempBit;                                /* desplazar C hacia A.7 */
return C;
}

void CADSAL(void)
{
while (1)
{
    A = 0;
    A = DPTR[A];                            /* obtener codigo en ASCII */
    if (A == 0)                             /* si es ultimo codigo, terminamos */
        break;
    CARSAL();                               /* si no es ultimo codigo, envialo */
    A++;                                    /* apunta al siguiente codigo */
}
}

void CARSAL(void)
{
PARIDAD();                                     /* obten paridad par de A y colocalo en PY */
PY = !PY;                                       /* cambia a paridad impar */
aMSB = PY;                                      /* agrega al codigo del caracter */
while (TI != 1);                             /* ¿Tx vacio? no: verifica de nuevo */
TI = 0;                                         /* si: limpia bandera y */
SBUF = A;                                       /* envia caracter */
aMSB = 0;                                       /* remueve bit de paridad */
}

void SALIDA2HEX(void)
{
unsigned char tempA = A;                      /* almacena A en tempA */
INTERCAMBIA_A();                            /* envia nibble superior primero */
A = A & 0xf;                                 /* enmascara nibble no deseado */
HTOA();                                      /* convierte nibble en hex a ASCII */
CARSAL();                                     /* envia al puerto serial */
A = tempA;                                    /* recupera A y envia nibble inferior */
}

void INTERCAMBIA_A(void)
{
A = (A >> 4) | (A << 4);                /* intercambia los nibbles superior e inferior
                                                 de A */
}

void PARIDAD(void)
{
int i;
PY = 0;                                         /* inicializa paridad a 0 */
}

```

```

for ( i = 0; i < 8 ; i++ )      /* calcula paridad de A*/
    PY ^= ( A >> i ) & 1;
}
void HTOA(void)
{
A = A & 0xF;                  /* asegura que nibble superior esta limpio */
if ( A >= 0xA )
    A = A + 7;
A = A + '0';                  /* si: agrega extra */
                                /* no: convierte directamente */
}

```

Análisis

El programa envía primero un mensaje de hola al dispositivo de visualización conectado al puerto serial. Esto involucra esencialmente apuntar al mensaje y llamar a OUTSTR(), la cual llama a la función CARSAL() para enviar el mensaje un carácter a la vez. Después se llama a la función OBTENBYTES() para obtener dos bytes del registro de desplazamiento. Estos bytes se presentan inmediatamente al llamar la función PRESENTA_RESULTADOS(). Básicamente, OBTENBYTES() desplaza los valores ubicados en el byte bit a bit. Por otra parte, PRESENTA_RESULTADOS() utiliza la función SALIDA2HEX() para convertir los valores de byte leídos en hexadecimal al formato ASCII antes de llamar a CARSAL() para enviar los caracteres en ASCII para su visualización.

El ejemplo de diseño presentado en la sección 11.8.2 del capítulo anterior trataba sobre cómo podemos utilizar el 8255 para agregar tres puertos de E/S adicionales. A manera de ilustración, escribimos un programa en lenguaje ensamblador para leer el estatus de ocho interruptores conectados al puerto A, y para cada interruptor cerrado encendimos un LED correspondiente conectado a una de las terminales del puerto B. El programa en C es como se muestra a continuación:

EJEMPLO

12.7

Interfaz para un 8255

Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un 8255 en la figura 11-18.

Solución

```

#include <reg51.h>
#include <stdio.h>
unsigned char xdata * idata DPTR; /* DPTR en datos, apunta a
                                    xdata */
unsigned char bdata A;           /* representa al ACC */
sbit a_0 = A^0;
sbit a_1 = A^1;
sbit a_2 = A^2;
sbit a_3 = A^3;
sbit a_4 = A^4;
sbit a_5 = A^5;
sbit a_6 = A^6;
sbit a_7 = A^7;
void CPL_A();
main()

```

```

{
A = 0x90;           /* Puerto A = entrada, puerto B = salida
                     */
DPTR = 0x0103;      /* apunta al registro de control */
*DPTR = A;          /* envia palabra de control */

while(1)
{
    DPTR = 0x0100;  /* apunta al puerto A */
    A = *DPTR;       /* lee interruptores */
    CPL_A();         /* complemento */
    DPTR = 0x0101;  /* apunta al puerto B */
    *DPTR = A;       /* enciende LEDs */
}
}

void CPL_A(void)
{
a_0 = (!a_0);
a_1 = (!a_1);
a_2 = (!a_2);
a_3 = (!a_3);
a_4 = (!a_4);
a_5 = (!a_5);
a_6 = (!a_6);
a_7 = (!a_7);
}

```

Análisis

Este ejemplo también utiliza al apuntador DPTR para acceder a las ubicaciones en memoria externa. La operación del complemento de todo el contenido de 8 bits en A lleva a cabo la función CPL_A(), una función simple y directa que tan sólo complementa cada bit dentro de A.

12.8 INTERFAZ SERIAL RS232 (EIA-232)

En la sección 11.9 vimos cómo conectar el 8051 a una PC mediante la interfaz serial RS232. El ejemplo de diseño fue escribir un programa para recibir números en decimal como entrada de la PC y después enviar el correspondiente código en ASCII a la pantalla de la PC. El ejemplo 12.8 muestra el programa en C para llevar esto a cabo.

EJEMPLO Interfaz a RS232
12.8

Vuelva a escribir en lenguaje C el software desarrollado para la interfaz RS232 en la figura 11-20.

Solución

```
#include <reg51.h>
#include <stdio.h>

unsigned char bdata A;           /* representa el ACC */
sbit aMSB = A ^ 7;              /* variable aMSB para referirnos a A.7 */
sbit RTS = P1^7;                /* variable RTS para referirnos a P1.7 */
```

```

sbit CTS = P1^6;                      /* variable CTS para referirnos a P1.6 */
bit C;                                /* representa al bit de acarreo */
unsigned char code * idata DPTR;       /* DPTR en datos, apunta a codigo */
unsigned char code ASC[10] ={0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38,
                            0x39};           /* tabla ASCII */
char code * MSJ
= {"POR FAVOR TECLEE UN NUMERO = "}; /* mensaje inicial */
void FAZ(void);
void CARENT(void);                     /* tomada y modificada del ejemplo 8-7 */
void CARSAL(void);                    /* tomada y modificada del ejemplo 8-6 */
void PARIDAD(void);
main()
{
FAZ();                                /* inicializa puerto serial y completa
                                         protocolo de intercambio */
DPTR = ASC;                           /* apunta a tabla ASCII */
while(1)
{
    {
        CARENT();                      /* obten numero decimal y colocalo en A */
        A = DPTR[A];                  /* convierte a ASCII y lo almacena en A */
        CARSAL();                     /* envia ASCII al puerto serial */
    }
}
void FAZ (void)
{
TMOD = 0x20;                          /* temporizador 1, modo 2 */
TH1 = 0x98;                           /* valor de recarga para 9600 baudios */
SCON = 0x52;                          /* puerto serial, modo 1 */
DPTR = MSG;                           /* apuntador al mensaje inicial */
TR1 = 1;                             /* inicia temporizador 1 */
RTS = 0;                             /* asegura RTS */
while (CTS == 1);                   /* espera a recibir CTS */
do
{
    {
        A = DPTR[A];                /* obtiene caracteres en ASCII */
        CARSAL();                  /* los envía */
        DPTR++;                    /* si no es fin del mensaje, obtiene el
                                     siguiente caracter */
    }
}
while (A != 0);                      /* si es fin de mensaje, alto */
}
void CARENT(void)
{
while (RI != 1);                    /* espera por un caracter */
RI = 0;                            /* limpia bandera */
}

```

```

A = SBUF;
PARIDAD();
C = PY;
C = !C;
aMSB = 0;
}

void CARSAL(void)
{
PARIDAD();
PY = !PY;
aMSB = PY;
while (TI != 1);
TI = 0;
SBUF = A;
aMSB = 0;
}

void PARIDAD(void)
{
int i;
PY = 0;
for ( i = 0; i < 8 ; i++ )
    PY ^= ( A >> i ) & 1;
}

```

/* lee caracter a A */
 /* obtiene paridad par de A y la coloca en PY */
 /* para una paridad impar en el acumulador,
 PY debe activarse */
 /* complemento correcto indica si hubo
 "error" */
 /* remueve paridad */

/* obtiene paridad par de A y la coloca en PY */
 /* cambia a paridad impar */
 /* agrega a codigo de caracter */
 /* Tx vacio? no: verifica de nuevo */
 /* si: limpia bandera y */
 /* envia caracter */
 /* remueve bit de paridad */

/* inicializa paridad a 0 */
 /* calcula paridad de A */

Análisis

Este ejemplo utiliza conceptos familiares tales como el uso de una variable direccionable por un bit temporal, A, para representar al acumulador, la utilización de un arreglo en memoria para código para representar una tabla de búsqueda, y la utilización del apuntador, DPTR, para acceder a dicha tabla de búsqueda. Además de esto, recordemos que en el lenguaje ensamblador algunas veces deseamos almacenar datos en ubicaciones consecutivas en memoria para código mediante las directivas del ensamblador DB o DW. Podemos implementar esto en C utilizando un arreglo almacenado en la memoria para código. También hacemos uso de la función CARENT(), la cual es similar a la función CARENT() estudiada en el capítulo 8, sólo que, como utilizamos A en lugar del acumulador, necesitamos calcular la paridad nosotros mismos llamando a la función PARIDAD(), ello causa que la paridad se almacene en PY.

12.9 INTERFAZ PARALELA CENTRONICS

Las impresoras interactúan con las computadoras a través del puerto paralelo; el estándar común utilizado se conoce como interfaz paralela Centronics. En el capítulo anterior mostramos un programa para verificar el estado de la impresora antes de enviar un mensaje de prueba en forma continua para imprimirla. El ejemplo 12.9 muestra cómo podemos realizar esto en C.

EJEMPLO Interfaz para la interfaz paralela Centronics

12.9 Vuelva a escribir en lenguaje C el software desarrollado para la interfaz paralela en la figura 11-22.

Solución

```
#include <reg51.h>
#include <stdio.h>

unsigned char bdata A;                                /* representa el ACC */
unsigned char MASCARA = 0x3C;                         /* solo verifica 4 de los bits */
unsigned char OK = 0x30;                               /* valores normales de los 4 bits de
                                                         estatus */

sbit STR = P3^0;                                      /* apunta al mensaje de prueba */
sbit ACK = P3^1;                                       /* DPTR en datos, apunta al codigo */
sbit BUSY = P3^2;

char code * MSJ =
    { "ESTA ES UNA PRUEBA PARA LA IMPRESORA" }; /* mensaje de prueba */
unsigned char code * idata DPTR;                      /* DPTR en datos, apunta al codigo */

main()
{
    while(1)
    {
        DPTR = MSJ;                                     /* apunta al mensaje de prueba */
        do
        {
            P3 = MASCARA;                             /* active señal de STROBE, P3.1 a P3.5
                                                       como entrada */
            A = P3;                                    /* lee estado de la impresora */
            A = A & MASCARA;                          /* solo deseamos P3.1 a P3.5 */
            if (A != OK)                            /* algun error? */
                return;
            A = 0;                                   /* si: alto */
            while (ACK == 1);                        /* no hay error, prepararse para
                                                       enviar */
            A = DPTR[                                /* antes de enviar, espera la señal
                                                       ACK */
            P1 = A;                                 /* obtiene caracter del mensaje de
                                                       prueba */
            DPTR++;                                /* envia caracter a la impresora */
            /* si no es el fin, obtiene el
               siguiente caracter */
        }
        while (A != 0);                            /* si es fin del mensaje, alto */
    }
}
```

Análisis

Aquí utilizamos la variable de 8 bits MASCARA para almacenar el valor de enmascarado, y especificamos cuáles bits estamos enmascarando (ignorando) al asignarles un cero. Recordemos que una operación lógica AND está denotada por un símbolo de “&” y no por dos. Dos símbolos “&&” denotarían el operador relacional AND.

12.10 SALIDAS ANALÓGICAS

A menudo, la interfaz para un dispositivo análogo requiere el uso de ADCs y DACs. En la sección 11.11 mostramos cómo conectar el 8051 al DAC MC1408L8 para generar una onda sinusoidal análoga utilizando una tabla de búsqueda. Aquí presentamos el programa en C correspondiente.

EJEMPLO Salida analógica

12.10 Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un DAC en la figura 11-25.

Solución

```
#include <reg51.h>

#define MAX 1024
                /* TABLA truncada de 1024 registros */
unsigned char TABLA[MAX] = {127, 128, 129, 130, 131, ...};

int data PASO = 1;          /* puede inicializarse a cualquier valor de
                            incremento */
unsigned char A;           /* representa al ACC */
int indice = 0;            /* utilizado para apuntar a los registros en TABLA
                            */

main()
{
    TMOD = 0x2;             /* 8 bits, autorrecarga */
    TH0 = -100;              /* retraso de 100 ms */
    TR0 = 1;                 /* inicia temporizador */
    IE = 0x82;               /* habilita interrupcion del temporizador 0 */
    while(1);                /* ;ciclo principal no hace nada! */
}

void T0RSI (void) interrupt 1
{
    index = indice + PASO;   /* agrega PASO al indice */
    if (index > MAX)
        index = 0;           /* si llegamos al fin de la TABLA, regresamos al
                            inicio */
    A = TABLA[index];         /* obtiene entrada */
    P1 = A;                  /* la envía */
}
```

Análisis

El programa principal inicializa el temporizador 0, habilita la interrupción del temporizador 0 correspondiente, y luego ya no hace nada. La mayor parte del programa reside en la función de interrupción del temporizador 0 la cual, cada vez que ocurre una interrupción, accede a un registro de TABLA[] que es un arreglo de 1024 registros entre 0 y 255 correspondientes a las magnitudes de un periodo de una onda sinusoidal. (Observe que en el programa anterior, TABLA[] se muestra en forma truncada). Cuando llegamos al fin de la tabla, su índice reinicia a 0 para que el siguiente registro obtenido empiece de nuevo desde el principio de la tabla.

Un programa separado parecido al de la figura 11-24 se utiliza para generar los 1024 registros de TABLA [] antes de ejecutar nuestro programa.

12.11 ENTRADAS ANALÓGICAS

La figura 11-27 mostró el uso de un programa en lenguaje ensamblador, en el que se utiliza un ADC para leer y convertir el voltaje de la toma central de un potenciómetro de ajuste pequeño a su forma digital y poder enviar el correspondiente código en ASCII como salida a la consola. A continuación presentamos el programa escrito en C.

EJEMPLO Entrada analógica

12.11 Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un ADC en la figura 11-27.

Solución

```
#include <reg51.h>

#define PUERTOA 0x101      /* puerto A del 8155 */
int xdata * idata DPTR;    /* DPTR en idata, apunta a xdata */
unsigned char bdata A;     /* representa al ACC */
sbit aMSB = A ^ 7;        /* variable aMSB para referirnos a A.7
                           */
sbit aLSB = A ^ 0;        /* variable aLSB para referirnos a A.0
                           */
sbit ESCRIBE = P1^0;       /* linea WR del ADC0804 */
sbit INTR = P1^1;          /* linea INTR del ADC0804 */
bit PY;                   /* representa al bit de paridad */
char * MENSAJE = { "**** PRUEBA DEL ADC0804 ***\n" };

void CADSAL(void);
void CARSAL(void);
void SALIDA2HEX(void);
void INTERCAMBIA_A(void);
void PARIDAD(void);
void HTOA(void);

main()
{
    DPTR = MENSAJE;           /* envia mensaje */
    CADSAL();
    while(1)
    {
        ESCRIBE = 0;           /* dispara linea WR */
        ESCRIBE = 1;
        while (INTR == 1);    /* espera a que INTR = 0 */
        DPTR = PORTA;          /* init DPTR -> puerto A */
        A = *DPTR;              /* lee datos del ADC0804 */
        SALIDA2HEX();          /* envia datos a la consola */
    }
}
```

```

void CADSAL(void)
{
while (1)
{
    {
        A = 0;
        A = DPTR[A];           /* obtiene codigo en ASCII */
        if (A == 0)            /* si es el ultimo codigo, terminamos */
            break;
        CARSAL();              /* si no es el ultimo codigo, lo envia */
        A++;                  /* apunta al siguiente codigo */
    }
}
void CARSAL(void)
{
PARIDAD();
PY = !PY;
aMSB = PY;
while (TI != 1);
TI = 0;
SBUF = A;
aMSB = 0;
}
/* obtiene paridad par de A y la coloca en PY */
/* cambia a paridad impar */
/* agrega al codigo del caracter */
/* ¿Tx vacio? no: verifica de nuevo */
/* si: limpia bandera y */
/* envia caracter */
/* remueve bit de paridad */

void SALIDA2HEX (void)
{
unsigned char tempA = A;
INTERCAMBIA_A();
A = A & 0xf;
HTOA();
CARSAL();
A = tempA;
}
/* almacena A en tempA */
/* envia nibble superior primero */
/* enmascara nibble no deseado */
/* convierte nibble en hex a ASCII */
/* envia al puerto serial */
/* recupera A y envia nibble inferior */

A = A & 0xf;
HTOA();
CARSAL();
A = tempA;
}

void INTERCAMBIA_A(void)
{
A = (A >> 4) | (A << 4);      /* intercambia los nibbles superior e inferior de A */
}

void PARIDAD(void)
{
int i;
PY = 0;                         /* inicializa paridad a 0 */
for ( i = 0; i < 8 ; i++ ) /* calcula paridad de A*/
    PY ^= ( A >> i ) & 1;
}

```

```

void HTOA(void)
{
    A = A & 0xF;                      /* asegura que el nibble superior esta
                                         limpio */
    if ( A >= 0xA )
        A = A + 7;                   /* De 'A' a 'F'? */
    A = A + '0';                     /* si: agrega extra */
}                                     /* no: convierte directamente */

```

Análisis

El programa principal presenta primero un mensaje a la consola y después dispara la línea WR para iniciar la conversión de análoga a digital. Después espera a que la línea INTR del ADC cambie a nivel bajo, lo cual indica que la conversión se ha completado antes de leer datos del ADC a través del puerto A del 8155. Los datos se convierten al formato hexadecimal antes de enviarlos a la consola para su visualización.

12.12 INTERFAZ PARA SENSORES

En esta sección veremos de nuevo cómo es que el 8051 puede interactuar con un sensor típico, el sensor de temperatura. Recuerde que en el capítulo 11 presentamos un programa en lenguaje ensamblador para monitorear la temperatura ambiental utilizando el sensor de temperatura DS1620. Cuando la temperatura ambiental es superior a 23°C, el horno se enciende y la alarma se activa. Si la temperatura cae por debajo de 17°C, el horno se apaga y la alarma se desactiva. El ejemplo 12.12 muestra el correspondiente programa en C para llevar esto a cabo.

EJEMPLO **Interfaz a un DS1620**

12.12 Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un sensor en la figura 11-29.

Solución

```

#include <reg51.h>
#include <stdio.h>
unsigned char bdata A;      /* representa al ACC */
sbit aMSB = A ^ 7;         /* variable aMSB para referirnos a A.7
                           */
sbit aLSB = A ^ 0;          /* variable aLSB para referirnos a A.0
                           */
sbit DQ = P1^0;
sbit CLK = P1^1;
sbit RST = P1^2;
sbit THI = P1^3;
sbit TLO = P1^4;
sbit TC0M = P1^5;
sbit HORNO = P1^6;
int i;
void ENVIAR(void);
bit RRC_A(bit);

```

```

main()
{
HORNO = 0;                                /* apaga horno */
RST = 1;                                   /* inicia transferencia */
A = 0xC;                                    /* escribe configuración */
ENVIAR();                                    /* envia al DS1620 */
RST = 0;                                   /* detiene transferencia */
RST = 1;                                   /* inicia transferencia */
A = 1;                                     /* escribe TH */
ENVIAR();                                    /* envia al DS1620 */
A = 44;                                    /* TH = 44 × 0.5 °C = 22 °C */
ENVIAR();                                    /* envia al DS1620 */
RST = 0;                                   /* detiene transferencia */
RST = 1;                                   /* inicia transferencia */
A = 2;                                     /* escribe TL */
ENVIAR();                                    /* envia al DS1620 */
A = 36;                                    /* TL = 36 × 0.5 °C = 18 °C */
ENVIAR();                                    /* envia al DS1620 */
RST = 0;                                   /* detiene transferencia */
RST = 1;                                   /* inicia transferencia */
A = 0xEE;                                  /* inicia detección de temperatura */
ENVIAR();                                    /* envia al DS1620 */
RST = 0;                                   /* detiene transferencia */
while(1)                                    /* continua detectando infinitamente */
{
    if (THI == 1)                           /* if T ≥ 22 °C, horno = apagado */
        HORNO = 0;
    if (TLO == 1)                           /* if T ≤ 18 °C, horno = encendido */
        HORNO = 1;
}
void ENVIAR(void)
{
for (i = 8; i > 0; i--) /* ciclo para todos los 8 bits */
{
    CLK = 0;                                /* inicia ciclo de reloj/sincronización */
    CY = RRC_A(CY);                         /* desplaza A hacia CY, LSB primero */
    DQ = CY;                                /* envia bit hacia DQ */
    CLK = 1;                                /* completa el ciclo de
                                                reloj/sincronización */
}
}

```

```

bit RRC_A(bit C)
{
    bit tempBit;
    tempBit = C;           /* respalda C */
    C = aLSB;              /* desplaza A.0 hacia C */
    A = A >> 1;           /* desplaza A hacia la derecha */
    aMSB = tempBit;        /* desplaza C hacia A.7 */
    return C;
}

```

Análisis

En este ejemplo, la mayor parte de la funcionalidad se lleva a cabo mediante la función `envia()`, la cual envía en serie (comenzando con el LSB) un dato de 8 bits a la terminal de puerto conectada a la entrada DQ (datos) del sensor.

12.13 INTERFAZ PARA RELEVADORES

En el capítulo 11 presentamos una interesante aplicación de los relevadores, la de un sistema de semáforos peatonales. El 8051 controló eléctricamente al relevador para cambiar entre dos estados, ya fuese el estado de encender la luz de tráfico ROJA y la luz peatonal VERDE para permitir que los peatones cruzaran la calle, o encender la luz de tráfico VERDE y la luz peatonal ROJA. El ejemplo 12.13 presenta el programa escrito en C para controlar todo esto.

EJEMPLO
12.13
Interfaz a un relevador: sistema de semáforos peatonales

Vuelva a escribir en lenguaje C el software desarrollado para una interfaz a un relevador en la figura 11-33.

Solución

```

#include <reg51.h>
#include <stdio.h>

sbit LEDs = P1^0;
int mil = 100;           /* 1000 × 10000 µs = 10 segundos */
int conteo;              /* para almacenar los valores de conteo */

void retraso(void);

main()
{
    IE = 0x81;           /* habilita INT0 */
    IT0 = 1;               /* disparada por borde negativo */
    TMOD = 1;              /* temporizador 0 en modo 1 */

```

```

LEDs = 0;                                /* al principio, trafico = VERDE, peaton =
                                             ROJA */
while (1);                                /* espera para siempre */
}

void retraso(void)           /* retraso de 10 segundos */
for (conteo = mil; conteo > 0; conteo--)
{
    TH0 = 0x6C;
    TL0 = 0x70;
    TR0 = 1;
    while(TFO != 1);
    TF0 = 0;
    TR0 = 0;
}
}

void EX0RSI(void) interrupt 0
{
    LEDs = 1;                            /* luz de trafico = ROJA, peaton = VERDE
                                             */
    retraso();                           /* espera 10 segundos */
    LEDs = 0;                            /* luz de trafico = VERDE, peaton = ROJA
                                             */
}

```

Análisis

Si comparamos este programa con su versión en lenguaje ensamblador, parecerá que ambos son bastante similares. La única diferencia está en el retraso generado por la función `retraso()`, la cual supuestamente debe esperar 10 segundos para permitir que los peatones crucen la calle antes de que las luces de tráfico y peatonal cambien a sus estados iniciales. Sin embargo, el retraso real generado por esta función `retraso()` depende del compilador de C del 8051 utilizado. Para poder determinar el retraso exacto generado, debemos examinar las instrucciones correspondientes generadas por el compilador en lenguaje ensamblador. En el IDE μ Vision2 de Keil, esto se lleva a cabo al seleccionar la opción “Disassembly Window” del menú “View”.

12.14 INTERFAZ PARA UN MOTOR PASO A PASO

El motor paso a paso encuentra aplicaciones interesantes cuando se requieren componentes de posicionamiento preciso. En el capítulo 11 vimos un ejemplo de un programa en lenguaje ensamblador que demostraba inicialmente cómo girar un motor paso a paso en el sentido de las manecillas del reloj, y que cuando un interruptor conectado a la interrupción externa 0 (INT0) genera una transición de nivel alto a nivel bajo, la dirección del giro debería invertirse. En esta sección presentamos el programa equivalente escrito en C.

EJEMPLO Interfaz a un motor paso a paso**12.14**

Vuelva a escribir en lenguaje C el software desarrollado para la interfaz a un motor paso a paso en la figura 11-38.

Solución

```
#include <reg51.h>
#include <stdio.h>

unsigned char A; /* espejo del ACC */
unsigned char tempA; /* variable temporal */
unsigned char code * idata DPTR; /* DPTR en datos, apunta a codigo */
bit D; /* bit de direccion para la direccion actual */
int CIEN = 100; /* 100 × 10000 µs = 1 seg */
/* secuencia de 8 pasos para una rotacion en el sentido de las manecillas del reloj */
unsigned char code SEC[8] = {0x9, 0x8, 0xC, 0x4, 0x6, 0x2, 0x3, 0x1};

void SR(void);
void CSR(void);
void retraso(void);

main()
{
IE = 0x81; /* habilita INT0 */
IT0 = 1; /* disparado por borde negativo */
TMOD = 1; /* temporizador 0 en modo 1 */
D = 0; /* inicializa D = 0 para rotacion en el sentido de las manecillas del reloj */
SR(); /* La rotacion inicial es en el sentido de las manecillas del reloj */
/* repetir para siempre */

while(1)
{
if (D != 1) /* En el sentido de las manecillas del reloj previamente? */
CSR(); /* no: cambia a sentido de las manecillas del reloj */
else /* si: cambia en contra del sentido de las manecillas del reloj */
SR();
}
}

void SR(void)
{
DPTR = SEC; /* funcion para giro en el sentido de las manecillas del reloj */
for (A = 0; A < 8; A++)
{
tempA = A; /* respalda el indice en A */
/* apunta al inicio de la tabla */
}
```

```

        A = DPTR[A];           /* obtiene patron de pasos a los 4 LSB
                                   de A */
        A = A | (P1&0xF0);   /* retener los 4 MSB de P1 y colocarlos en
                                   A */
        P1 = A;                /* envia al motor paso a paso */
        retraso();             /* espera 1 segundo */
        A = tempA;              /* recupera indice a A */
    }

}

void CSR(void)           /*funcion para rotacion en contra del
                           sentido de las manecillas del reloj */
{
    DPTR = SEC;
    for (A = 7; A >= 0; A--)
    {
        tempA = A;          /* respalda indice en A */
        A = DPTR[A];         /* obtiene patron de pasos a los 4 LSB
                               de A */
        A = A | (P1&0xF0);   /* retener los 4 MSB de P1 y colocarlos en
                               A */
        P1 = A;                /* envia al motor paso a paso */
        retraso();             /* espera 1 segundo */
        A = tempA;              /* recupera indice a A */
    }
}

void retraso (void)      /* retraso de 1 segundo */
{
    for (tempA = CIEN; tempA > 0; tempA--)
    {
        TH0 = 0x6C;
        TL0 = 0x70;
        TR0 = 1;
        while(TFO != 1);
        TFO = 0;
        TR0 = 0;
    }
}

void EX0RSI(void) interrupt 0
{
D = !D;                  /* complemento del bit de direccion */
}

```

Análisis

Este programa consta básicamente de las funciones SR() y CSR(), las cuales interactúan directamente con el motor paso a paso y lo dirigen para que gire en el sentido de las manecillas del reloj o en sentido contrario, respectivamente. Sin embargo, la única diferencia entre estas dos funciones es la secuencia de los patrones de pasos que se escriben al motor paso a paso. En el transcurso de la escritura de un patrón de pasos al motor paso a paso, llamamos a la función

`delay()` con el propósito de que el motor paso a paso tenga el tiempo suficiente para procesar cada patrón de pasos

PROBLEMAS

12.1 Vuelva a escribir el software para la pantalla LCD con el fin de presentar en forma continua el mensaje: “Bienvenido a la experiencia del microcontrolador 8051. Esperamos que disfrute su aventura de lectura”.

12.2 Imagine que el motor paso a paso se utiliza para girar la manija de una bóveda. Basado en una secuencia de números dados por el usuario, el motor paso a paso gira la manija la cantidad de pasos especificada en el sentido de las manecillas del reloj, luego en la dirección contraria, luego en el sentido de las manecillas del reloj, etc. Por ejemplo, si la secuencia es 8, 0, 1, 6, 0, 9, 9, entonces el motor paso a paso gira primero 8 pasos en el sentido de las manecillas del reloj, gira 0 pasos en sentido contrario, vuelve a girar 1 paso en el sentido de las manecillas del reloj, y así sucesivamente.

Escriba una función `boveda(int * sec, int tamsec)` para recibir como entrada una secuencia de números que han sido almacenados en un arreglo de tamaño `tamsec`, y después gire el motor paso a paso de acuerdo con la secuencia de entrada. Suponga la disponibilidad de las funciones `SR()` y `CSR()`.

12.3 El microcontrolador 8051 se utiliza por lo general en tarjetas inteligentes, las cuales no sólo cuentan con memoria incorporada sino que también poseen un microcontrolador como su cerebro, ello les permite ejecutar programas desde su interior. Parte de la información contenida en las tarjetas inteligentes es confidencial, así que hay necesidad de garantizar cierta forma de protección contra accesos no autorizados. Esto a menudo se realiza codificando la información. La codificación es el proceso de transformar información confidencial (normalmente llamada texto plano) en una forma no legible (llamada criptograma). Una vez que la información está en forma de criptograma, hasta alguien que quizás esté espiando no podrá entender su significado. La única manera de recuperar la información original es realizando el proceso inverso, llamado decodificación. El criptograma César es un método de codificación que data desde tiempos del Imperio Romano, y tiene el nombre de su inventor: Julio César. Aunque resulta simple ante los estándares actuales, y ya no se utiliza para proteger información, entender cómo funciona nos ayudará a darnos una idea básica sobre la codificación. En el capítulo 13 trataremos con más detalle la seguridad y la codificación.

Consideremos el alfabeto:

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Supongamos que desplazamos todas las letras tres posiciones hacia la izquierda para obtener un nuevo conjunto:

D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Entonces, dado un mensaje confidencial, podemos reemplazar cada carácter del primer conjunto de letras con un carácter del segundo conjunto. Por ejemplo, cada ‘A’ que veamos la reemplazamos con una ‘D’, cada ‘B’ por una ‘E’, cada ‘C’ por una ‘F’, y así sucesivamente.

Así que el mensaje:

VEAMONOS ESTA NOCHE

se codificaría a:

YHDPRQRV HVWD QRFKH

Escriba una función CodificacionCesar(char * plano, char * cripto) que reciba dos cadenas de mensajes como entrada (arreglos de caracteres), `plano` y `cripto`, respectivamente. Codifique el mensaje `plano` y almacene el mensaje codificado en `cripto`.

- 12.4** Con referencia al problema anterior, escriba la función correspondiente DecodificacionCesar(char * plano, char * cripto) para descifrar un mensaje almacenado en `cripto` y almacenarlo descifrado en `plano`.

Proyectos de ejemplo para los estudiantes

13.1 INTRODUCCIÓN

Con frecuencia los estudiantes se muestran impacientes por trabajar en proyectos basados en microcontroladores y en robots. Cuando se les asignan tareas para construir proyectos basados en el 8051, a menudo el maestro se queda asombrado por el gran entusiasmo y profundo interés que demuestran los alumnos. El lector estudiante ya ha visto algunos problemas de diseño e interfaz en los capítulos 11 y 12. Como extensión de esos ejemplos, en este capítulo trataremos de esbozar proyectos potenciales para estudiantes, los cuales ayudarán a enlazar todos los conceptos aprendidos en los capítulos previos.

Empezaremos con uno de los más simples de tales proyectos: un sistema de seguridad residencial basado en el 8051. Otros proyectos de nivel más avanzado son: un sistema simple de elevador, un juego Tres en raya, una calculadora basada en el 8051, un microrratón, un robot que juega fútbol soccer, y una aplicación de tarjeta inteligente.

13.2 SISTEMA DE SEGURIDAD RESIDENCIAL

Uno de los proyectos más simples para mejorar las habilidades de diseño de los estudiantes es un sistema de seguridad residencial basado en el 8051. Este proyecto es, en esencia, una demostración de cómo interactuar con los dispositivos de E/S a través de los puertos de E/S del 8051.

13.2.1 Descripción del proyecto

Una descripción de ejemplo para el proyecto de un sistema de seguridad residencial podría ser:

Diseñe un sistema de seguridad residencial, basado en el microcontrolador 8051, que tenga una memoria para código externa de 64Kbytes y una memoria para datos externa de 64Kbytes. Suponga que sólo tiene a su disposición EPROMs de 8K y RAMs de 8K.

Su sistema de seguridad deberá ser capaz de detectar cuándo un ladrón ha abierto o penetrado a cualquiera de estas secciones de su hogar:

1. reja
2. puerta frontal
3. puerta trasera
4. cualquier ventana

Cuando se haya abierto una de estas secciones, el sistema deberá encender una alarma y utilizar también una pantalla de siete segmentos para mostrar la sección correspondiente de la casa. Por ejemplo, si se abrió la reja, deberá mostrar un 1; si se abrió cualquier ventana, deberá mostrar un 4.

13.2.2 Especificaciones del sistema

Lo primero que debe hacer el estudiante al empezar con este proyecto es determinar los componentes requeridos para integrar el sistema. Con base en la descripción del proyecto, está claro que se necesita lo siguiente:

- 1 microcontrolador 8051
- 8 ROMs/EPROMs/EEPROMs de 8Kbytes
- 8 RAMs de 8Kbytes
- 4 sensores
- 1 alarma
- 1 pantalla de 7 segmentos

El requerimiento es que el sistema debe tener 64Kbytes de memoria externa para código, pero como sólo hay chips de ROM de 8Kbytes disponibles, tenemos que conectar ocho de ellos en cascada. Lo mismo se aplica para implementar la memoria de datos externa. Necesitamos detectar intrusiones en cuatro puntos distintos de la casa, como se plantea en la descripción del proyecto, por lo cual requerimos de cuatro sensores que pueden ser infrarrojos, de luz, o incluso de interruptor. Además, el proyecto requiere también del uso de una alarma y una pantalla de siete segmentos para indicar el punto de intrusión. Observe que la lista anterior no incluye los componentes necesarios para el circuito de reloj de cristal externo y el circuito de restablecimiento. Para conocer los detalles sobre tales componentes, consulte la figura 2-2 en el capítulo 2.

13.2.3 Diseño del sistema

El siguiente paso implica tomar decisiones de diseño para implementar un sistema que cumpla con los requerimientos y especificaciones descritos. Esto incluye decidir en dónde conectar qué componentes, para qué utilizar los puertos de E/S del 8051; culminar en un diagrama esquemático general que muestre el 8051 y la forma de conectar sus terminales (E/S, control y reloj) a la memoria externa y a diversos dispositivos de E/S, tales como los sensores, la alarma, y la pantalla de siete segmentos. Los estudiantes podrían utilizar también un decodificador 74LS47 de BCD a siete segmentos para minimizar las terminales de los puertos de E/S.

13.2.4 Diseño del software

Una vez decididas las conexiones de hardware, el estudiante puede proceder a diseñar y programar el software para controlar el 8051 y la forma en que debe interactuar con los dispositivos externos. A menudo, esto se hace con referencia al diagrama esquemático general. Algunas preguntas que

debe hacerse usted mismo, y por ende las decisiones de programación correspondientes que debe tomar, incluyen lo siguiente:

- ¿Cómo debo detectar los puntos de intrusión? ¿Mediante sondeo o interrupción?
- ¿Qué prioridad debo asignar a cada punto de intrusión?
- Cuando ocurra una intrusión, ¿qué debo hacer?
- ¿Mi sistema debe estar programado para detectar más de una intrusión a la vez?
- ¿Debo incluir la capacidad para restablecer o desactivar el sistema?
- ¿Debo permitir que el propietario de la casa personalice el sistema?
- ¿Tendría el sistema alguna seguridad? Por ejemplo, ¿cómo podría asegurarme de que sólo el propietario de la casa pueda personalizar o desactivar el sistema?

Con frecuencia, al formular tales preguntas debemos ponernos en los zapatos del verdadero usuario para poder comprender mejor la situación y el problema que tratamos de resolver con el sistema.

A continuación se muestra una parte de seudocódigo de ejemplo para este sistema de seguridad residencial:

Seudocódigo:

```

WHILE [1] DO BEGIN
    WHILE [sensor == false] DO
        [esperar]
    CASE [sensor] OF
        'reja': [pantalla = 0]
        'puertafrontal': [pantalla = 1]
        'puertatrasera': [pantalla = 2]
        'ventanas': [pantalla = 3]
    END_CASE
    [alarma = encendida]
    IF [restablecer == TRUE && contrasenia == TRUE]
        THEN [alarma = apagada]
END

```

El seudocódigo anterior muestra que el programa se encuentra en un ciclo infinito. Mientras no se active ningún sensor, el programa espera y no hace nada. Al dispararse un sensor, se comprueba para ver a qué punto de intrusión corresponde, después de lo cual se muestra un número del 0 al 3 en la pantalla de siete segmentos para indicar el punto de intrusión. Entonces, la alarma se enciende. El programa también permite un modo de restablecimiento, donde al teclear la contraseña correcta del usuario se restablece el sistema de seguridad y se apaga la alarma.

13.3 SISTEMA DE ELEVADOR

Un sistema de elevador es otro proyecto interesante que podrían realizar los estudiantes. En aras de la simplicidad, sólo consideraremos un sistema de elevador de tres pisos.

13.3.1 Descripción del proyecto

A continuación se muestra la descripción del proyecto:

Diseñe un sistema de elevador basado en el 8051 con soporte para tres pisos: planta baja, primer piso y segundo piso.

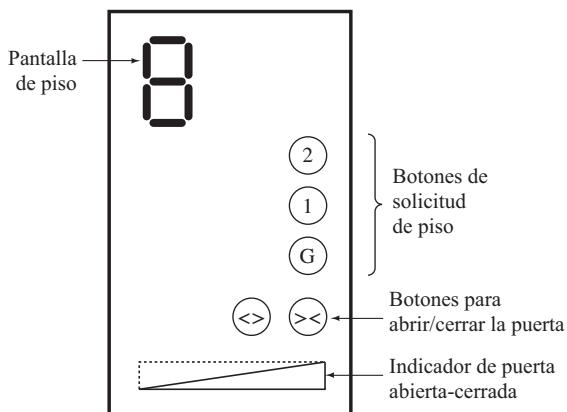


FIGURA 13-1
Dentro del elevador

Su sistema debe constar de dos partes:

1. Dentro del elevador

La figura 13-1 muestra el interior del elevador. El pasajero del elevador utiliza los botones de solicitud de piso, PB, 1 y 2, para solicitar el piso al que desea ir. Además, utiliza los botones abrir/cerrar puerta para abrir y cerrar la puerta, respectivamente. Mientras tanto, hay también un indicador correspondiente de puerta abierta/cerrada, el cual consiste básicamente en una serie de ocho LEDs ordenados de izquierda a derecha que destellan uno a la vez. Cuando el LED destellante es el de más a la izquierda, esto indica que la puerta está cerrada; cuando el LED destellante es el de más a la derecha, la puerta está completamente abierta.

2. Fuera del elevador

La figura 13-2 muestra el exterior del elevador. En cada uno de los pisos, hay indicadores de piso disponibles para señalar si el elevador se encuentra en ese piso en un momento dado. Los botones para llamar al elevador los utiliza el pasajero que espera fuera del mismo para llamar al elevador a su piso e indicar la dirección en que desea trasladarse, hacia arriba o hacia abajo.

13.3.2 Especificaciones del sistema

Una lectura minuciosa de la descripción del proyecto nos revela que necesitamos:

- 1 microcontrolador 8051
- 9 botones de interruptor (cinco para el interior, cuatro para el exterior)

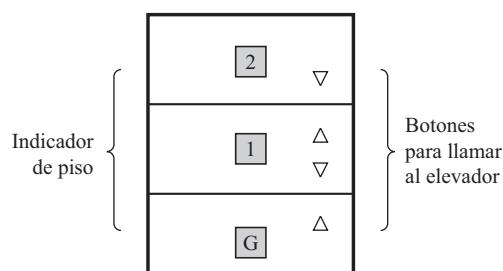


FIGURA 13-2
Fuera del elevador

- 11 LEDs (ocho para el indicador de puerta abierta/cerrada, tres para los indicadores de piso)
- 1 pantalla de 7 segmentos

Queda claro que este proyecto implica simplemente la interacción con interruptores y LEDs, además de una pantalla de siete segmentos. Por lo tanto, no difiere mucho del sistema de seguridad residencial de la sección anterior. Lo único diferente aquí es que estamos considerando un escenario totalmente distinto, y la secuencia de sucesos (abrir la puerta, el movimiento del elevador, etc.) es más compleja que la del sistema de seguridad residencial.

13.3.3 Diseño del sistema

En este caso, la etapa de diseño del sistema implica decidir qué terminales de puerto conectar a los botones de interruptor, a los LEDs, y a la pantalla de siete segmentos. Además, como el sistema de elevador no requiere el uso de memoria externa, los cuatro puertos del 8051 están disponibles para propósitos de E/S, por lo que tenemos mucha reserva.

13.3.4 Diseño del software

El programa resultante para este sistema de elevador sería más complejo ya que se deben considerar diversos escenarios, incluyendo:

- En este momento, ¿el elevador se encuentra en el mismo piso en que se oprimió el botón arriba/abajo?
- ¿Va el elevador en la misma dirección que la solicitud?
- Si no es así, pero el piso solicitado está más cerca, ¿se detendría y atendería esta solicitud primero, o la ignoraría y terminaría de atender una solicitud existente?
- ¿Atendería el elevador las solicitudes de pisos que estén más cerca primero o se movería constantemente en una dirección desde la solicitud del piso más bajo hasta la del más alto, antes de cambiar de dirección y repetir el proceso?
- Si no hay solicitud, ¿qué debería estar haciendo el elevador?

Existen diversas alternativas para encarar el problema; algunos estudiantes de cursos anteriores han ideado todo tipo de formas de tratar con esto. Un posible programa podría ser similar a las líneas de seudocódigo que se muestran a continuación:

Seudocódigo:

```
[empieza en planta baja]
WHILE [1] DO BEGIN
    WHILE [llamar elevador == FALSE] DO
        [esperar]
    IF [piso desde el que se llamó == piso actual]
        THEN BEGIN
            [puerta = abierta]
            WHILE [botón puerta abierta == TRUE] DO
                [esperar]
            [puerta = cerrada]
        END
    [dirección = arriba]
```

```

BEGIN
    WHILE [ [piso desde el que se llamó != piso actual] &&
           [piso solicitado != piso actual] ] DO BEGIN
        IF [piso actual == PISO_SUPERIOR]
            THEN [dirección = abajo]
        IF [piso actual == PLANTA_BAJA]
            THEN [dirección = arriba]
        IF [dirección == arriba]
            THEN [piso actual = piso actual + 1]
        ELSE [piso actual = piso actual - 1]
        [indicador de piso = piso actual]
        [pantalla de piso = piso actual]
    END
    BEGIN
        [puerta = abierta]
        WHILE [botón abrir puerta == TRUE] DO
            [esperar]
        [puerta = cerrada]
    END
END

```

Aquí, *piso actual* es el piso donde se encuentra el elevador en ese momento; *piso desde el que se llamó* es el piso donde un pasajero que espera fuera del elevador ha oprimido el botón para llamarlo; y *piso solicitado* se refiere al *piso solicitado* por el pasajero desde el interior del elevador al oprimir el botón de solicitud de piso.

El programa es un ciclo infinito, y al principio el elevador está en la planta baja esperando a que lo llamen las personas que necesiten el elevador en su piso. Cuando se llama al elevador, el programa revisa si éste se encuentra ya en ese piso, y de ser así abre su puerta, espera cierto tiempo y sólo cierra la puerta cuando ya no esté oprimido el botón *abrir puerta*. Después, el elevador avanza hacia arriba.

A medida que el elevador pasa de un piso al siguiente, el programa comprueba si hay llamadas al elevador en el piso actual o si algún pasajero dentro del elevador ha solicitado ir a ese piso. De no ser así, el elevador continúa avanzando hasta el siguiente piso. Si *piso actual* es el *piso superior* o el *inferior*, se invierte la dirección del recorrido. Al mismo tiempo se actualizan los indicadores de piso colocados fuera del elevador, así como las pantallas de piso instaladas dentro del elevador, para que correspondan con *piso actual*.

Si *piso actual* es el *piso desde el que se llamó* o el *piso solicitado*, entonces el elevador abre su puerta y espera a que el pasajero suelte el botón *abrir puerta* antes de cerrar las puertas.

13.4 TRES EN RAYA

Un proyecto más desafiante para los estudiantes sería el desarrollo de juegos, y la correspondiente inteligencia artificial (IA) que permita al programa competir con las personas y que éstas compitan entre sí. Uno proyecto de este tipo es el juego denominado Tres en raya basado en el 8051.

TABLA 13-1
Modos de juego

MODO	P1.3 (M1)	P1.4 (M0)
0 - Humano contra humano	0	0
1 - Humano contra IA	0	1
2 - IA contra humano	1	0

13.4.1 Descripción del proyecto

En primer lugar, debemos bosquejar la descripción del proyecto y las reglas del juego:

Diseñe un juego Tres en raya basado en el 8051 que conste de dos partes:

1. Entorno del juego Tres en raya

Al principio del juego, el(es) jugador(es) selecciona(n) uno de tres modos de juego: Humano contra humano, Humano contra IA, e IA contra humano. El modo de juego se introduce a través de dos terminales de puerto: P1.3 y P1.4; esto se muestra con más detalle en la tabla 13-1.

Debe haber un panel Tres en raya de 3×3 (vea la figura 13-3) para indicar cuál cuadro ha sido seleccionado ya por un jugador. Un cuadro seleccionado podría destacarse en color amarillo (jugador 1) o rojo (jugador 2). Cuando se seleccione una fila vertical, horizontal o diagonal de tres cuadros con el mismo color, el jugador correspondiente gana el juego. En caso contrario, si están seleccionadas todas las casillas pero no hay una fila con los tres cuadros del mismo color, el juego termina en empate. Cada uno de los nueve cuadros consiste básicamente en un LED amarillo y rojo; todos estos LEDs están conectados al 8051 mediante los puertos 0, 1 y 2, como indica la figura 13-14.

Cada jugador toma turnos para interactuar con un teclado numérico de matriz de 3×3 y seleccionar un cuadro específico durante su turno. Las conexiones del 8051 al teclado numérico se realizan mediante el puerto 3. El jugador 1 indica el fin de su turno al activar P0.4 (E1), lo cual enciende un LED de fin de turno. De la misma forma, el jugador 2 termina su turno al activar su LED de fin de turno conectado a P0.3 (E2).

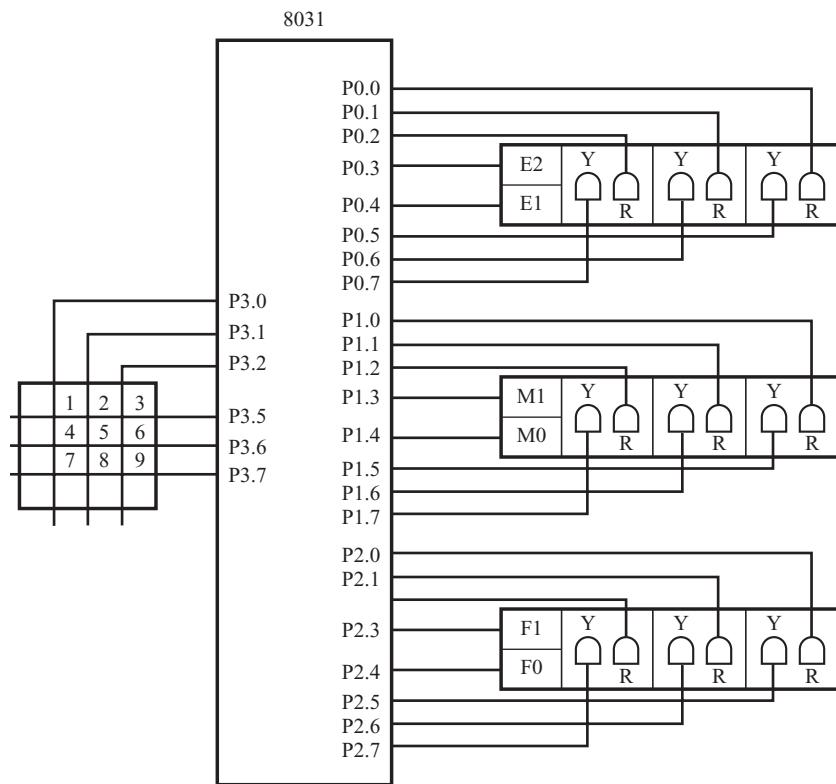
Los resultados se indicarán al final del juego en los LEDs conectados a P2.3 y P2.4, cuyos detalles podemos encontrar en la tabla 13-2.

2. Componente de la inteligencia artificial (IA) para el juego Tres en raya

El componente de IA debe ser capaz de jugar ya sea como el jugador 1 o como el jugador 2, y debe tratar de ganar lo mejor que pueda al jugador humano.

FIGURA 13-3
Panel del juego
Tres en raya

1	2	3
4	5	6
7	8	9

**FIGURA 13-4**

Conexiones del 8051 al panel del juego Tres en raya

13.4.2 Especificaciones del sistema

El sistema del juego Tres en raya consta principalmente de LEDs y de un teclado numérico de 3×3 para interactuar con un jugador humano. A continuación presentamos una lista detallada de los componentes necesarios:

- 1 microcontrolador 8051
- 1 teclado numérico (3×3)
- 11 LEDs amarillos
- 11 LEDs rojos

TABLA 13-2

Resultados finales indicados en los LEDs conectados a P1.3 y P1.4

MODO	P2.3 (F1)	P2.4 (F0)
Jugador 1 (amarillo) es el ganador	1	0
Jugador 2 (rojo) es el ganador	0	1
Empate	1	1

Como podemos ver, en realidad no es mucho lo necesario para implementar el hardware del juego Tres en raya. Lo más importante es la organización lógica y clara de los LEDs para evitar confusiones por parte del jugador humano.

13.4.3 Diseño del software

Al escribir el componente del entorno del juego, surgen las siguientes preguntas:

- ¿Cómo podemos determinar el fin de turno de un jugador?
- ¿Cómo debemos verificar y, por consiguiente, ejecutar un modo de juego?
- ¿Cómo podemos determinar si el ganador del juego fue un jugador o si el juego terminó en empate?

Por otra parte, para el componente de IA del proyecto:

- ¿Cómo podemos determinar la mejor posición para un cuadro?
- ¿Cómo podemos impedir que el contrincante pueda completar una fila de tres en uno?
- Existen algunas técnicas más avanzadas y sutiles para engañar al contrincante y hacer que seleccione un cuadro en nuestra ventaja?
- La estrategia de IA deberá ser fija o adaptable a cambios con base en la situación en curso?

A continuación presentamos el seudocódigo para un posible entorno del juego Tres en raya: Seudocódigo:

```

WHILE [entrada == FALSE]
    [esperar]
    [jugador actual = amarillo]
CASE [modo] OF
    '0': BEGIN
        WHILE (1) DO BEGIN
            [verifica el color del jugador actual]
            WHILE DO [entrada del jugador == FALSE]
                [esperar]
                [cuadro seleccionado == color del jugador]
            IF [tres en una fila == TRUE]
                THEN BEGIN
                    [presenta al ganador]
                    [fin del juego]
                END
            IF [todos los cuadros están seleccionados == TRUE]
                THEN BEGIN
                    [presenta un empate]
                    [fin del juego]
                END
            END
        END
    END
    '1': BEGIN
        [jugador actual = humano]
        WHILE [1] DO BEGIN
            [verifica el color del jugador actual]
        END
    END

```

```

        IF [jugador actual == IA]
            THEN [llamar a IA]
        WHILE DO [entrada == FALSE]
            [esperar]
            [cuadro seleccionado == color del jugador]
        IF [tres en una fila == TRUE]
            THEN BEGIN
                [presenta al ganador]
                [fin del juego]
            END
        IF [todos los cuadros están seleccionados == TRUE]
            THEN BEGIN
                [presenta un empate]
                [fin del juego]
            END
            [jugador actual = el otro jugador]
        END
    END

'2': BEGIN
    [jugador actual = IA]
    WHILE (1) DO BEGIN
        [verifica el color del jugador actual]
        IF [jugador actual == IA]
            THEN [llamar a IA]
        WHILE DO [entrada == FALSE]
            [esperar]
            [cuadro seleccionado == color del jugador]
        IF [tres en una fila == TRUE]
            THEN BEGIN
                [presenta al ganador]
                [fin del juego]
            END
        IF [todos los cuadros están seleccionados == TRUE]
            THEN BEGIN
                [presenta un empate]
                [fin del juego]
            END
            [jugador actual = el otro jugador]
        END
    END
END_CASE

```

Este programa espera a que el jugador proporcione una entrada para empezar el juego Tres en raya. Una vez que se ha detectado una entrada, el programa verifica el modo de juego seleccionado. Si es el modo 0, “Humano contra humano”, el programa inicia un ciclo infinito y verifica quién es el jugador actual, luego espera a que el jugador actual actúe en su turno. Cuando se ha realizado una entrada, lo cual significa que se ha seleccionado un cuadro específico, el programa

cambia el color de ese cuadro al color correspondiente al jugador actual. Después que el jugador actual ha tomado su turno, el programa verifica si existen tres cuadros del mismo color en una fila vertical, horizontal o diagonal. En tal situación, se presenta al ganador y el juego termina. En caso contrario, el programa verifica si todos los cuadros se han seleccionado, ello significa que el juego termina en empate y también se presenta este resultado.

De otra manera, si fue seleccionado el modo 1, “Humano contra IA”, el jugador que empieza es el jugador humano. El programa inicia un ciclo infinito y verifica quién es el jugador actual. Si el jugador actual es la IA, el programa llamará a la función de la IA para calcular el mejor cuadro a escoger. Cuando se ha seleccionado un cuadro, ya sea por la IA o por el jugador humano, el cuadro se presenta con el color del jugador. El programa procede a verificar si existen tres cuadros del mismo color en la misma fila, con lo que detecta a un ganador, presenta el resultado, y termina el juego.

El modo 2, “IA contra humano”, es parecido al modo 1, con la diferencia de que el primer jugador que empieza es la IA.

A continuación presentamos el seudocódigo para una posible IA del juego Tres en raya:

Seudocódigo:

```

IF [contrincante tiene dos en una fila == TRUE]
    THEN [interferir con la oportunidad del contrincante]
ELSE BEGIN
    [calcular la posición más favorable para un cuadro]
    WHILE [cuadro == ocupado] DO
        [calcular la siguiente posición más favorable
         para un cuadro]
    [seleccionar cuadro]
END

```

La IA parece muy simple. La prioridad principal es interferir con la oportunidad del contrincante de obtener tres cuadros en una sola fila, por lo que si se detectan dos cuadros en una sola fila, la alternativa de la IA sería seleccionar el tercer cuadro para poder interferir y evitar que gane el contrincante. Después, si no hay tres cuadros en una sola fila, significa que la IA está segura y tiene más flexibilidad para escoger un cuadro basado en sus cálculos. Si el cuadro más favorable está ocupado, la IA calcula el siguiente cuadro más favorable, y así sucesivamente hasta que encuentra un cuadro libre al cual selecciona de inmediato. Aunque parezca simple, la verdadera tarea de programación de la IA radica en los cálculos necesarios para obtener la posición más favorable para un cuadro. Esto es lo que distingue a la mejor IA del resto.

La separación del software en dos partes ayuda en la tarea del programador. También proporciona un ejercicio interesante sobre la **programación modular**, donde un programa se divide en módulos y cada módulo está escrito por un programador distinto. Una vez terminada la programación, los módulos se integran en un solo programa completo. La programación modular permite que diversos componentes de un programa se puedan desarrollar en común y es más adecuada para los proyectos de programación grandes y complejos.

13.5 CALCULADORA

Las calculadoras son tan comunes en la actualidad que a veces podríamos olvidar que no siempre existieron. Como ejercicio adicional en el diseño e interfaz de sistemas basados en el 8051, una calculadora basada en el 8051 sería un buen proyecto. Por supuesto que para mantenerlo simple, sólo consideraremos las operaciones elementales.

13.5.1 Descripción del proyecto

La descripción del proyecto es como sigue:

Diseñe una calculadora basada en un 8051 de 8 bits que realice operaciones aritméticas simples tales como:

- Suma
- Resta
- Multiplicación
- División
- AND lógico
- OR lógico
- NOT lógico
- XOR lógico
- Cuadrado
- Raíz cuadrada
- Elevación al cubo
- Raíz cúbica
- Inversión
- Exponenciación
- Factorización
- Operación módulo
- Porcentaje
- Cálculo de pi
- Operaciones básicas de memoria

Su calculadora deberá soportar respuestas de 16 bits, las cuales tendrán que presentarse ya sea en LCD o en pantallas de 7 segmentos. La entrada del usuario será a través de un teclado numérico de matriz cuya distribución habrá que diseñar.

13.5.2 Especificaciones del sistema

La calculadora requiere un teclado numérico de matriz para las entradas y una pantalla LCD o de 7 segmentos para mostrar los resultados. Además, también deberá tener un interruptor con botón para encender la calculadora y el LED correspondiente para indicar que está encendida. Por lo tanto, los componentes necesarios para este sistema son:

- 1 microcontrolador 8051
- 1 teclado numérico de matriz
- 1 pantalla LCD o 5 pantallas de 7 segmentos
- 1 LED
- 1 interruptor con botón

13.5.3 Diseño del software

El software para controlar este sistema será parecido al de sistemas operativos tales como Windows o MS-DOS, donde todas las operaciones aritméticas y lógicas, además de las interacciones con los dispositivos de entrada y salida, las controla el programa. Al escribir este programa deberemos considerar lo siguiente:

- ¿Cómo podemos determinar si la tecla oprimida por el usuario es un número o una operación?
- ¿Cómo debemos manejar la precedencia de los operadores?
- ¿Debemos permitir operaciones acumulativas?
- ¿La calculadora deberá operar con números que tengan o no tengan signo?
- ¿La calculadora deberá soportar puntos decimales?
- ¿Cómo podemos implementar cada tipo de operación aritmética o lógica?
- ¿Cómo utilizaríamos tablas de búsqueda?

Una posible calculadora basada en el 8051 podría operar a partir del siguiente seudocódigo:
Seudocódigo:

```

WHILE (1) DO BEGIN
    WHILE [tecla oprimida == FALSE]
        [esperar]
    CASE [tecla oprimida] OF
        'numérica': BEGIN
            [almacena valor numérico]
            [convertir a formato para pantalla de 7
            segmentos]
            [enviar a pantalla de 7 segmentos]
        END
        'operación': BEGIN
            [determinar operación]
            [almacena operación]
        END
        'operación de memoria': BEGIN
            [determinar operación]
            [realiza operación]
        END
        'igual': BEGIN
            [recupera valores numéricos y operaciones
            almacenadas]
            [realiza operaciones]
            [convertir resultado a formato para
            pantalla de 7 segmentos]
            [enviar a pantalla de 7 segmentos]
        END
    END_CASE
END

```

El programa espera indefinidamente a que el usuario oprima una tecla. Una vez detectada una tecla, se verifica para saber si corresponde a un valor numérico, a una operación aritmética o lógica, a una operación de memoria, o a la tecla de igual. Cuando se detectan las primeras dos teclas, el programa verifica específicamente qué valor u operación es y los almacena. Un valor numérico también puede convertirse al formato correspondiente y enviarse a la pantalla 7 segmentos. Si se pide una operación de memoria, el programa determina inmediatamente cuál fue la memoria de operación solicitada y le da servicio. Finalmente, cuando se oprime la tecla de igual, significa que el usuario está listo para ver los resultados, así que se recuperan todos los valores numéricos y operaciones previamente almacenadas y se llevan a cabo las operaciones necesarias de acuerdo con su precedencia. El resultado se convierte al formato correspondiente y se envía a la pantalla de 7 segmentos.

13.6 MICRORRATÓN

Cuando conectamos un 8051 (el cerebro) con algunos sensores (los ojos, oídos, etc.) y algunos motores (manos y pies), obtenemos un robot básico. Uno de estos robots es el **microrratón**, que es el nombre asignado para identificar un robot que imita a un ratón e intenta resolver un laberinto. La solución a un laberinto requiere que el microrratón empiece desde un extremo y aprenda lentamente, la mayor parte de las veces por ensayo y error además de memorización, el camino correcto para llegar hasta el otro extremo del laberinto. Una vez encontrado el camino correcto y llegado al otro extremo del laberinto, el microrratón deberá ser capaz de seguir el mismo camino la siguiente vez que se le coloque en el laberinto. Los concursos de microrratones son muy populares y se llevan a cabo en muchas partes del mundo. En esta sección consideraremos un proyecto de un microrratón.

13.6.1 Descripción del proyecto

Como es costumbre, describiremos los detalles del proyecto:

Diseñe un microrratón basado en el 8051 que tenga la capacidad de moverse hacia delante, voltear a la izquierda y a la derecha, además de dar vuelta en U. También deberá ser capaz de resolver un laberinto lo más pronto posible y almacenar el camino correcto en su memoria, una vez que lo haya encontrado.

13.6.2 Especificaciones del sistema

Para poder moverse hacia delante y voltear, el microrratón debe tener dos motores conectados, respectivamente, a sus llantas izquierda y derecha. También debe ser capaz de saber si ha llegado a un callejón sin salida o si hay una pared a su izquierda o a su derecha, si desea voltear en esa dirección. Para esto necesitará de algunos sensores. Dos sensores infrarrojos serán suficientes. También podríamos incluir algunos LEDs para indicar lo que el microrratón haga en un momento dado; es decir, si está desplazándose hacia delante, volteando a la izquierda, a la derecha, o dando una vuelta en U. El microrratón también deberá contar con un buen tamaño de memoria externa para memorizar los caminos recorridos previamente. Por lo tanto, los componentes necesarios para implementar el microrratón son:

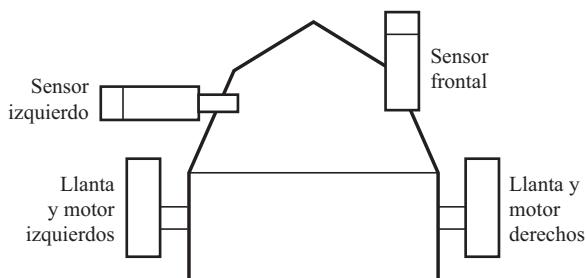
- 1 microcontrolador 8051
- 2 motores
- 2 sensores infrarrojos
- 1 RAM de 64Kbytes
- 2 LEDs

13.6.3 Diseño del sistema

La parte principal de la etapa de diseño del sistema es la decisión sobre dónde colocar los sensores infrarrojos. Por lo general, el sensor infrarrojo está formado a partir de dos componentes posicionados uno al lado del otro: un LED infrarrojo para emitir luz infrarroja y un detector infrarrojo (fototransistor) para detectar la presencia de luz infrarroja. Si hay una pared en la trayectoria del sensor, la luz infrarroja emitida por el LED infrarrojo será reflejada y subsequentemente detectada por el detector infrarrojo. De lo contrario, si no existe alguna pared, no habrá reflejo, y por lo tanto no se detectará luz infrarroja. Así, el microrratón sabrá si hay una pared en cierta dirección.

FIGURA 13–5

Vista superior de un microrratón



Debemos colocar uno de los sensores al frente del microrratón para detectar si hay alguna pared que impida su movimiento hacia delante. El otro sensor deberá colocarse ya sea en el lado izquierdo o en el lado derecho del microrratón. Supongamos que decidimos colocarlo en el lado izquierdo, como se muestra en la figura 13–5.

La siguiente pregunta es: si tenemos sólo un sensor en un lado, ¿cómo podemos detectar la presencia de paredes en el otro lado? Respuesta: no tenemos que hacerlo directamente. Consideremos esto con algunos ejemplos de situaciones como las que presentan los ejemplos 13.1, 13.2 y 13.3.

EJEMPLO 13.1 El microrratón llega a una esquina izquierda

¿Cómo sabrá el microrratón que ha llegado a una esquina izquierda?

Solución

El sensor frontal detecta una pared. El sensor izquierdo no detecta una pared.

Análisis

En este caso, el sensor frontal del microrratón detecta una pared que impide su movimiento hacia delante. Mientras tanto, el sensor izquierdo no detecta una pared. Por lo tanto, sabrá que ha llegado a una esquina donde no puede moverse hacia delante, pero sí puede voltear a la izquierda.

Por supuesto, es posible que el microrratón haya llegado a un cruce en forma de T, en cuyo caso pudiera haber volteado a la izquierda o a la derecha. No obstante, debido a que es libre de voltear a la izquierda, el microrratón debe intentar tomar primero el camino hacia la izquierda y regresar más tarde para intentar por el camino a la derecha.

EJEMPLO 13.2 El microrratón llega a una esquina derecha

¿Cómo sabrá el microrratón que ha llegado a una esquina derecha?

Solución

Tanto el sensor frontal como el izquierdo detectan paredes.

Análisis

Debido a que los dos sensores detectan la presencia de paredes, el microrratón no puede moverse hacia delante ni voltear a la izquierda. Por lo tanto, adivina que no debe haber ninguna pared a su derecha de modo que voltea hacia la derecha.

Sin embargo, es posible que el microrratón haya llegado a un callejón sin salida, en cuyo caso habría una pared a su lado derecho. Consideraremos esto en el siguiente ejemplo.

EJEMPLO El microrratón llega a un callejón sin salida

13.3 ¿Cómo sabrá el microrratón que ha llegado a un callejón sin salida?

Solución

Con anterioridad, tanto el sensor frontal como el izquierdo detectaron paredes. Después que el microrratón volteó a la derecha, el sensor frontal detecta de nuevo una pared.

Análisis

Antes, tanto el sensor frontal como el izquierdo detectaron paredes, así que el microrratón volteó hacia la derecha. Sin embargo, después de voltear, su sensor frontal todavía detecta una pared. Esto puede ocurrir sólo si está en un callejón sin salida. En este caso, deberá voltear a la derecha una vez más, para que efectivamente haya volteado dos veces a la derecha. Esto es esencialmente una vuelta en U. Ahora podrá proceder hacia delante para volver a encaminar sus pasos y salir del callejón sin salida.

13.6.4 Diseño del software

El software necesario para implementar el microrratón es un tanto complicado y requiere de muy buen razonamiento. Algunos problemas incluyen:

- ¿Cómo puede detectar el microrratón las esquinas izquierdas, derechas, y los callejones sin salida?
- ¿Cómo podemos hacer que el microrratón voltee a la izquierda o a la derecha?
- ¿En qué orden debe probar el microrratón todos los caminos posibles? ¿Deberán probarse primero las bifurcaciones más cercanas o las más lejanas?
- ¿Cómo debe recordar el microrratón los caminos transitados previamente?

El seudocódigo para el programa del microrratón es como sigue:

Seudocódigo:

```

WHILE [1] DO BEGIN
    [motor izquierdo = motor derecho = hacia delante]
    IF [sensor frontal == pared]
        IF [sensor izquierdo == no hay pared]
            THEN BEGIN
                [motor izquierdo = hacia atrás]
                [motor derecho = hacia delante]
            END
        ELSE BEGIN
            [motor izquierdo = hacia delante]
            [motor derecho = hacia atrás]
            IF [sensor frontal == pared]
                THEN BEGIN
                    [motor izquierdo = hacia delante]
                    [motor derecho = hacia atrás]
                    IF [sensor izquierdo == pared]
                        THEN BEGIN
                            [motor izquierdo = hacia delante]
                            [motor derecho = hacia atrás]
                        END
                    END
                END
            END
        END
    END
    [memoriza camino]
END

```

En los ejemplos 13.1, 13.2 y 13.3 hemos analizado el problema de detección de las esquinas y de callejones sin salida. ¿Pero qué hay acerca de voltear a la izquierda y a la derecha? Esto puede

resolverse fácilmente. Si el microrratón se moverá hacia delante, ambos motores deberán moverse hacia delante. Si debe moverse a la izquierda, el motor izquierdo deberá ir hacia atrás mientras el motor derecho va hacia delante. De la misma forma, para voltear a la derecha, el motor izquierdo debe moverse hacia delante mientras el motor derecho se mueve hacia atrás.

El seudocódigo anterior muestra que el programa entra a un ciclo infinito, y al principio mueve el microrratón hacia delante. Si detecta una pared enfrente del microrratón, éste procede a detectar si hay una pared a su izquierda. Si no, voltea a la izquierda. De otra manera, volteá a la derecha. Después de voltear a la derecha, detecta si hay una pared enfrente de él. Si la hay, entonces ha llegado a un callejón sin salida; así que vuelve a voltear a la derecha. Ya que ha dado las vueltas necesarias, puede proceder hacia delante. Durante todo este proceso, el microrratón también memoriza los caminos transitados previamente.

El reto principal en la programación del microcontrolador reside en una sola línea: [memoriza camino]. ¿Cómo deberá el microrratón memorizar los caminos para asegurarse de que la siguiente vez que llegue a un cruce en forma de T o a un cruce de caminos no volverá a elegir los caminos con callejones sin salida? Esto será el tema de un razonamiento profundo y de ensayos de programación eficaces.

13.7 UN ROBOT QUE JUEGA FÚTBOL SOCCER

En la sección anterior vimos cómo puede usarse un 8051 como cerebro para implementar un robot tipo ratón, el cual le permite orientarse a través de un laberinto y, por ende, resolverlo. De hecho, el 8051 puede utilizarse para controlar cualquier robot, y se encuentra entre los microcontroladores más populares utilizados por los fanáticos de robots como cerebros. En todo el mundo se llevan a cabo interesantes torneos con robots y los participantes tienen que realizar todo tipo de tareas, desde jugar deportes simples hasta desarrollar labores más complicadas como subir escaleras. En esta sección veremos cómo se utiliza un 8051 para controlar un robot que juega fútbol soccer.

13.7.1 Descripción del proyecto

Diseñe un robot basado en el microcontrolador 8051 para que participe en un torneo de robosoccer que dura 10 minutos. Durante cada partido, un robot de color azul y otro de color rojo se colocan en una arena con alfombra color gris, con pelotas azul y rojo de 15mm de diámetro colocadas al azar. Hay una meta común en cada extremo de la arena, iluminada por luces brillantes. Cada robot debe detectar la presencia de pelotas, identificar su color, y recolectar las de su propio color. No obstante, sólo debe tomar una pelota a la vez. Al recolectar una pelota, debe buscar la meta común y patearla para que entre en la meta.

13.7.2 Especificaciones del sistema

De manera similar al microrratón, el robot debe tener ruedas unidas a motores. También necesita algunos sensores de luz (pares de LEDs y fototransistores) para detectar la meta y los colores de las pelotas, además de algunos sensores de contacto del tipo interruptor para detectar paredes o al robot oponente. Como el robot debe funcionar por su cuenta, debe tener memoria no volátil. En este caso, una EPROM sería una elección excelente, ya que permitiría sobreescribir el programa a voluntad, lo cual a menudo es muy conveniente. Entonces, los componentes requeridos son:

- 1 microcontrolador 8051
- 2 motores
- 4 ruedas

- 2 sensores de luz (par de LEDs y fototransistor)
- 4 interruptores
- 1 memoria EPROM

13.7.3 Diseño del sistema

Al diseñar este robot, es importante considerar en dónde se van a colocar los sensores de luz y de contacto. Por ejemplo, los sensores de luz para detectar los colores de las pelotas deben estar muy aislados de todo lo que haya a su alrededor en el exterior, de no ser así, la detección de color se verá afectada. Esto resulta especialmente cierto cuando el robot está muy cerca de la meta. El brillo de las luces de la meta interferiría con la luz proveniente del LED del sensor, y provocaría confusión.

Otra cuestión de diseño es el mecanismo de pateo. Existen diversos métodos para patear la pelota, y ver cómo a los distintos estudiantes se les ocurren ideas diferentes es parte de la diversión. Algunos utilizan motores para balancear un eje y hacer contacto con la pelota, algo similar a la manera en que los jugadores de béisbol pegan con sus bates. No obstante, la limitación de esta técnica es que la patada sería menos poderosa que si se utilizara una energía elástica. Por ende, algunos prefieren unir el eje a una pieza de liga de hule y utilizar un motor para estirarla. Después, cuando el robot está listo para patear, el motor avanza a una posición tal que se libera la liga de hule, obligando al eje a balancearse hacia la pelota y mandarla fuera del contacto con el robot.

13.7.4 Diseño del software

Algunas cuestiones relacionadas con el software del robot son:

- ¿Cómo detectaría el robot las paredes y los robots oponentes?
- ¿Cómo debe dar vuelta el robot?
- Cuando el robot llegue a una esquina, ¿debe voltear a la derecha o a la izquierda? ¿Siempre debe voltear en una dirección o puede alternar, o voltear al azar?
- ¿Cómo debe implementarse el azar?
- ¿Cómo debe detectar el robot los colores de las pelotas?
- ¿Cómo debe detectar el robot la dirección de la meta?
- ¿Cómo debe patear el robot?
- ¿Qué estrategia debe usar el robot contra su oponente? ¿Debe ser más ofensivo, defensivo, o cambiar de estrategia a mitad del partido?
- ¿Cómo debe desplazarse el robot alrededor de la arena? ¿Debe moverse al azar o con cierto patrón? En el último caso, ¿cuál sería el mejor patrón de prueba para asegurar que cubra el mayor terreno posible?

A continuación se muestra el seudocódigo de ejemplo para el robot de color azul:

Seudocódigo:

```

WHILE [1] DO BEGIN
    [camina en patrón espiral]
    IF [sensor de contacto == TRUE]
        THEN BEGIN
            [reversa]
            [voltea a la izquierda o a la derecha]
        END
    END

```

```

IF [sensor de luz == AZUL]
    IF [pelotas recolectadas < 1]
        THEN BEGIN
            [recolecta pelota]
            [detecta luces de la meta]
            IF [se detectó la meta == TRUE]
                THEN BEGIN
                    [alto]
                    [patear]
                [END]
            END
        IF [sensor de luz == ROJO]
            IF [tiempo restante == 5 minutos]
                IF [pelotas recolectadas < 1]
                    THEN BEGIN
                        [recolecta pelota]
                        [avanza hacia delante]
                        IF [se detectó pared]
                            [soltar pelota]
                    END
                END
            END
        END
    END

```

El programa dirige al robot para que camine en un patrón espiral, que se cree es un buen patrón para permitir al robot cubrir mucho terreno. Cuando un sensor de contacto devuelve TRUE, esto indica que se ha topado con una pared o con un oponente, en cuyo caso el robot invierte su camino y gira a la izquierda o a la derecha para alejarse del obstáculo, antes de continuar con su movimiento hacia delante. Si el sensor de luz detecta una pelota AZUL, que es de su mismo color, comprueba para ver si ya tiene una pelota dentro de su espacio de almacenamiento. En caso contrario, recolecta la pelota y de inmediato va en busca de la meta. Una vez detectada la meta, el robot se detiene y patea la pelota hacia la meta. Si el sensor de luz detecta una pelota ROJA, el robot verifica si sólo faltan 5 minutos de partido. Si es así, cambia a una estrategia más ofensiva recolectando una pelota del oponente y avanzando hacia delante, en espera de toparse con una pared tarde o temprano. Si lo hace, suelta la pelota. Por lo general, es mucho más difícil recolectar las pelotas pegadas a la pared. Esta sería la estrategia del robot AZUL contra su oponente ROJO.

13.8 UNA APLICACIÓN DE TARJETA INTELIGENTE

Los microcontroladores se están usando cada vez más en **tarjetas inteligentes**, las cuales son tarjetas que integran tanto memoria como un microcontrolador dentro de sus límites físicos. Imagine tener una computadora tan pequeña que sea posible protegerla con un estuche de plástico y acomodarla en nuestra cartera. Esa es una tarjeta inteligente. Las tarjetas inteligentes se utilizan como tarjetas de identidad nacional, licencias de conducir, información de pasaporte, y dinero electrónico. Como estas tarjetas inteligentes contienen información delicada, tanto personal como financiera, y se otorgan al público, es imprescindible que la información que contengan esté protegida contra el acceso no autorizado. La siguiente subsección describe en forma breve conceptos de seguridad básicos para comprender mejor cómo proteger la información almacenada dentro de las tarjetas inteligentes.

13.8.1 Conceptos básicos de seguridad

Por lo general, la seguridad de la información requiere que se garanticen la **confidencialidad**, la **autenticación** y la **integridad**. La confidencialidad asegura que solamente las partes autorizadas puedan acceder a la información y verla. La autenticación comprueba la identidad de una parte, mientras que la integridad permite determinar si se modificó la información sin permiso.

Todo esto es posible mediante el uso de técnicas criptográficas tales como el **encriptado**, las **firmas digitales**, y la **autenticación de mensajes**. El encriptado es el proceso de transformar cierta información confidencial en un formato ininteligible para protegerla del acceso no autorizado. Los métodos de encriptado estándar son el Data Encryption Standard (DES) y su variante, el DES triple. A propósito, el DES triple también se utiliza en muchas tarjetas de cajeros automáticos (ATM, por sus siglas en inglés) para proteger los Números de Identificación Personal (NIP).

En tanto, una firma digital es similar en concepto a su contraparte manuscrita. No obstante, en vez de usar una firma escrita o una huella digital, una firma digital utiliza ciertos bits de datos secretos para demostrar la identidad de una parte. La autenticación de mensajes se lleva a cabo mediante el uso de ciertos bits secretos de datos para realizar una operación con el mensaje y obtener lo que se conoce como **código de autenticación de mensaje** (MAC, por sus siglas en inglés). Esto es similar a la suma de comprobación y se utiliza para verificar si se ha modificado el mensaje. Como el MAC sólo puede obtenerse cuando se conocen los datos secretos, esto asegura que las partes no autorizadas no puedan modificar el mensaje sin ser detectadas, ya que el MAC sería distinto.

13.8.2 Descripción del proyecto

Recuerde que en la sección de problemas del capítulo anterior vimos un método de encriptado simple, llamado cifrado del César. Ahora vamos a considerar una versión un poco más complicada y general: el **cifrado polialfabético**. Aún así, este método de encriptado no se utiliza hoy en día para proteger la información, ya que es muy fácil de burlar mediante el uso de programas de software. No obstante, para facilitar la ilustración, lo consideraremos aquí, y veremos algunas generalidades sobre los métodos seguros que se utilizan actualmente en la sección de problemas, al final de este capítulo.

El cifrado polialfabético es similar al cifrado del César, sólo que en vez de desplazar el conjunto de letras del texto original tres posiciones fijas a la izquierda, se desplaza a la izquierda un número variable de posiciones. Por ejemplo, al desplazarse 10 posiciones a la izquierda, el conjunto del alfabeto de texto simple se sustituye por un conjunto de alfabeto de criptograma, como se muestra a continuación:

Alfabetos de texto simple ABCDEFGHIJKLMNOPQRSTUVWXYZ

Alfabetos del criptograma KLMNOPQRSTUVWXYZABCDEFGHIJ

por lo tanto, el mensaje ESTO ES SECRETO se encripta como OCDY OC COMBODY.

Diseñe una tarjeta inteligente basada en el microcontrolador 8051 que encripte mediante el uso del cifrado polialfabético los mensajes con terminación nula almacenados en su EPROM, empezando en la ubicación 1234H.

13.8.3 Especificaciones del sistema

El sistema de tarjeta inteligente en realidad es bastante simple, ya que sólo consiste en el 8051 y una EPROM externa o integrada en el microcontrolador, en cuyo caso estaríamos usando un 8071

en vez de un 8051. Podríamos optar por agregar algunas características, por ejemplo una pantalla LCD para mostrar los mensajes de texto simple y de texto cifrado. Los componentes son:

- 1 microcontrolador 8051
- 1 EPROM
- 1 LCD

13.8.4 Diseño del software

Al escribir el software de cifrado, algunas de las cuestiones a tratar podrían ser:

- ¿Cómo debe determinarse el número de desplazamientos? Al azar, o el usuario debe tener la capacidad de especificarlo?
- ¿Hay alguna cantidad de desplazamientos que no deba permitirse? Por ejemplo, un desplazamiento de 0, 26, 52, o cualquier múltiplo de 26 posiciones haría que el criptograma fuera exactamente igual al criptograma base, lo cual no produciría encriptado.

Seudocódigo:

```

BEGIN
    [determina cantidad de desplazamientos]
    [ubicación = 1234H]
REPEAT
    BEGIN
        [obtiene carácter de ubicación]
        [encripta carácter]
        [almacena de vuelta en ubicación]
    END
    UNTIL [carácter == NULL]
END

```

El software es bastante simple. Primero se determina el número de desplazamientos y se establece la primera ubicación en 1234H. Despues el programa obtiene un carácter de la ubicación actual, lo encripta con el cifrado polialfabético y almacena el carácter encriptado de vuelta en la ubicación actual, con lo que se sobreescribe el carácter almacenado antes. Esto se repite hasta detectar un carácter NULL, lo cual indica el fin del mensaje.

RESUMEN

Hemos descrito algunos proyectos avanzados que se basan en el 8051, los cuales creemos son un reto y de interés para los estudiantes. Las posibles soluciones se describen en forma ordenada, empezando con un cuidadoso estudio de la descripción del proyecto y pasando a la especificación de los requerimientos y componentes del sistema. A esto le sigue una subsección acerca del diseño de software, en la cual se consideran algunas decisiones de programación y la correspondiente descripción del software mediante seudocódigo.

PROBLEMAS

- 13.1** Trace el diagrama esquemático para el proyecto del sistema de seguridad residencial de la sección 13.2; muestre todas las conexiones entre el 8051 y la alarma, los sensores, la pantalla de siete segmentos, y los chips de memoria externa. Por último, escriba el software (en lenguaje ensamblador o en C) necesario para controlar este sistema.
- 13.2** Trace el diagrama esquemático para el proyecto del sistema de elevador de la sección 13.3; muestre todas las conexiones entre el 8051 y los LEDs, los interruptores, y la pantalla de siete segmentos.
- 13.3** Observe el comportamiento de varios sistemas de elevadores localizados en su universidad, o en cualquier edificio a su alrededor. ¿En qué orden atienden estos sistemas las solicitudes del elevador? ¿Por qué cree usted que se utilice un orden así?
- 13.4** Trace el diagrama esquemático para el proyecto del juego Tres en raya de la sección 13.4; muestre todas las conexiones entre el 8051, el teclado numérico, los LEDs, y los interruptores. Por último, escriba el software (en lenguaje ensamblador o en C) necesario para controlar este sistema.
- 13.5** Trace el diagrama esquemático para el proyecto de la calculadora de la sección 13.5; muestre todas las conexiones entre el 8051 y el teclado numérico, la pantalla, el LED, y el interruptor.
- 13.6** Escriba la subrutina en lenguaje ensamblador o la función en C necesarias para realizar dos operaciones aritméticas cualesquiera de las que se listan en la descripción del proyecto de calculadora basada en el 8051.
- 13.7** Trace el diagrama esquemático para el proyecto del microrratón de la sección 13.6; muestre todas las conexiones entre el 8051 y los motores, sensores, LEDs, y la memoria externa.
- 13.8** Profundice un poco en el problema de programación del microrratón:
- Prueba de todas las rutas posibles: ¿en qué orden cree usted que deberían probarse? ¿Las bifurcaciones más cercanas primero, o las más lejanas? ¿Por qué?
 - Memorización de rutas recorridas con anterioridad: ¿cuál cree usted que sea la mejor forma de que el microrratón logre memorizarlas? ¿Por qué?
- 13.9** Al utilizar tarjetas inteligentes para identificación personal, a menudo es conveniente incrustar la foto del individuo en la tarjeta. No obstante, una limitación de la tarjeta inteligente es su pequeño tamaño de memoria. Investigue qué técnicas actuales se utilizan para incrustar fotos en tarjetas inteligentes.
- 13.10** Un día de clases, usted encuentra un pedazo de papel doblado donde cierto compañero de clases ha escrito un mensaje dirigido a un compañero del sexo opuesto. “Interesante”, piensa usted, a medida que aumenta la sonrisa en su rostro. Pero su regocijo se convierte en desaliento cuando ve que el resto del mensaje aparece así:

atih spit idcxvwi

Usted acaba de aprender sobre el cifrado del César en una clase previa, entonces empieza a suponer que el compañero que escribió la nota pudo haber utilizado esa técnica. Pero después de sentarse y tratar de descifrar el mensaje, descubre que no se utilizó el cifrado del César. Sin embargo, usted está seguro de que debe ser algo similar; sólo que a diferencia del cifrado del César, donde las letras se desplazan tres posiciones a la derecha, no sabe cuántos desplazamientos utilizó su compañero. ¡Utilizó un cifrado polialfabético!

De repente, le llega una ráfaga de inspiración: ¡Usar el 8051 para descifrar el mensaje! Escriba un programa en el 8051 para hacer esto.

Pista: Necesitará realizar lo que se conoce como **ataque de diccionario**: pruebe todos los posibles desplazamientos y compare cada palabra descifrada con las palabras de un diccionario para ver si hay coincidencias. Una vez descubiertas coincidencias para las tres palabras, probablemente haya descifrado el mensaje con éxito. Suponga que cuenta con un diccionario de 1000 entradas, incluyendo las palabras del mensaje original, representado mediante el siguiente arreglo de caracteres:

```
char * diccionario = {"a", "able", "about", ... };
```

donde diccionario[0] contiene "a", diccionario[1] contiene "able", etcétera.

Escriba el seudocódigo para un programa de este tipo.

TABLE 13–3
Tabla ASCII parcial

Carácter	Código ASCII		
	Decimal	Hex	Binario
a	97	61H	01100001
b	98	62H	01100010
c	99	63H	01100011
d	100	64H	01100100
e	101	65H	01100101
f	102	66H	01100110
g	103	67H	01100111
...

Escriba el seudocódigo para un programa de este tipo.

- 13.11** Escriba la solución correspondiente en C del 8051 para el problema 13.10. Después utilice el programa para descifrar el mensaje.
- 13.12** El Advanced Encryption Standard (AES) es un reciente estándar de cifrado elegido para sustituir al DES y al DES triple en las futuras aplicaciones de seguridad. Investigue acerca de cómo funciona el AES y escriba un programa para el 8051 (ya sea en lenguaje ensamblador o en C) donde se implemente el estándar AES. En el apéndice J aparece también una breve descripción del AES.

Derivados del 8051

14.1 INTRODUCCIÓN

Desde la aparición de la familia de circuitos integrados microcontroladores MCS-51TM, han surgido versiones más nuevas y más avanzadas. Estos **derivados del 8051** cuentan con memoria adicional, E/S incorporada tal como ADCs y DACs, y otros periféricos extendidos. En esta sección repasaremos brevemente algunos de estos derivados.

14.2 MCS-151TM Y MCS-251TM

Intel ha producido versiones avanzadas de la familia MCS-51TM. Éstas son el MCS-151TM, que aumenta cinco veces su rendimiento, seguido de la familia MCS-251TM con un aumento de 15 veces en su rendimiento. La familia MCS-251TM tiene una arquitectura avanzada que aumenta la eficiencia de programación en lenguaje C del 8051, un conjunto de instrucciones extendido que incluye instrucciones para efectuar operaciones aritméticas y lógicas de 16 y 32 bits, además de opciones de aumento en el tamaño de la memoria (consulte la tabla 14-1). Aquí utilizaremos el término **ROM programable una sola vez (OTP)** para referirnos a la memoria para código parecida a la EPROM, pero sin la ventana de vidrio de cuarzo para borrar su contenido. Esto reduce el costo del empaque, pero no permite que la OTP ROM pueda borrarse con luz ultravioleta (UV); por lo tanto, sólo puede programarse una sola vez. También hay disponibles versiones sin ROM.

14.3 MICROCONTROLADORES CON MEMORIA FLASH Y NVRAM

En el capítulo 2 presentamos el MCS-51TM con diferentes tipos de memoria para código incorporada en el chip, los cuales variaban desde el 8031 sin ROM hasta el 8051 con ROM incorporada en el chip, y el 8751 con EPROM incorporada en el chip.

TABLA 14-1

Comparación de los circuitos integrados MCS-251™

Número de pieza	Memoria para código incorporada en el chip	Memoria para datos incorporada en el chip
80251SA	8K ROM/OTPROM	1K
80251SB	16K ROM/OTPROM	1K
80251SP	8K ROM/OTPROM	512 bytes
80251SQ	16K ROM/OTPROM	512 bytes
80251TA	8K ROM	1K
80251TB	16K ROM	1K
80251TP	8K ROM	512 bytes
80251TQ	16K ROM	512 bytes

Aún con un 8751, que permite al programador reprogramarlo con un programador PROM, primero tendríamos que borrarlo con un borrador de EPROM ultravioleta (UV) antes de poder programarlo de nuevo. En cambio, el 8951 es un derivado del 8051 con **memoria flash** incorporada en el chip, que funciona básicamente como una EPROM, pero su contenido puede borrarse electrónicamente por el propio programador PROM; por lo tanto, no requiere de un borrador UV-EPROM separado. Atmel Corporation¹ es uno de los fabricantes del 8951.

Además de los derivados que hacen uso de la memoria flash, otros como el DS5000 de Maxim Integrated Products² tienen una NVRAM para la memoria para código. La ventaja de una NVRAM en comparación con una memoria flash es que podemos cambiar nuestro programa en la memoria para código, un byte a la vez, en lugar de tener que borrarlo por completo antes de la reprogramación. La NVRAM del DS5000 también permite que los programas puedan volverse a cargar en la memoria para código incluso a través del puerto serial de la PC, con lo cual se elimina la necesidad de implementar un programador PROM separado.

14.4 MICROCONTROLADORES CON ADCS Y DACS

Una de las características más comunes de los derivados del 8051 son los convertidores analógicos/digitales y los convertidores digitales/análogicos incorporados, los cuales reducen la necesidad de conectarse a ADCs y DACs externos cuando se requiere de una interfaz con el 8051 para dispositivos de E/S analógicos. Un ejemplo de uno de esos derivados del 8051 es el microcontrolador SAB80C515A fabricado por Siemens, el cual cuenta con un ADC de 10 bits incorporado. La colección de derivados del 8051 de Atmel son versiones más avanzadas que también funcionan como reproductores de MP3, incluyendo al AT89C51SND1C y al AT89C51SND2C, que incluso tienen la reciente y popular interfaz para computadoras personales USB 1.1.

14.5 MICROCONTROLADORES DE ALTA VELOCIDAD

El 8051 se ejecuta a 12 ciclos de reloj por cada ciclo de máquina. Algunos derivados de alta velocidad, como el MCS-151™ y el MCS-251™, se ejecutan a sólo dos ciclos de reloj por cada ciclo de máquina y, por lo tanto, son capaces de ejecutar más instrucciones dentro de un tiempo dado.

¹Atmel Corporation, 2125 O'Nel Drive, San Jose, CA 95131.

²Maxim Integrated Products, Inc., incluye a Dallas Semiconductors.

Maxim también produce varios tipos de derivados del 8051, incluyendo microcontroladores de alta velocidad, microcontroladores para redes, y microcontroladores seguros. Sus controladores de alta velocidad se ejecutan a cuatro ciclos de reloj por cada ciclo de máquina, en comparación con los 12 ciclos de reloj del 8051, mientras que su microcontrolador de ultra alta velocidad, el DS89C420, se ejecuta a sólo un ciclo de reloj por cada ciclo de máquina. Otras mejoras en relación con el 8051 incluyen más fuentes de interrupción y aumento en el tamaño de su memoria.

14.6 MICROCONTROLADORES PARA REDES

Los microcontroladores para redes fabricados por Maxim soportan diversos protocolos de red, tales como Ethernet y Controller Area Network (CAN). Otros microcontroladores para redes, como el 83751 de Philips,³ soportan la interfaz a redes de circuitos interintegrados (I^2C), mientras que el COM20051 de Standard Microsystems⁴ soporta el protocolo de red token ring ARCNET. Tales protocolos de red permiten que varios microcontroladores y otros procesadores puedan conectarse como una red para compartir e intercambiar datos. El ATWebSEG-32 de Atmel es un derivado del 8051 que soporta la conexión a internet (TCP/IP) y el protocolo Ethernet.

14.7 MICROCONTROLADORES SEGUROS

En el capítulo anterior vimos la forma en que el 8051 puede utilizarse como cerebro en tarjetas inteligentes. Esto incluyó un análisis sobre cómo podemos proteger información confidencial mediante la codificación por software. De hecho, la codificación puede realizarse con el hardware de seguridad dedicado, y varios derivados del 8051, como los microcontroladores seguros 8051 de Maxim, se han producido para lograr esto.

Al sistema de seguridad incorporado en la tarjeta inteligente y que permite la codificación, las firmas digitales, y la autenticación de mensajes se le conoce comúnmente como **infraestructura de llave pública (PKI)**. Por lo general, esta PKI se integra en los microcontroladores seguros y es soportada por los periféricos de hardware incorporados en estos microcontroladores; por ejemplo, en el DS5240 de Maxim, el cual tiene un acelerador de módulo aritmético (MAA) para soportar operaciones de módulo aritmético que se utilizan mucho en las PKI. Otros microcontroladores seguros, como el DS5000 de Maxim, soportan codificación por hardware de los programas cargados en su memoria para código. Al codificar estos programas, aunque se ataque la tarjeta inteligente y se acceda a sus programas sin autorización, el intruso no podrá entender el significado de los programas.

RESUMEN

En este capítulo describimos muchos de los derivados del 8051. Estas versiones actualizadas por lo general tienen mejores tipos de memoria incorporados en el chip, mayor tamaño de memoria, velocidades más altas, y más periféricos incorporados para soportar la interacción con dispositivos de E/S analógicos, redes y aplicaciones de seguridad.

³Philips Semiconductors, 811 E. Arques Avenue, Box 3409, Sunnyvale, CA 94088.

⁴Standard Microsystems Corporation, 80 Arkay Drive, Hauppauge, NY 11788.

Una vez expresado esto, la decisión final de la selección del 8051 o del derivado más adecuado para su proyecto depende de una serie de criterios tales como el tamaño y el tipo de ROM y RAM incorporadas en el chip, la velocidad, así como E/S avanzada, redes, y requisitos de seguridad.

PROBLEMAS

- 14.1** Investigue sobre los diversos fabricantes de los derivados del 8051 y haga una lista de otros derivados con ADCs o DACs incorporados.
- 14.2** Investigue acerca de los fabricantes del 8051 y haga una lista de otros microcontroladores seguros. Incluya sus características de seguridad.
- 14.3** ¿Existen otras mejorías al 8051 que no estudiamos en este capítulo? Haga una lista de las mejorías, si las hay, y ejemplos de tales derivados.

A

Diagrama de referencia rápida

MNEMÓNICO	DESCRIPCIÓN	MNEMÓNICO	DESCRIPCIÓN
Operaciones aritméticas			
ADD A,fuente	suma entre origen y A	XRL directa,#datos	
ADD A,#datos		CLR A	limpia A
ADDC A,fuente	suma con acarreo	CPL A	complementa A
ADDC A,#datos		RL A	desplaza A hacia la izquierda
SUBB A,source	resta de A	RLC A	(a través de C)
SUBB A,#datos	con acarreo de suma	RR A	desplaza A hacia la derecha
INC A	incremento	RRC A	(a través de C)
INC fuente		SWAP A	intercambia nibbles
DEC A	disminución		
DEC fuente			
INC DPTR	incrementa DPTR		
MUL AB	multiplicación de A por B		
DIV AB	división de A entre B		
DA A	ajuste decimal de A		
Operaciones lógicas		LEYENDA	
ANL A,fuente	AND lógico	Rn	direcciónamiento de registro utilizando R0-R7
ANL A,#datos		directa	dirección interna de 8 bits (00H-FFH)
ANL directa,A		@Ri	direcciónamiento indirecto utilizando R0 o R1
ANL directa,#datos		fuente	cualquiera de [Rn, directa, @Ri]
ORL A,fuente	OR lógico	destino	cualquiera de [Rn, directa, @Ri]
ORL A,#datos		#datos	constante de 8 bits incluida en la instrucción
ORL directa,A		#datos16	constante de 16 bits
ORL directa,#datos		bit	dirección de bit directa de 8 bits
XRL A,fuente	XOR lógico	rel	desplazamiento con signo de 8 bits
XRL A,#datos		dir11	dirección de 11 bits en la página de 2k actual
XRL directa,A		dir16	dirección de 16 bits

FIGURA A-1

Diagrama de referencia rápida



Mapa de códigos de operación

Instruction Code Summary

L	H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	JBC bit, rel	JB bit, rel	JNB bit, rel	JC rel	JNC rel	JZ rel	JNZ rel	SIMP rel	MOV DPTR, # datos16	ORL C, /bit	ANL C, /bit	PUSH dir	POP dir	MOVX A, @DPTR	MOVX @DPTR, A	
1	AJMP (P0)	ACALL (P0)	AJMP (P1)	ACALL (P1)	AJMP (P2)	ACALL (P2)	AJMP (P3)	ACALL (P3)	AJMP (P4)	ACALL (P4)	AJMP (P5)	ACALL (P5)	AJMP (P6)	ACALL (P6)	AJMP (P7)	ACALL (P7)	
2	LJMP dir16	LCALL dir16	RET	RETI	ORL dir, A	ANL dir, A	XRL dir, A	ORL C, bit	ANL C, bit	MOV bit, C	MOV C, bit	CPL bit	CLR bit	SETB bit	MOVX A, @R0	MOVX @R0, A	
3	RR A	RRC A	RL A	RLC A	ORL dir, # datos	ANL dir, # datos	XRL dir, # datos	JMP @A+DPTR	MOVCA, @A-PC	MOVCA, @A+DPTR	INC DPTR	CPL C	CLR C	SETB C	MOVX A, @R1	MOVX @R1, A	
4	INC A	DEC A	ADD A, # datos	ADDC A, # datos	ORL A, # datos	ANL A, # datos	XRL A, # datos	MOV A, # datos	DIV AB	SUBB A, # datos	MUL AB	CINE A, # datos, rel	SWAP A	DA A	CLR A	CPL A	
5	INC dir	DEC dir	ADD A, dir	ADDC A, dir	ORL A, dir	ANL A, dir	XRL A, dir	MOV dir, # datos	MOV dir, dir	SUBB A, dir		CINE A, dir, rel	XCH A, dir	DJNZ dir, rel	MOV A, dir	MOV dir, A	
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	MOV @R0, # datos	MOV dir, @R0	SUBB A, @R0	MOV @R0, dir	CINE @R0, # datos, rel	XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @R0, A	
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	MOV @R1, # datos	MOV dir, @R1	SUBB A, @R1	MOV @R1, dir	CINE @R1, # datos, rel	XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @R1, A	
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	MOV R0, # datos	MOV dir, R0	SUBB A, R0	MOV R0, dir	CINE R0, # datos, rel	XCH A, R0	DJNZ R0, rel	MOV A, R0	MOV R0, A	
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	MOV R1, # datos	MOV dir, R1	SUBB A, R1	MOV R1, dir	CINE R1, # datos, rel	XCH A, R1	DJNZ R1, rel	MOV A, R1	MOV R1, A	
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	MOV R2, # datos	MOV dir, R2	SUBB A, R2	MOV R2, dir	CINE R2, # datos, rel	XCH A, R2	DJNZ R2, rel	MOV A, R2	MOV R2, A	
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	MOV R3, # datos	MOV dir, R3	SUBB A, R3	MOV R3, dir	CINE R3, # datos, rel	XCH A, R3	DJNZ R3, rel	MOV A, R3	MOV R3, A	
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	MOV R4, # datos	MOV dir, R4	SUBB A, R4	MOV R4, dir	CINE R4, # datos, rel	XCH A, R4	DJNZ R4, rel	MOV A, R4	MOV R4, A	
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	MOV R5, # datos	MOV dir, R5	SUBB A, R5	MOV R5, dir	CINE R5, # datos, rel	XCH A, R5	DJNZ R5, rel	MOV A, R5	MOV R5, A	
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	MOV R6, # datos	MOV dir, R6	SUBB A, R6	MOV R6, dir	CINE R6, # datos, rel	XCH A, R6	DJNZ R6, rel	MOV A, R6	MOV R6, A	
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	MOV R7, # datos	MOV dir, R7	SUBB A, R7	MOV R7, dir	CINE R7, # datos, rel	XCH A, R7	DJNZ R7, rel	MOV A, R7	MOV R7, A	

2Bytes 3Bytes
2Ciclos 4Ciclos

FIGURA B-1

Mapa de códigos de operación



Definiciones de las instrucciones¹

LEYENDA

Símbolo	Interpretación
←	es reemplazado por . . .
()	el contenido de . . .
(())	los datos apuntados por . . .
rr	uno de ocho registros; 000 = R0, 001 = R1, etc.
ddddd़dd	bits de datos
aaaaaaaa	bits de dirección
bbbbbbb	dirección de un bit
i	direccionalamiento indirecto utilizando R0 (i = 0) o R1 (i = 1)
eeeeeee	dirección relativa de 8 bits

ACALL addrl

Función:	Llamada absoluta
Descripción:	ACALL llama incondicionalmente a una subrutina que se ubica en la dirección indicada. La instrucción incrementa dos veces al contador de programa (PC) para obtener la dirección de la instrucción que le sigue, y entonces almacena el resultado de 16 bits en la pila (con el byte inferior primero) e incrementa dos veces al apuntador de pila. La dirección de destino se obtiene al concatenar satisfactoriamente los cinco bits de orden superior del PC ya incrementado, los bits 7 a 5 del código de operación, y el segundo byte de la instrucción.

¹ Adaptadas del documento Controladores de 8 bits (270645) incrustados, Santa Clara, CA: Intel Corporation, 1991, con permiso de Intel Corporation.

La subrutina llamada debe, por lo tanto, iniciar dentro del mismo bloque de 2K de la memoria de programa donde inicia el primer byte de la instrucción que sigue de la instrucción ACALL. Ninguna bandera se ve afectada.

Ejemplo: Al principio, SP equivale a 07H. La etiqueta “SUBRTN” es una ubicación en memoria del programa 0345H. Después de ejecutar la instrucción,

ACALL SUBRTN

en la ubicación 0123H, el SP contiene el valor 09H, las ubicaciones en memoria RAM interna 08H y 09H contienen los valores 25H y 01H, respectivamente, y el PC contiene el valor 0345H.

Bytes: 2

Ciclos: 2

Codificación: aaa10001 aaaaaaaaaa

Nota: aaa = A10 ← A8 y aaaaaaaaaa = A7 – A0 de la dirección de destino.

Operación: $(PC) \cdot (PC) + 2$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC7 - PC0)$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC15 - PC8)$

$(PC10 = PC0) \leftarrow$ dirección de página

ADD A,<byte origen>

Función: Suma

Descripción: ADD hace una suma entre la variable byte indicada y el acumulador, y deja el resultado en el acumulador. Las banderas de acarreo y de acarreo auxiliar se activan, respectivamente, si hay un acarreo del bit 7 o del bit 3, y se borran en el caso contrario. Cuando se hace una suma de enteros sin signo, la bandera de acarreo indica si ocurrió un desbordamiento.

OV se activa si ocurre un acarreo del bit 6 pero no del bit 7, o un acarreo del bit 7 pero no del bit 6; en el caso contrario, OV se borra. Cuando se hace una suma de enteros con signo, OV indica que se produjo un número negativo resultante de la suma de sus dos operandos positivos, o que se produjo una suma positiva de dos operandos negativos.

Se permiten cuatro modos de direccionamiento para el operando de origen: por registros, directo, indirecto por registros, o inmediato.

Ejemplo: El acumulador contiene el valor 0C3H (000011B) y el registro 0 contiene 0AAH (10101010B). La instrucción,

ACC A, R0

deja el valor 6DH (01101110B) en el acumulador con la bandera AC en cero, y tanto la bandera de acarreo como la OV se establecen a 1.

ADD A,Rn

Bytes: 1
 Ciclos: 1
 Codificación: 00101rrr
 Operación: $(A) \leftarrow (A) + (R_n)$

ADD A,directa

Bytes: 2
 Ciclos: 1
 Codificación: 00100101 aaaaaaaa
 Operación: $(A) \leftarrow (A) + (\text{directa})$

ADD A,@Ri

Bytes: 1
 Ciclos: 1
 Codificación: 0010011i
 Operación: $(A) \leftarrow (A) + ((R_i))$

ADD A,#datos

Bytes: 2
 Ciclos: 1
 Codificación: 00100100 dddddddd
 Operación: $(A) \leftarrow (A) + \#datos$

ADDC A,<byte origen>

Función:	Suma con acarreo
Descripción:	ADDC suma en forma simultánea la variable byte indicada, la bandera de acarreo, y el contenido del acumulador, y deja el resultado en el acumulador. Las banderas de acarreo y de acarreo auxiliar se activan, respectivamente, si hay un acarreo del bit 7 o del bit 3, y se borran en el caso contrario. Cuando se hace una suma de enteros sin signo, la bandera de acarreo indica que ocurrió un desbordamiento.

OV se activa si hay un acarreo del bit 6 pero no del bit 7, o un acarreo del bit 7 pero no del bit 6; en cualquier otro caso, OV se borra. Cuando se hace una suma de enteros con signo, OV indica que se produjo un número negativo como la suma de dos operandos positivos o que se produjo una suma positiva con dos operandos negativos.

Se permiten cuatro modos de direccionamiento a los operandos de origen: por registros, directo, indirecto por registros, o inmediato.

- Ejemplo: El acumulador contiene el valor 0C3H (11000011B) y el registro 0 contiene el valor 0AAH (10101010B) con la bandera de acarreo activada. La instrucción,

ADDC A, R0

deja el valor 6EH (01101110B) en el acumulador con AC en cero, y activa tanto la bandera de acarreo como la OV con el valor de 1.

ADDC A,Rn

Bytes:	1
Ciclos:	1
Codificación:	00110rrr
Operación:	$(A) \leftarrow (A) + (C) + (R_n)$

ADDC A,directa

Bytes:	2
Ciclos:	1
Codificación:	00110101 aaaaaaaaaa
Operación:	$(A) \leftarrow (A) + (C) + (\text{directa})$

ADDC A,@Ri

Bytes:	1
Ciclos:	1
Codificación:	0011011i
Operación:	$(A) \leftarrow (A) + (C) + (R_i)$

ADDC A,#datos

Bytes:	2
Ciclos:	1
Codificación:	00110100 dddddddd
Operación:	$(A) \leftarrow (A) + (C) + \#datos$

AJMP dir11

Función:	Salto absoluto
Descripción:	AJMP transfiere la ejecución del programa a la dirección indicada, la cual se forma al tiempo de ejecución concatenando los cinco bits de orden superior del PC (<i>después</i> de incrementar dos veces al PC), los bits 7 a 5 del código de operación, y el segundo byte de la instrucción. El destino debe encontrarse, por lo tanto, dentro del mismo bloque de 2K de la memoria de programa que el primer byte de la instrucción que sigue de la instrucción AJMP.
Ejemplo:	La etiqueta “SALTADIR” está en la ubicación en memoria de programa 0123H. La instrucción, AJMP SALTADIR está en la ubicación 0345H y carga al PC con el valor 0123H.
Bytes:	2
Ciclos:	2
Codificación:	aaa00001 aaaaaaaaaa
	<i>Nota:</i> aaa = A10–A8 y aaaaaaaaaa = A7–A0 de la dirección de destino.
Operación:	(PC) ← (PC) + 2 (PC10–PC0) ← dirección de página

ANL <byte destino>,<byte origen>

Función:	AND lógico para variables tipo byte
Descripción:	ANL realiza una operación de AND lógico orientada a bits entre las variables indicadas, y almacena el resultado en la variable de destino. Ninguna bandera se ve afectada.
	Los dos operandos permiten seis combinaciones para el modo de direccionamiento. Cuando el destino es el acumulador, el origen puede utilizar un direccionamiento por registros, directo, indirecto por registros, o inmediato; cuando el destino es una dirección directa, el origen puede ser el acumulador o datos inmediatos.
	<i>Nota:</i> Cuando se utiliza esta instrucción para modificar un puerto de salida, el valor utilizado como el dato original del puerto se lee del latch de datos de salida, <i>no</i> se lee de las terminales de entrada.
Ejemplo:	Si el acumulador contiene el valor 0C3H (11000011B) y el registro 0 contiene el valor 55H (01010101B), entonces la instrucción,

ANL A, R0

deja el valor 41H (01000001B) en el acumulador.

Cuando el destino es un byte direccionado directamente, la instrucción borra las combinaciones de bits en cualquier ubicación en RAM o registro en hardware. El byte de máscara determina si el patrón de los bits a borrar es una constante contenida en la instrucción o si es el valor calculado en el acumulador al tiempo de ejecución.

La instrucción,

```
ANL P1,#01110011B
```

borra los bits 7, 3 y 2 del puerto 1 de salida.

ANL A,Rn

Bytes:	1
Ciclos:	1
Codificación:	01011rrr
Operación:	(A) \leftarrow (A) AND (Rn)

ANL A,directa

Bytes:	2
Ciclos:	1
Codificación:	01010101 aaaaaaaaa
Operación:	(A) \leftarrow (A) AND (directa)

ANL A,@Ri

Bytes:	1
Ciclos:	1
Codificación:	0101011i
Operación:	(A) \leftarrow (A) + (C) + ((Ri))

ANL A,#datos

Bytes:	2
Ciclos:	1
Codificación:	01010100 dddddddd
Operación:	(A) \leftarrow (A) AND #datos

ANL directa,A

Bytes:	2
--------	---

Ciclos: 1
 Codificación: 01010010 aaaaaaaaa
 Operación: (directa) \leftarrow (directa) AND (A)

ANL directa,#datos

Bytes: 3
 Ciclos: 2
 Codificación: 01010011 aaaaaaaaa dddddddd
 Operación: (directa) \leftarrow (directa) AND #datos

ANL C,<bit origen>

Función:	AND lógico para variables tipo bit
Descripción:	Si el valor booleano del bit de origen es un 0 lógico, entonces borra la bandera de acarreo; de lo contrario, deja la bandera de acarreo en su estado actual. Un signo de división (/) incluido antes del operando en el programa en lenguaje ensamblador indica que se debe utilizar el complemento lógico del bit direccionado como el valor de origen, <i>pero no afecta al propio bit de origen</i> . Ninguna otra bandera se ve afectada.
	Sólo se permite el direccionamiento directo para el operando origen.
Ejemplo:	Activa la bandera de acarreo si, y sólo si, P1.0 = 1, ACC.7 = 1, y OV = 0:

```
MOV C, P1.0      ;CARGA C CON EL ESTADO DE LA
                   ;TERMINAL DE ENTRADA
ANL C,ACC.7      ;EFFECTÚA UN AND LÓGICO DEL
                   ;ACARREO CON EL BIT 7 DEL ACC
ANL C,/OV        ;EFFECTÚA UN AND LÓGICO CON EL
                   ;INVERSO DE LA BANDERA OV
```

ANL C,bit

Bytes: 2
 Ciclos: 2
 Codificación: 10000010 bbbbbbbb
 Operación: (C) \leftarrow (C) AND (bit)

ANL C,/bit

Bytes: 2
 Ciclos: 2
 Codificación: 10110000 bbbbbbbb
 Operación: (C) \leftarrow (C) AND NOT(bit)

CALL (Consulte ACALL o LCALL)

CJNE <byte destino>,<byte origen>,rel

Función:	Compara y salta si no es igual
Descripción:	CJNE compara las magnitudes de los primeros dos operandos y realiza una bifurcación si sus valores no son iguales. El destino de la bifurcación se calcula mediante la suma de desplazamiento relativo con signo en el último byte de la instrucción con el PC, después de incrementar el PC al inicio de la siguiente instrucción. La bandera de acarreo se activa si el valor entero sin signo de <byte destino> es menor que el valor entero sin signo de <byte origen>; de lo contrario, la bandera de acarreo se borra. Ningún operando se ve afectado.
	Los primeros dos operandos permiten cuatro combinaciones de modo de direccionamiento: el acumulador puede compararse con cualquier byte direccionado directamente o con datos inmediatos, y cualquier ubicación indirecta en RAM o registro habilitado puede compararse con una constante inmediata.
Ejemplo:	El acumulador contiene el valor 34H. El registro 7 contiene el valor 56H. La primera instrucción en la secuencia

```

CJNER7, #60H, NO_ES_IGUAL
;
    . . . . . ;R7 = 60H
NO_ES_IGUAL: JC REG_BAJO ;IF R7 < 60H
;
    . . . . . ;R7 > 60H
REG_BAJO   . . . . . ;R7 < 60H

```

activa la bandera de acarreo y se bifurca hacia la instrucción en la etiqueta NO_ES_IGUAL. Al probar la bandera de acarreo, esta instrucción determina si R7 es mayor o menor que 60H.

Si los datos presentados en el puerto 1 también equivalen a 34H, entonces la instrucción,

```
ESPERA: CJNE A, P1, ESPERA
```

borra la bandera de acarreo y continúa con la siguiente instrucción, ya que el acumulador no es igual al dato leído del puerto 1. (Si algún otro valor se envía como entrada a P1, el programa entra en un ciclo hasta que el dato en P1 cambia su valor a 34H.)

CJNE A,directa,rel

Bytes:	3
Ciclos:	2
Codificación:	10110101 aaaaaaaaaa eeeeeeee
Operación:	(PC) ← (PC) + 3

```

IF (A) <>(directa)
THEN
    (PC) ← (PC) + dirección relativa
IF (A) <(directa)
THEN
    (C) ← 1
ELSE
    (C) ← 0

```

CJNE A,#datos,rel

Bytes:	3
Ciclos:	2
Codificación:	10110100 dddddddd eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 3$ IF(A) <> datos THEN $(PC) \leftarrow (PC) + \text{dirección relativa}$ IF(A)<datos THEN $(C) \leftarrow 1$ ELSE $(C) \leftarrow 0$

CJNE Rn,#datos,rel

Bytes:	3
Ciclos:	2
Codificación:	10111rrr dddddddd eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 3$ IF (Rn) <> datos THEN $(PC) \leftarrow (PC) + \text{dirección relativa}$ IF (Rn) <datos THEN $(C) \leftarrow 1$ ELSE $(C) \leftarrow 0$

CJNE @Ri,#datos,rel

Bytes: 3
 Ciclos: 2
 Codificación: 1011011i eeeeeeee
 Operación: $(PC) \leftarrow (PC) + 3$
 $IF (Ri) <> \text{datos}$
 THEN
 $(PC) \leftarrow (PC) + \text{dirección relativa}$
 $IF ((Ri)) < \text{datos}$
 THEN
 $(C) \leftarrow 1$
 ELSE
 $(C) \leftarrow -0$

CLR A

Función: Borra acumulador
 Descripción: El acumulador se borra (todos los bits se igualan a 0). Ninguna bandera se ve afectada.
 Ejemplo: El acumulador contiene el valor 5CH (01011100B). La instrucción,

CLR A
 deja el acumulador igualado a 00H (00000000B).

Bytes: 1
 Ciclos: 1
 Codificación: 11100100
 Operación: $(A) \leftarrow 0$

CLR bit

Función: Borra bit
 Descripción: El bit indicado se borra (se reinicializa a 0). Ninguna otra bandera se ve afectada. CLR puede operar en la bandera de acarreo o en cualquier bit directamente direccionable.
 Ejemplo: El puerto 1 contiene el valor 5DH (01011101B). La instrucción,

CLR P1 . 2
 deja al puerto con el valor 59H (01011001B).

CLR C

Bytes: 1
 Ciclos: 1
 Codificación: 11000011
 Operación: $(C) \leftarrow 0$

CLR bit

Bytes: 2
 Ciclos: 1
 Codificación: 11000010 bbbbbbbb
 Operación: $(\text{bit}) \leftarrow 0$

CPL A

Función: Complementa al acumulador
 Descripción: Cada bit del acumulador se complementa en forma lógica (complementos a 1). Los bits que antes contenían el valor 1 se cambian a 0, y viceversa. Ninguna bandera se ve afectada.
 Ejemplo: El acumulador contiene el valor 5CH (01011100B). La instrucción

CPL A

deja al acumulador con el valor 0A3H (10100011B).

Bytes: 1
 Ciclos: 1
 Codificación: 11110100
 Operación: $(A) \leftarrow \text{NOT}(A)$

CPL bit

Función: Complementa bit
 Descripción: La variable bit especificada se complementa. Un bit que era 1 se cambia a 0, y viceversa. Ninguna otra bandera se ve afectada. CLR puede operar sobre la bandera de acarreo o sobre cualquier bit directamente direccionable.
Nota: Cuando esta instrucción se utiliza para modificar una terminal de salida, el valor utilizado como el dato original proviene del latch de datos de salida, *no* de la terminal de entrada.
 Ejemplo: El puerto 1 contiene el valor 5BH (01011011B).
 Las instrucciones,

CPL	P1.1
CPL	P1.2

dejan al puerto con el valor 5BH (01011011B).

CPL C

Bytes:	1
Ciclos:	1
Codificación:	10110011
Operación:	(C) \leftarrow NOT(C)

CPL bit

Bytes:	2
Ciclos:	1
Codificación:	1010010 bbbbbbbb
Operación:	(bit) \leftarrow NOT(bit)

DA A

Función:	Ajuste decimal del acumulador para la suma
Descripción:	DA A ajusta el valor de 8 bits en el acumulador resultante de una suma previa de dos variables (cada una en formato de BCD empaquetado), y produce dos dígitos de 4 bits. Se puede utilizar cualquier instrucción ADD o ADDC para realizar la suma.
	Si los bits 3 a 0 del acumulador son mayores a 9 (xxxx1010-xxxx1111), o si la bandera AC es 1, se agrega un 6 al acumulador para producir el dígito BCD adecuado en el nibble de orden inferior. Esta suma interna activa la bandera de acarreo si un acarreo del campo de 4 bits de orden inferior se propagó a lo largo de todos los bits de orden superior, pero no borra la bandera de acarreo en el caso contrario.
	Si la bandera de acarreo ahora está activada, o si los cuatro bits de orden superior ahora exceden a 9 (1010xxxx-1111xxxx), estos bits de orden superior se incrementan por 6, con lo cual se produce el dígito BCD adecuado en los bits de orden superior, pero no se borra el acarreo. Por lo tanto, la bandera de acarreo indica si la suma de las dos variables BCD originales es mayor a 99, ello permite obtener una suma decimal precisa. La bandera OV no resulta afectada.
	Todo lo descrito líneas arriba ocurre durante un ciclo de instrucción. Esencialmente, esta instrucción realiza la conversión decimal al

agregar 00H, 06H, 60H, o 66H al acumulador, dependiendo de las condiciones iniciales en el acumulador y en la PSW.

Nota: DA A no puede simplemente convertir un número en hexadecimal que haya en el acumulador a notación BCD, y la instrucción DA A tampoco se aplica a la resta decimal.

- Ejemplo: El acumulador contiene el valor 56H (01010110B), el cual representa los dígitos en el BCD empaquetado del número decimal 56. El registro 3 contiene el valor 67H (01100111B), el cual representa los dígitos en el BCD empaquetado del número decimal 67. La bandera de acarreo está activada. Las instrucciones,

```
ADDC A, R3
DA A
```

realizan primero una suma binaria de complementos a 2 estándar, lo que resulta en el valor OBEH (10111110B) en el acumulador. Las banderas de acarreo y de acarreo auxiliar se borran.

Después, la instrucción de ajuste decimal altera el acumulador al valor 24H (00100100B), indicando los dígitos en el BCD empaquetado del número decimal 24, los dos dígitos de orden inferior de la suma decimal de 56, 67, y el acarreo. La bandera de acarreo se activa mediante la instrucción de ajuste decimal, lo cual indica que ocurrió un desbordamiento decimal. La verdadera suma de 56, 67 y 1 es 124.

Las variables en BCD pueden incrementarse o disminuir agregando 01H o 99H. Si el acumulador contiene inicialmente 30H (lo que representa los dígitos 30 en decimal), entonces las instrucciones,

```
ADD      A, #99H
DA       A
```

dejan activada la bandera de acarreo y el valor 29H en el acumulador, ya que $30 + 99 = 129$. El byte de orden inferior de la suma puede interpretarse con el significado de $30 - 1 = 29$.

- | | |
|---------------|---|
| Bytes: | 1 |
| Ciclos: | 1 |
| Codificación: | 11010100 |
| Operación: | (Suponga que en el acumulador los contenidos están en BCD.)
IF [(A3 – A0) >9]AND [(AC = 1)]
THEN (A3 – A0) ← (A3 – A0) + 6
AND
IF [(A7 – A4) >9]AND [(C = 1)]
THEN (A7 – A4) ← (A7 – A4) + 6 |

DEC BYTE

Función:	Disminución
Descripción:	La variable indicada se disminuye en 1. Un valor original de 00H provoca un desbordamiento negativo a 0FFH. Ninguna bandera se ve afectada. Se permiten cuatro modos de direccionamiento de operando: por acumulador, por registros, directo, o indirecto por registros.
	<i>Nota:</i> Cuando esta instrucción se utiliza para modificar un puerto de salida, el valor utilizado como el dato del puerto original es del latch de datos de salida, <i>no</i> es de las terminales de entrada.
Ejemplo:	El registro 0 contiene el valor 7FH (0111111B). Las ubicaciones en RAM interna 7EH y 7FH contienen los valores 00H y 40H, respectivamente. Las instrucciones,

```
DEC    @R0
DEC    R0
DEC    @R0
```

dejan al registro 0 con el valor 7EH y a las ubicaciones en RAM interna 7EH y 7FH con el valor 0FFH y 3FH.

DEC A

Bytes:	1
Ciclos:	1
Codificación:	00010100
Operación:	$(A) \leftarrow (A) - 1$

DEC Rn

Bytes:	1
Ciclos:	1
Codificación:	00011rrr
Operación:	$(Rn) \leftarrow (Rn) - 1$

DEC directa

Bytes:	2
Ciclos:	1
Codificación:	00010101 aaaaaaaaa
Operación:	$(directa) \leftarrow (directa) - 1$

DEC @Ri

Bytes: 1
 Ciclos: 1
 Codificación: 0001011i
 Operación: $((Ri)) \leftarrow ((Ri)) - 1$

DIV AB

Función:	División
Descripción:	DIV AB divide un entero de 8 bits sin signo que haya en el acumulador entre un entero de 8 bits sin signo que esté en el registro B. El acumulador recibe la parte entera del cociente; el registro B recibe el resto del entero. Las banderas de acarreo y OV se borran.
	<i>Excepción:</i> Si al principio B contenía el valor 00H, los valores regresados en el acumulador y en el registro B están indefinidos, y la bandera de desbordamiento se activa. La bandera de acarreo se borra en cualquier caso.
Ejemplo:	El acumulador contiene el valor 251 (0FBH o 11111011B), y B contiene el valor 18 (12H o 00010010B). La instrucción,

DIV AB

deja el valor 13 en el acumulador (0DH o 00001101B), y el valor 17 (11H o 00010000B) en el registro B, ya que $251 = 13 \times 18 + 17$. Las banderas de acarreo y la OV se borran.

Bytes: 1
 Ciclos: 4
 Codificación: 10000100
 Operación: $(A) \leftarrow \text{COCIENTE DE } (A)/(B)$
 $(B) \leftarrow \text{RESTO DE } (A)/(B)$

DJNZ<byte>,<dirección relativa>

Función:	Disminuye y salta si no es igual a cero
Descripción:	DJNZ disminuye la ubicación indicada por el primer operando y se bifurca a la dirección indicada por el segundo operando si el valor resultante no es igual a 0. Un valor original de 00H causa un desbordamiento negativo al valor OFFH. Ninguna bandera se ve afectada. El destino de la bifurcación se calcula mediante una suma del valor de desplazamiento relativo con signo que haya en el último byte de la instrucción para el PC, después de incrementar el PC al primer byte de la siguiente instrucción.

La ubicación disminuida puede ser un registro o un byte directamente direccionado.

Nota: Cuando esta instrucción se utiliza para modificar un puerto de salida, el valor utilizado como el dato del puerto original se lee del latch de datos de salida, *no* de las terminales de entrada.

Ejemplo: Las ubicaciones en memoria RAM interna 40H, 50H, y 60H contienen los valores 01H, 70H, y 15H, respectivamente. Las instrucciones,

```
DJNZ 40H, ETIQUETA1
DJNZ 50H, ETIQUETA2
DJNZ 60H, ETIQUETA3
```

provocan un salto a la instrucción en ETIQUETA2 con los valores 00H, 6FH, y 15H en las 3 ubicaciones en RAM. El primer salto *no se realiza*, debido a que el resultado fue 0.

Esta instrucción proporciona una manera simple para ejecutar un ciclo de programa cierto número de veces, o para agregar un retraso moderado (de 2 a 512 ciclos de máquina) con una sola instrucción. Las instrucciones,

```
MOV    R2, #8
DISPARA: CPL   P1.7
          DJNZ  R2, DISPARA
```

disparan a P1.7 ocho veces, lo que genera cuatro pulsos de salida en el bit 7 del puerto 1. Cada pulso tiene duración de tres ciclos de máquina, dos para la instrucción DJNZ y uno más para alterar la terminal.

DJNZ Rn,rel

Bytes:	2
Ciclos:	2
Codificación:	11011rrr eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 2$ $(Rn) \leftarrow (Rn) - 1$ IF $(Rn) \neq 0$ THEN $(PC) \leftarrow (PC) + \text{byte_2}$

DJNZ directa,rel

Bytes:	3
Ciclos:	2

Codificación: 11010101 aaaaaaaaaaaaaaaaaaaaa
 Operación: $(PC) \leftarrow (PC) + 2$
 $(directa) \leftarrow (directa) - 1$
 $IF (directa) <> 0$
THEN
 $(PC) \leftarrow (PC) + \text{byte_2}$

INC <byte>

Función:	Incremento
Descripción:	INC incrementa la variable indicada por 1. Un valor original de 0FFH causa un desbordamiento a 00H. Ninguna bandera se ve afectada. Se permiten tres modos de direccionamiento: por registros, directo, o indirecto por registros.
Nota:	Cuando esta instrucción se utiliza para modificar un puerto de salida, el valor utilizado como el dato del puerto original es del latch de datos de salida, <i>no</i> de las terminales de entrada.

Ejemplo:

El registro 0 contiene el valor 7EH (01111110B). Las ubicaciones en memoria RAM interna 7EH y 7FH contienen los valores 0FFH y 40H, respectivamente. Las instrucciones,

```
INC    @R0
INC    R0
INC    @R0
```

dejan al registro 0 con el valor de 7FH y a las ubicaciones en memoria RAM interna 7EH y 7FH con los valores de 00H y 41H, respectivamente.

INC A

Bytes: 1
 Ciclos: 1
 Codificación: 00000100
 Operación: $(A) \leftarrow (A) + 1$

INC Rn

Bytes: 1
 Ciclos: 1
 Codificación: 00001rrr
 Operación: $(Rn) \leftarrow (Rn) + 1$

INC directa

Bytes: 2
 Ciclos: 1
 Codificación: 00000101 aaaaaaaaaa
 Operación: (directa) \leftarrow (directa) + 1

INC @Ri

Bytes: 1
 Ciclos: 1
 Codificación: 0000011i
 Operación: ((Ri)) \leftarrow ((Ri)) + 1

INC DPTR

Función: Incrementa el apuntador de datos
 Descripción: Incrementa el apuntador de datos de 16 bits en 1. Se realiza un incremento de 16 bits (módulo 2^{16}); un desbordamiento del byte de orden inferior del apuntador de datos (DPL) del valor OFFH a 00H incrementa el byte de orden superior (DPH). Ninguna bandera se ve afectada.
 Ejemplo: Este es el único registro de 16 bits que puede incrementarse.
 Los registros DPH y DPL contienen los valores 12H y 0FEH, respectivamente. Las instrucciones,

```
INC DPTR
INC DPTR
INC DPTR
```

cambian DPH y DPL a 13H y 01H.

Bytes: 1
 Ciclos: 2
 Codificación: 10100011
 Operación: (DPTR) \leftarrow (DPTR) + 1

JB bit, rel

Función: Salta si el bit está activado
 Descripción: Si el bit indicado es igual a 1, la instrucción salta a la dirección indicada; de lo contrario, procede con la siguiente instrucción. El destino de la bifurcación se calcula mediante una suma del desplazamiento relativo con signo en el tercer byte de la instrucción con el PC, después de

incrementar el PC al primer byte de la siguiente instrucción. *El bit verificado no se modifica.* Ninguna bandera se ve afectada.

Ejemplo: El dato presente en el puerto 1 de entrada es 11001010B. El acumulador contiene el valor 56H (01010110B). Las instrucciones,

```
JB P1.2, ETIQUETA1
JB ACC.2, ETIQUETA2
```

provocan que la ejecución del programa se bifurque hacia la instrucción en ETIQUETA2.

Bytes:	3
Ciclos:	2
Codificación:	0100000 bbbbbbbb eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 3$
	IF (bit) + 1
	THEN
	$(PC) \leftarrow (PC) + \text{byte_2}$

JBC bit,rel

Función:	Salta si el bit está activado y después borra el bit
Descripción:	Si el bit indicado es igual a 1, la instrucción borra el bit y salta a la dirección indicada; de lo contrario, procede con la siguiente instrucción. <i>El bit no se borra si ya es igual a 0.</i> El destino de la bifurcación se calcula mediante una suma del desplazamiento relativo con signo en el tercer byte de la instrucción con el PC, después de incrementar el PC al primer byte de la siguiente instrucción. Ninguna bandera se ve afectada.

Nota: Cuando esta instrucción se utiliza para modificar un puerto de salida, el valor utilizado como el dato del puerto original se lee del latch de datos de salida, *no* de las terminales de entrada.

Ejemplo: El acumulador contiene el valor 56H (01010110B). Las instrucciones,

```
JBC ACC.3, ETIQUETA1
JBC ACC.2, ETIQUETA2
```

provocan que la ejecución del programa continúe en la instrucción identificada por ETIQUETA 2, y el acumulador se modifica al valor 52H (01010010B).

Bytes:	3
Ciclos:	2
Codificación:	00010000 bbbbbbbb eeeeeeee

Operación: $(PC) \leftarrow (PC) + 3$
 $IF(bit) = 1$
THEN
 $(bit) \leftarrow 0$
 $(PC) \leftarrow (PC) + byte_2$

JC rel

Función:	Salta si la bandera de acarreo está activada
Descripción:	Si la bandera de acarreo está activada, la instrucción salta a la dirección indicada; de lo contrario, procede con la siguiente instrucción. El destino de la bifurcación se calcula mediante una suma del desplazamiento relativo con signo en el segundo byte de la instrucción con el PC, después de incrementar dos veces al PC. Ninguna bandera se ve afectada.
Ejemplo:	La bandera de acarreo está en cero. Las instrucciones,

```
JC    ETIQUETA1
CPL   C
JC    ETIQUETA2
```

activan la bandera de acarreo, y causan que la ejecución del programa continúe en la instrucción identificada por ETIQUETA2.

Bytes:	2
Ciclos:	2
Codificación:	01000000 eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 2$
	$IF(C) = 1$
	THEN
	$(PC) + (PC) + byte_2$

JMP <destino> (Consulte SJMP, AJMP o LJMP)**JMP@A+DPTR**

Función:	Salto indirecto
Descripción:	Suma el contenido sin signo de 8 bits del acumulador con el apuntador de 16 bits, y carga el resultado de la suma al contador de programa. Este resultado es la dirección para las subsiguientes búsquedas de instrucciones. Se realiza una suma de 16 bits (módulo 2^{16}): un acarreo de los ocho bits de orden inferior se propaga a lo largo de los bits de orden superior. No se alteran ni el acumulador ni el apuntador de datos. Ninguna bandera se ve afectada.

Ejemplo: El acumulador contiene un número par de 0 a 6. Las siguientes instrucciones generan una bifurcación a una de cuatro instrucciones AJMP en una tabla de saltos empezzando en TBL_SALTO:

```

MOV DPTR, #TBL_SALTO
JMP @A + DPTR
TBL_SALTO: AJMP ETIQUETA0
            AJMP ETIQUETA1
            AJMP ETIQUETA2
            AJMP ETIQUETA3

```

Si el acumulador es igual a 04H cuando inicia esta secuencia, la ejecución del programa se bifurca a ETIQUETA2. Recuerde que AJMP es una instrucción de dos bytes, así que la instrucción de salto empieza en una dirección, ignora a otra, continúa en la siguiente, y así sucesivamente.

Bytes:	1
Ciclos:	2
Codificación:	01110011
Operación:	$(PC) \leftarrow (PC) + (A) + (DPTR)$

JNB bit,rel

Función:	Salta si el bit no está activado
Descripción:	Si el bit indicado es igual a 0, se genera una bifurcación a la dirección indicada; de lo contrario, procede con la siguiente instrucción. El destino de la bifurcación se calcula mediante una suma del desplazamiento relativo con signo en el tercer byte de la instrucción con el PC, después de incrementar el PC al primer byte de la siguiente instrucción. <i>El bit verificado no se modifica.</i> Ninguna bandera se ve afectada.
Ejemplo:	Los datos presentes en el puerto 1 de entrada son 110010108. El acumulador contiene el valor 56H (01010110B). Las instrucciones,

```

JNB P1.3, ETIQUETA1
JNB ACC.3, ETIQUETA2

```

provocan que la ejecución del programa continúe en la instrucción en ETIQUETA2.

Bytes:	3
Ciclos:	2
Codificación:	00110000 bbbbbbbb eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 3$
	IF (bit) = 0

THEN

$$(PC) \leftarrow (PC) + \text{byte_2}$$

JNC rel

Función:	Salta si la bandera de acarreo no está activada
Descripción:	Si la bandera de acarreo es igual a 0, genera una bifurcación a la dirección indicada; de lo contrario, procede con la siguiente instrucción. El destino de la bifurcación se calcula mediante una suma del desplazamiento relativo con signo en el segundo byte de la instrucción con el PC, después de incrementar dos veces al PC para que apunte hacia la siguiente instrucción. La bandera de acarreo no se modifica.
Ejemplo:	La bandera de acarreo está activada. Las instrucciones,

```

JNC ETIQUETA1
CPL C
JNC ETIQUETA2

```

borran la bandera de acarreo y causan que la ejecución del programa continúe en la instrucción identificada mediante ETIQUETA2.

Bytes:	2
Ciclos:	2
Codificación:	01010000 eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 2$
	$IF(C) = 0$
	THEN
	$(PC) \leftarrow (PC) + \text{byte_2}$

JNZ rel

Función:	Salta si el acumulador no es igual a cero
Descripción:	Si cualquier bit del acumulador es igual a 1, genera una bifurcación a la dirección indicada; de lo contrario procede con la siguiente instrucción. El destino de la bifurcación se calcula mediante la suma del desplazamiento relativo con signo en el segundo byte de la instrucción con el PC, después de incrementar dos veces al PC. El acumulador no se modifica. Ninguna bandera se ve afectada.
Ejemplo:	El acumulador originalmente contiene el valor 00H. Las instrucciones,

```

JNZ    ETIQUETA1
INC    A
JNZ    ETIQUETA2

```

igualan el acumulador a 01H y continúan en ETIQUETA2.

Bytes:	2
Ciclos:	2
Codificación:	01110000 eeeeeeee

Operación: $(PC) \leftarrow (PC) + 2$
 IF $(A) <> 0$
 THEN
 $(PC) \leftarrow (PC) + \text{byte_2}$

JZ rel

Función:	Salta si el acumulador es igual a cero
Descripción:	Si todos los bits del acumulador son iguales a 0, genera una bifurcación a la dirección indicada; de lo contrario procede con la siguiente instrucción. El destino de la bifurcación se calcula mediante una suma del desplazamiento relativo con signo en el segundo byte de la instrucción con el PC, después de incrementar dos veces al PC. El acumulador no se modifica. Ninguna bandera se ve afectada.
Ejemplo:	Al principio, el acumulador contiene el valor 01H. Las instrucciones,

```
JZ ETIQUETA1
DEC A
JZ ETIQUETA2
```

cambian al acumulador al valor 00H y causan que la ejecución del programa continúe en la instrucción identificada mediante ETIQUETA2.

Bytes:	2
Ciclos:	2
Codificación:	01100000 eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 2$ $\text{IF } (A) = 0$ THEN $(PC) \leftarrow (PC) + \text{byte_2}$

LCALL dir16

Función:	Llamada larga a la subrutina
Descripción:	LCALL llama a una subrutina que se ubica en la dirección indicada. La instrucción agrega el valor 3 al contador de programa para generar la dirección de la siguiente instrucción y entonces almacena el resultado de 16 bits en la pila (byte inferior primero), e incrementa el apuntador de pila por 2. Los bytes de orden superior e inferior del PC se cargan, respectivamente, con los bytes segundo y tercero de la instrucción LCALL. La ejecución del programa continúa con la instrucción en dicha dirección. La subrutina puede, por lo tanto, iniciar en cualquier parte de los 64K-byte de espacio de direcciones de memoria de programa. Ninguna bandera se ve afectada.

Ejemplo: Al principio, el apuntador de pila es igual a 07H. La etiqueta “SUBRTN” se asigna a la ubicación en memoria de programa 1234H. Después de ejecutar la instrucción,

LCALL SUBRTN

en la ubicación 0123H, el apuntador de pila contiene el valor 09H, las ubicaciones en memoria RAM interna 08H y 09H contienen los valores 26H y 01H, respectivamente, y el PC contiene el valor 1234H.

Bytes: 3

Ciclos: 2

Codificación: 00010010 aaaaaaaaaaaaaaaaaaa

Nota: El segundo byte contiene los bits de dirección 15 a 8, y el tercer byte contiene los bits de dirección 7 a 0.

Operación: $(PC) \leftarrow (PC) + 3$

$(SP) \leftarrow (SP) + 1$

$(SP) \leftarrow (PC7 - PC0)$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC15 - PC8)$

$(PC) \leftarrow \text{dir15} - \text{dir0}$

LJMP dir16

Función: Salto largo

Descripción: LJMP carga los bytes de orden superior e inferior del PC (respectivamente) con los byte segundo y tercero de la instrucción, y con esto provoca una bifurcación incondicional hacia la dirección indicada. El destino, por lo tanto, puede estar en cualquier parte de los 64K del espacio de direcciones de memoria de programa. Ninguna bandera se ve afectada.

Ejemplo: La etiqueta “DIRSALTO” se asigna a la instrucción en la ubicación 1234H de la memoria de programa. La instrucción,

LJMP DIRSALTO

en la ubicación 0123H carga el contador de programa con el valor 1234H.

Bytes: 3

Ciclos: 2

Codificación: 00010010 aaaaaaaaaaaaaaaaaaa

Nota: El segundo byte contiene los bits de dirección 15 a 8, y el tercer byte contiene los bits de dirección 7 a 0.

Operación: $(PC) \leftarrow \text{dir15} - \text{dir0}$

MOV<byte destino>,<byte origen>

Función:	Transfiere una variable de byte
Descripción:	La variable de byte indicada por el segundo operando se copia a la ubicación especificada por el primer operando. El byte de origen no se ve afectado. Ningún registro ni bandera se ven afectados.
	Esta es definitivamente la operación más flexible. Se permiten quince combinaciones de modo de direccionamiento para el origen y el destino.
Ejemplo:	La ubicación en RAM interna 30H contiene el valor 40H. El valor de la ubicación 40H en RAM es 10H. El dato presente en el puerto 1 de entrada es 11001010B (0CAH). Las instrucciones,

```

MOV R0 , #30H      ;R0 ← 30H
MOV A, @R0          ;A ← 40H
MOV R1, A;          ;R1 ← 40H
MOV B, @R1          ;B ← 10H
MOV @R1, P1         ;RAM (40H)
                    ; ← 0CAH
MOV P2, P1          ;P2 ← 0CAH

```

dejan los valores 30H en el registro 0, 40H tanto en el acumulador como en el registro 1, 10H en el registro B, y 0CAH (11001010B) en la ubicación 40H en RAM y en el puerto 2 de salida.

MOV A,Rn

Bytes:	1
Ciclos:	1
Codificación:	11101rrr
Operación:	(A) ← (Rn)

MOV A,directa

Bytes:	2
Ciclos:	1
Codificación:	11100101 aaaaaaaaa
Operación:	(A) ← (directa)

Nota: MOV A,ACC no es una instrucción válida.

MOV A,@Ri

Bytes:	1
Ciclos:	1

Codificación: 1110011i
 Operación: $(A) \leftarrow ((R_i))$

MOV A,#datos

Bytes: 2
 Ciclos: 1
 Codificación: 01110100 dddddddd
 Operación: $(A) \leftarrow \#datos$

MOV Rn,A

Bytes: 1
 Ciclos: 1
 Codificación: 11111rrr
 Operación: $(R_n) \leftarrow (A)$

MOV Rn,directa

Bytes: 2
 Ciclos: 2
 Codificación: 10101rrr
 Operación: $(R_n) \leftarrow (\text{directa})$

MOV Rn,#datos

Bytes: 2
 Ciclos: 1
 Codificación: 01111rrr dddddddd
 Operación: $(R_n) \leftarrow \#datos$

MOV directa,A

Bytes: 2
 Ciclos: 1
 Codificación: 11110101 aaaaaaaaaa
 Operación: $(\text{directa}) \leftarrow A$

MOV directa,Rn

Bytes: 2
 Ciclos: 2
 Codificación: 10001rrr aaaaaaaaa
 Operación: (directa) \leftarrow (Rn)

MOV directa,directa

Bytes: 3
 Ciclos: 2
 Codificación: 10000101 aaaaaaaaa aaaaaaaaa
Nota: El segundo byte contiene la dirección de origen;
 el tercer byte contiene la dirección destino.
 Operación: (directa) \leftarrow (directa)

MOV directa,@Ri

Bytes: 2
 Ciclos: 2
 Codificación: 1000011i aaaaaaaaa
 Operación: (directa) \leftarrow ((Ri))

MOV directa,#datos

Bytes: 3
 Ciclos: 2
 Codificación: 01110101 aaaaaaaaa dddddddd
 Operación: (directa) \leftarrow #datos

MOV @Ri,A

Bytes: 1
 Ciclos: 1
 Codificación: 1111011i
 Operación: ((Ri)) \leftarrow A

MOV @R_i,directa

Bytes: 2
 Ciclos: 2
 Codificación: 1010011i aaaaaaaaa
 Operación: ((R_i) ← (directa))

MOV @R_i,#datos

Bytes: 2
 Ciclos: 1
 Codificación: 0111011i dddddddd
 Operación: ((R_i) ← #datos)

MOV <bit destino>,<bit origen>

Función: Transfiere variable de bit
 Descripción: La variable booleana indicada por el segundo operando se copia a la ubicación especificada por el primer operando. Uno de los operandos debe ser la bandera de acarreo; el otro puede ser cualquier bit directamente direccionable. Ningún otro registro o bandera se ven afectados.
 Ejemplo: La bandera de acarreo está activada originalmente. El dato presente en el puerto 2 de entrada es 11000101B. El dato previamente enviado al puerto 1 de salida es el valor 35H (00110101B). Las instrucciones,

```
MOV P1.3,C
MOV C,P3.3
MOV P1.2,C
```

dejan limpio al acarreo y cambian el puerto 1 al valor 39H (00111001B).

MOV C,bit

Bytes: 2
 Ciclos: 1
 Codificación: 10100010 bbbbbbbb
 Operación: (C) ← (bit)

MOV bit,C

Bytes: 2
 Ciclos: 2

Codificación: 10010010 dirección de bit
 Operación: (bit) ← (C)

MOV DPTR,#datos16

Función:	Carga el apuntador de datos con una constante de 16 bits
Descripción:	El apuntador de datos se carga con la constante de 16 bits indicada. La constante de 16 bits se ubica en los bytes segundo y tercero de la instrucción. El segundo byte (DPH) es el byte de orden superior, mientras que el tercer byte (DPL) contiene el byte de orden inferior. Ninguna bandera se ve afectada.
Ejemplo:	La instrucción, MOV DPTR, #1234H carga el valor 1234H al apuntador de datos: DPH contiene el valor 12H y DPL contiene el valor 34H.
Bytes:	3
Ciclos:	2
Codificación:	10010000 dddddddd dddddddd
	<i>Nota:</i> El segundo byte contiene los bits de datos inmediatos 15 a 8, y el byte 3 contiene los bits 7 a 0.
Operación:	(DPTR) ← #datos16

MOVC A,@A+<registro base>

Función:	Transfiere un byte de código o un byte constante
Descripción:	Las instrucciones MOVC cargan al acumulador con un byte de código o un byte constante de la memoria de programa. La dirección del byte obtenido es la suma del contenido original de 8 bits sin signo en el acumulador y del contenido de un registro base de 16 bits, el cual puede ser el apuntador de datos o el PC. En el segundo caso, el PC se incrementa a la dirección de la siguiente instrucción antes de agregarlo al acumulador; en caso contrario, el registro base no se altera. Se realiza una suma de 16 bits para que el acarreo de los 8 bits de orden inferior pueda propagarse a lo largo de los bits de orden superior. Ninguna bandera se ve afectada.
Ejemplo:	El acumulador contiene un valor ubicado entre 0 y 3. La siguiente subrutina convierte el valor en el acumulador a 1 de 4 valores definidos por la directiva DB (definición de byte).

```

REL_PC:      INC A
              MOVC A,@A + PC
              RET
              DB 66H
              DB 77H
  
```

DB 88H
DB 99H

Si la subrutina se llama con el acumulador igual a 01H, regresa con el valor 77H en el acumulador. Se necesita la instrucción INC A antes de la instrucción MOVC para “sobrepasar” la instrucción RET por encima de la tabla. Si varios bytes de código separan la instrucción MOV de la tabla, el número correspondiente debe agregarse al acumulador.

MOVC A,@A+DPTR

Bytes: 1
Ciclos: 2
Codificación: 10010011
Operación: $(A) \leftarrow ((A) + (DPTR))$

MOVC A,@A+PC

Bytes: 1
Ciclos: 2
Codificación: 10000011
Operación: $(PC) \leftarrow (PC) + 1$
 $(A) \leftarrow ((A) + (PC))$

MOVX <byte destino>,<byte origen>

Función:	Transferencia externa
Descripción:	Las instrucciones MOVX transfieren datos entre el acumulador y un byte de memoria externa para datos; es por esto que se agrega una “X” al término MOV. Existen dos tipos de instrucciones, y la única diferencia entre ellas es si proporcionan una dirección indirecta de 8 o de 16 bits a la memoria RAM externa para datos.
	En el primer tipo de instrucción, el contenido de R0 o R1 en el banco de registros actual proporciona una dirección de 8 bits multiplexada con los datos en P0. Ocho bits son suficientes para implementar la decodificación de expansión de E/S externa o para un arreglo relativamente pequeño en memoria RAM. Podemos utilizar cualquiera de las terminales de puerto de salida en arreglos un poco más grandes para enviar los bits de dirección de orden superior como salida. Estas terminales estarían controladas por una instrucción de salida procediendo a la instrucción MOVX.
	En el segundo tipo de instrucción MOVX, el apuntador de datos genera una dirección de 16 bits. P2 envía como salida los ocho bits de dirección de orden superior (el contenido de DPH), mientras que P0 multiplexa los

ocho bits de orden inferior (DPL) con los datos. El registro con funciones especiales P2 retiene su contenido anterior, mientras que los búferes de salida de P2 emiten el contenido de DPH. Esta forma resulta más rápida y eficiente cuando accedemos a arreglos muy grandes (de hasta 64K bytes), ya que no necesitamos de ninguna instrucción adicional adicional para configurar los puertos de salida.

Es posible que en ciertas situaciones se puedan mezclar los dos tipos de la instrucción MOVX. Un arreglo en RAM grande, en donde P2 controla las líneas de orden superior de su dirección, puede direccionarse mediante el apuntador de datos o mediante código para enviar los bits de dirección de orden superior a P2 como salida, seguido de una instrucción MOVX utilizando R0 o R1.

- Ejemplo: Una memoria RAM externa de 256 bytes utilizando líneas de dirección y de datos multiplexadas (por ejemplo, un 8155 RAM/E/S/TEMPORIZADOR de Intel) está conectada al puerto 0 del 8051. El puerto 3 proporciona las líneas de control para la RAM externa. Los puertos 1 y 2 se utilizan para E/S normal. Los registros 0 y 1 contienen los valores 12H y 34H. La ubicación 34H en la RAM externa contiene el valor 56H. La secuencia de instrucciones,

```
MOVX A, @R1
MOVX @R0, A
```

copia el valor 56H tanto al acumulador como a la ubicación 12H en memoria RAM externa.

MOVX A,@Ri

Bytes:	1
Ciclos:	2
Codificación:	1110001i
Operación:	(A) ← ((Ri))

MOVX A,@DPTR

Bytes:	1
Ciclos:	2
Codificación:	11100000
Operación:	(A) ← ((DPTR))

MOVX @Ri,A

Bytes:	1
Ciclos:	2
Codificación:	11110011
Operación:	((Ri)) ← (A)

MOVX @DPTR,A

Bytes:	1
Ciclos:	2
Codificación:	11110000
Operación:	(DPTR) ← (A)

MUL AB

Función:	Multiplicación
Descripción:	MUL AB multiplica los enteros sin signo de 8 bits en el acumulador y en el registro B. El byte de orden inferior del producto de 16 bits queda en el acumulador y el byte de orden superior queda en el registro B. Si el producto es mayor a 255 (0FFH), la bandera de desbordamiento se activa; ésta se borra en el caso contrario. La bandera de acarreo siempre se borra.
Ejemplo:	Al principio, el acumulador contiene el valor 80 (50H). El registro B contiene el valor 160 (0A0H). La instrucción,

MUL AB

resulta en un producto de 12,800 (3200H), así que B cambia a 32H (00110010B) y el acumulador se borra. La bandera de desbordamiento se activa, y el acarreo se borra.

Bytes:	1
Ciclos:	4
Codificación:	10100100
Operación:	(B) ← BYTE SUPERIOR DE (A) × (B) (A) ← BYTE INFERIOR DE (A) × (B)

NOP

Función:	No hay operación
Descripción:	La ejecución continúa en la siguiente instrucción. Ningún registro o bandera, a excepción del PC, se ven afectados.
Ejemplo:	En el bit 7 del puerto 2, deseamos producir un pulso de salida a nivel bajo que dure exactamente cinco ciclos. Una secuencia simple de SETB/CLR genera un pulso de un ciclo, así que debemos insertar cuatro ciclos adicionales. Esto puede llevarse a cabo (si suponemos que las interrupciones no están habilitadas) con las instrucciones,

CLR P2.7
NOP

```

NOP
NOP
NOP
SETB P2.7

```

Bytes:	1
Ciclos:	1
Codificación:	00000000
Operación:	$(PC) \leftarrow (PC) + 1$

ORL <byte destino>,<byte origen>

Función:	OR lógico para variables de byte
Descripción:	ORL realiza una operación de OR lógico orientada a bits entre las variables indicadas y almacena el resultado en el byte de destino. Ninguna bandera se ve afectada.
	Los dos operandos permiten seis combinaciones de modo de direccionamiento. Cuando el destino es el acumulador, el origen puede utilizar un direccionamiento por registros, directo, indirecto por registros, o inmediato; cuando el destino es una dirección directa, el origen puede ser el acumulador o datos inmediatos.
	<i>Nota:</i> Cuando esta instrucción se utiliza para modificar un puerto de salida, el valor utilizado como el dato del puerto original se lee del latch de datos de salida, <i>no</i> de las terminales de entrada.
Ejemplo:	Si el acumulador contiene el valor 0C3H (11000011B) y R0 contiene el valor 55H (01010101) entonces la instrucción,

ORL A, R0

deja el valor 0D7H (11010111B) en el acumulador. Cuando el destino es un byte directamente direccionable, la instrucción puede activar combinaciones de bits en cualquier ubicación en RAM o registro en hardware. El patrón de bits a activar está determinado por un byte de enmascarado que puede ser ya sea un valor de datos constante en la instrucción o una variable calculada en el acumulador al tiempo de ejecución. La instrucción,

ORL P1,00110010B

activa los bits 5, 4 y 1 del puerto 1 de salida.

ORL A,Rn

Bytes:	1
Ciclos:	1

Codificación: 01001rrr
 Operación: (A) ← (A) OR (Rn)

ORL A,direct

Bytes: 2
 Ciclos: 1
 Codificación: 01000101 aaaaaaaaaa
 Operación: (A) ← (A) OR (direct)

ORL A,@Ri

Bytes: 1
 Ciclos: 1
 Codificación: 0100011i
 Operación: (A) ← (A) OR ((Ri))

ORL A,#datos

Bytes: 2
 Ciclos: 1
 Codificación: 01000100 dddddddd
 Operación: (A) ← (A) OR #datos

ORL directa,A

Bytes: 2
 Ciclos: 1
 Codificación: 01000010 aaaaaaaaaa
 Operación: (directa) ← (directa) OR (A)00

ORL directa,#datos

Bytes: 3
 Ciclos: 2
 Codificación: 01000011 aaaaaaaaaa dddddddd
 Operación: (directa) ← (directa) OR #datos

ORL C,<bit origen>

Función:	OR lógico para variables de bit
Descripción:	Activa la bandera de acarreo si el valor booleano es un 1 lógico; en caso contrario, deja el acarreo en su estado actual. Un signo de división (/) precediendo al operando en el lenguaje ensamblador indica que se utiliza el complemento lógico del bit direccionado como el valor de origen, pero el bit de origen no se ve afectado. Ninguna otra bandera se ve afectada.
Ejemplo:	Activa la bandera de acarreo si, y sólo si, P1.0 = 1, ACC.7 = 1, u OV = 0.

```

MOV C,P1.0      ;CARGA CY CON LA TERMINAL DE
                  ENTRADA P1.0
ORL C,ACC.7     ;OR DE CY CON EL BIT 7 DEL ACC
ORL C,/OV       ;OR DE CY CON EL INVERSO DE OV

```

ORL C,bit

Bytes:	2
Ciclos:	2
Codificación:	01110010 bbbbbbbb
Operación:	(C) ← –(C) OR (bit)

ORL C,/bit

Bytes:	2
Ciclos:	2
Codificación:	10100000 bbbbbbbb
Operación:	(C) ← (C) OR NOT(bit)

POP directa

Función:	Recuperación de la pila
Descripción:	Se lee el contenido de la ubicación en RAM interna direccionada por el apuntador de pila, y el apuntador de pila se disminuye por 1. Después el valor leído se transfiere al byte directamente direccionado indicado. Ninguna bandera se ve afectada.
Ejemplo:	El apuntador de pila originalmente contiene el valor 32H, y las ubicaciones en RAM interna 30H hasta la 32H contienen los valores 20H, 23H, y 01H, respectivamente. Las instrucciones,

```

POP      DPH
POP      DPL

```

dejan el apuntador de pila igual al valor 30H y el apuntador de datos en 0123H. En este punto, la instrucción,

POP SP

deja el apuntador de pila establecido en 20H. Observe que, en este caso especial, el apuntador de pila se decrementa a 2FH antes de cargarse con el valor que se sacó (20H).

Bytes:	2
Ciclos:	2
Codificación:	11010000 aaaaaaaa
Operación:	(directa) \leftarrow ((SP)) (SP) \leftarrow (SP) - 1

PUSH directa

Función:	Meter datos en la pila
Descripción:	El apuntador de pila se incrementa por 1. El contenido de la variable indicada se copia entonces en la ubicación de RAM interna direccionalizada por el apuntador de pila. En caso contrario no se afectan las banderas.
Ejemplo:	Al entrar en una rutina de interrupción, el apuntador de pila contiene el valor 09H. El apuntador de datos mantiene el valor 0123H. Las instrucciones,

PUSH DPL
PUSH DPH

dejan el valor 0BH en el apuntador de pila y almacenan los valores 23H y 01H en las ubicaciones en RAM interna 0AH y 0BH, respectivamente.

Bytes:	2
Ciclos:	2
Codificación:	11000000 aaaaaaaa
Operación:	(SP) \leftarrow (SP) + 1 ((SP)) \leftarrow (directa)

RET

Función:	Regresar de la subrutina
Descripción:	RET recupera los bytes de orden superior e inferior del PC en forma sucesiva de la pila, y disminuye el apuntador de pila en 2. La ejecución del programa continúa en la dirección resultante, la cual es por lo general la instrucción que va inmediatamente después de una instrucción ACALL o LCALL. Ninguna bandera se ve afectada.

Ejemplo: Al principio, el apuntador de pila contiene el valor 0BH. Las ubicaciones en RAM interna 0AH y 0BH contienen los valores 23H y 01H, respectivamente. La instrucción,

RET

deja el valor 09H en el apuntador de pila. La ejecución del programa continúa en la ubicación 0123H

Bytes:	1
Ciclos:	2
Codificación:	00100010
Operación:	$(PC1-PC8) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$ $(PC7-PC0) \leftarrow ((SP))$ $(SP) \leftarrow (SP) - 1$

RETI

Función: Regresar de una interrupción

Descripción: RETI recupera los bytes de orden superior e inferior del PC en forma sucesiva de la pila, y restablece la lógica de interrupciones para aceptar interrupciones adicionales al mismo nivel de prioridad que el que se acaba de procesar. El apuntador de pila disminuye en 2. Ningún otro registro se ve afectado; la PSW no se restablece automáticamente al estado que tenía antes de la interrupción. La ejecución del programa continúa en la dirección resultante, la cual es por lo general una instrucción que va inmediatamente después del punto en donde se detecta la petición de interrupción. Si una interrupción del mismo nivel (o de nivel más bajo) está pendiente cuando la instrucción RETI se ejecuta, entonces una instrucción se ejecuta antes de que la interrupción pendiente sea procesada.

Ejemplo: Al principio, el apuntador de pila contiene el valor 0BH. Se detecta una interrupción durante la instrucción que termina en la ubicación 0123H. Las ubicaciones en RAM interna 0AH y 0BH contienen los valores 23H y 01H, respectivamente. La instrucción,

RETI

deja el valor 09H en el apuntador de pila y regresa la ejecución del programa a la ubicación 0123H.

Bytes:	1
Ciclos:	2
Codificación:	00110010
Operación:	$(PC15-PC8) \leftarrow ((SP))$

$$\begin{aligned}
 (\text{SP}) &\leftarrow (\text{SP}) - 1 \\
 (\text{PC7} - \text{PCO}) &\leftarrow ((\text{SP})) \\
 (\text{SP}) &\leftarrow (\text{SP}) - 1
 \end{aligned}$$
RL A

Función:	Desplaza el acumulador hacia la izquierda
Descripción:	En el acumulador los ocho bits son desplazados un bit hacia la izquierda. El bit 7 se desplaza hacia la posición del bit 0. Ninguna otra bandera es afectada.
Ejemplo:	El acumulador contiene el valor 0C5H (11000101B). La instrucción,

RL A

deja el valor 8BH (10001011B) en el acumulador sin afectar al acarreo.

Bytes:	1
Ciclos:	1
Codificación:	00100011
Operación:	$(A_n + 1) \leftarrow (A_n), n = 0-6$ $(A_0) \leftarrow (A_7)$

RLC A

Función:	Desplaza el acumulador hacia la izquierda a través de la bandera de acarreo
Descripción:	Los ocho bits en el acumulador y la bandera de acarreo se acarrean juntos un bit hacia la izquierda. El bit 7 se transfiere a la bandera de acarreo; el estado original de la bandera de acarreo se transfiere a la posición del bit 0. Ninguna otra bandera se ve afectada.
Ejemplo:	El acumulador contiene el valor 0C5H (11000101B), y el acarreo es igual a 0. La instrucción,

RLC A

deja el valor 8BH (10001011B) en el acumulador con el acarreo activado.

Bytes:	1
Ciclos:	1
Codificación:	00110011
Operación:	$(A_n + 1) \leftarrow (A_n), n = 0-6$

$$(A0) \leftarrow (C)$$

$$(C) \leftarrow (A7)$$
RR A

Función:	Desplaza el acumulador hacia la derecha
Descripción:	Los ocho bits en el acumulador se desplazan un bit hacia la derecha. El bit 0 se desplaza hacia la posición del bit 7. Ninguna bandera se ve afectada.
Ejemplo:	El acumulador contiene el valor 0C5H (11000101B) sin afectar el acarreo. La instrucción,

RR A

deja el valor 0E2H (11100010B) en el acumulador sin afectar al acarreo.

Bytes:	1
Ciclos:	1
Codificación:	00000011
Operación:	$(An) \leftarrow (An + 1), n = 0 - 6$ $(A7) \leftarrow (A0)$

RRC A

Función:	Desplaza el acumulador hacia la derecha a través de la bandera de acarreo
Descripción:	Los ocho bits en el acumulador y la bandera de acarreo se desplazan juntos un bit hacia la derecha. El bit 0 se transfiere a la bandera de acarreo; el valor original de la bandera de acarreo se transfiere a la posición del bit 7. Ninguna otra bandera se ve afectada.
Ejemplo:	El acumulador contiene el valor 0C5H (11000101B), y el acarreo es igual a 0. La instrucción,

RRC A

deja el valor 62H (01100010B) en el acumulador y activa el acarreo.

Bytes:	1
Ciclos:	1
Codificación:	00010011
Operación:	$(An) \leftarrow (An + 1), n = 0 - 6$ $(A7) \leftarrow (C)$ $(C) \leftarrow (A0)$

SETB <bit>

Función:	Activa un bit
Descripción:	SETB activa el bit indicado al valor 1. SETB puede operar en la bandera de acarreo o en cualquier bit directamente direccionable. Ninguna otra bandera se ve afectada.
Ejemplo:	La bandera de acarreo está en cero. El puerto 1 de salida contiene el valor 34H (00110100B). Las instrucciones,

```
SETB C
SETB P1.0
```

dejan la bandera de acarreo igual a 1 y cambian la salida de datos en el puerto 1 a 35H (00110101B).

SETB C

Bytes:	1
Ciclos:	1
Codificación:	11010011
Operación:	(C) ← 1

SETB bit

Bytes:	2
Ciclos:	1
Codificación:	11010010 bbbbbbbb
Operación:	(bit) ← 1

SJMP rel

Función:	Salto corto
Descripción:	El control del programa se bifurca incondicionalmente a la dirección indicada. El destino de la bifurcación se calcula mediante una suma del desplazamiento con signo en el segundo byte de la instrucción con el PC, después de incrementar dos veces el PC. Por lo tanto, el rango de destinos permitido es desde los 128 bytes precediendo a esta instrucción hasta los 127 bytes que le siguen.
Ejemplo:	La etiqueta “DIRREL” se asigna a una instrucción en la ubicación en memoria de programa 0123H. La instrucción,

```
SJMP RELADR
```

se ensambla a la ubicación 0100H. Después de que se ejecuta la instrucción, el PC contiene el valor 0123H.

(Nota: Bajo las condiciones antes descritas, la instrucción que sigue de la instrucción SJMP está en la ubicación 0102H. Por lo tanto, el byte de desplazamiento de la instrucción es el desplazamiento relativo (0123H – 0102H = 21H). Descrito de otra manera, una instrucción SJMP con un desplazamiento de 0FEH es un ciclo infinito de una sola instrucción.

Bytes:	2
Ciclos:	2
Codificación:	10000000 eeeeeeee
Operación:	$(PC) \leftarrow (PC) + 2$ $(PC) \leftarrow (PC) + \text{byte_2}$

SUBB A,<byte origen>

Función:	Suma con préstamo
Descripción:	SUBB resta la variable indicada y la bandera de acarreo del acumulador, y deja el resultado en el acumulador. SUBB activa la bandera de acarreo (préstamo) si se requiere de un préstamo para el bit 7 y borra C en caso contrario. (Si C se activa <i>antes</i> de ejecutar una instrucción SUBB, esto indica que el préstamo es necesario para el paso anterior en una resta de precisión múltiple, así que el acarreo se resta del acumulador junto con el operando de origen). AC se activa si se necesita un préstamo para el bit 3 y se borra en caso contrario. OV se activa cuando se necesita un préstamo hacia el bit 6 pero no hacia el bit 7, o hacia el bit 7 pero no hacia el bit 6.
	Cuando se restan enteros con signo, la bandera OV indica que se produjo un número negativo cuando un valor negativo se resta de un valor positivo, o que se produjo un número positivo cuando un número positivo se resta de un número negativo.
	El operando de origen permite cuatro modos de direccionamiento: por registro, directo, indirecto por registro, o inmediato.
Ejemplo:	El acumulador contiene el valor 0C9H (11001001B), el registro 2 contiene el valor 54H (01010100B), y la bandera de acarreo está activada. La instrucción,

SUBB A, R2

deja el valor 74H (01110100B) en el acumulador, y borra las banderas de acarreo y de AC pero deja activada la bandera OV.

Observe que 0C9H menos 54H es 75H. La diferencia entre este resultado y el anterior se debe a que la bandera de acarreo (préstamo) se activó antes de la operación. Si no se conoce el estado del acarreo antes de iniciar una resta de precisión simple o múltiple, el acarreo debe borrarse con una instrucción CLR C.

SUBB A,Rn

Bytes: 1
 Ciclos: 1
 Codificación: 10011rrr
 Operación: $(A) \leftarrow (A) - (C) - (R_n)$

SUBB A,direct

Bytes: 2
 Ciclos: 1
 Codificación: 10010101 aaaaaaaa
 Operación: $(A) \leftarrow (A) - (C) - (\text{directa})$

SUBB A,@Ri

Bytes: 1
 Ciclos: 1
 Codificación: 1001011i
 Operación: $(A) \leftarrow (A) - (C) - ((R_i))$

SUBB A,#datos

Bytes: 2
 Ciclos: 1
 Codificación: 10010100 dddddddd
 Operación: $(A) \leftarrow (A) - (C) - \#datos$

SWAP A

Función:	Intercambia los nibbles dentro del acumulador
Descripción:	SWAP A intercambia los nibbles de orden inferior y superior (campos de 4 bits) del acumulador (bits 3 a 0 y bits 7 a 4). La operación puede también considerarse como una instrucción de desplazamiento de 4 bits. Ninguna bandera se ve afectada.
Ejemplo:	El acumulador contiene el valor 0C5H (11000101B). La instrucción,

SWAP A

deja el valor 5CH (01011100B) en el acumulador.

Bytes:	1
Ciclos:	1
Codificación:	11000100
Operación:	(A3–A0) ↔ (A7–A4)

XCH A,<byte>

Función:	Intercambia el acumulador con la variable de byte
Descripción:	XCH carga el acumulador con el contenido de la variable indicada y al mismo tiempo escribe el contenido original del acumulador a dicha variable. El operando origen/destino puede utilizar el direccionamiento por registro, el directo o el indirecto por registro.
Ejemplo:	R0 contiene la dirección 20H. El acumulador contiene el valor 3FH (00111111B). La ubicación en la RAM interna 20H contiene el valor 75H (01110101B). La instrucción,

XCH A, @R0

deja el valor 3FH (00111111B) en la ubicación en RAM 20H y el valor 75H (01110101B) en el acumulador.

XCH A,Rn

Bytes:	1
Ciclos:	1
Codificación:	11001rr
Operación:	(A) ↔ (Rn)

XCH A,directa

Bytes:	2
Ciclos:	1
Codificación:	11000101 aaaaaaaaaa
Operación:	(A) ↔ (directa)

XCH A,@Ri

Bytes:	1
Ciclos:	1

Codificación: 1100011i
 Operación: $(A) \leftrightarrow ((Ri))$

XCHD A,@Ri

Función:	Intercambia un dígito
Descripción:	XCHD intercambia el nibble de orden inferior del acumulador (bits 0 a 3), el cual generalmente representa un dígito en hexadecimal o en BCD, con el nibble inferior de la ubicación en RAM interna direccionado indirectamente por el registro especificado. Los nibbles de orden superior (bits 7 a 4) de cada registro no se ven afectados. Ninguna bandera resulta afectada.
Ejemplo:	R0 contiene la dirección 20H. El acumulador contiene el valor 36H (00110110B). La ubicación en RAM interna 20H contiene el valor 75H (01110101B). La instrucción,

XCHD A,@R0

deja el valor 76H (01110110B) en la ubicación en RAM 20H y el valor 35H (00110101B) en el acumulador.

Bytes:	1
Ciclos:	1
Codificación:	1101011i
Operación:	$(A3-A0) \leftrightarrow ((Ri3-Ri0))$

XRL<byte destino>,<byte origen>

Función:	OR exclusivo lógico para variables de byte
Descripción:	XRL realiza una operación lógica de OR exclusivo orientada a bits entre las variables indicadas y almacena los resultados en el destino. Ninguna bandera se ve afectada.
	Los dos operandos permiten seis combinaciones de modo de direccionamiento. Cuando el destino es el acumulador, el origen puede utilizar un direccionamiento por registro, directo, indirecto por registro, o inmediato; cuando el destino es una dirección directa, el origen puede ser el acumulador o datos inmediatos.
	<i>Nota:</i> Cuando esta instrucción se utiliza para modificar un puerto de salida, el valor utilizado como el dato del puerto original se lee del latch de datos de salida, no de las terminales de entrada.
Ejemplo:	Si el acumulador contiene el valor 0C3H (11000011B) y el registro 0 contiene el valor 0AAH (10101010B), entonces la instrucción,

XRL A, R0

deja el valor 69H (01101001B) en el acumulador.

Cuando el destino es un byte directamente direccionado, la instrucción puede complementar combinaciones de bits en cualquier ubicación en RAM o registro en hardware. El patrón de los bits a complementar se determina en el acumulador al tiempo de ejecución.

La instrucción,

XRL P1, #00110001B

complementa los bits 5, 4 y 0 del puerto de salida 1.

XRL A,Rn

Bytes:	1
Ciclos:	1
Codificación:	01101rrr
Operación:	$(A) \leftarrow (A) \oplus (R_n)$

XRL A,direct

Bytes:	2
Ciclos:	1
Codificación:	01100101 aaaaaaaaaa
Operación:	$(A) \leftarrow (A) \oplus (\text{directa})$

XRL A,@Ri

Bytes:	1
Ciclos:	1
Codificación:	0110011i
Operación:	$(A) \leftarrow (A) \oplus ((R_i))$

XRL A,#data

Bytes:	2
Ciclos:	1
Codificación:	01100100 dddddddd
Operación:	$(A) \leftarrow (A) \oplus \#datos$

XRL direct,A

Bytes: 2
Ciclos: 1
Codificación: 01100010 aaaaaaaaa
Operación: (directa) \leftarrow (directa) \oplus (A)

XRL direct,#data

Bytes: 3
Ciclos: 2
Codificación: 01100011 aaaaaaaaa ddddddd
Operación: (directa) \leftarrow (directa) \oplus #datos



Registros con funciones especiales¹

Los registros del 8051 que tienen funciones especiales se muestran en el mapa de memoria de los SFR en la figura D-1. Las ubicaciones en blanco están reservadas para futuros productos y no se debe escribir en ellas. Los SFR identificados con un asterisco contienen los bits que están definidos como bits de modo o de control. En las siguientes páginas describiremos estos registros y sus definiciones de bit.

Algunos bits están identificados como “no implementados”. El software de usuario no debe escribir números 1 a estos bits, ya que pueden utilizarse en futuros productos MCS-51™ para invocar nuevas características. En ese caso, el valor inactivo o de reinicialización del nuevo bit será 0, y su valor activo será 1.

PCON (registro de control de energía)

Símbolo:

PCON

Función:

Control de energía y diversas características

Dirección de bit:

87H

Direccionable por bit : No

Resumen:

7	6	5	4	3	2	1	0
SMOD	-	-	-	GF1	GF0	PD	IDL

¹Adaptadas del documento *Controladores incrustados de 8 bits* (270645), Santa Clara, CA: Intel Corporation, 1991, con permiso de Intel Corporation.

Definiciones de bit:

Símbolo de bit	Descripción de bit
SMOD	Duplica la velocidad en baudios. Si el temporizador 1 se utiliza para generar la velocidad en baudios y SMOD = 1, la velocidad en baudios se duplica cuando el puerto serial se utiliza en los modos 1, 2 o 3 —No implementado; reservado para uso futuro. —No implementado; reservado para uso futuro. —No implementado; reservado para uso futuro.
GF1	Bit 1 de bandera de propósito general.
GF0	Bit 0 de bandera de propósito general.
PD	Bit de apagado. Al activar este bit se activa la operación de apagado en la versión de CMOS del 8051. ²
IDL	Bit de modo de descanso. Al activar este bit se activa la operación en el modo de inactividad en las versiones de CMOS del 8051. ²

8 Bytes						
F8						
F0	B					
E8						
E0	ACC					
D8						
D0	PSW*					
C8	T2CON*	RCAP2L	RCAP2H	TL2	TH2	
C0						
B8	IP*					
B0	P3					
A8	IE*					
A0	P2					
98	SCON*	SBUF				
90	P1					
88	TCON*	TMOD*	TL0	TL1	TH0	TH1
80	P0	SP	DPL	DPH		PCON*

↑

Direccional por bit

*SFRs que contienen bits de modo o de control

FIGURA D-1

Mapa de memoria de registro con funciones especiales.

²Si se escriben números 1 al PD y al IDL al mismo tiempo, PD tiene precedencia.

TCON (REGISTRO DE CONTROL DEL TEMPORIZADOR/CONTADOR)

Símbolo: TCON
 Función: Control del temporizador/contador
 Dirección de bit: 88H
 Direccionable por bit: Yes
 Resumen:

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Definiciones de bit:

Símbolo de bit	Posición	Dirección de bit	Descripción
TF1	TCON.7	8FH	Bandera de desbordamiento del temporizador 1. Se activa mediante el hardware cuando el temporizador/contador 1 se desborda; se borra mediante software o hardware cuando el procesador se vectoriza a la rutina de servicio de interrupción.
TR1	TCON.6	8EH	Bit de control de ejecución del temporizador 1. Se activa/borra mediante software para encender o apagar el temporizador/contador 1.
TF0	TCON.5	8DH	Bandera de desbordamiento del temporizador 0. (Consulte TF1)
TR0	TCON.4	8CH	Bit de control de ejecución del temporizador 0. (Consulte TR1)
IE1	TCON.3	8BH	Bandera de borde de interrupción externa 1. Se activa mediante hardware cuando se detecta un borde de la interrupción externa; se borra mediante hardware cuando se procesa la interrupción.
IT1	TCON.2	8AH	Bit de control del tipo de interrupción 1. Se activa/borra mediante software para especificar una interrupción externa disparada por borde de caída/nivel bajo.
IE0	TCON.1	89H	Bandera de borde de interrupción externa 0. (Consulte IE1)
IT0	TCON.0	88H	Bit de control del tipo de interrupción 0. (Consulte IT1)

SCON (REGISTRO DE CONTROL DEL PUERTO SERIAL)

Símbolo: SCON
 Función: Control del puerto serial
 Dirección de bit: 98H
 Direccionable por bit: Sí

TABLA D-1

SM0	SM1	Modo	Descripción	Velocidad en baudios
0	0	0	Registro de desplazamiento	$F^{OSC} \div 12^*$
0	1	1	UART de 8 bits	Variable
1	0	2	UART de 9 bits	$F^{OSC} \div 64$ o $F^{OSC} \div 32$
1	1	3	UART de 9 bits	Variable

*FOSC es la frecuencia de oscilación del 8051. Por lo general, se deriva de un origen de cristal de 12 MHz.

Resumen:

7	6	5	4	3	2	1	0
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Definiciones de bit:

Símbolo de bit	Posición	Dirección de bit	Descripción
SM0	SCON.7	9FH	Bit 0 del modo del puerto serial. (Consulte la tabla D-1)
SM1	SCON.6	9EH	Bit 1 del modo del puerto serial. (Consulte la tabla D-1)
SM2	SCON.5	9DH	Bit 2 del modo del puerto serial. Habilita la característica de comunicación entre múltiples procesadores en los modos 2 y 3. En el modo 2 o en el 3, si SM2 es igual a 1, entonces RI no se activará cuando el noveno bit de datos recibido (RB8) sea igual a 0. En el modo 1, si SM2 = 1, entonces RI no se activará si no se recibió un bit válido de paro. En el modo 0, SM2 deberá ser igual a 0.
REN	SCON.4	9CH	Habilitación del receptor. Se activa/borra mediante software para habilitar/deshabilitar la recepción.
TB8	SCON.3	9BH	Bit 8 de transmisión. El noveno bit que se transmite en los modos 2 y 3. Se activa/borra mediante software.
RB8	SCON.2	9AH	Bit 8 de recepción. En los modos 2 y 3, RB8 es el noveno bit recibido. En el modo 1, si SM2 = 0, RB8 es el bit de paro recibido. TB8 no se utiliza en el modo 0.
TI	SCON.1	99H	Interrumpe transmisión. Se activa mediante hardware al final del tiempo del octavo bit en el modo 0, o al inicio del bit de paro en los otros modos. Debe borrarse utilizando software.

RI	SCON.0	98H	Interrumpe recepción. Se activa mediante hardware al final del tiempo del octavo bit en el modo 0, o a la mitad del tiempo del bit de paro en los otros modos (consulte SM2 para ver la excepción). Debe borrarse utilizando software.
----	--------	-----	--

IE (REGISTRO DE HABILITACIÓN DE INTERRUPCIONES)

Símbolo: IE
 Función: Habilitación de interrupciones
 Dirección de bit: A8H
 Direccionable por bit: Sí
 Resumen:

7	6	5	4	3	2	1	0
EA	—	ET2	ES	ET1	EX1	ET0	EX0

Definiciones de bit:

Símbolo	Dirección	de bit	Descripción (1 = habilita, 0 = deshabilita)
EA	IE.7	AFH	Habilita/deshabilita todas las interrupciones. Si EA = 0, ninguna interrupción será aceptada. Si EA = 1, cada fuente de interrupción se habilita o deshabilita individualmente al activar o borrar su bit de habilitación.
—	IE.6	AEH	No implementado; reservado para uso futuro.
ET2	IE.5	ADH	Habilita/deshabilita la interrupción de desbordamiento o de captura del temporizador 2 (sólo 80 × 2).
ES	IE.4	ACH	Habilita/deshabilita la interrupción del puerto serial.
ET1	IE.3	ABH	Habilita/deshabilita la interrupción de desbordamiento del temporizador 1.
EX1	IE.2	AAH	Habilita/deshabilita la interrupción externa 1.
ET0	IE.1	A9H	Habilita/deshabilita la interrupción de desbordamiento del temporizador 0.
EX0	IE.0	A8H	Habilita/deshabilita la interrupción externa 0.

IP (REGISTRO DE PRIORIDAD DE INTERRUPCIONES)

Símbolo: IP
 Función: Prioridad de interrupciones

Dirección de bit: B8H

Direccionable por bit: Sí

Resumen:

7	6	5	4	3	2	1	0
-	-	PT2	PS	PT1	PX1	PT0	PX0

Definiciones de bit:

Símbolo de bit	Posición	Dirección de bit	Descripción (1 = habilita, 0 = deshabilita)
—	IP.7	BFH	No implementado; reservado para uso futuro.
—	IP.6	BEH	No implementado; reservado para uso futuro.
PT2	IP.5	BDH	Nivel de prioridad de interrupción del temporizador 2 (sólo 80 × 2).
PS	IP.4	BCH	Nivel de prioridad de interrupción del puerto serial
PT1	IP.3	BBH	Nivel de prioridad de interrupción del temporizador 1.
PX1	IP.2	BAH	Nivel de prioridad de interrupción externa 1.
PT0	IP.1	B9H	Nivel de prioridad de interrupción del temporizador 0.
PX0	IP.0	B8H	Nivel de prioridad de interrupción externa 0.

T2CON (REGISTRO DE CONTROL DEL TEMPORIZADOR/CONTADOR 2)

Símbolo: T2CON

Function: Control del temporizador/contador 2

Dirección de bit: C8H

Direccionable por bit: Sí

Resumen:

7	6	5	4	3	2	1	0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

Definiciones de bit:

Símbolo de bit	Posición	Dirección de bit	Descripción
TF2	T2CON .7	CFH	Bandera de desbordamiento del temporizador 2. Se activa mediante hardware y se borra con software. TF2 no puede ser activada cuando RCLK = 1 o TCLK = 1.

EXF2	T2CON.6	CEH	Bandera externa del temporizador 2. Se activa cuando una captura o una recarga es causada por transición negativa en T2EX y cuando EXEN2 = 1. Cuando la interrupción del temporizador 2 está habilitada, EXF2 = 1 causará que la CPU se vectorice a la rutina de servicio de interrupción. EXF2 debe limpiarse mediante software.
RCLK	T2CON.5	CDH	Reloj de recepción. Cuando se activa causa que el puerto serial utilice los pulsos de desbordamiento del temporizador 2 para su reloj (sincronización) de recepción en los modos 1 y 3. RCLK = 0 causa que el desbordamiento del temporizador 1 se utilice como el reloj (sincronización) de recepción.
TCLK	T2CON.4	CCH	Reloj de transmisión. Cuando se activa causa que el puerto serial utilice los pulsos de desbordamiento del temporizador 2 para su reloj (sincronización) de transmisión en los modos 1 y 3. TCLK = 0 causa que el desbordamiento del temporizador 1 se utilice como el reloj (sincronización) de transmisión.
EXEN2	T2CON.3	CBH	Bandera de habilitación externa del temporizador 2. Cuando se activa permite la captura de una recarga como resultado de una transición negativa en T2EX si el temporizador 2 no se está utilizando para sincronizar al puerto serial. EXEN2 = 0 causa que el temporizador 2 ignore los eventos en T2EX.
TR2	T2CON.2	CAH	Bit de ejecución del temporizador 2. Control mediante software de INICIO/PARO para el temporizador 2; un 1 lógico inicia el temporizador.
C/T2	T2CON.1	C9H	Selección de contador/temporizador para el temporizador 2. 0 = temporizador interno, 1 = contador de eventos externos (disparado por una caída negativa).
CP/RL2	T2CON.0	C8H	Bandera de captura/recarga. Cuando se active, las capturas ocurrirán en transiciones negativas en T2EX si EXEN2 = 1. Cuando se borra, ocurren autorrecargas ya sea con los desbordamientos del temporizador 2 o en transiciones negativas en T2EX cuando EXEN2 = 1. Si RCLK = 1 o TCLK = 1, este bit se ignora y el temporizador se ve forzado a realizar una autorrecarga en los desbordamientos del temporizador 2.

PSW (PALABRA DE ESTADO DEL PROGRAMA)

Símbolo: PSW
 Función: Estado del programa
 Dirección de bit: D0H
 Direccionable por bit: Sí

TABLA D-2

RS1	RS0	Banco	Direcciones activas
0	0	0	00H–07H
0	1	1	08H–0FH
1	0	2	10H–17H
1	1	3	18H–1FH

Resumen:

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	—	P

Definiciones de bit:

Símbolo de bit	Posición	Dirección de bit	Descripción
CY	PSW.7	D7H	Bandera de acarreo. Se activa si ocurre un acarreo del bit 7 durante una suma, o se activa cuando ocurre un préstamo al bit 7 durante una resta.
AC	PSW.6	D6H	Bandera de acarreo auxiliar. Se activa durante las instrucciones de suma si ocurre un acarreo del bit 3 hacia el bit 4 o si el resultado en el nibble inferior está en el rango de 0AH a 0FH.
F0	PSW.5	D5H	Bandera 0. Disponible para el usuario para propósitos generales.
RS1	PSW.4	D4H	Bit 1 de selección de banco de registros. (Consulte la tabla D-2)
RS0	PSW.3	D3H	Bit 0 de selección de banco de registros. (Consulte la tabla D-2)
OV	PSW.2	D2H	Bandera de desbordamiento. Se activa después de una operación de suma o resta si ocurrió un desbordamiento aritmético (en otras palabras, si el resultado con signo es mayor a 127 o menor que –128).
—	PSW.1	D1H	—
P	PSW.0	D0H	Bandera de paridad. Se activa o borra mediante hardware en cada ciclo de instrucción para indicar un número impar/par de bits con el valor de “1” en el acumulador.



*Hoja de datos del 8051*¹

¹Reimpresas con el permiso de Intel Corporation del documento *Controladores incrustados de 8 bits* (270645), Santa Clara, CA: Intel Corporation, 1991..



MCS®-51

MICROCOMPUTADORAS DE 8 BITS ORIENTADAS AL CONTROL

8031/8051

8031AH/8051AH

8032AH/8052AH

8751H/8751H-8

- Proceso HMOS de alto rendimiento
- Temporizadores/Contadores de eventos internos
- Estructura de prioridad de interrupciones de 2 niveles
- 32 líneas de E/S (cuatro puertos de 8 bits)
- Espacio de memoria de programa de 64K
- Característica de seguridad protege a los componentes EPROM contra la piratería de software
- Procesador booleano
- Memoria RAM direccionable por bit
- Canal serial full dúplex programable
- 111 instrucciones (64 de un solo ciclo)
- Espacio de memoria para datos de 64K

Los productos MCS®-51 están optimizados para las aplicaciones de control. El procesamiento de bytes y las operaciones numéricas en estructuras de datos pequeñas se facilitan gracias a una variedad de modos de direccionamiento rápidos para acceder a la memoria RAM interna. El conjunto de instrucciones proporciona un menú conveniente de instrucciones aritméticas de 8 bits, el cual incluye multiplicación y división. Se proporciona un extenso soporte incorporado en el chip para variables de un bit como un tipo de datos separado, lo que permite la manipulación directa de bits y efectuar pruebas en sistemas lógicos y de control que requieren procesamiento booleano.

El 8051 es el miembro original de la familia MCS-51. El 8051AH es idéntico al 8051, pero está fabricado con tecnología HMOS II.

El 8751H es una versión con EPROM del 8051AH; esto es, la memoria de programa incorporada en el chip puede programarse eléctricamente y borrarse al exponerla a luz ultravioleta. Es completamente compatible con su predecesor, el 8751-8, pero incorpora dos nuevas características: un bit de seguridad de memoria de programa que puede utilizarse para proteger la EPROM contra lecturas no autorizadas, y un bit programable para modificar la velocidad en baudios (SMOD). El 8751H-8 es idéntico al 8751H, aunque opera sólo hasta 8 MHz.

El 8052AH es una versión mejorada del 8051AH. Es compatible con el 8051AH y está fabricado con tecnología HMOS II. La tabla siguiente muestra las mejoras del 8052AH. También puede referirse a esta tabla para las versiones con ROM, sin ROM, y con EPROM para cada producto.

Dispositivo	Memoria interna		Temporizadores/ contadores de eventos	Interrupciones
	De programa	Para datos		
8052AH	8K × 8 ROM	256 × 8 RAM	3 × 16 bits	6
8051AH	4K × 8 ROM	128 × 8 RAM	2 × 16 bits	5
8051	4K × 8 ROM	128 × 8 RAM	2 × 16 bits	5
8032AH	none	256 × 8 RAM	3 × 16 bits	6
8031AH	none	128 × 8 RAM	2 × 16 bits	5
8031	none	128 × 8 RAM	2 × 16 bits	5
8751H	4K × 8 EPROM	128 × 8 RAM	2 × 16 bits	5
8751-8	4K × 8 EPROM	128 × 8 RAM	2 × 16 bits	5



MCS®-51

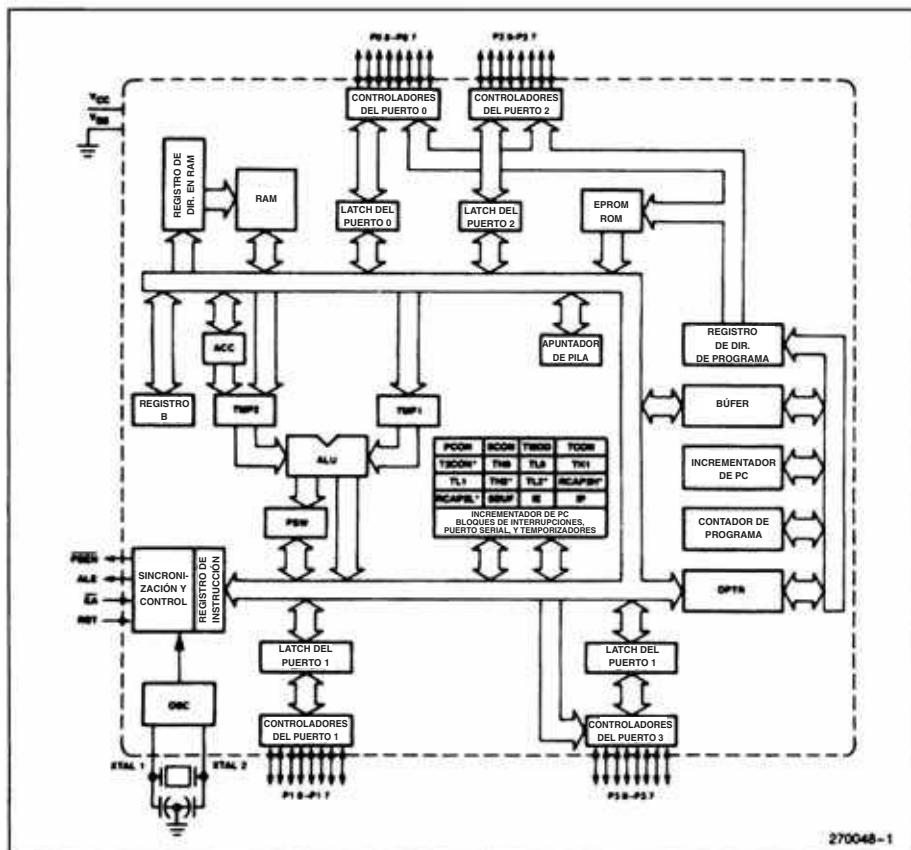


Figura 1. Diagrama de bloques del MCS®-51

CÁPSULAS

Pieza	Prefijo	Tipo de encapsulado
8051AH/	P	Encapsulado DIP de 40 terminales
8031AH	D	Encapsulado CERDIP de 40 terminales
	N	Encapsulado PLCC de 44 terminales
8052AH/	P	Encapsulado DIP de 40 terminales
8032AH	D	Encapsulado CERDIP de 40 terminales
	N	Encapsulado PLCC de 44 terminales
8751H/		
8751H-8	D	Encapsulado CERDIP de 40 terminales

DESCRIPCIÓN DE TERMINALES

V_{cc}: voltaje de alimentación.

V_{ss}: tierra del circuito.

Puerto 0: El puerto 0 es un puerto de E/S bidireccional de drenaje abierto de 8 bits. Como puerto de salida, cada terminal puede absorber 8 entradas LS TTL.

Las terminales del puerto 0 que contienen números 1 flotan y, en dicho estado, pueden utilizarse como entradas de alta impedancia.

El puerto 0 es también el bus de orden inferior de direcciones y de datos multiplexado durante el acceso a memoria externa para programa y para datos. En esta aplicación utiliza poderosas resistencias elevadoras internas cuando emite números 1, y puede actuar como fuente o absorber 8 entradas LS TTL.

El puerto 0 también recibe los bytes de código durante la programación de los componentes de la EPROM, y envía los bytes de código como salida durante la verificación de la programación de las partes de la ROM y de la EPROM. Se requieren resistencias elevadoras externas cuando se verifica la programación.

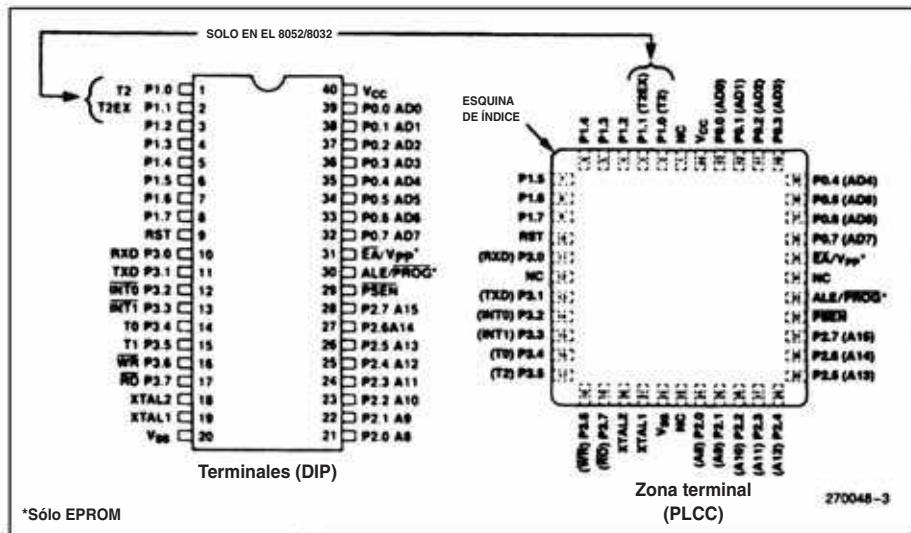


Figura 2. Conexiones del MCS®-51

Puerto 1: el puerto 1 es un puerto de E/S bidireccional de 8 bits con resistencias elevadoras internas. Los búferes de salida del puerto 1 pueden absorber/servir como fuente a 4 entradas LS TTL. Las terminales del puerto 1 que contienen números 1 se elevan a nivel alto mediante las resistencias elevadoras internas y en ese estado se pueden utilizar como entradas. Como entradas, las terminales del puerto 1 que están elevándose a nivel bajo externamente servirán como fuente de corriente (I_{IL} en la hoja de datos) debido a las resistencias elevadoras internas.

El puerto 1 también recibe los bytes de direcciones de orden inferior durante la programación de las partes de la EPROM, y durante la verificación de la programación de las partes de la ROM y la EPROM.

En el 8032AH y el 8052AH, las terminales P1.0 y P.1 del puerto 1 también sirven a las funciones T2 y T2EX, respectivamente.

Puerto 2: el puerto 2 es un puerto de E/S bidireccional de 8 bits con resistencias elevadoras internas. Los búferes de salida del puerto 2 pueden absorber/servir como fuente a 4 entradas LS TTL. Las terminales del puerto 2 que contienen números 1 se elevan a nivel alto mediante las resistencias elevadoras internas, y en ese estado se pueden utilizar como entradas. Como entradas, las terminales del puerto 2 que se están elevando a nivel bajo externamente servirán como fuentes de corriente (I_{IL} en la hoja de datos) debido a las resistencias elevadoras internas.

El puerto 2 emite el byte superior de direcciones durante las búsquedas de memoria externa para programa y durante los accesos a la memoria externa para datos que utilizan direcciones de 16 bits (MOVX @DPTR). En esta aplicación se utilizan poderosas resistencias elevadoras internas cuando se emiten números 1. Durante los accesos a la memoria externa para datos que

utilizan direcciones de 8 bits (MOVX @Ri), el puerto 2 emite el contenido del registro con funciones especiales P2.

El puerto 2 también recibe los bits superiores de direcciones durante la programación de las partes de la EPROM, y durante la verificación de la programación de las partes de la ROM y la EPROM.

Puerto 3: el puerto 3 es un puerto de E/S bidireccional de 8 bits con resistencias elevadoras internas. Los búferes de salida del puerto 3 pueden absorber/servir como fuente a 4 entradas LS TTL. Las terminales del puerto 3 que contienen números 1 se elevan a nivel alto mediante las resistencias elevadoras internas, y en ese estado se pueden utilizar como entradas. Como entradas, las terminales del puerto 3 que están elevándose en forma externa a nivel bajo servirán como fuentes de corriente (I_{IL} en la hoja de datos) debido a las resistencias elevadoras.

El puerto 3 también sirve a las funciones de diversas características especiales de la familia MCS-51, como se muestra a continuación:

Terminal de puerto	Función alternativa
P3.0	RXD (puerto de entrada serial)
P3.1	TXD (puerto de salida serial)
P3.2	INT01 (interrupción externa 0)
P3.3	INT1 (interrupción externa 1)
P3.4	T0 (entrada externa del temporizador 0)
P3.5	T1 (entrada externa del temporizador 1)
P3.6	WR (pulso de escritura a memoria externa para datos)
P3.7	RD (pulso de lectura de memoria externa para datos)



MCS®-51

RST: entrada de reinicialización. Un nivel alto en esta terminal para dos ciclos de máquina reinicializa el dispositivo positivo mientras el oscilador está funcionando.

ALE/PROG: pulso de salida de habilitador de latch de direcciones para hacer un latch del byte inferior de la dirección durante los accesos a la memoria externa. Esta terminal también es el pulso de entrada de programa (PROG) durante la programación de las partes de la EPROM.

En su operación normal, el ALE se emite a una velocidad constante de 1/6 de la frecuencia del oscilador y puede ser utilizado para propósitos de sincronización externa. Sin embargo, hay que considerar que se ignora un pulso del ALE durante cada acceso a la memoria externa para datos.

PSEN: habilitador de almacenamiento de programa, es el pulso de lectura a la memoria externa para programa.

Cuando el dispositivo está ejecutando código de una memoria externa para programa, la señal **PSEN**: se activa dos veces en cada ciclo de máquina, con la excepción de que se ignoran dos activaciones de la **PSEN**: durante cada acceso a la memoria externa para datos.

EA/V_{PP}: este habilitador de acceso externo **EA** debe estar conectado a **V_{ss}** para poder habilitar cualquier dispositivo MCS-51 y realizar una búsqueda de código desde las ubicaciones en memoria externa para programa empezando desde 0000H hasta FFFFH. **EA** debe conectarse a **V_{cc}** para efectuar la ejecución interna de programas.

Sin embargo, observe que si se programa el bit de seguridad en los dispositivos EPROM, el dispositivo no realizará una búsqueda de código de ninguna ubicación en memoria externa para programa.

Esta terminal también recibe los 21V del voltaje de alimentación para programación (VPP) durante la programación de las partes de la EPROM.

XTAL1: entrada al amplificador de oscilador inversor.

XTAL2: salida del amplificador de oscilador inversor.

CARACTERÍSTICAS DEL OSCILADOR

XTAL1 y XTAL2 son la entrada y la salida, respectivamente, de un amplificador inversor que se puede configurar para utilizarlo como un oscilador incorporado en el chip, según ilustra la figura 3. Se puede utilizar ya sea un cristal de cuarzo o un resonador de cerámica. Información más detallada acerca del uso del oscilador incorporado en el chip está disponible en la Nota de Aplicación AP-155, "Osciladores para microcontroladores".

Para controlar el dispositivo desde una fuente de sincronización externa, XTAL1 debe estar conectada a tierra mientras que el XTAL2 debe estar controlado, como se muestra en la figura 4. No existen requerimientos sobre el ciclo de trabajo de la señal externa de sincronización, ya que la entrada al circuito interno de sincronización es a través de un flip-flop de división entre dos, pero debemos respetar los tiempos mínimos y máximos a nivel alto y a nivel bajo especificados en la hoja de datos.

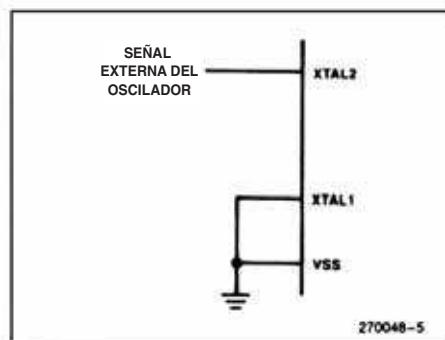


Figura 4. Configuración para control externo

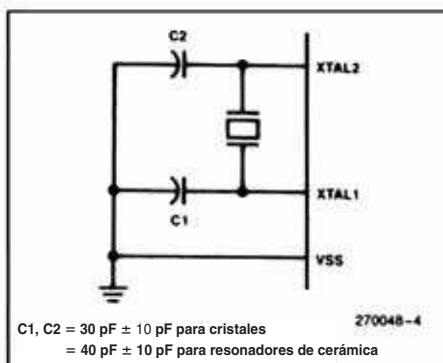


Figura 3. Conexiones del oscilador

CONSIDERACIONES DE DISEÑO

Si un 8751BH o un 8752BH pueden reemplazar un 8751H en un futuro diseño, el usuario debe comparar cuidadosamente ambas hojas de datos, por si existen diferencias en las características de corriente directa (CD) o corriente alterna (CA). Observe que las especificaciones de V_{IH} e I_{IH} para la terminal **EA** difieren significativamente entre ambos dispositivos.

La exposición a la luz mientras un dispositivo EPROM está en operación puede causar errores lógicos. Por esta razón, sugerimos se coloque una etiqueta opaca sobre la ventana cuando la pastilla esté expuesta a la luz ambiental.



ESPECIFICACIONES MÁXIMAS ABSOLUTAS*

Temperatura ambiental bajo polarización	0°C a 70°C
Temperatura de almacenamiento	-65°C a 150°C
Voltaje en la terminal $\bar{E}A/V_{pp}$ a V_{SS}	-0.5V a + 21.5V
Voltaje en cualquier otra terminal a V_{SS}	-0.5V a + 7V
Disipación de energía	1.5W

AVISO: Esta es una hoja de datos de producción. Las especificaciones están sujetas a cambios sin previo aviso.

*PRECAUCIÓN: Si las condiciones de carga del dispositivo van más allá de las "especificaciones máximas absolutas", pueden causar daños permanentes. Estas son especificaciones de condiciones de carga solamente. La operación más allá de las "condiciones de operación" no es recomendable, y una exposición prolongada más allá de las "condiciones de operación" puede afectar la confiabilidad del dispositivo.

Condiciones de operación: T_A (bajo polarización) = 0°C a +70°C; $V_{CC} = 5V \pm 10\%$; $V_{SS} = 0V$.

CARACTERÍSTICAS DE CORRIENTE DIRECTA (CD) (bajo condiciones de operación)

Símbolo	Parámetro	Mín	Máx	Unids.	Condiciones de prueba
V_{IL}	Voltaje bajo de entrada (excepto la terminal EA de 8751H & 8751H-8)	-0.5	0.8	V	
V_{IL1}	Voltaje bajo de entrada a la terminal EA de 8751H & 8751H-8	0	0.7	V	
V_{IH}	Voltaje alto de entrada (excepto XTAL2, RST)	2.0	$V_{CC} + 0.5$	V	
V_{IH1}	Voltaje alto de entrada a XTAL2, RST	2.5	$V_{CC} + 0.5$	V	$XTAL1 = V_{SS}$
V_{OL}	Voltaje bajo de salida (puertos 1, 2 y 3)*		0.45	V	$I_{OL} = 1.6\text{ mA}$
V_{OL1}	Voltaje bajo de salida (puerto 0, ALE, PSEN)*				
	8751H, 8751H-8		0.60 0.45	V V	$I_{OL} = 3.2\text{ mA}$ $I_{OL} = 2.4\text{ mA}$
	Todos los demás		0.45	V	$I_{OL} = 3.2\text{ mA}$
V_{OH}	Voltaje alto de salida (puertos 1, 2, 3, ALE, PSEN)	2.4		V	$I_{OH} = -80\text{ }\mu\text{A}$
V_{OH1}	Voltaje alto de salida (puerto 0 en el modo de bus externo)	2.4		V	$I_{OH} = -400\text{ }\mu\text{A}$
I_{IL}	Corriente de entrada lógica 0 (puertos 1, 2, 3, RST) 8032AH, 8052AH Todos los demás		-800 -500	μA μA	$V_{IN} = 0.45V$ $V_{IN} = 0.45V$
I_{IL1}	Corriente de entrada lógica 0 a terminal EA de 80751H & 8751H-8 solamente		-15	mA	$V_{IN} = 0.45V$
I_{IL2}	Corriente de entrada lógica 0 (XTAL2)		-3.2	mA	$V_{IN} = 0.45V$
I_{I1}	Corriente de fugas de entrada (puerto 0) 8751H & 8751H-8 Todos los demás		± 100 ± 10	μA μA	$0.45 \leq V_{IN} \leq V_{CC}$ $0.45 \leq V_{IN} \leq V_{CC}$
I_{IH}	Corriente de entrada lógica 1 a terminal EA de 80751H & 8751H-8		500	μA	$V_{IN} = 2.4V$
I_{IH1}	Corriente de entrada a RST para activar reinic.		500	μA	$V_{IN} < (V_{CC} - 1.5V)$
I_{CC}	Corriente de alimentación de energía: 8031/8051 8031AH/8051AH 8032AH/8052AH 8751H/8751H-8		160 125 175 250	mA mA mA mA	Todas las salidas desconectadas; EA = VCC
C_{IO}	Capacitancia de terminal		10	pF	Frec. de prueba = 1 MHz

*NOTA:

La carga capacitiva en los puertos 0 y 2 puede causar que pulsos de ruido falso se traslapan con los VOLs del ALE y de los puertos 1 y 3. El ruido es causado por la descarga de la capacitancia en el bus externo a las terminales de puerto 0 y el puerto 2 cuando estas terminales hacen una transición de 1 a 0 durante las operaciones de bus. En el peor de los casos (carga capacitiva > 100 pF), el pulso de ruido en la línea ALE puede exceder de 0.8V. En esas circunstancias, es deseable cualificar el ALE con un disparador Schmitt o utilizar un latch de dirección con un Schmitt.



MCS®-51

CARACTERÍSTICAS DE CORRIENTE ALTERNA (CA) (bajo condiciones de operación):

Capacitancia de carga para el puerto 0, ALE y PSEN = 100 pF;

Capacitancia de carga para todas las demás salidas = 80 pF.

Símbolo	Parámetro	Oscilador de 12 MHz		Oscilador variable		Unida-des
		Mín	Máx	Mín	Máx	
1/TCLCL	Frecuencia del oscilador			3.5	12.0	MHz
TLHLL	Ancho de pulso del ALE	127		2TCLCL – 40		ns
TAVLL	Dir. válida al ALE de nív. bajo	43		TCLCL – 40		ns
TLLAX	Almac. de dirección después de un ALE de nivel bajo	48		TCLCL – 35		ns
TLLIV	ALE de nivel bajo a una instrucción válida en 8751H Todos los demás		183 233		4TCLCL – 150 4TCLCL – 100	ns ns
TLLPL	ALE de nivel bajo a PSEN de nivel bajo	58		TCLCL – 25		ns
TPLPH	Ancho de pulso de PSEN 8751H Todos los demás	190 215		3TCLCL – 60 3TCLCL – 35		ns ns
TPLIV	PSEN de nivel bajo a instrucción válida en 8751H Todos los demás		100 125		3TCLCL – 150 3TCLCL – 125	ns ns
TPXIX	Almacenamiento de instrucción de entrada después de PSEN	0		0		ns
TPXIZ	Flotación de instrucción de entrada después de PSEN		63		TCLCL – 20	ns
TPXAV	PSEN a dirección válida	75		TCLCL – 8		ns
TAVIV	Dirección a instrucción válida en 8751H Todos los demás		267 302		5TCLCL – 150 5TCLCL – 115	ns ns
TPLAZ	PSEN de nivel bajo a flotación de dirección		20		20	ns
TRLRH	Ancho de pulso de RD	400		6TCLCL – 100		ns
TWLWH	Ancho de pulso de WR	400		6TCLCL – 100		ns
TRLDV	RD de nivel bajo a entrada de datos válidos		252		5TCLCL – 165	ns
TRHDX	Almacenamiento de datos después de una RD	0		0		ns
TRHDZ	Flotación de datos después de una RD		97		2TCLCL – 70	ns
TLLDV	ALE de nivel bajo a entrada de datos válidos		517		8TCLCL – 150	ns
TAVDV	Dirección a entrada de datos válidos		585		9TCLCL – 165	ns
TLLWL	ALE de nivel bajo a RD o WR de nivel bajo	200	300	3TCLCL – 50 3TCLCL + 50		ns
TAVWL	Dirección a RD o WR de nivel bajo	203		4TCLCL – 130		ns
TQVWX	Datos válidos a transición de WR 8751H Todos los demás	13 23		TCLCL – 70 TCLCL – 60		ns ns
TQVWH	Datos válidos a WR de nivel alto	433		7TCLCL – 150		ns
TWHQX	Almacenamiento de datos después de una WR	33		TCLCL – 50		ns
TRLAZ	RD de nivel bajo a flotación de dirección		20		20	ns
TWHLH	RD o WR de nivel alto a ALE de nivel alto 8751H Todos los demás	13 43	133 123	TCLCL – 50 TCLCL – 40	TCLCL + 50 TCLCL + 40	ns ns

NOTA:

*Esta tabla no incluye las características de CA del 8751-8 (consulte la siguiente página).



Esta tabla es sólo para el 8751H-8

CARACTERÍSTICAS DE CORRIENTE ALTERNA (CA) (bajo condiciones de operación):

Capacitancia de carga para el puerto 0, ALE y PSEN = 100 pF;

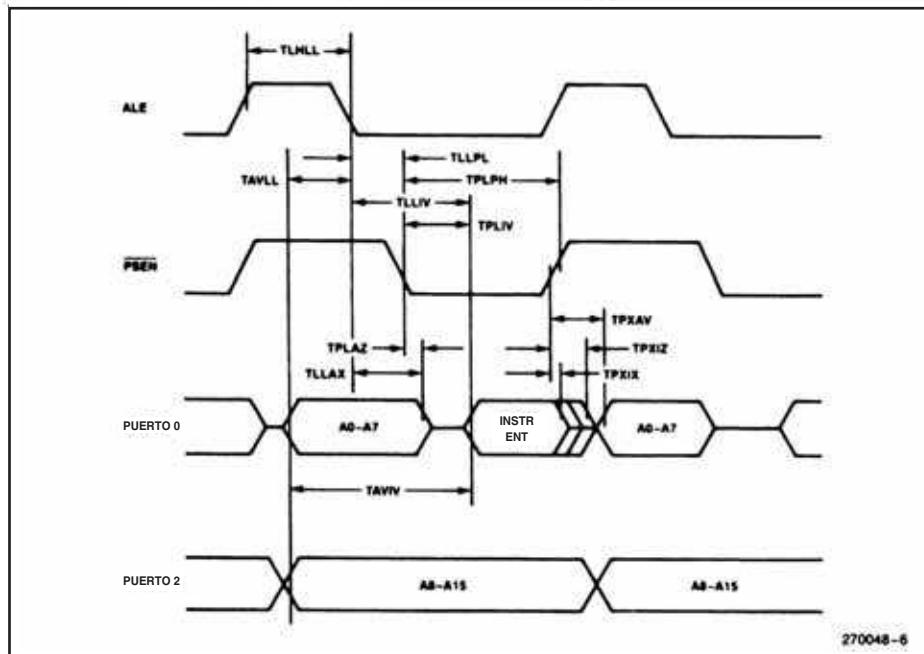
Capacitancia de carga para todas las demás salidas = 80 pF.

Símbolo	Parámetro	Oscilador de 8 MHz		Oscilador variable		Unida-des
		Mín	Máx	Mín	Máx	
1/TCLCL	Frecuencia del oscilador			3.5	8.0	MHz
TLHLL	Ancho de pulso del ALE	210		2TCLCL – 40		ns
TAVLL	Dir. válida al ALE de niv. bajo	85		TCLCL – 40		ns
TLLAX	Almac. de dirección después de un ALE de nivel bajo	90		TCLCL – 35		ns
TLLIV	ALE de nivel bajo a una instrucción válida en		350		4TCLCL – 150	ns
TLLPL	ALE de nivel bajo a PSEN de nivel bajo	100		TCLCL – 25		ns
TPLPH	Ancho de pulso de PSEN	315		3TCLCL – 60		ns
TPLIV	PSEN de nivel bajo a instrucción válida en		225		3TCLCL – 150	ns
TPXIX	Almacenamiento de instrucción de entrada después de PSEN	0		0		ns
TPXIZ	Flotación de instrucción de entrada después de PSEN		105		TCLCL – 20	ns
TPXAV	PSEN a dirección válida	117		TCLCL – 8		ns
TAVIV	Dirección a instrucción válida en		475		5TCLCL – 150	ns
TPLAZ	PSEN de nivel bajo a flotación de dirección		20		20	ns
TRLRH	Ancho de pulso de RD	650		6TCLCL – 100		ns
TWLWH	Ancho de pulso de WR	650		6TCLCL – 100		ns
TRLDV	RD de nivel bajo a entrada de datos válidos		460		5TCLCL – 165	ns
TRHDX	Almacenamiento de datos después de una RD	0		0		ns
TRHDZ	Flotación de datos después de una RD		180		2TCLCL – 70	ns
TLLDV	ALE de nivel bajo a entrada de datos válidos		850		8TCLCL – 150	ns
TAVDV	Dirección a entrada de datos válidos		960		9TCLCL – 165	ns
TLLWL	ALE de nivel bajo a RD o WR de nivel bajo	325	425	3TCLCL – 50	3TCLCL + 50	ns
TAVWL	Dirección a RD o WR de nivel bajo	370		4TCLCL – 130		ns
TQVWX	Datos válidos a transición de WR	55		TCLCL – 70		ns
TQVWH	Datos válidos a WR de nivel alto	725		7TCLCL – 150		ns
TWHQX	Almacenamiento de datos después de una WR	75		TCLCL – 50		ns
TRLAZ	RD de nivel bajo a flotación de dirección		20		20	ns
TWHLH	RD o WR de nivel alto a ALE de nivel alto	75	175	TCLCL – 50	TCLCL + 50	ns

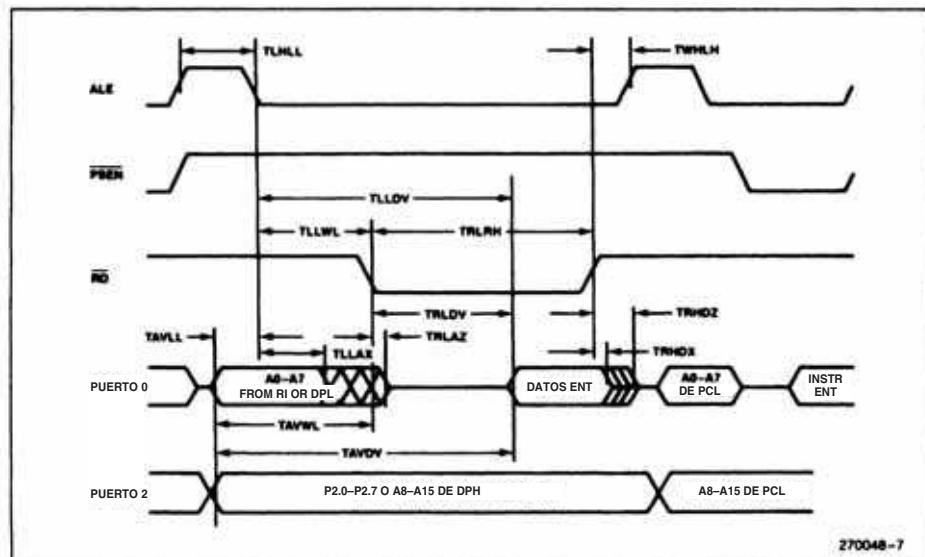


MCS®-51

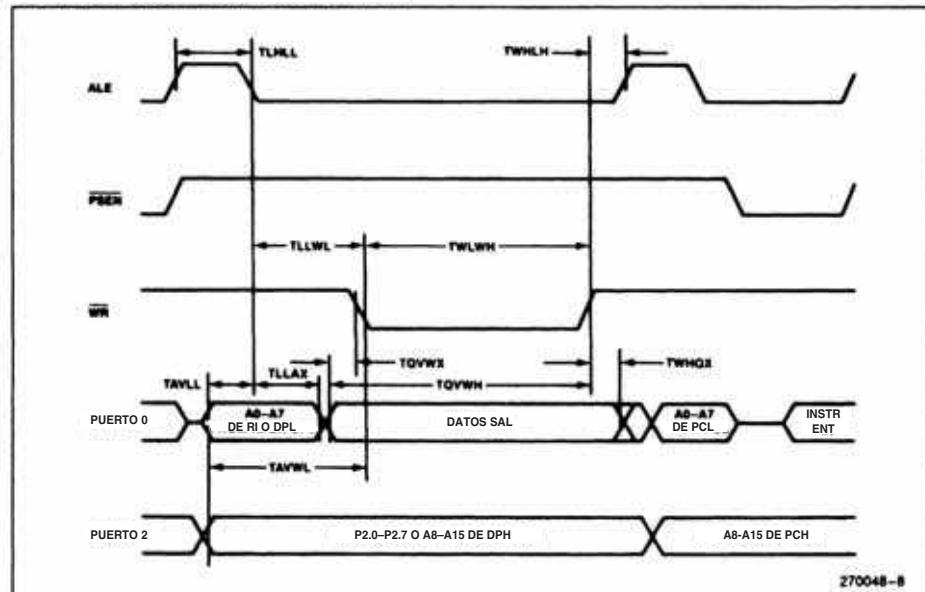
CICLO DE LECTURA DE MEMORIA EXTERNA PARA PROGRAMA



CICLO DE LECTURA DE MEMORIA EXTERNA PARA DATOS



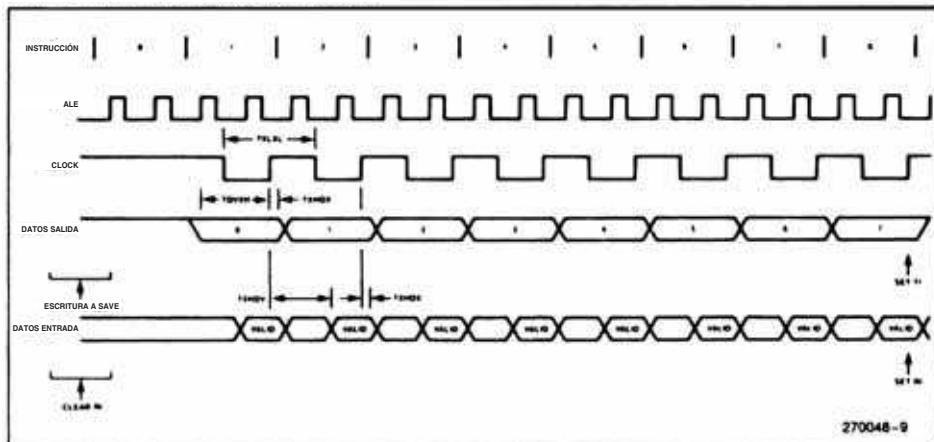
CICLO DE ESCRITURA DE MEMORIA EXTERNA PARA DATOS



**SINCRONIZACIÓN DEL PUERTO SERIAL – MODO DE REGISTRO DE DESPLAZAMIENTO**

Condiciones de prueba: TA = 0°C a 70°C; VCC = 5V ± 10%; VSS = 0V; Capacitancia de carga = 80 pF.

Símbolo	Parámetro	Oscilador de 12 MHz		Oscilador variable		Unidades
		Mín	Máx	Mín	Máx	
TXLXL	Tiem. del cic. de rel. del pto. serial	1.0		12TCLCL		μs
TQVXH	Configuración de datos de salida al borde de subida del reloj	700		10TCLCL – 133		ns
TXHQX	Almacenamiento de datos de salida después del borde de subida del reloj	50		2TCLCL – 117		ns
TXHDX	Almacenamiento de datos de entrada después del borde de subida del reloj	0		0		ns
TXHDV	Borde de subida del reloj a datos de entrada válidos		700		10TCLCL – 133	ns

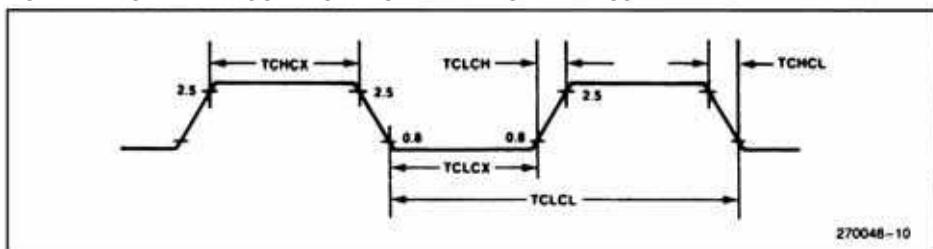
FORMAS DE ONDA DE LA SINCRONIZACIÓN DEL REGISTRO DE DESPLAZAMIENTO



CONTROLADOR EXTERNO DE RELOJ

Símbolo	Parámetro	Mín	Máx	Unidades
1/TCLCL	Frecuencia del oscilador (excepto el 8751H-8) 8751H-8	3.5 3.5	12 8	MHz MHz
TCHCX	Tiempo a nivel alto	20		ns
TCLCX	Tiempo a nivel bajo	20		ns
TCLCH	Tiempo de subida		20	ns
TCHCL	Tiempo de bajada		20	ns

FORMA DE ONDA DEL CONTROLADOR EXTERNO DE RELOJ

FORMA DE ONDA DE PRUEBAS DE CA
DE ENTRADAS Y SALIDAS



CARACTERÍSTICAS DE LA EPROM

Tabla 3. Modos de programación de la EPROM

Mode	PSEN	ALE	EA	P2.7	P2.6	P2.5	P2.4
Programa	1	0	0*	VPP	1	0	X
Prohibe	1	0	1	X	1	0	X
Verifica	1	0	1	1	0	0	X
Seguridad act.	1	0	0*	VPP	1	1	X

NOTA:

"1" = nivel lógico alto para esa terminal

"0" = nivel lógico bajo para esa terminal

"X" = "no importa"

"VPP" = $+21V \pm 0.5V$

*ALE se pulsa a nivel bajo por 50 ms.

Programación de la EPROM

El componente debe estar ejecutándose con un oscilador de 4 a 6 MHz para ser programado. (La razón por la cual el oscilador debe estar ejecutándose es que el bus interno se está utilizando para transferir los datos de direcciones y de programa a los registros internos adecuados). La dirección de una ubicación en EPROM que será programada se aplica al puerto 1 y a las terminales P2.0 a P2.3 del puerto 2, mientras que el byte de código que será programado a esa ubicación se aplica al puerto 0. Las otras terminales del puerto 2 y RST, PSEN y EA deben mantenerse a los niveles de "Programa" indicados en la tabla 3. La señal ALE se pulsa a nivel bajo durante 50 ms para programar el byte de código hacia la ubicación en EPROM direccionada. La figura 5 muestra la configuración.

EA se mantiene generalmente a un nivel lógico alto hasta justo antes de que el pulso sea enviado a la ALE. EA se eleva después a $+21V$, se envía un pulso a la ALE, y entonces EA regresa a su nivel lógico alto. Las formas de onda y las especificaciones de sincronización detalladas se muestran en secciones posteriores de esta hoja de datos.

Observe que la terminal de EA/VPP no puede llegar más arriba del máximo nivel especificado para VPP de 21.5V durante cualquier cantidad de tiempo. Tan sólo un pequeño error arriba del nivel de ese voltaje puede dañar permanentemente al dispositivo. La fuente de VPP debe estar muy bien regulada y libre de posibilidades de error o de ruido.

Verificación de programación

Si no se ha programado el bit de seguridad, podemos leer la memoria para programa incorporada en el chip para propósitos de verificación. La dirección de la ubicación en la memoria para programa a leer se aplica al puerto 1 y a las terminales P2.0 a P2.3. Las otras terminales se deben mantener en sus niveles de "Verificación" indicados en la tabla 3. El contenido de la ubicación direccionada saldrá en el puerto 0. Se requieren resistencias elevadoras externas en el puerto 0 para ejecutar esta operación.

La configuración, se muestra en la figura 6, es la misma que para la programación de la EPROM, excepto que el pin P2.7 se mantiene a nivel lógico bajo o puede utilizarse como un pulso de lectura activo a nivel bajo.

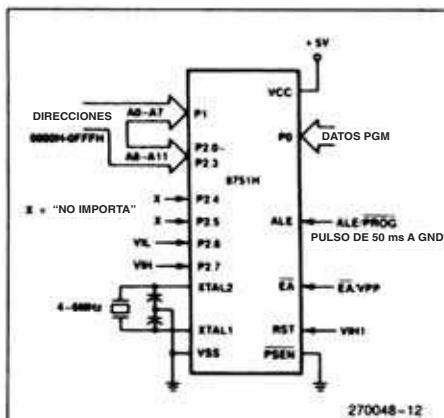


Figura 5. Configuración para la programación

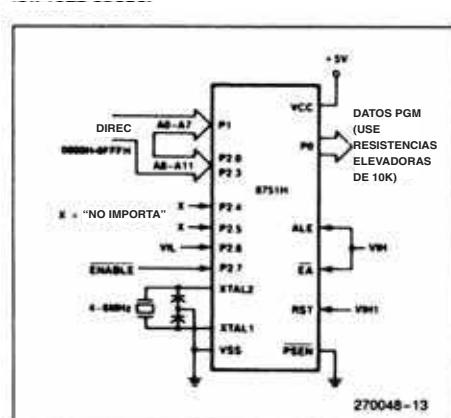


Figura 6. Verificación de programación



Seguridad de la EPROM

La característica de seguridad consta de un bit de "observación" el cual, una vez programado, niega el acceso eléctrico por cualquier medio externo a la memoria para programa incorporada en el chip. El bit se programa como se muestra en la figura 7. La configuración y el procedimiento son iguales que los empleados para la programación normal de la EPROM, excepto que P2.6 se mantiene a un nivel lógico alto. Los puertos 0 y 1, y las terminales P2.0 a P2.3 pueden estar en cualquier estado. Las otras terminales se deben mantener en los niveles de "Seguridad" indicados en la tabla 3.

Una vez que el bit de seguridad ha sido programado, sólo se puede poner en cero mediante un borrado completo de la memoria para programa. Mientras que el bit está programado, no se puede leer la memoria interna para programa, el dispositivo no se puede programar más, y **no puede ejecutarse desde la memoria externa para programa**. Si se borra la EPROM, y por lo tanto se limpia el bit de seguridad, se recupera toda la funcionalidad del dispositivo. Éste se puede reprogramar luego de hacer esto.

Características de borrado

El borrado de la EPROM empieza a ocurrir cuando el chip es expuesto a luz con longitudes de onda más cortas que aproximadamente 4,000 Angstroms. Debido a que la luz solar y la luz fluorescente tienen longitudes de onda en este rango, la exposición a estas fuentes de luz durante un período amplio (aproximadamente una semana en luz solar o tres años bajo luz fluorescente de nivel ambiental) puede causar un borrado inadvertido. Si una aplicación somete al dispositivo a este tipo de exposición, sugerimos colocar una etiqueta opaca sobre la ventana.

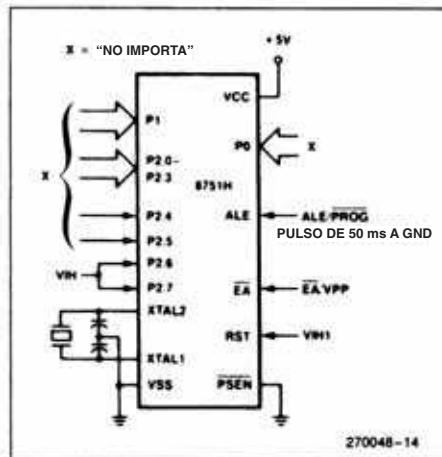


Figura 7. Programación del bit de seguridad

El procedimiento de borrado que se recomienda es la exposición a luz ultravioleta (a 2537 Angstroms) a una dosis integrada de, por lo menos, 15 W·seg/cm². La exposición de la EPROM a una lámpara ultravioleta de una medida de 12,000 $\mu\text{W}/\text{cm}^2$ durante 20 a 30 minutos a una distancia de aproximadamente 1 pulgada será suficiente.

El procedimiento de borrado deja el arreglo en un estado en donde todos los bits son números 1.

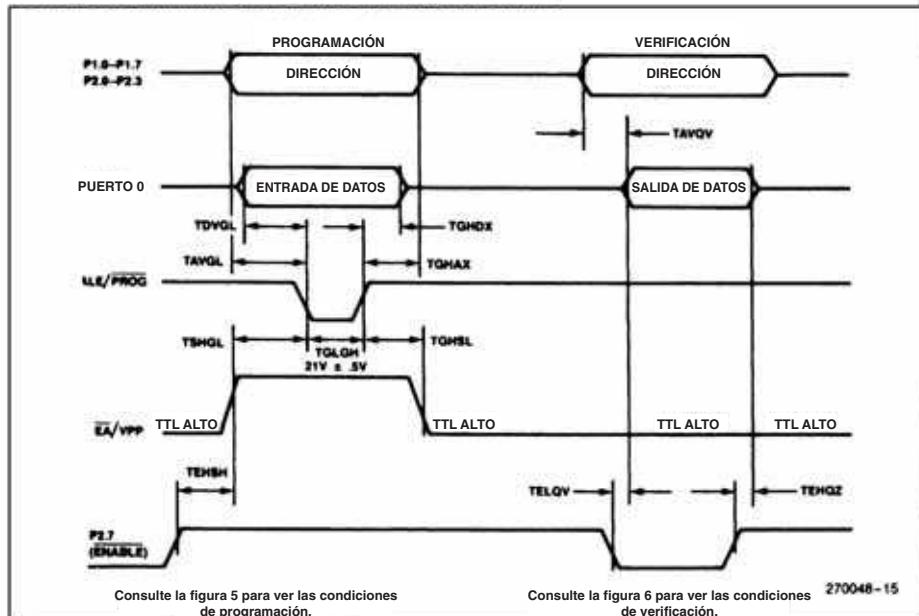
CARACTERÍSTICAS DE PROGRAMACIÓN Y VERIFICACIÓN DE LA EPROM

$T_A = 21^\circ\text{C}$ a 27°C ; $V_{CC} = 5\text{V} \pm 10\%$; $V_{SS} = 0\text{V}$

Símbolo	Parámetro	Mín	Máx	Unidades
V _{PP}	Volt. de alimentación para la prog.	20.5	21.5	V
I _{PP}	Corr. de alimentación para la prog.		30	mA
1/TCLCL	Frecuencia del oscilador	4	6	MHz
TAVGL	Config. de dir. a PROG a nivel bajo	48TCLCL		
TGHAX	Almac. de dir. después de PROG	48TCLCL		
TDVGL	Confi. de dats. a PROG a nivel bajo	48TCLCL		
TGHDX	Almac. de dats. después de PROG	48TCLCL		
TEHSH	P2.7 (ENABLE) a nivel alto a V _{PP}	48TCLCL		
TSHGL	Config. de V _{PP} a PROG a niv. bajo	10		μs
TGHSL	Almac. de V _{PP} después de PROG	10		μs
TGLGH	Anchura de PROG	45	55	ms
TAVOV	Dirección a datos válidos		48TCLCL	
TELOV	ENABLE a nivel bajo a dats. válidos		48TCLCL	
TEHOZ	Flot. de dats. después de ENABLE	0	48TCLCL	



FORMAS DE ONDA DE LA PROGRAMACIÓN Y VERIFICACIÓN DE LA EPROM



HISTORIA DE REVISIÓN DE LA HOJA DE DATOS

Las siguientes son las diferencias clave que hay entre la versión -004 y esta versión de la hoja de datos.

1. El estado de la hoja de datos se cambió de "Preliminar" a "Producción".
2. Se eliminó la oferta del paquete LCC.
3. Se revisó tanto la advertencia acerca de las especificaciones máximas como la historia de revisión de la hoja de datos.

Las siguientes son las diferencias clave que hay entre la versión -003 y esta versión de la hoja de datos.

1. Se amplió la introducción para incluir descripciones de los productos.
2. Se añadió la tabla de paquetes.
3. Se añadieron las consideraciones de diseño.
4. Se añadieron las condiciones de prueba para las especificaciones de I_{IL1} e I_{IH} a las características de CD.
5. Se añadió la historia de revisión de la hoja de datos.

F

Diagrama de códigos en ASCII

Bits	7	0	0	0	0	1	1	1	1
	6	0	0	1	1	0	0	1	1
4 3 2 1 5 0	1	0	1	0	1	0	1	0	1
0 0 0 0 NUL	DLE	SP	0	@	P	\	p		
0 0 0 1 SOH	DC1	!	1	A	Q	a	q		
0 0 1 0 STX	DC2	"	2	B	R	b	r		
0 0 1 1 ETX	DC3	#	3	C	S	c	s		
0 1 0 0 EOT	DC4	\$	4	D	T	d	t		
0 1 0 1 ENQ	NAK	%	5	E	U	e	u		
0 1 1 0 ACK	SYN	&	6	F	V	f	v		
0 1 1 1 BEL	ETB	'	7	G	W	g	w		
1 0 0 0 BS	CAN	(8	H	X	h	x		
1 0 0 1 HT	EM)	9	I	Y	i	y		
1 0 1 0 LF	SUB	*	:	J	Z	j	z		
1 0 1 1 VT	ESC	+	;	K	[k	l		
1 1 0 0 FF	FS	'	<	L	\	l	:		
1 1 0 1 CR	GS	-	=	M]	m)		
1 1 1 0 SO	RS	,	>	N	^	n	~		
1 1 1 1 SI	US	/	?	O	-	o	DEL		

FIGURA F-1

Diagrama de códigos en ASCII



MON51—un programa de monitoreo para el 8051

Este apéndice contiene el listado para un programa de monitoreo del 8051 (MON51), junto con una descripción general de su diseño y su operación. Muchos de los conceptos utilizados en la programación escrita en lenguaje ensamblador desarrollados previamente en ejemplos cortos (consulte el capítulo 7) pueden reforzarse repasando este apéndice.

El MON51 es un programa de monitoreo escrito para el microcontrolador 8051 y, más específicamente, para la computadora de una sola tarjeta SBC-51 (consulte el capítulo 11). Empezaremos con una descripción del propósito de un programa de monitoreo y después daremos un resumen de los comandos del MON51. También describimos la operación general del MON51 y algunos detalles del diseño. Las últimas páginas de este apéndice contienen los listados para cada uno de los archivos fuente ensamblados, el listado creado por el RL51, y un vaciado de memoria del MON51 en el formato hexadecimal de Intel.

Un programa de monitoreo no es un sistema operativo. Es un pequeño programa con comandos que proporciona un nivel primitivo de operación del sistema y la interacción con el usuario. La mayor diferencia es que los sistemas operativos se encuentran en computadoras grandes con unidades de disco, mientras que los programas de monitoreo se encuentran en sistemas pequeños, tales como la SCB-51, con un teclado (o teclado numérico) para captura de datos y un CRT (o LEDs) para salida de datos. Algunas computadoras de una sola tarjeta también proporcionan almacenamiento masivo utilizando la interfaz a una cinta de audio.

El MON51 es un programa escrito en lenguaje ensamblador del 8051 y tiene un tamaño aproximado de dos kilobytes. Fue codificado en el sistema de desarrollo iPDS100 de Intel utilizando un editor (CREDIT), un ensamblador cruzado (ASM51), y un enlazador/localizador (RL51). Las pruebas y la depuración se realizaron mediante un emulador de hardware (EMV51) conectado a una computadora de una sola tarjeta SDK-51™ de Intel. Es importante mencionar que los emuladores de hardware son herramientas tan poderosas, que la primera versión del MON51 almacenada en la EPROM estaba esencialmente libre de errores.

El MON51 se desarrolló aplicando técnicas de programación modular. Se utilizaron diez archivos fuente, incluyendo nueve archivos de programa y un archivo de definición de macros. Con el propósito de mantener los archivos de listado relativamente cortos para este apéndice, la opción de “crear tabla de símbolos” fue deshabilitada en cada archivo fuente utilizando el

control del ensamblador \$NOSYMBOLS. (Consulte MAIN.LST, línea 4, pág. 466). Sin embargo, debido a que también se utilizó el control del ensamblador \$DEBUG, el archivo listado creado por el enlazador/localizador, RL51, contiene una tabla de símbolos absoluta completa para todo el programa. La salida absoluta del enlazador/localizador se colocó en V12 (de “versión 12”), y el archivo de listado se colocó en V12.M51. La herramienta OH (objeto a hexadecimal) fue utilizada para convertir la salida absoluta a un archivo hexadecimal de Intel, V12.HEX, adecuada para su impresión, cargarlo a un sistema destino, o quemarlo a la EPROM. (Se han insertado espacios al archivo en hexadecimal para mejorar su legibilidad).

Los archivos que aparecen en este apéndice se identifican a continuación.

Archivo	Página
Listados del ASM51	
PRINCIPAL.LST	466
OBTENPAR.LST	470
ES.LST	473
CONVRT.LST	478
CARGA.LST	479
VACIA.LST	481
SFR.LST	484
ESR.LST	488
MODIFICA.LST	489
Definiciones de macros	
MACROS.SRC	493
Listado del RL51	
V12.M51	494
Archivo hexadecimal	
V12.HEX	496

COMANDOS Y ESPECIFICACIONES

El MON51 utiliza el puerto serial incorporado al chip del 8051 para su entrada/salida. Requiere de una VDT RS232C operando a 2400 baudios con siete bits de datos, un bit de paro, y una paridad impar. La edición de la línea de comandos es posible utilizando las teclas de BACKSPACE y DELETE para corregir errores. Se utilizan las secuencias de escape de la VT100 (vea a continuación).

Se soportan los siguientes cuatro comandos:

DUMP	vacía a la consola el contenido de un rango de ubicaciones en memoria
SET	examina ubicaciones individuales en memoria y modifica su contenido a un nuevo valor
GO	va al programa del usuario en una dirección especificada
LOAD	carga un archivo en formato hexadecimal de Intel

FORMATO DE COMANDOS

Los comandos se capturan utilizando el siguiente formato:

<c1><c2><p1>,<p2>,<p3>,<p4><CR>

donde <c1>y<c2>forman un comando de dos caracteres

<p1>a<p4>son parámetros a los comandos

<CR>es la tecla de RETURN

Los siguientes comentarios generales describen la captura de comandos en el MON51:

Cuando el sistema se inicializa o reinicializa, el indicador “V12>” aparece en la VDT.

- Los comandos se capturan desde el teclado y se ejecutan al oprimir la tecla INTRO.
- Durante la captura de comandos, cualquier error puede corregirse utilizando las teclas de RETROCESO o SUPRIMIR.
- La longitud máxima de una línea es de 20 caracteres.
- La combinación CONTROL-C termina la ejecución de un comando en cualquier momento y regresa el control al indicador.
- Una línea de comandos vacía (sólo la tecla INTRO) equivale de manera predeterminada al comando SET.
- Cada comando utiliza de 0 a 4 parámetros capturados en hexadecimal. (*Nota:* No se necesita la “H”).
- Los parámetros están separados por coma.
- Cualquier parámetro puede omitirse utilizando dos comas.
- Los parámetros omitidos equivaldrán a sus valores anteriores de manera predeterminada.
- La combinación CONTROL-P dispara un bit de habilitación para la salida a impresora. (*Nota:* Si la salida a impresora está habilitada, la impresora debe estar conectada y en línea; de lo contrario, el MON51 queda esperando a la impresora y no se envía salida a la consola).

SINTAXIS COMÚN

Dump

Formato: D<m><inicio>,<fin><CR>

Donde: D es el identificador del comando

<m>es el selector de espacio en memoria como sigue:

B—memoria direccionable por bit

C—memoria para código

I—memoria interna para datos

R—registros con funciones especiales

X—memoria externa para datos

<inicio>es la dirección inicial

<fin>es la dirección final

<CR>es la tecla de RETURN

(*Nota:* La salida a la consola puede suspenderse mediante la combinación CONTROL-S (XOFF). CONTROL-Q (XON) vuelve a iniciar la salida).

Ejemplo: DX8000,8100<CR>vacía el contenido de 257 ubicaciones externas para datos de las direcciones 8000H hasta la 8100H.

Set

Formato: S<m><dirección><CR>seguido por uno de estos elementos:

<valor>

<SP>

—

<CR>

Q

Donde: S es el identificador del comando

<m>es el selector de espacio en memoria como sigue:

B—memoria direccionable por bit

C—memoria para código

I—memoria interna para datos

R—registros con funciones especiales

X—memoria externa para datos

<dirección>es la dirección de una ubicación en memoria a examinar y/o modificar a cierto valor

<valor>es un valor de datos a escribir en la memoria

<SP>es la barra espaciadora utilizada para escribir el mismo valor en la siguiente ubicación

<CR>es la tecla de RETURN utilizada para examinar la siguiente ubicación

Q terminar y regresar al indicador del MON51

Ejemplo: SI90<CR> seguido de A5<CR> modifica el valor del puerto 1 al valor A5H

Go

Formato: GO<dirección><CR>

Donde: GO es el identificador del comando

<dirección>es la dirección en memoria para código donde inicia la ejecución

<CR>es la tecla de RETURN

Ejemplo: GO8000<CR>carga el valor 8000H al contador de programa

Lo

Formato: LO<CR>

Donde: LO es el identificador del comando

<CR>es la tecla de RETURN

Ejemplo: LO<CR> El mensaje “<descarga del anfitrón al SBC-51”> aparece. El MON51 recibe un archivo en formato hexadecimal de Intel en el puerto serial y escribe los bytes recibidos a la memoria externa para datos. Cuando la transferencia se completa, aparece el mensaje “EOF—descarga de archivo OK”. *Nota:* Utilice el comando DUMP para verificar que la transferencia fue exitosa.

OPERACIÓN GENERAL DEL MON51

Podemos describir la operación general del MON51 siguiendo estos pasos:

1. Inicializar los registros y las ubicaciones en memoria.
2. Enviar el indicador a la VDT.
3. Obtener una línea de comando del teclado de la VDT.
4. Decodificar el comando.
5. Ejecutar el comando.
6. Ir al paso 2.

Aunque en la operación general del MON51 participan muchas subrutinas y módulos de programa, la estructura básica para implementar los pasos indicados líneas arriba se encuentra en PRINCIPAL.LST. Las instrucciones para inicializar los registros y las ubicaciones en memoria (paso 1) son las líneas 171 a 200 (consulte las páginas 467-468). El indicador se envía a la VDT en las líneas 201 y 202, y entonces se captura un comando mediante el teclado de la VDT en las líneas 203 y 204 (pasos 3 y 4). El comando se decodifica en las líneas 205 a 224. El MON51 está diseñado para soportar 26 comandos, uno para cada letra del alfabeto. Sin embargo, sólo cuatro comandos están implementados, mientras que los otros son para futuras expansiones o aplicaciones personalizadas. La decodificación del comando involucra (a) la utilización del primer carácter localizado en el búfer de la línea de entrada para buscar la dirección del comando desde la tabla en las líneas 227 a 252, (b) el almacenamiento de la dirección del comando en la pila (1 byte a la vez), y (c) recuperar la dirección en el contador de programa utilizando la instrucción RET (línea 225).

El código para los comandos DUMP, SET y LOAD está contenido en archivos separados, mientras que el código para el comando GO está en PRINCIPAL.LST en las líneas 268 a 274. Se utilizan muchas subrutinas a lo largo del MON51, y pueden encontrarse en los diversos listados proporcionados.

La subrutina OBTENPAR (línea 224) es llamada durante la decodificación de un comando. Esta subrutina obtiene los parámetros de la línea de entrada (que están almacenados en el búfer de línea como caracteres en ASCII) y los convierte a binario, colocando los resultados en la memoria RAM interna en cuatro ubicaciones de 16 bits comenzando en la etiqueta PARMTR (consulte OBTENPAR.LST, línea 58, pág. 471). Los comandos DUMP, SET y GO requieren de estos parámetros.

Después de ejecutar cada comando, el control regresa al MON51 en la etiqueta OBTENCMD (línea 200 en PRINCIPAL.LST) y se repiten los pasos anteriores. Los programas de usuario pueden terminar utilizando la dirección de OBTENCMD (00BCH) como el destino de una instrucción LJMP.

El MON51 contiene numerosos comentarios, y cada subrutina y archivo contienen un bloque de comentarios que explica la operación general de la sección de código que le sigue. Los lectores interesados en conocer lo intrincado de la operación del MON51 pueden consultar los archivos de listado y las líneas de comentarios para obtener mayores detalles. En la siguiente sección nos ocuparemos del diseño general del MON51.

EL DISEÑO DEL MON51

También es importante desarrollar cierta comprensión del “diseño” (aparte de la operación), ya que el MON51 incorpora —en una escala relativamente grande— muchos de los conceptos desarrollados previamente en los capítulos en ejemplos breves. Los siguientes párrafos explican con detalle las características de diseño clave que pueden adoptarse en el desarrollo de software para los diseños basados en el 8051. En cierto sentido, entonces, este apéndice es como un caso de estudio —describe los detalles del diseño de un producto de software basado en el 8051 existente.

Los controles del ensamblador, como se mencionó en el capítulo 7, se utilizan principalmente para controlar el formato y el contenido de los archivos de listado creados por el ASM51 y el RL51. Varios controles del ensamblador aparecen al inicio de cada uno de los nueve listados del programa; y fueron seleccionados sobre todo para producir archivos de listado con los cuales satisfacer los requisitos de su reproducción en este apéndice. Observemos que en PRINCIPAL.LST, por ejemplo, ese utiliza el control \$NOLIST en la línea 6. Utilizamos esto porque la línea 7 (no presente en el listado) contenía el control \$INCLUDE(MACROS.SRC), el cual dirigía al ASM51 al archivo que contiene las definiciones de las macros (págs. 493-494). Al deshabilitar o habilitar la opción “de listado” justo antes y después de la sentencia \$INCLUDE, pudimos prevenir que el ASM51 incluyera las definiciones de las macros innecesariamente en PRINCIPAL.LST.

Muchas de las directivas del ensamblador soportadas por el ASM51 aparecen a lo largo de los archivos de listado del MON51. Recordemos (sección 7.5 Directivas del ensamblador) que el ASM51 soporta cinco categorías de directivas:

- Control del estado del ensamblador
- Definición de símbolos
- Inicialización/reservación de almacenamiento
- Enlazado de programa
- Selección de segmentos

En los listados aparecen ejemplos para cada una de estas categorías.

Control del estado del ensamblador. Las únicas directivas del control del estado del ensamblador utilizadas en el MON51 son END y USING. La directiva ORG nunca se utilizó, debido a que los segmentos para código y para datos fueron diseñados para ser relocalizables con las direcciones establecidas sólo al tiempo de enlazado.¹ Una aparición de USING puede encontrarse en OBTENPAR.LST, línea 108 en la página 472.

Definiciones de símbolos. Muchos de los símbolos están definidos a lo largo del MON51, como por ejemplo en las líneas 115 a 127 de PRINCIPAL.LST (págs. 466-467). Observe en particular las definiciones de EPROM, ENCHIP y BITRAM. EPROM está definido como el nombre para un segmento de código. De la misma forma, ENCHIP está definido como el segmento de datos y BITRAM como el segmento de bit. Estos tres segmentos son, por definición, relocalizables. Recordemos que el segmento de tipo “DATA” corresponde al espacio para datos incorporado en el chip del 8051 accesible mediante un direccionamiento directo (00H a 7FH). En el MON51, todas las instrucciones se ubican en el segmento para código de la EPROM, todas las ubicaciones de datos incorporados en el chip están definidas en el segmento para datos ENCHIP, y todas las ubicaciones de bits se definen en el segmento BITRAM.

Inicialización/reservación de almacenamiento. La directiva de definición de byte (DB) se utiliza a lo largo del MON51 para inicializar memoria para código con constantes de byte. Generalmente, la definición es para cadenas de caracteres en ASCII enviadas a la consola como

¹La única excepción a esto es la definición absoluta de la pila utilizando la directiva DSEG (consulte la página 469, línea 288).

parte de la operación normal del MON51. Por ejemplo, los caracteres para el indicador del MON51 están definidos en las líneas 288 y 289 de PRINCIPAL.LST.

Otras definiciones incluyen la secuencia de escape enviada a la consola para regresar el cursor. El MON51 está diseñado para trabajar con terminales compatibles con la VT100. Si se presiona la tecla BACKSPACE o la DELETE mientras se captura una línea, la secuencia de escape <ESC>[D(o los bytes hexadecimales 1BH, 5BH y 44H)] debe enviarse a la terminal. Estos bytes están definidos como una cadena en ASCII terminada en nulo (consulte el archivo ES.LIST, línea 144, pág. 475) utilizando la directiva DB. Una vez detectado el código para las teclas BACKSPACE o DELETE (vea las líneas 17 y 18, pág. 473, y las líneas 133 y 135, pág. 475), el apuntador del búfer de la línea de entrada disminuye dos veces y la secuencia de escape del “regreso del cursor” se envía a la consola utilizando la subrutina de cadena de salida (CADSAL). (Vea las líneas 136 a 139, pág. 475).

Las ubicaciones de almacenamiento se reservan utilizando ya sea la directiva de definición de almacenamiento (DS) o la directiva de definición de bit (DBIT). La primera ocurrencia de una directiva DS define una pila de 24 bytes (vea la línea 288 en PRINCIPAL.LST) en un segmento para datos absoluto, comenzando en la dirección 08H en la memoria RAM interna. El MON51 no utiliza los bancos de registros 1 a 3, así que las direcciones de memoria RAM interna 08H a 1FH han sido reservadas para la pila. El valor predeterminado 07H para el apuntador de pila asegura que la primera escritura a la pila es en la dirección 08H. En la línea 296 de PRINCIPAL.LST se reserva un búfer de 20 bytes para la línea de comandos. Diversas ubicaciones de bit se reservan en el segmento de bit BITRAM al final de PRINCIPAL.LST.

Enlazado del programa. Las directivas PUBLIC y EXTERN se utilizan cerca del inicio de la mayoría de los archivos fuente. PUBLIC declara qué símbolos definidos en cierto archivo estarán disponibles para utilizarse en otros archivos fuente. EXTRN declara cuáles símbolos se utilizan en el archivo fuente actual pero que están definidos en otro archivo fuente. La directiva NAME no se utiliza en el MON51; por lo tanto, cada archivo es un módulo donde el nombre del archivo sirve como nombre del módulo.

Diversos archivos del MON51 contienen el código fuente para subrutinas utilizadas en otros lugares. Por ejemplo, ES.LST (págs. 473-478) contiene el código fuente para las subrutinas de entrada/salida CARENT, CARSAL, LINEAENT, CADSAL, SALIDAHEX, y SALIDA2HX. Estas subrutinas se declaran como “públicas” en la línea 26 (pág. 473). Otros archivos que utilicen estas subrutinas deben contener una instrucción que declare a estos símbolos como símbolos de direcciones de código externos (por ejemplo, en PRINCIPAL.LST, línea 108, pág. 466). Cuando llamamos a estas subrutinas, el ASM51 no conoce la dirección para la instrucción LCALL o ACALL, así que la letra “F” aparece en el archivo de listado para indicar que se requiere de enlace/localización para corregir (“fix”) la instrucción. Por ejemplo, en PRINCIPAL.LST el indicador se envía a la consola en la línea 202 utilizando ACALL CADSAL. Esta instrucción aparece en el listado como 1100H con una “F” a su lado. El enlazador/localizador, RL51, determinará la dirección absoluta correcta de las subrutinas CADSAL y corregirá la instrucción. En este ejemplo, debido a que el modo de direccionamiento es absoluto dentro de la página de 2K actual, la corrección sustituye 11 bits en la instrucción —tres bits en el byte superior y los ocho bits del byte inferior.

Selección de segmentos. Las directivas de selección de segmentos son RSEG, para seleccionar un segmento relocalizable, y las cinco directivas para seleccionar un segmento absoluto de un espacio de memoria especificado (DSEG, BSEG, XSEG, CSEG e ISEG). La directiva RSEG se utiliza por lo menos una vez en cada archivo para iniciar el segmento de código EPROM (por ejemplo, en PRINCIPAL.LST, línea 129, pág. 467). Otras instancias de RSEG aparecen según sea necesario para seleccionar el segmento de datos ENCHIP (por ejemplo, en la línea 295 de PRINCIPAL.LST) o el segmento de bit BITRAM (línea 306, pág. 470, de PRINCIPAL.LST). La única instancia de un segmento absoluto es la definición de la pila en la dirección de RAM interna absoluta 08H (PRINCIPAL.LST, línea 287, pág. 469).

PUENTES DE CONFIGURACIÓN DE OPCIONES

La SBC-51 tiene tres puentes de configuración en el puerto 1 que se leen mediante software para evocar características especiales. El MON51 lee estos puentes después de una reinicialización del sistema y activa o borra las ubicaciones de bit correspondientes (vea PRINCIPAL.LST líneas 171 a 176). Cada puente tiene un propósito especial, como se describe a continuación.

Puente	Instalado	Propósito
X3	NO	ejecución normal
	SÍ	salta a 2000H después de una reinicialización
X4	NO	tabla de saltos de interrupciones en 80xxH
	SÍ	tabla de saltos de interrupciones en 20xxH
X13	NO	ejecución normal
	SÍ	UART por software (lea características en la página siguiente)

Hay que tener cuidado al crear una interfaz de circuitos de E/S con el puerto 1. Si la interfaz conectada presenta un 0 lógico a la terminal correspondiente durante una reinicialización del sistema, el MON51 lo interpreta como si el puente estuviera instalado, y la opción mencionada arriba surte efecto.

Punto de entrada de reinicialización. El puente X3 puede instalarse para forzar un salto a 2000H inmediatamente después de una reinicialización del sistema (vea la línea 199 en PRINCIPAL.LST). Esto es útil para poder tener tanto la EPROM del MON51 como una EPROM de usuario instaladas en forma simultánea, y para seleccionar cuál de ellas se ejecuta después de la reinicialización. Si, por ejemplo, el programa de usuario no utiliza una VDT, entonces no es conveniente encender en “modo de MON51” sólo para enviar el comando GO2000H para evocar el programa de usuario. La instalación del puente X3 evita la necesidad de hacer esto.

Tablas de saltos de interrupciones. Aunque el MON51 no utiliza interrupciones, las aplicaciones de usuario que coexisten con él pueden adoptar un diseño controlado por interrupciones. Ya que el MON51 reside en la EPROM empezando en la dirección 0000H, y *ya que* todas las interrupciones se vectorizan a ubicaciones cerca de la parte inferior de la memoria, se desarrolló una tabla de saltos para permitir que las aplicaciones de usuario se ejecuten en otras direcciones para utilizar las interrupciones. Basándonos en la configuración del hardware predeterminado de la SBC-51, las aplicaciones de usuario generalmente se ejecutan ya sea en RAM, empezando en la dirección 8000H, o en EPROM, empezando en 2000H. Si una aplicación de usuario habilita las interrupciones y enseguida ocurre y se acepta una interrupción, la dirección de vector de interrupción (en otras palabras, el punto de entrada) es una de las direcciones 0003H, 000BH, y 0013H, etc., dependiendo de la fuente de interrupción (consulte la tabla 6-4). En cada dirección de vector de interrupción del MON51 existe una secuencia corta de instrucciones que salta a una ubicación ya sea en la

PUNTO DE ENTRADA DE INTERRUPCIÓN

Fuente de interrupción	X4 instalado	X4 no instalado
(punto de entrada predeterminado)	2000H	8000H
External 0	2003H	8003H
Timer 0	2006H	8006H
External 1	2009H	8009H
Timer 1	200CH	800CH
Serial Port	200FH	800FH
Timer 2	2012H	8012H

EPROM de usuario en 20xxH o en RAM en 80xxH, dependiendo de si el puente X4 está instalado o no en la SBC-51 (vea las líneas 150 a 183 de PRINCIPAL.LST). A continuación presentamos el punto de entrada correcto para cada interrupción.

Los puntos de entrada mencionados están separados por tres ubicaciones entre cada uno, lo cual deja justo el espacio suficiente para enviar una instrucción LJMP a la rutina de servicio de interrupción. Cuando la operación está basada en un cristal de 12 MHz, se agregan 6 μ s al tiempo de espera de la interrupción por el tiempo perdido en este esquema (debido a las instrucciones JNB y LJMP en el MON51 y a la instrucción LJMP en el programa de usuario). Por supuesto que los programas pueden comenzar en las direcciones 2000H u 8000H y proceder a través de los puntos de entrada mencionados líneas arriba, siempre y cuando las aplicaciones de usuario no utilicen interrupciones.

Como ejemplo de una aplicación de usuario que utiliza interrupciones y coexiste con el MON51, consulte el proyecto de interfaz a un MC14499 en el capítulo 11.

UART por software. Cuando el puente X13 está instalado, el MON51 implementa un UART por software para la entrada/salida serial en lugar de utilizar el puerto serial incorporado en el chip del 8051. Esta característica es útil para desarrollar interfaces a dispositivos seriales que no sean la terminal predeterminada. Observe que la velocidad en baudios es de 1200 si se utiliza el UART por software. La operación del UART por software se describe en ES.LST (líneas 219-278, pág. 477).

EL MAPA DE ENLACE Y LA TABLA DE SÍMBOLOS

Uno de los listados más importantes es el que produce el enlazador/localizador RL51. Este listado, V12.M51 (págs. 494-496), contiene un mapa de enlace y una tabla de símbolos. El mapa de enlace muestra la dirección inicial y el tamaño de los tres segmentos relocalizables utilizados por el MON51 (pág. 495). Las direcciones son absolutas en esta etapa, ya que los segmentos han sido ubicados por el RL51 con base en las especificaciones proporcionadas en la línea de invocación (pág. 494). La tabla de símbolos indica los valores absolutos asignados a todos los símbolos relocalizables (como fueron asignados por el RL51 al tiempo de enlazado). Está dividida alfabéticamente por módulo (en otras palabras, archivo) en el orden en que fueron enlazados los símbolos.

Como ejemplo de esto, la dirección absoluta de la subrutina CADSAL se determina buscando el símbolo CADSAL en el módulo ES (págs. 495-496). La dirección de CADSAL en el MON51 versión 12 es 0282H.

ARCHIVO HEXADECIMAL DE INTEL

El último listado (que encontrará en el sitio Web de este libro) se produce mediante la herramienta de conversión OH (objeto a hexadecimal). El listado V12.HEX (págs. 496-498) contiene los bytes en lenguaje máquina del programa MON51 en el formato hexadecimal de Intel. Por ejemplo, la subrutina CADSAL inicia en la dirección 0282H. En la página 497, los primeros tres bytes de esta subrutina aparecen como E4H, 93H, y 60H. Si nos referimos directamente a la subrutina CADSAL en el archivo ES.LST, podremos confirmar que estos valores son correctos (vea líneas 156-158, pág. 476).

MÓDULO PRINCIPAL PARA EL PROGRAMA DE MONITOREO DEL 8051

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN MAIN.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE MAIN.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$title(** MAIN MODULE FOR 8051 MONITOR PROGRAM **)
		3	\$pagewidth(98)
		4	\$nosymbols ;symbol table in .M51 file created by RL51
		5	\$nopaging
		6	\$nolist ;next line contains \$include(macros.src)
		76	;previous line contains \$list
		77	*****
		78	;
		79	MON51
		80	;
		81	(An 8051 Monitor Program)
		82	;
		83	: Copyright (c) I. Scott MacKenzie, 1988, 1991
		84	: Dept. of Computing & Information Science
		85	: University of Guelph
		86	: Guelph, Ontario N
		87	: Canada NIG 2W1
		88	;
		89	: VERSION 12 - April 1991
		90	;
		91	: COMMANDS: D - dump memory to console
		92	: G - go to user program
		93	: L - load hex file
		94	: S - set memory to value
		95	;
		96	: RESET OPTIONS:
		97	;
		98	: JUMPER INSTALLED RESULT
		99	-----
		100	: X3 NO Execute MON51
		101	: X3 YES Jump to 2000H upon reset
		102	: X4 NO Interrupt jump table in user RAM (8000H)
		103	: X4 YES Interrupt jump table in user EPROM (2000H)
		104	: X13 NO Serial I/O using P3.1 (TXD) and P3.0 (RXD)
		105	: X13 YES Serial I/O using P1.7 (TXD) and P1.6 (RXD) and
		106	interrupts (except if X4 installed)
		107	*****
		108	extr code(outchr, inline, outstr, htoa, atoh, getpar)
		109	extr code(load, dump, setcmd, serial_io_using_interrupts)
		110	extr data(parmtr, tb_count)
		111	extr bit(t_flag, r_flag, r_idle)
		112	public getcmd, linbuf, buflen, endbuf, notice
		113	public error, p_bit, riot, ram, rom, x13_bit
		114	;
0014		115	buflen equ 20 ;maximum line length
0007		116	bel equ 07H ;ASCII bell code
000D		117	cr equ 0DH ;ASCII carriage return code
000A		118	lf equ 0AH ;ASCII line feed code
0100		119	riot xdata 0100h ;8155 RAM/IO/TIMER SBC51
8000		120	ram equ 8000h ;RAM address for user software
2000		121	rom code 2000h ;EPROM address for user firmware
		122	eprom segment code
		123	onchip segment data
		124	bitram segment bit
0095		125	x13 bit p1.5 ;reset option

0094		126	x3	bit	p1.4
0093		127	x4	bit	p1.3
---		128			
0000 020000	F	129	rseg	eprom	
0003 300034	F	130	main:	ljmp	skip ;jump above interrupt vectors
0006 028003		131		jnb	x4_bit,rom3 ;external 0 interrupt entry point
0009 0000		132		ljmp	ram+3
0008 30002F	F	133		dw	0
000E 028006		134		jnb	x4_bit,rom6 ;timer 0 interrupt entry point
0011 0000		135		ljmp	ram+6
0013 30002A	F	136		dw	0
0016 028009		137		jnb	x4_bit,rom9 ;external 1 interrupt entry point
0019 0000		138		ljmp	ram+9
0018 300025	F	139		dw	0
001E 020000	F	140		jnb	x4_bit,rom12 ;timer 1 interrupt entry point
0021 0000		141		ljmp	check ;check jumper X13 first
0023 300020	F	142		dw	0
0026 02800F		143		jnb	x4_bit,rom15 ;serial port interrupt entry point
0029 0000		144		ljmp	ram+15
0028 30001B	F	145		dw	0
002E 028012		146		jnb	x4_bit,rom18 ;timer 2 interrupt entry point
		147		ljmp	ram+18
		148			
		149			*****
		150			; Timer 1 interrupts are used for serial I/O if jumper X13 is *
		151			; installed. P1.7 is used for TXD and P1.6 is used for RXD. *
		152			; (See IO module) *
		153			*****
0031 200003	F	154	check:	jb	x13_bit,ram12 ;if X13 installed, serial I/O using
0034 020000	F	155		ljmp	serial_io_using_interrupts ;interrupts
0037 02800C		156	ram12:	ljmp	ram+12 ;if X13 not installed, use RAM table
003A 022003		157	rom3:	ljmp	rom+3 ;if X4 is installed, interrupts
003D 022006		158	rom6:	ljmp	rom+6 ;are directed to a jump table in
0040 022009		159	rom9:	ljmp	rom+9 ;user EPROM at 2003H (otherwise
0043 02200C		160	rom12:	ljmp	rom+12 ;directed to jump table in user
0046 02200F		161	rom15:	ljmp	rom+15 ;RAM at 8003H; see above)
0049 022012		162	rom18:	ljmp	rom+18
		163			
004C 436F7079		164	notice:	db	'Copyright (c) I. Scott MacKenzie, 1988, 1991'
0050 72696768					
0054 74202863					
0058 2920492E					
005C 2053636F					
0060 7474204D					
0064 61634865					
0068 6E7A6965					
006C 2C203139					
0070 38382C20					
0074 31393931					
		165			
		166			*****
		167			; Copy state of pins used for reset jumpers to flag bits. Reset *
		168			; jumpers are only read upon reset. Subsequent tests are on the *
		169			; flag bits. *
		170			*****
0078 A294		171	skip:	mov	c,x3
007A 9200	F	172		mov	x3_bit,c
007C A293		173		mov	c,x4
007E 9200	F	174		mov	x4_bit,c
0080 A295		175		mov	c,x13
0082 9200	F	176		mov	x13_bit,c
0084 200003	F	177		jb	x3_bit,skip4 ;If x3 jumper installed,
0087 022000		178		ljmp	rom ; jump to user EPROM at 2000H
008A 200017	F	179	skip4:	jb	x13_bit,skip5 ;If x13 jumper installed,

008D C200	F	180	clr	r_flag	; r_flag = 0 (no character waiting)
008F D200	F	181	setb	r_idle	; r_idle = 1 (receiver idle)
0091 D200	F	182	setb	t_flag	; t_flag = 1 (ready to transmit)
0093 750008	F	183	mov	tb_count,#11	; 11 bits sent (start + 8 + stop)
0096 758920		184	mov	tmod,#20H	; 8-bit auto-reload mode
0099 758098		185	mov	th1,#-104	; 1 / (8 x 0.0012) us for 1200 baud
009C D28E		186	setb	tr1	; start timer 1 (interrupts coming)
009E D2AF		187	setb	ea	; turn on timer 1 interrupts - serial
00A0 D2AB		188	setb	et1	; I/O uses P1.7 (TXD) and P1.6 (RXD)
00A2 8008		189	sjmp	skip6	; and skip over serial port init
00A4 759852		190	skip5:	mov scon,#01010010B	;if x13 not installed, initialize
00A7 758DF3		191	mov	th1,#0F3H	; for 2400 baud reload value
00AA 758920		192	mov	tmod,#20H	;auto reload mode
00AD D28E		193	setb	tr1	;start timer
00AF C200	F	194	skip6:	clr p_bit	;default = no printer output
0081 900100		195	mov	dptr,#riot	;address of 8155
0084 7401		196	mov	a,#1	;Port A = output
0086 F0		197	movx	@dptr,a	;initialize printer port
0087 900000	F	198	mov	dptr,#hello	;send hello message to console
008A 1100	F	199	acall	outstr	
008C 758107		200	getcmd:	mov sp,#stack - 1	;initialize stack pointer
008F 900000	F	201	mov	dptr,#prompt	;send prompt to console
00C2 1100	F	202	acall	outstr	
00C4 7800	F	203	mov	r0,#linbuf	;R0 points to line buffer
00C6 1100	F	204	acall	inline	;input command line from console
00C8 E500	F	205	mov	a,linbuf	;get first character
00CA 840003		206	cjne	a,#cr,skip7	;empty line?
00CD 020000	F	207	jmp	setcmd	;yes: default to "set" command
		208	skip7:		;check for alphabetic character
0000 844100		209 +2	cjne	a,#'A', \$+3	;J0R
0003 404C		210 +2	jc	error	
0005 845800		211 +2	cjne	a,#'Z'+1, \$+3	
0008 5047		212 +2	jnc	error	
00DA 541F		213	anl	a,#1fh	;reduce to 5 bits
00DC 14		214	dec	a	;reduce ASCII 'A' to 0
00DD 23		215	rl	a	;adjust to word boundary
00DE F8		216	mov	r0,a	;save
00DF 04		217	inc	a	
00E0 900000	F	218	mov	dptr,#table	command address table
00E3 93		219	movc	a,@a+dptr	;get address low byte of command
00E4 COEO		220	push	acc	;save on stack
00E6 E8		221	mov	a,r0	;restore accumulator
00E7 93		222	movc	a,@a+dptr	;get address high byte of command
00E8 COEO		223	push	acc	;sneaky: pushing address on stack
00EA 1100	F	224	acall	getpar	;get parameters from linbuf
00EC 22		225	ret		;pop command address into PC and
		226			; off we go!
00ED 0000	F	227	table:	dw	;A (Note: most commands undefined)
00EF 0000	F	228		dw	;B
00F1 0000	F	229		dw	;C
00F3 0000	F	230		dw	;Dump memory locations
00F5 0000	F	231		dw	;E
00F7 0000	F	232		dw	;F
00F9 0000	F	233		dw	;Go to user program
00FB 0000	F	234		dw	;H
00FD 0000	F	235		dw	;I
00FF 0000	F	236		dw	;J
0101 0000	F	237		dw	;K
0103 0000	F	238		dw	;Load hex file
0105 0000	F	239		dw	;M
0107 0000	F	240		dw	;N
0109 0000	F	241		dw	;O
0108 0000	F	242		dw	;P
010D 0000	F	243		dw	;Q

```

010F 0000 F 244 dw error ;R
0111 0000 F 245 dw setcmd ;Set memory to value
0113 0000 F 246 dw error ;T
0115 0000 F 247 dw error ;U
0117 0000 F 248 dw error ;V
0119 0000 F 249 dw error ;W
011B 0000 F 250 dw error ;X
011D 0000 F 251 dw error ;Y
011F 0000 F 252 dw error ;Z
0121 900000 F 253 error: mov dptr,#emess ;unimplemented command
0124 1100 F 254 acall outstr
0126 E500 F 255 mov a,linbuf ;send out 1st two characters in
0128 1100 F 256 acall outchr ; line buffer too
012A E500 F 257 mov a,linbuf + 1
012C 1100 F 258 acall outchr
012E 808C 259 jmp getcmd
260
261 ****
262 ; GO to user program command. This command is small enough to *
263 ; include with the main module. *
264 ;
265 ; FORMAT: GO<address>
266 ;
267 ****
0130 7400 F 268 go: mov a,#low(getcmd) ;if "ret" from user program,
0132 COE0 269 push acc ; save "getcmd" address on stack
0134 7400 F 270 mov a,#high(getcmd)
0136 COE0 271 push acc
0138 C000 F 272 push permtr + 1 ;push address of user program on
013A C000 F 273 push permtr ; stack
013C 22 274 ret ;pop address into PC (Off we go!)
275
276 ****
277 ; ASCII null-terminated strings *
278 ****
013D 00 279 hello: db cr,'MON51',0
013E 404F4E35
0142 31
0143 00
0144 0D 280 prompt: db cr,'V12>',0
0145 5631323E
0149 00
014A 07 281 emess: db bel,cr,'Error: Invalid Command - ',0
014B 00
014C 4572726F
0150 723A2049
0154 6E76616C
0158 69642043
015C 6F606061
0160 6E642020
0164 20
0165 00
282
283 ****
284 ; Reserve 24 bytes for the stack in an absolute segment starting at *
285 ; 08H. (Note: MON51 does not use register banks 1-3.) *
286 ****
---- 287 dseg at 8
0008 288 stack: ds 24
289
290 ****
291 ; Create a line buffer for monitor commands in a relocatable data *
292 ; segment. See linker/locator listing (.M51) for absolute address *
293 ; assigned.
294 ****

```

```

0000      296    linbuf:      ds      buflen      ;input command line goes here
0014      297    endbuf:     equ      $ 
298
299    ****
300    ; Create 1-bit flags. If p_bit flag set, OUTCHR transmits to the *
301    ; console and the printer; if clear, output only sent to the   *
302    ; console (default). "x" bits are set to the state of the reset   *
303    ; jumpers as read immediately after reset. Consult the link map   *
304    ; to determine the exact (i.e., absolute) location of these bits. *
305    ****
306          rseg    bitram
---- 
0000      307    p_bit:      dbit    1
0001      308    x3_bit:    dbit    1
0002      309    x4_bit:    dbit    1
0003      310    x13_bit:   dbit    1
311

```

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

OBTÉN PARÁMETROS DE LA LÍNEA DE COMANDOS

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN GETPAR.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE GETPAR.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$title(** GET PARAMETERS FROM INPUT LINE ***)
		3	\$pwidth(98)
		4	\$nosymbols
		5	\$nopaging
		6	*****
		7	;
		8	GETPAR.SRC - GET PARAMETERS
		9	;
		10	; This routine scans the characters in the input buffer "linbuf"
		11	; and extracts the parameters used by the monitor commands. The
		12	; input line must be of the form:
		13	;
		14	; <C1><C2><P1>,<P2>,<P3>,<P4><CR>0
		15	;
		16	; where <C1> and <C2> are the 2-character command and <P1> to <P4>
		17	; are up to 4 parameters required by the commands. Scanning begins
		18	; after the 2-character command (i.e., at linbuf+2).
		19	;
		20	; Parameters are entered as hex ASCII codes separated by
		21	; commas. The hex ASCII codes are converted to 16-bit binary
		22	; values and placed in memory beginning at "permtr" in the order
		23	; high byte/low byte.
		24	;
		25	; Any parameters omitted retain their value from the previous
		26	; command. Any illegal hex ASCII code terminates the scanning.
		27	;
		28	*****
		29	extrn code(ishex, atoh)
		30	extrn data(linbuf)

```

31     public getpar, parmtr, reg_sp, bit_sp, int_sp, ext_sp, cde_sp
32
0004      numpar equ 4           ;4 input parameters
33      eeprom segment code
34      onchip segment data
35      bitram segment bit
36
37
---- 38      rseg eeprom
39      getpar: mov r0,#linbuf+2 ;R0 points to input line data
0002 7800 F   40      mov r1,#parmtr ;R1 points to parameters destination
0002 7900 F   41      mov r7,#numpar ;use R7 as counter
0004 7F04      42      cjne @r0,',',getp3 ;parameter missing?
0006 B62C02      43      sjmp getp4 ;yes: try to convert next parameter
0009 8008      44      getp3: acall gethx ;no: convert it
0008 1100 F   45      jnb found,getp5 ;if illegal byte, exit
0000 300008 F   46      cjne @r0,',',getp5 ;next character should be ','
0010 B62C05      47      getp4: inc r0
0013 08      48      inc r1           ;point to next parameter destination
0014 09      49      inc r1
0015 09      50      djnz r7,getp2 ;and let's try for another
0016 DFEE      51      getp5: acall space ;determine memory space selected
0018 1100 F   52      ret
001A 22
53
54      ****
55      ; Create space for the parameters in the "onchip" data segment.
56      ****
---- 57      rseg onchip
58      parmtr: ds numpar*2 ;2 bytes required for each parameter
59
60      ****
61      ; GETHX - GET HeX word from "linbuf"
62      ;
63      ; enter: R0 points to string
64      ;          R1 points to 2-byte destination
65      ; exit: hex word in memory
66      ;          R0 points to next non-hex character in string
67      ;          'found' = 1 if legal value found or 0 otherwise
68      ; uses: atoh, insrt, ishex
69      ;
70      ****
---- 71      rseg eeprom
72      gethx: clr found ;default = no value found
001B C200 F   73      mov a,@r0
0010 E6      74      acall ishex ;any value?
001E 1100 F   75      jnc gethx9 ;no: return
0020 5013      76      gethx2: setb found ;set value found flag
0022 D200 F   77      clr a
0024 E4      78      mov @r1,a ;clear destination value
0025 F7      79      inc r1
0026 09      80      mov @r1,a
0027 F7      81      dec r1
0028 19
0029 E6      82      gethx3: mov a,@r0 ;get value
002A 1100 F   83      acall ishex ;value found?
002C 5007      84      jnc gethx9 ;no: return
002E 1100 F   85      acall atoh ;yes: attempt to convert it
0030 1100 F   86      acall insrt ;and insert into parameter
0032 08      87      inc r0 ;increment buffer pointer
0033 80F4      88      sjmp gethx3 ;repeat until illegal value
0035 22      89      gethx9: ret
90
91      ****
92      ; Create space for a "found" bit which is set as the command line
93      ; is scanned.
94      ****

```

```

----          95      rseg    bitram
0000        96      found:  dbit     1
97
98 ;*****
99 ; INSRT - INSeRT nibble from A into 16-bit value pointed at by R1   *
100 ;
101 ;      enter: ACC.0 to ACC.3 contains hex nibble                   *
102 ;      R1 points to 2-byte RAM location                            *
103 ;      exit:  16-bit value shifted left 4 and nibble in ACC       *
104 ;              inserted into least significant digit               *
105 ;
106 ;*****
----          107     rseg    eprom
108     using    0
0036 C007 109     insrt: push  AR7           ;use R7 as counter
0038 C000 110     push    AR0           ;use R0 to point to lsb
003A A801 111     mov     r0,1          ;R0 <- R1
003C 08 112     inc     r0
003D 7F04 113     mov     r7,#4         ;4 rotates
003F C4 114     swap    a
0040 33 115     insrt2: rlc   a
0041 COE0 116     push    acc
0043 E6 117     mov     a,@r0
0044 33 118     rlc   a
0045 F6 119     mov     @r0,a
0046 E7 120     mov     a,@r1
0047 33 121     rlc   a
0048 F7 122     mov     @r1,a
0049 D0E0 123     pop     acc
004B DFF3 124     djnz   r7,insrt2
004D D000 125     pop     AR0          ;R0
004F D007 126     pop     AR7          ;R7
0051 22 127     ret
128
129 ;*****
130 ; SPACE - determine memory SPACE selector as per second character   *
131 ;      in command line as follows:                                     *
132 ;
133 ;      B = bit space                                         *
134 ;      C = code space                                         *
135 ;      I = internal data                                     *
136 ;      R = Special Functions Registers                      *
137 ;      X = external data                                     *
138 ;
139 ;*****
0052 C200 140     space: clr   bit_sp      ;default: no space selected
0054 C200 141     clr    cde_sp      ; clear all bits
0056 C200 142     clr    int_sp
0058 C200 143     clr    reg_sp
005A C200 144     clr    ext_sp
005C E500 145     mov    a,linbuf+1   ;memory space selector in command
005E B44202 146     cjne   a,'#'B',spac2 ;determine space and set selector bit
0061 D200 147     setb   bit_sp
0063 B44302 148     spac2: cjne   a,'#'C',spac3
0066 D200 149     setb   cde_sp
0068 B44902 150     spac3: cjne   a,'#'I',spac4
006B D200 151     setb   int_sp
006D B45202 152     spac4: cjne   a,'#'R',spac5
0070 D200 153     setb   reg_sp
0072 B45802 154     spac5: cjne   a,'#'X',spac6
0075 D200 155     setb   ext_sp
0077 22 156     spac6: ret
157

```

```

158 ;*****
159 ; Create space for the memory space selector bits in the 8031 *
160 ; bit-addressable space. See link map for exact address of each of *
161 ; these bits.
162 ;*****
163     rseg    bitram
0001   bit_sp: dbit  1
0002   cde_sp: dbit  1
0003   int_sp: dbit  1
0004   reg_sp: dbit  1
0005   ext_sp: dbit  1
169      end

```

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

RUTINAS DE ENTRADA/SALIDA

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN IO.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE IO.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$title(*** INPUT/OUTPUT ROUTINES ***)
		3	\$pagewidth(98)
		4	\$nopaging
		5	\$nosymbols
		6	;*****
		7	;
		8	; IO.SRC - INPUT/OUTPUT SUBROUTINES
		9	;
		10	;*****
0007		11	bel equ 07H ;ASCII bell code
000D		12	cr equ 0DH ;ASCII carriage return code
0018		13	esc equ 1BH ;ASCII escape code
000A		14	lf equ 0AH ;ASCII line feed code
0003		15	etx equ 3 ;control-C (terminate input, get next cmd)
0010		16	dle equ 10h ;control-P (output to printer as well)
007F		17	del equ 7FH ;delete
0004		18	bs equ 4 ;back space (VT100)
0097		19	TX_pin bit p1.7
0096		20	RX_pin bit p1.6
		21	
		22	extrn code(getcmd, htoa, touppr)
		23	extrn data(linbuf)
		24	extrn bit(p_bit, x13_bit)
		25	extrn number(endbuf, riot)
		26	public inchar, outchr, inline, outstr, outhex, out2hx
		27	public serial_io_using_interrupts, t_flag, r_flag
		28	public tb_count, r_idle
		29	
		30	eprom segment code
		31	onchip segment data
		32	bitram segment bit
		33	

```

34 ;*****
35 ; The following buffers & flags are needed for character I/O using *
36 ; interrupts. Interrupts are used only if jumper X13 is installed. *
37 ; Jumpers X9 and X10 should be removed and jumpers X11 and X12 *
38 ; should be installed. This connects the RS232 TXD and RXD lines *
39 ; to P1.7 and P1.6 (rather than to P3.1 and P3.0). *
40 ;*****
41 ;          rseg    onchip
0000 42 t_buff:      ds     1      ;transmit_buffer (character to send)
0001 43 r_buff:      ds     1      ;receive_buffer (character received)
0002 44 ti_count:   ds     1      ;transmit_interrupt_count
0003 45 ri_count:   ds     1      ;receive_interrupt_count
0004 46 tb_count:   ds     1      ;transmit_bit_count
0005 47 rb_count:   ds     1      ;receive_bit_count
48 ;          rseg    bitram
0000 49 t_flag:     dbit   1      ;1 = end of character transmission
0001 50 r_flag:     dbit   1      ;1 = end of character reception
0002 51 r_idle:     dbit   1      ;1 = idle state
52 ;
53 ;*****
54 ; OUTCHR -  OUTPut CHaRacter to serial port
55 ;           with odd parity added in ACC.7
56 ;
57 ; enter:   ASCII code in accumulator
58 ; exit:    character written to SBUF; <cr> sent as <cr><lf>; all
59 ;           registers intact
60 ;
61 ;*****
62 ;          rseg    eeprom
0000 COEO 63 outchr: push acc
0002 A2D0 64 nl:    mov c,p      ;add odd parity
0004 83 65     cpl c
0005 92E7 66     mov acc.7,c
0007 200000 F 67     jb  x13_bit,out1 ;if X13 installed, use interrupts
000A 3000FD F 68     jnb t_flag,$ ;wait for transmitter ready
0000 C2AB 69     clr et1      ;begin "critical section"
000F C200 F 70     clr t_flag ;>>> clear flag and ...
0011 F500 F 71     mov t_buff,a ;>>> load data to transmit
0013 D2AB 72     setb et1      ;end "critical section"
0015 8007 73     sjmp out2    ;if X13 not installed, test TI
0017 3099FD 74     out1: jnb ti,$ ;wait for transmitter ready
001A C299 75     clr ti      ;clear TI flag
001C F599 76     mov sbuf,a ;done!
001E C2E7 77     out2: clr acc.7 ;remove parity bit
0020 B40004 78     cjne a,#cr,out3
0023 740A 79     mov a,#lf
0025 8008 80     jmp nl      ;if <cr>, send <lf> as well
0027 D0E0 81     out3: pop acc
0029 300003 F 82     out4: jnb p_bit,out4 ;if printer control = off, exit
002C 120000 F 83     call pchar ;if on, print character
002F 22 84     ret
85     out4: ret
86 ;
87 ;*****
88 ; INCHR -  INPut CHaRacter from serial port
89 ;
90 ; enter:   no conditions
91 ; exit:    ASCII code in ACC.0 to ACC.6; ACC.7 cleared; control-C
92 ;           aborts to prompt
93 ;
94 ;*****
0030 200000 F 95 inchar: jb  x13_bit,in1 ;if X13 installed, use interrupts
0033 3000FD F 96     jnb r_flag,$ ;wait for receive_flag to be set
0036 C2AB 97     clr et1      ;begin "critical section"

```

```

0038 C200 F 98 clr r_flag      ;>>> clear receive_flag and ...
003A E500 F 99 mov a,r_buff   ;>>> read receive_buffer
003C D2AB 100 setb et1        ;end "critical section"
003E 8007 101 sjmp in2       ;if X13 not installed, test RI
0040 3098FD 102             ;wait for receiver interrupt
0043 C298 103 in1: jnb ri,$ ;clear RI flag
0045 E599 104 clr ri        ;done!
0047 C2E7 105 mov a,dbuf     ;done!
0049 B40303 106 in2: clr acc.7 ;clear parity bit (no error checking)
004C 020000 F 107 cjne a,#etx,in3 ;if control-C,
004F 22    108 jmp getcmd    ; warm start
0050 120000 F 109 in3: ret     ;*****+
0051 120000 F 110             ;*****+
0052 120000 F 111 ;*****+ ; INLINE - Input LINE of characters
0053 120000 F 112 ;      line must end with <cr>; maximum size set by buflen;
0054 120000 F 113 ;      <bs> or <del> deletes previous character
0055 120000 F 114 ;      enter: R0 points to input buffer in internal data RAM
0056 120000 F 115 ;      exit: ASCII codes in internal RAM; 0 stored at end of line
0057 120000 F 116 ;      uses: inchar, outchr
0058 120000 F 117 ;*****+
0059 120000 F 118 ;*****+
0060 120000 F 119 ;*****+
0061 120000 F 120 ;*****+
0062 120000 F 121 inline: call inchar ;get character from console
0063 120000 F 122 call touppr ;convert to uppercase, if necessary
0064 120000 F 123 cjne a,#idle,inlin7 ;is it control-P?
0065 120000 F 124 cpl p_bit ;yes: toggle printing flag bit
0066 120000 F 125 jmp inline
0067 120000 F 126 inlin7: call outchr ;echo character to port
0068 120000 F 127 mov @r0,a ;put in buffer
0069 120000 F 128 inc r0
0070 120000 F 129 cjne r0,#endbuf,inlin2 ;overflow?
0071 120000 F 130 mov dptr,#inlin8
0072 120000 F 131 acall outstr
0073 120000 F 132 jmp getcmd
0074 120000 F 133 inlin2: cjne a,#bs,inlin3 ;back space?
0075 120000 F 134 sjmp inlin5
0076 120000 F 135 inlin3: cjne a,#del,inlin6 ;delete?
0077 120000 F 136 inlin5: dec r0 ;yes: back up pointer twice
0078 120000 F 137 dec r0
0079 120000 F 138 mov dptr,#backup ;back up cursor
0080 120000 F 139 call outstr
0081 120000 F 140 jmp inline
0082 120000 F 141 inlin6: cjne a,#cr,inline ;character = <cr> ?
0083 120000 F 142 mov @r0,#0 ;yes: store 0
0084 120000 F 143 ret ;and return
0085 120000 F 144 backup: db esc,'D',0 ;VT100 sequence to backup cursor
0086 120000 F 145             ;*****+
0087 120000 F 146             ;*****+
0088 120000 F 147 ;*****+ ; OUTSTR - OUTput a STRING of characters
0089 120000 F 148 ;      enter: DPTR points to character string in external code
0090 120000 F 149 ;      memory: string must end with 00H
0091 120000 F 150 ;*****+

```

```

152 ; exit:    characters written to SBUF
153 ; uses:    outchr
154 ;
155 ;*****
00A4 E4 156 outstr: clr    a
00A5 93 157    movec   a,@a+dptr ;get ASCII code
00A6 6006 158    jz      outst2 ;if last code, done
00A8 120000 F 159    call    outchr ;if not last code, send it
00AB A3 160    inc     dptr   ;point to next code
00AC 80F6 161    sjmp   outstr ; and get next character
00AE 22 162    outst2: ret
163
164 ;*****
165 ; OUT2HX - OUTput 2 HeX characters to serial port
166 ; OUTHEX - OUTput 1 HEX character to serial port
167 ;
168 ; enter:  accumulator contains byte of data
169 ; exit:   nibbles converted to ASCII & sent out serial port;
170 ;          OUT2HX sends both nibbles; OUTHEX only sends lower
171 ;          nibble; all registers intact
172 ; uses:   outchr, htoa
173 ;
174 ;*****
00AF COEO 175 out2hx: push   acc      ;save data
00B1 C4 176    swap    a       ;send high nibble first
00B2 540F 177    anl     a,#0FH ;make sure upper nibble clear
00B4 120000 F 178    call    htoa   ;convert hex nibble to ASCII
00B7 120000 F 179    call    outchr ;send to serial port
00BC COEO 180    pop     acc
00BE 540F 181 outhex: push   acc      ;send low nibble
00C0 120000 F 182    anl     a,#0FH
00C3 120000 F 183    call    htoa
00C6 D0E0 184    call    outchr
00C8 22 185    pop     acc
186    ret
187
188 ;*****
189 ; PCHAR - Print CHARacter
190 ; send character to Centronics interface on 8155;
191 ; implements handshaking for -ACK, BUSY, and -STROBE
192 ;
193 ; enter:  ASCII code in accumulator
194 ; exit:   character written to 8155 Port A
195 ;
196 ;*****
0092 197 strobe equ    92h      ;8051 port 1 interface line for
0091 198 busy   equ    91h      ; printer interface
0090 199 ack    equ    90h
200
00C9 COEO 201 pchar: push   acc
00CB C083 202    push    dph
00CD C082 203    push    dpl
00CF 2091FD 204 wait: jb     busy,wait ;wait for printer ready, i.e.,
00D2 3090FA 205    jnb    ack,wait ; BUSY = 0 AND -ACK = 1
00D5 900000 F 206    mov    dptr,#riot + 1 ;8155 Port A address
00D8 C2E7 207    clr    acc.7   ;clear 8th bit, if set
00DA B40002 208    cjne   a,#cr,pchar2 ;substitute <lf> for <cr>
00DD 740A 209    mov    a,#lf
00DF F0 210 pchar2: movx   @dptr,a ;send data to printer
00E0 D292 211    setb   strobe ;toggle strobe bit
00E2 C292 212    clr    strobe
00E4 D292 213    setb   strobe
00E6 D082 214    pop    dpl
00E8 D083 215    pop    dph

```

00EA DOEO	216	pop	- acc
00EC 22	217	ret	
218			
219		*****	
220		; The following code, which executes upon a Timer 1 interrupt, *	
221		; implements a full duplex software UART. Timer 1 interrupts occur *	
222		; at a rate of 8 times the baud rate, or every 1 / (8 x 0.0012) = *	
223		; 104 microseconds. For transmit, a bit is output on P1.7 every 8 *	
224		; interrupts. For receive, a bit is read in P1.6 every 8 *	
225		; interrupts starting 12 interrupts after the start bit is *	
226		; detected. At 12 MHz, worse case execution time for this routine *	
227		; is 53 us. (This occurs for the last interrupt for simultaneous *	
228		; transmit and receive operations.) *	
229		*****	
230		serial_io_using_interrupts:	
00ED COEO	231	push	acc
00EF C0D0	232	push	psw
00F1 A200	F	233	mov c,r_idle ;if r_idle = 1 & p1.6 = 1, no RX
00F3 8296		234	ant c,RX_pin ; activity, therefore ...
00F5 4027		235	jc tx1 ; check for TX activity
00F7 30000A	F	236	jnb r_idle,rx1 ;else, if r_idle = 0 & p1.6 = x,
		237	; reception in progress, check it out
		238	; else, r_idle = 1 & p1.6 = 0, there-
		239	; fore, start bit detected
00FA C200	F	240	clr r_idle ;clear r_idle (begin reception)
00FC 75000C	F	241	mov r1_count,#12 ;11 interrupts to first data bit
00FF 750009	F	242	mov rb_count,#9 ;9 bits received (includes stop bit)
0102 801A		243	sjmp tx1
0104 D50017	F	244	rx1: djnz ri_count,tx1 ;decrement receive_interrupt_count
0107 750008	F	245	mov ri_count,#8 ;if 8th interrupt, reload and
010A E500	F	246	mov a,r_buff ; read next serial bit on RXD (P1.6)
010C A296		247	mov c,RX_pin
010E 13		248	rrc a
010F F500	F	249	mov r_buff,a
0111 D5000A	F	250	djnz rb_count,tx1 ;decrement receive_bit_count
0114 750009	F	251	mov rb_count,#9 ;if 9th bit received, done
0117 33		252	rlc a ;re-align bits (discard stop bit) and
0118 F500	F	253	mov r_buff,a ; save ASCII code in receive_buffer
011A D200	F	254	setb r_idle
011C D200	F	255	setb r_flag ;done! (now check for TX activity)
011E 20002C	F	256	tx1: jb t_flag,tx4 ;if t_flag = 1, nothing to transmit
0121 E500	F	257	mov a,tb_count ;check transmit_bit_count
0123 840809		258	cjne a,#11,tx2 ;if 11, first time here, therefore
0126 C297		259	clr TX_pin ; begin with start bit (p1.7 = 0)
0128 1500	F	260	dec tb_count ; decrement transmit_bit_count
012A 750008	F	261	mov ti_count,#8 ;8 interrupts for each data bit
012D 801E		262	sjmp tx4
012F D50018	F	263	tx2: djnz ti_count,tx4 ;if transmit_interrupt_count not 0
0132 750008	F	264	mov ti_count,#8 ; exit, otherwise reset count to 8 &
0135 E500	F	265	mov a,t_buff ; get next data bit to transmit
0137 03		266	setb c ;(ensures 9th bit = stop bit)
0138 13		267	rrc a ;right rotate (sends LSB first)
0139 9297		268	mov TX_pin,c ;put bit on p1.7 (TXD)
013B F500	F	269	mov t_buff,a ;save in t_buff for next rotate
013D E500	F	270	mov a,tb_count ;check for stop bit (stretch count)
013F 840203		271	cjne a,#2,tx3 ;if count = 1, stop bit just sent
0142 750010	F	272	mov ti_count,#16 ;stretch count to 16 (2 stop bits)
0145 D50005	F	273	tx3: djnz tb_count,tx4 ;if last bit,
0148 750008	F	274	mov tb_count,#11 ; reset count and
0148 D200	F	275	setb t_flag ; hoist flag (done!)
014D 0000		276	tx4: pop psw
014F D0E0		277	pop acc
0151 32		278	reti
		279	end

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

SUBRUTINAS DE CONVERSIÓN

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN CONVRT.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE CONVRT.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$title(** CONVERSION SUBROUTINES **)
		3	\$pagewidth(98)
		4	\$nopaging
		5	\$nosymbols
		6	\$nolist ;next line contains \$include(macros.srC)
		76	;previous line contains \$list
		77	*****
		78	;
		79	; CONVRT.SRC - CONVERSION SUBROUTINES
		80	;
		81	*****
		82	public atoh, htoa, touppr, tolowlr
		83	extrn code(isalph)
		84	
		85	eprom segment code
		86	rseg eeprom
		87	
		88	*****
		89	; ATOH - Ascii TO Hex conversion
		90	;
		91	; enter: ASCII code in accumulator (assume hex character)
		92	; exit: hex nibble in ACC.0 - ACC.3; ACC.4 - ACC.7 cleared
		93	;
		94	*****
0000 C2E7		95	atoh: clr acc.7 ;ensure parity bit is off
		96	; '0' to '9'?
0002 B43A00		97 +2	cjne a,#'9'+1,\$+3 ;JLE
0005 4002		98 +2	jc atoh2
0007 2409		99	add a,#9 ;no: adjust for range A-F
0009 540F		100	atoh2: anl a,#0FH ;yes: convert directly
000B 22		101	ret
		102	
		103	*****
		104	; HTOA - Hex TO Ascii conversion
		105	;
		106	; enter: hex nibble in ACC.0 to ACC.3
		107	; exit: ASCII code in accumulator
		108	;
		109	*****
000C 540F		110	htoa: anl a,#0FH ;ensure upper nibble clear
		111	; 'A' to 'F'?
000E B40A00		112 +2	cjne a,#0AH,\$+3 ;JLT
0011 4002		113 +2	jc htoe2
0013 2407		114	add a,#7 ;yes: add extra
0015 2430		115	htoe2: add a,#'0' ;no: convert directly
0017 22		116	ret
		117	

```

118 ;*****
119 ; TOUPPR - convert character TO UPPeRcase
120 ;
121 ;      enter: ASCII code in accumulator
122 ;      exit: converted to uppercase if alphabetic (left as is
123 ;            otherwise)
124 ;      uses: isalpH
125 ;
126 ;*****
0018 120000 F 127 touppr: call isalpH      ;is character alphabetic?
0018 5002        128 jnc skip          ;no: leave as is
001D C2E5        129 clr acc.5       ;yes: convert to uppercase by
001F 22          130 skip: ret         ;    clearing bit 5
131
132 ;*****
133 ; TOLoWR - convert character TO LOWeRcase
134 ;
135 ;      enter: ASCII code in accumulator
136 ;      exit: converter to lowercase if alphabetic (left as is
137 ;            otherwise)
138 ;      uses: isalpH
139 ;
140 ;*****
0020 120000 F 141 tolowlr: call isalpH     ;is character alphabetic?
0023 5002        142 jnc skip2         ;no: leave as is
0025 D2E5        143 setb acc.5       ;yes: convert to lowercase by setting
0027 22          144 skip2: ret        ;    bit 5
145 end

```

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

CARGA ARCHIVO HEXADECIMAL DE INTEL

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN LOAD.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE LOAD.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	Stitle(** LOAD INTEL HEX FILE ***)
		3	\$pagewidth(98)
		4	\$nopaging
		5	\$nosymbols
		6	*****
		7	;
		8	: LOAD.SRC - LOAD INTEL HEX FILE COMMAND
		9	;
		10	: FORMAT: LO - load and return to MON51
		11	; LG - load and go to user program in RAM
		12	;
		13	*****
		14	extrn code (inchar, outstr, atoh, getcmd,touppr)
		15	extrn data (linbuf,parmtr)
		16	extrn number (ram)
		17	public load
		18	
0007		19	bel equ 07H ;ASCII bell code
000D		20	cr equ 00H ;ASCII carriage code

```

000A          21    lf     equ    0AH      ;ASCII line feed code
001A          22    eof    equ    1AH      ;ASCII control-z (end-of-file)
0003          23    cancel equ    03H      ;ASCII control-c (cancel)
0004          24    eprom   segment code
0005          25    rseg    eprom
0006          26
0000 900000  F    27    load:   mov    dptr,#mess1
0003 1100    F    28    acall  outstr
0005 1100    F    29    load1:  acall  inchar
0007 841A02  30    cjne   a,#eof,skip1
000A 8031    31    sjmp   done
000C 840303  32    skip1:  cjne   a,#cancel,skip2
000F 020000  F    33    jmp    getcmd
0012 843AF0  34    skip2:  cjne   a,'#:',load1 ;each record begins with a ':'
0015 7900    35    mov    r1,#0 ;initialize checksum to 0
0017 1100    F    36    acall  gethex ;get byte count from serial port
0019 F5F0    37    mov    b,a ;use B as byte counter
001B 6020    38    jz    done ;if B = 0, done
001D 05F0    39    load2: inc    b ;no:
001F 1100    F    40    acall  gethex ;get address high byte
0021 F583    41    mov    dph,a
0023 1100    F    42    acall  gethex ;get address low byte
0025 F582    43    mov    dpl,a
0027 1100    F    44    acall  gethex ;get record type (ignore)
0029 1100    F    45    load4: acall  gethex ;get data byte
002B F0      46    movx   @dptra ;store in 8031 external memory
002C A3      47    inc    dptr
002D 15F0    48    dec    b ;repeat until count = 0
002F E5F0    49    mov    a,b
0031 70F6    50    jnz   load4
0033 E9      51    mov    a,r1 ;checksum should be zero
0034 60CF    52    jz    load1 ;if so, get next record
0036 900000  F    53    error: mov    dptr,#mess4 ;if not, error
0039 1100    F    54    acall  outstr
0038 800F    55    sjmp   wait
003D E500    F    56    done:  mov    a,linbuf+1 ;Load and Go command?
003F 1100    F    57    acall  touppr
0041 844703  58    cjne   a,'#'G',skip ;no: normal end to command
0044 020000  F    59    ljmp   ram ;yes: go to user program
0047 900000  F    60    skip:  mov    dptr,#mess3
004A 1100    F    61    acall  outstr
004C 1100    F    62    wait:  acall  inchar ;wait for eof before returning
004E 841AF8  63    cjne   a,#eof,wait
0051 020000  F    64    jmp    getcmd
0052
0053          65    ;*****
0054 1100    F    66    ; Get two characters from serial port and form a hex byte. Also *
0055          67    ; add byte to checksum in R1. *
0056 1100    F    68    ;*****
0057          69    ;*****
0058 C4      70    gethex: acall  inchar ;get first character
0059 F8      71    acall  atoh ;convert to hex
005A 1100    F    72    swap   a ;put in upper nibble
005B 1100    F    73    mov    r0,a ;save it
005C 1100    F    74    acall  inchar ;get second character
005D 1100    F    75    acall  atoh ;convert to hex
005E 48      76    orl    a,r0 ;OR with first nibble
005F FA      77    mov    r2,a ;save byte
0060 29      78    add    a,r1 ;add byte to checksum
0061 F9      79    mov    r1,a ;restore checksum in R1
0062 EA      80    mov    a,r2 ;retrieve byte
0063 22      81    ret
0064 486F7374 82    mess1: db    'Host download to SBC-51',cr
0068 20646F77
006C 6E6C6F61

```

```

0070 6420746F
0074 20534243
0078 2D3531
007B 0D
007C 5E5A203D      83      db      '^Z = end-of-file',cr,'^C = cancel',cr,0
0080 20656E64
0084 2D6F662D
0088 66696C65
008C 0D
008D 5E43203D
0091 2063616E
0095 63656C
0098 0D
0099 00
009A 656F6620      84      mess3: db      'eof - file downloaded OK',0
009E 2D206669
00A2 6C652064
00A6 6F776E6C
00AA 6F616465
00AE 64204F4B
00B2 00
00B3 07      85      mess4: db      bel,cr,'Error: ^Z Terminates',0
00B4 0D
00B5 4572726F
00B9 723A205E
00BD 5A205465
00C1 7260696E
00C5 61746573
00C9 00
          86      end

```

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

VACÍA LA MEMORIA A LA CONSOLA

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN DUMP.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE DUMP.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$title(*** DUMP MEMORY TO CONSOLE ***)
		3	\$pagewidth(98)
		4	\$nopaging
		5	\$nosymbols
		6	;*****
		7	;
		8	; DUMP MEMORY COMMAND
		9	;
		10	FORMAT: D<char><start>,<end>
		11	;
		12	D - dump memory command
		13	<char> - memory space selector as follows:
		14	I = internal data RAM
		15	X = external data RAM
		16	C = code memory
		17	B = bit address space

```

18      ;                                     R = SFRs
19      ;      <start> - starting address to dump
20      ;      <end>   - ending address to dump
21      ;
22      ;*****
23      extrn code (getcmd, outchr, htoe, out2hx, rsfr, isgrph) *
24      extrn code (getval, inchar, outhex) *
25      extrn data (parmr, linbuf) *
26      extrn bit (bit_sp, int_sp, reg_sp, x13_bit, r_flag) *
27      public dump, outdat, outadd
28
29      onchip segment data
30      eprom segment code
0000    31      cr      equ     0DH
0013    32      xoff    equ     13H
0020    33      space   equ     ' '
34
35      rseg    eprom
0000  850083 F 36      dump:  mov    dph,parmr      ;starting address to dump
0003  850082 F 37          mov    dpl,parmr+1
0006  750000 F 38      dump1: mov    ascbuf,#0      ;initialize ASCII string to NULL
0009  7900 F 39          mov    r1,#ascbuf      ;R1 = pointer into ASCII string
0008  200005 F 40          jb    x13_bit,dump1a ;check keyboard status, which is
000E  30000C F 41          jnb   r_flag,dump2  ; r_flag if x13 jumper installed, or
0011  8003 F 42          sjmp  dump1b
0013  309807 F 43      dump1a: jnb   ri,dump2      ; ri if x13 jumper not installed
0016  1100 F 44      dump1b: acall  inchar
0018  B41302 F 45          cjne  a,#xoff,dump2 ;if Control-S
0018  1100 F 46          acall  inchar      ; wait for next key
0010  1100 F 47      dump2:  acall  getval      ;read byte of data
001F  COE0 F 48          push   acc
0021  1100 F 49          acall  outadd
0023  D0E0 F 50      dump3:  pop    acc
0025  1100 F 51          acall  insert      ;add to ASCII buffer
0027  1100 F 52          acall  outdat      ;send to console
0029  4008 F 53          jc    dump1      ;if C, new line
0028  1100 F 54          acall  getval
002D  COE0 F 55          push   acc
002F  80F2 F 56          sjmp  dump3
57
58      ;*****
59      ; OUTADD - OUTPut ADDress to console
60      ;
61      ; enter: DPTR contains address; one memory space selector
62      ;           bit set
63      ; exit: appropriate size (8-bit or 16-bit) address sent to
64      ;           console; followed by " = " if "set" command active
65      ; uses:  out2hx, outchr
66      ;
67      ;*****
0031  200020 F 68      outadd: jb    bit_sp,outad5 ;if bit, register, or internal space
0034  20001D F 69          jb    reg_sp,outad5 ;only send 8-bit address
0037  20001A F 70          jb    int_sp,outad5
003A  E583 F 71          mov    a,dph
003C  1100 F 72          acall  out2hx
003E  E582 F 73      outad2: mov    a,dpl
0040  1100 F 74          acall  out2hx
0042  E500 F 75          mov    a,linbuf      ;get command character
0044  845308 F 76          cjne  a,'#$',outad3
0047  7420 F 77      outad4: mov    a,#space
0049  1100 F 78          acall  outchr
0048  7430 F 79          mov    a,'='        ;send '=' if "set" command
004D  1100 F 80          acall  outchr
004F  7420 F 81      outad3: mov    a,#space

```

```

0051 1100 F 82 acall outchr
0053 22 83 ret
0054 E583 84 outad5: mov a,dph ;address must be in range
0056 60E6 85 jz outad2 ; 00H to FFH for bit, register,
0058 020000 F 86 jmp getcmd ; or internal memory spaces
87
88 ****
89 ; OUTDAT - OUTput DATA to console
90 ;
91 ; enter: data in acc; one memory space selector bit set
92 ; exit: appropriate size value (1 bit or 8 bit) sent to
93 ; console; ASCII buffer flushed if dump command and
94 ; end of boundary reached
95 ; uses: outhex, outasc
96 ;
97 ****
0058 C4 98 outdat: swap a ;this gets tricky!
005C 200002 F 99 jb bit_sp,outdt2 ;if bit space,
005F 1100 F 100 acall outhex ;only send nibble
0061 C4 101 outdt2: swap a
0062 1100 F 102 acall outhex
0064 E500 F 103 mov a,linbuf ;if "set" command,
0066 6453 104 xrl a,#'S' ; don't check <end>
0068 6000 105 jz outad4 ; send ' = '
006A E583 106 mov a,dph ;DPTR = <end>?
006C 6500 F 107 xrl a,parmtr+2
006E 7011 108 jnz outdt3
0070 E582 109 mov a,dpl
0072 6500 F 110 xrl a,parmtr+3
0074 7008 111 jnz outdt3
0076 200005 F 112 jb reg_sp,outdt8 ;if bit or register space,
0079 200002 F 113 jb bit_sp,outdt8 ;don't send ASCII
007C 1100 F 114 acall outasc ;yes: flush buffer
007E 020000 F 115 outdt8: jmp getcmd ; & get next command
0081 A3 116 outdt3: inc dptr ;no: onwards
0082 200012 F 117 jb reg_sp,outdt7 ;if register space, one byte/line
0085 300006 F 118 jnb bit_sp,outdt6 ;if bit space, check for 8-bit
0088 E582 119 mov a,dpl ;boundary
008A 5407 120 anl a,#07H
008C 6009 121 jz outdt7
008E E582 122 outdt6: mov a,dpl ;if internal, external, or code
0090 540F 123 anl a,#0FH ; space, check for 16-byte boundary
0092 C3 124 cir c
0093 7007 125 jnz outdt4
0095 1100 F 126 acall outasc ;if there, flush ASCII buffer
0097 7400 127 outdt7: mov a,#cr
0099 1100 F 128 acall outchr
009B 03 129 setb c
009C 22 130 outdt4: ret
131
132 ****
133 ; INSERT - INSERT ASCII code into buffer
134 ;
135 ; enter: byte of data in accumulator
136 ; exit: ASCII code in buffer ('.' substituted for control
137 ; codes)
138 ; uses: isgrph
139 ;
140 ****
009D COE0 141 insert: push acc
009F 1100 F 142 acall isgrph ;is it a displayable character?
00A1 4002 143 jc insrt2 ;yes: leave as is
00A3 742E 144 mov a,'.'
00A5 F7 145 insrt2: mov @r1,a

```

```

00A6 09      146      inc    r1
00A7 7700    147      mov    @r1,#0          ;null character at end
00A9 00E0    148      pop    acc
00AB 22      149      ret
150
151      ****
152      ; OUTASC - OUTput ASCII codes to console *
153      ;
154      ;   enter: -
155      ;   exit:  buffer of ASCII graphic codes sent to console *
156      ;
157      ****
00AC 7420    158      outasc: mov   a,#space
00AE 1100    F       159      acall  outchr
00B0 7900    F       160      mov    r1,#ascbuf
00B2 E7      161      out3:  mov   a,@r1
00B3 7001    162      jnz   out2
00B5 22      163      ret
00B6 1100    F       164      out2: acall  outchr
00B8 09      165      inc    r1
00B9 80F7    166      sjmp  out3
167
168      ****
169      ; Create a buffer in internal RAM to hold ASCII codes to be dumped *
170      ; to console for DUMP command.
171      ;
172      rseg   onchip
0000      173      ascbuf: ds    17          ;ascii buffer
174      end

```

REGISTER BANK(S) USED: 0

LECTURA Y ESCRITURA DE SFRs

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN SFR.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE SFR.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$debug
		2	\$title (** READ AND WRITE SFRs ***)
		3	\$pagewidth(98)
		4	\$nopaging
		5	\$nosymbols
		6	*****
		7	;
		8	; SFR.SRC - READ AND WRITE SPECIAL FUNCTION REGISTERS *
		9	;
		10	*****
		11	
		12	public rsfr, wsfr
		13	
		14	eprom segment code
		15	rseg eprom
		16	
		17	*****
		18	;
		19	; RSFR - Read Special Function Register *
		20	;

```

21      :      enter: R0 contains address of SFR to read      *
22      :      exit:   accumulator contains value and C = 0      *
23      ;      if invalid SFR, C = 1      *
24      :      ****
25      ****
26
27      ; Let's get lazy and define a macro to do all the work.
28
29      rsfr:
0000 888003 30 +2      cjne    r0,#80h,SKIP00      ;p0
0003 E580 31 +2      mov     a,80h
0005 22 32 +1      ret
33 +2      SKIP00:
0006 888103 34 +2      cjne    r0,#81h,SKIP01      ;sp
0009 E581 35 +2      mov     a,81h
0008 22 36 +1      ret
37 +2      SKIP01:
000C 888203 38 +2      cjne    r0,#82h,SKIP02      ;dpl
000F E582 39 +2      mov     a,82h
0011 22 40 +1      ret
41 +2      SKIP02:
0012 888303 42 +2      cjne    r0,#83h,SKIP03      ;dph
0015 E583 43 +2      mov     a,83h
0017 22 44 +1      ret
45 +2      SKIP03:
0018 888803 46 +2      cjne    r0,#88h,SKIP04      ;tcon
001B E588 47 +2      mov     a,88h
001D 22 48 +1      ret
49 +2      SKIP04:
001E 888903 50 +2      cjne    r0,#89h,SKIP05      ;tmmod
0021 E589 51 +2      mov     a,89h
0023 22 52 +1      ret
53 +2      SKIP05:
0024 888A03 54 +2      cjne    r0,#8ah,SKIP06      ;tl0
0027 E58A 55 +2      mov     a,8ah
0029 22 56 +1      ret
57 +2      SKIP06:
002A 888803 58 +2      cjne    r0,#8bh,SKIP07      ;tl1
002D E588 59 +2      mov     a,8bh
002F 22 60 +1      ret
61 +2      SKIP07:
0030 888C03 62 +2      cjne    r0,#8ch,SKIP08      ;th0
0033 E58C 63 +2      mov     a,8ch
0035 22 64 +1      ret
65 +2      SKIP08:
0036 888003 66 +2      cjne    r0,#8dh,SKIP09      ;th1
0039 E580 67 +2      mov     a,8dh
0038 22 68 +1      ret
69 +2      SKIP09:
003C 889003 70 +2      cjne    r0,#90h,SKIP0A      ;p1
003F E590 71 +2      mov     a,90h
0041 22 72 +1      ret
73 +2      SKIP0A:
0042 889803 74 +2      cjne    r0,#98h,SKIP0B      ;scon
0045 E598 75 +2      mov     a,98h
0047 22 76 +1      ret
77 +2      SKIP0B:
0048 889903 78 +2      cjne    r0,#99h,SKIP0C      ;sbuf
0048 E599 79 +2      mov     a,99h
0040 22 80 +1      ret
81 +2      SKIP0C:
004E 88A003 82 +2      cjne    r0,#0a0h,SKIP00      ;p2
0051 E5A0 83 +2      mov     a,0a0h
0053 22 84 +1      ret

```

```

      85 +2 SKIP0D:
0054 88A803    86 +2   cjne r0,#0a8h,SKIP0E ;ie
0057 E5A8      87 +2   mov  a,0a8h
0059 22        88 +1   ret
      89 +2 SKIP0E:
005A 888003    90 +2   cjne r0,#0b0h,SKIP0F ;p3
005D E5B0      91 +2   mov  a,0b0h
005F 22        92 +1   ret
      93 +2 SKIP0F:
0060 888803    94 +2   cjne r0,#0b8h,SKIP10 ;ip
0063 E5B8      95 +2   mov  a,0b8h
0065 22        96 +1   ret
      97 +2 SKIP10:
0066 88E003    98 +2   cjne r0,#0d0h,SKIP11 ;psw
0069 E500      99 +2   mov  a,0d0h
006B 22        100 +1  ret
      101 +2 SKIP11:
006C 88E003   102 +2   cjne r0,#0e0h,SKIP12 ;acc
006F E5E0      103 +2   mov  a,0e0h
0071 22        104 +1  ret
      105 +2 SKIP12:
0072 88F003   106 +2   cjne r0,#0f0h,SKIP13 ;b
0075 E5F0      107 +2   mov  a,0f0h
0077 22        108 +1  ret
      109 +2 SKIP13:
0078 D3        110      setb c           ;C = 1 if invalid SFR
0079 22        111      ret
      112
113      ****
114      ;
115      ; WSFR - Write Special Function Register
116      ;
117      ; enter: R0 contains address of SFR
118      ; accumulator contains value to write
119      ; exit: value written to SFR and C = 0
120      ; if invalid SFR, C = 1
121      ;
122      ****
123
124      wsfr:
007A 888003   125 +2   cjne r0,#80h,SKIP14 ;p0
007D F580      126 +2   mov  80h,a
007F 22        127 +1  ret
      128 +2 SKIP14:
0080 888103   129 +2   cjne r0,#81h,SKIP15 ;sp
0083 F581      130 +2   mov  81h,a
0085 22        131 +1  ret
      132 +2 SKIP15:
0086 888203   133 +2   cjne r0,#82h,SKIP16 ;dpl
0089 F582      134 +2   mov  82h,a
0088 22        135 +1  ret
      136 +2 SKIP16:
008C 888303   137 +2   cjne r0,#83h,SKIP17 ;dph
008F F583      138 +2   mov  83h,a
0091 22        139 +1  ret
      140 +2 SKIP17:
0092 888803   141 +2   cjne r0,#88h,SKIP18 ;tcon
0095 F588      142 +2   mov  88h,a
0097 22        143 +1  ret
      144 +2 SKIP18:
0098 888903   145 +2   cjne r0,#89h,SKIP19 ;tmov
009B F589      146 +2   mov  89h,a
009D 22        147 +1  ret
      148 +2 SKIP19:

```

009E 888A03	149 +2	cjne	r0,#8ah,SKIP1A	;t10
00A1 F58A	150 +2	mov	8ah,a	
00A3 22	151 +1	ret		
	152 +2	SKIP1A:		
00A4 888B03	153 +2	cjne	r0,#8bh,SKIP1B	;t11
00A7 F58B	154 +2	mov	8bh,a	
00A9 22	155 +1	ret		
	156 +2	SKIP1B:		
00AA 888C03	157 +2	cjne	r0,#8ch,SKIP1C	;th0
00AD F58C	158 +2	mov	8ch,a	
00AF 22	159 +1	ret		
	160 +2	SKIP1C:		
00B0 888D03	161 +2	cjne	r0,#8dh,SKIP1D	;th1
00B3 F58D	162 +2	mov	8dh,a	
00B5 22	163 +1	ret		
	164 +2	SKIP1D:		
00B6 889003	165 +2	cjne	r0,#90h,SKIP1E	;p1
00B9 F590	166 +2	mov	90h,a	
00B8 22	167 +1	ret		
	168 +2	SKIP1E:		
00BC 889803	169 +2	cjne	r0,#98h,SKIP1F	;scon
00BF F598	170 +2	mov	98h,a	
00C1 22	171 +1	ret		
	172 +2	SKIP1F:		
00C2 889903	173 +2	cjne	r0,#99h,SKIP20	;sbuf
00C5 F599	174 +2	mov	99h,a	
00C7 22	175 +1	ret		
	176 +2	SKIP20:		
00C8 88A003	177 +2	cjne	r0,#0a0h,SKIP21	;p2
00CB F5A0	178 +2	mov	0a0h,a	
00CD 22	179 +1	ret		
	180 +2	SKIP21:		
00CE 88A803	181 +2	cjne	r0,#0a8h,SKIP22	;ie
00D1 F5A8	182 +2	mov	0a8h,a	
00D3 22	183 +1	ret		
	184 +2	SKIP22:		
00D4 88B003	185 +2	cjne	r0,#0b0h,SKIP23	;p3
00D7 F5B0	186 +2	mov	0b0h,a	
00D9 22	187 +1	ret		
	188 +2	SKIP23:		
00DA 88B803	189 +2	cjne	r0,#0b8h,SKIP24	;ip
00DD F5B8	190 +2	mov	0b8h,a	
00DF 22	191 +1	ret		
	192 +2	SKIP24:		
00E0 880003	193 +2	cjne	r0,#0d0h,SKIP25	;psw
00E3 F500	194 +2	mov	0d0h,a	
00E5 22	195 +1	ret		
	196 +2	SKIP25:		
00E6 88E003	197 +2	cjne	r0,#0e0h,SKIP26	;acc
00E9 F5E0	198 +2	mov	0e0h,a	
00EB 22	199 +1	ret		
	200 +2	SKIP26:		
00EC 88F003	201 +2	cjne	r0,#0f0h,SKIP27	;b
00EF F5F0	202 +2	mov	0f0h,a	
00F1 22	203 +1	ret		
	204 +2	SKIP27:		
00F2 03	205	setb	c	;if reached here, invalid
00F3 22	206	ret		;SFR, set carry flag
	207	end		

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

RUTINAS "ES"

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2

OBJECT MODULE PLACED IN IS.OBJ

ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE IS.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$title (*** IS ROUTINES ***)
		2	\$debug
		3	\$pagewidth (98)
		4	\$nopaging
		5	\$nosymbols
		6	\$nolist ;next line contains \$include(macros.src)
		76	;previous line contains \$list
		77	*****
		78	;
		79	; IS.SRC - IS SUBROUTINES
		80	;
		81	; These subroutines test a byte to see if it matches a certain
		82	; condition. If so, they return with the carry flag set; if not,
		83	; they return with the carry flag clear. These subroutines only
		84	; alter the carry flag; the accumulator is left intact.
		85	;
		86	*****
		87	;
		88	public isgrph, ishex, isdig, isalph
		89	eprom segment code
		90	rseg eprom
		91	;
		92	*****
		93	;
		94	; ISGRPH - IS the byte an ascii GRaphic code (i.e., in the range
		95	20H to 7EH)
		96	;
		97	enter: ASCII code in accumulator
		98	exit: C = 1 if code is ASCII graphic character
		99	C = 0 if code is ASCII control character
		100	;
		101	*****
0000	842000	102	isgrph: cjne a,#20h,\$+3 ;set C if < space
0003	100703	103	jbc cy,isgrp2 ;if set, clear and return
0006	847F00	104	cjne a,#7fh,\$+3 ;set C, if graphic
0009	22	105	isgrp2: ret
		106	;
		107	*****
		108	;
		109	; ISHEX - IS character ascii HEX?
		110	;
		111	enter: ASCII code in ACC
		112	exit: C = 1 if in range 0-9, a-f, or A-F
		113	C = 0 otherwise
		114	uses: isdig
		115	;
		116	*****
000A	COEO	117	ishex: push acc
000C	120000	F	118 call isdig
000F	4008	119 jc skip ;if digit, then ishex = true	
0011	D2E5	120 setb acc.5 ;convert to lowercase	
0013	846100	121 cjne a,'#a',,\$+3 ;C = 1 if < 'a'	
0016	100703	122 jbc cy,skip ;if 1, clear and return	
0019	846700	123 cjne a,'#f'+1,\$+3 ;carry set if hex	
001C	D0E0	124 skip: pop acc	
001E	22	125 ret	

```

126
127
128
129 ; ISDIG - IS ascii code a DIGIT?
130 ;
131 ;      enter: ASCII code in accumulator
132 ;      exit:  C = 1 if code is character in range 0-9
133 ;            C = 0 otherwise
134 ;
135 ;*****
001F 843000 136 isdig: cjne    a,#'0',$+3    ;carry set if < 0
0022 100703 137 jbc      cy.skip2   ;if set, clear and return
0025 843A00 138 cjne    a,#'9'+1,$+3  ;carry set if digit
0028 22     139 skip2: ret
140
141 ;*****
142 ;
143 ; ISALPH - IS ascii code an ALPHabetic character
144 ;
145 ;      enter: ASCII code in accumulator
146 ;      exit:  C = 1 if code is character in range a-z or A-Z
147 ;            C = 0 otherwise
148 ;
149 ;*****
0029 846100 150 +2 isalph: cjne    a,#'a',$+3    ;JIR
002C 4005 151 +2 jc       SKIP00
002E 847800 152 +2 cjne    a,#'z'+1,$+3
0031 400C 153 +2 jc       yes
154 +2 SKIP00:
0033 844100 155 +2 cjne    a,#'A',$+3    ;JIR
0036 4005 156 +2 jc       SKIP01
0038 845800 157 +2 cjne    a,#'Z'+1,$+3
0038 4002 158 +2 jc       yes
159 +2 SKIP01:
003D C3 160    clr      c           ;if reached here, can't be alpha
003E 22 161    ret
003F D3 162    yes:    setb    c           ;must be alphabetic character
0040 22 163    ret
164    end

```

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

MODIFICA LA MEMORIA CON UN VALOR

DOS 3.31 (038-N) MCS-51 MACRO ASSEMBLER, V2.2
OBJECT MODULE PLACED IN SET.OBJ
ASSEMBLER INVOKED BY: C:\ASM51\ASM51.EXE SET.SRC EP

LOC	OBJ	LINE	SOURCE
		1	\$title (*** SET MEMORY TO VALUE ***)
		2	\$debug
		3	\$pagewidth (98)
		4	\$nopaging
		5	\$nosymbols
		6	\$nolist ;next line contains \$include(macros.src)
		76	;previous line contains \$list

```

77 ;*****
78 ;*
79 ; SET.SRC - SET MEMORY TO VALUE
80 ;*
81 ; FORMAT: S<char><addr><cr>
82 ;*
83 ;      S      - set command
84 ;      <char> - memory space selector
85 ;                  B = bit address space
86 ;                  R = SFRs
87 ;                  I = internal data RAM
88 ;                  X = external data RAM
89 ;                  C = code ROM
90 ;      <addr> - address to set
91 ;      <cr>  - carriage return
92 ;*
93 ; FOLLOWED BY:   <cr>    - examine next
94 ;                  <value> - write data
95 ;                  -       - examine previous
96 ;                  Q      - quit
97 ;                  <sp>   - write same value into next
98 ;                  address
99 ;*
100 ;*****
101         extrn  code (rsfr, wsfr, outadd, getpar, getcmd)
102         extrn  code (inline, outhex, outdat, outstr)
103         extrn  bit (bit_sp, cde_sp, int_sp, reg_sp, ext_sp)
104         extrn  data (parmtr, linbuf)
105         public  setcmd, getval
106
0007      107     bel    equ    07H
000D      108     cr     equ    0DH
      ----
      109     eprom  segment code
      110     rseg   eprom
111
0000 850083 F 112     setcmd: mov    dph,parmtr      ;address to examine and/or set
0003 850082 F 113     mov    dpl,parmtr+1
0006 1100 F 114     set4:  acall  getval
0008 COE0 F 115     push   acc
000A 1100 F 116     acall  outadd
000C D0E0 F 117     pop    acc
000E 1100 F 118     acall  outdat
0010 7800 F 119     mov    r0,#linbuf+2    ;start address of data
0012 1100 F 120     acall  inline
0014 E500 F 121     mov    a,linbuf+2      ;check response character
0016 845103 F 122     cjne  a,'#0',set6    ;if 'Q', quit set command
0019 020000 F 123     ljmp   getcmd
001C B42D0F 124     set6:  cjne  a,'-',set7    ;if '--', decrement address pointer
001F COE0 125 +1    push   acc
0021 1582 126 +1    dec    dpl
0023 E582 127 +1    mov    a,dpl
0025 B4FF02 128 +2   cjne  a,#OFFH,SKIP00
0028 1583 129 +1    dec    dph
002A D0E0 130 +2   SKIP00: pop    acc
002C 8008 131     sjmp  set4
002E B40003 132     set7:  cjne  a,#0dh,set8    ;if <cr>, examine next location
0031 A3 133     inc    dptr
0032 8002 134     sjmp  set4
0034 1100 F 135     set8:  acall  getpar      ;otherwise a value has been entered
0036 E500 F 136     mov    a,parmtr+1    ;convert value and
0038 1100 F 137     acall  putval     ;put it into memory
003A A3 138     inc    dptr
003B 80C9 139     sjmp  set4
140

```

```

141      *****
142      : PUTVAL - PUT VALUE into memory
143
144      : enter: DPTR contains address; ACC contains value; one
145      :   memory space selector bit set as follows:
146      :     bit_sp = 1 for bit address space
147      :     cde_sp = 1 for code ROM
148      :     int_sp = 1 for internal data RAM
149      :     reg_sp = 1 for SFRs
150      :     ext_sp = 1 for external data RAM
151      : If bit address space selected then
152      :   R4 = byte address
153      :   R5 = read byte value
154      : (register bank 0 assumed)
155      : Bit will be inserted into read byte
156      :   value and byte value will be
157      :   written back to byte address
158      : exit: value written into memory
159
160      *****
161      putval: mov    r0,dpl      ;put address in R0 also
162          jnb    int_sp,put2
163          mov    @r0,a      ;internal data RAM
164          ret
165      put2: jnb    ext_sp,put3
166          movx   @dptr,a      ;external data RAM
167          ret
168      put3: jnb    cde_sp,put4
169          ret
170      put4: jnb    bit_sp,put5
171          anl    a,#1      ;make sure bits 1-7 = 0
172          anl    0,#7      ;reduce R0 to bit address
173
174          mov    r5,#0feh
175          inc    r0
176      put9: djnz   r0,put7
177          sjmp   put8
178      put7: rl     a      ;adjust bit position
179          xch    a,r5
180          rl     a      ;adjust mask
181          xch    a,r5
182          sjmp   put9
183      put8: xch    a,r4
184          anl    a,r5
185          orl    a,r4
186
187          mov    r0,3      ;recover byte address
188          cjne   r0,#80h,$+3
189          jnc    put10
190          mov    @r0,a      ;Done!! Whew!
191          ret
192      put5: jnb    reg_sp,put6
193      put10: acall   wsfr
194          jc     put6
195          ret
196      put6: mov    dptr,#put11
197          scall   outstr
198          ljmp    getcmd
199          db     bel,cr,'Error: no memory space selected',cr,0
007F 00
0080 4572726F
0084 723A206E
0088 6F206065
008C 606F7279
0090 20737061

```

```

0094 63652073
0098 656C6563
009C 746564
009F 0D
00A0 00
200
201 ;*****
202 ; GETVAL - GET VALue from memory
203 ;
204 ; enter: DPTR contains address; One memory space selector
205 ; bit set (see PUTVAL)
206 ; exit: Value in ACC, C = 0
207 ; If bit address space selected then
208 ; R3 = byte address
209 ; R4 = read byte value
210 ; (register bank 0 assumed)
211 ; If invalid SRF address, advance
212 ; DPTR until valid address found
213 ;
214 ;*****
00A1 A882 215 getval: mov r0,dpl ;put addres in R0 also
00A3 300002 F 216 jnb int_sp,get2
00A6 E6 217 mov a,@r0 ;internal data RAM
00A7 22 218 ret ;C = 0
00A8 300002 F 219 get2: jnb ext_sp,get3
00AB E0 220 movx a,@dptr ;external data RAM
00AC 22 221 ret ;C = 0
00AD 300003 F 222 get3: jnb cde_sp,get4
00B0 E4 223 clr a
00B1 93 224 movc a,@a+dptr ;code ROM
00B2 22 225 ret ;C = 0
00B3 300020 F 226 get4: jnb bit_sp,get5
227
228
229
00B6 5300F8 230 anl 0,#0f8h ;So you want to read a bit
00B9 B88000 231 cjne r0,#80H,$+3 ;value, do you?
00BC 400A 232 jc get6 ;This gets tricky!!
00BE AB00 233 mov r3,0 ;reduce R0 to byte address
00C0 1100 F 234 acall rsfr ;if >= 80H, read SFR
00C2 FC 235 mov r4,a ;first get byte value
00C3 500D 236 jnc get8 ;save byte address in R3
00C5 A3 237 inc dptr ;save byte value in R4
00C6 80D9 238 sjmp getval ;now extract bit
00C8 E8 239 get6: mov a,r0 ;if C = 1, try next SFR
00C9 03 240 rr a
00CA 03 241 rr a
00CB 03 242 rr a
00CC D2E5 243 setb acc.5 ;internal RAM, therefore
00CE F8 244 mov r0,a ;build byte address in acc
00CF FB 245 mov r3,a ;by translating as shown
00D0 E6 246 mov a,@r0
00D1 FC 247 mov r4,a
00D2 AF82 248 get8: mov r7,dpl ;eg: bit 35H is at byte address 26H
00D4 530707 249 anl 7,#7
00D7 0F 250 inc r7
00D8 DF02 251 get9: djnz r7,get10
00DA 8003 252 sjmp get11
00DC 03 253 get10: rr a
00DD 80F9 254 sjmp get9
00DF 5401 255 get11: anl a,#1 ;here's your bit
00E1 C3 256 clr c
00E2 22 257 ret

```

```

00E3 300009 F 258    get5: jnb    reg_sp,get7
00E6 1100 F 259    get13: acall   rsfr
00E8 5004          260    jnc    get12      ;if invalid SFR address,
00EA A3            261    inc    dptr       ;increment DPTR & try again
00EB 08            262    inc    r0
00EC 80F8          263    sjmp   get13
00EE 22            264    get12: ret
00EF 8085          265    get7: jmp    put6      ;Error: no memory space selected
00E6 266          end

REGISTER BANK(S) USED: 0

ASSEMBLY COMPLETE, NO ERRORS FOUND

```

MACROS.SRC

```

;*****
; JLT - Jump to "label" if accumulator Less Than "value" *
;*****
%define(jlt(value,label))
  (cjne  a,#%value,$+3           ;JLT
   jc    %label)

;*****
; JGT - Jump to "label" if accumulator Greater Than "value" *
;*****
%define(jgt(value,label))
  (cjne  a,#%value+1,$+3         ;JGT
   jnc   %label)

;*****
; JLE - Jump to "label" if accumulator Less than or Equal to "value"
;*****
%define(jle(value,label))
  (cjne  a,#%value+1,$+3         ;JLE
   jc    %label)

;*****
; JGE - Jump to "label" if accumulator Greater than or Equal to "value"
;*****
%define(jge(value,label))
  (cjne  a,#%value,$+3           ;JGE
   jnc   %label)

;*****
; JOR - Jump to "label" if accumulator Out of Range of "lower_value"
;       and "upper_value"
;*****
%define(jor(lower_value,upper_value,label))
  (cjne  a,#%lower_value,$+3     ;JOR
   jc    %label
  (cjne  a,#%upper_value+1,$+3
   jnc   %label)

```

```

;*****+
; JIR - Jump to "label" if accumulator In Range of "lower_value" and*
;       "upper_value" *
;*****+
/*define(jir(lower_value,upper_value,label)) local skip
    (cjne  a,#lower_value,$+3      ;JIR
     jc    %skip
     cjne  a,#upper_value+1,$+3
     jc    %label
%skip:  )

;*****+
; DECREMENT DPTR
;*****+
/*define(dec   dptra) local skip
    (push   acc          ;DEC_DPTR
    dec    dpl
    mov    a,dpl
    cjne  a,#0FFH,%skip
    dec    dph
%skip:  pop   acc)

;*****+
; PUSH DPTR ONTO STACK
;*****+
/*define(push   dptra)
    (push   dpl          ;PUSH_DPTR
    push   dph)

;*****+
; POP DPTR FROM STACK
;*****+
/*define(pop   dptra)
    (pop   dph          ;POP_DPTR
    pop   dpl)

;*****+
/*define (enable_register_bank(n))
    (%if (%n) then (setb   rs0) else (clr    rs0) fi
     %if (%n gt 1) then (setb   rs1) else (clr    rs1) fi)

;*****+
/*define (send_message(p))
    (push   dpl
    push   dph
    mov    dptra,#%p
    call   outstr
    pop   dph
    pop   dpl)

```

V12.M51

DATE : 04/21/91
DOS 3.31 (038-N) MCS-51 RELOCATOR AND LINKER V3.0, INVOKED BY:
C:\ASM51\RL51.EXE MAIN.OBJ,GETPAR.OBJ,IO.OBJ,CONVRT.OBJ,LOAD.OBJ,DUMP.OBJ,SFR.
>> OBJ,IS.OBJ,SET.OBJ TO V12 DATA(ONCHIP(30H))CODE(EPROM(0))

INPUT MODULES INCLUDED
MAIN.OBJ(MAIN)
GETPAR.OBJ(GETPAR)
IO.OBJ(IO)

CONVRT.OBJ(CONVRT)
 LOAD.OBJ(LOAD)
 DUMP.OBJ(DUMP)
 SFR.OBJ(SFR)
 IS.OBJ(IS)
 SET.OBJ(SET)

LINK MAP FOR V12(MAIN)

TYPE	BASE	LENGTH	RELOCATION	SEGMENT NAME
REG	0000H	0008H		"REG BANK 0"
DATA	0008H	0018H	ABSOLUTE	
BIT	0020H	0001H.5	UNIT	BITRAM
		0021H.5		*** GAP ***
DATA	0030H	0033H	UNIT	ONCHIP
CODE	0000H	0703H	UNIT	EPROM

Note: The following symbol table has been abbreviated for this printout. Only symbols declared as "public" are shown.

SYMBOL TABLE FOR V12(MAIN)

VALUE	TYPE	NAME
-----	-----	-----
-----	MODULE	MAIN
N:0014H	PUBLIC	BUFLEN
D:0044H	PUBLIC	EMDBUF
C:0121H	PUBLIC	ERROR
C:008CH	PUBLIC	GETCMD
D:0030H	PUBLIC	LINBUF
C:004CH	PUBLIC	NOTICE
B:0020H	PUBLIC	P_BIT
N:8000H	PUBLIC	RAM
X:0100H	PUBLIC	RIOT
C:2000H	PUBLIC	ROM
B:0020H.3	PUBLIC	X13_BIT
-----	ENDMOD	MAIN
-----	MODULE	GETPAR
B:0020H.5	PUBLIC	BIT_SP
B:0020H.6	PUBLIC	CDE_SP
B:0021H.1	PUBLIC	EXT_SP
C:0166H	PUBLIC	GETPAR
B:0020H.7	PUBLIC	INT_SP
D:0044H	PUBLIC	PARMTR
B:0021H	PUBLIC	REG_SP
-----	ENDMOD	GETPAR
-----	MODULE	IO
C:020EH	PUBLIC	INCHAR
C:022EH	PUBLIC	INLINE
C:028DH	PUBLIC	OUT2HX

C:01DEH	PUBLIC	OUTCHR
C:029AH	PUBLIC	OUTHEX
C:0282H	PUBLIC	OUTSTR
B:0021H.3	PUBLIC	R_FLAG
B:0021H.4	PUBLIC	R_IDLE
C:02CBH	PUBLIC	SERIAL_IO_USING_INTERRUPTS
B:0021H.2	PUBLIC	T_FLAG
D:0050H	PUBLIC	TB_COUNT
-----	ENDMOD	IO
-----	MODULE	CONVRT
C:0330H	PUBLIC	ATOH
C:033CH	PUBLIC	HTOA
C:0350H	PUBLIC	TOLOWR
C:0348H	PUBLIC	TOUPPR
-----	ENDMOD	CONVRT
-----	MODULE	LOAD
C:0358H	PUBLIC	LOAD
-----	ENDMOD	LOAD
-----	MODULE	DUMP
C:0422H	PUBLIC	DUMP
C:0453H	PUBLIC	OUTADD
C:047DH	PUBLIC	OUTDAT
-----	ENDMOD	DUMP
-----	MODULE	SFR
C:04DDH	PUBLIC	RSFR
C:0557H	PUBLIC	WSFR
-----	ENDMOD	SFR
-----	MODULE	IS
C:05FAH	PUBLIC	ISALPH
C:05FOH	PUBLIC	ISDIG
C:05D1H	PUBLIC	ISGRPH
C:05DBH	PUBLIC	ISHEX
-----	ENDMOD	IS
-----	MODULE	SET
C:0683H	PUBLIC	GETVAL
C:0612H	PUBLIC	SETCMD
-----	ENDMOD	SET

V12.HEX

:10 0000 00 02 00 78 30 02 34 02 80 03 00 00 30 02 2F 02 80	A8
:10 0010 00 06 00 00 30 02 2A 02 80 09 00 00 30 02 25 02 00	9A
:10 0020 00 31 00 00 30 02 20 02 80 0F 00 00 30 02 1B 02 80	ED
:10 0030 00 12 20 03 03 02 02 CB 02 80 0C 02 20 03 02 20 06	DE
:10 0040 00 02 20 09 02 20 0C 02 20 0F 02 20 12 43 6F 70 79	57
:10 0050 00 72 69 67 68 74 20 28 63 29 20 49 2E 20 53 63 6F	D2
:10 0060 00 74 74 20 4D 61 63 48 65 6E 7A 69 65 2C 20 31 39	58
:10 0070 00 38 38 2C 20 31 39 39 31 A2 94 92 01 A2 93 92 02	5E
:10 0080 00 A2 95 92 03 20 01 03 02 20 00 20 03 17 C2 0B D2	85
:10 0090 00 0C D2 0A 75 50 08 75 89 20 75 80 98 D2 8E D2 AF	0F
:10 00A0 00 D2 AB 80 08 75 98 52 75 8D F3 75 89 20 D2 8E C2	B4
:10 00B0 00 00 90 01 00 74 01 F0 90 01 30 51 82 75 81 07 90	1C
:10 00C0 00 01 44 51 82 78 30 51 2E E5 30 84 00 03 02 06 12	FE
:10 00D0 00 84 41 00 40 4C B4 5B 00 50 47 54 1F 14 23 F8 04	53
:10 00E0 00 90 00 ED 93 C0 E0 EB 93 C0 E0 31 66 22 01 21 01	69

:10 00F0 00 21 01 21 04 22 01 21 01 21 01 30 01 21 01 21 01 00
:10 0100 00 21 01 21 03 58 01 21 01 21 01 21 01 21 01 21 01 21 01 A6
:10 0110 00 21 06 12 01 21 01 21 01 21 01 21 01 21 01 21 01 21 01 D9
:10 0120 00 21 90 01 4A 51 82 E5 30 31 DE E5 31 31 DE 80 8C AB
:10 0130 00 74 BC C0 E0 74 00 C0 E0 C0 45 C0 44 22 00 4D 4F 07
:10 0140 00 4E 35 31 00 00 56 31 32 3E 00 07 00 45 72 72 6F 4B
:10 0150 00 72 3A 20 49 6E 76 61 6C 69 64 20 43 6F 60 60 61 FF
:10 0160 00 6E 64 20 2D 20 00 78 32 79 44 7F 04 86 2C 02 80 02
:10 0170 00 08 31 81 30 04 08 86 2C 05 08 09 09 DF EE 31 88 D2
:10 0180 00 22 C2 04 E6 81 DB 50 13 02 04 E4 F7 09 F7 19 E6 02
:10 0190 00 81 DB 50 07 71 30 31 9C 08 80 F4 22 C0 07 C0 00 E9
:10 01A0 00 A8 01 08 7F 04 C4 33 C0 E0 E6 33 F6 E7 33 F7 D0 94
:10 01B0 00 E0 DF F3 00 00 D0 07 22 C2 05 C2 06 C2 07 C2 08 A2
:10 01C0 00 C2 09 E5 31 B4 42 02 D2 05 B4 43 02 D2 06 B4 49 B1
:10 01D0 00 02 D2 07 B4 52 02 D2 08 B4 58 02 D2 09 22 C0 E0 87
:10 01E0 00 A2 D0 B3 92 E7 20 03 00 30 0A FD C2 AB C2 0A F5 DC
:10 01F0 00 4C D2 AB 80 07 30 99 FD C2 99 F5 99 C2 E7 B4 00 96
:10 0200 00 04 74 DA 80 DB D0 E0 30 00 03 12 02 A7 22 20 03 2E
:10 0210 00 00 30 0B FD C2 AB C2 0B E5 4D 02 A8 80 07 30 98 61
:10 0220 00 FD C2 98 E5 99 C2 E7 B4 03 03 02 00 BC 22 12 02 A2
:10 0230 00 0E 12 03 48 B4 10 04 82 00 80 F3 12 01 DE F6 08 77
:10 0240 00 B8 44 08 90 02 67 51 82 02 00 BC B4 04 02 80 03 E3
:10 0250 00 B4 7F 0A 18 18 90 02 63 12 02 82 80 D1 B4 00 CE C6
:10 0260 00 76 00 22 18 58 44 00 07 00 45 72 72 6F 72 3A 20 C4
:10 0270 00 43 6F 60 60 61 6E 64 20 74 6F 6F 20 6C 6F 6E 67 7D
:10 0280 00 00 00 E4 93 60 06 12 01 DE A3 80 F6 22 C0 E0 C4 F4
:10 0290 00 54 0F 12 03 3C 12 01 DE D0 E0 22 C0 E0 C0 83 C0 82 20 91 FD 7C
:10 02A0 00 3C 12 01 DE D0 E0 22 C0 E0 C0 83 C0 82 20 91 FD 7C
:10 02B0 00 30 90 FA 90 01 01 C2 E7 B4 00 02 74 0A F0 D2 92 B4
:10 02C0 00 C2 92 02 92 00 82 D0 83 D0 E0 22 C0 E0 C0 D0 A2 2D
:10 02D0 00 0C 82 96 40 27 30 0C 0A C2 0C 75 4F 0C 75 51 09 E0
:10 02E0 00 80 1A 05 4F 17 75 4F 08 E5 4D A2 96 13 F5 4D D5 D9
:10 02F0 00 51 0A 75 51 09 33 F5 4D 02 D0 02 08 20 0A 2C E5 69
:10 0300 00 50 84 08 09 C2 97 15 50 75 4E 08 80 1E D5 4E 18 70
:10 0310 00 75 4E 08 E5 4C D3 13 92 97 F5 4C E5 50 84 02 03 A3
:10 0320 00 75 4E 10 D5 50 05 75 50 08 D2 0A D0 D0 D0 E0 32 A2
:10 0330 00 C2 E7 B4 3A 00 40 02 24 09 54 0F 22 54 0F B4 0A 11
:10 0340 00 00 40 02 24 07 24 30 22 12 05 FA 50 02 C2 E5 22 9E
:10 0350 00 12 05 FA 50 02 02 E5 22 90 03 BC 51 82 51 0E B4 2C
:10 0360 00 1A 02 80 31 B4 03 03 02 00 BC B4 3A F0 79 00 71 80
:10 0370 00 AC F5 F0 60 20 05 F0 71 AC F5 83 71 AC F5 82 71 DD
:10 0380 00 AC 71 AC F0 A3 15 F0 E5 F0 70 F6 E9 60 CF 90 04 25
:10 0390 00 08 51 82 80 0F E5 31 71 48 B4 47 03 02 80 00 90 11
:10 03A0 00 03 F2 51 82 51 0E B4 1A FB 02 00 BC 51 0E 71 30 9F
:10 03B0 00 C4 F8 51 0E 71 30 48 FA 29 F9 EA 22 48 6F 73 74 73
:10 03C0 00 20 64 6F 77 6E 6C 6F 61 64 20 74 6F 20 53 42 43 BA
:10 03D0 00 2D 35 31 00 5E 5A 20 30 20 65 6E 64 2D 6F 66 2D E2
:10 03E0 00 66 69 6C 65 00 5E 43 20 30 20 63 61 6E 63 65 6C DC
:10 03F0 00 00 00 65 6F 66 20 2D 20 66 69 6C 65 20 64 6F 77 3F
:10 0400 00 6E 6C 6F 61 64 65 64 20 4F 48 00 07 00 45 72 72 1E
:10 0410 00 6F 72 3A 20 5E 5A 20 54 65 72 6D 69 6E 61 74 65 20
:10 0420 00 73 00 85 44 83 85 45 82 75 52 00 79 52 20 03 05 07
:10 0430 00 30 08 0C 80 03 30 98 07 51 0E B4 13 02 51 0E D1 CB
:10 0440 00 B3 C0 E0 91 53 00 E0 91 8F 91 7D 40 DB D1 B3 C0 08
:10 0450 00 E0 80 F2 20 05 20 20 08 1D 20 07 1A E5 83 51 8D 39
:10 0460 00 E5 82 51 8D E5 30 84 53 08 74 20 31 DE 74 3D 31 9E
:10 0470 00 DE 74 20 31 DE 22 E5 83 60 E6 02 00 8C C4 20 05 84
:10 0480 00 02 51 9A C4 51 9A E5 30 64 53 60 00 E5 83 65 46 84
:10 0490 00 70 11 E5 82 65 47 70 08 20 08 05 20 05 02 91 CE 9A
:10 04A0 00 02 00 8C A3 20 08 12 30 05 06 E5 82 54 07 60 09 4B

```

:10 0480 00 E5 82 54 0F C3 70 07 91 CE 74 0D 31 DE D3 22 C0 94
:10 04C0 00 E0 81 D1 40 02 74 2E F7 09 77 00 00 E0 22 74 20 09
:10 0400 00 31 DE 79 52 E7 70 01 22 31 DE 09 80 F7 88 80 03 FE
:10 04E0 00 E5 80 22 88 81 03 E5 81 22 88 82 03 E5 82 22 88 43
:10 04F0 00 83 03 E5 83 22 88 88 03 E5 88 22 88 89 03 E5 89 68
:10 0500 00 22 88 8A 03 E5 8A 22 88 88 03 E5 88 22 88 8C 03 D4
:10 0510 00 E5 8C 22 88 8D 03 E5 8D 22 88 90 03 E5 90 22 88 D2
:10 0520 00 98 03 E5 98 22 88 99 03 E5 99 22 88 A0 03 E5 A0 8D
:10 0530 00 22 88 8A 03 E5 A8 22 88 80 03 E5 80 22 88 88 03 F2
:10 0540 00 E5 88 22 88 00 03 E5 00 22 88 E0 03 E5 E0 22 88 50
:10 0550 00 F0 03 E5 F0 22 D3 22 88 80 03 F5 80 22 88 81 03 AE
:10 0560 00 F5 81 22 88 82 03 F5 82 22 88 83 03 F5 83 22 88 8D
:10 0570 00 88 03 F5 88 22 88 89 03 F5 89 22 88 8A 03 F5 8A A9
:10 0580 00 22 88 88 03 F5 88 22 88 8C 03 F5 8C 22 88 8D 03 2F
:10 0590 00 F5 8D 22 88 90 03 F5 90 22 88 98 03 F5 98 22 88 0B
:10 05A0 00 99 03 F5 99 22 88 A0 03 F5 A0 22 88 A8 03 F5 A8 ED
:10 0580 00 22 88 80 03 F5 80 22 88 88 03 F5 88 22 88 00 03 1A
:10 05C0 00 F5 D0 22 88 E0 03 F5 E0 22 88 F0 03 F5 F0 22 D3 2D
:10 05D0 00 22 84 20 00 10 07 03 84 7F 00 22 C0 E0 12 05 F0 3F
:10 05E0 00 40 08 D2 E5 84 61 00 10 07 03 84 67 00 00 E0 22 1D
:10 05F0 00 84 30 00 10 07 03 84 3A 00 22 84 61 00 40 05 84 0F
:10 0600 00 7B 00 40 0C 84 41 00 40 05 84 58 00 40 02 C3 22 83
:10 0610 00 03 22 85 44 83 85 45 82 D1 83 C0 E0 91 53 00 E0 95
:10 0620 00 91 7D 78 32 51 2E E5 32 84 51 03 02 00 8C 84 2D 05
:10 0630 00 0F C0 E0 15 82 E5 82 84 FF 02 15 83 D0 E0 80 D8 88
:10 0640 00 B4 00 03 A3 80 D2 31 66 E5 45 01 4F A3 80 C9 A8 7C
:10 0650 00 82 30 07 02 F6 22 30 09 02 F0 22 30 06 01 22 30 F1
:10 0660 00 05 1E 54 01 53 00 07 7D FE 08 D8 02 80 06 23 CD E5
:10 0670 00 23 CD 80 F6 CC 5D 4C A8 03 88 80 00 50 05 F6 22 4F
:10 0680 00 30 08 05 81 57 40 01 22 90 06 90 51 82 02 00 BC 0B
:10 0690 00 07 00 45 72 72 6F 72 3A 20 6E 6F 20 60 65 6D 6F 37
:10 06A0 00 72 79 20 73 70 61 63 65 20 73 65 6C 65 63 74 65 2E
:10 06B0 00 64 00 00 A8 82 30 07 02 E6 22 30 09 02 E0 22 30 F1
:10 06C0 00 06 03 E4 93 22 30 05 20 53 00 F8 88 80 00 40 0A 59
:10 06D0 00 AB 00 91 DD FC 50 00 A3 80 D9 E8 03 03 03 D2 E5 04
:10 06E0 00 F8 FB E6 FC AF 82 53 07 07 0F DF 02 80 03 03 80 AD
:10 06F0 00 F9 54 01 C3 22 30 08 09 91 DD 50 04 A3 08 80 F8 A1
:03 0700 00 22 80 85 CF
:00 0000 01 FF

```



Una guía al IDE μ VISION2 de Keil

INTRODUCCIÓN

El entorno de desarrollo integrado (IDE)¹ μ Vision2 de Keil es un software que permite al programador en C del 8051 editar, compilar, ejecutar y depurar un programa escrito en C del 8051. Este apéndice presenta una guía breve sobre cómo utilizar el IDE μ Vision2.

EL ESPACIO DE TRABAJO DE μ VISION2

Al hacer doble clic en el IDE μ Vision2 veremos una pantalla similar a la que se muestra en la figura H-1. Las ventanas principales que aparecen son el espacio de trabajo, la ventana de edición, y la ventana de salida.

El espacio de trabajo es un panel con tres tabuladores que abren diferentes ventanas —archivos (“Files”), registros (“Regs”), y libros (“Books”). La ventana de archivos (vea la figura H-2) nos permite administrar los archivos de código fuente que deseamos incluir en nuestro proyecto en curso. Haga un clic con el botón derecho del ratón en la opción de simulación (“Simulator”), y después seleccione la opción para seleccionar el dispositivo para el destino del ‘simulador’ (“Select Device for Target ‘Simulator’”), para indicar el 8051 de destino o un dispositivo derivado para el cual usted desea escribir un programa en C. Para agregar un archivo a su proyecto, simplemente haga clic con el botón derecho del ratón en la opción del grupo fuente 1 (“Source Group 1”) y busque el archivo en C deseado.

Si prefiere crear un nuevo programa, entonces seleccione la opción “New” del menú de archivos (“File”) y empiece a capturar el código para su programa. Cuando haya terminado, almacénelo como un archivo C y añádalo a su proyecto. En la figura H-2 podemos ver que el único archivo incluido en el proyecto es test.c. Haga doble clic en un nombre de archivo para abrirlo y se mostrará en la ventana de edición, tal como ilustra la figura H-1.

COMPILACIÓN Y DEPURACIÓN

Cuando haya terminado de editar su programa, el siguiente paso será compilarlo para ver si existen errores de compilación. Para hacer esto, puede seleccionar la opción para compilar un destino

¹Una versión de evaluación está disponible para descarga en <http://www.keil.com/demo/>.

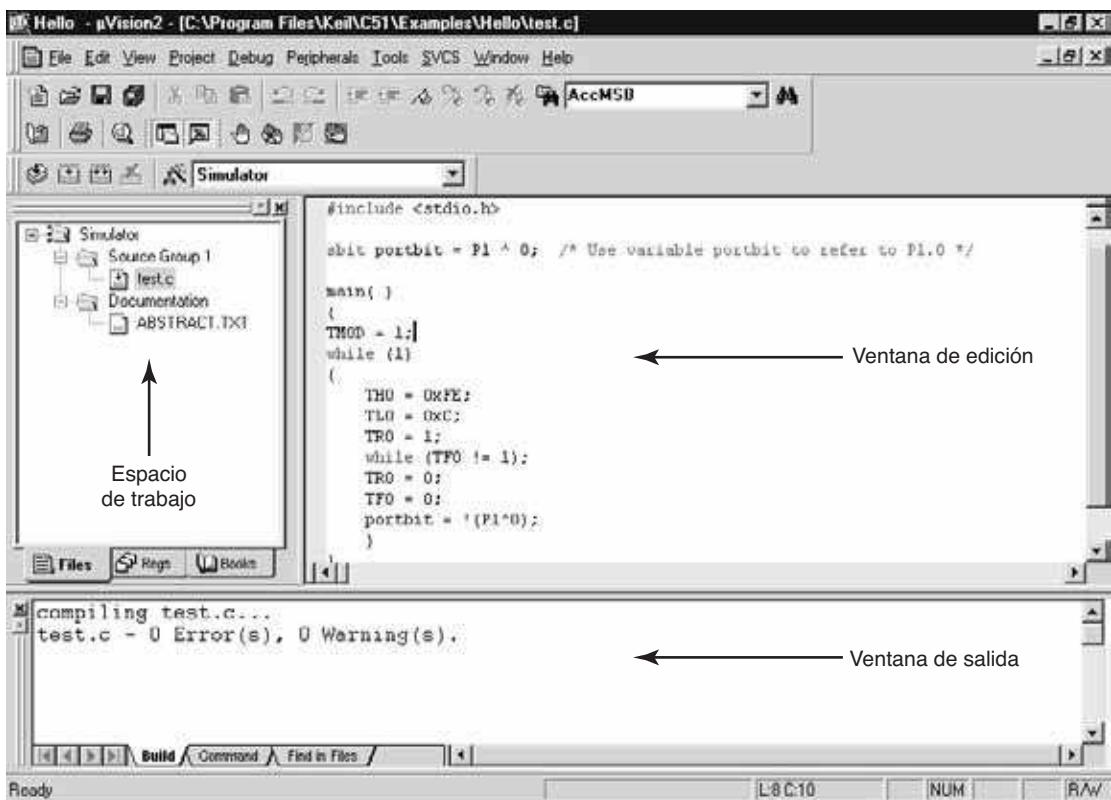
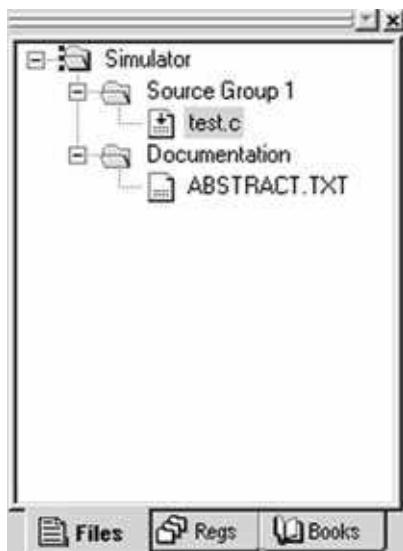


FIGURA H-1
IDE μ Vision2

FIGURA H-2
La ventana de archivos



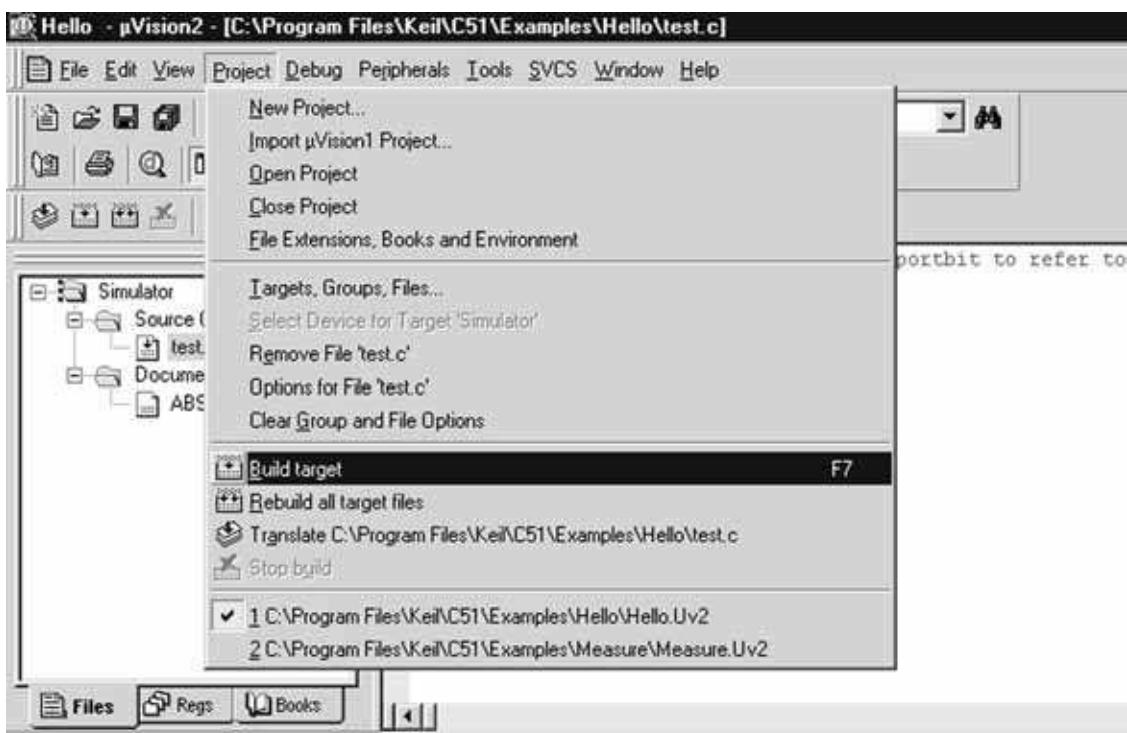


FIGURA H-3

Seleccione la opción "Build target" del menú de proyectos ("Project") para compilar su programa.

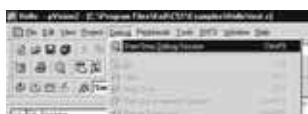
(“Build target”) del menú de proyectos (“Project”, vea la figura H-3). Esto compilará todos los archivos recientemente actualizados en su proyecto. También puede seleccionar la opción de traducción/conversión (“Translate...”) para compilar sólo el archivo C actualmente resaltado, o la opción para recompilar todos los archivos de destino (“Rebuild all target files”) para compilar todos los archivos en su proyecto independientemente de si los acaba de actualizar o no. El tabulador de salida deberá mostrar 0 errores.

Le recomendamos probar y depurar el programa compilado antes de descargarlo a la memoria ROM de su 8051, con ello podrá determinar si se ejecuta en la forma que usted lo desea. Seleccione la opción para iniciar/terminar una sesión de depuración (“Start/Stop Debug Session”) del menú de depuración (“Debug”, vea la figura H-4). Si quiere ejecutar el programa de principio a fin, seleccione la opción de ejecución (“Go”) del menú de depuración (“Debug”). En caso contrario, si quiere ejecutar una instrucción en C a la vez, seleccione la opción para ejecución por pasos (“Step”).

Conforme ejecutamos nuestro programa, ya sea paso a paso o en su totalidad, en la sesión de depuración es posible seleccionar una opción para examinar el estado de los registros, seleccionando el tabulador “Regs” para abrir la ventana de registros “Regs”, (vea la figura H-5).

FIGURA H-4

Seleccione la opción “Start/Stop Debug Session” del menú de depuración (“Debug”) para iniciar/terminar una sesión de depuración.



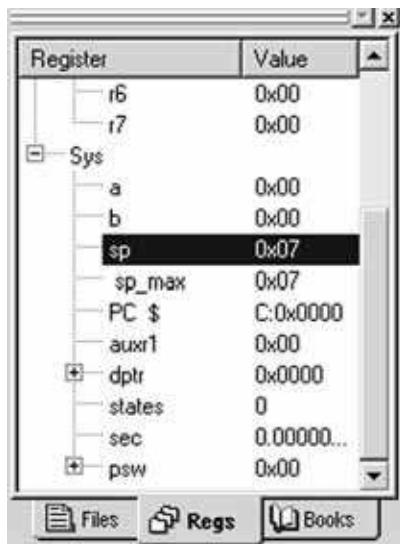


FIGURA H-5
Ventana de registros (“Regs”)

La figura H-5 muestra la ventana de registros (“Regs”) con el apuntador de pila (SP) y su contenido resaltados. Podemos examinar también el contenido de la memoria al seleccionar la ventana de memoria (“Memory Window”) del menú “View”. La ventana de memoria (vea la figura H-6) aparece normalmente en la esquina inferior derecha de la pantalla. La figura H-6 muestra el contenido de las ubicaciones en memoria externa empezando desde 0103H. Si queremos ver el contenido de la memoria interna para datos, hacemos clic en el control de edición de texto localizado a la derecha de “Address:” (dirección) y escribimos D:0000, aquí D se refiere a la memoria interna para datos y 0000 especifica la ubicación inicial que queremos examinar. Si escribimos C en lugar de D, nos permite examinar la memoria para código. Para modificar cierta ubicación en memoria basta con hacer doble clic en la ubicación y seleccionar la opción para modificar la memoria en una dirección (“Modify Memory at ...”, vea la figura H-7) y escribir el nuevo valor en el control de edición que aparece. Puede escribir los valores en decimal o en hexadecimal utilizando la notación estándar en C.

También es fácil examinar el contenido y el estado de los periféricos incorporados en el chip si los seleccionamos en el menú de periféricos (“Peripherals”). La figura H-8 muestra un ejemplo de cómo examinar el contenido del puerto 0.

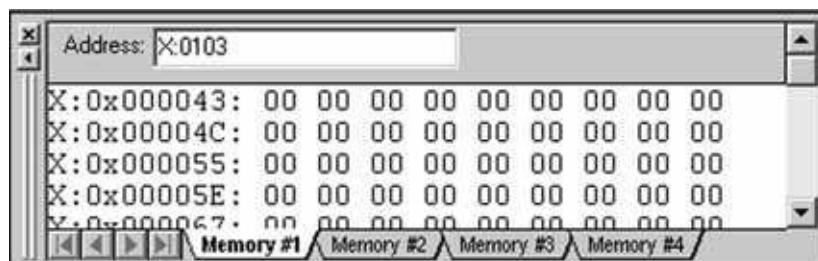
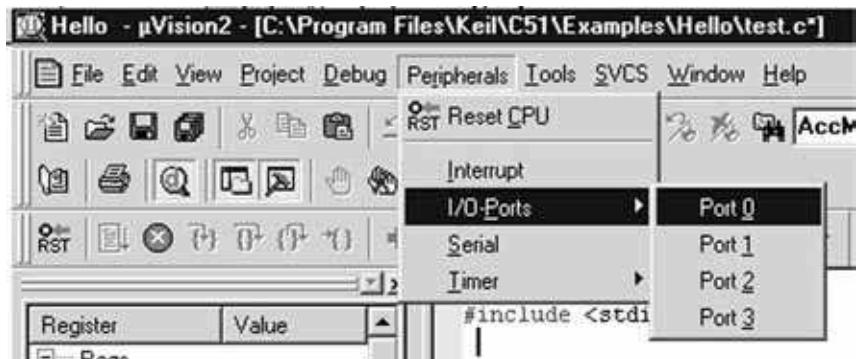


FIGURA H-6
Ventana de memoria

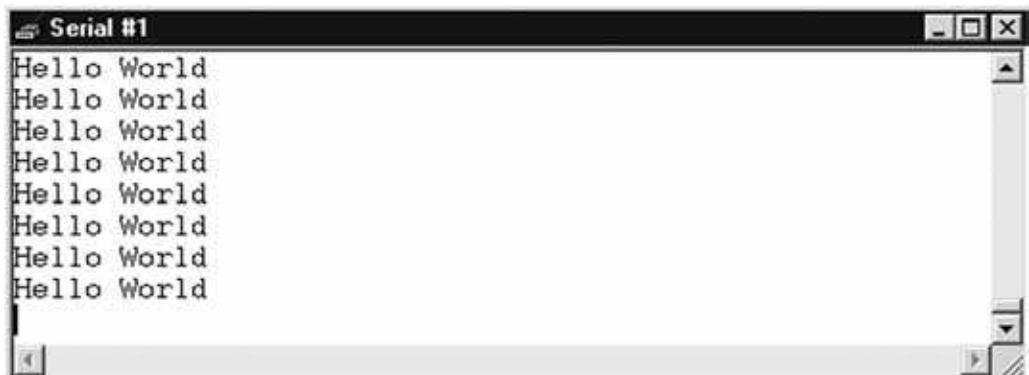
FIGURA H-7

Modificación de una ubicación en memoria

**FIGURA H-8**

Examen del contenido y estado de un periférico

Recuerde que en el primer programa en C, “Hello World” visto en el capítulo 8, mencionamos que en el IDE μ Vision2, el dispositivo conectado al puerto serial del 8051 es la simulación de una ventana serial de manera predeterminada. Si utilizamos esta ventana serial, podremos examinar cualquier mensaje que envíemos con la instrucción `printf` al puerto serial. Para tener acceso a esta ventana serial, seleccione la opción “Serial Window #1” del menú “View” durante la sesión de depuración. La figura H-9 muestra la ventana serial cuando se ejecuta el programa “Hello world” en el capítulo 8.

**FIGURA H-9**

Ventana serial

En la sección sobre los temporizadores vista en el capítulo 8, aprendimos que podemos dirigir al compilador de C del μ Vision2 para desensamblar nuestros programas en C al lenguaje ensamblador. Para examinar la versión en lenguaje ensamblador correspondiente a nuestro programa, seleccione la ventana de desensamblado (“Disassembly Window”) del menú “View”. La figura H-10 muestra el desensamblado de nuestro programa “Hello world” con las instrucciones en C originales justo arriba de las instrucciones en lenguaje ensamblador correspondientes.



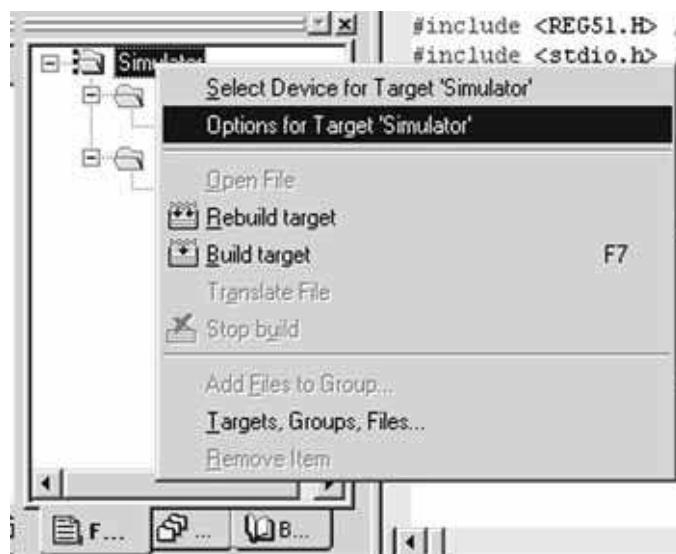
```

Disassembly
4: main ()
5: {
6: SCON = 0x52; /* serial port, mode 1 */
C:0x0C11 759852 MOV SCON(0x98),#0x52
7: TMOD = 0x20; /* timer 1, mode 2 */
C:0x0C14 758920 MOV TMOD(0x89),#0x20
8: TH1 = -13; /* reload count for 2400 bps */
C:0x0C17 758DF3 MOV TH1(0x8D),#0xF3
9: TR1 = 1; /* start timer 1 */
10:
C:0x0C1A D28E SETB TR1(0x88.6)
11: while (1) /* repeat forever */
12: {
13: printf ("Hello World\n"); /* Display "Hello World" */

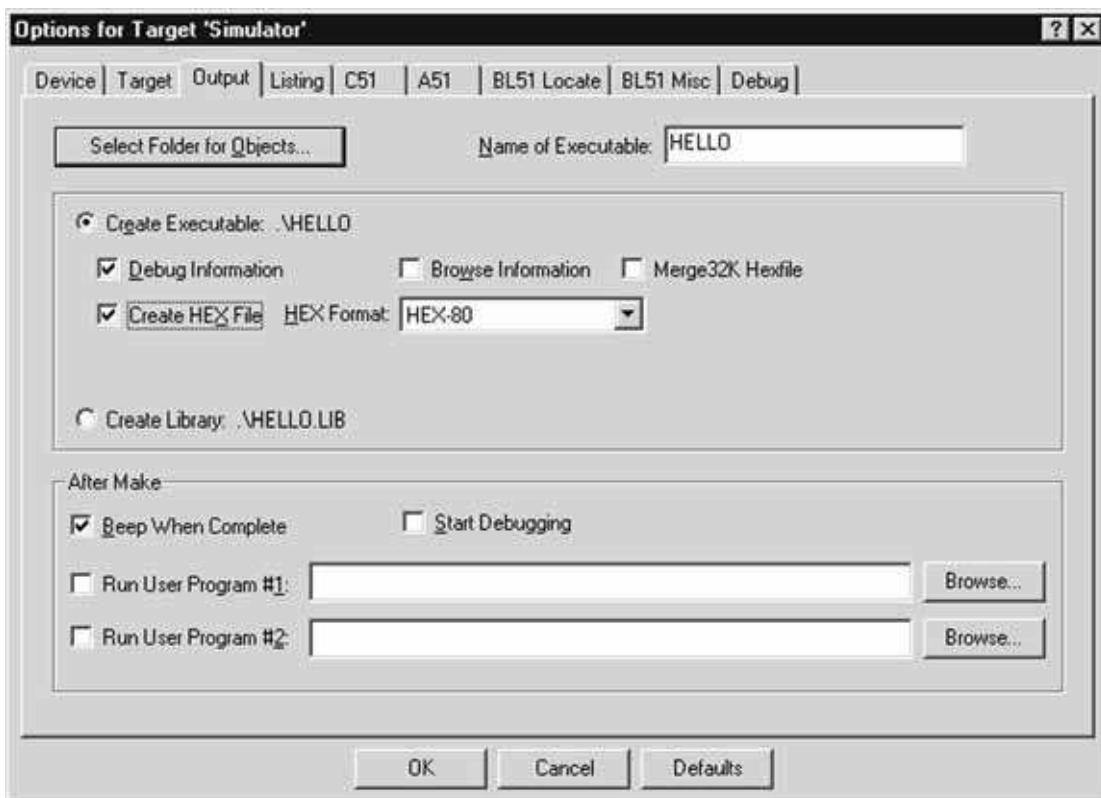
```

FIGURA H-10

Ventana del desensamblador

**FIGURA H-11**

Opciones para el destino

**FIGURA H-12**

Tabulador de opciones de salida

Una vez que estemos satisfechos con la depuración y la simulación de nuestro programa, podemos dirigir al μ Vision2 a que genere el código máquina como un archivo HEX. Haga clic en el botón derecho en la opción de simulador (“Simulator”) de la ventana de archivos (“File”), y seleccione las opciones para el destino (“Options for Target...”, vea la figura H-11). Haga clic en el tabulador de salida y habilite la opción para crear un archivo HEX (“Create HEX File”, vea la figura H-12).

Cuando esta opción está seleccionada, la siguiente vez que compilemos el destino se creará un archivo HEX en el mismo directorio donde está nuestro archivo en C. Entonces podremos descargar este archivo HEX a la memoria ROM, y el 8051 estará listo para ejecutar el programa al momento de encenderlo.



Una guía para el simulador del 8052¹

INTRODUCCIÓN

El simulador del 8052 (“8052 Simulator”) para Windows es un software que simula la operación del microcontrolador 8051 u 8052 en su totalidad, además de todos sus registros, ubicaciones en RAM interna, puertos de E/S, temporizadores, puertos seriales e interrupciones. Durante las etapas iniciales de desarrollo de un sistema basado en el 8051 o en el 8052, podemos verificar el funcionamiento del programa ensamblado o compilado completamente en software con la ayuda de este simulador, sin tener que preocuparnos sobre el hardware verdadero. Este apéndice proporciona una breve guía acerca de cómo utilizar el simulador del 8052.

El simulador del 8052

La figura I-1 muestra la ventana principal del simulador del 8052. Lo primero que tenemos que hacer es cargar un programa ya ensamblado o compilado que haya sido almacenado en el formato HEX de Intel. La mayoría de los ensambladores y compiladores del 8051 tienen una opción para almacenar la salida ensamblada o compilada en este formato. La figura I-2 muestra cómo se lleva esto a cabo al seleccionar la opción de abrir un archivo en el estándar de Intel (“Open Intel Standard File”) del menú de archivos (“File”) y buscando después el archivo HEX de Intel.

El programa cargado está ahora listo para probarse. Primero, podemos optar por examinar varios componentes del 8051, tales como la memoria RAM interna, los puertos de E/S, registros con funciones especiales (SFRs) y temporizadores. Seleccionamos cada uno de estos componentes en el menú “View”, como se muestra en la figura I-3.

La figura I-4 muestra la ventana de memoria interna (RAM) que seleccionamos del menú “View”. Todos los renglones de la primera columna representan las direcciones de 16 ubicaciones en memoria, cuyo contenido se muestra en las 16 columnas de la derecha. Por ejemplo, el primer renglón consta de las ubicaciones en memoria interna 00H a 0FH y, como podemos ver en la figura I-4, el contenido de cada una es 0. Para modificar cualquier ubicación, haga clic sobre ella y escriba el nuevo valor en forma hexadecimal. También puede habilitar (activar) o dejar deshabilitado (borrar) cualquiera de los 8 bits de esa ubicación.

¹Creado por Vault Information Services, 8174 S. Holly PMB 272, Littleton, CO 80122, EUA.
Correo electrónico: sim8052@vaultbbs.com.

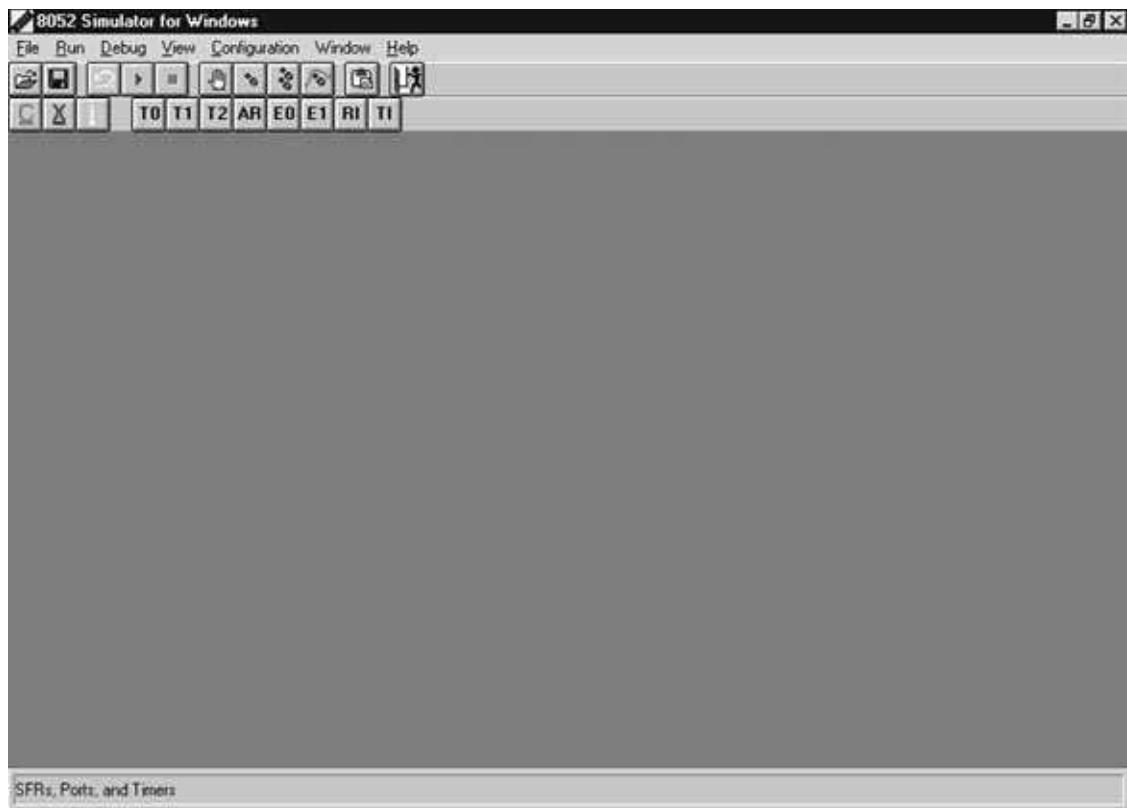


FIGURA I-1
Simulador del 8052

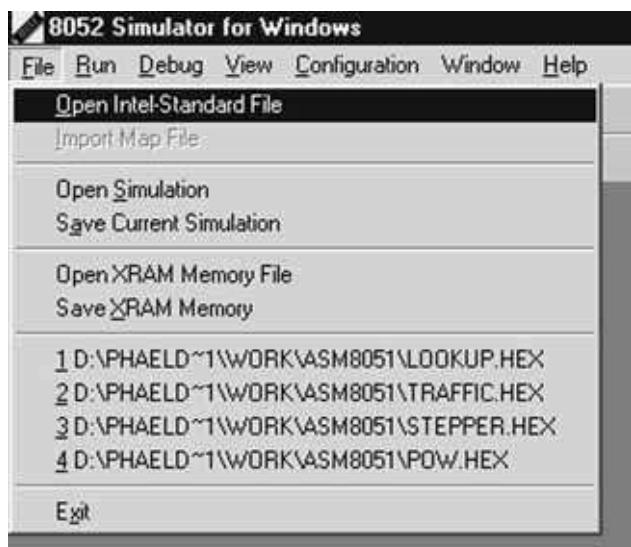
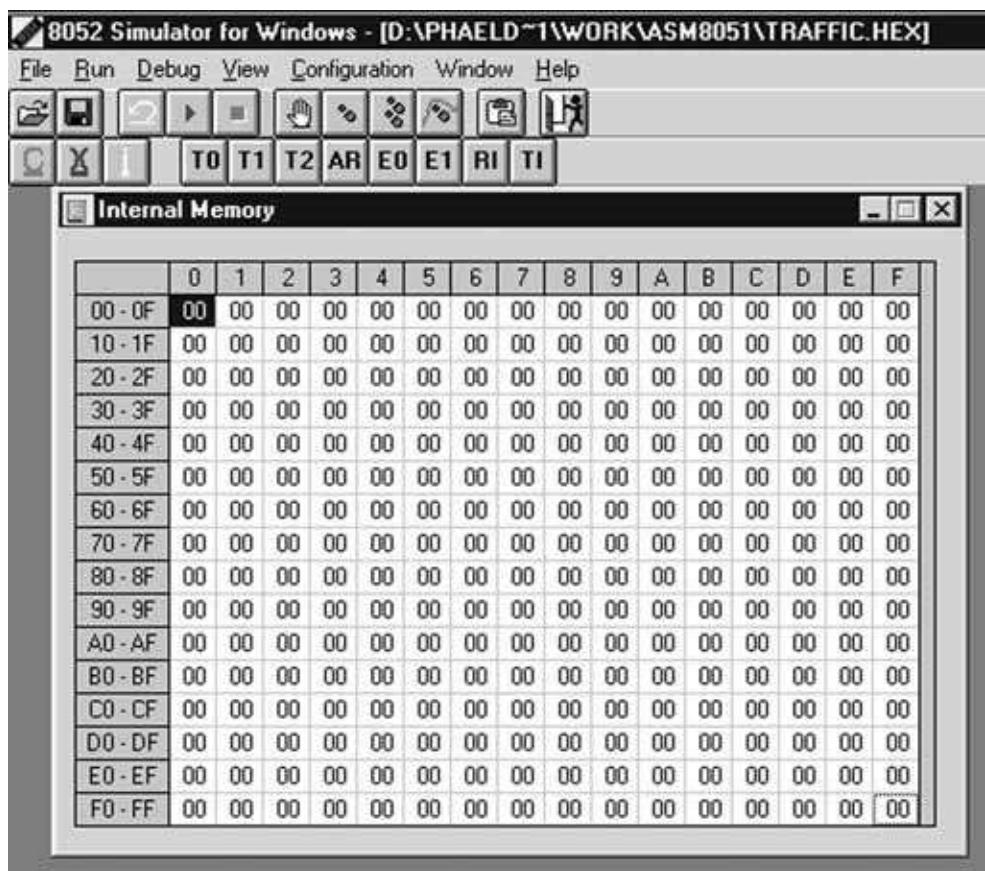
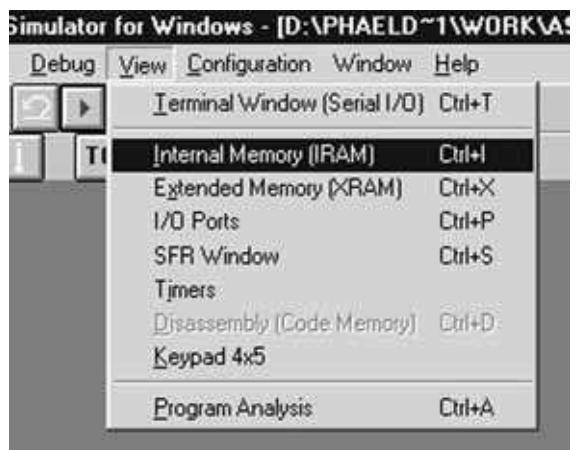


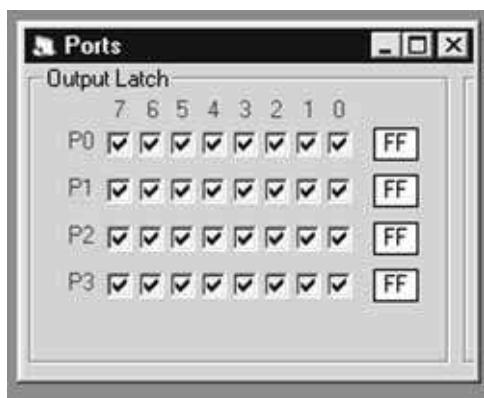
FIGURA I-2
Apertura de un archivo HEX
de Intel

FIGURA I-3

Examen de los componentes internos del 8051

**FIGURA I-4**

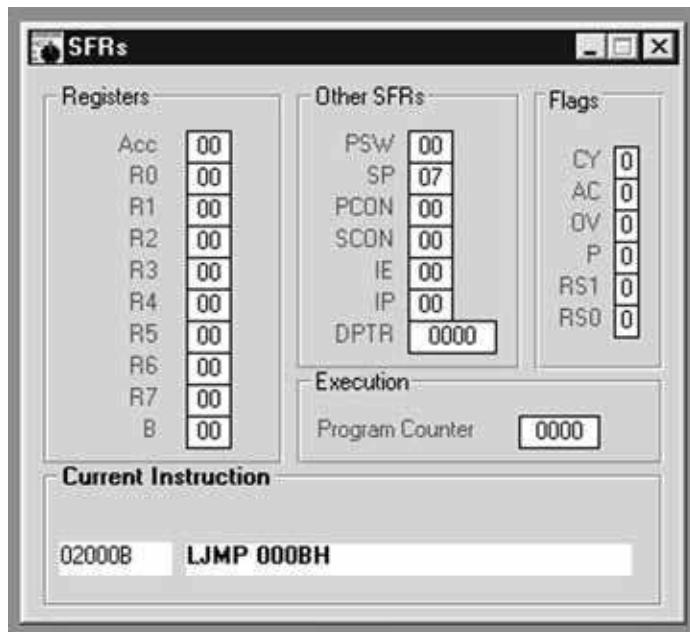
Examen de la memoria RAM interna

**FIGURA I-5**

Examen de los puertos de E/S

También podemos examinar el contenido de hasta 64K de memoria RAM externa desde el menú “View”. Al seleccionar la opción para los puertos de E/S (“I/O Ports”) del menú “View” se presenta la ventana de puertos (“Ports”, vea la figura I-5), lo cual es simple y directo. La figura muestra el contenido de los cuatro puertos de E/S. Podemos modificar su contenido utilizando un proceso similar al empleado para modificar la memoria RAM interna o externa.

La figura I-6 muestra la ventana de SFRs, la cual presenta el contenido de todos los SFRs localizados en las 128 ubicaciones superiores de memoria RAM interna del 8051. Podemos modificar su contenido haciendo clic en los controles de edición y escribiendo el nuevo valor en

**FIGURA I-6**

Examen de los SFRs

hexadecimal. Observe también que la sección para la instrucción actual (“Current Instruction”) aparece en la parte inferior de la ventana de SFRs. El control de edición situado en la parte izquierda de esta área muestra la ubicación en memoria para código de la siguiente instrucción a ser ejecutada, mientras que el control de edición de la derecha muestra esa siguiente instrucción.

La ventana de los temporizadores (“Timers”) también está disponible para examinar (vea la figura I-7), y presenta el contenido de los registros TCON y TMOD además de los valores de conteo de los temporizadores 0, 1 y 2 (para el 8052).

El puerto serial se simula también mediante una ventana de terminal (vea la figura I-8) que muestra cualquier carácter enviado al puerto serial. La versión completa del simulador del 8052 soporta además la conexión del puerto serial del 8051 al puerto serial/COMM de la computadora,

FIGURA I-7

Examen de los temporizadores

**FIGURA I-8**

Examen de la ventana de terminal

Apparent Baud Rate:

921583

(Serial Port Configured Ok, Reception Disabled)

de tal forma que cualquier dato enviado o recibido del puerto serial del 8051 se enviará o recibirá desde el puerto serial/COMM de la computadora.

Suponga que hemos cargado el archivo HEX correspondiente al programa escrito en lenguaje ensamblador de la figura 11-33. Este es el programa para el sistema de semáforos peatonal. Para ejecutarlo y probarlo, muestre primero las ventanas relevantes utilizando el menú “View”. Por ejemplo, podemos mostrar las ventanas de RAM interna, puertos de E/S y de los SFRs. A continuación, elegimos la ejecución del programa completo mediante “Execute Program” del menú de ejecución (“Run”). Podemos detener la ejecución del programa en cualquier momento utilizando “Stop” en el mismo menú. En forma alternativa, podemos ejecutar cada instrucción una por una seleccionando “Single Step” del menú de depuración (“Debug”) o utilizando la tecla F8, que es la tecla aceleradora para elegir la opción que nos permitirá ejecutar cada paso por separado.

Por ejemplo, la primera instrucción en el programa del semáforo es saltar a PRINCIPAL. Oprima F8 una vez para ejecutar esta instrucción y después oprima F8 de nuevo para ejecutar la siguiente instrucción, la cual transferirá el valor 81H al registro IE (vea la figura I-9). Veremos que el contenido mostrado en el registro IE es ahora 81H. De manera similar, podemos recorrer el resto del programa paso por paso, examinando los SFRs, las ubicaciones en RAM interna, los puertos de E/S, o los temporizadores adecuados para ver si están cambiando como lo esperamos. Para reiniciar el programa, simplemente seleccionamos “Reset Program” en el menú de ejecución (“Run”).

En cualquier momento durante la ejecución del programa o conforme lo ejecutamos paso por paso, también podemos seleccionar la opción para realizar un análisis (“Program Analysis”) en el menú “View”, el cual presenta información tan útil como el número de instrucciones ejecutadas hasta ese momento y cuántos ciclos de máquina se han utilizado. También existe un área para el historial de la ejecución (“Execution History”), en la ventana de análisis del programa

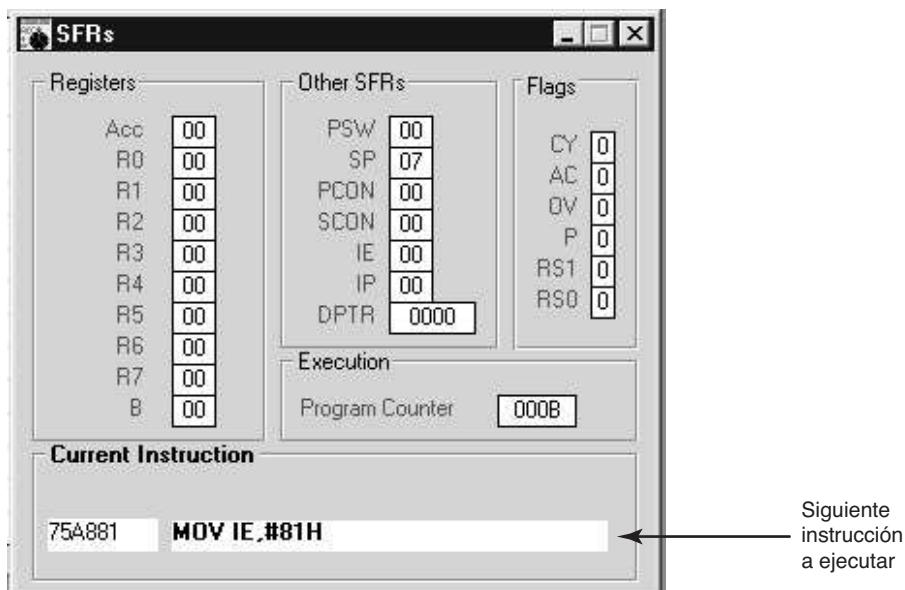


FIGURA I-9

Verificación del programa paso por paso

(“Program Analysis”), que presenta una lista de todas las instrucciones ejecutadas previamente (vea la figura I-10).

Una característica adicional incluida en el simulador del 8052 es el simulador de un teclado numérico de 4×5 (vea la figura I-11). Éste simula las funciones de un teclado numérico de 4×5 , parecido al teclado numérico hexadecimal de 4×4 que vimos en el capítulo 11. Podemos personalizar los nombres de cualquier tecla con un simple clic del botón derecho en la tecla que queremos cambiar y escribiendo entonces el nombre nuevo.

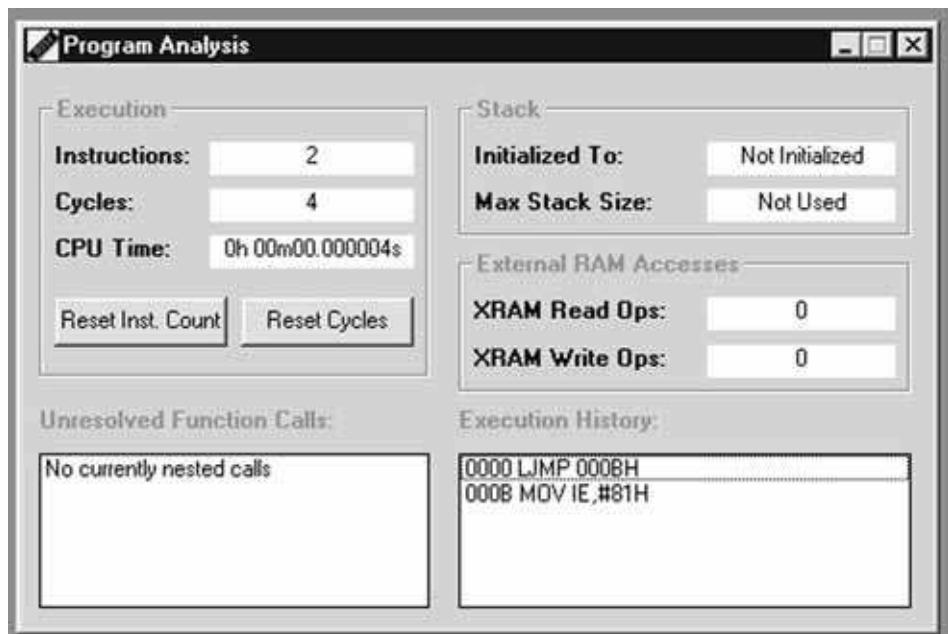
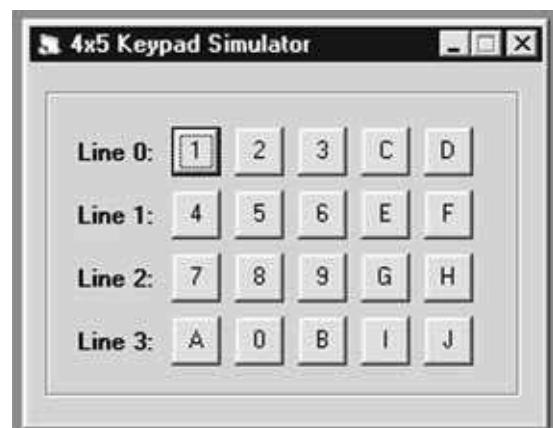


FIGURA I-10

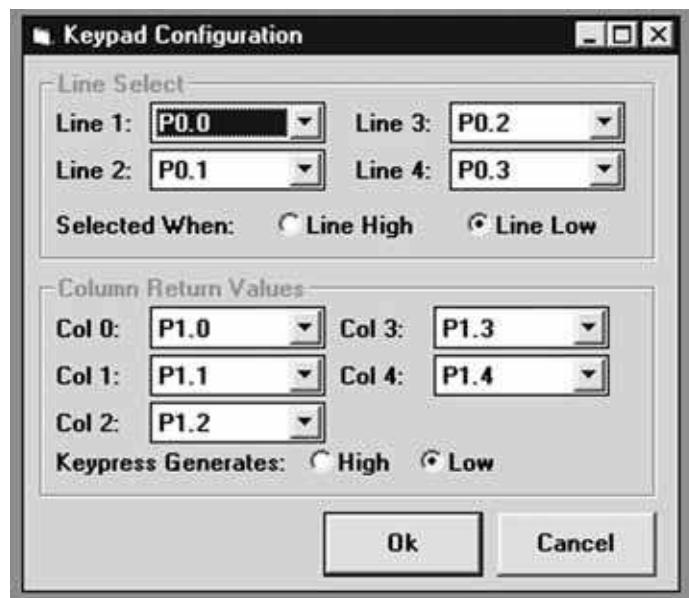
Ventana de análisis del programa (“Program Analysis”)

FIGURA I-11

Simulador de un teclado numérico



Podemos personalizar también las terminales de puerto que serán utilizadas para conectar los renglones y las columnas, seleccionando la opción para configurar el teclado numérico (“Keypad Configuration”) del menú de configuración (“Configuration”). Esto se muestra en la figura I-12.

**FIGURA I-12**

Ventana de configuración del teclado numérico (“Keypad Configuration”)



El Advanced Encryption Standard

INTRODUCCIÓN

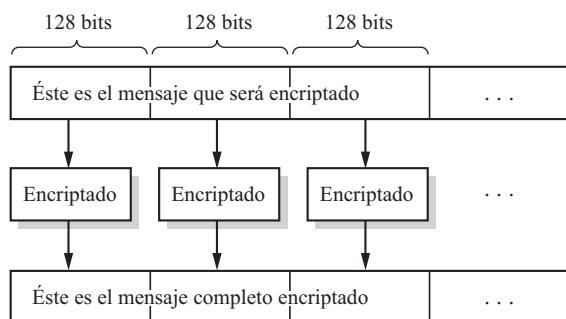
Conforme el 8051 se utiliza cada vez más en tarjetas inteligentes, donde la seguridad de la información es casi siempre importante, a menudo se le emplea para ejecutar implementaciones en software o en hardware de métodos de encriptado. El estándar actual para efectuar el encriptado, o cifrado, es el Advanced Encryption Standard (AES), un método reciente y seguro que adoptó oficialmente en el año 2000 el U. S. National Institute of Standards & Technology” (NIST, Instituto Nacional Estadounidense de Estándares y Tecnología) para encriptar información confidencial y no clasificada en los Estados Unidos. Se espera que el AES reemplace al Data Encryption Standard en las tarjetas de cajero automático, tarjetas inteligentes, transacciones en línea, y en otras aplicaciones relacionadas con la seguridad. Por lo tanto, el estudio del 8051 no estaría completo sin un análisis sobre el AES. Este apéndice tiene la intención de familiarizar al lector con el funcionamiento del AES para que sepa cómo implementarlo con el 8051.

ENCRYPTADO DE BLOQUES

En la actualidad, los mensajes y la información se almacenan digitalmente en archivos en la computadora, en términos de bits de unos y ceros. Estos mensajes varían en tamaño, desde unos pocos bytes hasta cientos de megabytes. Por esta razón, los actuales métodos de encriptado especifican un tamaño de bloque de entrada estándar, el cual es generalmente de 128 bits. Un mensaje a encriptar debe dividirse primero en bloques de 128 bits, y entonces se encripta cada bloque. Después, los bloques encriptados se concatenan para formar un mensaje encriptado. Vea la figura J-1 para apreciar una ilustración de este proceso.

FUNCIONAMIENTO DEL AES

Para cumplir con el propósito de describir el AES, sus diseñadores representaron el bloque de entrada de 128 bits como un arreglo de bytes de 4×4 . Un elemento de este arreglo se denota mediante a_{ij} , donde i y j son los índices de fila y de columna, respectivamente, numerados del 0 al 3, como se muestra en la figura J-2.

**FIGURA J-1**

Encriptado de un mensaje, un bloque a la vez

Para poder utilizar el AES y encriptar el bloque de 128 bits, se pide una contraseña al usuario, conocida como **llave secreta**. Esta llave puede ser de 128, 192 o 256 bits. Para facilitar la descripción en este ejemplo, nos concentraremos de aquí en adelante en el AES con una llave de 128 bits. Los detalles de cómo utilizar el AES con las otras dos versiones de la clave son parecidos.

Esta llave secreta de 128 bits se modifica a lo largo de una serie de operaciones, conocidas colectivamente como **plan de llave**. El propósito de esto es utilizar la llave secreta para generar un total de 11 llaves diferentes también de 128 bits, cada una de las cuales se conoce como **llave de redondeo**.

Veamos ahora cómo funciona el encriptado AES. Antes de comenzar, recordemos que aunque se involucran numerosos nombres específicos, el AES está diseñado para adecuarse a cualquier implementación en procesadores y microcontroladores de 8 bits tales como el 8051, así que sus operaciones son simples y directas.

El encriptado AES implica básicamente la iteración de una operación, llamada **redondeo**, por un total de 10 veces. Dentro de cada operación de redondeo ocurren las siguientes 4 operaciones más pequeñas: **SubBytes**, **DesplazaRenglones**, **CombinaColumnas**, y **AgregaLlaveRedondeo**.

La operación de SubBytes está basada en una tabla, conocida como **caja de sustitución** o caja s. Cada byte, a_{ij} , del arreglo de 4×4 mostrado en la figura J-2 se sustituye por un byte completamente diferente, b_{ij} . ¿Cómo se obtiene este b_{ij} ? El elemento original a_{ij} se utiliza como un índice para referirnos al elemento correspondiente en la tabla caja s. Dicho elemento es b_{ij} . La figura J-3 muestra la operación SubBytes.

FIGURA J-2

Representación de un bloque de 128 bits del AES

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

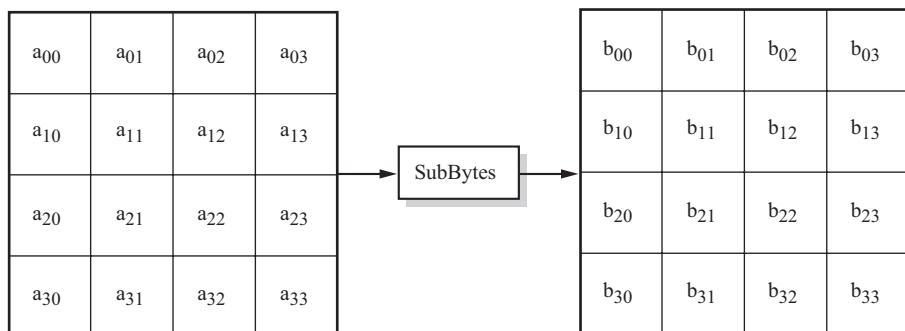


FIGURA J-3
Operación SubBytes

La operación DesplazaRenglones es aún más simple. En el arreglo de 4×4 , la primera fila se deja intacta. La segunda fila se desplaza un byte (8 bits) hacia la izquierda, la tercera fila se desplaza 2 bytes hacia la izquierda, y la cuarta fila se desplaza 3 bytes hacia la izquierda (vea la figura J-4).

La operación CombinaColumnas es un poco más compleja, por lo que no describiremos los detalles técnicos aquí. Si el lector está interesado en conocerlos, puede consultar la documentación del AES en la bibliografía para obtener más detalles. Explicada de una manera simple, esta operación sustituye cada columna con una columna completamente distinta, basada en una serie de operaciones que pueden implementarse con eficiencia en el 8051. La figura J-5 ilustra la operación CombinaColumnas, donde la columna sombreada muestra que la primera columna en la entrada se sustituye por una nueva columna que aparece también en la primera columna en la salida.

La operación AgregaLlaveRedondeo toma los 128 bits del arreglo de 4×4 y realiza una operación OR exclusivo con una llave de redondeo de 128 bits, que se generó con base en la llave secreta, como indicamos antes.

Hemos descrito todas las operaciones que se pueden encontrar dentro del AES. Como resumen, el encriptado AES toma un bloque de entrada de 128 bits y lo somete a 10 ciclos de operaciones, cada uno de los cuales consta de las operaciones SubBytes, DesplazaRenglones, CombinaColumnas, y AgregaLlaveRedondeo aplicadas en secuencia. Sin embargo, se agrega

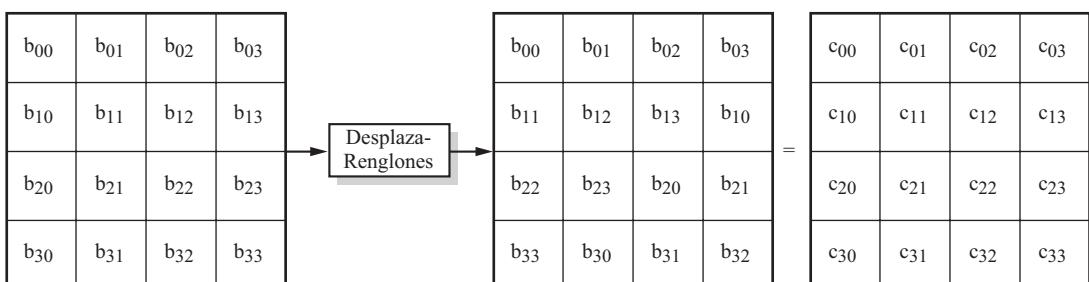
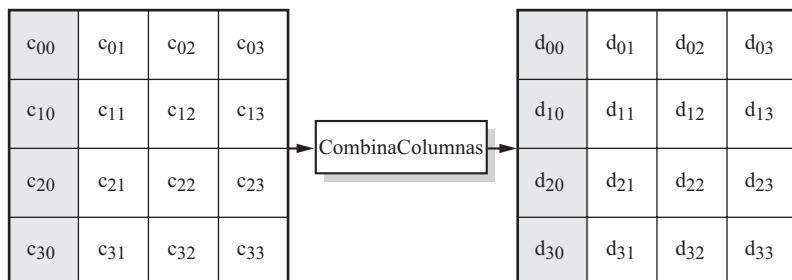


FIGURA J-4
Operación DesplazaRenglones

**FIGURA J-5**

Operación CombinaColumnas

una operación adicional de AgregaLlaveRedondeo antes del primer ciclo y la operación Combinacolumnas se omite en el último ciclo. Estos ligeros ajustes son consideraciones de diseño para lograr que el encriptado y el desencriptado sean parecidos en cuanto a su estructura. Esta similitud es muy conveniente ya que permite efectuar implementaciones más compactas en hardware y en software. La figura J-6 presenta una ilustración del proceso completo de encriptado AES.

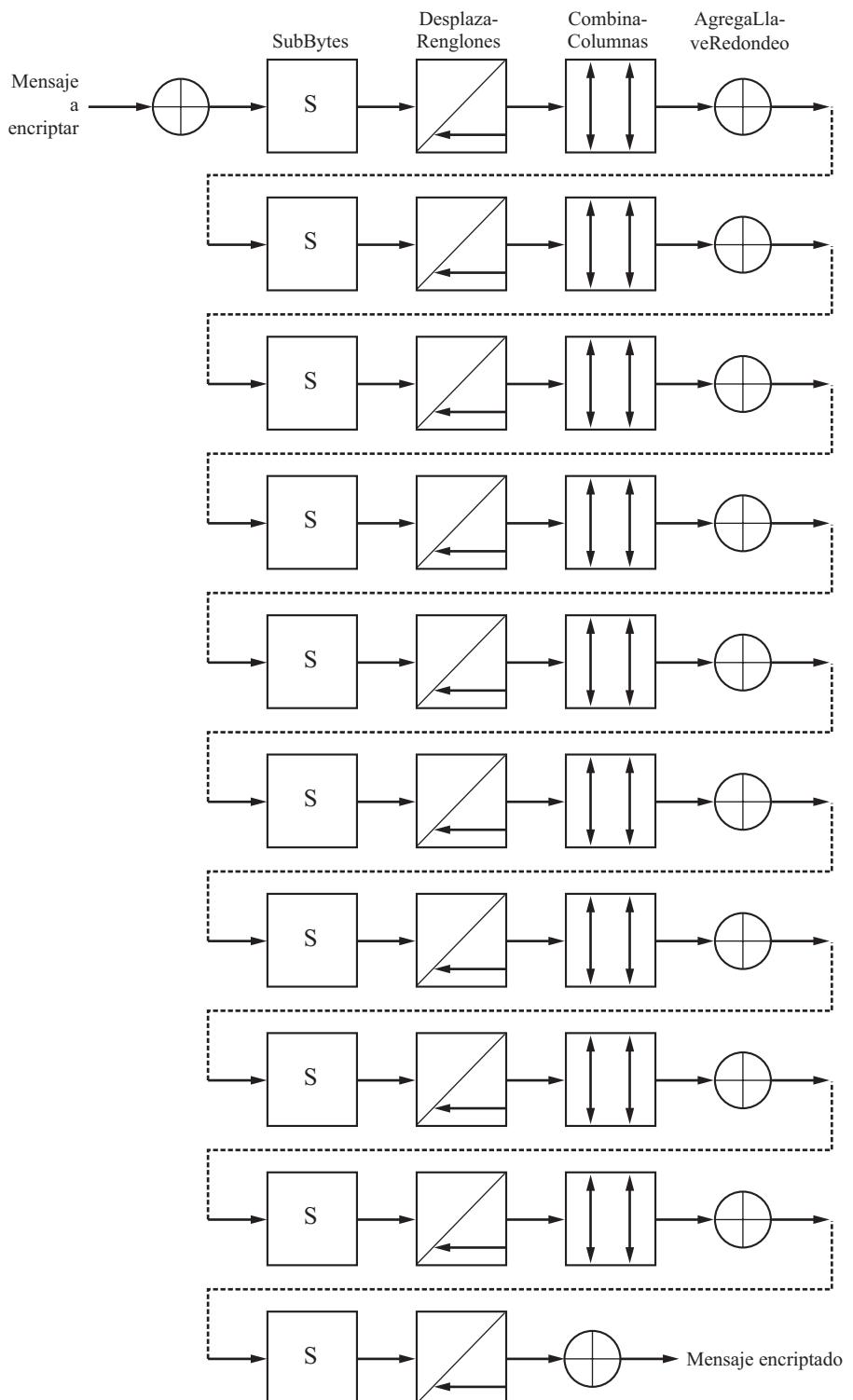


FIGURA J-6
Encriptado AES



Fuentes de productos de desarrollo para el 8051

Allen Systems, 2151 Fairfax Road, Columbus, OH 43221

<i>Producto:</i>	FX-31 8052-BASIC SBC
<i>Descripción:</i>	Computadora de una sola tarjeta basada en el 8052 con un interpretador de BASIC incorporado
<i>Producto:</i>	CA-51
<i>Descripción:</i>	Ensamblador cruzado para el 8051
<i>Computadora anfitriona:</i>	IBM PC y compatibles
<i>Producto:</i>	DP-31/535
<i>Descripción:</i>	Single-board computer based on Siemens 80535 CPU

Applied Microsystems Corp., 5020 148th Ave. N. E., P.O. Box 97002, Redmond, WA 98073-9702

<i>Producto:</i>	EC7000
<i>Descripción:</i>	Emulador del microcontrolador 8051
<i>Computadora anfitriona:</i>	IBM PC y compatibles

Aprotek, 1071-A Avenida Acaso, Camarillo, CA 93010

<i>Producto:</i>	PA8751
<i>Descripción:</i>	Adaptador para programación del 8751
<i>Features:</i>	Adapta los microcontroladores 8751 EPROM a cualquier programador de EPROM como un 2732

**AvoSET Systems, Inc., 804 South State St.,
Dover, DE 19901**

<i>Producto:</i>	XASM51
<i>Descripción:</i>	Ensamblador cruzado para el 8051
<i>Computadora anfitriona:</i>	IBM PC y compatibles
<i>Producto:</i>	AVSIM51
<i>Descripción:</i>	Simulador de la familia 8051
<i>Computadora anfitriona:</i>	IBM PC y compatibles

**Binary Technology, Inc., P.O. Box 67, Meridan,
NH 03770**

<i>Producto:</i>	SIBEC-II
<i>Descripción:</i>	Computadora de una sola tarjeta basada en el 8052 con interpretador de Basic incorporado

**Cybernetic Micro Systems, Inc., P.O. Box 3000,
San Gregorio, CA 94074**

<i>Producto:</i>	CYS8051
<i>Descripción:</i>	Ensamblador cruzado para el 8051
<i>Computadora anfitriona:</i>	IBM PC y compatibles
<i>Características:</i>	No genera módulos relocalizables
<i>Producto:</i>	SIM8051
<i>Descripción:</i>	Simulador del 8051
<i>Computadora anfitriona:</i>	IBM PC y compatibles
<i>Producto:</i>	CYP8051
<i>Descripción:</i>	Programador de EPROM para el 8751
<i>Interfaz:</i>	Serial RS232

**Decimation, Inc., 3375 Scott Blvd., Suite #236,
Santa Clara, CA 95054**

<i>Producto:</i>	ASM51, PLM51, SIM51, etc.
<i>Descripción:</i>	Software de desarrollo para el 8051
<i>Computadora anfitriona:</i>	VAX, PDP-11, IBM PC y compatibles

**HiTech Equipment Corp., 9560 Black Mountain Road,
San Diego, CA 92126**

Producto: 8051 SIM
Descripción: Simulador del 8051
Computadora anfitriona: Microcomputadoras IBM PC/XT o Z80 CP/M

**Huntsville Microsystems, Inc., P.O. Box 12415, 4040
South Memorial Parkway, Huntsville, AL 35802**

Producto: SBE-31, HMI-200-8051
Descripción: Emulador del 8051
Interfaz: Serial
Computadora anfitriona: Diversas computadoras con el sistema operativo CP/M o MS-DOS

**Keil Software, Inc., 1501 10th Street, Suite 110,
Plano, TX 75074**

Producto: μVision2 IDE
Descripción: Compilador y depurador para C del 8051
Computadora anfitriona: IBM PC y compatibles

**Logical Systems Corp., 6184 Teall Station, Syracuse,
NY 13217**

Producto: UPA8751
Descripción: Adaptador para la programación del 8751
Características: Adapta un 8751 a un dado 2732 para su instalación en cualquier programador de EPROM
Producto: SIM51
Descripción: Simulador y depurador del 8051
Computadora anfitriona: IBM PC (MS-DOS, CP/M) y compatibles

Micromint, Inc., 4 Park Street, Vernon, CT 06066

Producto: BCC52
Descripción: Computadora de una sola tarjeta basada en el 8052 con un interpretador incorporado

**Nohau Corp., 51 East Campbell Ave., Suite 107E, Campbell,
CA 95008**

Producto: EMV51-PC
Descripción: Emulador del 8051
Computadora anfitriona: IBM PC y compatibles

**Relational Memory Systems, Inc., P.O. Box 6719, San Jose,
CA 95150**

Producto: ASM51, RLINK, RLOC, RLIB, OBJCON
Descripción: Herramientas para el desarrollo de software en el 8051
Computadora anfitriona: IBM PC y compatibles

**Scientific Engineering Laboratories, Inc., 104 Charles Street,
Suite 143, Boston, MA 02114**

Producto: XPAS51
Descripción: Compilador cruzado de PASCAL para el 8051
Computadora anfitriona: IBM PC y compatibles

Single Board Systems, P.O. Box 3788, Salem, OR 97306

Producto: SBS-52
Descripción: Computadora de una sola tarjeta basada en el 8051 con un interpretador de BASIC incorporado

**Software Development Systems, Inc., 3110 Woodstock Drive,
Downers Grove, IL 60515**

Producto: A51
Descripción: Ensamblador cruzado para el 8051
Computadora anfitriona: Diversos sistemas ejecutando en MS-DOS, Xenix o Unix

**Universal Cross Assemblers, P.O. Box 384, Bedford,
Nova Scotia B4A 2X3, Canada**

Producto: CROSS-16
Descripción: Ensamblador cruzado para el 8051
Computadora anfitriona: IBM PC y compatibles

URDA, Inc., 1811 Jancey St., Suite #200, Pittsburgh, PA 15206

Producto: SBC-51

Descripción: Versión en tarjeta para PC de la SBC-51 descrita en este texto

Z-World, 2065 Martin Ave. #110, Santa Clara, CA 95050

Producto: Coprocesador para la IBM PC

Descripción: Coprocesador que se incorpora a una PC o PC/AT; ejecuta el sistema operativo y herramientas de desarrollo del software ISIS de Intel

FABRICANTES DE CIRCUITOS INTEGRADOS (CI)

Las siguientes compañías son fabricantes y/o desarrolladores de los CI's del 8051 y sus derivados.

Intel Corp., 3065 Bowers Avenue, Santa Clara, CA 95051

Siemens Components, Inc., 2191 Laurelwood Road, Santa Clara, CA 95054

Signetics/Philips, 811 East Arques Ave., Sunnyvale, CA 94088-3409

Advanced Micro Devices, Inc., 901 Thompson Place, P.O. Box 3453, Sunnyvale,
CA 94088-3453

Fujitsu Microelectronics, Inc., 3320 Scott Blvd., Santa Clara, CA 95054-3197

BIBLIOGRAFÍA

ARTÍCULOS

- Ball, S. Embedded debugging tricks. *Circuit Cellar* (1995, septiembre): 20-23.
- Boyet, H., y R. Katz. The 8051 one-chip microcomputer. *Byte* (1981, diciembre): 288-311.
- Cantrell, T. Audio processor chips for the masses. *Circuit Cellar* (1995, noviembre): 70-76.
- Cantrell, T. Chip on patrol. *Circuit Cellar* (1995, junio): 64-71.
- Cantrell, T. Plan '251 for outer space!: Intel's 8xC251SB. *Circuit Cellar* (1995, marzo): 72-77.
- Cantrell, T. UFO alert. *Circuit Cellar* (1995, enero): 100-107.
- Cheung, H. DRAM on an 8031: It's not as hard as you'd think. *Circuit Cellar* (1994, septiembre): 24-31.
- Ciarcia, S. Build the BASIC-52 computer/controller. *Byte* (1985, agosto): 105-117.
- Ciarcia, S. Build a gray-scale video digitizer —Part 1: Display/receiver. *Byte* (1987, mayo): 95-106.
- Ciarcia, S. Build a gray-scale video digitizer —Part 2: *Byte* (1987, junio): 129-138.
- Ciarcia, S. Build an intelligent serial EPROM programmer. *Byte* (1986, octubre): 103-119.
- Ciarcia, S. Build a trainable infrared master controller. *Byte* (1987, marzo): 113-123.
- Ciarcia, S. Why microcontrollers? —Part 1: *Byte* (1988, agosto): 239-245.
- Ciarcia, S. Why microcontrollers? —Part 2: *Byte* (1988, septiembre): 303-323.
- Collier, M., y F. Gweme. Preventing the ultimate blow: A portable checking unit for 8751s. *Circuit Cellar* (1994, septiembre): 48-43.
- Dinwiddie, G. An 8031 in-circuit emulator. *Byte* (1986, julio): 181-194.
- Dybowski, J. Atmel's AT29C2051 flash-based microcontroller. *Circuit Cellar* (1995, febrero): 76-80.
- Dybowski, J. Beef up the 8052 with the DS87C520. *Circuit Cellar*, núm. 52 (1994, noviembre): 76-82.
- Dybowski, J. Embedded development. *Circuit Cellar* (1995, marzo): 78-84.
- Messick, P. y J. Battle. Build a MIDI input for your Casio SK-1. *Keyboard* (1987, agosto): 34-40.
- Natarajan, K. S., y C. Eswarn. Design of a CCITT V.22 modem. *Microprocessors and Microsystems* 12(9) (1988, noviembre): 532-535.

- National Institute of Standards and Technology (NIST). *Advanced Encryption Standard, Federal Information Processing Standard (FIPS) 197.* (2002). Internet: <http://csrc.nist.gov/encryption/aes>.
- Schenker, J. Controller chip cuts keyboard redesign to weeks. *Electronics* 61(2) (1988, 21 de enero): 42E-42F.
- Vaidya, D. M. Microsystem design with the 8052-BASIC microcontroller. *Microprocessors and Microsystems* 9(8) (1985, octubre): 405-411.
- Vaidya, D. M. Software development for the 8052-BASIC microcontroller. *Microprocessors and Microsystems* 9(10) (1985, diciembre): 481-485.
- Wallace, H. No emulator? Try a one-wire debugger. *Circuit Cellar* (1995, enero): 20-23.
- Warner, W. Use a single-chip microprocessor as the heart of your position controller. *EDN* (1988, 1 de septiembre): 161-168.
- Yeskey, D. In-circuit 8051 emulator is a tool for the masses. *Electronic Design* (1988, 7 de enero): 202.

LIBROS

- 8-bit Embedded Controllers (270645)*, Santa Clara, CA: Intel, 1991.
- Ayala, K. J. *The 8051 Microcontroller: Architecture, Programming, and Applications*. Nueva York: West, 1991.
- Barnett, R. J. *The 8051 Family of Microcontrollers*. Nueva Jersey: Prentice Hall, 1995.
- Boyet, H. y R. Katz. *The 8051: Programming Interfacing Applications*. Nueva York: MTI Publications, 1981.
- Cx51 Compiler: Optimizing C Compiler and Library Reference for Classic and Extended 8051 Microcontrollers*. Keil Software, 2001.
- Getting Started with µVision2 and the C51 Microcontroller Development Tools*. Keil Software, 2001.
- Mazidi, M. A., y J. G. Mazidi: *The 8051 Microcontroller and Embedded Systems*. Nueva Jersey: Prentice Hall, 2000.
- MCS-51 Macro Assembler User's Guide (9800937-03)*. Santa Clara, CA: Intel, 1983.
- Schultz, T. C and the 8051: *Hardware, Modular Programming, and Multitasking*. Nueva Jersey: Prentice Hall, 1998.
- Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Nueva York: John Wiley & Sons, 1996.
- Stallings, W. *Cryptography and Network Security: Principles and Practice*. Nueva Jersey: Prentice Hall, 1999.
- Stewart, J. W., y K. X. Miao. *The 8051 Microcontroller: Hardware, Software and Interfacing*. Nueva Jersey: Prentice Hall, 1999.
- Yeralan, S., y A. Ahluwalia. *Programming and Interfacing the 8051 Microcontroller*. Massachusetts: Prentice Hall, 1995.

ÍNDICE

- @, símbolo arroba, 157, 235
#, símbolo, 157-158
\$LIST, directiva, 300
10kHz, onda cuadrada de, ejemplo, 98-99, 206-207
13 bits, modo de temporizador de, 91
16 bits, modo de temporizador de, 92, 94
1kHz, onda cuadrada de, ejemplo, 99, 206-207
4 pasos, motor paso a paso de, 311
74HC165, 286
75HC165, 287-288
8 bits, UART de, con velocidad en baudios variable, 115-117
8 pasos, motor paso a paso de, 311
8031
 acceso externo (EA), 21
 resumen de espacio de memoria, 24-25
8032, 19
 acceso externo (EA), 21
 mejoras, 41-43
8051
 acceso externo (EA), 21
 características, 17-22
 configuración, 260-263
 control, mediante un oscilador TTL, 22-23
 diagrama de bloques, 19
 diagramas de terminales, 18-22
 fuentes de productos desarrollados, 521-525
 ilustración, 2
 operación NAND utilizando el, 13-15
 resumen de memoria para datos incorporada al chip, 24, 26-27
 Vea también MCS-51
8051, estilo de programación en C para el, 243-245
8051, hoja de datos del, 439-453
8051, lenguaje C para el
 apuntadores, 199-202
 arreglos, 198-199
como alternativa a la programación en lenguaje ensamblador, 191
compiladores, 192-193
ejemplos, 202-214
estructuras, 199
funciones, 202-204
memoria, tipos y modelos, 197-198
tipos de datos, 193-196
ventajas y desventajas, 191-192
8051, microcontroladores de alta velocidad derivados, 378-379
introducción, 377
MCS-151 y MCS-251, 377
microcontroladores con ADCs y DACs, 378
microcontroladores con memoria flash y NVRAM, 377-378
red, microcontroladores de, 379
seguros, microcontroladores, 379
8051, programa de monitoreo (MON51)
 del, 259, 457-498
“ES”, rutinas de, 488-489
carga archivo HEX de Intel, 479-481
comandos y especificaciones, 458
comandos, formato de, 459
comandos, sintaxis de, 459-461
conversión, subrutinas de, 478-479
descripción general, 457-458
diseño en, 462-463
enlazado, mapa de y tabla de símbolos, 465
entrada/salida, rutinas de, 473-478
Go, comando, 460
Intel, archivo HEX de, 465
lectura y escritura de registros con funciones especiales (SFRs), 484-487
Lo, comando, 460-461
macro.src, 493-494
modifica la memoria con un valor, 489-493
módulo principal para, 466-470
obtener parámetros de la línea de entrada, 470-473
operación general, 461
puentes de configuración de opciones, 464-465
Set, comando, 460
V12.HEX, 496-498
V12.M51, 494-496
vacía la memoria a la consola, 481-484
8052
 acceso externo (EA), 21
 mejoras, 41-43
 puerto 1, 19
 temporizador 2 de, 105-106
8052, simulador del, 507-514
abrir archivo HEX de Intel, 508
examen de componentes internos del 8051, 509
examen de puertos de E/S, 510
examen de RAM interna, 509
examen de registros con funciones especiales (SFRs), 510-511
examen de temporizadores, 511
examen de ventana de terminal, 511-512
introducción, 507-508
Keypad Configuration, ventana de configuración de teclado numérico, 514
Program Analysis, ventana de análisis del programa, 513
teclado numérico, simulador del, 513
verificación de programa, 512-513
8085, 21
8086, 21
8088, 21
80C31BH, 22
8255, 285-291, 336-337
9 bits, UART de, 117

Absoluto
 direcccionamiento, 57-58
 segmento de código, 171-173
 selección de segmento, 171-173
AC. *Vea* Acarreo auxiliar, bandera de (AC)

- ACALL. *Vea Llamada absoluta, instrucciones de (ACALL)*
- Acarreo
auxiliar, bandera de (AC), 30-31
bandera de, 29-30
- Acceso externo (EA), 21
- Acelerador de módulo aritmético (MAA), 379
- Activa bit, 424
- Actuador, 8
- ADC. *Vea Analógico a digital, convertidor (ADC)*
- ADC0804, 300-302
- ADD, 63, 386-387
- ADD A, 386-387
- ADDC. *Vea Suma con acarreo (ADDC)*
- ADELANTE, etiqueta, 238
- Advanced Encryption Standard (AES)
bloque, encriptado de, 515
definido, 375
ilustración del proceso, 519
introducción, 515
trabaja, como, 515-519
- Advanced Micro Devices, Inc. (AMD), 525
- AES. *Vea Advanced Encryption Standard (AES)*
- AgregaLlaveRedondeo, 516-519
- AJMP. *Vea Salto absoluto*
- Ajuste decimal del acumulador (DA A), 30-31, 63, 396-397
- ALE. *Vea Dirección, habilitación de latch de (ALE)*
- Alfabeto, 350
- Allen Systems, 521
- Almacenamiento en la pila (PUSH directa), instrucción, 420
- Almacenamiento
en línea, 7
masivo, 7
initialización/reservación de, 166-169
- Alta velocidad, microcontroladores de, 378-379
- Altavoz, 7
interfaz a, 276-279, 327-329
ejemplo, 100-101, 207-208
interfaz utilizando interrupciones, 146-148
- “Al tiempo de ejecución”, 96
- ALU. *Vea Aritmética lógica, unidad (ALU)*
- Analógica
entrada, 300-302, 342-344
salida, 296-300, 342-344
- Analógico a digital, convertidor (ADC), 378
- AND, 64, 71-72
lógico para variables bit, 391
lógico para variables byte, 389-391
- ANDED, operación realizada de, 26
- ANDing, realizando operaciones de, 12
- Anidada, subrutina, 84, 239
- ANL, 389-391
- ANL C, 391
- Apagado, modo de, 35-36
- Aplicación, software de, 8-9
- Aplicaciones, 10-12
- Apple Macintosh, 9
- Applied Microsystems Corp., 521
- Aprotek, 521
- Apuntador
con tipo, 201-202
de pila (SP), 33
tipo de memoria de, 201-202
- Apuntadores
con tipo, 201-202
definidos, 199
diferentes maneras para acceder a los datos, 200-201
sin tipo, 202
tipo de memoria, 201
uso de, en el 8051, estilo de programación en C para, 244
- Archivo
de listado, 155
almacenamiento de, 7
- ARCNET, protocolo de red “token ring”, 379
- Aritmética lógica, unidad (ALU), 4
- Aritméticas, instrucciones, 59-64
- Aritméticos, operadores, 161, 235
- Arreglo, 198-199, 244
- ASCII
códigos, 142-143, 155, 160, 455
tabla, para dígitos decimales, 198-199
- Asignación, operadores de, 235-236
- Asíncrono, formato, 111-112, 115
- ASM51
ensamblador, controles de, soportados por, 173-175
macros, facilidad de procesamiento de (MPL), 183-188
- AT89C15SND1C, 378
- AT89C15SND2C, 378
- Ataque de diccionario, 375
- ATM. *Vea Cajero automático (ATM)*
- Atmel, 378-379
- ATRAS, etiqueta, 237
- ATWebSEG-32, 379
- Autentificación, 372
- Autodisminución, modos de, 11
- Autoincremento, 11
- Autorrecarga
de 8 bits, modo de, 92
modo de, 105-106
- Avoset Systems, Inc., 522
- B, registro, 32
- “Bailando con bits”, 66
- Banco de registros, 27-28
- Bandera, 87
- Bases de números, 160
- Baudios
generación de velocidad en, 87, 106
resumen de velocidades en, 122
velocidad en, 111
- Bifurcación de programa, instrucción de, 60
- interrupciones, 75-77
saltos condicionales, 77-78
subrutinas, 75-77
tablas de salto, 74-75
variaciones de, 73
- Binary Technology, Inc., 522
- BIOS. *Vea Sistema básico de entradas/salidas (BIOS)*
- Bit
dirección de, 158
tipo de datos, 193-195
- Bits de bandera de interrupción, 134
- Bloque
de decisión, 218
de entrada/salida, 218
- Booleana, instrucción de variable, 60
- Booleanas
instrucciones, 71-73
operaciones, 64
- Booleano, acumulador, 30
- Búfer de recepción, 111
- Bus, definido, 6. *Vea también Direcciones, bus de; Control, bus de; Datos, bus de*
- Búsqueda (“fetch”), 4-5, 37-38
- Búsqueda, subrutina de, 226-228
- Bytes, 11
- C, 151. *Vea también 8051, lenguaje C para; 8051, estilo de programación en C para, 243-245*
- CAD. *Vea Software asistido por computadora (CAD)*
- Cadenas de caracteres, 160
- CADSAL, subrutina, 240-242
- Cajero automático (ATM), 372
- Calculadora, proyecto para estudiantes, 363-365
- CALL, instrucciones, 73-76, 158-160.
Vea también Llamada absoluta,

- instrucciones de (ACALL);
LCALL
- Campos, 155-157. Vea también los índices para campos individuales
- CAN. Vea Red de área de controlador (CAN)
- Captura, modo de, 106
- Carácter de entrada, ejemplo de subrutina, 125, 209
- Carácter de salida (CARSAL), subrutina, 123-124, 209, 240-241
- Carga apuntador de datos con una constante de 16 bits, 413
- Cargador de inicialización, 8
- CASE, sentencia, 230-234, 237
- CD-ROM. Vea Disco compacto de memoria de sólo lectura (CD-ROM)
- Central, entorno, 255-256
- Centronics, interfaz paralela, 294-296, 340-341
- César
- cifrado del, 351, 372
 - Julio, 351
- Chips integrados a gran escala (LSI), 8
- CI. Vea Circuito integrado (CI)
- CICLO, etiqueta, 237
- Ciclo de desarrollo
- definido, 247-248
 - desarrollo de hardware, 249-251
 - desarrollo de software, 248-249
 - diagrama de flujo para, 248
 - pasos detallados en, 252
- “Ciclo de espera”, 124, 137
- Ciclo, estructura, 220
- búsqueda, subrutina de, 226-228
 - REPEAT/UNTIL, instrucción, 226
 - SUMA, subrutina, 222-224
 - WHILE/DO, estructura, 221-222, 224-225, 237
- Circuitería de interrupción, 10-11
- Circuito
- de interfaz, definido, 3
 - integrado (CI), 1, 5, 9. Vea también Memoria de acceso aleatorio (RAM); Memoria de sólo lectura (ROM)
- Circuito interintegrado, 379
- CJNE, instrucción, 77-78, 185, 392-394
- CLOCK, 7
- CLR bit. Vea Limpiar bit (CLR bit)
- CODE, directiva, 166
- Código, 172, 197
- dirección de, 158-159
 - segmento de, 243
- de autentificación de mensaje (MAC), 372
- de operación, ciclo de búsqueda, actividad de los buses para, 5 de operación de instrucción, 4-5
- Códigos de operación, 21, 37-38
- COM20051, 379
- Comandos, 255-257
- CombinaColumnas, 516-519
- Comentarios, 155, 238-239, 243
- bloque de, 237
 - campo de, 156-157
- Commodore Amiga, 9
- Compara acumulador, instrucción, 185
- Compara y salta si no es igual. Vea CJNE
- Compilador, 192-193
- cruzado, 251
- Complementa
- acumulador, 395
 - bit, 395-396
- Computadora
- de una sola tarjeta (SBC), 1, 250
 - definida, 3
- Comunicación
- con 3 conexiones, 303
 - entre múltiples procesadores, 119-120
- Condición, 221
- Conector de fuera de página, 218
- Confidencialidad, 372
- Conjunto de instrucciones, 4
- características, 10-12
 - descripción general, 49
 - direcccionamiento, modos de, 50-59
 - tipos de instrucciones, 59-78
- Conjunto, 226
- SETB, 424
- Constantes, definiciones de, 245
- Contador de programa (PC), 4
- Contador de ubicación, 153
- Control
- bus de, 3, 6
 - de flujo, operaciones de, 187-188
- Control/monitoreo, dispositivo de, 7-8
- Controlador de horno, ejemplo, 144-145, 212-213
- Controles generales, 173
- CPL A, 395
- CPL bit, 395-396
- CPU. Vea Unidad de procesamiento central (CPU)
- Criptograma, 350
- CRT. Vea Tubo de rayos catódicos (CRT)
- CTS. Vea Listo para enviar (CTS)
- Cuadrada, onda
- generación de, ejemplo, 98
- utilizando interrupciones de temporizador, ejemplo, 139-141
- CUADRADO, subrutina, 70
- Cybernetic Micro Systems, Inc., 522
- D2 por Motorola, 1
- DA A. Vea Ajuste decimal del acumulador (DA A)
- DAC. Vea Digital a analógico, convertidor (DAC)
- Dato binario, convertir voltaje a, 8
- Datos
- apuntador de (DPTR), 33-34, 402
 - bus de, 3, 6, 36-37
 - definidos, 6. Vea también Memoria interna para datos (data)
 - dirección de, 158
 - directivas de (DBs o DWs), 153, 243
 - inmediatos, 157-158
 - instrucciones de transferencia de, 59
 - RAM externa, 68-69
 - RAM interna, 66-68
 - tablas de búsqueda, 69-70
- internos indirectos, 172
- internos, 172
- segmento de, 243
- DB-25, 291, 294
- DB-9, 291
- DBIT, directiva, 168
- DEC byte, 398-399
- Decimal codificado en binario (BCD), 30, 63
- Decimation, Inc., 522
- Decodificación de dirección, 40-41
- Definición
- de almacenamiento (DS), directiva, 166-167
 - de byte (DB), directiva, 168-169
 - de palabra (DW), directiva, 169
- Definiciones, 244
- DENUEVO, etiqueta, 238
- Depuración, 248
- Depurador, 249
- DES. Vea Data Encryption Standard (DES)
- Desarrollo, entorno de, 256
- Desarrollo de programa
- ciclo de desarrollo, 247-251
 - comandos y entorno, 255-257
 - integración y verificación, 251-255
- Desbordamiento
- de temporizador, bandera de (TF2), 134
 - bandera de (OV)
 - definida, 31-32
 - modos de temporizador y, 90-92

- Descifrado, 350
 Desplazamiento, registros de, 282, 285-286, 333-336
 Desplazar acumulador, 423
 hacia la derecha a través de la bandera de acarreo, 423
 hacia la izquierda, 422
 hacia la izquierda a través de la bandera de acarreo, 422-423
DesplazaRenglones, 516-517
 Diagramas de flujo, 218
 Digital a analógico, convertidor (DAC), 296-300, 378
 DIP, interruptor, 277
 Dirección
 indirecta, 53-54, 157, 236
 habilitación de latch de (ALE), 21, 35-36
 Direcccionamiento, modos de
 absoluto, direcccionamiento, 57-58
 directo, direcccionamiento, 52-53
 indexado, direcccionamiento, 59
 indirecto, direcccionamiento, 53-54
 inmediato, direcccionamiento, 54-55
 largo, direcccionamiento, 58-59
 registro, direcccionamiento por, 50-52
 relativo, direcccionamiento, 55-57
 Direcciones, bus de,
 definido, 3, 6
 multiplexaje de, y de bus de datos, 36-37
 Directo, direcccionamiento, 52-53
 Disco
 compacto de memoria de sólo lectura (CD-ROM), 7
 interfaz a, 10
 óptico, 7
 Disminución, 398-399
 Disminuye y salta si no es cero (DJNZ), 77, 399-401
DJNZ. Vea Disminuye y salta si no es cero (DJNZ)
 DMA. Vea Memoria de acceso directo (DMA)
 Dos ondas cuadradas utilizando interrupciones, ejemplo, 140-141, 210-211
DPTR. Vea Datos, apuntador de (DPTR)
DS. Vea Reserva de memoria, directiva (DS)
DS1620
 configuración/estado, registro de, 306
 conjunto de comandos, 303
 interfaz a, 345-346
 software para interfaz, 304-306
DS5000, 378-379
DS5240, 379
DS89C420, 379
E/S
 dispositivo de. Vea Entrada/salida,
 dispositivo de (E/S, dispositivo de)
 estructura de puertos de, 22-23
EA. Vea Acceso externo (EA)
 Emulador en circuito, 253
ENCHIP, 172
 Encriptación, técnica de, 372
 Encriptado, 350-351, 372. Vea también Advanced Encryption Standard (AES)
 de bloque, 515
END, directiva, 164
 Enlazado
 de programa, 169-171
 operación de, 173-176
 Enlazador, 151-152
 operación de, 176
 /localizador, 173
 Ensamblado
 error al tiempo de, 249
 evaluación de expresión al tiempo de, 160-163
 Ensamblador
 control de estado de, 164-165
 controles de, 155, 173-175
 cruzado, 152
 de dos pasos, 153-155
 de un solo paso, 153
 definido, 151-152
 programa en lenguaje, 152
 símbolos especiales de, 157
 Ensamblador, directivas
 almacenamiento,
 inicialización/reservación de, 166-169
 ensamblador, control de estado de, 164-165
 programa, enlazado de, 169-171
 segmento, selección de, 171-173
 símbolos, definición de, 165-166
 Ensamblador, operación
 ASM51, 152-153
 paso dos, 153-155
 paso uno, 153
 Ensamblador, programación en lenguaje 8051, lenguaje en C de, como alternativa a, 191
 almacenando registros en la pila, 239
 comentarios, 238-239
 descripción general, 151-152
 enlace de segmentos y módulos
 relocalizables, 176-183
 enlazador, operación de, 173-176
 ensamblado, evaluación de expresión al tiempo de, 160-163
 ensamblador, controles de, 173
 ensamblador, directivas de, 164-173
 ensamblador, operación de, 152-155
 etiquetas, 237-238
 formato, 155-160
 macros, 183-188
 organización del programa, 243
 uso de igualdades, 239-240
 uso de subrutinas, 240-242
 Vea también Programación estructurada
 Entorno de desarrollo integrado
 μVision2 (IDE), 499-505
 archivos, ventana de ("File"), 500
 compilación y depuración, 499-505
 desensamblado, ventana de ("Disassembly"), 504
 espacio de trabajo para el, 499
 introducción, 499
 memoria, ventana de ("Memory"), 502
 registros, ventana de ("Regs"), 502
 salida, tabulador de opciones de ("Output"), 505
 serial, ventana ("Serial"), 503
 Entornos, 255-257
 Entrada externa, bandera de (EXF2), 134
 Entrada/salida
 ampliación del
 8255, 282-294, 336-337
 desplazamiento, registros de, 282, 285-286, 333-336
 dispositivo de, (E/S, dispositivo de), 9
 dispositivo de almacenamiento masivo, 7
 dispositivo de control/monitoreo, 8
 dispositivo de interfaz humana, 7
 ejemplo de programa, 176-177
 subrutinas de, 8-9
ENVIA, etiqueta, 238
EPROM. Vea Memoria borrable y programable de sólo lectura (EPROM)
 Error al tiempo de ejecución, 249
 Errores de redondeo, ejemplo, 103, 127
ES.LST, 178-183
 Espacio externo para direcciones de datos, 172
 Especial, operador, 162
ESPERA, etiqueta, 238
 Estado, 23-24

- Estado de programa, palabra de (PSW), 437-438
 acarreo auxiliar, bandera de (AC), 30-31
 acarreo, bandera de, 29-30
 desbordamiento, bandera de (OV), 31-32
 paridad, bit de (P), 32
 selección de banco de registros, bits de, 31
- Estándar de encriptado de datos (DES), 372, 515
- Estator, 310
- Estructura, 199
 de opción, 220
 CASE, instrucción, 230-231
 GOTO, instrucción, 234
 IF/THEN/ELSE, instrucción, 228-230
- Estructurada, programación
 8051, estilo de programación en C para, 243-245
 bloque de sentencia, 220
 ciclos, 220-228
 descripción general, 217-219
 ensamblador, estilo de programación en lenguaje, 237-243
 estructura de opción, 228-234
 opción, 220, 228-234
 sintaxis de seudocódigo, 234-237
 ventajas y desventajas, 219-220
- Ethernet, 379
- Etiqueta, 237-238
 local, 186-187
 campo de, 155-157
- Evento. *Vea Interrupciones*
- Eventos, conteo de, 87-88, 93
- EXF2. *Vea Entrada externa, bandera de (EXF2)*
- EXIT, subrutina, 241
- Expresión, ejemplo, 163
- EXTRN, directiva, 170
- Fabricantes de CIs, 525
- Facilidad
 de procesamiento de macros (MPL), 183-188
 de sustitución de cadenas, 183.
Vea también Macros
- Factorial, operación (!), 84
- Firma digital, 372
- Firmware, 13, 254
- Flecha de flujo de programa, 218
- Flip-flop de bandera, 87
- Forma de onda, ejemplo, 140
- Fujitsu Microelectronics, Inc., 525
- Full dúplex, 111, 117-118, 125-126
- Funciones
 definiciones de, 245
 definidas, 202
 ejemplo de, 203
 prototipos de, 245
 transferencia de parámetros, 203
 uso de, en el estilo de programación del lenguaje C del 8051, 244
 valores de regreso, 204
- G6RN, 307-310
- GND. *Vea Tierra (GND)*
- GOTO, sentencia, 234
- Habilitación
 de almacenamiento de programa (PSEN), 20-21, 35-37
 de interrupciones, registro de (IE), 35, 132-133, 435
 de recepción (REN), 118
 de salidas (OE), 20
- Hardware
 arquitectura de, 10-12
contra software, 8-9
 definido, 3
 desarrollo de, 249-251. *Vea también Ciclo de desarrollo*
 emulador de, 253
- HEX, registros, 253-254
- HiTech Equipment Corp., 523
- Huntsville Microsystems, Inc., 523
- IBM, 8-9
- IDE. *Vea Entorno de desarrollo integrado μVision2 (IDE)*
- IE. *Vea Habilitación de interrupciones, registro de (IE)*
- IF/THEN/ELSE, instrucción, 228-230, 237
- IGNORA, etiqueta, 238
- Igualdad, directiva (EQU), 153, 166
- Igualdades, 239-240
- Impresora, 7
- INC DPTR, 402
- INC, 401
- Inclusiones, 245
- Incrementa apuntador de datos, 402
- Incremento, 401-402
- Índice, direccionamiento por, 59
- Infraestructura de llave pública (PKI), 379
- Inicializando el puerto serial, 123, 208
- Inmediato, direccionamiento, 54-55
- Instrucción
 de intervalo largo, 96-102
 de intervalo mediano, 96-102
- de intervalo pequeño, 96-102
- registro de (IR), 4
- Instrucciones
 compensación por la pérdida de tiempo debido a, 103-105
 definiciones, 385-430
Vea también Ensamblador, directivas de Integridad, 372
- Intel Corporation, 1-2, 173, 431, 525
- Intel
 archivo HEX de, 508
 formato hexadecimal de, 253-254
- Intercambia al acumulador con variable de byte, 427-428
- Intercambia dígito (XCHD), 428
- Intercambia nibble dentro de acumulador (SWAP A), 66, 426-427
- Interfaz humana, 7
- Interrupción
 a nivel base, 131-132
 bandera de, 119
 de primer plano (a nivel base), 131-132
 de recepción, bandera de (RI), 119, 134
 de temporizador, 139-141
 de transmisión, bandera de (TI), 119, 134
 externa, 143-148
 nivel de, 131-132
 tiempo de propagación de, 149
- Interrupciones, 75-77
 altavoz utilizando, 146-148
 control de horno, ejemplo, 212-213
 descripción general, 131-132
 diseño de programas utilizando, 137-139
 dos ondas cuadradas utilizando temporizador, ejemplo, 210-211
 ejemplo, 131-132
 externas, 143-148
 habilitación y deshabilitación, 132-133
 onda cuadrada utilizando temporizador, ejemplo, 210
 organización de memoria cuando se usan, 138
 organización, 132-135
 procesamiento, 136
 puerto serial, 142-143
 salida de caracteres utilizando, 211-212
 secuencia de sondeo, 134-135
 sistema de detección de intrusos, ejemplo, 213-214
 temporizador, 139-141

- Intervalos sincronizados, 96-102
 INTR, línea, 244
 Invocación de comandos, 255-257
 IP. *Vea* Prioridad de interrupciones, registro de (IP)
 IR. *Vea* Instrucciones, registro de (IR)
 ISR. *Vea* Rutina de servicio de interrupciones (ISR)
- JBC bit, 403-404
 JC rel, 404
 JMP @A+DPTR, 404-405
 JMP nemónico genérico, 159
 JNB bit,rel, 405
 JNC rel, 406
 JNC, instrucción, 78
 JNZ, instrucción, 77, 406-407
 Juego de Tres en raya, proyecto para estudiantes, 358-363
 JZ rel, 407
 JZ, instrucción, 77
- Keil Software, Inc., 192-193, 204, 523
 KIM-1, 1
- Largo, direccionamiento, 58-59
 "Latch de lectura", 22
 LCALL, instrucción, 75-76, 407-408
 Lectura-modificación-escritura, operación, 22
 "Lee terminal", 22
 Lenguaje de comandos, 9
 LIB51, 251
 Limpia acumulador, 394
 Limpia bit (CLR bit), 394-395
 Limpio, 26
 LINEAENT, subrutina, 238-239
 Listo para enviar (CTS), 291
 LJMP. *Vea* Salto largo (LJMP)
 Llamada
 absoluta, instrucciones de (ACALL), 75-76, 385-386
 larga a subrutina, 407-408
 Llave
 de redondeo, 516
 secreta, 516
 Logical Systems Corp., 523
 Lógicas
 instrucciones, 59, 64-66
 operaciones. *Vea* Booleanas, instrucciones
 Lógicos, operadores, 161
 LSI. *Vea* Chips integrados a gran escala (LSI)
- MAA. *Vea* Accelerador de módulo aritmético (MAA)
 MAC. *Vea* Código de autenticación de mensaje (MAC)
 Macros, 183
 ejemplo, 184
 etiquetas locales, 186-187
 flujo de control, operaciones de, 187-188
 operaciones repetitivas, 187
 transferencia de parámetros, 185-186
 ventajas de utilizar, 184-185
 Manejador de interrupciones, 131
 Máquina
 ciclo de, 23-24
 lenguaje, 151
 lenguaje, programa en, 152
 Más significativo, bit (MSB), 92
 MAS, etiqueta, 237
 Maxim, 379
 Maxim Integrated Product, 378
 MC14499, interfaz a, 270-272
 MCS-151, 378-379
 MCS-251, 377-379
 MCS-48, 1
 MCS-51, 2, 377
 CA, características de, 445-446
 características, 440
 ciclo de escritura de memoria externa para datos, 448
 ciclo de lectura de memoria externa para datos, 448
 ciclo de lectura de memoria externa para programa, 447
 comparación de diferentes versiones, 17-18, 24
 conexiones, 442
 consideraciones de diseño, 443
 controlador de reloj externo, 450
 descripción de terminales, 441-443
 diagrama de bloques, 441
 EPROM, características de la, 451-453
 especificaciones máximas absolutas, 444
 hardware, arquitectura de, 17-43
 instrucciones, conjunto de, 49-85
 oscilador, características, 443
 puerto serial, sincronización, modo de desplazamiento de registros, 449
Vea también 8051; 8051, hoja de datos de
 Memoria
 compacta, 198
 de semiconductores, 5-6
 externa, 36-41
- flash, 377-378
 grande, 198
 mapa de, 173-175, 289
 modelos de, para el 8051, lenguaje en C del, 198
 para código (code), 166, 197
 organización
 bancos de registros, 27-28
 cuando se utilizan interrupciones, 138
 pequeña, mediana y grande, 198
 RAM de propósito general, 24-26
 semiconductores, 5-6
 tipos para el 8051, lenguaje C de, 197
- Memoria borrable y programable de sólo lectura (EPROM), 2, 20-21, 37-38, 172, 254-255
- Memoria de acceso aleatorio (RAM), 8
 características, 5-6
 de propósito general, 24-26
 definida, 3
 direccionable por bit, 26-27
 externa, 68-69
 interna, 66-68
 probando software, 253-254
- Memoria de acceso directo (DMA), 6
- Memoria de sólo lectura (ROM), 8
 características, 5-6
 definida, 3
- Memoria externa
 acceso a memoria externa para código, 36-41
 acceso a, 38-41
 decodificación de dirección, 40-41
 multiplexaje del bus de direcciones y del bus de datos, 36-37
 para código, acceso a, 37-38
 para datos (xdata), 166, 197, 201-202
 para datos paginada (pdata), 197, 202
 traslape de los espacios externos para código y para datos, 41
- Memoria interna para datos (data), 166, 197, 201-202
 direccionable por bit (bdata), 166, 197, 202
 directamente direccionable (data), 197, 201-202
 indirectamente direccionable (idata), 166, 197, 202
- Microcomputadora, sistema de, diagrama de bloques de, 3, 11
- Microcontrolador(es)
 alta velocidad, 378-379
 con ADCs y DACs, 378
 con memoria flash y NVRAM, 377-378

- definido(s), 1, 2
 ilustración, implementación de operación lógica simple, 14
 microprocesador *contra*, 10-12
 necesidades de mercado y tecnología, 12-13
 poder, tamaño y complejidad de, 2
 red, 379
 seguros, 379
 velocidad, 13-15
- Micrófono, 7
 Micromint, Inc., 523
 Microprocesador, 9
 conceptos nuevos, 12-13
contra microcontrolador, 10-12
- Microrratón, proyecto para estudiantes, 366-369
- Minicomputadora, 9-10
 Mnemónico, campo de, 155-156
 Mnemónicos, 151
 MOD, operador, 161
 Modo de registro de desplazamiento, 115
 Modular, programación, 173, 363
 Módulo
 definido, 152
 nombre de, 171
 MON51. *Vea* 8051, programa de monitoreo para (MON51)
 Monitoreo, programa de. *Vea* 8051, programa de monitoreo para (MON51)
- MOS, tecnología, 1
- Motor paso a paso
 configuración de rebobinados de estator del, 310
 interfaz a, 312, 348-350
 rotación a medio paso de, 311
 rotación de rotor de, 311
 software para una interfaz a, 313-315
- Motorola, 1
 MOV DPTR,#datos16, 413
 MOVX, instrucción, 38-39
 MPL. *Vea* Facilidad de procesamiento de macros (MPL)
- MSB. *Vea* Más significativo, bit (MSB)
- Múltiple, 416
- NAME, directiva, 171
 NAND, 13-15, 72
 National Institute of Standards & Technology (NIST), 515
 National Semiconductor Corp., 300-302
 Nibbles, 11
 NIPs. *Vea* Números de identificación personal (NIPs)
- NIST. *Vea* National Institute of Standards & Technology (NIST)
- Nivel de interrupción de segundo plano, 131-132
 No hay operación (NOP), 416-417
 Nohau Corp., 524
 NOR, operador, 72
 NOT, operador, 64, 72, 161
 Noveno bit de datos, 118
 Número, signo de (#), 54
 Números de identificación personal (NIPs), 372
 NVRAM, 377-378
- Objeto relocalizable
 archivo de, 155
 modelos de, 251
- Objeto, archivo, 155
 OCUPADO, 34
 OE. *Vea* Habilitación de salida (OE)
 Onda de pulso, ejemplo de generación de, 97, 205-206
 Operaciones repetitivas, 187
 Operando, campo de, 155-157
 OR exclusivo, 64
 para variables byte (XRL), 428-430
 realizando una operación sobre bits de, 12
 OR, 64, 71-72
 lógico para variables bit, 419
 lógico para variables byte, 417-418
 operación realizada de, 26
 realizando operaciones de, 12
 Orden, 219
 ORG, directiva, 164, 172
 Orientados a bit, operadores lógicos, 235-236
 Oscilador, 22
 ciclo de reloj de, 23-24
 entrada de, incorporado en el chip, 22-23
 OV. *Vea* Desbordamiento, bandera de (OV)
- P. *Vea* Paridad, bit de (P)
- Palabra clave, 219
 Palabras, 11
 dobles, 11
 reservadas, 235
 Palanca de mando/juegos, 7
 Pantalla de cristal líquido (LCD), interfaz a, 273-276, 325-327
 Paralela, interfaz, 10-11
 Paridad, bit de (P)
 agregar, 118-119
 definida, 32
- Pascal, 151
 Paso
 de traducción/conversión, 250-252
 dos, 153-155
 por paso, 249
 uno, 153
 PC, 8-9
 PCBs. *Vea* Tarjetas de circuito impreso (PCBs)
- PCON. *Vea* Poder, registro de control de (PCON)
- Pérdida de tiempo debido a las instrucciones, compensación por, 103-105
- Periférico
 dispositivo, definido, 3
 programable, interfaz a (PPI), 285.
Vea también 8255
- Philips, 379
 Pila, almacenando registros en, 239
 PKI. *Vea* Infraestructura de llave pública (PKI)
- Plan de llave, 516
 Pluma óptica, 7
 Poder
 conexión de, 22
 registro de control de (PCON), 35-36, 431-432
 Polialfabético, cifrado, 372
 PPI. *Vea* Periférico programable, interfaz a (PPI)
- Precedencia
 de operadores, 163, 236
 operación, 236
- Precisión múltiple, aritmética de, 64
 PRESENTA.LST, 177-182
 Principal, función, 245
 Prioridad de interrupciones, registro de (IP), 35, 133-134, 435-436
 Proceso
 de enmascarado de fábrica, 254-255
 predefinido (subrutina), 218
 en bloque, 218
 Producción de frecuencias exactas, 102-105
 Programa(s)
 bibliotecario, 251
 de compuerta lógica, diagrama de flujo para, 14-15
 de inicialización, 8-9
 definido(s), 5-6, 152
 herramienta, 9
 organización de, en 8051, estilo de programación en C para, 245
 tipos de, 8-9
Vea también Software

- PROM, programador, 378
 Prueba de bits, 73
 Prueba de caracteres, ejemplo, 228-230
 PSEN. *Vea* Habilitación de almacenamiento de programa (PSEN)
 PSW. *Vea* Estado de programa, palabra de (PSW)
 PUBLIC, directiva, 169-170
 Puerto, registro de, 34
 Puerto 0, 18
 Puerto 1, 18-19
 Puerto 2, 19
 Puerto 3, 19-21
 Puerto serial
 activación de bandera TI, 116
 búfer de datos (SBUF), 34
 carácter de entrada, subrutina, 209
 carácter de salida, subrutina, 209
 diagrama de bloques del, 112
 initialización, 123, 208
 interrupciones, 142-143
 modo de registro de desplazamiento, 115
 modos, 114
 registro, 34-35, 118-119
 registro de búfer (SBUF), 111-112
 registro de control (SCON), 35, 111, 113, 433-435
 sincronización, 116
 sincronización de recepción, 115
 sincronización de transmisión, 114
 Puerto serial, operación
 comunicación entre múltiples procesadores, 119-120
 full dúplex, comunicación serial, 117-118
 initialización y acceso a registros de puerto serial, 118-119
 modos, 113-117
 puerto serial, registro de búfer (SBUF), 112
 puerto serial, registro de control (SCON), 113
 puerto serial, velocidades en baudios, 120-127
 serial, comunicación, 111-112
 Puerto serial, velocidad en baudios
 “ciclo de espera”, 124
 carácter de entrada, subrutina, ejemplo, 125
 carácter de salida, subrutina, ejemplo, 123-124
 compensación por errores de redondeo, ejemplo, 127
 full dúplex, operación, ejemplo, 125-126
 initialización del puerto serial, ejemplo, 123
 predeterminado, 120-121
 utilizando temporizador 1 como reloj de velocidad en baudios, 121-127
 velocidad en baudios, resumen de, 122
 Punto de interrupción, 249
 PUSH, instrucción, 67
 RAM
 de propósito general, 24-26
 direccional por bit, 26-27
 externa, 68-69
 interna, 24, 66-68
 no-volátil (NVRAM), interfaz a, 277-282, 329-332
 ubicación de datos en, 243. *Vea también* Datos, segmento de *Vea* Memoria de acceso aleatorio (RAM)
 Ratón, 7
 RCA, 1
 READ, 7
 señal de control, 4
 Recepción de datos (RXD), 291
 Receptor/transmisor asíncrono universal (UART), 115-117
 Recupera de la pila (POR directa), 419-420
 Red de área de controlador (CAN), 379
 Red, microcontroladores de, 379
 Redondeo, 516
 “Referencias hacia delante”, 155
 Reg51.h, 195-196
 Registros con funciones especiales (SFR)
 8051, instrucción, 28
 acumulador, 28
 apuntador de datos (DPTR), 33-34
 apuntador de pila (SP), 33
 B, registro, 32
 estado de programa, palabra de (PSW), 29-32, 437-438
 habilitación de interrupciones, registro de (IE), 35, 435
 memoria, mapa de, 432
 poder, registro de control de (PCON), 35-36, 431-432
 prioridad de interrupciones, registro de (IP), 435-436
 puerto serial, registro de control de (SCON), 433-435
 puerto serial, registro de, 34-35
 puerto, registro de, 34
 temporizador, 88-89
 temporizador/contador, registro de control de (TCON), 433
 temporizador/contador 2, registro de control de (T2CON), 436-437
 tiempo, registro de, 34
 Registro(s), 4, 239
 de desplazamiento de 8 bits, 113-115
 direcccionamiento por, 50-52
 Regresa
 de interrupción, 421-422
 de subrutina, 420-421
 Reinicialización (RST), 22, 43
 Relacional
 expresión, 221
 operador, 162, 235-236
 Relational Memory Systems, Inc., 524
 Relativo, direcccionamiento, 55-57
 Relevadores, interfaz a, 306-310, 346-348
 Reloj/sincronización, ciclo de, 23-24
 REN. *Vea* Habilitación de recepción (REN)
 REPEAT/UNTIL, sentencia, 226, 237
 Reposo, modo de, 35
 Reservación de memoria, directiva (DS), 153
 Residente, 255
 Respuesta de usuario, ejemplo, 231-234
 Resta con préstamo (SUBB), 61-62, 425-426
 RET, instrucción, 76, 420-421
 RETI, instrucción, 76, 138-139, 421-422
 Retorno de carro (CR), 239-240
 RETURN, 240
 RI. *Vea* Interrupción de recepción, bandera de (RI)
 RL A, 422
 RL51, 173, 251
 RLC A, 422-423
 Robot, 366
 jugador de fútbol, 369-371
 ROM. *Vea* Memoria de sólo lectura (ROM)
 programable una vez (OTP), 377
 Rotor, 310-311
 RR A, 423
 RRC A, 423
 RS232 (IEA-232), interfaz serial, 294-296, 337-340
 RSEG. *Vea* Segmento relocalizable (RSEG)
 directiva, 165-166
 RST. *Vea* Reinicialización (RST)

- Rutina
de servicio de interrupciones (ISR), 76, 131, 137-139, 249
grande de servicio de interrupciones, 138-139
pequeña de servicio de interrupciones, 137-138
- RXD. *Vea Recepción de datos (RXD)*
- SAB80C51A, 378
- Salida de caracteres utilizando interrupciones, ejemplo, 142-143, 211-212
- Salta
si acumulador es cero, 407
si acumulador no es cero, 406-407
si bandera de acarreo está activada, 404
si bandera de acarreo no está activada, 406
si bit está activado y limpia bit, 403-404
si bit no está activado, 405
- Salto(s), 158-160, 238
absoluto, 73, 76, 158-159, 389
condicionales, 77-78, 238
corto (SJMP), 73, 424-425
de página (LF), 240
indirecto, 404-405
largo (LJMP), 73, 76, 158-159, 273, 408
relativo, 158-159
- Sangría, 219
- SBC. *Vea Computadora de una sola tarjeta (SBC)*
- SBC-51, 259
componentes y partes, 260
diseño esquemático para, 260-265
versión de tarjeta de circuito impreso del, 266
- Sbit, tipo de datos, 193-195
- SBUF. *Vea Puerto serial, registro de búfer para (SBUF)*
- Schottky TTL, 22
- Scientific Engineering Laboratories, Inc., 524
- SCON. *Vea Puerto serial, registro de control de (SCON)*
- SDK-85, 1
- SEGMENT, directiva, 165
- Segmento, definido, 152
- Segmento relocable (RSEG), 171, 176-183
- Selección
de banco de registros, bits de, 31
de segmento, directivas, 171-173
- Sensores, 8, 303-306, 344-346
- Sentencia, bloque de, 220
- Serial
comunicación, 111-112, 117-118
interfaz, 10-11
- SET, directiva, 166
- Seudocódigo, 217-218
ejemplos de subrutinas, 234-235
sintaxis sugerida para, 235-237
- Sfr, tipo de datos, 194-195
- SFR. *Vea Registros con funciones especiales (SFR)*
- Sfr16, tipo de datos, 194-195
- Siemens Corporation, 2, 378
- Signetics/Philips, 525
- Símbolo(s), 156. Vea también el índice para símbolos individuales
definición de, 165-166
especiales del ensamblador, 157
tabla de, 153, 173-175, 183
- Siemens Components, Inc., 525
- Simulación en software, 251-253
- Simulador, 251-253
- Sincronización
de interrupciones, 148-149
de intervalo, 87-88, 92-93
fuentes de, 92-93
puerto serial, 116
- Síncrono, formato, 111-112, 115
- Sintaxis, error de, 249
- Sinusoidal, tabla de onda, 297-300
- Sistema
computacional, definido, 3
controlado por interrupciones, 131, 249
de detección de intrusos, ejemplo, 145-148, 213-214
de elevador, proyecto para estudiantes, 355-358
de interrupciones incorporado, 10
de seguridad residencial, proyecto para estudiantes, 353-355
de semáforos peatonal, 307-310, 347-348
de una sola tarjeta, 524
- Sistema básico de entradas/salidas (BIOS), 8
- Sistema operativo, software de, 8-9
- SJMP. *Vea Salto corto (SJMP)*
- Software
asistido por computadora (CAD), 250
definido, 3, 5
diseñando, 249
edición y traducción/conversión, 249
hardware *contra*, 8-9
niveles, 9
- pruebas preliminares, 249
Vea también Ciclo de desarrollo; Programa
- Software Development Systems, Inc., 524
- Sondeo, secuencia de, 134-135
- SP. *Vea Apuntador de pila (SP)*
- Standard Microsystems, 379
- SUBB. *Vea Resta con préstamo (SUBB)*
- SubBytes, 516-517
- Subrutinas, 75-77
uso de registros dentro de, 239
uso de, 240-242
Vea también Interrupciones y el índice para tipos individuales de subrutinas
- Sufijo del nombre de archivo, 251
- Suma con acarreo (ADDC), 63, 387-388
- SUMA, subrutina, 222-224
- Supercomputadoras, 9-10
- T2CON. *Vea Temporizador/contador 2, registro de control de (T2CON)*
- Tablas
de búsqueda, 69-70
de salto, 74-75
- Tareas cotidianas, 8
- Tarjeta inteligente, 350, 371-374
- Tarjetas de circuito impreso (PCBs), 250
- TCON. *Vea Temporizador/contador, registro de control de (TCON)*
- Teclado numérico en hexadecimal, interfaz a, 265-267, 319-323
- Teclado, 7
numérico, software para interfaz a, 268-269. *Vea también Visualizador LED de 7 segmentos, interfaz a*
- Temporizador(es)
compartido, modo de, 92
definido(s), 87-88
ejemplo, 99-100, 205
inimando, deteniendo y controlando, 93-95
lectura “al tiempo de ejecución”, 96
modos, 34, 90-92
registro con funciones especiales de, 88-89
registro de, 34, 42, 95-96
registro de modo de (TMOD), 34, 89
- Temporizador, operación, 87-109
8052, temporizador 2 del, 105-106
descripción general, 87-89
fuentes de sincronización, 92-93
generación de velocidad en baudios, 106

- inicializar y acceder a registros de temporizador, 95-96
- iniciar, detener y controlar temporizadores, 93-95
- intervalos cortos, medianos y largos, 96-102
- modos de y bandera de desbordamiento (OV), 90-92
- producción de frecuencias exactas, 102-105
- registro de control de (TCON), 89
- registro de modo de (TMOD), 34, 89
- Temporizador/contador, registro de control de (TCON), 34, 89-90, 93-94, 433
- Temporizador/contador 2, registro de control de (T2CON), 42, 102
- Terminador de programa, 218
- Terminal de video (VDT), 7
- TF2. *Vea* Desbordamiento de temporizador, bandera de (TF2)
- TI. *Vea* Interrupción de transmisión, bandera de (TI)
- Tierra (GND), 291
- Tipos de instrucciones aritméticas, 59-64
- bifurcación de programa, instrucciones de, 73-78
- booleanas, 71-73
- lógicas, 59, 64-66
- transferencia de datos, instrucciones de, 59, 66-70
- Vea también el índice para los tipos individuales*
- TMOD. *Vea* Temporizador, registro de modo de (TMOD)
- Transferencia de parámetros, 185-186, 203-204
- externa, 414-416
- Transfiere byte de código o byte de constante, 413-414
- variable bit, 412-413
- variable byte, 409-412
- Transistor-transistor, lógica de (TTL), 15
- oscilador, 22-23
- Transitorio, 255
- Transmisión de datos (TXD), 291
- TTL. *Vea* Transistor-transistor, lógica de (TTL)
- Tubo de rayos catódicos (CRT), 7
- TXD. *Vea* Transmisión de datos (TXD)
- UART. *Vea* Receptor/transmisor asíncrono universal (UART)
- Ultravioleta (UV), borrador de EPROM, 378
- Unidad de decodificación de instrucciones y control, 4
- Unidad de procesamiento central (CPU), 3-5
- Universal Cross Assemblers, 524
- URDA, Inc., 525
- USB 1.1, 378
- USING, directiva, 164-165
- Valores de regreso (devolución de), 204
- Variable, definiciones de, 245
- VDT. *Vea* Terminal de video (VDT)
- Vector de interrupción, 136
- Visualizador LED de 7 segmentos, interfaz a, 267-273, 323-325
- Voltaje, convertir, a datos binarios, 8
- WHILE/DO, estructura, 221-222, 224-225, 237
- WORM, 7
- WR, línea, 244
- WRITE, 7
- X2444 RAM no volátil, 282
- interfaz a, 281
- portada de la hoja de datos, 280
- sincronización de instrucción de recuperación, 285
- software para interfaz a, 283-284
- XCH. *Vea* Intercambia acumulador con variable byte
- XCHD. *Vea* Intercambia dígito (XCHD)
- Xdata. *Vea* Memoria externa para datos (xdata)
- XICOR, Inc., 291
- XRL. *Vea* OR exclusivo lógico para variables byte (XRL)
- XTAL1, 22
- XTAL2, 22
- Zilog, 1
- Z-World, 525