



CENGAGE  
Learning®


Professional • Technical • Reference

# Microcontrollers

— S E C O N D E D I T I O N —

FROM ASSEMBLY LANGUAGE  
TO C USING THE PIC24 FAMILY

BRYAN A. JONES, ROBERT REESE,  
AND J.W. BRUCE



# **MICROCONTROLLERS, SECOND EDITION: FROM ASSEMBLY LANGUAGE TO C USING THE PIC24 FAMILY**

**BRYAN A. JONES  
ROBERT B. REESE  
J.W. BRUCE**

**Cengage Learning PTR**



---

Australia, Brazil, Japan, Korea, Mexico, Singapore, Spain, United Kingdom, United States

**Microcontrollers, Second Edition:  
From Assembly Language to C  
Using the PIC24 Family**  
**Bryan A. Jones, Robert B. Reese,  
and J.W. Bruce**

**Publisher and General Manager,  
Cengage Learning PTR:**  
Stacy L. Hiquet

**Associate Director of Marketing:**  
Sarah Panella

**Manager of Editorial Services:**  
Heather Talbot

**Senior Product Manager:**  
Mitzi Koontz

**Project and Copy Editor:**  
Kezia Endsley

**Interior Layout:**  
Shawn Morningstar

**Cover Designer:**  
Luke Fletcher

**Proofreader:**  
Kelly Talbot Editing Services

**Indexer:**  
Kelly Talbot Editing Services

© 2015 Cengage Learning PTR.

CENGAGE and CENGAGE LEARNING are registered trademarks of Cengage Learning, Inc., within the United States and certain other jurisdictions.

ALL RIGHTS RESERVED. No part of this work covered by the copyright herein may be reproduced, transmitted, stored, or used in any form or by any means graphic, electronic, or mechanical, including but not limited to photocopying, recording, scanning, digitizing, taping, Web distribution, information networks, or information storage and retrieval systems, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the publisher.

For product information and technology assistance, contact us at  
**Cengage Learning Customer & Sales Support, 1-800-354-9706.**

For permission to use material from this text or product,  
submit all requests online at [cengage.com/permissions](http://cengage.com/permissions).

Further permissions questions can be emailed to  
[permissionrequest@cengage.com](mailto:permissionrequest@cengage.com).

All trademarks are the property of their respective owners. Noted figures have been reprinted with permission of the copyright owner, Microchip Technology Inc. All rights reserved. No further reprints or reproduction may be made without Microchip Inc.'s prior written consent.

All images © Cengage Learning unless otherwise noted.

Library of Congress Control Number: 2014945697

ISBN-13: 978-1-305-07655-6

ISBN-10: 1-305-07655-9

eISBN-10: 1-305-07656-7

Cengage Learning PTR  
20 Channel Center Street  
Boston, MA 02210  
USA

Cengage Learning is a leading provider of customized learning solutions with office locations around the globe, including Singapore, the United Kingdom, Australia, Mexico, Brazil, and Japan. Locate your local office at:  
**[international.cengage.com/region](http://international.cengage.com/region)**.

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

For your lifelong learning solutions, visit [cengageptr.com](http://cengageptr.com).

Visit our corporate Web site at [cengage.com](http://cengage.com).

*RBR: To my wife (Donna) and sons (Bryan and Brandon)—thanks for putting up with me.*

*BAJ: To my beloved wife and to my Lord; soli Deo gloria.*

*JWB: To all of my teachers. Thank you.*

# ACKNOWLEDGMENTS

---

The authors would like to thank the following individuals for their assistance in preparing this book:

- ECE 4723/6723 and ECE 3724 students for their patience during the development of this text and the accompanying software libraries. That includes ECE 3724 TAs Hejia Pan, Ian Turnipseed, and Ryan Nazaretian for their assistance during this transition. Ryan also served as an ECE 4723/6723 TA.
- To the members of the Microchip Academic Program team at Microchip Technology Inc. for their support in using Microchip products in a higher-education environment.

# ABOUT THE AUTHORS

---

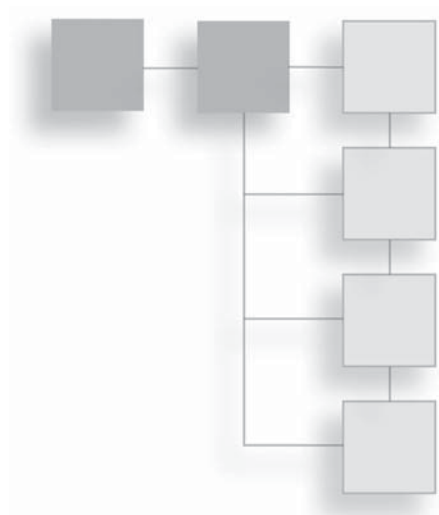
**BRYAN A. JONES** received B.S.E.E. and M.S. degrees in electrical engineering from Rice University, Houston, TX, in 1995 and 2002, respectively, and a Ph.D. in electrical engineering from Clemson University, Clemson, SC, in 2005. From 1996 to 2000, he was a Hardware Design Engineer for Compaq, specializing in board layout for high-availability RAID controllers. Since 2005, he has served in the Department of Electrical and Computer Engineering at Mississippi State University, Mississippi State, where he is an Associate Professor. His research interests include literate programming, engineering education, embedded systems, and visual guidance for micro air vehicles.

**ROBERT B. REESE** received a B.S. from Louisiana Tech University, Ruston, in 1979 and M.S. and Ph.D. degrees from Texas A&M University, College Station, in 1982 and 1985, respectively, all in electrical engineering. He served as a member of the technical staff of the Microelectronics and Computer Technology Corporation (MCC), Austin, TX, from 1985 to 1988. Since 1988, he has been with the Department of Electrical and Computer Engineering at Mississippi State University, Mississippi State, where he is an Associate Professor. Courses that he teaches include Microprocessors, VLSI systems, Digital System Design, and Senior Design. His research interests include self-timed digital systems and computer architecture.

**J.W. BRUCE** received a B.S.E. from the University of Alabama in Huntsville in 1991, an M.S.E.E. from the Georgia Institute of Technology in 1993, and a Ph.D. from the University of Nevada Las Vegas in 2000, all in electrical engineering. Dr. Bruce has served as a member of the technical staff at the Mevatec Corporation, providing engineering support to the Marshall Space Flight Center Microgravity Research Program. He also worked in the 3D Workstation Graphics Group at the Integraph Corporation, designing the world's first OpenGL graphics accelerator for the Windows operating system. Since 2000, Dr. Bruce has served in the Department of Electrical and Computer Engineering at Mississippi State University. Dr. Bruce has contributed to the research areas of data converter architecture design and embedded systems design. He has published more than 35 technical publications, several book chapters, and one book.

*This page intentionally left blank*

# CONTENTS



Introduction .....	.xv
--------------------	-----

## PART I

### DIGITAL LOGIC REVIEW AND COMPUTER ARCHITECTURE FUNDAMENTALS . . . 1

#### Chapter 1 Number System and Digital Logic Review . . . . . 3

Learning Objectives .....	3
Using Binary Data .....	4
Unsigned Number Conversion .....	7
Hex to Binary, Binary to Hex .....	7
Combinational Logic Functions .....	12
Combinational Building Blocks .....	19
The Multiplexer .....	19
The Adder .....	20
The Incrementer .....	21
The Shifter .....	22
Memory .....	22
Understanding Sequential Logic .....	23
The Clock Signal .....	24
The D Flip-Flop .....	25
Sequential Building Blocks .....	27
The Register .....	27
The Counter .....	28
The Shift Register .....	28
Encoding Character Data .....	30
Summary .....	31
Review Problems .....	32



<b>Chapter 2</b>	<b>The Stored Program Machine</b>	<b>33</b>
	Learning Objectives	33
	Problem Solving the Digital Way	34
	Finite State Machine Design	35
	Finite State Machine Implementation	37
	A Stored Program Machine	40
	Instruction Set Design and Assembly Language	40
	Hardware Design	44
	Modern Computers	48
	Summary	48
	Review Problems	48
<b>PART II</b>		
	<b>PIC24 <math>\mu</math>C ASSEMBLY LANGUAGE PROGRAMMING</b>	<b>51</b>
<b>Chapter 3</b>	<b>Introduction to the PIC24 Microcontroller Family</b>	<b>53</b>
	Learning Objectives	53
	Introduction to Microprocessors and Microcontrollers	54
	The PIC24 Microcontroller Family	55
	Program Memory Organization	57
	Data Memory Organization	58
	Arrangement of Multibyte Values in Data Memory	60
	Data Transfer Instructions and Addressing Modes	62
	Register Direct Addressing	62
	File Register Addressing	65
	WREG—The Default Working Register	67
	Immediate Addressing	69
	Indirect Addressing	70
	Instruction Set Regularity	72
	Basic Arithmetic and Control Instructions	74
	Three-Operand Addition/Subtraction	74
	Two-Operand Addition/Subtraction	76
	Increment, Decrement Instructions	77
	Program Control: goto	77
	A PIC24 Assembly Language Program	79
	C-to-PIC24 Assembly Language	80
	16-Bit (Word) Operations	88
	The Clock and Instruction Execution	91
	Summary	92
	Review Problems	92

<b>Chapter 4</b>	<b>Unsigned 8/16-Bit Arithmetic, Logical, and Conditional Operations</b> . . . . .	<b>95</b>
	Learning Objectives . . . . .	95
	Bitwise Logical Operations, Bit Operations . . . . .	96
	Using the Status Register . . . . .	100
	Using Shift and Rotate Operations . . . . .	102
	Using Mixed 8-Bit/16-Bit Operations, Compound Operations . . . . .	105
	Working Register Usage . . . . .	108
	LSB and MSB Operations . . . . .	108
	Conditional Execution Using Bit Tests . . . . .	109
	Unsigned Conditional Tests . . . . .	111
	Conditional Tests in C . . . . .	111
	Zero, Non-Zero Conditional Tests . . . . .	112
	Bit Tests . . . . .	115
	Equality, Inequality Conditional Tests . . . . .	116
	Conditional Tests for $>=$ , $>$ , $<$ , and $<=$ . . . . .	116
	Comparison and Unsigned Branch Instructions . . . . .	118
	Complex Conditional Expressions . . . . .	123
	Looping . . . . .	126
	Summary . . . . .	128
	Review Problems . . . . .	129
<b>Chapter 5</b>	<b>Extended Precision and Signed Data Operations</b> . . . . .	<b>133</b>
	Learning Objectives . . . . .	133
	Extended Precision Operations . . . . .	134
	32-Bit Assignment Operations . . . . .	134
	32-Bit Bitwise Logical Operations . . . . .	136
	32-Bit Addition/Subtraction . . . . .	137
	32-Bit Logical Shift Right/Shift Left Operations . . . . .	141
	Zero, Non-Zero Conditional Tests . . . . .	141
	Equality, Inequality . . . . .	144
	Comparisons of $>$ , $>=$ , $<$ , and $<=$ on Unsigned 32-Bit Operands . . . . .	145
	64-Bit Operations . . . . .	146
	Signed Number Representation . . . . .	147
	Signed Magnitude . . . . .	147
	One's Complement . . . . .	148
	Two's Complement . . . . .	149
	Sign Extension . . . . .	151
	Two's Complement Overflow . . . . .	152
	Operations on Signed Data . . . . .	153
	Shift Operations on Signed Data . . . . .	155
	Comparisons of $>$ , $>=$ , $<$ , and $<=$ on Signed Operands . . . . .	157

Sign Extension for Mixed Precision ..... 159  
 Branch Instruction Encoding .....161  
 Summary .....163  
 Review Problems .....164

**Chapter 6 Pointers and Subroutines .....167**

Learning Objectives .....167  
 PIC24 Indirect Addressing Modes .....168  
     Register Indirect with Signed Constant Offset ..... 170  
     What Instruction Forms Support Indirect Addressing? ..... 170  
     Instruction Stalls Due to Data Dependencies ..... 171  
 Using Subroutines .....172  
 The Stack and Call/Return, Push/Pop .....174  
     The Data Memory Stack ..... 175  
     Call/Return and the Data Memory Stack ..... 178  
     Stack Overflow/Underflow ..... 179  
 Implementing Subroutines in Assembly Language .....179  
     Static versus Dynamic Parameter Allocation ..... 180  
     Using Working Registers for Subroutine Parameters and Locals... 182  
     The Shadow Registers ..... 186  
 C Pointers and Arrays .....186  
     Implementation of C Pointer/Array Operations in Assembly ..... 190  
     A Subroutine That Manipulates 32-Bit Data ..... 193  
     C Strings ..... 195  
     The repeat Instruction ..... 196  
 Stack Frames for Function Parameters and Local Variables .....199  
 Program Space Visibility and Global Variable Initialization .....203  
 Summary .....206  
 Review Problems .....208

**Chapter 7 Advanced Assembly Language: Higher Math .....213**

Learning Objectives .....213  
 Multiplication .....214  
     64-Bit Multiplication ..... 218  
 Division .....220  
 Fixed-Point and Saturating Arithmetic .....225  
     Decimal to x.y Binary Format ..... 226  
     x.y Binary Format to Decimal Conversion ..... 226  
     Signed Fixed-Point ..... 227  
     0.n Fixed-Point Format and Saturating Operations ..... 228  
 The dsPIC® Microcontroller Family .....230

Floating-Point Number Representation .....	230
IEEE 754 Floating-Point Encoding .....	230
Floating-Point Operations .....	233
BCD Arithmetic .....	235
ASCII Data Conversion .....	237
Binary to ASCII-Hex .....	237
Binary to ASCII-Decimal .....	239
ASCII-Hex to Binary .....	240
ASCII-Decimal to Binary .....	242
Summary .....	242
Review Problems .....	243

## PART III

### PIC24 $\mu$ C INTERFACING USING THE C LANGUAGE .....245

#### Chapter 8 System Startup and Parallel Port I/O .....247

Learning Objectives .....	247
High-Level Languages versus Assembly Language .....	248
C Compilation for the PIC24 $\mu$ C .....	250
Special Function Registers and Bit References .....	251
PIC24 Compiler Runtime Code, Variable Qualifiers/Attributes .....	255
C Macros, Inline Functions .....	256
Conditional Compilation .....	256
PIC24 Startup Schematic .....	258
Startup Schematic: Power .....	260
Startup Schematic: Reset .....	261
Startup Schematic: PC Serial Communication Link .....	262
Startup Schematic: In-Circuit Serial Programming .....	262
Startup Schematic: Application Components .....	263
<i>ledflash.c</i> —The First C Program for PIC24 Startup .....	263
Clock Configuration .....	263
Flashing the LED .....	264
An Improved LED Flash Program .....	265
<i>echo.c</i> —Testing the Serial Link .....	267
<i>asm_echo.s</i> —Implementing Echo in Assembly .....	269
Datasheet Reading—A Critical Skill .....	270
Configuration Bits .....	272
Clock Generation .....	273
Power-On Reset Behavior and Reset Sources .....	274
Watchdog Timer, Sleep, Idle, and Doze .....	276
The <i>reset.c</i> Test Program .....	280

Parallel Port Operation .....284

    Tristate Drivers ..... 287

    Schmitt Trigger Input..... 288

    Open-Drain Output ..... 288

    Internal Weak Pull-Ups and Pull-Downs..... 289

    Digital versus Analog Inputs..... 290

    PIO Control Bits Summary..... 291

    PIO Configuration Macros/Functions ..... 291

LED/Switch I/O and State Machine Programming .....293

    State Machine I/O Programming ..... 295

    Extended State in a More Complex LED/Switch I/O Problem ..... 298

Interfacing to an LCD Module .....302

    3.3 V to 5 V Interfacing..... 303

    LCD Commands ..... 304

    LCD Code Example..... 306

The PIC24E versus the PIC24F and PIC24H Families .....310

Summary .....311

Review Problems .....312

**Chapter 9 Interrupts and a First Look at Timers .....317**

Learning Objectives .....317

Interrupt Basics .....318

PIC24  $\mu$ C Interrupt Details .....320

    Vector Table ..... 320

    Interrupt Priorities..... 322

    Traps ..... 322

    Interrupt Latency ..... 323

    ISR Overhead ..... 324

ISR Functions in C .....325

    The Default Interrupt ..... 325

    An Example ISR ..... 327

Change Notification Interrupts .....329

    Wake from Sleep/Idle ..... 330

    Using a Change Notification Interrupt to Measure Interrupt Latency ..... 330

INTx External Interrupts and Remappable Pins .....332

    Switch Inputs and Change Notification/INTx Interrupts ..... 336

Periodic Timer Interrupts .....336

    Timer Macros and Support Functions..... 339

    Square Wave Generation ..... 341

Interrupt-Driven LED/Switch I/O .....	343
Input Sampling .....	343
Change Notification with a Timer .....	346
Filtering Noisy Inputs .....	353
A Rotary Encoder Interface .....	355
A Keypad Interface .....	359
On Writing and Debugging ISRs .....	365
Summary .....	366
Review Problems .....	366
<b>Chapter 10 Asynchronous and Synchronous Serial I/O .....</b>	<b>371</b>
Learning Objectives .....	371
I/O Channel Basics .....	372
Synchronous, Asynchronous Serial I/O .....	374
Asynchronous Serial I/O Using NRZ Encoding .....	375
The PIC24 UART .....	380
UARTx Transmit Operation .....	383
UARTx Receive Operation .....	384
Baud Rate Configuration .....	384
Using the PIC24 UART with C .....	386
<stdio.h> Library Functions .....	389
Interrupt-Driven I/O with the PIC24 UART .....	390
Interrupt-Driven UART Receive .....	390
Interrupt-Driven UART Transmit .....	394
The RS-232 Standard .....	399
The Serial Peripheral Interface (SPI) .....	401
SPI Example: The MCP41xxx Digital Potentiometer .....	408
SPI Example: PIC24 $\mu$ C Master to DS1722 Thermometer .....	411
SPI Example: PIC24 $\mu$ C Master to PIC24 $\mu$ C Slave .....	414
The I2C Bus .....	419
I2C Physical Signaling .....	421
I2C Transactions .....	423
Library Functions for I2C Transactions .....	424
I2C on the PIC24 $\mu$ C .....	427
I2C Example: PIC24 $\mu$ C Master to DS1631 Thermometer .....	432
I2C Example: PIC24 $\mu$ C Master to 24LC515 Serial EEPROM .....	436
Ping-Pong Buffering for Interrupt-Driven Streaming Data .....	441
Summary .....	445
Review Problems .....	445

<b>Chapter 11</b>	<b>Data Conversion</b> .....	<b>449</b>
	Learning Objectives .....	449
	Data Conversion Basics .....	450
	Sensors and Transducers .....	450
	Analog-to-Digital Conversion .....	453
	Successive Approximation ADC .....	457
	Sample and Hold Amplifiers .....	459
	The PIC24 Analog-to-Digital Converter .....	460
	PIC24 ADC Configuration .....	463
	PIC24 ADC Operation: Manual .....	469
	PIC24 ADC Operation: Recap .....	474
	Digital-to-Analog Conversion .....	474
	Flash DACs .....	475
	R-2R Resistor Ladder Flash DAC .....	475
	External Digital-to-Analog Converter Examples .....	482
	DAC Example: The Maxim548A .....	483
	Summary .....	486
	Review Problems .....	486
<b>Chapter 12</b>	<b>Timers</b> .....	<b>489</b>
	Learning Objectives .....	489
	Pulse Width Measurement .....	490
	Using a 32-Bit Timer .....	492
	Pulse Width, Period Measurement Using Input Capture .....	496
	The Input Capture Module .....	497
	Pulse Width Measurement Using Input Capture .....	498
	Using Cascade Mode for 32-Bit Precision Input Capture .....	504
	Period Measurement Using Input Capture .....	504
	Application: Using Capture Mode for an Infrared Decoder .....	507
	The Output Compare Module .....	516
	Square Wave Generation .....	519
	Pulse Width Modulation .....	520
	A PWM Example .....	521
	PWM Application: DC Motor Speed Control and Servo Control .....	523
	DC Motor Speed Control .....	523
	Hobby Servo Control .....	524
	PWM Control of Multiple Servos .....	526
	A PWM DAC .....	530
	Time Keeping Using Timer1 and RTCC (PIC 24H/F Families) .....	532
	The Real-Time Clock Calendar Module .....	534
	Summary .....	538
	Review Problems .....	538

<b>Chapter 13</b>	<b>Advanced Hardware Topics</b>	<b>541</b>
	Learning Objectives	541
	Direct Memory Access	542
	Using the PIC24 $\mu$ C as an I2C Slave	549
	Bus Arbitration for the I2C Bus	553
	Reverse String Revisited	556
	The Controller Area Network (CAN)	558
	The PIC24 ECAN™ Module	562
	Using an ECAN RX FIFO	570
	Using an Extended Data Frame	571
	Run-Time Self-Programming	572
	A Sample Flash Application	576
	Summary	580
	Review Problems	581
<b>Chapter 14</b>	<b>Operating Systems for Embedded Systems</b>	<b>583</b>
	Learning Objectives	583
	Operating System Concepts	584
	Tasks	587
	Multitasking and Schedulers	588
	Inter-Task Coordination: Semaphores	592
	Inter-Task Coordination and Communication: Messaging	593
	OS Services	594
	Embedded Systems Operating System for the Microchip PIC24 $\mu$ C	596
	ESOS Overview	597
	User Tasks	598
	Your First ESOS Program	602
	ESOS Communication Services	604
	ESOS Timer Services	608
	ESOS Semaphore Services	611
	ESOS Messaging Services	615
	ESOS User Flags	619
	ESOS Child Tasks	621
	ESOS Interrupt Services	624
	Design: Adding an ESOS Service for I2C	628
	I2C Operations Under ESOS	629
	I2C Transactions Under ESOS	632
	Application Using the ESOS I2C Service and Semaphores	636
	Application Using the ESOS I2C Service and Messaging	638
	Summary	641
	Review Problems	642



**PART IV**

**APPENDIXES** .....643

**Appendix A PIC24 Architecture and Instruction Set Summary** .....645

**Appendix B Circuits 001** .....653

    Voltage, Current, and Resistance .....653

        Ohm’s Law .....654

        Resistors in Series.....655

        Resistors in Parallel .....656

        Polarization .....657

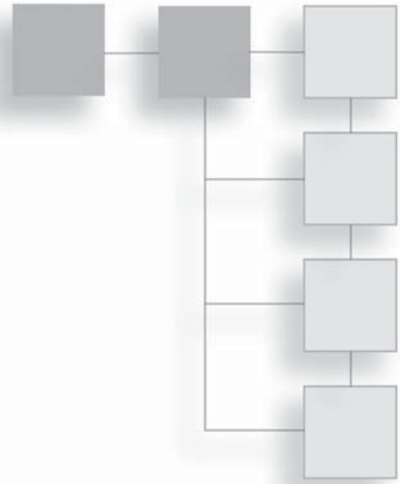
        Diodes.....657

        Capacitors .....658

**Appendix C Problem Solutions** .....661

**Appendix D References** .....689

**Index** .....695



# INTRODUCTION

This book and its accompanying website—[www.reesemicro.com](http://www.reesemicro.com)—are intended as an introduction to microprocessors ( $\mu$ Ps) and microcontrollers ( $\mu$ Cs) for the student or hobbyist. The book structure is as follows:

- **Chapter 1:** Review of digital logic concepts.
- **Chapter 2:** Computer architecture fundamentals.
- **Chapters 3 through 6:** Coverage of assembly language programming in a C language context using the PIC24 family.
- **Chapter 7:** Advanced assembly language programming structured around computer arithmetic topics.
- **Chapters 8 through 12:** Fundamental microcontroller interfacing topics such as parallel IO, asynchronous serial IO, synchronous serial IO (I<sup>2</sup>C and SPI), interrupt-driven IO, timers, analog-to-digital conversion, and digital-to-analog conversion.
- **Chapter 13:** Some advanced interfacing topics such as DMA, the ECAN standard, and slave/multi-master I<sup>2</sup>C operations.
- **Chapter 14:** An advanced chapter that covers the basics of real-time operating systems using a cooperative multitasking OS written by the authors. Topics include tasks, schedulers, scheduling algorithms, task synchronization and communication, semaphores, mailboxes, and queues.
- **Chapter 15:** Advanced techniques and examples for use in a senior capstone design course. This chapter is available online only at [www.reesemicro.com](http://www.reesemicro.com).

- **Appendix A:** A compact summary of the PIC24E/dsPIC33E instruction set.
- **Appendix B:** A hobbyist-level introduction to basic circuits. It covers the basic components (resistors, capacitors, and diodes) used in this book's schematics.
- **Appendix C:** Solutions to odd-numbered end-of-chapter problems.
- **Appendix D:** References.

## This Book's Development

At Mississippi State University, majors in Electrical Engineering (EE), Computer Engineering (CPE), Computer Science (CS), and Software Engineering (SE) take our first course in microprocessors. Previous to Spring 2002, this course emphasized X86 assembly language programming with the lab experience being 100 percent assembly language based and containing no hardware component. We found that students entering our senior design course, which has the expectation of something “real” being built, were unprepared for doing prototyping activities or for incorporating a microcontroller component into their designs. We did offer a course in microcontrollers, but it was an elective senior-level course and many students had not taken that course previous to senior design. In Spring 2002, the Computer Engineering Steering Committee reexamined our goals for the first course in microprocessors, and the approach for this book's predecessor (*From Assembly Language to C Using the PIC18Fxx2*) was developed. From Fall 2003 through Spring 2004, we used the Microchip PIC16 family, and then used the PIC18 family from Summer 2004 through Spring 2008. In late Fall 2007, the authors reexamined the course once again and decided to switch to the PIC24 family because of its rich instruction set architecture, 16-bit organization, and advanced on-chip peripherals. In 2013, significant advances in the field prompted the second edition, which focuses on the redesigned and improved PIC24E/dsPIC33E family of PIC24/dsPIC33 microprocessors.

## Using This Book in an Academic Environment

This book is intended for use as a first course in microcontrollers/microprocessors ( $\mu\text{C}/\mu\text{P}$ ) using the PIC24 family, with prerequisites of basic digital design and exposure to either C or C++ programming. The book begins with simple microprocessor architecture concepts, moves to assembly language programming in a C language context, and then covers fundamental hardware interfacing topics such as parallel IO, asynchronous serial IO, synchronous serial I/O (I<sup>2</sup>C and SPI), interrupt-driven IO, timers, analog-to-digital conversion, and digital-to-analog conversion. Programming topics are discussed using both assembly language and C, while hardware interfacing examples use C to keep code complexity low and improve clarity. The assembly language programming chapters emphasize the linkage between C language constructs and their assembly language equivalent so that students clearly understand the impact of C coding choices in terms of execution time and memory requirements. A textbook with an assembly-only focus creates students who are experts only in assembly language programming, with no understanding of high-level language programming

techniques and limited hardware exposure. Most embedded software is written in C for portability and complexity reasons, which argues favorably for reduced emphasis on assembly language and increased emphasis on C. Embedded system hardware complexity is steadily increasing, which means a first course in  $\mu\text{C}/\mu\text{P}$  that reduces assembly language coverage (but does not eliminate it) in favor of hands-on experience with fundamental interfacing allows students to begin at a higher level in an advanced course in embedded systems, the approach chosen for this textbook.

Hardware interface topics included in this book cover the fundamentals (parallel IO, serial IO, interrupts, timers, analog-to-digital conversion, digital-to-analog conversion) using devices that do not require extensive circuits knowledge because of the lack of a circuits course prerequisite. The microcontroller interfacing topics presented in this textbook are sufficient for providing a skill set that is extremely useful to a student in a senior design capstone course or in an advanced embedded system course.

Thus, a principal motivation for this book is that microcontroller knowledge has become essential for successful completion of senior capstone design courses. These capstone courses are receiving increased emphasis under ABET 2000 guidelines. This places increased pressure on Computer Engineering and Electrical Engineering programs to include significant exposure to embedded systems topics as early in the curriculum as possible. A second motivation for this book is that the ACM/IEEE Computer Engineering model curriculum recommends 17 hours of embedded system topics as part of the Computer Engineering curriculum core, which is easily satisfied by a course containing the topics in this book. A third motivating factor is the increased pressure on colleges and universities to reduce hours in engineering curriculums; this book shows how a single course can replace separate courses in assembly language programming and basic microprocessor interfacing.

The course sequence used at Mississippi State University that this book fits into is as follows:

- Basic digital design (Boolean algebra and combinational and sequential logic), which is required by EE, CPE, CS, and SE majors.
- Introduction to microprocessors (this book), which is required for EE, CPE, CS, and SE majors.
- Computer architecture as represented by the topic coverage of the Hennessy and Patterson textbook, *Computer Organization & Design: The Hardware/Software Interface*. This includes reinforcement of the assembly language programming taught in the microprocessor course via a general-purpose instruction set architecture (e.g., the MIPS), along with coverage of traditional high-performance computer architecture topics (pipelined CPU design, cache strategies, and parallel bus I/O). Required for CPE, CS, and SE majors.
- Advanced embedded systems covering topics such as real-time operating systems, Internet appliances, and advanced interfaces such as USB, CAN, Ethernet, and FireWire. Required for CPE majors.

Chapter 1 provides a broad review of digital logic fundamentals. Chapters 2 through 6 and 8 through 13 cover the core topics of assembly language programming and microcontroller interfacing. Chapters 7 and 14 have optional topics on advanced assembly language programming and the basics of real-time operating systems, which can be used to supplement the core material. The accompanying website provides a sequence of 11 laboratory experiments that comprise an off-the-shelf lab experience: one experiment on fundamental computer architecture topics, four experiments on PIC24 assembly language, and six hardware experiments. In addition, the website provides Chapter 15 of the textbook in an online form; this chapter demonstrates a set of techniques and projects that integrate and supplement material from the previous chapters.

The hardware labs cover all major subsystems on the PIC24  $\mu$ C: A/D, timers, asynchronous serial interface, SPI, and the I<sup>2</sup>C interface. The hardware experiments are based on a breadboard/parts kit approach where the students incrementally build a PIC24 system that includes a serial EEPROM, an external 8-bit DAC, and an asynchronous serial port via a USB-to-serial cable. A breadboard/parts kit approach is used instead of a preassembled printed circuit board (PCB) for several important reasons:

- When handed a preassembled PCB, students tend to view it as a monolithic element. A breadboard/parts kit approach forces students to view each part individually and read datasheets to understand how parts connect to each other.
- Hardware debugging and prototyping skills are developed during the painful process of bringing the system to life. These hard-won lessons prove useful later when students must do the same thing in a senior design context. This also provides students with the confidence that, having done it one time, they can do it again—this time outside of a fixed laboratory environment with guided instruction.
- A breadboard/parts kit approach gives the ultimate flexibility to modify experiments from semester to semester by simply changing a part or two; also, when the inevitable part failures occur, individual components are easily replaced.

In using this laboratory approach at Mississippi State University, the authors have seen a “Culture of Competence” develop in regard to microcontrollers and prototyping in general. All senior design projects now routinely include a microcontroller component (not necessarily Microchip-based). Students concentrate their efforts on design definition, development, and refinement instead of spending most of their time climbing the learning curve on prototyping and microcontroller usage.

There are more topics in this book than can be covered in a 16-week semester. In our introductory microprocessor course, we cover Chapters 1 through 6 for the assembly language coverage (about 6 weeks) and selected topics from Chapters 8 through 12 for the interfacing component. A course with more emphasis on assembly language may include Chapter 7 and fewer interfacing topics.

Our follow-on embedded systems course uses Chapters 8 through 14, with an emphasis on writing applications using the embedded operating system approach described in Chapter 14 and a more in-depth coverage of all interfacing topics. A first course in microcontrollers that contains no assembly language component may want to assign Chapters 1 through 7 as background reading and use Chapters 8 through 14 as the primary course material.

This book's *C* examples on hardware interfacing strive for code clarity first and optimization second. A prefix naming convention (`u8_`, `u16_`, `i32_`, `pu8_`, and so on) is used for all variables, and a robust set of macros and library functions have been developed to make access to the on-chip resources easier for those encountering microcontrollers for the first time. The library functions emphasize run-time error trapping and reporting as a way of shedding more light on malfunctioning applications. Please check the [www.reesemicro.com](http://www.reesemicro.com) website for updates to the library functions.

## For the Hobbyist

This book assumes very little background, and thus is appropriate for readers with widely varying experience levels. First, read Chapter 8 and visit the companion website at [www.reesemicro.com](http://www.reesemicro.com) to build and install the hardware and software PIC24 development environment. Next, peruse the example programs at this website and find the ones that interest you. Then, read the chapter that is referenced by the experiment for the necessary background. This textbook includes numerous examples complete with schematics and working code to operate a number of useful peripherals, including temperature sensors, LCD displays, and RC servo control, providing a good starting point for your designs.

## Final Thoughts

We hope readers have as much fun exploring the world of  $\mu$ Cs/ $\mu$ Ps and the PIC24 family as the authors had in creating this text. Because we know that  $\mu$ C/ $\mu$ P development does not sit still, let us all look forward to new learning experiences beyond this text.

Bryan A. Jones, Bob Reese, and J. W. Bruce  
Mississippi State University  
Starkville, Mississippi

*This page intentionally left blank*



**PART I**

**DIGITAL LOGIC  
REVIEW AND  
COMPUTER  
ARCHITECTURE  
FUNDAMENTALS**

**CHAPTER 1**

Number System and Digital Logic Review .....3

**CHAPTER 2**

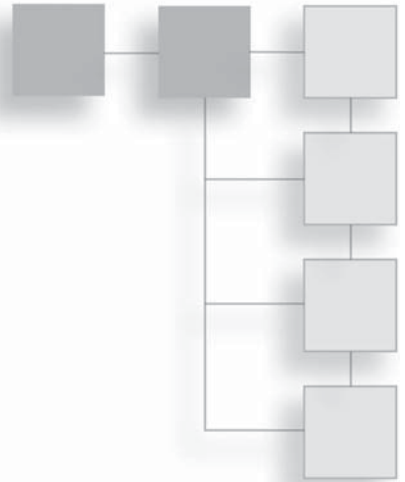
The Stored Program Machine .....33



*This page intentionally left blank*

# CHAPTER 1

## NUMBER SYSTEM AND DIGITAL LOGIC REVIEW



This chapter reviews number systems, Boolean algebra, logic gates, combinational logic gates, combinational building blocks, sequential storage elements, and sequential building blocks.

### Learning Objectives

After reading this chapter, you will be able to:

- Create a binary encoding for object classification.
- Convert unsigned decimal numbers to binary and hex representations and vice versa.
- Perform addition and subtraction on numbers in binary and hex representations.
- Identify NOT, OR, AND, NOR, NAND, and XOR logic functions and their symbols.
- Evaluate simple Boolean functions.
- Describe the operation of CMOS P and N transistors.
- Identify the CMOS transistor-level implementations of simple logic gates.
- Compute clock period, frequency, and duty cycle given appropriate parameters.
- Identify common combinational building blocks.
- Identify common sequential building blocks.
- Translate a character string into ASCII encoded data, and vice versa.

Binary number system representation and arithmetic is fundamental to all computer system operations. Basic logic gates, CMOS (Complementary Metal Oxide Semiconductor) transistor operation, and combinational/sequential building block knowledge will help your comprehension of the diagrams found in datasheets that describe microprocessor subsystem functionality. A solid understanding of these subjects ensures better understanding of the microprocessor topics that follow in later chapters.

## Using Binary Data

Binary logic, or digital logic, is the basis for all computer systems built today. *Binary* means two, and many concepts can be represented by two values: true/false, hot/cold, on/off, 1/0, to name a few. A single binary datum whose value is 1 or 0 is referred to as a *bit*. Groups of bits are used to represent concepts that have more than two values. For example, to represent the concepts hot/warm/cool/cold, two or more bits can be used as shown in Table 1.1.

**Table 1.1:** Digital Encoding Examples

Value	Encoding A	Encoding B	Encoding C
Cold	0 0	0 0	0 0 0 1
Cool	0 1	1 0	0 0 1 0
Warm	1 0	1 1	0 1 0 0
Hot	1 1	0 1	1 0 0 0

To encode  $n$  objects, the minimum number of bits required is  $k = \text{ceil}(\log_2 n)$ , where *ceil* is the ceiling function that takes the nearest integer greater than or equal to  $\log_2 n$ . For the four values in Table 1.1, the minimum number of bits required is  $\text{ceil}(\log_2(4)) = 2$ . Both encoding A and encoding B use the minimum number of bits, but differ in how codes are assigned to the values. Encoding B uses a special encoding scheme known as *Gray code*, in which adjacent table entries only differ by at most one bit position. Encoding C uses more than the minimum number of bits; this encoding scheme is known as *one-hot encoding*, as each code only has a single bit that is a 1 value.

Encoding A uses *binary counting order*, which means that the code progresses in numerical counting order if the code is interpreted as a *binary number* (base 2). In an unsigned binary number, each bit is weighted by a power of two. The rightmost bit, or *least significant bit* (LSb), has a weight of  $2^0$ , with each successive bit weight increasing by a power of two as you move from right to left. The leftmost bit, the *most significant bit* (MSb), has a weight of  $2^{n-1}$  for an  $n$ -bit binary number.

A lowercase “b” is purposefully used in the LSb and MSb acronyms since the reference is to a single bit; the use of an uppercase “B” in LSB and MSB acronyms is discussed in Chapter 3.

The formal term for a number’s base is *radix*. If  $r$  is the radix, then a binary number has  $r = 2$ , a decimal number has  $r = 10$ , and a hexadecimal number has  $r = 16$ . In general, each digit of a number of radix  $r$  can take on the values 0 through  $r - 1$ . The least significant digit (LSD) has a weight of  $r^0$ , with each successive digit increasing by a power of  $r$  as you move from right to left. The leftmost digit, the *most significant digit* (MSD), has weight of  $r^{n-1}$ , where  $n$  is the number of digits in the number. For hexadecimal (hex) numbers, letters A through F represent the digits 10 through 15, respectively.

Decimal, binary, and hexadecimal numbers are used extensively in this book. If the base of the number cannot be determined by context, a 0x is used as the radix identifier for hex numbers (i.e., 0x3A) and 0b for binary numbers (i.e., 0b01101000). No radix identifier is used for decimal numbers. Table 1.2 lists the binary and hex values for the decimal values 0 through 15. Note that 4 bits are required to encode these 16 values since  $2^4 = 16$ . The binary and hex values in Table 1.2 are given without radix identifiers. The \* symbol in Table 1.2 is a multiplication operation; other symbols used in this book for multiplication are  $\times (a \times b)$  and  $\cdot (a \cdot b)$  with the usage made clear by the context.

**Table 1.2:** Binary Encoding for Decimal Numbers 0–15

Decimal	Binary	Binary to Decimal	Hex	Hex to Decimal
0	0000	$0*2^3 + 0*2^2 + 0*2^1 + 0*2^0$	0	$0*16^0$
1	0001	$0*2^3 + 0*2^2 + 0*2^1 + 1*2^0$	1	$1*16^0$
2	0010	$0*2^3 + 0*2^2 + 1*2^1 + 0*2^0$	2	$2*16^0$
3	0011	$0*2^3 + 0*2^2 + 1*2^1 + 1*2^0$	3	$3*16^0$
4	0100	$0*2^3 + 1*2^2 + 0*2^1 + 0*2^0$	4	$4*16^0$
5	0101	$0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$	5	$5*16^0$
6	0110	$0*2^3 + 1*2^2 + 1*2^1 + 0*2^0$	6	$6*16^0$
7	0111	$0*2^3 + 1*2^2 + 1*2^1 + 1*2^0$	7	$7*16^0$
8	1000	$1*2^3 + 0*2^2 + 0*2^1 + 0*2^0$	8	$8*16^0$
9	1001	$1*2^3 + 0*2^2 + 0*2^1 + 1*2^0$	9	$9*16^0$
10	1010	$1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$	A	$10*16^0$
11	1011	$1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$	B	$11*16^0$

continues...

**Table 1.2:** Binary Encoding for Decimal Numbers 0–15 (continued)

Decimal	Binary	Binary to Decimal	Hex	Hex to Decimal
12	1100	$1*2^3 + 1*2^2 + 0*2^1 + 0*2^0$	C	$12*16^0$
13	1101	$1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$	D	$13*16^0$
14	1110	$1*2^3 + 1*2^2 + 1*2^1 + 0*2^0$	E	$14*16^0$
15	1111	$1*2^3 + 1*2^2 + 1*2^1 + 1*2^0$	F	$15*16^0$

A binary number of  $n$  bits can represent the unsigned decimal values of 0 to  $2^{n-1}$ . A common size for binary data is a group of eight bits, referred to as a *byte*. A byte can represent the unsigned decimal range of 0 to 255 (0x00 to 0xFF in hex). Groups of bytes are often used to represent larger numbers; this topic is explored in Chapter 5. Common powers of two are given in Table 1.3. Powers of two that are evenly divisible by  $2^{10}$  can be referred to by the suffixes Ki (kibi, kilobinary,  $2^{10}$ ), Mi (mebi, megabinary,  $2^{20}$ ), and Gi (gibi, gigabinary,  $2^{30}$ ). The notation of Ki, Mi, and Gi is adopted from IEEE Standard 1541-2002, which was created to avoid confusion with the suffixes k (kilo,  $10^3$ ), M (mega,  $10^6$ ), and G (giga,  $10^9$ ). Thus, the value of 4,096 can be written in the abbreviated form of 4 Ki ( $4 * 1 \text{ Ki} = 2^2 * 2^{10} = 2^{12} = 4096 = 4.096 \text{ k}$ ). However, be aware that the terminology of this IEEE standard is not in widespread use yet. The terms kBytes and MBytes are commonly used when referring to memory sizes and these mean the same as KiBytes and MiBytes.

**Table 1.3:** Common Powers of Two

Power	Decimal	Hex	Power	Decimal	Hex
$2^0$	1	0x1	(Ki), $2^{10}$	1024	0x400
$2^1$	2	0x2	$2^{11}$	2048	0x800
$2^2$	4	0x4	$2^{12}$	4096	0x1000
$2^3$	8	0x8	$2^{13}$	8192	0x2000
$2^4$	16	0x10	$2^{14}$	16384	0x4000
$2^5$	32	0x20	$2^{15}$	32768	0x8000
$2^6$	64	0x40	$2^{16}$	65536	0x10000
$2^7$	128	0x80	(Mi), $2^{20}$	1,048,576	0x100000
$2^8$	256	0x100	(Gi), $2^{30}$	1,073,741,824	0x40000000
$2^9$	512	0x200	$2^{32}$	4,294,967,296	0x100000000

**Sample Question:** What is the largest unsigned decimal number that can be represented using a binary number with 16 bits?

**Answer:** From Table 1.3, you can see that  $2^{16} = 65,536$ , so  $2^{16} - 1 = 65,535$ .

---

## Unsigned Number Conversion

To convert a number of any radix to decimal, simply multiply each digit by its corresponding weight and sum the result. The example that follows shows binary-to-decimal and hex-to-decimal conversion:

$$\begin{aligned} \text{(binary to decimal) } 0b0101\ 0010 &= 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= 0 + 64 + 0 + 16 + 0 + 0 + 2 + 0 = 82 \end{aligned}$$

$$\text{(hex to decimal) } \quad 0x52 = 5 \cdot 16^1 + 2 \cdot 16^0 = 80 + 2 = 82$$

To convert a decimal number to a different radix, perform successive division of the decimal number by the radix; at each step the remainder is a digit in the converted number, and the quotient is the starting value for the next step. The successive division ends when the quotient becomes less than the radix. The digits of the converted number are determined rightmost to leftmost, with the last quotient being the leftmost digit of the converted number. The following sample problem illustrates the successive division algorithm.

**Sample Question:** Convert 435 to hex.

**Answer:**

Step 1:  $435/16 = 27$ , remainder = 3 (rightmost digit).

Step 2:  $27/16 = 1$ , remainder = 11 = 0xB (next digit).

Step 3:  $1 < 16$ , so leftmost digit = 1.

Final answer:  $435 = 0x1B3$

To check your work, perform the reverse conversion:

$$0x1B3 = 1 \cdot 16^2 + 11 \cdot 16^1 + 3 \cdot 16^0 = 1 \cdot 256 + 11 \cdot 16 + 3 \cdot 1 = 256 + 176 + 3 = 435$$


---

## Hex to Binary, Binary to Hex

Hex can be viewed as a shorthand notation for binary. A quick method for performing binary-to-hex conversion is to convert each group of four binary digits (starting with the rightmost digit) to one

hex digit. If the last (leftmost) group of binary digits does not contain four bits, then pad with leading zeros to reach four digits. Converting hex to binary is the reverse procedure of replacing each hex digit with four binary digits. The easiest way to perform decimal-to-binary conversion is to first convert to hex then convert the hex number to binary. This involves fewer division operations and hence fewer chances for careless error. Similarly, binary-to-decimal conversion is best done by converting the binary number to a hex value, and then converting the hex number to decimal. The following examples illustrate binary-to-hex, hex-to-binary, and decimal-to-binary conversion.

**Sample Question:** Convert 0b010110 to hex.

**Answer:** Starting with the rightmost digit, form groups of four: 01 0110. The leftmost group has only two digits, so pad this group with zeros as: 0001 0110. Now convert each group of four digits to hex digits (see Table 1.2):

0b 0001 0110 = 0x16.

---

**Sample Question:** Convert 0xF3C to binary.

**Answer:** Replace each hex digit with its binary equivalent:

0xF3C = 0b 1111 0011 1100

---

**Sample Question:** Convert 243 to binary.

**Answer:** First, convert 243 to hex:

Step 1:  $243/16 = 15$ , remainder 3 (rightmost digit).

Step 2:  $15 < 16$ , so leftmost digit is 0xF (15). Hex result is 0xF3.

$243 = 0xF3 = 0b 1111 0011$  (final answer, in binary)

Check:  $0xF3 = 15 * 16 + 3 = 240 + 3 = 243$

---

### ***Binary and Hex Arithmetic***

Addition, subtraction, and shift operations are implemented in some form in most digital systems. The fundamentals of these operations are reviewed in this section and revisited in Chapters 3 and 4 when discussing basic computer operations.

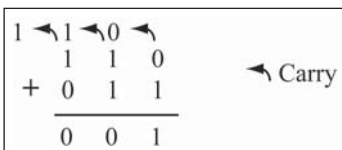
#### ***Binary and Hex Addition***

Adding two numbers,  $i + j$ , in any base is accomplished by starting with the rightmost digit and adding each digit of  $i$  to each digit of  $j$ , moving right to left. If the digit sum is less than the radix,

the result digit is the sum and a carry of 0 is used in the next digit addition. If the sum of the digits is greater than or equal to the radix, a carry of 1 is added to the next digit sum, and the result digit is computed by subtracting  $r$  from the digit sum. For binary addition, these rules can be stated as:

- $0 + 0 = 0$ , carry = 0
- $0 + 1 = 1$ , carry = 0
- $1 + 0 = 1$ , carry = 0
- $1 + 1 = 0$ , carry = 1

Figure 1.1 shows a digit-by-digit addition for the numbers  $0b110 + 0b011$ . Note that the result is  $0b001$  with a carry-out of the most significant digit of 1. A carry-out of the most significant digit indicates that the sum produced *unsigned overflow*; the result could not fit in the number of available digits. A carry-out of the most significant digit is an unsigned error indicator if the numbers represent unsigned integers. In this case, the sum  $0b110 + 0b011$  is  $6 + 3$  with the correct answer being 9. However, the largest unsigned integer that can be specified in three bits is  $2^3 - 1$ , or 7. The value of 9 is too large to be represented in three bits, and thus the result is incorrect from an arithmetic perspective, but is correct by the rules of binary addition. This is known as the *limited precision* problem; only increasing the number of bits used for binary encoding can increase the number range. You'll study this problem and the consequences of using more or fewer bits for number representation in later chapters.



**Figure 1.1**  
Binary addition example

**Sample Question:** Compute  $0x1A3 + 0x36F$ .

**Answer:** A digit-by-digit addition for the operation  $0x1A3 + 0x36F$  is as follows. The rightmost result digit is formed by adding:

$$0x3 (3) + 0xF (15) = 18$$

Note the digit sum is greater than 16, so a carry of 1 is produced and the rightmost result digit is computed by subtracting the radix, or:

$$18 - 16 = 2 = 0x2$$

The middle digit sum is then:

$$0xA (10) + 0x6 (6) + 1 (\text{carry}) = 17$$