

ARM Assembly for Embedded Applications

Daniel W. Lewis
Computer Engineering
Santa Clara University

Copyright © 2017-2023 by Daniel W. Lewis

All rights reserved. This book or any portion thereof may not be reproduced or used in any manner whatsoever without the express written permission of the author except for the use of brief quotations in a book review.

Printed in the United States of America

5th edition, 9th Printing, December 2023

Author contact information:

Daniel W. Lewis, Professor Emeritus
Computer Science and Engineering
Santa Clara University
500 El Camino Real
Santa Clara, California 95053

Book website: <http://www.engr.scu.edu/~dlewis/book3/>

CONTENTS

PREFACE	IX
CHAPTER 1 INTRODUCTION	1
1.1 WHY YOU SHOULD LEARN ASSEMBLY	2
1.2 WHEN ASSEMBLY IS NEEDED	4
1.3 THE MAJOR COMPONENTS OF A MICROCONTROLLER	7
1.4 WHAT ASSEMBLY LANGUAGE LOOKS LIKE	9
1.5 HOW ASSEMBLERS WORK	11
1.6 BYTES, HALFWORDS, WORDS, AND DOUBLE WORDS	12
1.7 INTEGER DATA TYPES USED IN THIS BOOK	13
1.8 IDENTIFIER CONVENTIONS USED IN THIS BOOK	14
CHAPTER 2 BINARY NUMBER SYSTEMS	17
2.1 CONVERTING UNSIGNED BINARY TO DECIMAL	18
2.2 CONVERTING UNSIGNED DECIMAL TO BINARY	19
2.2.1 Method #1a: Converting the Whole Part	20
2.2.2 Method #1b: Converting the Fractional Part	20
2.2.3 Method #2: Starting with the largest k such that $2^k \leq N$	22
2.2.4 Fixed Precision and Representation Error	23
2.2.5 Resolution, Precision, and Accuracy	24
2.3 SIGNED 2'S COMPLEMENT INTEGERS	25
2.3.1 Converting Decimal to Two's Complement	28
2.3.2 Converting Two's Complement to Decimal	29
2.4 EXTENDING REPRESENTATION WIDTH	30
2.5 HEX AS A SHORTHAND FOR BINARY	31
2.6 PROBLEMS	33
CHAPTER 3 WRITING FUNCTIONS IN ASSEMBLY	37
3.1 INSTRUCTION SEQUENCING	37
3.2 FUNCTION CALL AND RETURN	38
3.2.1 Using Identifiers to Access Variables	42
3.3 FUNCTION PARAMETERS	43
3.4 FUNCTION RETURN VALUE	44
3.4.1 Functions Returning 8 or 16-bit Values	46
3.5 REGISTER USAGE CONVENTIONS	48

3.6 STACK ALIGNMENT	50
3.7 FUNCTION CODING CONVENTIONS	51
3.8 PROBLEMS	52
CHAPTER 4 COPYING DATA.....	55
4.1 CONSTANT AND EXPRESSION SYNTAX	56
4.2 COPYING CONSTANTS INTO REGISTERS	56
4.3 COPYING DATA FROM MEMORY TO REGISTERS	59
4.3.1 Zero-Extending 8 and 16-bit Unsigned Integers	61
4.3.2 Zero-Extending 32-Bit Unsigned Values to 64-Bits	61
4.3.3 Sign-Extending 8 and 16-Bit 2's Complement Integers.....	62
4.3.4 Sign-Extending 32-Bit 2's Complement Integers to 64-Bits	63
4.4 COPYING DATA FROM ONE REGISTER TO ANOTHER.....	63
4.5 COPYING DATA FROM REGISTERS TO MEMORY	64
4.6 EXAMPLES OF COPYING DATA	66
4.7 ADDRESSING MODES	67
4.8 POINTERS AND ARRAYS	70
4.9 ADDRESS ALIGNMENT	74
4.9.1 The <i>.align</i> Assembler Directive	75
4.10 POINTERS AND STRUCTURES	76
4.11 COPYING A BLOCK OF DATA QUICKLY	78
4.12 PROBLEMS	80
CHAPTER 5 INTEGER ARITHMETIC	85
5.1 CONDITION FLAGS	85
5.2 ADDITION AND SUBTRACTION.....	86
5.2.1 Carries and Overflow	88
5.2.2 Multiple Precision	89
5.3 CODE SIZE AND THE <i>.N</i> SUFFIX	90
5.4 MULTIPLICATION	91
5.4.1 Signed versus Unsigned Products	91
5.4.2 Example: Efficient Polynomial Evaluation	94
5.4.3 64x64 Single-Length Products	95
5.4.4 Multiplication Overflow	96
5.5 DIVISION	97
5.5.1 Signed versus Unsigned	98
5.5.2 Division Overflow.....	98
5.5.3 Calculating a Remainder.....	99
5.6 PROBLEMS	100

CHAPTER 6	MAKING DECISIONS AND WRITING LOOPS.....	103
6.1	COMPARE AND TEST INSTRUCTIONS	103
6.2	CONDITIONAL BRANCH INSTRUCTIONS.....	104
6.2.1	<i>Understanding Condition Codes</i>	<i>108</i>
6.3	IF-THEN-ELSE SEQUENCES	109
6.4	COMPARING 64-BIT INTEGERS.....	110
6.5	IT BLOCKS.....	112
6.5.1	<i>IT Instructions and the AL Condition Code</i>	<i>116</i>
6.6	COMPOUND CONDITIONALS	117
6.6.1	<i>Decomposing Compound Boolean OR Conditionals</i>	<i>118</i>
6.6.2	<i>Decomposing Compound Boolean AND Conditionals.....</i>	<i>120</i>
6.6.3	<i>Using Vertically-Constrained Flowcharts.....</i>	<i>121</i>
6.6.4	<i>Range Check Optimization Trick</i>	<i>124</i>
6.7	WRITING LOOPS.....	125
6.7.1	<i>Branch Optimization in Loops.....</i>	<i>129</i>
6.7.2	<i>Optimizing Instruction Fetch.....</i>	<i>130</i>
6.8	PROBLEMS	131
CHAPTER 7	MANIPULATING BITS.....	135
7.1	BASIC SHIFT OPERATIONS	135
7.1.1	<i>Pre-Shifting Operands.....</i>	<i>138</i>
7.1.2	<i>Regular Shift Instructions.....</i>	<i>139</i>
7.1.3	<i>Shifting 64-bit Operands.....</i>	<i>140</i>
7.2	BITWISE OPERATORS IN C	142
7.3	BITWISE INSTRUCTIONS.....	144
7.4	TRICKS WITH BITWISE AND SHIFT INSTRUCTIONS	146
7.4.1	<i>Computation that Depends on the Sign of an Operand.....</i>	<i>146</i>
7.4.2	<i>An Efficient Absolute Value Function.....</i>	<i>147</i>
7.4.3	<i>Selection Without a Branch</i>	<i>148</i>
7.4.4	<i>Register Swap Using Exclusive-OR.....</i>	<i>149</i>
7.4.5	<i>Determining if x is a Power of 2</i>	<i>149</i>
7.5	BITFIELDS IN C.....	150
7.5.1	<i>Specifying substrings of 1's or 0's</i>	<i>152</i>
7.6	BITFIELD INSTRUCTIONS	153
7.7	MISCELLANEOUS BIT MANIPULATION INSTRUCTIONS	155
7.8	BIT-BANDING	156
7.9	PROBLEMS	160

CHAPTER 8	MULTIPLICATION AND DIVISION REVISITED	163
8.1	MULTIPLICATION BY A CONSTANT	164
8.1.1	<i>Constant Multiplication Using a Single Instruction</i>	164
8.1.2	<i>Constant Multiplication Using Multiple Instructions</i>	165
8.1.3	<i>Constant Multiplication Using Factoring</i>	167
8.2	DIVISION BY A POWER OF TWO	167
8.3	DIVISION BY AN ARBITRARY CONSTANT	170
8.4	REMAINDER WHEN DIVIDING BY 2^k	174
8.5	CALCULATING THE MODULUS WITH ARBITRARY DIVISOR	177
8.6	MEASURING EXECUTION TIME	178
8.7	PROBLEMS	180
CHAPTER 9	GETTING STARTED WITH FLOATING POINT	183
9.1	DATA TYPES FOR REAL NUMBERS	184
9.2	FLOATING-POINT REGISTERS	185
9.3	FUNCTION PARAMETERS AND RETURN VALUES	187
9.4	COPYING DATA	188
9.4.1	<i>Copying a Constant into a Register</i>	189
9.4.2	<i>Copying One Register to Another</i>	191
9.4.3	<i>Copying from Memory to a Register</i>	192
9.4.4	<i>Copying from a Register to Memory</i>	193
9.5	CONVERTING BETWEEN INTEGERS AND FLOATING POINT REALS ..	195
9.6	ARITHMETIC WITH REAL NUMBERS	197
9.7	COMPARING REAL NUMBERS	198
9.8	FPU INSTRUCTION TIMING	202
9.9	PROBLEMS	205
CHAPTER 10	FLOATING-POINT EMULATION	207
10.1	IEEE FLOATING-POINT FORMAT	208
10.1.1	<i>Floating-Point Anomalies</i>	210
10.2	CONSTANTS, STRUCTS AND MACROS	211
10.3	SUPPORT FUNCTIONS	214
10.3.1	<i>Extracting the Sign, Exponent and Significand</i>	214
10.3.2	<i>Shifting the Significand Left or Right</i>	216
10.3.3	<i>Normalization</i>	216
10.3.4	<i>Assembling the Components into a Float</i>	217
10.3.5	<i>Rounding to the Nearest Even</i>	218
10.3.6	<i>Special Case Handling</i>	218
10.4	FORMAT CONVERSIONS	219

10.4.1 Integer to Floating-Point	220
10.4.2 Floating-Point to Integer	221
10.5 ARITHMETIC OPERATIONS	222
10.5.1 Floating-Point Sum and Difference	223
10.5.2 Floating-Point Product	225
10.5.3 Floating-Point Quotient	226
10.6 PROBLEMS	228
CHAPTER 11 FIXED-POINT REALS	231
11.1 Q FORMAT AND THE IMAGINARY BINARY POINT	232
11.2 ADDITION AND SUBTRACTION OF FIXED-POINT REALS	234
11.3 MULTIPLICATION AND DIVISION OF FIXED-POINT REALS	235
11.4 FIXED-POINT USING A UNIVERSAL Q16.16 FORMAT	237
11.5 MULTIPLICATION OF Q32.32 FIXED-POINT REALS	240
11.5.1 The 128-bit Product of Two 64-bit Unsigned Integers	241
11.5.2 Converting a Product from Unsigned to 2's Complement	242
11.5.3 Example: Multiplying two Q4.4 Fixed-Point Reals	245
11.5.4 Implementation of Q32.32 Multiplication in Assembly	246
11.5.5 Multiplying a Q32 Value by an Integer Value	249
11.5.6 Multiplying a Q32 Value by a Fractional Value	251
11.6 DIVISION OF Q32.32 FIXED-POINT REALS	253
11.6.1 Dividing a Fractional Value by a Q32 Value	257
11.6.2 Dividing a Fixed-Point Value by an Integer Constant	258
11.6.3 Dividing a Fixed-Point Value by an Integer Variable	259
11.7 PROGRAMMING WITH Q32.32 REALS	260
11.8 PROBLEMS	263
CHAPTER 12 MULTIMEDIA PROCESSING	267
12.1 SIMD PROCESSING	268
12.1.1 SIMD Example: Color Negative of an Image	268
12.1.2 SIMD Add and Subtract Instructions	270
12.1.3 SIMD Example: Minimums and Maximums	271
12.1.4 SIMD Multiply Instructions	273
12.1.5 SIMD Example: Dot Product	274
12.1.6 SIMD Example: FIR Smoothing Filter	276
12.2 SATURATION PROCESSING	279
12.2.1 SIMD Saturating Add and Subtract Instructions	280
12.2.2 SIMD Saturating Example: Brightness Adjustment	280
12.3 MULTIMEDIA FLAGS	281
12.4 PROBLEMS	282

CHAPTER 13	COMPOSITE DATA TYPES.....	285
13.1	STRUCTURES AS CONTAINERS	286
13.2	BASIC DATA TYPES AS CONTAINERS.....	288
13.3	CASE STUDY – COMPLEX NUMBERS.....	288
13.3.1	<i>Structures as Containers for Complex Numbers</i>	<i>289</i>
13.3.2	<i>64-bit double as a Container for Complex Numbers.....</i>	<i>291</i>
13.3.3	<i>64-bit Integer as a Container for Complex Numbers</i>	<i>293</i>
13.4	CASE STUDY – RATIONAL NUMBERS	295
13.4.1	<i>64-bit Integer as a Container for Rational Numbers.....</i>	<i>297</i>
13.4.2	<i>32-bit Integer as a Container for Rational Numbers.....</i>	<i>299</i>
13.5	PROBLEMS	301
CHAPTER 14	INLINE CODE.....	303
14.1	INLINE FUNCTIONS	303
14.2	INLINE ASSEMBLY	304
14.2.1	<i>Input and Output Operands.....</i>	<i>306</i>
14.2.2	<i>Operand Constraints.....</i>	<i>308</i>
14.2.3	<i>Allowing Constants or Registers as Operands</i>	<i>309</i>
14.2.4	<i>Constraint Modifiers.....</i>	<i>309</i>
14.2.5	<i>The Clobbers Component.....</i>	<i>311</i>
14.2.6	<i>Working with 64-bit Operands</i>	<i>313</i>
14.2.7	<i>Statement Labels.....</i>	<i>314</i>
14.2.8	<i>The Optional Volatile Keyword</i>	<i>314</i>
14.2.9	<i>Using Artificial Dependencies</i>	<i>315</i>
14.3	COMBINING INLINE FUNCTIONS AND INLINE ASSEMBLY.....	317
14.4	INTRINSICS	319
14.5	PROBLEMS	321
CHAPTER 15	PROGRAMMING PERIPHERAL DEVICES	323
15.1	DEVICE DATA, CONTROL AND STATUS	324
15.2	MEMORY-MAPPED INPUT/OUTPUT	326
15.3	DATA RATES AND SYNCHRONIZATION	327
15.4	BLOCKING I/O: CRC32 PERIPHERAL.....	328
15.4.1	<i>The Basic CRC Algorithm.....</i>	<i>328</i>
15.4.2	<i>A Hardware-Assisted Implementation in C.....</i>	<i>330</i>
15.4.3	<i>A Hardware-Assisted Implementation in Assembly.....</i>	<i>332</i>
15.5	POLLED WAITING LOOP: RANDOM NUMBER GENERATOR.....	334
15.6	INTERRUPT-DRIVEN: TIMER TICK	337
15.7	DIRECT-MEMORY ACCESS: MEMORY-TO-MEMORY TRANSFER....	342

CONTENTS

vii

15.8 THE CPU CLOCK CYCLE COUNTER.....	345
15.9 PROBLEMS	346
A P P E N D I X A THE STM32F429I DISCOVERY BOARD	347
A P P E N D I X B USING THE GNU TOOLCHAIN.....	349
A P P E N D I X C BASIC SUPPORT LIBRARY.....	353
A P P E N D I X D GRAPHICS LIBRARY.....	357
A P P E N D I X E TOUCH SCREEN LIBRARY	359
I N D E X	361

PREFACE

This book is about programming in ARM assembly language within the context of embedded applications. The text presents assembly the way it is most commonly used in practice - to implement small, fast, or special-purpose routines called from a main program written in a high-level language such as C. The ARM Cortex-M4F processor was chosen as the target platform because it is the most prevalent processor for the vast number of embedded applications, including cell phones, tablet computers, disk drives, etc. This inexpensive processor also offers other advantages, including instructions for floating-point arithmetic, saturating arithmetic, and parallel (SIMD) processing.

The Cortex-M4F was introduced in 2010 as one of the Cortex-M family of processor designs developed by British company ARM Holdings. ARM also offers the high-performance Cortex-A and real-time Cortex-R processor families. ARM doesn't manufacture chips, but instead licenses their designs to other companies that incorporate the processor together with memory and peripherals into a single integrated circuit called a microcontroller (MCU). As of 2015, their licensees have shipped more than ten billion Cortex-M processors¹ deployed in over 3,000 embedded applications².

¹ Processors | Cortex-M – ARM (2017, June 21) Retrieved from <https://www.arm.com/products/processors/cortex-m>

² ARM's Cortex®-M and Cortex-R Embedded Processors (2017, June 21) Retrieved from

The book covers binary number systems, programming using regular integer arithmetic, arithmetic using both floating-point and fixed-point reals, multimedia processing, rational and complex arithmetic, and inline coding. In addition, there is extensive treatment of bit manipulation, shifting, extracting and inserting data that is stored in a packed format and material on programming peripheral devices.

INTENDED AUDIENCE

This book is intended for university and community college undergraduate majors, most likely at the sophomore level, who are pursuing a degree in Computer Science, Computer Engineering, or Electrical Engineering and who have completed an introductory programming sequence that includes data structures and programming in a modern high-level language such as C, C++, Java, or Python.

OBJECTIVE

The primary goal of this text is to get students engaged as early as possible. Rather than spending several weeks going over the architecture and detailed instruction set of the processor before having students write programs, the text gets them programming very early by introducing the C/Assembly interface (i.e., function call, parameter passing, return values and register usage conventions) before going into arithmetic, bit manipulation, making decisions, or writing loops. Example functions are presented starting in Chapter 3 and every chapter thereafter.

Writing functions instead of entire programs in assembly is not only representative of how assembly is most often used, but

also allows students to postpone the challenges of hardware initialization and how to setup and initialize a stack and heap.

Microcontrollers such as the one used in this text combine a processor, some amount of RAM for variables, some non-volatile Flash memory for code, and various peripheral devices - all in a single chip. With variables and instructions stored far apart in different parts of the address space, a variable usually cannot be accessed directly by an instruction because of range limitations on PC-relative references. Instead, a two-instruction sequence is required that first loads the address of the variable into a register followed by a second instruction that uses that register to access the variable. Functions called by C programs compiled with gcc don't have this issue because parameters and return value reside in registers, not in RAM. This considerably simplifies the student's task by allowing them to focus more on how to compute the result and less on how to access the data.

To facilitate testing functions written in assembly and called from a C program, the author has developed a gcc-compatible library for the STMicroelectronics STM32F429IDISCOVERY board that features a STM32F429ZI MCU based on the Cortex-M4F processor. The library is documented in the appendices, and is available on the textbook website³. It provides functions to initialize the MCU, to pause the program and wait for a push button to proceed, to measure execution time in CPU clock cycle times, and supports using the C library function `printf` to output text to the LCD display of the board. The library also provides additional functions to support drawing graphics and responding to input from the touch screen.

³ <http://www.engr.scu.edu/~dlewis/book3/>

The material is divided into chapters, some of which may be omitted. The first chapter is a brief introduction and the second could be skipped if students already know binary representation of integers. Chapters 3 through 7 and chapter 9 cover the basics of programming in assembly: function call and return, memory addressing modes, integer arithmetic, making decisions, bit manipulation and floating-point computation. Chapters 8, 10, 11 and 13 cover a number of elective topics for minimizing execution time, such as multiplication and division tricks, computation with fixed-point real numbers, parallel processing, and inline coding.

ACKNOWLEDGEMENTS

This book would not have been possible without the efforts of a number of people. Of them all, I owe a special thanks to the students and teaching assistants of COEN 20 at Santa Clara University whose experience in the course has helped to focus and fine-tune the organization and presentation of the material in the book.

I also owe a debt of gratitude to my colleagues Moe Amouzgar for enduring the unenviable task of teaching from my materials and who graciously suggested a number of improvements, to Darren Atkinson for his invaluable insights and expertise in C, assembly, and compilers, and to Andrea Vitali of STMicroelectronics for his thoughtful review of the chapter on floating-point emulation.

Finally, I would like to thank Santa Clara University for granting me the sabbatical leave that made it possible to find the time to complete the text and to Godfrey Mungal, Dean of Engineering, for his financial support during the preparation of the text.