ARM Assembly for Embedded Applications

Daniel W. Lewis
Computer Engineering
Santa Clara University

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

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 processors[1] deployed in over 3,000 embedded applications[2].

---

[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

---

https://www.arm.com/zh/files/event/2_2015_ARM_Embedded_Seminar_Ian_Johnson.pdf

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 website[3]. 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.