



THE MICROPROCESSOR

A Practical Introduction using
the **Arm Cortex-M0 Processor**

DARSHAK S. VASAVADA
S. K. SINHA

arm education



WILEY

The Microprocessor

The Microprocessor

A Practical Introduction using the Arm® Cortex®-M0 Processor

Darshak S. Vasavada

Robert Bosch Center for Cyber-Physical Systems,
Indian Institute of Science, Bangaluru, India

S. K. Sinha

Lab To Market Innovations Private Limited, FSID, IISc,
Bangaluru, India

This Work is a co-publication between Arm Education and John Wiley & Sons, Inc.

arm education

WILEY

Copyright © 2025 by John Wiley & Sons Inc. All rights reserved, including rights for text and data mining and training of artificial technologies or similar technologies.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permission>.

Any source code, models or other materials set out in this book should only be used for non-commercial, educational purposes (and/or subject to the terms of any license that is specified or otherwise provided by Arm). In no event shall purchasing this book be construed as granting a license to use any other Arm technology or know-how.

The manufacturer's authorized representative according to the EU General Product Safety Regulation is Wiley-VCH GmbH, Boschstr. 12, 69469 Weinheim, Germany, e-mail: Product_Safety@wiley.com.

Trademarks: Wiley and the Wiley logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries and may not be used without written permission. The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. All rights reserved. For more information about Arm's trademarks, please visit <https://www.arm.com/company/policies/trademarks>. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages. Further, readers should be aware that websites listed in this work may have changed or disappeared between when this work was written and when it is read. Neither the publisher nor authors shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our website at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Names: Vasavada, Darshak S., author. | Sinha, Sudhir Kumar, author.

Title: The microprocessor : a practical introduction using the Arm Cortex-M0 processor / Darshak S. Vasavada, S. K. Sinha.

Description: Hoboken, NJ : Wiley, 2025. | Includes bibliographical references and index.

Identifiers: LCCN 2024037432 (print) | LCCN 2024037433 (ebook) | ISBN 9781394245291 (hardback) | ISBN 9781394245314 (adobe pdf) | ISBN 9781394245307 (epub)

Subjects: LCSH: Microprocessors.

Classification: LCC TK7895.M5 V37 2025 (print) | LCC TK7895.M5 (ebook) | DDC 004.16—dc23/eng/20240923

LC record available at <https://lcn.loc.gov/2024037432>

LC ebook record available at <https://lcn.loc.gov/2024037433>

Cover Design: Wiley

Cover Image: © sankai/Getty Images

Set in 10.5/13pt Muli by Straive, Chennai, India

Contents

	List of Exercises	x
	Preface	xiii
	About This Book	xv
	How to Read This Book	xvi
	Acknowledgements	xviii
	About the Companion Website	xix
	Introduction	xxi
CHAPTER 1	Microprocessor System	1
	1.1 Introduction	2
	1.2 Processor	3
	1.3 Memory	5
	1.4 Memory Devices	7
	1.5 Bus	10
	1.6 IO Ports	14
	1.7 Reset, Clock and Power Management	16
	1.8 Overview of Arm Cortex-M0 Processor	17
	1.9 Summary	20
CHAPTER 2	Instruction Set Architecture	25
	2.1 Introduction	25
	2.2 Registers	27
	2.3 Instruction Set	28
	2.4 Structure of an Instruction	30
	2.5 Data-Processing Instructions	33
	2.6 Memory-Access Instructions	37
	2.7 Program-Control Instructions	43
	2.8 Summary	47
CHAPTER 3	Program Execution	49
	3.1 Introduction	49
	3.2 Program	50
	3.3 Inside the CPU	52
	3.4 Fetch Unit	55
	3.5 Decode Unit	57

3.6	Execution Unit	59
3.7	Instruction Execution	63
3.8	Processor Pipeline	66
3.9	Summary	68
CHAPTER 4	Assembly Programming	71
4.1	Statements	72
4.2	Labels	73
4.3	Sections	75
4.4	Text Section	77
4.5	Data Sections	84
4.6	Summary	90
CHAPTER 5	Arithmetic Operations	93
5.1	Arithmetic Instructions	94
5.2	Unsigned Integers	96
5.3	Signed Integers	99
5.4	APSR Flags	101
5.5	Condition Codes	106
5.6	Summary	110
CHAPTER 6	Bit-Level Operations	113
6.1	Boolean Instructions	114
6.2	Bit Manipulation	116
6.3	Shift Operations	119
6.4	Word-Length Extension	125
6.5	Byte Ordering Instructions	127
6.6	Summary	129
CHAPTER 7	Load and Store Operations	131
7.1	Introduction	131
7.2	Alignment	133
7.3	Endianness	135
7.4	Basic Load and Store Operations	140
7.5	Data Types	142
7.6	Offset Addressing	145
7.7	Summary	153
CHAPTER 8	Branch and Subroutine	155
8.1	Program-Control Instructions	155
8.2	Branch	156
8.3	Conditional Branch	158
8.4	Indirect Branch	162
8.5	Subroutines	166
8.6	Nested Subroutines	169
8.7	Summary	173

CHAPTER 9	Stack Operations	175
	9.1 Introduction	175
	9.2 What Is a Stack?	177
	9.3 Stack in Cortex-M0	178
	9.4 Stack Operations	180
	9.5 Creating a Stack	182
	9.6 Using the Stack	185
	9.7 Local Variables	190
	9.8 Summary	193
CHAPTER 10	Exceptions	195
	10.1 Introduction	196
	10.2 Exception Model	198
	10.3 Vector Table	200
	10.4 Reset	206
	10.5 Faults	209
	10.6 Exception Stack Frame	211
	10.7 Summary	215
CHAPTER 11	SysTick and Core Interrupts	217
	11.1 Introduction	218
	11.2 SysTick	218
	11.3 SysTick Programming Model	219
	11.4 Programming SysTick	222
	11.5 Using SysTick	224
	11.6 Polling with SysTick	226
	11.7 SysTick Interrupts	228
	11.8 Interrupt Masking	232
	11.9 Non-maskable Interrupt (NMI)	235
	11.10 Summary	237
CHAPTER 12	Introduction to C Programming	239
	12.1 Programming Languages	240
	12.2 Structure of a C Program	241
	12.3 Functions	244
	12.4 Data	247
	12.5 Header Files	250
	12.6 Overview of C Language	257
	12.7 Summary	259
CHAPTER 13	Basic Data Types	263
	13.1 Characters and Strings	263
	13.2 Integers	268
	13.3 Floating-Point Numbers	277
	13.4 Floating-Point Operations	280
	13.5 Type-Casting	284
	13.6 Summary	285

CHAPTER 14	Functions	289
14.1	Function Calls	290
14.2	Arguments	292
14.3	Local Variables	295
14.4	Conditional Execution	297
14.5	Selection	301
14.6	Loops	305
14.7	Summary	312
CHAPTER 15	Extended Data Types	315
15.1	Arrays	315
15.2	Structures	319
15.3	Pointers	323
15.4	Arrays and Pointers	327
15.5	Pointer to a Structure	330
15.6	Pointer Arithmetic	332
15.7	Uninitialized Pointers	334
15.8	Summary	339
CHAPTER 16	Compilation Process	341
16.1	Overview of the Compilation Process	342
16.2	Preprocessing	345
16.3	Compilation	348
16.4	Assembler	354
16.5	Linker	357
16.6	Executable Program	361
16.7	Summary	364
CHAPTER 17	Microcontroller	367
17.1	Introduction	368
17.2	Block Diagram	369
17.3	Pin Diagram	372
17.4	Reset, Clock and Power Management	373
17.5	Peripheral Interrupts	379
17.6	Peripheral Registers	382
17.7	Peripheral API	387
17.8	Summary	389
CHAPTER 18	IO Programming	393
18.1	IO Devices	394
18.2	GPIO	396
18.3	Timer	400
18.4	ADC	403
18.5	DAC	407
18.6	UART	410
18.7	Interrupts	414
18.8	Summary	419

CHAPTER 19	Microprocessor Applications	423
19.1	LED Brightness Controller	424
19.2	Ambient Light Sensor	427
19.3	Energy-Efficient Coding	430
19.4	Temperature Monitoring	433
19.5	Motor Speed Control	435
19.6	Summary	441
APPENDIX A	Programming Environment	443
A.1	Introduction	444
A.2	Keil MDK	446
A.3	Assembly Programming Setup	449
A.4	Writing and Building Assembly Programs	452
A.5	Debugging an Assembly Program	454
A.6	C Programming Setup	458
A.7	Writing and Building C Program	460
A.8	Debugging A C Program	461
A.9	Debugger	464
APPENDIX B	Advanced Topics	467
B.1	System-Control Instructions	467
B.2	OS Support	470
APPENDIX C	Startup Code	475
C.1	OS-Less System	475
C.2	Startup Code	478
C.3	Linker Script File	481
APPENDIX D	AM0 Header Files	483
D.1	Register Header File	483
D.2	AM0 Header File	484
	Glossary of Acronyms	487
	References	491
	Index	493

List of Exercises

Exercise 4.0	Create a new project	77
Exercise 4.1	My first assembly program	78
Exercise 4.2	Exploring the instruction codes	82
Exercise 4.3	Accessing data from the memory	87
Exercise 5.0	Create a new project	96
Exercise 5.1	Addition	96
Exercise 5.2	Multiplication	97
Exercise 5.3	Subtraction	99
Exercise 5.4	N and Z bits	102
Exercise 5.5	Carry and overflow	104
Exercise 5.6	Unsigned comparison	107
Exercise 5.7	Signed comparison	109
Exercise 6.0	Create a new project	115
Exercise 6.1	Boolean operations	115
Exercise 6.2	Setting, clearing and testing bits	118
Exercise 6.3	Logical shift	121
Exercise 6.4	Arithmetic shift	123
Exercise 6.5	Word-length extension	126
Exercise 7.0	Create a new project	133
Exercise 7.1	Data formats	133
Exercise 7.2	Endianness	138
Exercise 7.3	Basic load and store	141
Exercise 7.4	Sign extension	144
Exercise 7.5	Immediate offset	148
Exercise 7.6	Register offset	150
Exercise 8.0	Create a new project	157
Exercise 8.1	Jumping around	158
Exercise 8.2	<code>if-else</code> statements	160
Exercise 8.3	Loops	161
Exercise 8.4	Selection	163
Exercise 8.5	Subroutine	167

Exercise 8.6	Registers used in a subroutine	170
Exercise 9.0	Create a new project	183
Exercise 9.1	Where is the stack?	183
Exercise 9.2	Push and pop	186
Exercise 9.3	Registers on stack	187
Exercise 9.4	Nested subroutines	189
Exercise 9.5	Local variables on the stack	191
Exercise 10.1	Vector table	201
Exercise 10.2	Startup and main code	207
Exercise 10.3	HardFault	210
Exercise 10.4	Exception stack frame	212
Exercise 11.1	SysTick in action	223
Exercise 11.2	Polling with SysTick	226
Exercise 11.3	SysTick with interrupt	229
Exercise 11.4	Masking SysTick	233
Exercise 11.5	NMI	236
Exercise 12.0	Create a new project	243
Exercise 12.1	Function call	245
Exercise 12.2	Global variables	248
Exercise 12.3	Local variables	249
Exercise 12.4	Function declaration	250
Exercise 12.5	Header file	254
Exercise 13.0	Create a new project	264
Exercise 13.1	Characters	264
Exercise 13.2	Strings	266
Exercise 13.3	Integer data type	268
Exercise 13.4	Endianness and alignment	270
Exercise 13.5	Addition	272
Exercise 13.6	Sign and word-length	276
Exercise 13.7	Floating-point representation	279
Exercise 13.8	Floating-point addition	282
Exercise 13.9	Type-casting	284
Exercise 14.0	Create a new project	290
Exercise 14.1	Function call	291
Exercise 14.2	Arguments in registers	293
Exercise 14.3	Local variables	295
Exercise 14.4	Conditional execution	299

Exercise 14.5	<code>switch-case</code> statement	302
Exercise 14.6	<code>for</code> loop	306
Exercise 15.0	Create a new project	316
Exercise 15.1	Arrays	317
Exercise 15.2	Implementation of a structure	319
Exercise 15.3	Pointer as a fixed address	324
Exercise 15.4	Pointers	325
Exercise 15.5	Arrays and pointers	328
Exercise 15.6	Pointer to a structure	330
Exercise 15.7	Pointer arithmetic	333
Exercise 15.8	Accessing <code>NULL</code> pointer	335
Exercise 15.9	Accessing pointer with a non-address value	336
Exercise 16.0	Create a new project	345
Exercise 16.1	Preprocessing	346
Exercise 16.2	Declarations	352
Exercise 16.3	Definitions	352
Exercise 16.4	Translation	355
Exercise 16.5	Map file	362

Preface

Welcome to "*The Microprocessor: A Practical Introduction.*"

This is an entry-level book on microprocessor architecture, with a focus on Arm architecture for Cortex-M processors due to their widespread use in today's microcontrollers. The objective is to build a strong understanding of the core concepts that can serve as a foundation to study more complex processors and systems.

We have used Cortex-M0 processor as an example to explain the underlying concepts. We have chosen it for its simplicity; it allows us to cover all the fundamental concepts without overwhelming readers with complex details. Cortex-M0 has a small instruction set that includes all the key aspects of a typical RISC processor: pipelined execution, arithmetic and logic instructions, load-store and stack operations, and program flow control. It uses an intuitive exception model that remains consistent across all Cortex-M processors. Once learned with Cortex-M0, these same concepts will be applicable while working on more complex processors within Cortex-M family.

We believe that the best way to learn about microprocessor internals is through assembly language programming. Therefore, each chapter is organized as a set of topics, where each topic describes one basic concept, explains the relevant details from the reference manual, and then provides a programming exercise for clarity.

Subsequently, we introduce C programming and demonstrate how various constructs in C are implemented on the processor. The book also describes how a C program is compiled and debugged on a target hardware, providing insights into the internals of a compiler tool chain.

Lastly, we have defined an abstract microcontroller to illustrate an implementation of an Arm processor. We use this microcontroller to explain register-level peripheral programming and peripheral APIs, and demonstrate how real-world applications can be implemented on a microprocessor.

The assembly programming examples are based on the free version of Keil MDK, using the built-in simulator without requiring an additional hardware. At the same time, these programs can also run on any Cortex-M-based hardware, thanks to the compatibility of Cortex-M0 with all the Cortex-M processors.

Darshak S. Vasavada and S. K. Sinha
Bangalore, India
24 December 2024

About This Book

The book begins with an introduction that provides an overview of different categories of microprocessors and introduces Cortex-M processors.

Chapter 1 provides the hardware basics: the processor, memory and bus system, input/output (IO) ports and interrupts, and reset, clock and power management.

The following two chapters present the microprocessor architecture concepts. Chapter 2 describes the instruction-set architecture for Cortex-M0 processor, and chapter 3 explains how these instructions get executed inside a typical RISC processor.

Chapter 4 provides an introduction to assembly programming. Chapters 5–9 get into the details of various types of operations carried out by the processor: arithmetic and logic operations, load-store operations, program control and stack operations.

Chapter 10 introduces the exception model and explains the vector table, exception handling and reset processing. Chapter 11 describes SysTick timer and core interrupts.

Chapter 12 introduces C programming and describes the structure of a C program. Chapters 13, 14 and 15 demonstrate how various C elements get implemented and executed on the processor. Chapter 16 explains the compilation process, from source files to the binary program loaded in the processor's memory.

Chapter 17 introduces the microcontroller using an abstract implementation, simplified to explain the basic structure. Chapter 18 explains IO programming, and chapter 19 illustrates a few examples of real-world applications using the abstract microcontroller.

Appendix A describes Keil MDK development environment, used to carry out the programming exercises in this book.

Appendix B describes a few topics which are part of Cortex-M0 processor but beyond the scope of this book, included for completeness.

Appendix C lists the startup code used in the programming exercises in this book.

Appendix D lists the header files for the abstract microcontroller used in chapters 17–19.

How to Read This Book

If you are already familiar with microprocessor basics and are particularly interested in the architecture for Cortex-M0 processor, the introduction and chapters 2 and 10 will provide a comprehensive overview. The introduction provides a high-level overview of Arm Cortex processors, while chapter 2 covers Cortex-M0 instruction-set architecture and chapter 11 explains the core exception model. Additionally, chapter 11 describes SysTick timer and core interrupt handling.

If this is your first exposure to a microprocessor system, chapter 1 will introduce you to the necessary hardware basics. Following this, chapters 2, 3 and 10 explain the core architectural concepts. Chapters 4–9 will take you deeper into Cortex-M0 instruction set through hands-on programming, and chapters 10 and 11 will introduce you to system-level programming. Collectively, chapters 1–11 provide a strong foundation for the architectural concepts. Appendix A will get you started with assembly programming, and appendix B introduces advanced topics from Cortex-M0 processor for you to continue explorations beyond this book.

Chapters 12–16 are optional. These chapters introduce C programming constructs and show how they are implemented on Cortex-M0 processor. You may glance through them first, and then revisit them as you get deeper into embedded C programming. Appendix C explains the startup code, completing the overall picture of program execution.

Chapters 17–19 provide an introduction to the microcontroller. Here, we have defined an abstract microcontroller that explains the working principles without overwhelming you with details. If you have worked on a real microcontroller, these chapters may be an easy read for you, where you might get clarity on certain aspects you might have missed out earlier. On the other hand, if this is your first exposure to a microcontroller, the concepts in these chapters will guide you to program a real microcontroller. Appendix D provides header files for the abstract microcontroller, which can be used to implement peripheral API on a real hardware.

The companion website provides a template project along with the startup code to carry out the assembly programming exercises in this book.

The chapter map is shown in figure 1.

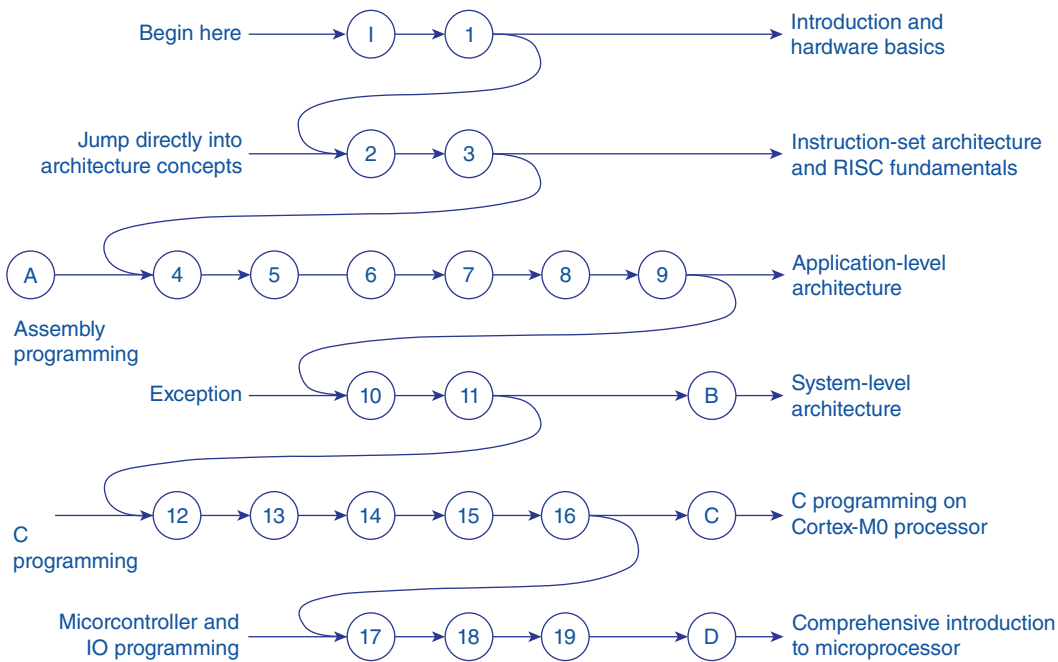


FIGURE 1 Chapter map.

We believe this structure will help you to navigate through the content effectively and select your engagement with the book according to your background and interests.

Acknowledgements

We express our heartfelt thanks to all our students and the many teachers who almost forced us to write this book, without whom this book would not have happened!

Our thanks to Professors T Matthew Jacob and YN Srikanth of IISc, Bangalore, for their thoughtful insights at the formative stage of the book. A big thank-you to Arun Hiregange for the working discussions and meticulous technical review.

Our sincere thanks to the Arm team: Joseph Yiu for the comprehensive technical review, Andrew Pickard for the code template, and Christopher Seidl and Andreas Barth for valuable inputs on Keil MDK. Special thanks to Liz Warman, Arm Education, for organizing such an accomplished team of reviewers. And thank you, our friends in Arm India, Apurva Varma and Neeraj Singh, for your guidance at the pre-publication stage of the book.

Working with Sandra Grayson at John Wiley Inc. has been an absolute pleasure. Thank you for your enthusiasm, commitment and unwavering support throughout the publication process. Journey from the first draft to the publication was arduous, but you made it so comfortable! Our thanks to the production team for their diligence in shaping this book through its refinements.

Finally, we owe our deepest gratitude to our students at the Indian Institute of Science. Their passion for learning and affection for us have been the source of energy, fuelling the journey from the conception of the book to its realization.

About the Companion Website

This book is accompanied by a companion website:

www.wiley.com/go/vasavada/Microprocessor



Introduction

Let us begin by asking:

What is a microprocessor?

Intel, which is credited with the first microprocessor (4004, in 1971), answers this question thus: "The microprocessor, also known as the central processing unit (CPU), is the brain of all computers."

This comparison of microprocessor with the human brain is quite apt. In a human mind–body system, the brain receives inputs from sensory organs (eyes, ears, skin), processes the information received and takes decisions. To the outside world, the decisions taken by our brain are reflected in what we speak, how we walk, etc.

We do not see our brain. And we do not see the microprocessor in a computer system – it is hidden inside the system and not directly accessible to the user. What we get to work with are the inputs and outputs of the system.

However, this comparison ends here! We have a very limited understanding of our brain. But we understand microprocessors very well. By the time we complete our journey, you will know much more about microprocessors than you do about your brain!

1 MICROPROCESSOR SYSTEM

Figure I.1 shows a diagram of a microprocessor system, simplified for the purpose of learning.

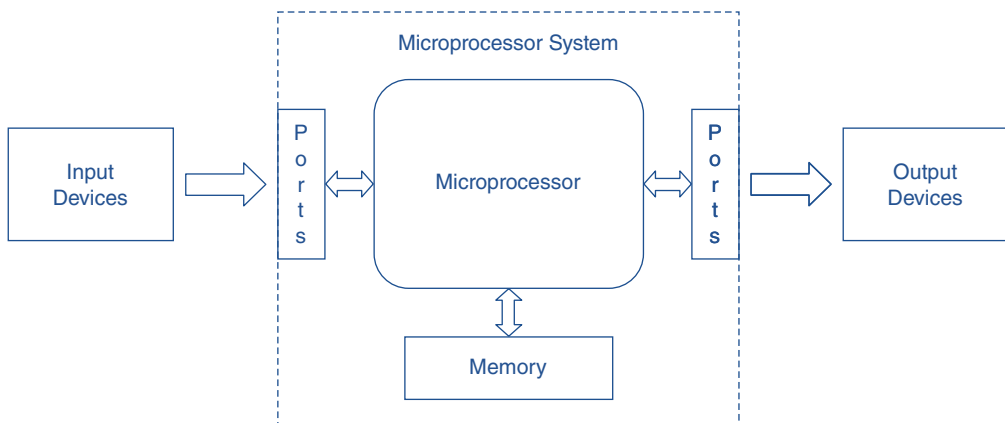


FIGURE I.1 A simplified microprocessor system

As shown in the figure, a microprocessor system receives inputs and produces outputs. An input device can be as simple as a push button, or very complex as a video camera. Similarly, an output device can be a simple two-lead LED, or a far more complex LCD screen. Processor may also interact with electro-mechanical devices, for example, motors in a robotic system. Such systems may involve inputs from a variety of sensors: distance, speed, acceleration, etc. Many systems use communication devices such as Universal Serial Bus (USB) or Ethernet for exchanging information with the external world. To interact with an input/output (IO) device, a microprocessor uses an IO port, which acts as a bridge between the processor and an IO device.

A microprocessor system processes the inputs and generates the outputs according to a program stored in the memory. A program could be an algorithm, a set of equations, or logical statements, typically written in a high-level language such as C or C++. Since the microprocessor can 'understand' only binary language, we convert a C/C++ program into binary machine codes using a software tool called a compiler. Just as our brain uses memory cells to remember various algorithms (for example, how to speak) and related data (various words that we speak), the microprocessor system uses memory devices to store the program instructions and the associated data.

Apart from being used in desktop and mobile computers, microprocessors are used in a large number of systems where they are embedded deep within the system. Examples of embedded applications are numerous, such as playing music or displaying pictures, controlling air-fuel mixture in a car's engine, managing the autopilot mode of an aeroplane, processing a variety of biomedical signals such as in an ECG or a blood-pressure monitor, processing camera information to drive motors in a robot, processing pressure and temperature in an industrial control system, processing enormous amounts of data on the servers – the list is endless.

Depending on the applications in which they are used, microprocessors vary in their capabilities. For example, a microprocessor that controls mixing air and fuel in a car engine has to be a lot more powerful than a microprocessor in a digital thermometer, but at the same time, it does not require capability to process audio, video and image data like the ones in our mobile phones do.

To understand this spectrum of microprocessors, ranging from a tiny microcontroller inside a remote control to the one powering a cloud server, let us divide them into three broad categories:

1. Microcontroller processors
2. Real-time processors
3. Application processors

Let us discuss each briefly.

2 MICROCONTROLLER PROCESSORS

Microcontrollers deploy the simplest form of microprocessors. They are used in many everyday devices such as remote controllers, digital thermometers, wearable devices and home appliances.

2.1 Hardware

Microcontrollers are designed for compact and power-efficient applications. They integrate a processor, memory and IO interfaces on a single chip. Microcontrollers are implemented as Systems-on-Chip (SoCs), reducing the overall system complexity and cost, and, at the same time, improving hardware reliability. Their small size and low power consumption make them ideal for applications where space and energy are at a premium.

2.2 Software

Microcontrollers are typically used in dedicated systems that perform specific tasks. Simple systems carry out repetitive activities running in a loop. Often known as 'bare-metal' systems, these are standalone systems without an operating system (OS). More complex systems deploy a real-time operating system (RTOS) to implement multitasking applications.

Power-sensitive systems go into a sleep state after completing their task, exploiting power-saving modes available in microcontrollers, and wake up again when new inputs are available.

2.3 Applications

Microcontroller processors are characterized by low software complexity, high power efficiency and small silicon area on the chip. They are tiny little brains that run numerous devices in automotive, industrial, medical, building automation and many more segments. Let us take a few examples:

Tens of microcontrollers are used inside a car, to carry out various tasks, such as checking seat-belt status, measuring tire pressure, controlling door lock, winding up window glass etc.

Microcontrollers are used extensively in home appliances. For example, microwave ovens use microcontrollers for on/off and timing control. Washing machines use them to control the speed and timings of their motors and operate water inlet and outlet valves.

Microcontrollers also proliferate medical equipment such as digital thermometers, blood-pressure meters, pulse-oximeters and so on, measuring various body parameters and displaying them on a small display.

Finally, microcontrollers play a major role in enabling the last leg of Internet of Things (IoT) systems. They provide the interface between the digital world and the physical sensors and actuators, enabling real-time data collection, processing and control in a variety of interconnected devices.

3 REAL-TIME PROCESSORS

Moving up the complexity scale, next in the series are the real-time processors. They are used in time-critical systems, where a significant amount of data needs to be processed within a specific deadline, such as an aircraft landing system or performing a robotic surgery. Often characterised as the systems where 'a late answer is a wrong answer', real-time processors are critically important in automotive, industrial, networking and many other systems.

3.1 Hardware

Real-time data-processing systems require significantly more computation power to process large volumes of data within a specific time frame. Hence, these processors run at a higher clock frequency compared to the microcontroller processors.

Real-time processors are accompanied by on-chip memories to store time-critical data and code. However, the on-chip memories are limited in size, and hence, such systems typically need a storage device (such as an SD card or a solid-state drive) to store the program, and a large external memory (RAM) to run the program. Since external memories are slow in speed, many systems also deploy on-chip cache memory that stores frequently accessed code and data to improve system performance.

3.2 Software

A real-time application is implemented as multiple threads running concurrently. Such systems use an RTOS that schedules threads according to their priorities to meet the deadlines. An RTOS is a very lightweight OS that provides multi-tasking capability to a system. Since it has little overheads, an RTOS is significantly more responsive to external inputs compared to a desktop OS, and hence, well-suited to meet real-time requirements.

Such systems often use a boot-loader code that loads the program from the storage device to the memory. Bootloaders often have the capability to upgrade the program for bug fixes and feature enhancements.

3.3 Applications

Real-time processors are used where time is of essence. These processors are designed to perform tasks within very strict time constraints, often requiring precision down to nanoseconds. Here are some examples of systems implemented using a real-time processor.

A power-train control module, also known as an engine control unit (ECU), processes signals from numerous sensors, including those measuring air and fuel intake, oxygen levels in the exhaust, and various shaft positions in the engine. It uses this data to control air and fuel injection and ignition timings,

ensuring precise operations that result in high fuel efficiency, even at high vehicle speeds.

A wireless modem in a mobile phone converts digital data into radio signals for transmission and vice versa at very high data rates. It runs sophisticated algorithms to improve reliability in the presence of interference and noise. Since the same medium is used by multiple devices for communication, these activities must be carried out with a very high degree of precision.

A solid-state drive (SSD) controller provides an interface between the processor and the storage device. It implements error correction, wear-levelling and garbage collection algorithms to improve the lifespan and efficiency of the storage device. These operations are carried out at a very high speed, allowing us to enjoy watching a video stored in an SSD without a glitch!

☛ **Microcontrollers vs Real-Time Processors**

Please note that while real-time processors are specifically designed to build high-speed, time-critical systems, many real-time systems are also implemented using microcontrollers. Audio equipment, ECG monitor, precision cutting machines are some of the examples that can be implemented using microcontrollers.

The key difference between microcontrollers and real-time processors lies in their data processing capabilities. Real-time processors are required when we need to process large amount of data within a very short time frame. In such scenarios, the processing power and the small size of the on-chip memory become limiting factors for microcontrollers.

With the advancement of technology, microcontrollers run at higher clock speeds and pack more on-chip memory. Hence, they are being increasingly used for applications in the real-time world, blurring the line between microcontrollers and real-time processors.

4 APPLICATION PROCESSORS

At the high end of the spectrum are application processors, the powerhouses behind computing devices like smartphones, tablets and cloud servers. Unlike the microcontrollers or real-time processors that run dedicated applications, application processors execute multiple applications using a general-purpose OS, which requires processing power, combined with the ability to manage large amounts of memory.

4.1 Hardware

Application processors use one or more powerful processor cores operating at a few GHz clock speed. These processors are accompanied by special-purpose

coprocessors for high-speed computations required in graphics, multimedia processing and machine-learning algorithms.

Application processors typically use several GBs of RAM, shared between multiple applications. They contain a memory management unit (MMU) that allows an OS to implement virtual memory to share the memory between multiple applications. An MMU is also a key component in providing security, and protects the OS and applications from unauthorized access either by a malfunctioning application or attacks from harmful programs.

Lastly, application processors are accompanied by high-speed peripheral ports for networking, storage and high data-rate IO devices such as a camera or a display.

4.2 Software Configuration

Application processors are designed to run operating systems that can run multiple application programs. Some of these processors also support hardware virtualization, allowing them to run multiple operating systems simultaneously on the same hardware.

An OS is a complex piece of software, typically spanning millions of lines of code that manages various resources in the system. At the heart of the OS is a scheduler that schedules different applications in a way that they appear to be running simultaneously to us. OS also includes a memory management subsystem to share memory between multiple applications, and file subsystems to efficiently store applications and related data on storage devices. Not directly visible to us are various low-level device drivers that communicate with networking and IO devices.

The system contains a large number of programs stored in the file system. Some of these are system programs that run continuously, while others run as required by the user. Many of such systems have the facility to download new applications and update them over the network on an ongoing basis.

4.3 Applications

Application processors are at the forefront, powering the cutting-edge devices such as mobile phones, personal computers and servers. With Linux kernel running on these processors, they are used in a vast number of embedded applications as well.

In consumer electronics, these processors enable multimedia functionalities in set-top boxes and in-car infotainment systems, supporting high-definition media playback, efficient content management and advanced user interfaces.

For networking devices like routers and firewalls, application processors perform complex routing and implement advanced security protocols.

In healthcare, application processors are used in patient monitoring systems and medical imaging devices to process biomedical signals and high-resolution images.

Application processors form the backbone of IoT systems; they process large volumes of data, run sophisticated algorithms and take complex decisions.

Many applications, such as kiosks or automatic teller machines (ATMs), run on application processors as they provide a desktop or a tablet-like environment for user interactions, even though the application themselves may not require high data-processing capabilities.

These are a few examples to give you an idea. The actual usage of application processors in embedded applications is vast and diverse, likely to explode with the use of AI and machine learning in years to come.

☛ Are application processors suited for real-time applications?

Traditionally, real-time applications have been designed with a stringent focus on guaranteeing response times and never missing deadlines. Use of a general-purpose OS for real-time systems was unacceptable – they were considered too ‘bulky’ as they could not respond to deadlines fast enough.

However, application processors are becoming more powerful, and hence, they are able to respond to the deadlines much better, even while running a general-purpose OS. At the same time, modern real-time systems require sophisticated user interfaces (including cameras, displays and speech recognition) and network connectivity for data analysis and running AI/ML algorithms. Hence, many software architects find it convenient to use an OS that provides all these features, and additionally, allows easy application updates, resulting in constantly evolving software.

It is important to note, however, that general-purpose OS is not designed to meet real-time requirements, and can miss deadlines once in a while. To address these, some systems simply use more processor cores, faster clock and more memory to reduce the chances of missing deadlines. They also implement acceptable workarounds in case the system occasionally misses a deadline.

Time-critical systems often deploy a combination of one or more application processors and a microcontroller on the same chip. The application processor runs the OS and applications, while the microcontroller is used to run time-critical tasks. This dual approach combines features of a modern system, while meeting the hard deadlines of a real-time system.

5 OVERVIEW OF ARM CORTEX PROCESSORS

Arm architecture has been constantly evolving over time.

Armv1 and Armv2 were early versions designed for personal computers, with processors running at lower tens of MHz. They established the foundational reduced instruction set computer (RISC) principles, achieving similar performance to more complex architectures but with significantly fewer

transistors. This efficiency was the key to their low power consumption and cost-effectiveness.

Armv3 onwards, cores were licensed to third-party vendors. Armv4 and Armv5 were extensively used in mobile phones for their power efficiency and compact design, evolving from feature phones to smartphones due to their multimedia capabilities. Beyond phones, these processors also powered embedded systems, automotive applications, and consumer electronics such as set-top boxes and digital cameras.

Armv6 introduced Cortex-A processors aimed at high-performance computing, and Cortex-M processors for low-power and cost-sensitive applications.

Armv7 further expanded Cortex-M family to include signal processing and floating-point computation capabilities. It also introduced Cortex-R processors, optimized for time-critical applications.

Armv8 introduced 64-bit processing with the AArch64 instruction set, alongside enhanced security features. It also included vector processing capabilities for signal processing and machine-learning tasks.

Armv9 is the latest architecture at the time of writing, building on Armv8 with further enhancements in security and performance optimizations for AI and machine learning computations.

Arm defines three architecture profiles for their Arm[®] Cortex[®] processors:

1. Arm Cortex-A, application processors
2. Arm Cortex-R, real-time processors
3. Arm Cortex-M, microcontroller processors

Let us discuss each briefly.

5.1 Cortex-A

Cortex-A processors are a series of high-performance processors designed to run multiple applications using a general-purpose OS.

High-end Cortex-A processors drive consumer products such as mobile phones, tablets and laptops. In embedded systems, they are ideal for demanding applications such as robotics and computer vision.

Mid-range Cortex-A processors provide reasonable computation power to entry-level phones as well as embedded systems that require moderate computation power, such as infotainment systems and smart-home devices.

Entry-level Cortex-A processors are power-efficient and often used in embedded systems primarily due to their ability to run an OS, for example, in point-of-sales terminals and industrial control panels.

5.2 Cortex-R

Cortex-R series includes a range of high-performance processors optimized for hard real-time applications. Unlike Cortex-A processors, which are designed for