

RP2040 Assembly Language Programming



ARM Cortex-M0+ on the
Raspberry Pi Pico

—
Stephen Smith

apress®

RP2040 Assembly Language Programming

**ARM Cortex-M0+
on the Raspberry Pi Pico**

Stephen Smith

Apress®

RP2040 Assembly Language Programming: ARM Cortex-M0+ on the Raspberry Pi Pico

Stephen Smith
Gibsons, BC, Canada

ISBN-13 (pbk): 978-1-4842-7752-2
<https://doi.org/10.1007/978-1-4842-7753-9>

ISBN-13 (electronic): 978-1-4842-7753-9

Copyright © 2022 by Stephen Smith

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-7752-2. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*This book is dedicated to my beloved wife and
editor Cathalynn Labonté-Smith.*

Table of Contents

About the Author	xv
About the Technical Reviewer	xvii
Acknowledgments	xix
Introduction	xxi
 Chapter 1: How to Set Up the Development Environment.....	 1
About the Raspberry Pi Pico	3
About the Host Computer.....	4
How to Solder and Wire.....	5
How to Install Software.....	6
A Simple Program to Ensure Things Are Working	7
Create Some Helper Script Files	8
Summary.....	9
 Chapter 2: Our First Assembly Language Program	 11
10 Reasons to Use Assembly Language	12
Computers and Numbers	15
ARM Assembly Instructions	17
CPU Registers	18
ARM Instruction Format.....	19
RP2040 Memory.....	21
About the GCC Assembler	23
Hello World.....	23

TABLE OF CONTENTS

Our First Assembly Language File.....28

 About the Starting Comment28

 Where to Start29

 Assembly Instructions30

 Data31

 Program Logic31

Reverse Engineering Our Program.....33

Summary.....36

Exercises.....36

Chapter 3: How to Build and Debug Programs39

 CMake39

 GNU Make42

 Print Statements45

 GDB46

 Preparing to Debug.....47

 Beginning GDB.....48

 Summary.....55

 Exercises.....56

Chapter 4: How to Load and Add57

 About Negative Numbers57

 About Two’s Complement57

 About Raspberry Pi OS Calculator59

 About One’s Complement60

 Big- vs. Little-Endian60

 About Bi-Endian.....61

 Pros of Little-Endian61

 Cons of Little-Endian62

How to Shift and Rotate Registers	62
About Carry Flag	63
Basics of Shifting and Rotating	63
How to Use MOV	65
Move Immediate	65
Moving Data from One Register to Another Using Register MOV	65
ADD/ADC	66
Add with Carry	68
SUB/SBC	69
Shifting and Rotating	70
Loading All 32 Bits of a Register	71
MOV/ADD/Shift Example	72
Summary	77
Exercises	78
Chapter 5: How to Control Program Flow	79
Unconditional Branch	79
About the CPSR	80
Branch on Condition	81
About the CMP Instruction	82
Loops	83
FOR Loops	83
While Loops	84
If/Then/Else	85
Logical Operators	86
AND	87
EOR	88

TABLE OF CONTENTS

ORR.....88

BIC88

MVN88

TST89

Design Patterns.....89

Converting Integers to ASCII90

 Using Expressions in Immediate Constants.....94

 Storing a Register to Memory.....94

 Why Not Print in Decimal?.....95

Performance of Branch Instructions95

Summary.....96

Exercises.....97

Chapter 6: Thanks for the Memories99

 How to Define Memory Contents100

 How to Align Data103

 How to Load a Register104

 How to Load a Register with an Address.....104

 How to Load Data from Memory.....106

 Optimizing Small Read-Only Data Access108

 Indexing Through Memory.....109

 How to Store a Register112

 How to Convert to Uppercase112

 How to Load and Store Multiple Registers.....118

 Summary.....119

 Exercises.....119

Chapter 7: How to Call Functions and Use the Stack.....	121
About Stacks on the RP2040.....	122
How to Branch with Link.....	123
About Nesting Function Calls.....	124
About Function Parameters and Return Values	126
How to Manage the Registers.....	127
Summary of the Function Call Algorithm	128
More on the Branch Instructions.....	129
About the X Factor	130
Uppercase Revisited	131
About Stack Frames.....	137
Stack Frame Example.....	138
How to Create Macros.....	139
About Include Directive	142
How to Define a Macro	142
About Labels.....	143
Why Macros?	144
Summary.....	144
Exercises.....	145
Chapter 8: Interacting with C and the SDK.....	147
How to Wire Flashing LEDs	148
How to Flash LEDs with the SDK.....	149
How to Call Assembly Routines from C.....	154
How to Embed Assembly Code Inside C Code.....	156
Summary.....	160
Exercises.....	160

TABLE OF CONTENTS

Chapter 9: How to Program the Built-in Hardware.....	161
About the RP2040 Memory Map	161
About C Header Files.....	162
About the Raspberry Pi Pico Pins.....	164
How to Set a Pin Function	165
About Hardware Registers and Concurrency	167
About Programming the Pads	169
How to Initialize SIO	169
How to Turn a Pin On/Off.....	170
The Complete Program	171
Summary.....	174
Exercises.....	175
Chapter 10: How to Initialize and Interact with Programmable I/O	177
About PIO Architecture	178
About the PIO Instructions	180
Flashing the LEDs with PIO	181
PIO Instruction Details and Examples	187
JMP	187
WAIT	188
IN	188
OUT	189
PUSH.....	190
PULL	190
MOV	191
IRQ	192
SET	193

About Controlling Timing.....	193
About the Clock Divider	193
About the Delay Operand.....	195
About Side-Set.....	197
More Configurable Options	198
Summary.....	199
Exercises.....	200
Chapter 11: How to Set and Catch Interrupts.....	201
Overview of the RP2040's Interrupts	201
About the RP2040's Interrupts.....	203
About the Interrupt Vector Table	205
About Saving Processor State	206
About Interrupt Priorities	207
Flashing LEDs with Timer Interrupts	208
About the RP2040 Alarm Timer.....	209
Setting the Interrupt Handler and Enabling IRQ0.....	210
The Complete Program.....	211
About the SVCcall Interrupt.....	218
Using the SDK	218
Summary.....	218
Exercises.....	219
Chapter 12: Multiplication, Division, and Floating Point.....	221
Multiplication	221
Division	222
About Division and Interrupts	224

TABLE OF CONTENTS

Interpolation.....225

 Adding an Array of Integers.....227

 Interpolating Between Numbers.....229

Floating Point.....232

 About the Structure of the Boot ROM233

 Sample Floating-Point Program236

 Some Notes on C and printf.....238

Summary.....239

Exercises.....240

Chapter 13: Multiprocessing241

 About Saving Power.....241

 About Interprocessor Mailboxes242

 How to Run Code on the Second CPU244

 A Multiprocessing Example.....246

 About Fibonacci Numbers246

 About Factorials.....246

 The Complete Program.....247

 About Spinlocks253

 Regulating Access to a Memory Table.....254

 A Word on the SDK.....261

 Summary.....262

 Exercises.....262

Chapter 14: How to Connect Pico to IoT265

 About the RP2040's Built-in Temperature Sensor.....266

 About Home-Brewed Communication Protocol.....270

 About the Server Side of the Protocol.....271

About the RP2040's UART	273
Mastering Math Routines.....	278
Viewing the Main Program.....	282
About IoT, Wi-Fi, Bluetooth, and Serial Communications	286
Summary.....	287
Exercises.....	288
Appendix A: ASCII Character Set	291
Appendix B: Assembler Directives.....	303
Appendix C: Binary Formats	305
Integers.....	305
Floating Point.....	306
Addresses	306
Appendix D: The ARM Instruction Set.....	307
Answers to Exercises	311
Chapter 2	311
Chapter 4	311
Chapter 6	311
Chapter 9	312
Chapter 10	312
Index.....	313

About the Author



Stephen Smith is also the author of the Apress titles *Raspberry Pi Assembly Language Programming* and *Programming with 64-Bit ARM Assembly Language*. He is a retired software architect, located in Gibsons, BC, Canada. He's been developing software since high school, or way too many years to record. He was the chief architect for the Sage 300 line of accounting products for 23 years. Since retiring, he has pursued artificial intelligence,

earned his Advanced HAM Radio License, and enjoys mountain biking, hiking, and nature photography, and is a member of the Sunshine Coast Search and Rescue group. He continues to write his popular technology blog at <http://smist08.wordpress.com> and has written two science fiction novels in a series, *Influence* and *Unification*, available on <http://amazon.com>.

About the Technical Reviewer

Stewart Watkiss is a keen maker, programmer, and author of *Learn Electronics with Raspberry Pi*. He studied at the University of Hull, where he earned a master's degree in electronic engineering, and more recently at Georgia Institute of Technology, where he earned a master's degree in computer science.

Stewart also volunteers as a STEM ambassador, helping teach programming and physical computing to schoolchildren and at Raspberry Pi events. He has created a number of resources using Pygame Zero, which he makes available on his website (www.penguintutor.com).

Acknowledgments

No book is ever written in isolation. I want to especially thank my wife, Cathalynn Labonté-Smith, for her support, encouragement, and expert editing.

I want to thank all the good folk at Apress who made the whole process easy and enjoyable. A special shout-out to Jessica Vakili, my coordinating editor, who kept the whole project moving quickly and smoothly. Thanks to Aaron Black, senior editor, who recruited me and got the project started. Thanks to Stewart Watkiss, my technical reviewer, who helped make this a far better book.

Introduction

There is an explosion of DIY electronics projects, largely fueled by the Arduino-based microcontrollers and Raspberry Pi computers. Electronics projects have never been easier to build, with hundreds of inexpensive modular components to choose from. People are designing robots, home monitoring and security systems, game devices, musical instruments, audio systems, and a lot more. The Raspberry Pi Pico is the Raspberry Pi Foundation's entry into the Arduino-style microcontroller market. A regular Raspberry Pi computer runs Linux and typically costs from \$35 to \$100 depending on memory and accessories. The Raspberry Pi Pico costs \$4 and doesn't run an operating system.

To power the Raspberry Pi Pico, the Raspberry Pi Foundation designed a custom system on a chip (SoC), called the RP2040, containing dual ARM Cortex-M0+ CPUs along with a raft of device controller components. This combination of a powerful CPU and ease of integration has made this a great choice for any DIY project. Further, Raspberry sells the RP2040 chips separately, and other companies such as Seeed Studio, Adafruit, and Pimoroni are selling their own versions of this microcontroller with extra built-in features like Bluetooth or Wi-Fi. You can even buy RP2040 chips yourself for \$1 each and build your own board.

At the basic level, how are these microcontrollers programmed? What provides the magical foundation for all the great projects that people build on them? Raspberry provides an SDK for C programmers as well as support for programming in MicroPython. This book answers these questions and delves into how these are programmed at the bare metal level and provides insight into the RP2040's architecture.

INTRODUCTION

Assembly Language is the native, lowest-level way to program a computer. Each processing chip has its own Assembly Language. This book covers programming the ARM Cortex-M0+ 32-bit processor. To learn how a computer works, learning Assembly language is a great way to get into the nitty-gritty details. The popularity and low cost of microcontrollers like the Raspberry Pi Pico provide ideal platforms to learn advanced concepts in computing.

Even though all these devices are low powered and compact, they're still sophisticated computers with a multicore processor, programmable I/O processors, and integrated hardware controllers. Anything learned about these devices is directly relevant to any gadget with an ARM processor, which by volume is the number one processor on the market today.

In this book, we cover how to program ARM Cortex-M0+ processors at the lowest level, operating as close to the hardware as possible. You will learn the following:

- How to format instructions and combine them into programs, as well as details of the operative binary data formats
- How to program the built-in programmable I/O, division, and interpolation coprocessors
- How to control the integrated hardware devices by reading and writing to the hardware control registers directly
- How to interact with the RP2040 SDK

The simplest way to learn these tasks is with a Raspberry Pi Pico connected to a Raspberry Pi running the Raspberry Pi OS, a version of Linux. This provides all the tools needed to learn Assembly Language programming. All the software required for this book is open source and readily available on the Raspberry Pi.

This book contains many working programs to play with, use as a starting point, or study. The only way to learn programming is by doing, so don't be afraid to experiment, as it is the only way to learn.

Even if Assembly programming isn't used in your day-to-day life, knowing how the processor works at the Assembly Language level and knowing the low-level binary data structures will make you a better programmer in all other areas. Knowing how the processor works will let you write more efficient C code and can even help with Python programming.

Enjoy your introduction to Assembly Language. Learning it for one processor family helps with learning and using any other processor architectures encountered throughout your career.

Source Code Location

The source code for the example code in the book is located on the Apress GitHub site at the following URL:

<https://github.com/Apress/RP2040-Assembly-Language-Programming>

The code is organized by chapter and includes answers to the programming exercises.

CHAPTER 1

How to Set Up the Development Environment

Microcontrollers like the Raspberry Pi Pico are typically utilized as the brains for smart devices, like microwave ovens, dishwashers, home security systems, weather stations, or irrigation monitors and controllers. At best, they have a small display and perhaps a couple of buttons for taking commands; however, they are still fully functioning computers. The programs that run on them can be quite powerful and sophisticated. Since the microcontroller usually doesn't have a keyboard, mouse, or monitor, we develop their programs on a regular computer, known as a host computer, and then upload the program to the microcontroller to test and finally deploy it.

The Raspberry Pi Pico is a board built around Raspberry's RP2040 ARM CPU chip. Not only is this the heart of the Raspberry Pi Pico, but also Raspberry sells this chip to other manufacturers, including Adafruit, Arduino, Seeed Studio, SparkFun, and Pimoroni. These other companies produce boards like the Raspberry Pi Pico but with different feature sets. For instance, some contain Wi-Fi or Bluetooth functions, easily connect to rechargeable batteries, or are in much smaller form factors. In this book, when we refer to the RP2040, it applies to all the brands of RP2040

boards. However, in some cases, we will talk about a specific board, perhaps, because we are discussing Wi-Fi or are referring to specific wiring connections for one board.

Programming the RP2040 in Assembly Language is the main emphasis of this book, but we want to do this by studying real working programs. To do this, we need to hook up our microcontroller to various pieces of hardware. This way we can see programs that perform useful tasks and learn all the flexible and powerful features the RP2040 has to connect to external sensors, controllers, and communication channels. To begin with, we set up the Raspberry Pi Pico on an electronics breadboard, so we can easily wire in the various devices to play with.

This chapter is concerned with physically setting up the Raspberry Pi Pico on a breadboard and wiring it up to a host computer to effortlessly program and debug programs, as well as hook up other components as we encounter them. The *Getting started with Raspberry Pi Pico* guide (from www.raspberrypi.org/documentation/rp2040/getting-started/) is an excellent reference on how to do these fundamental tasks. We will not duplicate the contents of the guide; instead, we will point out the important parts that are required for Assembly Language programming, debugging, and playing with the sample programs in this book.

To run most of the programs in this book, you will need

- A Raspberry Pi Pico
- An electronics breadboard
- Pins to attach the Pico to the breadboard
- Miscellaneous connecting wires
- A selection of LEDs
- A soldering iron and solder
- A Raspberry Pi 4 running Raspberry Pi OS

About the Raspberry Pi Pico

The heart of the Raspberry Pi Pico is a new chip developed by Raspberry and ARM. This chip is a system on a chip (SoC) that contains a dual core ARM Cortex-M0+ CPU, 264KB of SRAM, USB port, and support for several hardware devices. Compared to a full computer like the regular Raspberry Pi, the Raspberry Pico lacks a video output port, an operating system, and USB ports for a keyboard and a mouse. But it is possible to connect displays and input devices to the Raspberry Pi Pico, as we'll see later in the book. The specialty connections and input devices aren't used for general-purpose computing; rather, they solve specific problems, such as powering a vending machine and monitoring a greenhouse.

Unlike the CPUs found in desktop and laptop computers, the RP2040 doesn't contain a floating-point unit, vector processing unit, or graphic processing unit. However, one thing it has that regular CPUs lack is a set of eight programmable I/O (PIO) coprocessors. These PIOs have their own Assembly Language and can handle many I/O protocols and tasks independent of the two CPU cores. We'll cover PIOs in [Chapter 11](#). If you already have your RP2040 board wired up and know how to download and debug C programs, then you might want to skip ahead to [Chapter 2](#).

The RP2040 may look underpowered when comparing it to a modern Intel, AMD, or ARM processor, but for the price, it is quite a powerful computer. [Table 1-1](#) compares the RP2040 to some older and newer computers as well as competitors' microcontrollers.

Table 1-1. *Comparison of the Processing Power of the RP2040*

Computer	CPU	Speed (MHz)	Memory (KB)	Bits	Cores
Apple II	MOS 6502	1	48	8	1
IBM PC	Intel 8088	4.77	640	16	1
Arduino Nano	ATmega 328	16	2	8	1
Arduino Due	ARM M3	84	96	32	1
RP2040	ARM M0+	133	264	32	2
Pi Zero	ARM A53	1024	524,288	32	1
Pi 4	ARM A72	1536	8,388,608	64	4

About the Host Computer

Since microcontrollers don’t have a keyboard, a display, or even an operating system, their programs are written on a host computer. For RP2040-based microcontrollers, this could be on a MacOS, Windows, or Linux-based computer. The Raspberry Pi Pico documentation has instructions on how to connect it to all these platforms. The easiest solution is to use a Raspberry Pi 4 as the host vs. using a Windows or Mac computer. Raspberry has made this easy with a complete installation script and clear instructions on how to wire the Raspberry Pi 4 and Raspberry Pi Pico together. The wiring solution of these two boards is the easiest one since the Raspberry Pi 4 already exposes all the necessary pins via its GPIO pins. In this book, we’ll use the Raspberry Pi 4, point out the features we will be using, and let you follow the Raspberry-provided documentation to set it up.

How to Solder and Wire

You can't do much with a Raspberry Pi Pico without doing some soldering. Without soldering, you can download programs to the RP2040, flash the onboard LED, and send data back out the USB port to the host computer. However, even to just debug a program, you must do some soldering. The easiest way to set things up is to solder a set of pins to each side of the board, so it can be inserted into an electronics breadboard, which then allows us to connect things up without further soldering. This is great for experimenting. Typically, we would use a new RP2040 board to solder into a final project. At \$4 each, there isn't a significant overhead in having a development board and adding new boards to the package when you are finished. To perform debugging requires you to solder pins to the three debugging connections on the end of the board.

The minimum wiring needed is the following three connections between the Pico and the Raspberry Pi 4:

1. Using a micro-USB cable
2. Via the three debugging pins
3. Via a serial port using pins 1, 2, and 3

Don't be scared of soldering; it is actually quite simple and fun. The main trick is to heat up the area where you want the solder to go and touch a bit of solder there. Don't melt it onto the soldering iron's tip and then try to drip it from there. Some vendors provide an option to purchase boards with the pins presoldered for a few dollars extra. Others provide the pins separately, and it is up to you to ensure they are included in your order. Even if the main pins are presoldered, chances are you are going to need to solder pins to the three debug pads. Figure 1-1 shows the wiring, minus the USB cable, of a Raspberry Pi Pico connected to a Raspberry Pi 4.

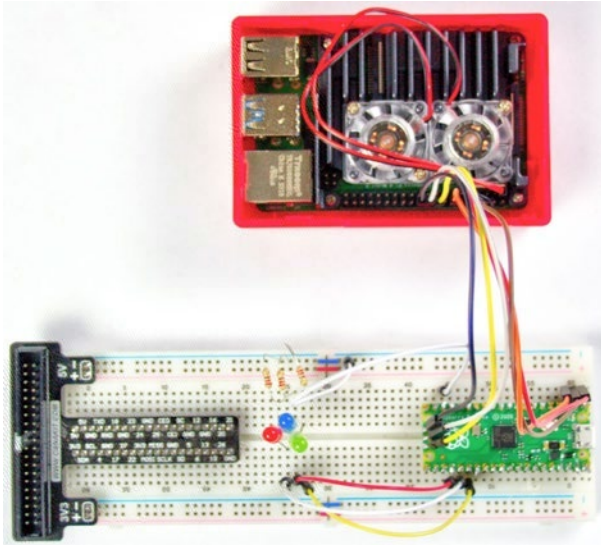


Figure 1-1. A Raspberry Pi Pico installed in a breadboard and connected to a Raspberry Pi4. The USB cable was removed for clarity. Three LEDs are connected as well.

Note If you are using an RP2040 board other than the Raspberry Pi Pico, then it is likely that the pins are in different locations on the board, and you will need to adapt the wiring for the location of the pins.

How to Install Software

If you are using a Raspberry Pi as your host computer, then this is straightforward. Use the Raspberry Pi OS as your operating system. This simplifies installation, since it runs 32-bit ARM code and shares development tools with the Raspberry Pi Pico and other RP2040-based

boards. The `pico_setup.sh` script downloads and installs everything required to develop code for RP2040-based systems. As Raspberry's *Getting Started* guide documents, you get `pico_setup.sh` using `wget`:

```
wget https://raw.githubusercontent.com/raspberrypi/pico-setup/master/pico_setup.sh
```

This script sets up both C and Assembly Language programming.

The *Getting Started* guide includes instructions for working with Visual Studio Code, which you are welcome to use, but we won't be covering in this book. This book covers text files that can be edited in any editor, using `cmake` and `make` for building, `gdb` and `openocd` for debugging, and the `minicom` for communications.

A Simple Program to Ensure Things Are Working

The easiest way to ensure everything is working is to compile and play with a couple of the SDK examples. The *Getting started with Raspberry Pi Pico* guide walks you through how to do this. Here, rather than duplicate, we'll list the key things you need to be comfortable with, since we will be doing them over and over throughout this book. Here is what you need to know:

1. How to load a program by powering on the Pico while holding down the BootSel button and copying a program to the shared drive
2. How to compile a program to either send its output to the USB or serial port
3. How to use the `minicom` to display the output that the Pico is sending

4. How to compile a program for debugging
5. How to use openocd and gdb to load and execute a program for debugging

Tip Building a program requires running both `cmake` and `make`. It isn't always clear which part does what. If you make configuration changes, it is best to delete and recreate the build folder ensuring everything is built from scratch.

Create Some Helper Script Files

When you follow along with the *Getting started with Raspberry Pi Pico* guide, there are quite a few long command lines to type in (or to copy/paste). It saves quite a bit of time to create a collection of small shell scripts to automate the common tasks. You can put these in `$HOME/bin` and then add

```
export PATH=$PATH:$HOME/bin
```

to the end of the `$HOME/.bashrc` file. You also need to make these executable with

```
chmod +x filename
```

Next, we need two scripts for `minicom`—one to listen on the UART and one to listen on the USB, as follows:

File `m-uart`:

```
minicom -b 115200 -o -D /dev/serial0
```

File `m-usb`:

```
minicom -b 115200 -o -D /dev/ttyACM0
```

To build debug, I have a script `cmaked` containing

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

To run `openocd`, ready to accept connections from `gdb`, I have the script `ocdg` containing

```
openocd -f interface/raspberrypi-swd.cfg -f target/rp2040.cfg
```

To run `gdb-multiarch` where the elf file to be debugged is passed as a parameter, I have `gdbm` containing

```
gdb-multiarch $1
```

When `gdb` starts, we need to connect to `openocd`. We can automate this by creating a `.gdbinit` file in `$HOME`. This file then contains

```
target remote localhost:3333
```

Note This `.gdbinit` will be used anytime you start `gdb`, so if you need to debug a local file without using `openocd`, then you might want to rename this file while you do that.

Summary

This chapter is the starting point. We haven't done any Assembly Language programming yet, but now we are set up to write, debug, test, and deploy programs written in either C or Assembly Language. The Raspberry Pi Pico is connected to the Raspberry Pi 4 through a USB cable, a serial port, and the debugging port. The Pico is installed in an electronics breadboard ready to have other components connected to it. In [Chapter 2](#), we will use all these tools to start our journey with RP2040 Assembly Language.

CHAPTER 2

Our First Assembly Language Program

Most of the functionality of a Raspberry Pi Pico is contained in the custom RP2040 chip that contains dual core ARM Cortex-M0+ CPUs. The ARM processor was originally developed by a group in Great Britain, who wanted to build a successor to the BBC Microcomputer used for educational purposes. The BBC Microcomputer used the 6502 processor, which was a simple processor with a simple instruction set. The problem was there was no successor to the 6502. They weren't happy with the microprocessors that were around at the time, since they were much more complicated than the 6502 and they didn't want to make another IBM PC clone. They took the bold move to design their own. They developed the Acorn computer that used it and tried to position it as the successor to the BBC Microcomputer. The idea was to use Reduced Instruction Set Computer (RISC) technology as opposed to Complex Instruction Set Computer (CISC) as championed by Intel and Motorola.

Developing silicon chips is an expensive proposition, and unless you can get a good volume going, manufacturing is expensive. The ARM processor probably wouldn't have gone anywhere except that Apple came calling looking for a processor for a new device they had under development—the iPod. The key selling point for Apple was that as the ARM processor was RISC, therefore, it used less silicon than CISC processors and as a result used far less power. This meant it was possible to build a device that ran for a long time on a single battery charge.

Unlike Intel, ARM doesn't manufacture chips, it just licenses the designs for others to optimize and manufacture chips. With Apple onboard, suddenly there was a lot of interest in ARM, and several big manufacturers started producing chips. With the advent of smartphones, the ARM chip really took off and now is used in pretty much every phone and tablet and even powers some Chromebooks, making it the number one processor in the computer market.

The designers at ARM are ambitious and architect their processors ranging from low-cost microcontrollers all the way up to the most powerful CPUs used in supercomputers. ARM's line of microcontroller CPUs is the Cortex-M series. We are most interested in the ARM Cortex-M0+ used in Raspberry's RP2040 SoC. To make this chip inexpensive, the transistor count is reduced as much as possible. The M-series CPUs are all 32 bits but have fewer registers and a smaller instruction set than the full A-series ARM CPUs like those used in the full Raspberry Pi. The M-series CPUs are optimized to use as little memory as possible as memory tends to be limited in microcontrollers, again to keep costs down. In this book, we'll see how the Cortex-M0+ works at the lowest level and will often have to deal with the trade-offs made by the chip designers keeping transistor counts down. There are several optional components available from ARM for these chips. We'll consider the ones included in the RP2040, such as the fast integer multiplier and divider (multiplication and division are an extra).

10 Reasons to Use Assembly Language

You can program the Raspberry Pi Pico in MicroPython or C/C++. These are productive languages that hide the details of all the bits and bytes, letting you focus on your application problem. When you program in Assembly Language, you are tightly coupled to a given CPU, and moving your program to another CPU requires a complete rewrite. Each Assembly