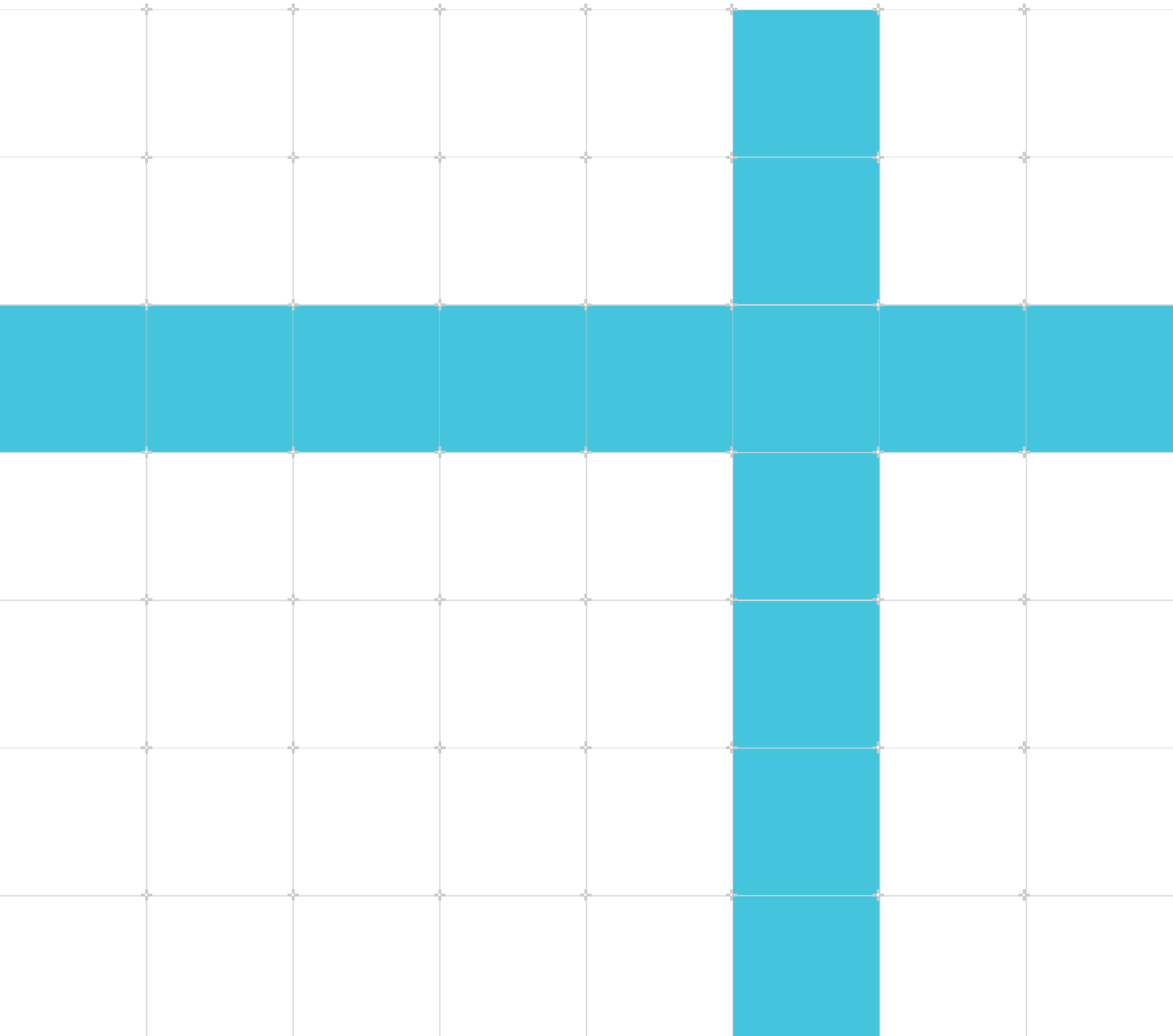
arm

Writing Arm assembly code

Version 1.0

Issue 02
102438_0100_02_en



Writing Arm assembly code

Release information

Document history

Issue	Date	Confidentiality	Change
0100-02	1 January 2021	Non-Confidential	Initial release

Proprietary Notice

or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

Most developers write most of their code using a high-level language such as C and C++. This high-level source code is then compiled to machine code which runs on a target device.

Sometimes, however, writing low-level assembly code has advantages. Perhaps you want to hand-optimize a critical algorithm to make it as fast as possible. Or perhaps you need to maintain legacy code that is already written in assembly. Or maybe you just want to learn more about the low-level operation of your code to understand more about how it works.

In all these situations, you will need to understand how to read and write Arm assembly code.

2. Learning about the instruction set

Assembly instructions are the fundamental building blocks of any program. If you are going to write assembly code, you will need to understand what instructions are available to you.

The precise set of available instructions for a particular device is called the instruction set. The Arm architecture supports three Instruction Set Architectures (ISAs): A64, A32 and T32. Use these resources for more information:

- Learn more about the different instruction sets supported by the Arm architecture.
- Use the ISA exploration tools to discover the available A64, A32, and T32 instructions in easy-to-browse XML and HTML formats.
- Refer to the Arm Architecture Reference Manuals (A-profile, R-profile, and M-profile) for definitive ISA details.

3. Learning about assembly language

Although the instruction set reference materials described in the Overview are comprehensive, they do not provide the best starting point for beginners.

The following resources introduce the basic concepts of programming in Arm assembly language:

- The Cortex-A Series Programmer's Guide explains architectural fundamentals and an introduction to assembly language code, along with other useful information for programmers.
- Arm Assembly Language: Fundamentals and Techniques by William Hohl is a popular resource with the Arm University Program. This book discusses the basics of assembly language.
- Embedded Systems Fundamentals with Arm Cortex-M based Microcontrollers: A Practical Approach by Dr Alexander G. Dean includes a chapter correlating C programming features with those in assembly code.

The Arm Compiler 5 toolchain (executable name `armasm`) uses a different syntax for assembly code to Arm Compiler 6 (executable name `armclang`) and GNU (executable name as). Although the instructions are mostly the same regardless of toolchain, the syntax around the instructions varies.

- Overview of differences between armasm and GNU syntax assembly code describes the syntax differences you must be aware of if you are using legacy tools or migrating assembly code.

4. Mixing assembly code with other programming languages

Even though you can now write Arm assembly code, you probably do not want to use it to hand-code your entire application.

You will probably want to write a small number of key functions in assembly, and call those functions from your main application code. You might want to do this to make use of existing assembly code, but the rest of your project is in C or C++.

- Calling assembly functions from C and C++ in the Arm Compiler User Guide shows you how to make function calls from C/C++ code to assembly code using the Arm Compiler 6 toolchain.
- Beyond Hello World: Advanced Arm Compiler 5 Features provides a similar example using Arm Compiler 5 within the DS-5 IDE.

5. Writing your first assembly program

Here are some examples that you can follow to get started with Arm assembly language.

- "Hello World" in Assembly is an Arm Community blog post that shows how to build a simple Arm assembly program with GCC, running either natively or with a cross-compiler.
- Assembling armasm and GNU syntax assembly code in the Arm Compiler Software Development Guide demonstrates another Arm assembly program, this time using the Arm Compiler 6 toolchain.
- Using the integrated assembler in the Arm Compiler User Guide provides another example.

6. Writing inline assembly code

Inline assembly lets you write assembly code directly in your C or C++ source code. This can simplify the task of writing assembly code. For example, you can leave register allocation to the compiler, instead of explicitly referring to specific registers in your code.

- Writing inline assembly code demonstrates how to use inline assembly code with the standalone Arm Compiler 6 toolchain.
- Using inline assembly to improve code efficiency shows an example of inline assembly code using Arm Compiler 5 within the DS-5 IDE.