

# ASL Syntax Reference

## DDI 0620

Arm Architecture Technology Group

January 23, 2024



# Contents

<b>1</b>	<b>Non-Confidential Proprietary Notice</b>	<b>5</b>
<b>2</b>	<b>Disclaimer</b>	<b>7</b>
<b>3</b>	<b>ASL Abstract Syntax</b>	<b>9</b>



# Chapter 1

## Non-Confidential Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, any patents, copyrights, trade secrets, trademarks, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly 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 this document shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. Please follow Arm’s trademark usage guidelines at <https://www.arm.com/company/policies/trademarks>.

Copyright © [2023,2024] Arm Limited (or its affiliates). All rights reserved.

Arm Limited. Company 02557590 registered in England. 110 Fulbourn Road, Cambridge, England CB1 9NJ. (LES-PRE-20349)

## Chapter 2

# Disclaimer

This document is part of the ASLRef material. It is a snapshot of: <https://github.com/herd/herdtools7/commit/6dd15fe7833fea24eb94933486d0858038f0c2e8>

This material covers both ASLv0 (viz, the existing ASL pseudocode language which appears in the Arm Architecture Reference Manual) and ASLv1, a new, experimental, and as yet unreleased version of ASL.

The development version of ASLRef can be found here `~/herdtools7/asllib`.

A list of open items being worked on can be found here `~/herdtools7/asllib/doc/ASLRefProgress.tex`.

This material is work in progress, more precisely at Alpha quality as per Arm's quality standards. In particular, this means that it would be premature to base any production tool development on this material.

However, any feedback, question, query and feature request would be most welcome; those can be sent to Arm's Architecture Formal Team Lead Jade Alglave ([jade.alglave@arm.com](mailto:jade.alglave@arm.com)) or by raising issues or PRs to the herdtools7 github repository.





## Chapter 3

# ASL Abstract Syntax

An Abstract Syntax Tree (AST for short) is a kind of labelled tree. A node in an AST is either a leaf, represented by its label, or an internal node of the form  $L(n_1, \dots, n_k)$  where  $L$  is its label and  $n_1, \dots, n_k$  are its ordered children nodes, which we also refer to as *components*. Components can be (possibly-empty) lists of nodes, shown as  $n^*$ , and optional nodes (lists of 0 or 1 elements), shown as  $n?$ . Tuples are shown as  $(n_1, \dots, n_k)$ .

An abstract syntax is similar to a context-free grammar, but defined over ASTs. A terminal derives leaf nodes while non-terminal use alternatives to derive internal nodes.

A major benefit of employing an abstract syntax is that it allows abstracting away syntactic details that are only important to enable correct parsing, such as punctuation, and succinctly representing lists and optional values. By defining an abstract syntax for ASL, we can uniformly represent programs in ASLv0 as well as ones in ASLv1 and define a single type system for them.

We define the abstract syntax of ASL below. We sometimes provide extra details to individual derivations by adding comments below them, in the form *(\* this is a comment \*)*.

node	components
unop ::=	"!"   "-"   "NOT"
binop ::=	"&&"   " "   "-->"   "<->"
	(* binop_boolean *)
	"=="   "!="   ">"   ">="   "<"   "<="
	(* binop_comparison *)
	"+"   "-"   "OR"   "XOR"   "EOR"   "AND"
	(* binop_add_sub_logic *)
	"*"   "/"   "DIV"   "DIVRM"   "MOD"   "<<"   ">>"
	(* binop_mul_div_shift *)
	"^"
	(* binop_pow *)
literal =	<int_lit>
	<hex_lit>
	(* merged into <int_lit>? *)
	<boolean_lit>
	<real_lit>
	<bitmask_lit>
	(* also represents <bitvector_lit> *)
	<string_lit>

node	components
expr ::=	E.Literal( <i>literal</i> )
	E.Var(<identifier>)
	E.CTC(expr, ty)
	(* A checked type constraint *)
	E.Binop(binop, expr, expr)
	E.Unop(unop, expr)
	E.Call(<identifier>, expr*, (<identifier>, expr)*)
	E.Slice(expr, slice*)
	E.Cond(expr, expr, expr)
	E.GetArray(expr, expr)
	E.GetField(expr, <identifier>)
	E.GetFields(expr, <identifier>*)
	E.Record(ty, (<identifier>, expr)*)
	(* Exception construction *)
	E.Concat(expr*)
	E.Tuple(expr*)
	E.Unknown(ty)
	E.Pattern(expr, pattern)

node	components
pattern ::=	Pattern.All
	E.Var(<identifier>)
	Pattern.Any(pattern*)
	Pattern.Geq(expr)
	Pattern.Leq(expr)
	Pattern.Mask(bitmask_lit)
	Pattern.Not(pattern)
	Pattern.Range(expr, expr)
	(* Lower to upper, included. *)
	Pattern.Single(expr)
	Pattern.Tuple(pattern*)

node	components
ty ::=	T.Int( <i>int_constraints?</i> )
	T.Real
	T.String
	T.Bool
	T.Bits(expr, <i>bitfield</i> *)
	(* <i>expr</i> is a statically evaluable expression denoting the length of the bit-vector. *)
	T.Enum(<identifier>*)
	T.Tuple(ty*)
	T.Array(expr, ty)
	T.Record(field*)
	T.Exception(field*)
	T.Named(<identifier>)
	(* A type variable. *)
	(* This is related to $I_{LDNP}$ *)

node	components
int_constraint ::=	Constraint.Exact(expr)
	(* A single value, given by a statically evaluable expression. *)
	Constraint.Range(expr, expr)
	(* An interval between two statically evaluable expression. *)

node	components
bitfield ::=	BitField.Simple(<identifier>, slice*)
	(* A name and its corresponding slice. *)
	BitField.Nested(<identifier>, slice*, bitfield*)
	(* A name, its corresponding slice and some nested bitfields. *)
	BitField.Type(<identifier>, slice*, ty)
	(* A name, its corresponding slice, and the type of the bitfield. *)