ARMCC 迁移毕昇编译器指南

文档版本 01
发布日期 2024-12-20

海思技术有限公司

地址：	上海市青浦区虹桥港路2号101室　　邮编：201721
网址：	https://www.hisilicon.com/cn/
客户服务邮箱：	support@hisilicon.com

前 言

概述

本文档用于指导工程代码从ARMCC编译器切换到毕昇编译器进行开发。本文主要介绍ARMCC编译器和毕昇编译器的差异和代码迁移方法。

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
⚠ 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
⚠ 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
⚠ 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
🕮 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修订记录

修订日期	版本	修订说明
2024-07-05	00B01	第1次临时版本发布。
2024-12-20	01	第1次正式版本发布。

目 录

插图目录

表格目录

1 概述

本文档主要是描述从ARMCC编译器的代码迁移到毕昇编译器。当用户进行代码迁移时，在遵循标准C的基础上重点关注非标准的关键字、pragmas、内建函数等，本文档重点描述ARMCC和毕昇编译器对比相关的差异。

图 **1-1** ARMCC 编译器

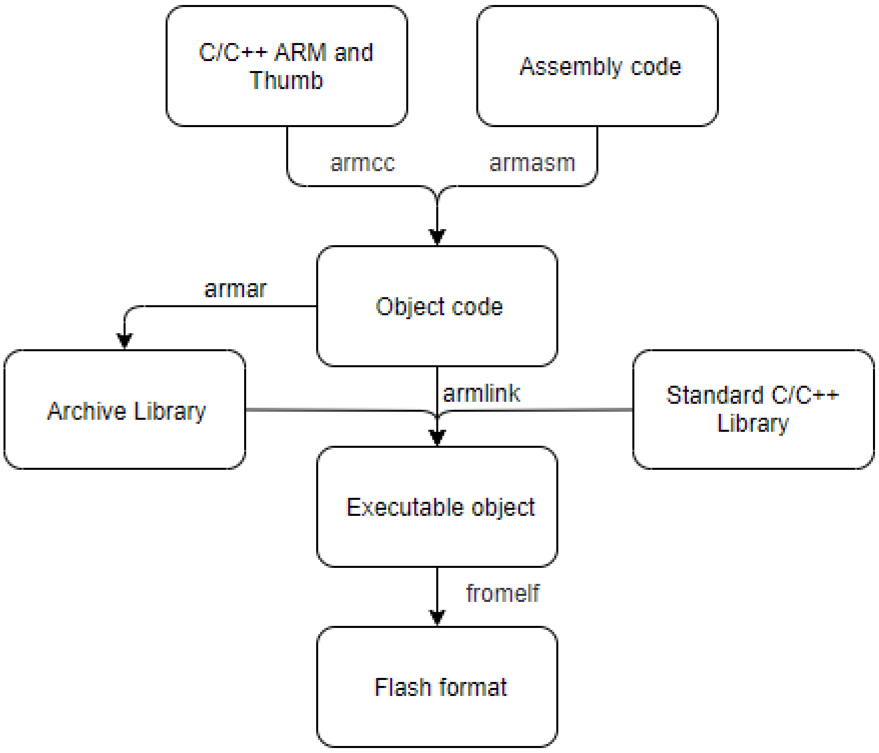

图 1-2 毕昇编译器

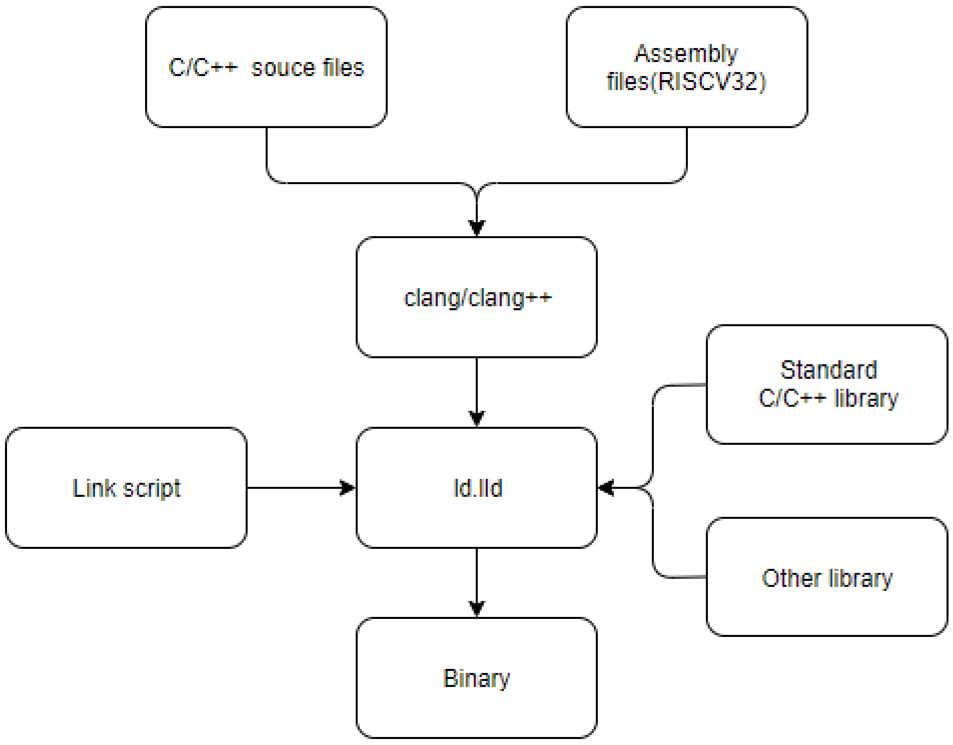

表 1-1 编译器对比

工具	ARMCC	毕昇编译器
Compiler	KEIL5 ARM C/C++ Compiler(ARMCC) 集成版本 ARM Compiler 5.06 update 7 (build 960) 支持标准C90/C99/C++03/C++11	clang for C clang++ for C++ 基于开源软件LLVM-15.0.4构建 支持标准到C17/C++17
Assembler	KEIL5 ARM Assembler(armasm) 汇编语言对应ARM指令集要求，汇编器的伪指令满足armasm风格。	1. LLVM的汇编器是集成到了clang工具命令中，我们可以使用clang命令编译汇编。 2. riscv32-linux-musl-as是毕昇编译器集成GNU binutils的工具，汇编语言对应RISCV指令集要求，汇编器的伪指令满足GNU风格，具体差异可见下文描述。

工具	ARMCC	毕昇编译器
Linker	KEIL5 ARM Linker(armlink) 链接脚本是arm自定义的语法。	1. ld.lld是LLVM原生链接器，毕昇编译器推荐使用ll.lld，未来会继续优化，从而生成性能更优、codesize更小的二进制程序。 2. riscv32-linux-musl-ld是毕昇编译器集成GNU binutils的工具，链接脚本是满足GNU Linker script定义的语法要求。
Archiver	KEIL5 ARM Archiver(armar)	1. llvm-ar 2. riscv32-linux-musl-ar
Image Conversion utility	KEIL5 ARM Image Converter(fromelf)	1. llvm-objcopy 2. riscv32-linux-musl-objdump

2 编译器

2.1 命令选项

编译器的命令选项差别如表2-1所示，下表列出场景编译选项的差异，具体的详细信息还需要参考相关文档。

表 2-1 编译器命令选项差异

ARMCC	毕昇编译器
-Aopt Pass option as an option to the assembler	**-Wa,<arg>** **-Xassembler <arg>**
--apcs=/fpic **--apcs=/nofpic** Enables or disables the generation of read-only position-independent code where relative address references are independent of the location where your program is loaded	**-fpic** Generate position-independent code **-fno-pic** Don't generate position-independent code (default)
--apcs=/hardfp **--apcs=/softfp** Requests hardware or software floating-point linkage.	**-march=rv32imfc_xhimideer -mabi=ilp32f** **-march=rv32imc_xhimideer -mabi=ilp32**
--asm_dir=directory_name Specifies a directory for disassembly output files created by the --asm and -S options.	Use **-S -o** to specifies a directory for disassembly file
--autoinline, --no_autoinline Enables and disables automatic inlining of functions.	BiSheng compiler automatically decides whether to inline functions depending on the optimization level

ARMCC	毕昇编译器
--branch_tables,--no_branch_tables Controls whether the compiler places branch tables for switch statements.	-fjump-tables,-fno-jump-tables
--bss_threshold=num Controls the placement of small global ZI data items in sections.	BiSheng compiler put zero initialized data in the bss section defaultly, disable it use -fno-zero-initialized-in-bss
-C Instructs the compiler to retain comments in preprocessor output.	**-E -C**
--c90 Enables the compilation of C90 source code.	**-std=c90**
--c90 --gnu Enables the compilation of C90 source code with additional GNU extensions.	**-std=gnu90**
--c99 Enables the compilation of C99 source code.	**-std=c99**
--c99 --gnu Enables the compilation of C99 source code with additional GNU extensions.	**-std=gnu99**
--compile_all_input, --no_compile_all_input Enables and disables the suppression of filename extension processing, enabling the compiler to compile files with any filename extensions. When enabled, the compiler suppresses filename extension processing entirely, treating all input files as if they have the suffix .c.	**-xc** Specify explicitly the C language for the following input files (rather than letting the compiler choose a default based on the file name suffix)
--cpp Enables the compilation of C++03 source code.	**-std=c++03**
--cpp --gnu Enables the compilation of C++03 source code with additional GNU extensions.	**-std=gnu++03**

ARMCC	毕昇编译器
--cpp11 Enables the compilation of C++11 source code.	**-std=c++11**
--cpp11 --gnu Enables the compilation of C++11 source code with additional GNU extensions.	**-std=gnu++11**
--data_reorder, --no_data_reorder Enables and disables automatic reordering of top-level data items, for example global variables.	Not supported.
--debug, --no_debug Enables and disables the generation of debug tables.	**-g** Generate debug information in default format.
--default_definition_visibility=visibility Controls the default ELF symbol visibility of extern variable and function definitions.	**-fvisibility=[default|internal|hidden|protected]** Set the default symbol visibility.
--default_extension=ext Changes the filename extension for object files from the default extension (.o) to an extension of your choice.	By default, the filename extension for object files is .o. Use **-c -o** specifies the full name of the object file. The following example creates an object file called test.obj, instead of test.o: clang -c -o test.obj test.c
--depend Specifies a filename for the makefile dependency rules.	**-M -MF** <file> Write make dependency output to the given file.
--depend_dir Specifies the directory for dependency output files.	Use **-MF** to specify dependency file with directory individually.
--depend_format=unix_escaped Dependency file entries use UNIX-style path separators and escapes spaces with \\.	BiShengcompiler use UNIX-style path separator symbol / and escapes spaces symbol \\ defaultly.

ARMCC	毕昇编译器
--depend_system_headers,--no_depend_system_headers Enables and disables the output of system include dependency lines when generating makefile dependency information using either the -M option or the --md option.	**-M** Generate make dependencie with system header files. **-MM** Generate make dependencie, but ignore system header files.
--depend_target Specifies the target for makefile dependency generation.	**-MT** <target> Add an unquoted target.
--diag_error=tag[,tag,...] Sets diagnostic messages that have a specific tag to Error severity.	**-Werror=**<specifier> Make the specified warning into an error. The specifier for a warning is appended; for example ‘**-Werror=switch**’ turns the warnings controlled by ‘**-Wswitch**’ into errors.
--diag_style=<string> Specifies the display style for diagnostic messages.	RISCV32 compiler produces diagnostic messages in the following format: <source-file>:<line-number>:<char-number>: <description> [<diagnostic-flag>]
--diag_suppress=<tag> Suppress warning message <flag>.	**-Wno-<flag>** Specific warning options has a negative form beginning ‘**-Wno-**’ to turn off warnings.
--diag_warning=tag[,tag,...] Sets diagnostic messages that have a specific tag to Warning severity.	**-W<flag>** You can request many specific warnings with options beginning with ‘**-W**’, for example '**-Wimplicit**’ to request warnings on implicit declarations.
--dollar,--no_dollar Enables and disables the use of dollar signs, $, in identifiers.	**-fdollars-in-identifiers,-fno-dollars-in-identifiers**
--dwarf2 Uses DWARF 2 debug table format.	**-gdwarf-2**
--dwarf3 Uses DWARF 3 debug table format.	**-gdwarf-3**

ARMCC	毕昇编译器
--echo Displays the complete expanded command line, and any separate commands that invoke other external applications, such as armasm or armlink.	**-v** Display the programs invoked by the compiler.
--enum_is_int Forces the size of all enumeration types to be at least four bytes.	**-fshort-enums, -fno-short-enums** Allocate to an enum type only as many bytes as it needs for the declared range of possible values.By default BiSheng compiler uses -fno-short-enums to set the minimum size to 32-bit.
--errors=filename Redirects the output of diagnostic messages from stderr to the specified errors file.	Use unix redirection operator to redirect the output of diagnostic messages from stderr to the specified errors file. clang test.c **&>** <log file>
--exceptions, --no_exceptions Enables and disables exception handling.	**-fexceptions,-fno-exceptions**
--exceptions_unwind, --no_exceptions_unwind Enables and disables function unwinding for exception-aware code.	Not supported.
--execstack, --no_execstack Generates a .note.GNU-stack section marking the stack as either executable or non-executable.	**-Wl,-z execstack, -Wl,-z noexecstack**
--extended_initializers, --no_extended_initializers Enables and disables the use of extended constant initializers even when compiling with --strict or --strict_warnings.	**-pedantic** Issue warnings needed for strict compliance to the standard.

ARMCC	毕昇编译器
--feedback=filename Enables the linker to communicate with the compiler to eliminate unused functions.	**-ffunction-sections -fdata-sections -Wl,--gc-sections** Option **-ffunction-sections** instruct compiler place each function into its own section. Option **-ffunction-sections** instruct compiler place each data in its own section. **Option -Wl,--gc-sections** instruct linker remove unused sections. This command line combination instruct compiler to eliminate unused functions.
--forceline Forces aggressive inlining of functions.	clang automatically decides whether to inline functions depending on the optimization level.
--friend_injection, --no_friend_injection Controls the visibility of friend declarations in C++.	Not supported.
--global_reg=<reg_name> Treats the specified register names as fixed registers, and prevents the compiler from generating code that uses these registers.	Not supported.
--gnu Enables the GNU compiler extensions that the ARM compiler supports.	Not supported.
--gnu_instrument, --no_gnu_instrument Inserts GCC-style instrumentation calls for profiling entry and exit to functions.	**-finstrument-functions** Instrument function entry and exit with profiling calls.
--hide_all, --no_hide_all Controls symbol visibility when building SVr4 shared objects.	**-fvisibility=default** Sets the default visibility of ELF symbols to the specified option, unless overridden in the source with the __attribute__((visibility("<visibility_type>"))) attribute. The default is -fvisibility=hidden.
--ignore_missing_headers Prints dependency lines for header files even if the header files are missing.	**-M -MG** Treat missing header files as generated files.

ARMCC	毕昇编译器
--implicit_include, --no_implicit_include Controls the implicit inclusion of source files as a method of finding definitions of template entities to be instantiated in C++.	Not supported.
--info=totals Reports total sizes of the object code and data for each object file.	Use llvm-size utility.
--inline Consider all functions for inlining, even if they are not declared inline	**-finline-functions** Integrate functions not declared "inline" into their callers when profitable.
--interleave Interleaves C or C++ source code line by line as comments within the assembly listing.	Compile source file with -g option, then use llvm-objdump utility with -S -l option to disassemble.
-J Adds the specified directories to the list of places that are searched to find included system header files.	**-isystem**
-L Pass option as an option to the linker.	**-Wl,option or -Xlinker option**
--list_macros List all the macros that are defined at the end of the translation unit, including the predefined macros.	**-E -dM**
--long_long Permits use of the long long data type in strict mode.	long long is supported by clang defaultly.

ARMCC	毕昇编译器
--loop_optimization_level=opt Trades code size for performance by controlling how much loop optimization the compiler performs. **0** Specifies that the compiler does not perform any loop optimization. This option is usually best for code size. **1** Specifies that the compiler performs some loop optimization. This option tries to balance code size and performance. **2** Specifies that the compiler performs high-level optimization, including aggressive loop optimization. This option is usually best for performance.	**-O0, -O1, -O2, -O3, -Ofast, -Os, -Oz** Specify which optimization level to use: **-O0** Means “no optimization”: this level compiles the fastest and generates the most debuggable code. **-O1** Somewhere between **-O0** and **-O2**. **-O2** Moderate level of optimization which enables most optimizations. **-O3** Like **-O2**, except that it enables optimizations that take longer to perform or that may generate larger code (in an attempt to make the program run faster). **-Ofast** Enables all the optimizations from **-O3** along with other aggressive optimizations that may violate strict compliance with language standards. **-Os** Like **-O2** with extra optimizations to reduce code size. **-Oz** Like **-Os** (and thus **-O2**), but reduces code size further.
--loose_implicit_cast Makes illegal implicit casts legal, such as implicit casts of a nonzero integer to a pointer.	Makes illegal implicit casts legal defaultly, but emit a warning diagnosis messages.
--multifile, --no_multifile Enables and disables optimizations between multiple source files.	No direct equivalent.However, use **-flto** runs the standard link-time optimizer， or **-fwhole-program** to perform whole program optimizations. **-fwhole-program** option should not be used in combination with **-flto.**
--narrow_volatile_bitfields Accesses volatile bitfields using the smallest access size that contains the entire bitfield.	Not supported.
--no_protect_stack Explicitly disables stack protection.	**-fno-stack-protector**

ARMCC	毕昇编译器
-Ospace Performs optimizations to reduce image size at the expense of a possible increase in execution time.	**-Os**
-Otime Performs optimizations to reduce execution time at the expense of a possible increase in image size.	**-O1/-O2/-O3/-Ofast** RISCV32 compiler optimizes for execution time by default, unless specify the **-Os** options.
--phony_targets Emits dummy makefile rules.	**-MP**
--preinclude Include the source code of a specified file at the beginning of the compilation.	**-include**
--preprocessed Forces the preprocessor to handle files with .i filename extensions as if macros have already been substituted.	Not supported.
--protect_stack Enables stack protection on vulnerable functions.	**-fstack-protector** Use propolice as a stack protection method. **-fstack-protector-strong** Use a smart stack protection method for certain functions
--reassociate_saturation, --no_reassociate_saturation Enables and disables more aggressive optimization in loops that use saturating arithmetic.	Not supported.
--protect_stack_all Enables stack protection on all functions.	**-fstack-protector-all**
--relaxed_ref_def Permits multiple object files to use tentative definitions of global variables	**-fcommon**
--retain Restricts the optimizations performed by the compiler.	**-O<num>**

ARMCC	毕昇编译器
--rtti, --no_rtti Controls support for the RTTI features dynamic_cast and typeid in C++.	**-frtti,-fno-rtti**
--show_cmdline Shows how the compiler processes the command-line.	**-v**
--signed_bitfields, --unsigned_bitfields Makes bitfields of type int signed or unsigned.	**-fsigned-char,-funsigned-char**
--split_sections Generates one ELF section for each function in the source file.	**-ffunction-sections**
--strict Generate errors if code violates strict ISO C and ISO C++.	**-pedantic** -Werror
--strict_warnings Generate warnings if code violates strict ISO C and ISO C++.	**-pedantic**
--use_frame_pointer, --no_use_frame_pointer Controls whether a register is reserved for storing the stack frame pointer.	Not supported.
--version_number Displays the version of ARMCC you are using.	**--version**
--wchar, --no_wchar ermits or forbids the use of wchar_t. **--wchar16, --wchar32** Sets the size of wchar_t type.	**-fshort-wchar** **-fno-short-wchar**
--wrap_diagnostics, --no_wrap_diagnostics Enables and disables the wrapping of error message text when it is too long to fit on a single line.	**-fmessage-length=<number>** Limit diagnostics to <number> characters per line. 0 suppresses line-wrapping.

2.2 扩展关键字

表 2-2 扩展关键字差异

ARMCC	毕昇编译器

__align(alignment)
The __align keyword instructs the compiler to align a variable on an n-byte boundary.
Example:
__align(8) char buffer[128];
	__attribute__ ((aligned (alignment)))
	Example:
	char buffer[128] __attribute__ ((aligned (8)));

__ALIGNOF__ The __ALIGNOF__ keyword returns the alignment requirement for a specified type, or for the type of a specified object.	**__alignof__**
__asm This keyword passes information from the compiler to the ARM assembler armasm.	**asm** **__asm__**
__forceinline The __forceinline keyword forces the compiler to compile a C or C++ function inline. Example: __forceinline static int max(int x, int y);	**__attribute__((always_inline)).** Example: static int max(int x, int y) __attribute__((always_inline));
__global_reg The __global_reg storage class specifier causes the compiler to reserve a register for a specific global variable. Example: __global_reg(2) int x;	You can define a global register variable and associate it with a specified register like this: register int x asm ("r12"); Here r12 is the name of the register that should be used. It suffices to specify the compiler option ‘-ffixed-reg’ to reserve the register. Alternatively, you can use equivalent inline assembler instructions.
__inline The __inline keyword suggests to the compiler that it compiles a C or C++ function inline, if it is sensible to do so.	Use the **inline** keyword in its declaration, like this: static inline int f(int x){} If you are writing a header file to be included in ISO C90 programs, write **__inline__** instead of inline.

ARMCC	毕昇编译器
__int64 The __int64 keyword is a synonym for the keyword sequence long long.	You can use int64_t, which is a 64-bit integer type defined in the header file <stdint.h> (for C source files) or <cstdint> (for C++ source files). You can also use long long, however, if you use long long in C90 mode, the compiler gives: ● a warning. ● an error, if you also use -pedantic-errors.
__INTADDR__ The __INTADDR__ operation treats the enclosed expression as a constant expression, and converts it to an integer constant.	Not supported.

__irq
The __irq keyword enables a C or C++ function to be used as an exception handler.
	Not supported
	NOTE: In clang(RISCV), interrupt functions are just normal functions.
	Example：
	void IRQ_Handler(void)
	{
	}

__packed
The __packed qualifier is useful to map a structure to an external data structure, or for accessing unaligned data, but it is generally not useful to save data size because of the relatively high cost of unaligned access.
Example：
typedef __packed struct {
char x; // all fields inherit the __packed qualifier
int y;
} X;
	__attribute__((packed))
	typedef struct __attribute__((packed))
	{
	char x; // all fields inherit the __packed qualifier
	int y;
	} X;

__pure The __pure keyword asserts that a function declaration is pure. A function is pure only if the result depends exclusively on the values of its arguments and has no side effects.	**__attribute__ ((pure))**

ARMCC	毕昇编译器
__smc The __smc keyword declares an SMC (Secure Monitor Call) function. A call to the SMC function inserts an SMC instruction into the instruction stream generated by the compiler at the point of function invocation.	This keyword is specific to ARM architecture and is not supported in clang for RISC-V architecture.
__softfp The __softfp keyword asserts that a function uses software floating-point linkage. It is implicitly added to functions when softfp linkage is used	This keyword is specific to ARM architecture and is not supported in GCC for RISC-V architecture. The alternative is place functions which need use software floating-point linkage in a source file, then specify **-mabi=ilp32** option to compile source file.
__svc The __svc keyword declares a SuperVisor Call (SVC) function taking up to four integer-like arguments and returning up to four results in a value_in_regs structure. **__svc_indirect** **__svc**	This keyword is specific to ARM architecture and is not supported in BiSheng for RISC-V architecture.
__value_in_regs The __value_in_regs qualifier instructs the compiler to return a structure of up to four integer words in integer registers or up to four floats or doubles in floating-point registers rather than using memory.	Not supported
__weak This keyword instructs the compiler to export symbols weakly. Example: __weak void f(void);	**__attribute__ ((weak))** Example: void f(void) __attribute__ ((weak));
__writeonly The __writeonly type qualifier indicates that a data object cannot be read from.	Not supported.

2.3 扩展属性

表 **2-3** 扩展关键字 attributes 差异

ARMCC	毕昇编译器

__declspec(dllexport)
The __declspec(dllexport) attribute exports the definition of a symbol through the dynamic symbol table when building DLL libraries.
__declspec(dllimport)
The __declspec(dllimport) attribute imports a symbol through the dynamic symbol table when linking against DLL libraries.
	Not supported.

__declspec(noinline) The __declspec(noinline) attribute suppresses the inlining of a function at the call points of the function.	**__attribute__((noinline))**
__declspec(noreturn) Informs the compiler that the function does not return. The compiler can then perform optimizations by removing code that is never reached.	**__attribute__((noreturn))** or keyword **_Noreturn**
__declspec(nothrow) The __declspec(nothrow) attribute asserts that a call to a function never results in a C++ exception being propagated from the callee into the caller.	**__attribute__((nothrow))**
__declspec(notshared) The __declspec(notshared) attribute prevents a specific class from having its virtual functions table and RTTI exported.	Not supported.
__declspec(thread) The __declspec(thread) attribute asserts that variables are thread-local and have thread storage duration, so that the linker arranges for the storage to be allocated automatically when a thread is created.	keyword **__thread**

ARMCC	毕昇编译器
__attribute__((const)) The const function attribute specifies that a function examines only its arguments, and has no effect except for the return value. That is, the function does not read or modify any global memory.	**__attribute__ ((pure))**
__attribute__((nomerge)) This function attribute prevents calls to the function that are distinct in the source from being combined in the object code.	**__attribute__ ((nomerge))**
__attribute__((notailcall)) This function attribute prevents tailcalling of the function. That is, the function is always called with a branch-and-link, even if (because the call occurs at the end of a function) the branch-and-link could be converted to a branch.	**__attribute__((disable_tail_calls))**
__attribute__((pcs("calling_convention"))) This function attribute specifies the calling convention on targets with hardware floating-point, as an alternative to the __softfp keyword.	Not supported.
__attribute__((bitband)) type attribute __attribute__((bitband)) is a type attribute that gives you efficient atomic access to single-bit values in SRAM and Peripheral regions of the memory architecture.	Not supported.
__attribute__((alias)) variable attribute This variable addttribute enables you to specify multiple aliases for a variable.	Not supported.
__attribute__((at(address))) variable attribute This variable attribute enables you to specify the absolute address of a variable.	Not supported.

ARMCC	毕昇编译器
__attribute__((noinline)) constant variable attribute The noinline variable attribute prevents the compiler from making any use of a constant data value for optimization purposes, without affecting its placement in the object.	Not supported.
__attribute__((weakref("target"))) variable attribute This variable attribute marks a variable declaration as an alias that does not by itself require a definition to be given for the target symbol.	Not supported.
__attribute__((zero_init)) variable attribute The section attribute specifies that a variable must be placed in a particular data section.	**__attribute__((section(".bss")))** BiSheng Compiler by default places zero-initialized variables in a .bss section. However, there is no equivalent to generate an error when you specify an initializer.

2.4 pragmas

表 2-4 pragmas 差异

ARMCC	毕昇编译器
#pragma anon_unions, #pragma no_anon_unions These pragmas enable and disable support for anonymous structures and unions.	In C, anonymous structs and unions are a C11 extension which is enabled by default in armclang. If you specify the -pedantic option, the compiler emits warnings about extensions do not match the specified language standard. In C++, anonymous unions are part of the language standard, and are always enabled. However, anonymous structs and classes are an extension. If you specify the -pedantic option, the compiler emits warnings about anonymous structs and classes.

ARMCC	毕昇编译器
#pragma arm This pragma switches code generation to the ARM instruction set. It overrides the --thumbcompiler option. **#pragma thumb** This pragma switches code generation to the THUMB instruction set. It overrides the --arm compiler option.	Not supported. This pragmas is specific to ARM architecture.
#pragma arm section This pragma specifies a section name to be used for subsequent functions or objects. This includes definitions of anonymous objects the compiler creates for initializations.	The #pragma clang section directive provides a means to assign section-names to global variables, functions and static variables. Reference: https://releases.llvm.org/15.0.0/tools/clang/docs/LanguageExtensions.html#specifying-section-names-for-global-objects-pragma-clang-section
#pragma diag_default **#pragma diag_suppress** **#pragma diag_remark** **#pragma diag_warning** **#pragma diag_error**	Not supported.
#pragma exceptions_unwind, #pragma no_exceptions_unwind These pragmas enable and disable function unwinding.	Use the **__attribute__((nothrow))** function attribute instead.
#pragma hdrstop This pragma enables you to specify where the set of precompilation header files end.	Not supported.
#pragma import (<symbol>) This pragma generates an importing reference to symbol_name.	Not supported. __asm__(".global <symbol>\n\t");

ARMCC	毕昇编译器
#pragma inline, #pragma no_inline These pragmas control inlining, similar to the --inline and --no_inline command-line options.	**inline** This attribute suggests a function from being considered for inlining. **__attribute__((aways_inline))** This attribute inlines the function even if no optimization level was specified. **__attribute__((noinline))** This attribute prevents a function from being considered for inlining.
#pragma no_pch This pragma suppresses Precompiled Header (PCH) processing.	Not supported.
#pragma Onum This pragma changes the optimization level for all subsequent functions.	Clang provides a mechanism for selectively disabling optimizations in one function or a range of function defition. Reference: https://releases.llvm.org/15.0.0/tools/clang/docs/LanguageExtensions.html#extensions-for-selectively-disabling-optimization
#pragma Ospace This pragma optimizes all subsequent functions for code size, performing optimizations to reduce image size at the expense of a possible increase in execution time.	
#pragma Otime This pragma optimizes all subsequent functions for speed, performing optimizations to reduce execution time at the expense of a possible increase in image size.	
#pragma pop This pragma restores the previously saved pragma state. **#pragma push** This pragma saves the current pragma state.	The #pragma clang attribute push variation of the directive pushes a new “scope” of #pragma clang attribute that attributes can be added to, and the #pragma clang attribute pop variation pops the scope. Reference: https://releases.llvm.org/15.0.0/tools/clang/docs/LanguageExtensions.html#specifying-an-attribute-for-multiple-declarations-pragma-clang-attribute

ARMCC	毕昇编译器
#pragma softfp_linkage, #pragma no_softfp_linkage These pragmas control software floating-point linkage.	Use the **-march=rv32imc,-mabi=ilp32** command-line option to set the calling convention on a per-file basis.
#pragma unroll [(n)] This pragma instructs the compiler to unroll a loop by n iterations.	#pragma clang loop unroll_count(value) #pragma clang loop unroll(full) Reference: https://releases.llvm.org/15.0.0/tools/clang/docs/LanguageExtensions.html#extensions-for-loop-hint-optimizations https://releases.llvm.org/15.0.0/tools/clang/docs/AttributeReference.html#pragma-unroll-pragma-nounroll
#pragma unroll_completely This pragma instructs the compiler to completely unroll a loop. It has an effect only if the compiler can determine the number of iterations the loop has.	

2.5 内建函数

表 **2-5** 内建函数差异

ARMCC	毕昇编译器
arm_neon.h The Neon co-processor implements the Advanced SIMD instruction set extension, as defined by the Arm architecture.	Not supported. NOTE: The functions defined in arm_neon.h are specific to ARM architecture and are not defined in GCC for RISC-V architecture.

2.6 默认行为差异

表 **2-6** 默认行为差异

常见默认行为差异	ARMCC	毕昇编译器
Floating-point linkage	--apcs=/hardfp or --apcs=/softfp	-march=rv32imc_xhimideer -mabi=ilp32 or -march=rv32imfc_xhimideer -mabi=ilp32f
Default output file	__image.axf	a.out
Optimization level	-O2	-O0

常见默认行为差异	ARMCC	毕昇编译器
Default C++ source language mode	C++03	C++17
Default C source language mode	C90	C17

3 汇编器

3.1 命令选项

表 **3-1** 汇编器命令选项差异

ARMCC	毕昇编译器
--apcs=/fpic, --apcs=/nofpic /fpic specifies that the code in the input file is read-only independent and references to addresses are suitable for use in a Linux shared object. The default is /nofpic.	**-fpic** Generate position-independent code **-fno-pic** Don't generate position-independent code (default)
--bi, --bigend Generates code suitable for an ARM processor using big-endian memory access.	**-mbig-endian** Assemble for big-endian
--comment_section, --no_comment_section Controls the inclusion of a comment section .comment in object files.	Not supported With the riscv32 assembler, use the GNU assembly .ident directive to manually add a comment section.
--debug Instructs the assembler to generate DWARF debug tables.	**-g --gen-debug** Generate debugging information
--depend=dependfile Writes makefile dependency lines to a file.	**--MD FILE**

ARMCC	毕昇编译器
--diag_error=tag[,tag,…] Sets diagnostic messages that have a specific tag to Error severity.	**--fatal-warnings** Treat warnings as errors
--diag_suppress=tag[,tag,…] Suppresses diagnostic messages that have a specific tag.	**-W --no-warn** Suppress warnings
--diag_warning=tag[,tag,…] Sets diagnostic messages that have a specific tag to Warning severity.	**--warn** Don't suppress warnings
--dwarf2 Uses DWARF 2 debug table format.	**--gdwarf-2**
--dwarf3 Uses DWARF 3 debug table format.	**--gdwarf-3**
--execstack, --no_execstack Generates a .note.GNU-stack section marking the stack as either executable or non-executable.	**--execstack,--noexecstack**
-idir[,dir, …] Adds directories to the source file include path.	**-I DIR** Add DIR to search list for .include directives
--keep Instructs the assembler to keep named local labels in the symbol table of the object file, for use by the debugger.	**-L,--keep-locals** Keep local symbols (e.g. starting with `L')
--length=n Sets the listing page length.	**--listing-cont-lines** Set the maximum number of continuation lines used
--no_warn Turns off warning messages.	**-W --no-warn**
--predefine "directive", --pd Instructs armasm to pre-execute one of the SETA, SETL, or SETS directives.	**--defsym SYM=VAL**

ARMCC	毕昇编译器
--unaligned_access, --no_unaligned_access Enables or disables unaligned accesses to data on ARM architecture-based processors.	With the RISCV32 assembler, use the RISC-V Directives instead. To enable unaligned access, use the directive as follows: **.attribute Tag_RISCV_unaligned_access, 1** To disable unaligned access, use the directive as follows: **.attribute Tag_RISCV_unaligned_access, 0**
--version_number Displays the version of armasm you are using.	**--version**
--via=filename Reads an additional list of input filenames and assembler options from filename.	**@FILE**
--width=n Sets the listing page width.	**--listing-lhs-width** Set the width in words of the output data column of

3.2 内联汇编

armasm的内联汇编和GNC GCC类似，内联汇编的格式如**表3-2**所示。

表 3-2 内联汇编差异

armasm	毕昇编译器

asm("instruction output,input,....");
asm{instruction output,input,....}
1.You can use C variable names directly inside inline assembly statements.
2.You do not have direct access to physical registers. You must use C or C++ variables names as operands, and the compiler maps them to physical register. You must set the value of these variables before you read them within an inline assembly statement.
3.If you use register names in inline assembly code, they are treated as C or C++ variables. They do not necessarily relate to the physical register of the same name. If the register name is not declared as a C or C++ variable, the compiler generates a warning.
Example：
int Add(int term1, int term2) {
int sum;
asm(
"add sum, term1, term2\n"
);
return sum;
}
	asm [volatile]("assemble code" ：output operands（optional）：input operands（optional）：list of clobbered registers（optional));
	1. The asm keyword can incorporate inline assembly code into a function using the GNU inline assembly syntax, this keyword also use __asm__ instead.
	2. The optional volatile keyword tells the compiler that the assembly code has side-effects that the output, input, and clobber lists do not represent，this keyword also use __volatile__ instead.
	Example：
	int Add(int term1, int term2) {
	int sum;
	asm("add %2, %1, %0 \n"
	: "=r"(sum)
	: "r"(term1), "r"(term2));
	return sum;
	}

3.3 汇编语法

表 **3-3** 汇编语法差异

armasm syntax	毕昇编译器
Comments A comment is the final part of a source line. The first semicolon on a line marks the beginning of a comment except where the semicolon appears inside a string literal. Example: ; This whole line is a comment MOV r1, #16 ; Load R0 with 16	There are two ways of rendering comments to as. In both cases the comment is equivalent to one space. ● Anything from ‘/*’ through the next ‘*/’ is a comment. This means you may not nest these comments. /* The only way to include a newline (’ \n’) in a comment is to use this sort of comment. */ /* This sort of comment does not nest. */ ● Anything from a line comment character up to the next newline is considered a comment and is ignored. // This sort of comment does not nest.
Labels A label is written as a symbol beginning in the first column. A label can appear either in a line on its own, or in a line with an instruction or directive. Whitespace separates the label from any following instruction or directive. Example: MOV R0,#16 loop SUB R0,R0,#1 ; "loop" is a label CMP R0,#0	A label is written as a symbol that either begins in the first column, or has nothing but whitespace between the first column and the label. A label can appear either in a line on its own, or in a line with an instruction or directive. A colon ":" follows the label (whitespace is allowed between the label and the colon). Example: mv a1,#16 loop: // "loop" label on its own line sub a1,a1,#1 cmp a1,#0 mv a1,#16 loop: sub a1,a1,#1 // "loop" label in a line with an instruction cmp a1,#0

armasm syntax	毕昇编译器
Numeric local labels A numeric local label is a number in the range 0-99, optionally followed by a scope name corresponding to a ROUT directive. Numeric local labels follow the same syntax as all other labels. Refer to numeric local labels using the following syntax: %[F|B][A|T]<n>[<routname>]	A numeric local label is a number. Numeric local labels follow the same syntax as all other labels. Refer to numeric local labels using the following syntax: <n>{f|b} Where: <n> is the number of the numeric local label. f and b instruct the assembler to search forwards and backwards respectively. There is no default. You must specify one of f or b.
Functions The FUNCTION directive marks the start of a function. PROC is a synonym for FUNCTION. The ENDFUNC directive marks the end of a function. ENDP is a synonym for ENDFUNC. example: myproc PROC ; Procedure body ENDP	Use the .type directive to identify symbols as functions. For example: .type myproc, "function" myproc: // Procedure body GNU syntax assembly code provides the .func and .endfunc directives. as uses the .size directive to set the symbol size:
Sections The AREA directive instructs the legacy assembler to assemble a new code or data section. Section attributes within the AREA directive provide information about the section. Available section attributes include the following: ● CODE specifies that the section contains machine instructions. ● READONLY specifies that the section must not be written to. ● ALIGN=<n> specifies that the section is aligned on a 2<n> byte boundary	The .section directive instructs the assembler to assemble a new code or data section. Flags provide information about the section. Common section flags include the following: ● a section is allocatable ● w section is writable ● x section is executable ● S section contains zero terminated strings

armasm syntax	毕昇编译器
Symbol naming rules armasm syntax symbols must start with a letter or the underscore character "_".	GNU syntax symbols must start with a letter, the underscore character "_", or a period ".".
Numeric literals ● **Hexadecimal literals** armasm syntax assembly provides two methods for specifying hexadecimal literals, the prefixes "&" and "0x". ● **<n-base>_<n-digits> format** armasm syntax assembly lets you specify numeric literals using the following format: <n-base>_<n-digits> For example: 2_1101 is the binary literal 1101 (13 in decimal). 8_27 is the octal literal 27 (23 in decimal).	GNU syntax assembly only supports the "0x" prefix for specifying hexadecimal literals. Convert any "&" prefixes to "0x". GNU syntax assembly does not support the <n-base>_<n-digits> format. Convert all instances to a supported numeric literal form.
Operators :OR:	|
:EOR:	^
:AND:	&
:NOT:	~
:SHL:	<<
:SHR:	>>
:LOR:	||
:LAND:	&&
:ROL:	Not supported
:ROR:	Not supported

3.4 汇编指示

表 3-4 汇编指示差异

armasm	毕昇编译器
Comments A comment is the final part of a source line. The first semicolon on a line marks the beginning of a comment except where the semicolon appears inside a string literal. example: ; This whole line is a comment MOV r1, #16 ; Load R0 with 16	The /* and */ markers identify multiline comments: /* This is a comment that spans multiple lines */ The // marker identifies the remainder of a line as a comment: MOV R0,#16 // Load R0 with 16
ALIAS The ALIAS directive creates an alias for a symbol.	**.weakref alias, target** This directive creates an alias to the target symbol that enables the symbol to be referenced with weak-symbol semantics, but without actually making it weak.
ALIGN The ALIGN directive aligns the current location to a specified boundary by padding with zeros or NOP instructions.	**.balign** GNU syntax assembly also provides the .align <n> directive. However, the format of <n> varies from system to system. The .balign directive provides the same alignment functionality as .align with a consistent behavior across all architectures.
AREA The AREA directive instructs the assembler to assemble a new code or data section.	**.section name**
ASSERT The ASSERT directive generates an error message during assembly if a given assertion is false.	Not supported，use follow directive combo instead **.ifeq absolute expression** **.error "string"** **.endif**

armasm	毕昇编译器
ATTR The ATTR set directives set values for the ABI build attributes. The ATTR scope directives specify the scope for which the set value applies to.	**.gnu_attribute tag,value** Record a gnu object attribute for this file.
COMMON The COMMON directive allocates a block of memory of the defined size, at the specified symbol.	**.comm symbol , length**
DCB The DCB directive allocates one or more bytes of memory, and defines the initial runtime contents of the memory.	**.byte expressions**
DCD and DCDU The DCD directive allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory. DCDU is the same, except that the memory alignment is arbitrary.	**.word expressions**
DCFD and DCFDU The DCFD directive allocates memory for word-aligned double-precision floating-point numbers, and defines the initial runtime contents of the memory. DCFDU is the same, except that the memory alignment is arbitrary.	**.double flonums** Emits double precision floating-point values.
DCFS and DCFSU The DCFS directive allocates memory for word-aligned single-precision floating-point numbers, and defines the initial runtime contents of the memory. DCFSU is the same, except that the memory alignment is arbitrary.	**.float flonums** Emits single precision floating-point values.

armasm	毕昇编译器
DCQ and DCQU The DCQ directive allocates one or more eight-byte blocks of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory. DCQU is the same, except that the memory alignment is arbitrary.	**.quad bignums** .quad expects zero or more bignums, separated by commas. For each bignum, it emits an 8-byte integer.
DCW and DCWU The DCW directive allocates one or more halfwords of memory, aligned on two-byte boundaries, and defines the initial runtime contents of the memory. DCWU is the same, except that the memory alignment is arbitrary.	**.hword expressions** This expects zero or more expressions, and emits a 16 bit number for each.
END The END directive informs the assembler that it has reached the end of a source file.	**.end**
ENDFUNC or ENDP The ENDFUNC directive marks the end of an AAPCS-conforming function. ENDP is a synonym for ENDFUNC.	**.endfunc**
ENTRY The ENTRY directive declares an entry point to a program.	Not supported，use follow command line instead. riscv32-linux-musl-ld -e ADDRESS, --entry ADDRESS
EQU The EQU directive gives a symbolic name to a numeric constant, a register-relative value or a PC-relative value.	**.equ symbol, expression** This directive sets the value of symbol to expression. It is synonymous with `.set'
EXPORTAS The EXPORTAS directive enables you to export a symbol from the object file, corresponding to a different symbol in the source file.	**.weakref alias, target**

armasm	毕昇编译器
FUNCTION or PROC The FUNCTION directive marks the start of a function. PROC is a synonym for FUNCTION.	**.func name[,label]** .func emits debugging information to denote function name, and is ignored unless the file is assembled with debugging enabled.
GET or INCLUDE The GET directive includes a file within the file being assembled. The included file is assembled at the location of the GET directive. INCLUDE is a synonym for GET.	**.include "file"** This directive provides a way to include supporting files at specified points in your source program. The code from file is assembled as if it followed the point of the .include; when the end of the included file is reached, assembly of the original file continues.
IF, ELSE, ENDIF, and ELIF The IF, ELSE, ENDIF, and ELIF directives allow you to conditionally assemble sequences of instructions and directives.	**.if, .ifndef, .else, .elseif, .endif**
IMPORT and EXTERN The IMPORT and EXTERN directives provide the assembler with a name that is not defined in the current assembly.	**.extern** .extern is accepted in the source program —for compatibility with other assemblers —but it is ignored. as treats all undefined symbols as external.
INCBIN The INCBIN directive includes a file within the file being assembled. The file is included as it is, without being assembled.	**.incbin "file"[,skip[,count]]** The incbin directive includes file verbatim at the current location.
INFO The INFO directive supports diagnostic generation on either pass of the assembly.	**.warning "string"** just emits a warning.
KEEP The KEEP directive instructs the assembler to retain named local labels in the symbol table in the object file.	**riscv32-linux-musl-as -L,--keep-locals** keep local symbols (e.g. starting with `L')
LCLA, LCLL, and LCLS The LCLA, LCLL, and LCLS directives declare and initialize local variables.	**.local names** marks each symbol in the commaseparated list of names as a local symbol so that it will not be externally visible

armasm	毕昇编译器
MACRO and MEND The MACRO directive marks the start of the definition of a macro. Macro expansion terminates at the MEND directive.	**.macro and .endm**
MEXIT The MEXIT directive exits a macro definition before the end.	**.exitm**
OPT The OPT directive sets listing options from within the source code.	**.list** Control (in conjunction with the .nolist directive) whether or not assembly listings are generated.
RELOC The RELOC directive explicitly encodes an ELF relocation in an object file.	**.reloc offset, reloc_name[, expression]** Generate a relocation at offset of type reloc name with value expression.
REQUIRE8 and PRESERVE8 The REQUIRE8 and PRESERVE8 directives specify that the current file requires or preserves eight-byte alignment of the stack.	**.gnu_attribute Tag_RISCV_stack_align 8** Emits a build attribute which guarantees that the functions in the file preserve 8-byte stack alignment.
SETA, SETL, and SETS The SETA, SETL, and SETS directives set the value of a local or global variable.	**.set symbol, expression**
SPACE or FILL The SPACE directive reserves a zeroed block of memory. The FILL directive reserves a block of memory to fill with a given value.	**.org new-lc , fill**
TTL and SUBT The TTL directive inserts a title at the start of each page of a listing file. The SUBT directive places a subtitle on the pages of a listing file.	**.title "heading"** Use heading as the title (second line, immediately after the source file name and pagenumber) when generating assembly listings.

4 链接器

4.1 命令选项

表 4-1 链接器命令选项差异

armlink	毕昇编译器
--add_needed, --no_add_needed Controls shared object dependencies of libraries that are not specified on the command-line.	**--as-needed** Only set DT_NEEDED for following dynamic libs if used **--no-as-needed** Always set DT_NEEDED for dynamic libraries mentioned on
--add_shared_references, --no_add_shared_references Affects the behavior of the --sysv mode. If you specify --add_shared_references when linking an application the linker adds references from shared libraries. The linker gives an undefined symbol error message if these references are not defined by the application or by some other shared library. These references can be satisfied by static archive format libraries. **--search_dynamic_libraries, --no_search_dynamic_libraries** Controls whether or not dynamic or static libraries are used for libraries specified with the --library option.	gnu linker will search a directory for a shared library before searching for static library. Use -static option instruct linker searching for static library only.

armlink	毕昇编译器
--diag_error=tag[,tag,…] Sets diagnostic messages that have a specific tag to Error severity.	**--fatal-warnings**
--diag_warning=tag[,tag,…] Sets diagnostic messages that have a specific tag to Warning severity. Syntax	**--no-fatal-warnings**
--compress_debug, --no_compress_debug Causes the linker to compress .debug_* sections, if it is sensible to do so.	**--compress-debug-sections=[none|zlib|zlib-gnu|zlib-gabi]** Compress DWARF debug sections using zlib
--dynamic_linker=name Specifies the dynamic linker to use to load and relocate the file at runtime.	**-I PROGRAM, --dynamic-linker PROGRAM** Set PROGRAM as the dynamic linker to use
--emit_non_debug_relocs Retains only relocations from non-debug sections in an executable file.	**-S, --strip-debug** Strip debugging symbols
--emit_relocs Retains all relocations in the executable file. This results in larger executable files.	**-q, --emit-relocs** Generate relocations in final output
--entry=location Specifies the unique initial entry point of the image. Although an image can have multiple entry points, only one can be the initial entry point.	**-e ADDRESS, --entry ADDRESS** Set start address
--execstack, --no_execstack Forces the linker to use either an executable stack or a non-executable stack.	**-z execstack, -z noexecstack**
--export_dynamic, --no_export_dynamic Controls the export of dynamic symbols to the dynamic symbols table.	**-E, --export-dynamic** Export all dynamic symbols **--no-export-dynamic** Undo the effect of --export-dynamic
--feedback=filename Generates a feedback file for input to the compiler. This file informs the compiler about unused functions.	Not supported Use follow option to reduce code size. **-ffunction-sections -fdata-sections -Wl,--gc-sections**

armlink	毕昇编译器
--feedback_image=option Changes the behavior of the linker when writing a feedback file with scatter-loading.	
--feedback_type=type Controls the information that the linker puts into the feedback file.	
--fini=symbol Specifies the symbol name to use to define the entry point for finalization code.	**-fini SYMBOL** Call SYMBOL at unload-time
--fpic Enables you to link Position-Independent Code (PIC).	**-pie, --pic-executable** Create a position independent executable
--init=symbol Specifies a symbol name to use for the initialization code. A dynamic linker executes this code when it loads the executable file or shared object.	**-init SYMBOL** Call SYMBOL at load-time
--libpath=pathlist Specifies a list of paths that the linker uses to search for the ARM standard C and C++ libraries.	**-L DIRECTORY, --library-path DIRECTORY** Add DIRECTORY to library search path
--library=name Enables the linker to search either a dynamic library or a static library without you having specifying the full library filename on the command-line.	**-l LIBNAME, --library LIBNAME** Search for library LIBNAME
--library_type=lib Selects the library to be used at link time. **--scanlib, --no_scanlib** Enables or disables scanning of the ARM libraries to resolve references.	**-nostdlib** Only use library directories specified on
--linker_script=ld_script Specifies a GNU linker ld script to use for linking images and shared objects for ARM Linux and partial linking.	**-T FILE, --script FILE** Read linker script

armlink	毕昇编译器
--locals, --no_locals Adds local symbols or removes local symbols depending on whether an image or partial object is being output.	**-x, --discard-all** Discard all local symbols **--discard-none** Don't discard any local symbols
--mangled, --unmangled Instructs the linker to display mangled or unmangled C++ symbol names in diagnostic messages, and in listings produced by the --xref, --xreffrom, --xrefto, and --symbols options.	**--demangle [=STYLE]** Demangle symbol names [using STYLE] **--no-demangle** Do not demangle symbol names
--map, --no_map Enables or disables the printing of a memory map.	**-M, --print-map** Print map file on standard output
--muldefweak, --no_muldefweak Enables or disables multiple weak definitions of a symbol.	**--allow-multiple-definition** Allow multiple definitions
--pagesize=pagesize Allows you to change the page size used when demand paging.	**-z common-page-size=SIZE** Set common page size to SIZE
--remove, --no_remove Enables or disables the removal of unused input sections from the image.	**--gc-sections, --no-gc-sections**
--runpath=pathlist Specifies a list of paths to be added to the search paths in the dynamic section.	**-rpath PATH** Set runtime shared library search path
--shared Creates a System V (SysV) shared object.	**-shared, -Bshareable** Create a shared library
--soname=name Specifies the shared object runtime name that is used as the dependency name by any object that links against this shared object.	**-h FILENAME, -soname FILENAME** Set internal name of shared library
--sort=algorithm Specifies the sorting algorithm used by the linker to determine the order of sections in an output image.	**--sort-section name|alignment** Sort sections by name or maximum alignment

armlink	毕昇编译器
--symver_script=filename Enables implicit symbol versioning.	**--version-script FILE** Read version information script
--symver_soname Enables implicit symbol versioning to force static binding.	**--default-symver** Create default symbol version
--sysroot=path Enables the linker to treat any absolute paths found in linker scripts to be treated as relative to the specified path.	**--sysroot=<DIRECTORY>** Override the default sysroot location
--undefined=symbol Prevents the removal of a specified symbol if it is undefined.	**--allow-shlib-undefined** Allow unresolved references in shared libraries
--userlibpath=pathlist Specifies a list of paths that the linker is to use to search for user libraries.	**-L DIRECTORY, --library-path DIRECTORY** Add DIRECTORY to library search path
--verbose Prints detailed information about the link operation, including the objects that are included and the libraries from which they are taken.	**--verbose [=NUMBER]** Output lots of information during link

4.2 链接脚本

本章仅描述概念上的差异，有关GNU链接器脚本的详细信息，请参阅**https://sourceware.org/binutils/docs-2.38/ld/Scripts.html#Scripts**。

GNU链接器脚本示例如下：

```
SECTIONS
{
    . = 0x10000;
    .text : { *(.text) }
    . = 0x8000000;
    .data : { *(.data) }
    .bss : { *(.bss) }
}
```

上述示例中，程序代码段应加载到地址0x10000，程序数据段应加载到地址0x8000000。

SECTIONS命令相当于分散加载中的load region，它指示链接器将输入文件组合成一个单独的输出文件。GNU链接器脚本的输出节必须满足输出格式的约束，例如，在System V中，只允许“.text”、“.data”、“.bss”或“Rodata”等格式。GNU链接器的输入节可以是符合输出格式约束的节名，也可以是自定义的节名，这与分散加载的输入节选择器不同。

MEMORY命令描述了内存块的位置和大小，例如：

```
MEMORY
{
    ROM(rx) : ORIGIN = 0x10000, LENGTH = 256K
    ROM(rwx) : ORIGIN = 0x8000000, LENGTH = 4M
}
SECTIONS
{
    .text : { *(.text) }> ROM
    .data : { *(.data) }> RAM
    .bss : { *(.bss) } > RAM
}
```

内存区域的属性类似于分散加载的输入节选择器。