IAR 迁移毕昇编译器指南

文档版本　01

发布日期　2024-12-20

商标声明

、HISILICON、海思和其他海思商标均为海思技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

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IAR（ARM）编译器切换到毕昇编译器进行开发。本文主要介绍IAR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C/C++代码从IAR C编译器（ARM）切换到毕昇编译器（RISCV）进行编译构建。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IAR 编译器（IAR）和毕昇编译器（RISCV）选项和扩展等方面差异。

图 1-1 IAR 编译器

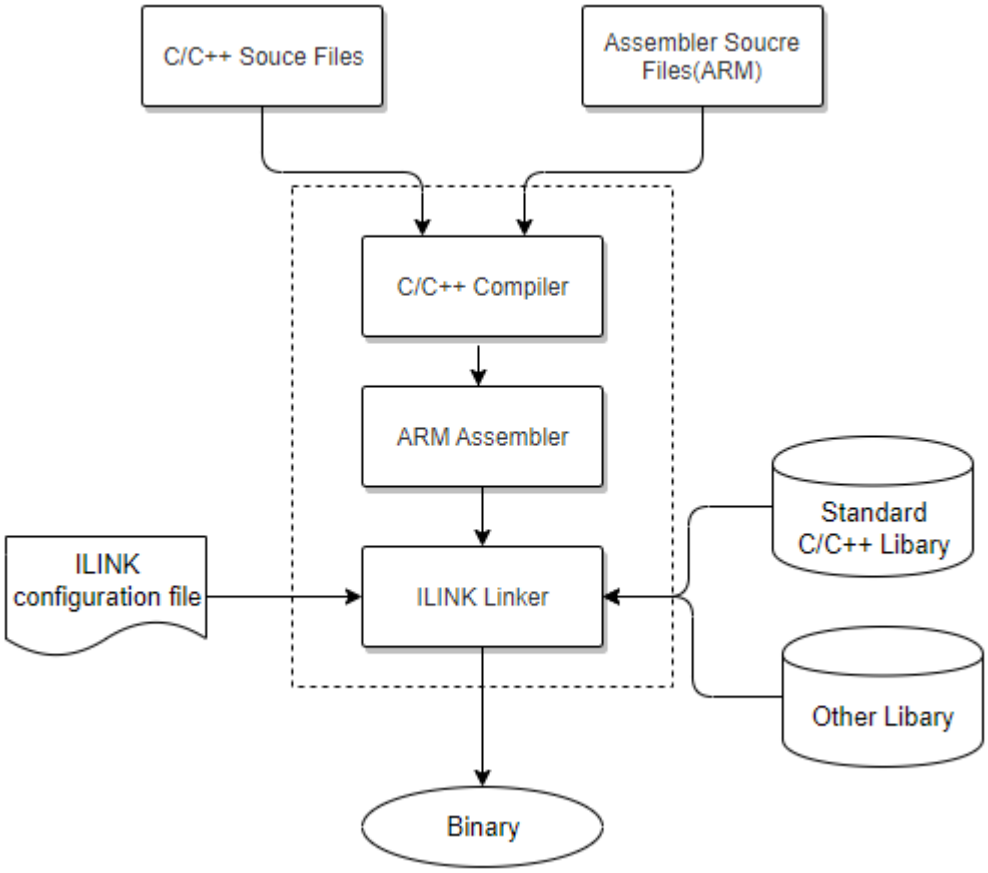

图 1-2 毕昇编译器

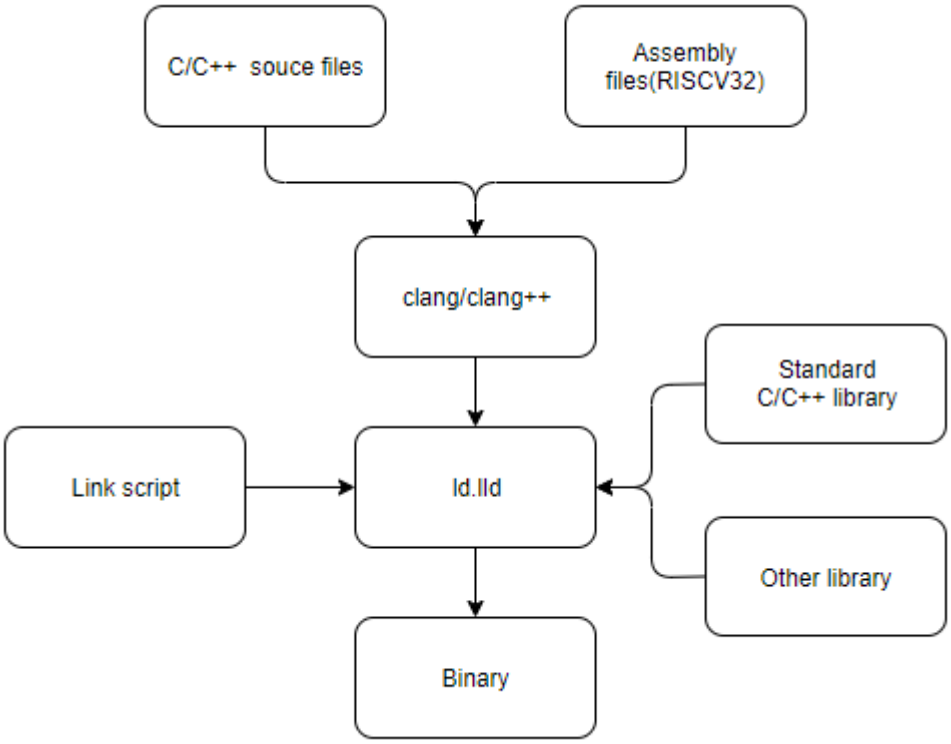

表 1-1 编译器对比

工具	IAR	毕昇编译器
Compiler	IAR C/C++ Compiler(iccarm)。 集成版本IAR Embedded wokbench IDE V9.30.1.x。 支持标准C89/C18。	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	IAR ARM Assembler(iasmarm) 汇编语言对应ARM指令集要求，汇编器的伪指令满足IAR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	IAR	毕昇编译器
Linker	IAR ILINK Linker(ilinkarm) 链接脚本是IAR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。

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，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

IAR	毕昇编译器
-preinclude = <file> Default include file.	**-include <file>** Include file before parsing
--align_func={1|2} By default, the compiler uses byte alignment for function entries. Use --align_func=2 to specify word alignment and force the compiler to align all function entries to an even address	-falign-functions indicates that the functions should be aligned to a 16-byte boundary. -falign-functions=n `-falign-functions=1` is the same as `-fno-align-functions`. The scalar `n` in `-falign-functions=n` must be an integral value between [0, 65536]. If the value is not a power-of-two, it will be rounded up to the nearest power-of-two.
--char_is_signed By default the compiler interprets the char type as unsigned.	**-fsigned-char** **-fno-unsigned-char** Let the type char be signed, like signed char.

IAR	毕昇编译器
--code_segment=name **-Rname** Place executable code in *name*.	**__attribute__ ((section ("SECTIONNAME")));** Puts the function foobar in the bar section. Example: void foobar (void) __attribute__ ((section ("bar"));
--diag_error=tag[,tag,...] Make all warnings into errors.	**-Werror**=<specified waring> Make the specified warning into an error. The specifier for a warning is appended. For example -Wno-error=switch makes -Wswitch warnings not be errors, even when -Werror is in effect.
--enable_multibytes Enables support for multibyte characters	Unicode characters are default supported. To add the character set "wchar.h" files need to be included passing "-std=c99" command line option.
--strict_ansi Enables strict ISO/ANSI	**-pedantic** Warn on language extensions.
-f filename Reads command line options from the named file, with the default extension xcl.	**@file** Read command-line options from file. The options read are inserted in place of the original @file option. If file does not exist, or cannot be read, then the option will be treated literally, and not removed.
--header_context It is necessary to know which header file that was included from what source line, to find the cause of a problem. Use this option to list, for each diagnostic message, not only the source position of the problem, but also the entire include stack at that point	**-Wsystem-headers** Print warning messages for constructs found in system header files.
--preprocess=c Preserve comments	**-C**
--preprocess=n Preprocess only	**-E**
--preprocess=l Generate #line directives	**-P**

IAR	毕昇编译器
--output {filename|directory} **-o {filename|directory}** specify an output file	**-o file**
--no_inline Disable function inlining	**-fno-inline**
--no_unroll Loop unroll.	**-fno-unroll-loops** **-fno-unroll-all-loops**
--no_warnings	By default all the warnings are disabled. **-Wall** enables the warning for normal code.
-r **--debug** Produce debugging information for supported debuggers.	**-g**
--require_prototypes This option forces the compiler to verify that all functions have proper prototypes.	Not supported NOTE: But some of the related optioned are: **-Wstrict-prototypes** (C only) Warn if a function is declared or defined without specifying the argument types. **-Wmissing-prototypes** (C only) Warn if a global function is defined without a previous prototype declaration. Warn when a declaration is found after a statement in a block.

IAR	毕昇编译器
-s[2|3|6|9|debug|low|medium|high] Use this option to make the compiler optimize the code for maximum execution speed. 2 – debug 3 – low 6 – medium **-z[2|3|6|9|debug|low|medium|high]** Use this option to make the compiler optimize the code for minimum size.	**-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.
-z9 Code size optimization.	**-Os -Oz** Code size optimization.
--align_data={1|2} **-u(1|2)** This attribute specifies a minimum alignment (in bytes) for variables of the specified type.	Not supported. __attribute__((aligned)) The aligned type attribute specifies a minimum alignment for the type. The aligned type attribute only increases the alignment of a struct or struct member, and does not decrease it.
--warnings_are_errors Make all warnings into errors.	**-Werror** Make all warnings into errors.

2.2 扩展关键字

表 2-2 扩展关键字差异

IAR	毕昇编译器
__absolute Object use absolute addressing	Control through section attributes and ld linker script.
__no_alloc **__no_alloc16** **__no_alloc_str** **__no_alloc_str16** Use the these object attribute on a constant to make the constant available in the executable file without occupying any space in the linked application.	Not supported. NOTE: In GCC/LLVM, you can use __attribute__((section('name'))) to achieve similar functionality.
__ro_placement This attribute specifies that a data object should be placed in read-only memory. Example： __ro_placement const volatile int x = 10;	Not supported. NOTE: In GCC/LLVM, you can use __attribute__((section('name'))) to achieve similar functionality. Example: __attribute__((section('.rodata')))
__no_init Use this keyword to place a data object in non-volatile memory.	Not supported. NOTE: In GCC/LLVM, you can use __attribute__((section('name'))) to achieve similar functionality. Example： __attribute__((section("no_init"))) const char reserved_area[2048];
__root Ensures that a function or variable is included in the object code even if unused. Example： __root int myarray[10];	**__attribute__((used))**
__weak Declares a symbol to be externally weakly linked	**__attribute__((weak))**

IAR	毕昇编译器
__naked Declares a function without generating code to set up or tear down the function’s frame.	**__attribute__((naked))**
__noreturn Informs the compiler that the function will not return	**_Noreturn** This keyword was introduced in C11.
__ramfunc Makes a function execute in RAM. Example: __ramfunc int FlashPage(char * data, char * page);	The alternative solution is to use attribute section to automatically run the code in RAM.
__stackless Makes a function callable without a working stack. Example: __stackless void start_application(void);	**__attribute__((naked,noreturn,optimize("-O3")))**
__task Relaxes the rules for preserving registers.	**__attribute__((naked))**

{ __irq | __fiq } [__nested] __arm void func(void) {}
Declares an interrupt function.
Example:
__irq __arm void IRQ_Handler(void)
{
}
	Not supported
	NOTE: In GCC/LLVM(RISCV), interrupt functions are just normal functions.
	Example:
	void IRQ_Handler(void)
	{
	}

2.3 pragmas

表 2-3 pragmas 差异

IAR	毕昇编译器
#pragma bitfield={reversed|default|disjoint_types|joined_types|reversed_disjoint_types Use this pragma directive to control the layout of bitfield members.	Not supported. NOTE: In GCC/LLVM, the equivalent of "bitfield" can be achieved using the __attribute__((packed)) attribute. You can use various combinations of the packed attribute, scalar_storage_order attribute, and other GCC-specific attributes such as aligned and packed_struct. For example, to create a bitfield with reversed bit order in GCC struct my_bitfield { uint8_t b0 : 1; uint8_t b1 : 1; uint8_t b2 : 1; uint8_t b3 : 1; uint8_t b4 : 1; uint8_t b5 : 1; uint8_t b6 : 1; uint8_t b7 : 1; } __attribute__((packed, scalar_storage_order("big-endian")));|
#pragma cstat_disable **#pragma cstat_enable** **#pragma cstat_restore** **#pragma cstat_suppress** Security standard requirements (MISRA, CWE and CERT C/C++), in conjunction with the C-STAT tool, are used to scan code blocks for control.	Not supported NOTE: These paragrams only control the code for tool scanning and inspection, and do not affect the code logic and can be deleted directly.

IAR	毕昇编译器

#pragma calls=arg[, arg...]
The function set declared can be called by subsequent statements, and this information can be used by the linker to calculate stack usage.
Example:
void Fun1(), Fun2();
void Caller(void (*fp)(void)) {
#pragma calls = Fun1, Fun2, "Cat1"
(*fp)(); // Can call Fun1, Fun2, and all
// functions in category "Cat1"
}
	Not supported
	NOTE: Stack statistics can be performed directly through the IDE, so these pragrams can be removed.

#pragma data_alignment=expression Gives a variable a higher (more strict) alignment.	**__attribute__ ((aligned (n)))** Example: typedef int more_aligned_int __attribute__ ((aligned (8)));
#pragma no_bounds **#pragma default_no_bounds** **#pragma define_without_bounds** C-SPY Debug Control Related	Not supported. NOTE: These paragrams are dedicated for C-SPY debugging, do not affect the code logic, and can be removed directly
#pragma error message Signals an error while parsing.	#pragma message llvm also supports the following features: **#pragma GCC error message**
#pragma function_category=category[, category...] Declares function categories for stack usage analysis.	Not supported. NOTE: Stack statistics can be performed directly through the IDE, so these pragrams can be removed.
#pragma call_graph_root[=category] Use this pragma directive to specify that, for stack usage analysis purposes, the immediately following function is a call graph root. Example: #pragma call_graph_root="interrupt"	Not supported. NOTE: Stack statistics can be performed directly through the IDE, so these pragrams can be removed.

IAR	毕昇编译器
#pragma include_alias ("orig_header" , "subst_header") Specifies an alias for an include file. Example： #pragma include_alias (<stdio.h> , <C:\MyHeaders\stdio.h>) #include <stdio.h>	Not supported. NOTE: In GCC/LLVM, the '-include' option can be used to include a specified header file.
#pragma inline[=forced|=never|=no_body|=forced_no_body] 1. Use #pragma inline to advise the compiler that the function defined immediately afterthe directive should be inlined according to C++ inline semantics. 2. Specifying #pragma inline=forced or #pragma inline=forced_no_body will always inline the defined function.	**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 location={address|register|NAME} Specifies the absolute address of a variable, places a variable in a register, or places groups of functions or variables in named sections.	**#define <var_name> (*(volatile char foo*) <abs add>)**
#pragma no_stack_protect Disable function stack protection feature.	**__attribute__((no_stack_protector))**
#pragma optimize=[goal][level][vectorize][disable] Control code optimization type and optimization level, where "global" can choose "size", "balanced", "speed", "no_size_constraints", and "disable" can choose "no_code_motion", "no_cse", "no_inline", "no_relaxed_fp", etc.	**#pragma optimize("[optimization-list]", on | off)**

IAR	毕昇编译器
#pragma __printf_args **#pragma __scanf_args** Check if the format string parameters for functions like scanf/printf are correct.	**__attribute__((format (archetype, string-index, first-to-check)))** The format attribute specifies that a function takes printf, scanf, strftime or strfmon style arguments which should be type-checked against a format string. Example: __attribute__ ((format (printf, 2, 3)))
#pragma public_equ="symbol",value Defines a public assembler label and gives it a value. Example: #pragma public_equ="MY_SYMBOL",0x123456	1. Method one: -defsym sym=value 2. Method two: int foo asm ("myfoo") = 2;
#pragma required=symbol Ensure that the symbol is retained in the link output even if it is not called.	**__attribute__((used))**

#pragma section_prefix="prefix"
All section names automatically have a prefix added, @notation or the #pragma location directive takes effect.
	Not supported
	NOTE: One way is to use the linker script to specify the section prefix.
	Example:
	int foo __attribute__((section("MYSECTION")));
	int foo __attribute__((section("MYSECTION")));
	int foo __attribute__((section("MYSECTION")));
	int foo __attribute__((section("MYSECTION")));
	int foo __attribute__((section("MYSECTION"));

#pragma section="NAME" Put a certain function into a segment named NAME. segment: same as section Example: pragma section="MYSECTION"	**__attribute__ ((section (sectionname)));** Example: char foo __attribute__ ((section ("MYSECTION")));
#pragma unroll=n Specify that the loop following immediately after the directive should be unrolled and that the unrolled loop should have n copies of the loop body	**#pragma unroll n**
#pragma vectorize [= never] Use this pragma directive to enable or disable generation of vector instructions for the loop that follows immediately after the pragma directive.	**#pragma clang loop vectorize(enable/disable)**

2.4 内建函数

表 2-4 内建函数差异

IAR	毕昇编译器
arm_acle.h ACLE (Arm C Language Extensions) intrinsic functions in an application.	Not supported. NOTE: arm_acle.h is specific to ARM architecture and provides functions for ARM processors that are not available on other architectures.
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/LLVM for RISC-V architecture.

3 汇编器

3.1 命令选项

表 3-1 汇编器命令选项差异

IAR	毕昇编译器
-B Macro execution information	**-a[cdhlmns]** Turn on listings, in any of a variety of ways: **-am** include macro expansions.
-Dsymbol[=value] Defines a symbol	**--defsym sym=value** Define the symbol sym to be value before assembling the input file.
-Enumber Maximum number of errors	**-ferror-limit=value**
-f filename Extends the command line	**@file** Read command-line options from file.
-Iprefix Includes paths	**-I path** Use this option to add a path to the list of directories as searches for files specified in .include directives.
-L prefix Lists to prefixed source name	**–a** Generates list file.
-l filename Lists to named file	**–a=file** Generates list file.=file, set the name of the listing file.

IAR	毕昇编译器
-Oprefix Sets object filename prefix.	Not supported NOTE: But using -o option you can specify the directory path and file name. it is important to specify the file name in this case. Default file name is not assigned.
-o filename Sets object filename	**-o filename** Sets object filename.
-plines Lines/page	Not supported. NOTE: Can be replaced by assembler directive ” .psize , ” which defines the number of lines and columns (optional) to be printed per page.
-r Generates debug information	**-g** Generates debugging information.
-S Set silent operation	Not supported NOTE: If you use the `-W' and `--no-warn' options, no warnings are issued, but errors are still reported.
-Usymbol Undefines a symbol	Not supported. NOTE: Can be replaced by assembler directive “.purgem name”, which undefines the macro name.
-w[string][s] Disables warnings	**-W –no_warn** Switch all warnings off.
-x Generate a cross-reference list	Not supported. NOTE: In GCC, use the `-Wl,-Map=<filename>` option to generate a linker map file, which contains information about the cross-reference table.

3.2 内联汇编

IAR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

IAR	毕昇编译器

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;
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.
Example:
int Add(int term1, int term2) {
int sum;
asm("add %2, %1, %0 \n"
: "=r"(sum)
: "r"(term1), "r"(term2));
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 汇编指示

表 3-3 汇编指示差异

IAR	毕昇编译器
$ **#include** Include file	**.include "file"**
#if Assembles instructions if a condition is true.	**.if**
#ifdef Assembles instructions if a symbol is defined.	**.ifdef**

IAR	毕昇编译器
#ifndef Assembles instructions if a symbol is undefined.	**.ifndef symbol**
#elif Introduces a new condition in a #if…#endif block	**.elseif**
#else Assembles instructions if a condition is false.	**.else**
#endif The ending assembly directive corresponding to #if, #ifdef, or #ifndef is #endif.	**.endif**
#error Define error	**.err , .error "string"**
#message Standard output generates a message.	Not supported NOTE: .err (could be alternative solution) If as assembles a .err directive, it will print an error message and, unless the `-Z' option was used, it will not generate an object file.
#undef Undefine micro	**.purgem name**
/*comment */ C-Style comment	**/*comment */**
// C-Style comment	Line comment use by character '#'.
label ALIAS expr Assigns a permanent value local to a module	To define a local symbol, write a label of the form `N:' (where N represents any positive integer). Example: 1: branch 1f 2: branch 1b 1: branch 2f 2: branch 1b

IAR	毕昇编译器
ALIGN align [,value2] Aligns the location counter by inserting zero-filled bytes.	**.align** Pad the location counter (in the current subsection) to a particular storage boundary.
label ASSIGN expr Assigns a temporary value.	**.equ symbol, expression** This directive sets the value of symbol to expression. It is synonymous with `.set'
BLKA Allocates space for 24-bit data objects.	**.comm symbol , length** .comm declares a common symbol named symbol. When linking, a common symbol in one object file may be merged with a defined or common symbol of the same name in another object file.
BLKB Allocates space for 8-bit data objects.	
BLKD Allocates space for 64-bit data objects.	
BLKF Allocates space for 32-bit data objects.	
BLKL Allocates space for 32-bit data objects.	
BLKW Allocates space for 16-bit data objects.	
DS8 Allocates space for 8-bit data objects.	
DS16 Allocates space for 16-bit data objects.	
DS24 Allocates space for 24-bit data objects.	
DS32 Allocates space for 32-bit data objects.	

IAR	毕昇编译器
BYTE Generates 8-bit byte constants, including strings.	**.byte expressions** .byte expects zero or more expressions, separated by commas. Each expression is assembled into the next byte.
COL Sets the number of columns per page.	**.psize lines , columns** Use this directive to declare the number of lines--and, optionally, the number of columns--to use for each page, when generating listings.
ELSE Assembles instructions if a condition is false.	**.else** .else is part of the as support for conditional assembly.
END Terminates the assembly of the last module in a file.	**.end** .end marks the end of the assembly file. as does not process anything in the file past the .end directive.
ENDIF Ends an IF block.	**.endif** .endif is part of the as support for conditional assembly; it marks the end of a block of code that is only assembled conditionally.
ENDM Ends a macro definition.	**.endm** The commands .macro and .endm allow you to define macros that generate assembly output.
label EQU expr	**.equ symbol, expression**
Assigns a permanent value local to a module.	**.set symbol, expression** This directive sets the value of symbol to expression.
EXITM Exits prematurely from a macro.	**.exitm** Exit early from the current macro definition.
EXTERN Imports an external symbol.	**.extern** .extern is accepted in the source program for compatibility with other assemblers, but it is ignored. “as “ treats all undefined symbols as external.

IAR	毕昇编译器
FLOAT Generates 32-bit floating point constants.	**.float flonums** **.single flonums** This directive assembles zero or more flonums, separated by commas. The exact kind of floating point numbers emitted depends on how as is configured.
IF Assembles instructions if a condition is true.	**.if absolute expression**
LSTCND Controls conditional assembler listing.	Not supported Some of the list control directives supported are. **.list, .nolist** Control (in conjunction with the .nolist directive) whether or not assembly listings are generated. **.line line-number** Change the logical line number. line-number must be an absolute expression.
LSTCOD Controls multi-line code listing.	
LSTEXP Controls the listing of macro generated lines.	
LSTMAC Controls the listing of macro definitions.	
LSTOUT Controls assembler-listing output.	
LSTPAG Controls the formatting of output into pages.	
LSTREP Controls the listing of lines generated by repeat directives.	
LSTXRF Generates a cross-reference table.	
LWORD Define 32-bit constant	**.long expressions**
MACRO Define macro	**.macro**
ORG Sets the location counter to a new absolute address.	Not supported. NOTE: The instruction .org in GNUC sets the address relative to the start of the actual section.

IAR	毕昇编译器
PAGE Generates a new page.	Not supported NOTE: **.eject** .eject force a page break at this point, when generating assembly listings.
PAGSIZ Sets the number of lines per page.	**.psize lines , columns** The "psize" directive in GNU assembler is used to specify the size of the page in terms of lines and columns for the listing file generated by the assembler. The "lines" parameter specifies the number of lines per page, while the "columns" parameter specifies the number of columns per line.
RSEG Begins a relocatable section.	**.section name** Use the .section directive to assemble the following code into a section named name.
REPT Assembles instructions a specified number of times.	**.rept count** Repeat the sequence of lines between the .rept directive and the next .endr directive times.
WORD Define 16-bit constant.	**.word expressions**

4 链接器

4.1 命令选项

表 4-1 链接器命令选项差异

IAR	毕昇编译器
--define_symbol Defines symbols that can be used by the application.	**--defsym=symbol=expression**
--enable_stack_usage Use this option to enable stack usage analysis.	Not supported NOTE: Stack statistics can be performed directly through the IDE, it can be replaced by using the IDE.
-o file **--output file** Specifies the name of the output file.	**-o output**
-Ipathname Specifies a list of paths used to search for ARM standard C/C++ libraries.	**-L searchdir**
--map {filename|directory} Produces a map file.	-M
--entry Specifies the unique entry point of the image.	-e entry

IAR	毕昇编译器
-f filename **--f filename** Use this option to make the linker read command line options from the named file.	@file
--remarks Indiacate the least severe diagnostic messages are called remarks.	Not supported. NOTE: In GNU linker, the "-M" or "--print-map" option can be used to generate a map file that contains information about the memory layout of the program, including the location and size of each section and symbol.
-Z type segment=start Sets load and execution addresses of the region containing the readonly output section.	**--section-start=sectionname=org**
--strip Omit all symbol information from the output file.	**-s**
--keep symbol[,symbol1,...] The linker keeps a symbol only if it is needed by your application.	**-u**
--warnings_are_errors Warnings are treated as errors.	**--fatal-warnings** **--no-fatal-warnings**
--no_library_search Use this option to disable the automatic runtime library search.	**-nostdlib**
--no_locals Delete all local symbols.	**-x**
--no_fragments Reduce empty areas of boundary alignment.	Not supported. NOTE: In GCC/LLVM linker, Use " --no-gc-sections" option to replace "no_fragment" option in IAR linker.
--config filename Specifies link script filename.	**-T scriptfile**