TI cl2000 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TI cl2000编译器切换到毕昇编译器进行开发。本文主要介绍TI cl2000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TI cl2000编译器切换到毕昇编译器（RISCV）进行开发。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TI cl2000和毕昇编译器对比相关的差异。

图 1-1 TI cl2000 编译器

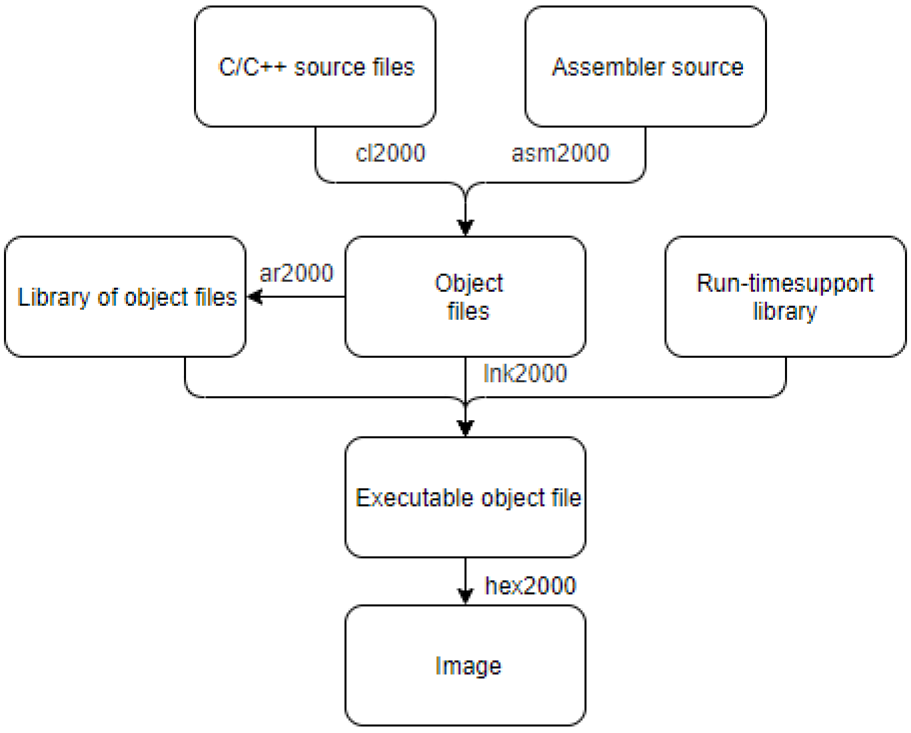

图 **1-2** 毕昇编译器

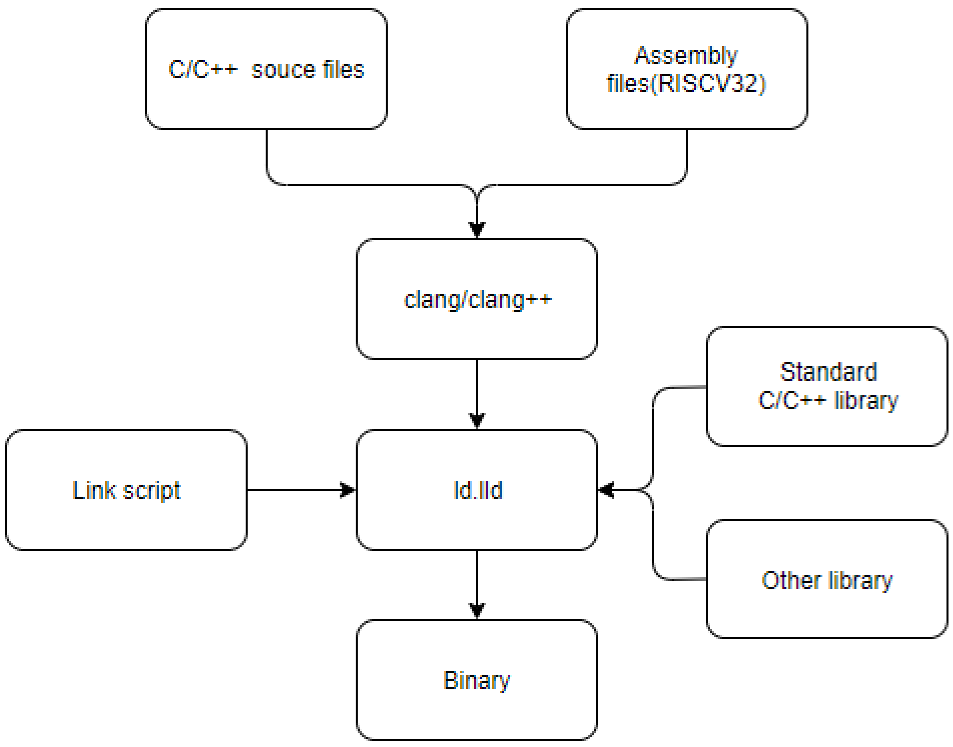

表 **1-1** 编译器对比

工具	CL2000	毕昇编译器
Compiler	cl2000 集成版本 TMS320C2000 C/C++ Parser v22.6.0.LTS 支持标准C89/C99/C11/C++03	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	cl2000(asm2000) 汇编语言对应C28x汇编语言指令要求	1.增加“-lld”选项使用clang汇编 2.riscv32-linux-musl-as
Linker	cl2000(lnk2000) 满足链接命令文件语法要求	1.增加“-lld”选项使用ld.lld链接器 2.riscv32-linux-musl-ld
Archiver	ar2000 将多个单独的文件合并为一个存档文件	1.llvm-ar 2.riscv32-linux-musl-ar 用于创建、修改和提取静态库文件

工具	CL2000	毕昇编译器
C++ Demangler	dem2000 C++ 名称还原器是一种调试辅助工具，其将检测到的每个已改编的名称转换为其在 C++ 源代码中找到的原始名称	1.llvm-cxxfilt 2.riscv32-linux-musl-c++filt 将C++符号进行解码，将其转换为易于阅读的形式
Disassembler	dis2000 接受目标文件或可执行文件作为输入，并将反汇编的目标代码写入标准输出或指定文件	1.llvm-objdump 2.riscv32-linux-musl-objdump 查看目标程序中的段信息和调试信息，也可以用来对目标程序进行反汇编
Hex Converter	hex2000 可以将可执行目标文件转换为适合输入到EPROM 编程器的格式	1.llvm-objcopy 2.riscv32-linux-musl-objcopy 可以对最后生成的程序文件进行一定的编辑、转换
Name Utility	nm2000 输出目标文件定义和引用的符号	1.llvm-nm 2.riscv32-linux-musl-nm 列出目标文件中的符号
Strip Utility	strip2000 从目标文件中删除符号表和调试信息	1.llvm-strip 2.riscv32-linux-musl-strip 去除目标文件中的一些符号表等信息

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

CL2000	毕昇编译器
--silicon_version=28 Specifies TMS320C28x architecture. The default (and only value accepted) is 28. This option is no longer required.	**-march=rv32im[fc]_xhimideer** Choose the extension for architecture. 'f' means float, 'c' means compress.
--float_support={ fpu32 | fpu64 | softlib } Specifies use of TMS320C28x 32-bit or 64-bit hardware floating-point support. The default is softlib. Use this option only if the targethardware provides this functionality.	**-mabi=ilp32[f]** It will automatic choose according march, can use –mabi=ilp32/ilp32f. The default is –mabi=ilp32.
--opt_level=off Disables all optimization (default).	**-O0** Disables optimization,This is the default.

CL2000	毕昇编译器
-On --opt_level=n Level 0 (-O0) optimizes register usage only. Level 1 (-O1) uses Level 0 optimizations and optimizes locally. Level 2 (-O2) uses Level 1 optimizations and optimizes globally. Level 3 (-O3) uses Level 2 optimizations and optimizes the file. Level 4 (-O4) uses Level 3 optimizations and performs link-time optimization.	**-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.
-ms --opt_for_space=n Controls code size on four levels (0, 1, 2, and 3)	**-Os** Optimize for space rather than speed.
-mf --opt_for_speed[=n] Controls the tradeoff between size and speed (0-5 range). If this option is specified without n, the default value is 4. If this option is notspecified, the default setting is 2.	**-Ofast** Optimize for speed disregarding exact standards compliance.
-oi --auto_inline=[size] Sets automatic inlining size (--opt_level=3 only). If size is not specified, the default is 1.	Not supported.
--disable_inlining Prevents any inlining from occurring.	**-fno-inline** Do not expand any functions inline apart from those marked with the always_inline attribute. This is the default when not optimizing.

CL2000	毕昇编译器
--fp_single_precision_constant Causes all unsuffixed floating-point constants to be treated as single precision values instead of as double-precision constants.	Not supported.
-onn --gen_opt_info=n Level 0 (-on0) disables the optimization information file. Level 1 (-on2) produces an optimization information file. Level 2 (-on2) produces a verbose optimization information file.	Not supported.
--pm --program_level_compile Combines source files to perform program-level optimization.	Not supported.
-ma --aliased_variables Notifies the compiler that addresses passed to functions may be modified by an alias in the called function.	Not supported.
--symdebug:dwarf_version=2|3|4 Specifies the DWARF format version. The default version is 3 for the COFF ABI and 4 for EABI.	Not supported.
--symdebug:none Disables all symbolic debugging.	Not supported. Disabling debugging is defualt.
--symdebug:profile_coff Enables profiling using the alternate STABS debugging format. STABS is supported only for the COFF ABI.	Not supported.
--preinclude=filename Includes filename at the beginning of compilation.	**-include <file>** Include the contents of <file> before other files.
-z --run_linker Causes the linker to be invoked from the compiler command line.	By default, the compiler does invoke the linker.
-n --skip_assembler Compiles C/C++ source file , producing an assembly language output file. The assembler is not run and no object file is produced.	**-S** Compile only; do not assemble or link.

CL2000	毕昇编译器
--c89 Processes C files according to the ISO C89 standard.	**-std=c89 or -std=c90** Support all ISO C90 programs (certain GNU extensions that conflict with ISO C90 are disabled). Same as ‘-ansi’ for C code.
--c99 Processes C files according to the ISO C99 standard.	**-std=c99** ISO C99.
--c11 Processes C files according to the ISO C11 standard.	**-std=c11** ISO C11, the 2011 revision of the ISO C standard.
--c++03 Processes C++ files according to the ISO C++03 standard.	**-std=c++03** The 1998 ISO C++ standard plus the 2003 technical corrigendum and some additional defect reports. Same as ‘-ansi’ for C++ code.
-fg --cpp_default Processes all source files with a C extension as C++ source files.	**-x c++**
--exceptions Enables C++ exception handling.	**-fexceptions** Enable exception handling
--pending_instantiations=# Specify the number of template instantiations that may be in progress at any given time. Use 0 to specify an unlimited number.	Not supported.
--rtti Enables C++ run-time type information (RTTI).	**-frtti** Generate run time type descriptor information
-ps --strict_ansi Enables strict ANSI/ISO mode (for C/C++, not for K&R C). In this mode, language extensions that conflict with ANSI/ISO C/C++ are disabled. In strict ANSI/ISO mode, most ANSI/ISO violations are reported as errors. Violations that are considered discretionary may be reported as warnings instead.	**-pedantic-errors** Issue all the warnings demanded by strict ISO C and ISO C++ as error; reject all programs that use forbidden extensions, and some other programs that do not follow ISO C and ISO C++.

CL2000	毕昇编译器
-ppd --preproc_dependency[=filename] Performs preprocessing only, but instead of writing preprocessed output, writes a list of dependency lines suitable for input to a standard make utility.	**-M -MF <filename>** Instead of outputting the result of preprocessing, output a rule suitable for make describing the dependencies of the main source file. The preprocessor outputs one make rule containing the object file name for that source file, a colon, and the names of all the included files, including those coming from '-include' or '-imacros' command-line options.
--preinclude=filename Includes filename at the beginning of compilation.	**-include <file>** Include the contents of <file> before other files.
-ppi --preproc_includes[=filename Performs preprocessing only, but instead of writing preprocessed output, writes a list of files included with the #include directive.	**-H** Print the name of each header file used, in addition to other normal activities. Each name is indented to show how deep in the '#include' stack it is.
-ppm --preproc_macros[=filename] Performs preprocessing only. Writes list of predefined and user defined macros to a file with the same name as the input but with a .pp extension	**-E -dM** Instead of the normal output, generate a list of '#define' directives for all the macros defined during the execution of the preprocessor, including predefined macros.
-ppo --preproc_only Performs preprocessing only. Writes preprocessed output to a file with the same name as the input but with a .pp extension	**-E -o <file>** Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code.
-ppc --preproc_with_comment Performs preprocessing only. Writes preprocessed output, keeping the comments, to a file with the same name as the input but witha .pp extension	**-E -CC/-C** Do not discard comments in macro expansions.
-ppl --preproc_with_line Performs preprocessing only. Writes preprocessed output with linecontrol information (#line directives) to a file with the same name as the input but with a .pp extension.	Writing preprocessed output with linecontrol information is the default behavior for RISCV32 compiler preprocess.

CL2000	毕昇编译器
--advice:performance[=all,none] Provides advice on ways to improve performance. Default is all.	Not supported.
--compiler_revision Prints out the compiler release revision and exits.	**--version** Print version information
-pdse --diag_error=num Categorizes the diagnostic identified by num as an error.	**-Werror=<flag>** 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.
-pds --diag_suppress=num Suppresses the diagnostic identified by num.	**-Wno-<flag>** Specific warning options has a negative form beginning ‘-Wno-’ to turn off warnings.
-pdsw --diag_warning=num Categorizes the diagnostic identified by num as a warning.	**-W<flag>** You can request many specific warnings with options beginning with ‘-W’, for example '-Wimplicit’ to request warnings on implicit declarations.
--diag_wrap={on|off} Wrap diagnostic messages (default is on).	**-fmessage-length=<number>** Limit diagnostics to <number> characters per line. 0 suppresses line-wrapping.
-pdew --emit_warnings_as_errors Treat warnings as errors.	**-Werror** Treat all warnings as errors.
-pdw --no_warnings Suppresses diagnostic warnings (errors are still issued).	**-w --no-warnings** Suppress warnings.
-q --quiet Suppresses progress messages (quiet).	This is the default behavior for RISCV32 compiler.
--set_error_limit=num Sets the error limit to num. The compiler abandons compiling after this number of errors. (The default is 100.)	Not supported. RISCV32 compiler abandons compiling if error encountered.

CL2000	毕昇编译器
-pdv --verbose_diagnostics Provides verbose diagnostic messages that display the original source with line-wrap.	This is the default behavior for RISCV32 compiler.
-pdf --write_diagnostics_file Generates a diagnostic message information file.	Use unix redirection operator instead compile cmd **&>**file.
--gen_data_subsections={on|off} Place all aggregate data (arrays, structs, and unions) into subsections. This gives the linker more control over removing unused data during the final link step. See the link to the right for details about the default setting.	**-fdata-sections** Place data items into their own section.
-mo --gen_func_subsections={on|off} Puts each function in a separate subsection in the object file. If this option is not used, the default is off. See the link to the right for details about the default setting.	**-ffunction-sections** Place each function into its own section.

--entry_hook[=name]
Enables entry hooks.
--exit_hook[=name]
Enables exit hooks.
	-finstrument-functions
	Generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions are called with the address of the current function and its call site.
	void __cyg_profile_func_enter (void *this_fn, void *call_site);
	void __cyg_profile_func_exit (void *this_fn, void *call_site);

-fa --asm_file=filename Identifies filename as an assembly source file regardless of its extension. By default, the compiler and assembler treat .asm files as assembly source files. **-ea --asm_extension=[.]extension** Sets a default extension for assembly source files.	**-x assembler**

CL2000	毕昇编译器
-fc --c_file=filename Identifies filename as a C source file regardless of its extension. By default, the compiler treats .c files as C source files. **-ec --c_extension=[.]extension** Sets a default extension for C source files.	**-x c <file>**
-fp --cpp_file=filename Identifies filename as a C++ file, regardless of its extension. By default, the compiler treats .C, .cpp, .cc and .cxx files as a C++ files. **-ep --cpp_extension=[.]extension** Sets a default extension for C++ source files	**-x c++**
-fo --obj_file=filename Identifies filename as an object code file regardless of its extension. By default, the compiler and linker treat .obj files as object code files, including both *.c.obj and *.cpp.obj files.	This is the default behavior for RISCV32 compiler.
-fs -asm_directory=directory Specifies an assembly file directory. By default, the compiler uses the current directory.	**-S -o <directory>/<file>**
-fr --obj_directory=directory Specifies an object file directory. By default, the compiler uses the current directory **-fe --output_file=filename** Specifies a compilation output file name; can override --obj_directory.	**-o <directory>/<file>**
--pp_directory=dir Specifies a preprocessor file directory. By default, the compiler uses the current directory	**-E -o <directory>/<file>**

2.2 扩展关键字

表 2-2 扩展关键字差异

TI C/C++ Compiler	毕昇编译器
__cregister The compiler extends the C/C++ language by adding the cregister keyword to allow high level language access to control registers. This keyword is available in normal mode, but not in strict ANSI/ISO mode (using the --strict_ansi compiler option).	Not supported.

__interrupt
The compiler extends the C/C++ language by adding the __interrupt keyword, which specifies that a function is treated as an interrupt function. This keyword is an IRQ interrupt. The alternate keyword, "interrupt", may also be used except in strict ANSI C or C ++ modes.
	Not supported.
	NOTE: In GCC(RISCV), interrupt functions are just normal functions.
	Example：
	void IRQ_Handler(void)
	{
	}

__asm The C/C++ compiler can embed assembly language instructions or directives directly into the assembly language output of the compiler. This capability is an extension to the C/C++ language implemented through the __asm keyword.	**asm** The asm keyword allows you to embed assembler instructions within C code.When writing code that can be compiled with ‘-ansi’ and the various ‘-std’ options, use __asm__ instead of asm

2.3 pragmas

表 **2-3** pragmas 差异

TI C/C++ Compiler	**毕昇编译器**
The syntax for the CALLS pragma in C is as follows: **#pragma CALLS** (calling_function, function_1, function_2, ..., function_n) The syntax for the CALLS pragma in C++ is: **#pragma CALLS** (function_1_mangled_name, ..., function_n_mangled_name) The CALLS pragma specifies a set of functions that can be called indirectly from a specified calling function.	Not Supported.
The syntax of the pragma in C is: **#pragma CLINK** (symbol) The syntax of the pragma in C++ is: **#pragma CLINK** The CLINK pragma can be applied to a code or data symbol. It causes a .clink directive to be generated into the section that contains the definition of the symbol. The .clink directive tells the linker that a section is eligible for removal during conditional linking. Thus, if the section is not referenced by any other section in the application being compiled and linked, it will not be included in the resulting output file.	Not Supported. Use -fdata-sections -ffunction-sections -Wl,--gc-sections to remove unused sections.
The syntax of the pragma in C is: **#pragma CODE_ALIGN** (func , constant) The syntax of the pragma in C++ is: **#pragma CODE_ALIGN** (constant) The CODE_ALIGN pragma aligns func along the specified alignment. The alignment constant must be a power of 2.	**__attribute__ ((aligned (alignment)))** This attribute specifies a minimum alignment for the function, measured in bytes. example: void __attribute__((aligned(alignment))) fun(void) {}

TI C/C++ Compiler	毕昇编译器
The syntax of the pragma in C is: **#pragma CODE_SECTION** (symbol , " section name ") The syntax of the pragma in C++ is: **#pragma CODE_SECTION** (" section name ") The CODE_SECTION pragma allocates space for the symbol in C, or the next symbol declared in C++, in a section named section name.	**__attribute__ ((section ("section-name")))** The section attribute specifies that a function lives in a particular section. example: void __attribute__((aligned("bar"))) foobar(void) {} puts the function foobar in the bar section.
The syntax of the pragma in C is: **#pragma DATA_ALIGN** (symbol , constant) The syntax of the pragma in C++ is: **#pragma DATA_ALIGN** (constant) The DATA_ALIGN pragma aligns the symbol in C, or the next symbol declared in C++, to an alignment boundary. The alignment boundary is the maximum of the symbol's default alignment value or the value of the constant in bytes. The constant must be a power of 2. The maximum alignment is 32768.	**__attribute__ ((aligned (alignment)))** The aligned attribute can also be used for variables and fields. This attribute specifies a minimum alignment for the variable or structure field, measured in bytes.
The syntax of the pragma in C is: **#pragma DATA_SECTION** (symbol , " section name ") The syntax of the pragma in C++ is: **#pragma DATA_SECTION** (" section name ") The DATA_SECTION pragma allocates space for the symbol in C, or the next symbol declared in C++, in a section named section name.	**__attribute__ ((section ("section-name")))** Normally, the compiler places the objects it generates in sections like data and bss. Sometimes, however, you need additional sections, or you need certain particular variables to appear in special sections, for example to map to special hardware. The section attribute specifies that a variable (or function) lives in a particular section.

TI C/C++ Compiler	毕昇编译器
#pragma diag_ xxx [=]num[, num_2, num_3...] The pragmas can be used to control diagnostic messages in the same ways as the corresponding command line options.	**#pragma clang diagnostic kind option** Example： #pragma clang diagnostic warning "-Wformat" #pragma clang diagnostic error "-Wformat" #pragma clang diagnostic ignored "-Wformat" kind is ‘error’ to treat this diagnostic as an error, ‘warning’ to treat it like a warning (even if ‘-Werror’ is in effect), or ‘ignored’ if the diagnostic is to be ignored. option is a double quoted string that matches the command-line option.
#pragma FAST_FUNC_CALL（func） The FAST_FUNC_CALL pragma, when applied to a function, generates a TMS320C28x FFC instruction to call the function instead of the CALL instruction.	Not Supported.
#pragma FORCEINLINE The FORCEINLINE pragma can be placed before a statement to force any function calls made in that statement to be inlined. It has no effect on other calls to the same functions.	**_attribute__ ((flatten))** Generally, inlining into a function is limited. For a function marked with this attribute, every call inside this function is inlined, if possible. Whether the function itself is considered for inlining depends on its size and the current inlining parameters.
#pragma FORCEINLINE_RECURSIVE The FORCEINLINE_RECURSIVE can be placed before a statement to force any function calls made in that statement to be inlined along with any calls made from those functions. That is, calls that are not visible in the statement but are called as a result of the statement will be inlined.	Not Supported.

TI C/C++ Compiler	毕昇编译器
The syntax of the pragma in C is: **#pragma FUNC_ALWAYS_INLINE** (func) The syntax of the pragma in C++ is: **#pragma FUNC_ALWAYS_INLINE** The FUNC_ALWAYS_INLINE pragma instructs the compiler to always inline the named function.	**__attribute__ ((always_inline))** Generally, functions are not inlined unless optimization is specified. For functions declared inline, this attribute inlines the function independent of any restrictions that otherwise apply to inlining. Failure to inline such a function is diagnosed as an error. Note that if such a function is called indirectly the compiler may or may not inline it depending on optimization level and a failure to inline an indirect call may or may not be diagnosed.
The syntax of the pragma in C is: **#pragma FUNC_CANNOT_INLINE** (func) The syntax of the pragma in C++ is: **#pragma FUNC_CANNOT_INLINE** The FUNC_CANNOT_INLINE pragma instructs the compiler that the named function cannot be expanded inline. Any function named with this pragma overrides any inlining you designate in any other way, such as using the inline keyword.	**__attribute__ ((noinline))** This function attribute prevents a function from being considered for inlining. If the function does not have side-effects, there are optimizations other than inlining that cause function calls to be optimized away, although the function call is live.
The syntax of the pragma in C is: **#pragma FUNC_EXT_CALLED** (func) The syntax of the pragma in C++ is: **#pragma FUNC_EXT_CALLED** When you use the --program_level_compile option, the compiler uses program-level optimization. When you use this type of optimization, the compiler removes any function that is not called, directly or indirectly, by main(). You might have C/C++ functions that are called instead of main().	**__attribute__ ((used))** This attribute, attached to a function, means that code must be emitted for the function even if it appears that the function is not referenced. This is useful, for example, when the function is referenced only in inline assembly. When applied to a member function of a C++ class template, the attribute also means that the function is instantiated if the class itself is instantiated.

TI C/C++ Compiler	毕昇编译器

The syntax of the pragma in C is:
#pragma FUNCTION_OPTIONS (func , " additional options ")
The syntax of the pragma in C++ is:
#pragma FUNCTION_OPTIONS(" additional options ")
The FUNCTION_OPTIONS pragma allows you to compile a specific function in a C or C++ file with additional command-line compiler options. The affected function will be compiled as if the specified list of options appeared on the command line after all other compiler options. In C, the pragma is applied to the function specified. In C++, the pragma is applied to the next function.
	#pragma clang optimize off
	Extensions for selectively disabling optimization.
	Example：
	#pragma clang optimize off
	// This function will be decorated with optnone.
	int foo() {
	// ... code
	}

The syntax of the pragma in C is: **#pragma INTERRUPT** (func) The syntax of the pragma in C++ is: **#pragma INTERRUPT** void func (void) The INTERRUPT pragma enables you to handle interrupts directly with C code. In C, the argument func is the name of a function. In C++, the pragma applies to the next function declared.	Not Supported.
The syntax of the pragma in C is: **#pragma LOCATION**(x , address) int x The syntax of the pragmas in C++ is: **#pragma LOCATION**(address) int x The compiler supports the ability to specify the run-time address of a variable at the source level.	Not Supported.

TI C/C++ Compiler	毕昇编译器
#pragma MUST_ITERATE (min, max, multiple) The MUST_ITERATE pragma specifies to the compiler certain properties of a loop. When you use this pragma, you are guaranteeing to the compiler that a loop executes a specific number of times or a number of times within a specified range.	Not Supported.
The syntax of the pragmas in C is: **#pragma NOINIT** (x) int x ; **#pragma PERSISTENT** (x) int x =10; The syntax of the pragmas in C++ is: **#pragma NOINIT** int x ; **#pragma PERSISTENT** int x =10; Global and static variables are zero-initialized by default. However, in applications that use non-volatile memory, it may be desirable to have variables that are not initialized. The NOINIT pragma may be used only with uninitialized variables. It prevents such variables from being set to 0 during a reset.The PERSISTENT pragma may be used only with statically-initialized variables. It prevents such variables from being initialized during a reset.	Not Supported.
#pragma NOINLINE The NOINLINE pragma can be placed before a statement to prevent any function calls made in that statement from being inlined. It has no effect on other calls to the same functions.	Not Supported.

TI C/C++ Compiler	毕昇编译器
The syntax of the pragma in C is: **#pragma NO_HOOKS** (func) The syntax of the pragma in C++ is: **#pragma NO_HOOKS** The NO_HOOKS pragma prevents entry and exit hook calls from being generated for a function.	**__attribute__ ((no_instrument_function))** If '-finstrument-functions' is given, profiling function calls are generated at entry and exit of most user-compiled functions. Functions with this attribute are not so instrumented.
#pragma once The once pragma tells the C preprocessor to ignore a #include directive if that header file has already been included. For example, this pragma may be used if header files contain definitions, such as struct definitions, that would cause a compilation error if they were executed more than once.	Not Supported.
The syntax of the pragma in C is: **#pragma RETAIN** (symbol) The syntax of the pragma in C++ is: **#pragma RETAIN** The RETAIN pragma can be applied to a code or data symbol. In EABI mode, which assumes that all sections are eligible for removal via conditional linking, this pragma causes a .retain directive to be generated into the section that contains the definition of the symbol. The .retain directive indicates to the linker that the section is ineligible for removal during conditional linking. Therefore, regardless whether or not the section is referenced by another section in the application that is being compiled and linked, it will be included in the output file result of the link.	Not Supported.
#pragma SET_CODE_SECTION (" section name ") **#pragma SET_DATA_SECTION** (" section name ") These pragmas can be used to set the section for all declarations below the pragma.	Not Supported.

TI C/C++ Compiler	毕昇编译器
#pragma UNROLL(n) The UNROLL pragma specifies to the compiler how many times a loop should be unrolled. The optimizer must be invoked (use --opt_level=[1|2|3] or -O1, -O2, or -O3) in order for pragma-specified loop unrolling to take place. The compiler has the option of ignoring this pragma.	Not Supported.
The syntax of the pragma in C is: **#pragma WEAK** (symbol) The syntax of the pragma in C++ is: **#pragma WEAK** The WEAK pragma gives weak binding to a symbol.	**__attribute__ ((weak))** The weak attribute causes the declaration to be emitted as a weak symbol rather than a global. This is primarily useful in defining library functions that can be overridden in user code, though it can also be used with non-function declarations.

2.4 默认行为差异

常见默认行为差异	TI C/C++ Compiler	毕昇编译器
ABI	COFF ABI defaultly，specify '--abi=eabi' option use ELF format.	ELF defaultly.
Invoking the linker	By default, the compiler does not invoke the linker. You can invoke the linker by using the --run_linker (-z) compiler option.	Invoking the linker defaultly. specify '-c' option to instruct compiler do not link, compile and assemble only.
Linkage	static link	Dynamic link defaultly，specify '-static' option to instruct compiler use static link.
Debug	Enables symbolic debugging defaultly.	Disables symbolic debugging defaultly.，specify '-g' option to instruct compiler generating debug information.
Default architecture	TMS320C28x	-march=rv32imcxlinxma_xlinxmb_xlinxmc_xlinxmd -mabi=ilp32

常见默认行为差异	TI C/C++ Compiler	毕昇编译器
Default C++ source language mode	C++03	C++17
Default C source language mode	C89	C17
Predefined Macro	**__PTRDIFF_T_TYPE__** Set to the type of ptrdiff_t. **__SIZE_T_TYPE__** Set to the type of size_t.	**__PTRDIFF_TYPE__** **__SIZE_TYPE__**
Directory Search	• Complier does not scan standard system directories for searching header files and standard libraray defaultly. • You can set C2000_C_DIR environment variable with standard system directories to scan it automatically.	• Complier scan standard system directories for searching header files and standard libraray defaultly. • Specifying '-nostdinc' or '-nostdinc++' options to instruct compiler do not search the standard system directories for header files. • Specifying '-nostdlib' options to instruct compiler do not search the standard system directories for standard system startup files or libraries.

常见默认行为差异	TI C/C++ Compiler	毕昇编译器
Data Types Size	● A byte is 16 bits. ● Size of [signed/unsigned] char is 16 bits, 1 byte. ● Size of _Bool is 16 bits, 1 byte. ● Size of [signed/unsigned] short is 16 bits, 1 byte. ● Size of [signed/unsigned] int is 16 bits, 1 byte. ● Size of [signed/unsigned] long is 32 bits, 2 bytes. ● Size of [signed/unsigned] long long is 64 bits, 4 bytes. ● Size of float is 32 bits, 2 bytes. ● Size of double(COFF) is 32 bits, 2 bytes. ● Size of long double is 64 bits, 4 bytes.	● A byte is 8 bits. ● Size of [signed/unsigned] char is 8 bits, 1 byte. ● Size of _Bool is 16 bits, 1 byte.. ● Size of [signed/unsigned] short is 16 bits, 2 bytes. ● Size of [signed/unsigned] int is 32 bits, 4 bytes. ● Size of [signed/unsigned] long is 32 bits, 4 bytes. ● Size of [signed/unsigned] long long is 64 bits, 8 bytes. ● Size of float is 32 bits, 4 bytes. ● Size of double is 64 bits, 8 bytes. ● Size of long double is 128 bits, 16 bytes.

3 汇编器

3.1 命令选项

表 **3-1** 汇编器命令选项差异

TI C/C++ Compiler	**毕昇编译器**
-aa --absolute_listing Creates an absolute listing.	Not supported.
-ad --asm_define=name[=def] Sets the name symbol.	**--defsym sym=value** Define the symbol sym to be value before assembling the input file.
-apd --asm_dependency Performs preprocessing for assembly files, but instead of writing preprocessed output, writes a list of dependency lines suitable for input to a standard make utility.	**-Wa,--MD FILE** write dependency information in FILE (default none).
-api --asm_includes Performs preprocessing for assembly files, but instead of writing preprocessed output, writes a list of files included with the .include directive.	Not supported.
-al --asm_listing Produces a listing file with the same name as the input file with a .lst extension.	**-Wa,-a[cdhlmns]** Turn on listings, in any of a variety of ways: **-al** include assembly

TI C/C++ Compiler	毕昇编译器
-ax --asm_cross_reference_listing A cross-reference listing shows symbols and their definitions.	Not supported.
-au --asm_undefine=name Undefines the predefined constant name, which overrides any --asm_define options for the specified constant.	Not supported.
--cla_support[=cla0|cla1|cla2] Specifies TMS320C28x Control Law Accelerator (CLA) Type 0, Type 1, or Type 2 support.	Not supported.
-@ --cmd_file=filename Appends the contents of a file to the command line.	**@file** Read command-line options from file.
--float_support={ fpu32 | fpu64 } Assembles code for C28x with 32-bit or 64-bit hardware FPU support.	**-mabi=ilp32f -march=rv32imfc_xhimideer** 32-bit hardware FPU support.
-ahi --include_file=filename Includes the specified file for the assembly module.	Not supported.
-I --include_path=pathname Specifies a directory where the assembler can find files named by the .copy, .include, or .mlib directives.	**-I path** Use this option to add a path to the list of directories as searches for files specified in .include directives.
-lfu=path --lfu_reference_elf=path In order to create a Live Firmware Update (LFU) compatible executable binary, specify the path to a previous ELF executable binary to use as a reference from which to obtain a list of the memory addresses of global and static symbols.	Not supported.
-q --quiet Suppresses the banner and progress information (assembler runs in quiet mode).	Not supported.

TI C/C++ Compiler	毕昇编译器
-g --symdebug:dwarf or --symdebug:none Enables assembler source debugging in the C source debugger(DWARF is on by default).	**-g** Generates debugging information.
--vcu_support[=vcu0|vcu2|vcrc] The vcu0 and vcu2 settings specify there is support for Type 0 or Type 2 of the Viterbi, Complex Math and CRC Unit (VCU). Note that there is no VCU Type 1. The default is vcu0.	Not supported.

3.2 内联汇编

内联汇编的格式如表3-2所示。

表 3-2 内联汇编差异

TI C/C++ Compiler	毕昇编译器

__asm(" assembler text ");
Using __asm is syntactically performed as a call to a function named __asm, with one string constant argument。
The assembler text must be enclosed in double quotes. All the usual character string escape codes retain their definitions.
Like all assembly language statements, the line of code inside the quotes must begin with a label, a blank, a tab, or a comment (asterisk or semicolon).
example:
__asm("STR: .byte \"abc\"");
__asm(" nop");
	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

TI C/C++ Compiler	毕昇编译器
Comment If it begins in column 1, it can start with a semicolon (;) or an asterisk (*). Comments that begin anywhere else on the line must begin with a semicolon. example: * This a comment. ; This a comment. OR AH, PH ; This a comment.	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.

3.4 汇编指示

表 3-4 汇编指示差异

TI C/C++ Compiler	毕昇编译器
.bss symbol, size in words [,blocking flag[,alignment]] Reserves size words in the .bss (uninitialized data) section.	**.bss subsection** .bss tells as to assemble the following statements onto the end of the bss section.
.data Assembles into the .data (initialized data) section.	**.data subsection** .data tells as to assemble the following statements onto the end of the data subsection numbered subsection (which is an absolute expression). If subsection is omitted, it defaults to zero.
.sblock Designates section for blocking.	Not supported.

TI C/C++ Compiler	毕昇编译器
.sect " section name " Assembles into a named (initialized) section.	**.section name** Use the .section directive to assemble the following code into a section named name.
.text Assembles into the .text (executable code) section.	**.text subsection** Tells as to assemble the following statements onto the end of the text subsection numbered subsection, which is an absolute expression.
symbol .**usect** " section name ", size in words[,blocking flag[,alignment flag]] Reserves size words in a named (uninitialized) section.	Not supported.
.endgroup Ends the group declaration. (EABI only).	Not supported.
.gmember section name Designates section name as a member of the group. (EABI only).	**.attach_to_group name** Attaches the current section to the named group.
.group group section name group type : Begins a group declaration. (EABI only).	Not supported.
.clink " section name " Enables conditional linking for the current or specified section.(COFF only).	Not supported.
.retain " section name " Instructs the linker to include the current or specified section in the linked output file, regardless of whether the section is referenced or not. (EABI only).	**.section name[, "flags"]** **R** retained section (apply SHF GNU RETAIN to prevent linker garbage collection, GNU ELF extension)
.retainrefs " section name " Instructs the linker to include any data object that references the current or specified section. (EABI only)	Not supported.
.bits value[, size in bits] Initializes one or more successive bits in the current section.	**.dc[size] expressions** The .dc directive expects zero or more expressions separated by commas. These expressions are evaluated and their values inserted into the current section.
.byte value$_1$[, ... , value$_n$] Initializes one or more successive words in the current section.	**.byte expressions** .byte expects zero or more expressions, separated by commas. Each expression is assembled into the next byte.

TI C/C++ Compiler	毕昇编译器
.char value$_1$[, … , value$_n$] Initializes one or more successive words in the current section.	
.cstring {expr$_1$|" string$_1$"}[,… , {expr$_n$|" string$_n$"}] Initializes one or more text strings.	**.asciz "string". . .** .asciz is just like .ascii, but each string is followed by a zero byte.
.field value[, size] Initializes a field of size bits (1-32) with value.	**.dc[size] expressions** The .dc directive expects zero or more expressions separated by commas. These expressions are evaluated and their values inserted into the current section.
.float value$_1$[, … , value$_n$] Initializes one or more 32-bit, IEEE single-precision, floating-point constants.	**.float flonums** This directive assembles zero or more flonums, separated by commas.
.int value$_1$[, … , value$_n$] Initializes one or more 16-bit integers.	**.int expressions** Expect zero or more expressions, of any section, separated by commas.
.long value$_1$[, … , value$_n$] Initializes one or more 32-bit integers.	**.long expressions** Expect zero or more expressions, of any section, separated by commas.
.pstring {expr$_1$|" string$_1$"}[,… , {expr$_n$|" string$_n$"}] Places 8-bit characters from a character string into the current section.	**.ascii "string". . .** .ascii expects zero or more string literals separated by commas. It assembles each string (with no automatic trailing zero byte) into consecutive addresses.
.string {expr$_1$|" string$_1$"}[,… , {expr$_n$|" string$_n$"}] Initializes one or more text strings.	
.ubyte value$_1$[, … , value$_n$] Initializes one or more successive unsigned bytes in current section.	**.byte expressions** .byte expects zero or more expressions, separated by commas. Each expression is assembled into the next byte.
.uchar value$_1$[, … , value$_n$] Initializes one or more successive unsigned bytes in current section.	
.uint value$_1$[, … , value$_n$] Initializes one or more unsigned 32-bit integers.	**.int expressions** Expect zero or more expressions, of any section, separated by commas.

TI C/C++ Compiler	毕昇编译器
.ulong value$_1$[, … , value$_n$] Initializes one or more unsigned 32-bit integers.	**.long expressions** Expect zero or more expressions, of any section, separated by commas.
.uword value$_1$[, … , value$_n$] Initializes one or more unsigned 16-bit integers.	**.word expressions** This directive expects zero or more expressions, of any section, separated by commas.
.word value$_1$[, … , value$_n$] Initializes one or more 16-bit integers.	
.xfloat value$_1$[, … , value$_n$] Places the 32-bit floating-point representation of one or more floating-point constants into the current section.	**.float flonums** This directive assembles zero or more flonums, separated by commas.
.xldouble value$_1$[, … , value$_n$] Places the 64-bit floating-point representation of one or more floating-point double constants into the current section.	**.double flonums** .double expects zero or more flonums, separated by commas.
.xlong value$_1$[, … , value$_n$] Places one or more 32-bit values into consecutive words in the current section.	**.long expressions** Expect zero or more expressions, of any section, separated by commas.
.align [size in words] Aligns the SPC on a boundary specified by size in words, which must be a power of 2; defaults to 64-byte or page boundary.	**.align [abs-expr[, abs-expr[, abs-expr]]]** Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the alignment required, as described below. If this expression is omitted then a default value of 0 is used, effectively disabling alignment requirements.
.bes size Reserves size bits in the current section; a label points to the end of the reserved space.	**.space size [,fill]** This directive emits size bytes, each of value fill.
.space size Reserves size words in the current section; a label points to the beginning of the reserved space.	
.drlist Enables listing of all directive lines. (default)	Not supported.

TI C/C++ Compiler	毕昇编译器
.drnolist Suppresses listing of certain directive lines.	Not supported.
.fclist Allows false conditional code block listing. (default)	Not supported.
.fcnolist Suppresses false conditional code block listing.	Not supported.
.length [page length] Sets the page length of the source listing.	Not supported.
.list Restarts the source listing.	**.list** By default, listings are disabled.
.mlist Allows macro listings and loop blocks. (default)	Not supported.
.mnolist Suppresses macro listings and loop blocks.	Not supported.
.nolist Stops the source listing.	**.nolist** Control (in conjunction with the .list directive) whether or not assembly listings are generated.
.option option$_1$[, option$_2$, . . .] Selects output listing options; available options are B, L, M, R, T, W, and X.	Not supported.
.page Ejects a page in the source listing.	Not supported.
.sslist Allows expanded substitution symbol listing.	Not supported.
.ssnolist Suppresses expanded substitution symbol listing. (default)	Not supported.
.tab size Sets tab to size characters.	Not supported.
.title " string " Prints a title in the listing page heading.	**.title "heading"** Use heading as the title (second line, immediately after the source file name and pagenumber) when generating assembly listings.

TI C/C++ Compiler	毕昇编译器
.width [page width] Sets the page width of the source listing.	Not supported.
.copy "filename" Includes source statements from another file.	**.incbin "file"[,skip[,count]]** The incbin directive includes file verbatim at the current location.
.include "filename" Includes source statements from another file.	**.include "file"** This directive provides a way to include supporting files at specified points in your source program.
.mlib "filename" Specifies a macro library from which to retrieve macro definitions.	Not supported.
.common symbol, size in bytes [, alignment] **.common** symbol, structure tag [, alignment] Defines a common symbol for a variable. (EABI only)	**.comm symbol , length** .comm declares a common symbol named symbol.
.def symbol$_1$ [, ... , symbol$_n$] Identifies one or more symbols that are defined in the current module and that can be used in other modules.	**.def name**
.global symbol$_1$ [, ... , symbol$_n$] Identifies one or more global (external) symbols.	**.global symbol, .globl symbol** .global makes the symbol visible to ld.
.preserve symbol Causes a symbol's address and value to be preserved during a warm start. This directive must be used in the asm header block. The executable must be in ELF format and compiled for Live Firmware Update (LFU).	Not supported.
.ref symbol$_1$ [, ... , symbol$_n$] Identifies one or more symbols used in the current module that are defined in another module.	**.global symbol, .globl symbol** .global makes the symbol visible to ld.
.symdepend dst symbol name[, src symbol name] Creates an artificial reference from a section to 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.

TI C/C++ Compiler	毕昇编译器
.weak symbol name Identifies a symbol used in the current module that is defined in another module. (EABI only)	**.weak names** This directive sets the weak attribute on the comma separated list of symbol names. If the symbols do not already exist, they will be created.
.asg "character string", substitution symbol Assigns a character string to substitution symbol. Substitution symbols created with .asg can be redefined.	**.equ symbol, expression** This directive sets the value of symbol to expression.
.define "character string", substitution symbol Assigns a character string to substitution symbol. Substitution symbols created with .define cannot be redefined.	**.equiv symbol, expression** The .equiv directive is like .equ and .set, except that the assembler will signal an error if symbol is already defined.
.elfsym name, SYM_SIZE(size) Provides ELF symbol information. (EABI only)	Not supported.
.eval expression ,substitution symbol Performs arithmetic on a numeric substitution symbol.	**.set** symbol, expression Set the value of symbol to expression.
.label symbol Defines a load-time relocatable label in a section.	A label is written as a symbol immediately followed by a colon ‘:’ .
.newblock Undefines local labels	Not supported.
symbol **.set** value Equates value with symbol.	**.set** symbol, expression Set the value of symbol to expression.
.unasg symbol Turns off assignment of symbol as a substitution symbol.	Not supported.
.undefine symbolvar Turns off assignment of symbvarol as a substitution symbol	Not supported.
.if condition Assembles code block if the condition is true.	**.if** absolute expression .if marks the beginning of a section of code which is only considered part of the source program being assembled if the argument (which must be an absolute expression) is nonzero.

TI C/C++ Compiler	毕昇编译器
.elseif condition Assembles code block if the .if condition is false and the .elseif condition is true. When using the .if construct, the .elseif construct is optional.	**.elseif** It is shorthand for beginning a new .if block that would otherwise fill the entire .else section.
.endif Ends .if code block.	**.endif** it marks the end of a block of code that is only assembled conditionally.
.loop [count] Begins repeatable assembly of a code block; the loop count is determined by the count.	Not supported.
.break [end condition] Ends .loop assembly if end condition is true. When using the .loop construct, the .break construct is optional.	Not supported.
.endloop Ends .loop code block.	Not supported.
.cstruct Acts like .struct, but adds padding and alignment like that which is done to C structures	Not supported.
.cunion Acts like .union, but adds padding and alignment like that which is done to C unions	Not supported.
.emember Sets up C-like enumerated types in assembly code.	Not supported.
.endenum Sets up C-like enumerated types in assembly code.	Not supported.
.endstruct Ends a structure definition.	Not supported.
.endunion Ends a union definition.	Not supported.
.enum Sets up C-like enumerated types in assembly code.	Not supported.

TI C/C++ Compiler	毕昇编译器
.union Begins a union definition.	Not supported.
.struct Begins structure definition.	Not supported.
.tag Assigns structure attributes to a label.	Not supported.
macname **.macro** [parameter1][,... , parametern] Begin definition of macro named macname.	**.macro** macname **.macro** macname macargs ... Begin the definition of a macro called macname.
.endm End macro definition.	**.endm** Mark the end of a macro definition.
.mexit Go to .endm.	**.exitm** Exit early from the current macro definition.
.mlib filename Identify library containing macro definitions.	Not supported.
.var Adds a local substitution symbol to a macro's parameter list.	Not supported.
.emsg string Sends user-defined error messages to the output device;produces no .obj file.	**.error** "string" Similarly to .err, this directive emits an error, but you can specify a string that will be emitted as the error message.
.mmsg string Sends user-defined messages to the output device.	**.print** string as will print string on the standard output during assembly. You must put string in double quotes.
.wmsg string Sends user-defined warning messages to the output device.	**.warning** "string" Similar to the directive .error (see Section 7.33 [.error "string"], page 62), but just emits a warning.
.asmfunc Identifies the beginning of a block of code that contains a function.	**.func** name[,label]
.endasmfunc Identifies the end of a block of code that contains a function.	**.endfunc** .endfunc marks the end of a function specified with .func.

TI C/C++ Compiler	毕昇编译器
.setsect Produced by absolute lister; sets a section.	Not supported.
.setsym Produced by the absolute lister; sets a symbol.	Not supported.
.cdecls [options ,]" filename "[, " filename2 "[, ...] Share C headers between C and assembly code.	Not supported.
.end Ends program.	**.end** .end marks the end of the assembly file. as does not process anything in the file past the .end directive.

4 链接器

4.1 命令选项

表 **4-1** 链接器命令选项差异

TI C/C++ Compiler	毕昇编译器
-z --run_linker Enables linking.	Default.
-o --output_file Names the executable output module. The default filename is a.out.	**-o FILE, --output FILE** Set output file name.
-m --map_file Produces a map or listing of the input and output sections, including holes, and places the listing in filename.	**-Wl,Map FILE/DIR** Write a linker map to FILE or DIR/<outputname>.map
-stack --stack_size Sets C system stack size to size words and defines a global symbol that specifies the stack size. Default = 1K words.	**-z stack-size=SIZE** Set size of stack segment.
-heap --heap_size Sets heap size (for the dynamic memory allocation in C) to size words and defines a global symbol that specifies the heap size. Default = 1K words.	Not supported.
-l --library Names an archive library or link command filename as linker input.	**-l LIBNAME, --library LIBNAME** Search for library LIBNAME.

TI C/C++ Compiler	毕昇编译器
--disable_auto_rts Disables the automatic selection of a run-time-support library.	**-nostdlib** Only use library directories specified on the command line.
-priority --priority Satisfies unresolved references by the first library that contains a definition for that symbol.	Default.
-x --reread_libs Forces rereading of libraries, which resolves back references.	**-Wl,--start-group** archives **-Wl,--end-group**
-i --search_path Alters library-search algorithms to look in a directory named with pathname before looking in the default location. This option must appear before the --library option.	**-L DIRECTORY, --library-path DIRECTORY** Add DIRECTORY to library search path.
--define Predefines name as a preprocessor macro.	Not supported.
--undefine Removes the preprocessor macro name.	Not supported.
--disable_pp Disables preprocessing for command files.	Not supported.
--emit_references:file[=file] Emits a file containing section information. The information includes section size, symbols defined, and references to symbols.	Not supported.
--no_demangle Disables demangling of symbol names in diagnostics.	**-Wl,--no-demangle** Do not demangle symbol names.
--set_error_limit Sets the error limit to num. The linker abandons linking after this number of errors. (The default is 100.)	Not supported.

TI C/C++ Compiler	毕昇编译器
--verbose_diagnostics Provides verbose diagnostics that display the original source with line-wrap.	Not supported.
-w --warn_sections Displays a message when an undefined output section is created.	Not supported.
-a --absolute_exe Produces an absolute, executable module. This is the default; if neither --absolute_exe nor --relocatable is specified, the linker acts as if --absolute_exe were specified.	Default.
--ecc={ on | off } Enable linker-generated Error Correcting Codes (ECC). The default is off.	Not supported.
--ecc:data_error Inject the specified errors into the output file for testing.	Not supported.
--ecc:ecc_error Inject the specified errors into the Error Correcting Code (ECC) for testing	Not supported.
--mapfile_contents Controls the information that appears in the map file.	Not supported.
-r --relocatable Produces a nonexecutable, relocatable output module.	**-r, -Wl,--relocatable** Generate relocatable output.
-abs --run_abs Produces an absolute listing file.	Not supported.
--xml_link_info Generates a well-formed XML file containing detailed information about the result of a link.	Not supported.
-e --entry_point Defines a global symbol that specifies the primary entry point for the output module.	**-e ADDRESS, --entry ADDRESS** Set start address.

TI C/C++ Compiler	毕昇编译器
--globalize Changes the symbol linkage to global for symbols that match pattern.	Not supported.
--hide Hides global symbols that match pattern.	Not supported.
--localize Changes the symbol linkage to local for symbols that match pattern.	Not supported.
-g --make_global Makes symbol global (overrides -h).	Not supported.
-h --make_static Makes all global symbols static	Not supported.
-b --no_sym_merge Disables merge of symbolic debugging information in COFF object files.	Not supported.
-s --no_symtable Strips symbol table information and line number entries from the output module.	**-s, --strip-all** Strip all symbols.
--retain Retains a list of sections that otherwise would be discarded. (EABI only).	Not supported.
-scanlibs --scan_libraries Scans all libraries for duplicate symbol definitions.	Not supported.
--symbol_map Maps symbol references to a symbol definition of a different name.	Not supported.
-u --undef_sym Places an unresolved external symbol into the output module's symbol table.	Not supported.
--unhide Reveals (un-hides) global symbols that match pattern.	Not supported.

TI C/C++ Compiler	毕昇编译器
--args --arg_size Allocates memory to be used by the loader to pass arguments.	Not supported.
-f --fill_value Sets default fill values for holes within output sections; fill_value is a 32-bit constant.	The linker uses 0 as the default fill value.
-cr --ram_model Initializes variables at load time.	Not supported.
-c --rom_model Autoinitializes variables at run time. (default)	Default.
--cinit_compression[=compression_kind] Specifies the type of compression to apply to the C auto initialization data.	Not supported.
--compress_dwarf Aggressively reduces the size of DWARF information from input object files.(EABI only)	**-Wl,--compress-debug-sections=[none|zlib|zlib-gnu|zlib-gabi]** Compress DWARF debug sections using zlib(Default: none).
--copy_compression[=compression_kind] Compresses data copied by linker copy tables (EABI only).	Not supported.
--unused_section_elimination Eliminates sections that are not needed in the executable module; on by default. (EABI only)	**-Wl,--gc-sections** Remove unused sections (on some targets).
--keep_asm Retain any post-link files (.pl) and .absolute listing files (.abs) generated by the –plink option. This allows you to view any changes the post-link optimizer makes. (Requires use of -plink)	Not supported.
-nf --no_postlink_across_calls Disable post-link optimizations across functions. (Requires use of -plink)	Not supported.

TI C/C++ Compiler	毕昇编译器
--plink_advice_only Annotates assembly code with comments if changes cannot be made safely due to pipeline considerations, such as when float support or VCU support is enabled. (Requires use of -plink)	Not supported.
-ex --postlink_exclude Exclude files from post-link pass. (Requires use of -plink)	Not supported.
-plink --postlink_opt Post-link optimizations. (Only after --run_linker or -z)	Not supported.
-j --disable_clink Disables conditional linking of COFF object modules. (COFF only)	Not supported.
-help --linker_help Displays information about syntax and available options	**--help** Print option help.
--preferred_order Prioritizes placement of functions.	Not supported.
--zero_init Controls preinitialization of uninitialized variables. Default is on.Always off if --ram_model is used. (EABI only)	Default.

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
}
```