



毕昇编译器

## 使用指南

文档版本 01

发布日期 2024-12-20

版权所有 © 海思技术有限公司2024。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

## 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](mailto:support@hisilicon.com)



# 前言

## 概述

本文档主要介绍毕昇编译器的使用方法，主要包括使用环境、使用方法、支持语言类型、以及注意事项。本文档帮助客户更快的掌握编译器的使用。






## 读者对象

本文档主要适用于升级的操作人员。操作人员必须具备以下经验和技能：

- 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
 <b>危险</b>	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。
 <b>警告</b>	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。
 <b>注意</b>	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。
 <b>须知</b>	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。
 <b>说明</b>	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
2024-08-23	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

前言.....	i
1 Overview.....	1
2 Software Introduction.....	2
2.1 Software.....	2
2.2 Completeness Check.....	2
2.3 Directory Hierarchy.....	2
2.4 Set Permission.....	3
2.5 Environment Request.....	3
3 Operation Description.....	5
3.1 Build.....	5
3.2 Run.....	5
4 Options.....	6
4.1 Linx-MCU Options.....	6
4.2 Himideer Options.....	8
4.3 Common Options.....	9
4.4 LLD Options.....	10
5 C language Support.....	12
5.1 C language specification.....	12
5.2 C library.....	12
6 C++ language Support.....	13
6.1 C++ language specification.....	13
6.2 C++ library.....	13
6.3 C++ Spec Constraints.....	13
7 LLD Support.....	14
7.1 Introduction.....	14
7.2 How to use.....	14
7.3 Constraints && Differences from binutils.....	14
7.4 Disassembler.....	15
8 Constraints.....	16
8.1 Potential risks of strict-aliasing optimization.....	16



8.2 Optimization problem based on global register.....	17
8.3 Encoding conflicts.....	17
8.4 Precision problem caused by -ffp-contract option.....	17
8.5 Unsupported option -ffixed-point.....	18
8.6 Assembly and disassembly non-consistent.....	18
8.7 Trigonometric Function Problems on Windows Platform.....	18
8.8 File size and function size limit for source code.....	19
8.9 Maximum Function contract.....	19
8.10 Intrinsic.....	19
<b>9 Assembler Instructions.....</b>	<b>20</b>
9.1 Usage of the jbcfetch.....	20
9.2 Usage of the c.jbcexecute.....	20
<b>10 External Interfaces.....</b>	<b>21</b>
<b>11 Appendix.....</b>	<b>22</b>
11.1 Terminology.....	22
11.2 Assembler Instructions with C Expression Operands.....	22



# 1 Overview

---

This document focus on the use of BiSheng Mobile MCU toolchain, this toolchain support LinxCore170 and Himideer architecture, including compilers, assemblers, linkers, other binary tools, and standard C/C++ libraries (musl, libc++), In below illustrate, only introduce toolchain which run on Linux OS platform, run on the Windows platform basic is same.



# 2 Software Introduction

## 2.1 Software

Name	Features
linx-llvm-binary-release-musl.tar.gz	Support for RISCV32 architecture toolchain, including compilers, assemblers, linkers, other binary tools, and standard C/C++ libraries (musl, libc++). Run on Linux.
linx-llvm-binary-release-win-musl.tar.gz	Support for RISCV32 architecture toolchain, including compilers, assemblers, linkers, other binary tools, and standard C/C++ libraries (musl, libc++). Run on Windows.
linx-llvm-binary-release-musl-arm.tar.gz	Support for RISCV32 architecture toolchain, including compilers, assemblers, linkers, other binary tools, and standard C/C++ libraries (musl, libc++). Run on Euler-os.

## 2.2 Completeness Check

Generate MD5 and compare with the MD5 value on CMC website where you download the package, command like below.

```
md5sum linx-llvm-binary-release-musl.tar.gz
```

## 2.3 Directory Hierarchy

1. Unpack delivery package.





### 须知

Windows version toolchain suggestion use 7-Zip unpack, because other unpack tool will broken file' s soft-link.

```
tar xf linx-llvm-binary-release-musl.tar.gz
```

2. Enter the directory

```
cd linx-llvm-binary-release-musl
```

3. The directory of linx-llvm-binary-release-musl like this

```
cd linx-llvm-binary-release-musl
|--bin
|--|--clang          C compiler
|--|--|--riscv32
|--|--|--riscv32-linux-musl-ar      Static packaging tools
|--|--|--riscv32-linux-musl-as      Assembler
|--|--|--riscv32-linux-musl-ld      Linker
|--|--|--riscv32-linux-musl-nm      List symbols in [file(s)]
|--|--|--riscv32-linux-musl-objcopy Copies a binary file
|--|--|--riscv32-linux-musl-objdump Display information from object
|--|--|--riscv32-linux-musl-ranlib  Generate an index to speed access to archives
|--|--|--riscv32-linux-musl-readelf Display information about the contents of ELF files
|--|--|--riscv32-linux-musl-size    Display the sizes of sections inside binary files
|--|--|--riscv32-linux-musl-strings Display printable strings in [file(s)]
|--|--|--riscv32-linux-musl-strip   Removes symbols and sections from files
|--include              Compiler-dependent header file
|--lib                  Compiler-dependent library
|--libexec              Compiler-dependent executable library
|--riscv32-elf           Compiler-dependent system library
|--share                Documents for install compiler to Linux
```

## 2.4 Set Permission

To use the compiler, the executable permissions of the files need to be added.

```
chmod -R u+x,g+x linx-llvm-binary-release-musl
```

## 2.5 Environment Request

1. The toolchain running on the Linux platform is built on suse12sp5 and does not support any OS system whose system glibc' s version that is lower than glibc 2.17.

OS system that can support:

- a. Suse12sp5 and above.
- b. CPU Architecture X86-64.

2. The toolchain running on the Euler-OS platform is built on EulerOS V200R012C00SPC100B150.

3. The toolchain running on the Windows platform is cross-compiled using the MinGW on the Linux platform. Therefore, the toolchain depends on the libssp-0.dll file in the MinGW. Can download the pre-built MinGW binary release package from the MinGW official website (<https://www.mingw-w64.org/downloads/#mingw-builds>). Get libssp-0.dll from the binary release package and save it to



C:\Windows\SysWOW64 in the Windows operating system. Download URL:  
<https://github.com/nixman/mingw-builds-binaries/releases>, i686-12.2.0-release-posix-dwarf-rt\_v10-rev0.7z.



# 3 Operation Description

---

## 3.1 Build

1. Prepare source code hello.c.

```
#include <stdio.h>
int main()
{
    printf("Hello, World!");
    return 0;
}
```

2. Run build command, xxx is the absolute directory you unpack delivery package, also the relative is work too.

```
xxx/bin/clang -march=rv32imc hello.c -o a.out
```

## 3.2 Run

Use qemu simulator run executable binary, qemu from core team. Please reference Hisilicon simulators user document.

```
./qemu a.out
```



# 4 Options

---

## 4.1 Linx-MCU Options

Option	Description
-O2/-O3/-Os/-Oz	Support for different optimization levels.
--target	--target=riscv32, choose the target architecture.



Option	Description
-march	<p>Choose the extension for architecture. default is -march=rv32imcxlina_xlinoxmb_xlinoxmc_xlinoxmd extension.</p> <p>Standard extension:</p> <p>rv32i[mfdbc]</p> <p>‘i’ means integer, ‘f’ means float, ‘d’ means double. ‘c’ means compress, ‘b’ means bitmanip(default is 1.0, and 0.92 ,0.921 version is supported. If enable b ,zbb and zbs extensions are enabled(0.92 and 0.921 only support zbb and zbs, 1.0 support zba, zbb, zbc, zbs, if want to enable b1p0 all, march should be b1p0_zba1p0_zbc1p0 or zba1p0_zbb1p0_zbc1p0_zbs1p0).Limit: can’ t link b0p92, b0p921 and b1p0 any two at the same time.).’ p’ means ‘zpn’ , ‘zbpbo’ , ‘zpsfoperand’ . <b>Note:” b0p921” means “b0p92s” , ‘ b’ and ‘zbpbo’ can’ t enable at the same time.</b></p> <p>Non-standard extension:</p> <p>Mapping between -march option and Non-standard extension group name.</p> <p>xlina -&gt; LXISA.LXM_16bit</p> <p>xlinoxmb -&gt; LXISA.LXM_16bit_sub_ext</p> <p>xlinoxmc -&gt; LXISA.LXM_32bit</p> <p>xlinoxmd -&gt; LXISA.LXM_48bit</p> <p>xlinoxme -&gt; LXISA.LXM_tes</p> <p>xlinoxmf -&gt; LXISA.LXM_java</p> <p>xlinoxmg -&gt; LXISA.LXM_coprocessor</p> <p>xlinoxmh -&gt; LXISA.LXM_32bit.0p5</p> <p>Non-standard extension options will be added after standard extension, and separated by underline.</p>
-mabi	<p>It will automatic choose according march, also can use -mabi=ilp32/ilp32f/ilp32d set.</p>
-enjal8m	<p>This is linker option(for clang use like this -WL,-enjal8m), optimization for code size, the original function call that is larger than +/-1M will need to use auipc + jalr two insn. We can combine the auipc + jalr of the function call to jal8m. If you are using ld for link, please add option -enjal8m. If you are using clang for link, please add option -WL,-enjal8m</p>
-mllvm -allow-unalign-ldst	<p>Allow access memory by non-aligned, default is false.</p>
-mllvm -enable-ldst-multiple-opt	<p>Enables generating lwm/swm from multiple ldst, default is false.</p>



Option	Description
-mcpu=linux-rv32	Enable the linuxM sched model(linuxcore170)
-mllvm -disable-indexed-ldst	Disable the use of indexed instruction. Default is false.
-mllvm -m-push-pop	Emit c.push/c.pop/c.popret when handling epilogue and prologue for each function. <b>Attention: Please do not use the -msave-restore option at the same time. The option is on by default.</b>
-mllvm -enable-maddr-msubr	Enable linuxm maddr msubr instructions, default is true.
-mllvm -enable-lwgp	When enable gp relaxation, enable lwgp inst can reduce codesize, default is false.

## 4.2 Himideer Options

Option	Description
--target	--target=riscv32, choose the target architecture.
-march	Choose the extension for HiMiDeer architecture, -march=rv32imc_xhimideer, <b>Attention: It can't use both xhimideer and non-standard extension of linuxm.</b>
-mabi	It will automatic choose according march, can use -march=ilp32/ilp32f/ilp32d set.
-enable-c-lbu-sb	Optimization for assembler, which is disabled by default. If enable this optimization, assembler will use compressed lbu & sb to replace lbu & sb. You should use the -Wa,-enable-c-lbu-sb option when you are assembly with clang and use the -enable-c-lbu-sb when you are assembly with as
-enable-c-lhu-sh	Optimization for assembler, which is disabled by default. If enable this optimization, assembler will use compressed lhu & sh to replace lhu & sh. You should use the -Wa,-enable-c-lhu-sh option when you are assembly with clang and use the -enable-c-lhu-sh when you are assembly with as
-mllvm -imm-compare	Optimization for code size, which can combine the two instructions for no zero immediate compare (li, bxx) to one instruction (bxxi).
-mllvm -merge-immshf	Optimization for code size, which can combine the immediate shift to one instructions.



Option	Description
-mllvm -emit-muliadd	Optimization for code size, which can combine the mul li add tree instruction to one instructions.
-mllvm -emit-uxtb-uxth	Optimization for code size, optimization the unsigned extend byte and unsigned extend half word to uxtb, uxth(16 byte).
--enjal16	Optimization for code size, the original function call that is larger than +/-1M will need to use auipc + jalr two insn. Using the link option, we can combine the auipc + jalr of the function call to jal16/j16. If you are using ld for link, please add option -enjal16. If you are using clang for link, please add option -Wl,--enjal16
-mllvm -himideer-push-pop	Emit push/pop/popret when handling epilogue and prologue for each function.
-mllvm -fldm-stm-optimize	Enable himideer ldmia/stmia instructions
-use-himideer-libs	Add option to control if use himideer libs, when march include xhimideer, can reduce codesize

## 4.3 Common Options

Option	Description
-mllvm -enable-lli	Use the 48 bit l.li instruction to replace the 64 bit instruction lui + addi for 32 bit long immediate load. This optimization is combined with insn combine.
-mllvm -riscv-memcpy-expand-threshold	Set the threshold value of lite memcpy expand to lwm/swm in -O0 and -O1. Default is 16.
-mllvm -riscv-enable-copyelim	Eliminate redundant copy in register X0. Default is true.
-mllvm -enable-libcall-optimize	Enable libcall optimize for performance, now support function of `strcmp`. Default is false.
-mllvm -riscv-memcpy-expand-size-threshold	Set threshold to control whether function "memcpy" expand to instructions in -Os and -Oz. Default is 4.
-mllvm -enable-marker	Enable the marker instruction for function call
-mllvm -enable-marker-wide	Enable the marker inst for indirect jump



Option	Description
-mllvm -enable-oz-inline-threshold	Enable Oz threshold and policy for Os optimize level ultimately making inline harder, default is false.
-mllvm -disable-div-rem-instruction	Disable division and remainder instructions from RISC V M extension. When this option used, “-mabi” can only set to “ilp32”. Default is false.
-mllvm -disable-shrink-to-bool	Disable global variable shrink to bool, default is false.
-mllvm -riscv-enable-gep-opt	Enable optimizations on complex GEPs, default is false, maybe reduce codesize and improve performance. Better for dhrystone.
-jal-transfer	Insert transfer function for jal insn, convert jal to c.jal, reduce codesize
-mllvm -enable-rodata-sections	Enable gc optimization of rodata sections, default is false; If enable this option, please add additional options: -fdata-sections -Wl,--gc-sections.
-mllvm -enable-branch-imm-no-zero	convert beqi/bnei to c.beqz/c.bnez when the imm operand is zero, default is false.
--jal-relax	Relax jal to c.jal if the jump offset is in range of +-2k.
-mllvm -enable-sequence-abstract	Abstract the same sequence in function to reduce codesize, default is false. If enable this option, please add additional options: -mllvm -enable-machine-outliner=always.
-mllvm -riscv-max-global-merge-offset=127	Adjust the max offset of global merge to reduce codesize, offset can also use 2047
-flto	MCU support lto, reduce codesize

## 4.4 LLD Options

Option	Description
-flag-write-default	Linker option; when the new section created by linker script, add “write” flag.
-mllvm -no-section-flag-warn	When the section flag is not set, may cause a running problem, report a warning for it. Default is true.
-disable-always-keep-ehframe	Do not keep .eh_frame section when gc-sections, default is false.





Option	Description
--relax-gp	Enable gp relaxation, default is false
-jal-transfer	Insert transfer function for jal insn, convert jal to c.jal, reduce codesize
-force-page-align	When ensure the monotonicity of linker relaxation and avoid link error, do extra page align even when section address is exactly on the page header, usually do not use this option.



# 5 C language Support

---

## 5.1 C language specification

The C language specification is supported to the C17 standard. For details about the C language specifications, see official website: [https://clang.llvm.org/c\\_status.html](https://clang.llvm.org/c_status.html)

## 5.2 C library

In the toolchain, open source MUSL (1.2.3) is used as the C language library, which is built as a static and shared library for the RISCV32 architecture. POSIX/GUN/BSD specification interfaces are controlled by special macros. The compiler does not define this macro (`_POSIX_SOURCE`, `_GNU_SOURCE`, `_BSD_SOURCE`). You must define this macro yourself either in the source code or in the compiled command line.

For details about the C library specifications, see official website: <https://musl.libc.org>



# 6 C++ language Support

---

## 6.1 C++ language specification

The C++ language specification is supported to the C++17 standard. For details about the C++ language specifications, see official website:

[https://clang.llvm.org/cxx\\_status.html](https://clang.llvm.org/cxx_status.html)

## 6.2 C++ library

In the toolchain, libcxx and libcxxabi from the llvm-project project are used as C++ libraries. These libraries are built into static and dynamic shared libraries targeting the RISCV32 architecture. These libraries depend on the MUSL C language library. For details about the C++ library specifications, see official website:

<https://releases.llvm.org/15.0.4/projects/libcxx/docs/ReleaseNotes.html>

## 6.3 C++ Spec Constraints

1. Not support C++ Exception Handling.
2. Not support `Template<class _Tp> abs(std::complex<_Tp>& x)`.
3. If you want to use `char_traits<char>`, must include `<__string>`.
4. If you want to use function `std::copy`, must include `<algorithm>`.
5. In `std::map<key, value>`, if you use keyword 'const' in key or value, then you can't use operator '=' to set value between two instances.



# 7 LLD Support

## 7.1 Introduction

In the latest release version of the MCU toolchain, we have added support for llvm's built-in assembler and lld linker. Compared with binutils, their advantages include better modularization, better readability, and a more commercial-friendly license, i.e., MIT.

## 7.2 How to use

Simply use an option passed to clang "-lld", which allows you to switch from the binutils toolchain to the lld toolchain for assembly and linking. Note that when you turn on the "-lld" option, you also switch the link-time library path specified by clang. The reason will be explained in detail in next section.

## 7.3 Constraints & Differences from binutils

1. Do not link object files compiled by binutils to lld, and vice versa. This is because the relocation type enumerations used by the two tool chains in the assembly and link phases are incompatible. This problem also exists in library files. Do not change the default library path provided by the compiler during the link process.
2. The differences in support of some options are as follows:
  - a. binutils can support both "-Wl,--gc-sections" and "-Wl,--gc-section" while lld can only support the version with s, that is, "-Wl,--gc-sections", which is also the standard writing in the official gcc document.
  - b. If you need to generate a dynamic library, be sure to add the option "-Wl,-allow-shlib-undefined", which is enabled by default by binutils.
  - c. When symbol address is not defined in an executable file, do not use "--just-symbols", use "-R" instead.
  - d. When you want to enable the option “-mllvm -enable-lwgp=true”, make sure you also enable “-Wl, --relax-gp”.



3. Some of the differences in the link script are as follows:  
Spaces should be reserved before and after the colon, which means do not write "os :ALIGN(64)", write "os : ALIGN(64)" instead.
4. The section flag of the customized section input in the assembly file must be added. Otherwise, the non-alloc attribute is used by default. Even if the link script specifies that the section is stored in a memory region, the section is actually discarded.
5. If you want to use the instruction of ".li" (add option "-mllvm -enable-lli" when compile or hand written in asm), you must enable standard extension "c".
6. LLD currently does not support assembly input of call instruction with immediates, which is consistent with the latest open source version.
7. If you want to use -r to generate a relocatable file when using LLD, you need to add static.
8. LLD does not support Windows versions.
9. If user uses jump instruction like "jal 0x2222", the binutils treats the 0x2222 as a target address, but lld treats it as an offset to target address.
10. In lld, 'b0p92' and 'zbpbo' can't enable at the same time, 'b1.0' and 'zbpbo' can enable at the same time.
11. Do not attempt to discard control sections, like section .shstrtab, lld will give an error "discarding .shstrtab section is not allowed", while GNU ld will simply ignore and retain them.

## 7.4 Disassembler

1. compression instruction:  
The disassembly of compressed instructions is currently displayed as complete instructions. Whether there is a compressed instruction depends on the length of instruction code.
2. pseudo instruction:  
The instruction form is sometimes presented as a pseudo-instruction, which only ensures the correctness of the instruction code.



# 8 Constraints

## 8.1 Potential risks of strict-aliasing optimization

For C/C++ program language, alias optimizations is based on the type of expressions, which assumes an object of one type is never to reside at the same address as an object of a different type, unless the types are almost the same. For example, an "unsigned int" can alias an "int", but can't alias with "void\*" or "double", and character type may alias any other type.

Pay attention to code like this (it might not work as expected).

```
union un {  
    int i;  
    double d;  
};  
int f() {  
    union un t;  
    int *ip;  
    t.d = 3.0;  
    ip = &t.i;  
    return *ip;  
}
```

Similarly, access by taking the address, casting the resulting pointer and dereferencing the result has undefined behavior, even if the cast uses a union type.

```
int f() {  
    double d = 3.0;  
    return ((union un *) &d)->i;  
}
```

### 须知

Option `-fno-strict-aliasing` is turned on by default, if you encounter the above situation, you will be at risk of running the target program incorrectly, you should add option `-fno-strict-aliasing`.



## 8.2 Optimization problem based on global register

If use relaxation optimizations of global register (GP), you should confirm segment alignment will not occur errors. If want disable GP-based optimize, can use link time option `--no-relax-gp`.

## 8.3 Encoding conflicts

These custom instructions have encoding conflicts with RISC-V standard C-extension. They can't be enabled when the RISC-V standard C-extension and D-extension are configured at the same time.

Handling under different cases	c.sb, c.lbu, c.sh, c.shu	c.fsd, c.fld, c.fsdsp, c.fldsp
Case1	No	No
Case2	No	No
Case3	Yes	No
Case4	No	Yes
Case5	No	Yes

1. Case1: RV32 C-extension is not configured, regardless of D-extension and LXM\_16bit\_sub\_ext.
2. Case2: RV32 C-extension is configured, D-extension and LXM\_16bit\_sub\_ext are not configured.
3. Case3: RV32 C-extension and LXM\_16bit\_sub\_ext are configured, D-extension is not configured.
4. Case4: RV32 C-extension and D-extension are configured, LXM\_16bit\_sub\_ext is not.
5. Case5: RV32 C-extension, D-extension, LXM\_16bit\_sub\_ext are configured.

## 8.4 Precision problem caused by -ffp-contract option

The value of this option can be off, on, or fast. By default, the Bisheng Mobile MCU compiler set this value to fast. The FMA operation on floating-point numbers is enabled, and multiplication and addition are combined into a fused multiply-add (FMA or `fmadd`) to speed up and bring acceptable precision bias.

The reasons for this are:

1. The precision of floating point is limited.
2. FMA is a floating-point multiply-add operation performed in one step with a single rounding, while an unfused multiply-add would perform with two



roundings. Fused operations are permitted to produce more precise results than performing the same operations separately.

Note that `-ffp-contract=fast` will override pragmas to fuse multiply and addition across statements regardless of any controlling pragmas. The option of LLVM is not affected by the language standard.

For details, see:

<https://clang.llvm.org/docs/LanguageExtensions.html#extensions-to-specify-floating-point-flags>

<https://clang.llvm.org/docs/UsersManual.html#controlling-floating-point-behavior>

[https://en.wikipedia.org/wiki/Multiply%E2%80%93accumulate\\_operation](https://en.wikipedia.org/wiki/Multiply%E2%80%93accumulate_operation)

## 8.5 Unsupported option -ffixed-point

Fixed point is not support in riscv32, so this option is unsupport in riscv32. If you use this option, you will get a compile error.

## 8.6 Assembly and disassembly non-consistent

Because of alias instruction:

`sll`, `srl`, `sra` and `csrrs`, `csrrw`, `csrrc`, `csrrsi`, `csrrwi`, `csrrci` will preferential use it's alias instruction. Such as: `srl a5,a2,0x17` after assembly and disassembly, we will get `srl a5,a2,0x17`. `srl` is the alias instruction of `srl`.

Because of CSR register encode or name display:

CSR-related instructions support CSR register encode input (e.g. `csrrw x0, 0x345, x0`). We will get `csrw mnxti, zero` after assembly and disassembly. If the address has a corresponding CSR name, disassembly replaces the CSR register encode with the name.

However, some CSRs are still encode after disassembly, even if the input to the assembly is the name of the CSR. They are `hstatus`, `hedeleg`, `hideleg`, `hie`, `htvec`, `hscratch`, `hepc`, `hcause`, `hbadaddr`, `hip`, `mbase`, `mbound`, `mibase`, `mibound`, `mdbase`, `mdbound`, `mscounteren`, `mhcounteren`. (e.g. assembly: `csrrw x0, hcause, x0` -> disassembly: `csrw 0x242, zero`).

## 8.7 Trigonometric Function Problems on Windows Platform

When parameters of trigonometric functions are transferred as constants, constant folding is performed. Constant folding calls the local function library. The Windows math function library has precision problems. If the parameter value is greater than or equal to 263, or less than or equal to -263, a loss of significance in the result occurs. If it is beyond this range, use `-fno-builtin` during compilation to avoid precision problems. (This problem occurs only on Windows platform).

For details, see <https://learn.microsoft.com/en-us/cpp/c-runtime-library/reference/cos-cosf-cosl?view=msvc-170#return-value>





## 8.8 File size and function size limit for source code

Structure nesting: 1500 layers

Maximum lines in the file: 100 million

Maximum number of functions: 2000000, each function has five lines, total 10000000 lines

Number of variables: 20,000

Loop nesting: 5000 layers

Branch nesting: 6000 layers

Macro nesting 10000 Layers

Function nesting 10000 Layers

Variable name length: 20,000 characters

## 8.9 Maximum Function contract

The maximum offset of internal jumps in the function cannot exceed 1 MB. Otherwise, an error will be reported during the link.

## 8.10 Intrinsic

Only guarantee the quality of the Intrinsic provided by HiMCUIntrinsicIntf.doc.

Note: Intrinsic with version(eg. \*\_B0P92), the version b in march must be 0.92. For example:

```
#include <HiMCUGenIntrinsics.h>
int max(int a, int b)
{
    int c = RISC_V_MAX(a, b);
    return c;
}
```

The march need to be -march=rv32imcb0p92.



# 9 Assembler Instructions

## 9.1 Usage of the jbcfetch.

Involved Registers:

Need to config : jpc\_upper, jpc\_lower

input : rs1

output : rs1, jbc

**步骤1** Set jpc\_upper and jpc\_lower to ensure that  $rs1 \leq jpc\_upper$  and  $rs1 \geq jpc\_lower$ ;

**步骤2** set value in address(rs1)

**步骤3** `__asm__("jbcfetch x1");`. Then the output is  $jbc = *rs1$  if we set do step2, else  $jbc = 0xcd(default)$ ;  $rs1 = rs1 + 1$ ;

----结束

## 9.2 Usage of the c.jbcexecute.

Involved Registers:

Need to config : jbc\_upper, jbcft, jbc

**步骤1** Set the jump address:  $addr = jbcft + ((jbc \& 0xFF) \ll 2)$ ;

**步骤2** Set jbc\_upper to ensure that  $jbc \leq jbc\_upper$ ;

**步骤3** set value in address(addr);

**步骤4** `__asm__("c.jbcexecute"::"%ra", "%a0", "%a1", "%a2", "%a3", "%a4", "%a5", "%a6", "%a7", "%t0", "%t1", "%t2", "%t4", "%t5", "%t6");` Run the c.jbcexecute command we should ensure that ra is not changed by other program. Then program will jump to \*addr that we set in step3. If the value of \*addr is not set, the program will jump to 0xcdcdcdcc(default).

----结束



# 10 External Interfaces

---

Currently, all external interfaces are provided in the HiMCUIntrinsicIntf.doc file. The compiler supports only the instructions listed in the file. Other built-in instructions are not supported and compiler may not ensure correctness.



# 11 Appendix

---

## 11.1 Terminology

riscv32: RISC-V 32bit backend architecture.

qemu simulator: instructor accurate simulator.

## 11.2 Assembler Instructions with C Expression Operands

If you want to know how to use “asm” in your project, you can refer to the following address. <https://gcc.gnu.org/onlinedocs/gcc/Extended-Asm.html>