RISC-V编译工具链GCC介绍

软件所智能软件中心PLCT实验室郑志文实习生

2020/02/26

目 录

01 RISC-V仓库简介

RISC-V仓库

仓库连接：https://github.com/riscv/riscv-gnu-toolchain

riscv-gnu-toolchain 是支持RISC-V的GNU工具链，包含了以下内容。

riscv-gcc GCC 编译器。
riscv-binutils-gdb 二进制工具（链接器，汇编器等）、GDB调试工具等。
riscv-glibc GNU C标准库实现。
riscv-llvm 是一个基于LLVM编译器的框架。
riscv-openocd 是一个基于OpenOCD的RISC-V调试器（Debugger）软件。
riscv-opcodes 是一个RISC-V操作码信息转换脚本。
riscv-tests 是一组RISC-V指令集测试用例。
riscv-qemu 是一个支持RISC-V的QEMU模拟器。

01 RISC-V仓库简介

关于RISCV-GCC

RISC-V GCC工具链种类

RISC-V GCC工具链与普通的GCC工具链基本相同，用户可以遵照开源的riscv-gnu-toolchain项目的说明自行生成全套的GCC工具链。

01 RISC-V仓库简介

关于RISCV-GCC

RISC-V GCC工具链种类

以“riscv64-unknown-linux-gnu-”为前缀的版本

前缀表示该版本的工具链是64位架构的Linux版本工具链。注意：此Linux不是指当前版本工具链一定要运行在Linux操作系统的电脑上，此Linux是指该GCC工具链会使用Linux的Glibc作为C运行库，

同理，“riscv32-unknown-linux-gnu-”前缀的版本则是32位架构。

01 RISC-V仓库简介

关于RISCV-GCC

RISC-V GCC工具链种类

以“riscv64-unknown-elf-”为前缀的版本

表示该版本为非Linux（Non-linux）版本的工具链。此Non-Linux不是指当前版本工具链一定不能运行在Linux操作系统的电脑上，此Non-Linux是指该GCC工具链会使用newlib作为C运行库。

同理，“riscv32-unknown-elf-”前缀的版本则是32位架构。

01 RISC-V仓库简介

关于RISCV-GCC

RISC-V GCC工具链种类

以“riscv-none-embed-”为前缀的版本

为前缀的版本，则表示是最新为裸机（bare-metal）嵌入式系统而生成的交叉编译工具链，所谓裸机（bare-metal）是嵌入式领域的一个常见形态，表示不运行操作系统的系统。该版本使用新版本的newlib作为C运行库，并且支持newlib-nano，能够为嵌入式系统生成更加优化的代码体积（Code Size）。

02 GCC的参数讲解

关于RISCV-GCC

RISC-V GCC工具链的（–march=）和（–mabi=）选项

由于RISC-V的指令集是模块化的指令集，因此在为目标RISC-V平台进行交叉编译之时，需要通过选项指定目标RISC-V平台所支持的模块化指令集组合（之前失败的原因之一就是没有设定好指定的指令集组合，而是按照默认选项进行编译）

02 GCC的参数讲解

关于RISCV-GCC

（–march=）选项

rv32i[m][a][f[d]][c]

rv32g[c]

rv64i[m][a][f[d]][c]

rv64g[c]

在上述选项中rv32表示目标平台是32位架构，rv64表示目标平台是64位架构，其他i/m/a/f/d/c/g分别代表了RISC-V模块化指令子集的字母简称

02 GCC的参数讲解

关于RISCV-GCC

（–mabi=）选项

RISC-V定义了**两种整数的ABI调用规则和三种浮点ABI调用规则，通过**选项（-abi=）**指明，有效的**选项值如下：

ilp32, ilp32f, ilp32d

lp64, lp64f, lp64d

前缀ilp32表示目标平台**是32位架构**

前缀lp64表示目标平台**是64位架构**

02 GCC的参数讲解

关于RISCV-GCC

（–mabi=）选项

C type	Description	Bytes in RV32	Bytes in RV64
`char`	Character value/byte	1	1
`short`	Short integer	2	2
`int`	Integer	4	4
`long`	Long integer	4	8
`long long`	Long long integer	8	8
`void*`	Pointer	4	8
`float`	Single-precision float	4	4
`double`	Double-precision float	8	8
`long double`	Extended-precision float	16	16

图片来源：http://www.elecfans.com/d/694957.html

02 GCC的参数讲解

关于RISCV-GCC

（–mabi=）选项

无后缀、后缀f、后缀d

无后缀：**在此架构下，如果使用了浮点**类型的操作，直接使用RISC-V**浮点指令**进行支持。
但是当浮点数作为函数参数进行传递之时，无论单精度浮点数还是**双精度浮点数均需要通**过存储器中的堆栈进行传递。

f： **表示目**标平台支持硬件单精度浮点指令。

d： **表示目**标平台支持硬件双精度浮点指令。

02 GCC的参数讲解

关于RISCV-GCC

同样一个程序采用不同编译参数编译出来的结果也不一样

```
double dmul(double a,
double b) {
        return b * a;
}
```

-march=rv64imafdc -mabi=lp64d

```
dmul:
    fmul.d  fa0,fa0,fa1
    ret
```

-march=rv32imac -mabi=ilp32

```
dmul:
  mv      a4,a2
  mv      a5,a3
  add     sp,sp,-16
  mv      a2,a0
  mv      a3,a1
  mv      a0,a4
  mv      a1,a5
  sw      ra,12(sp)
  call    __muldf3
  lw      ra,12(sp)
  add     sp,sp,16
  jr      ra
```

02 GCC的参数讲解

关于RISCV-GCC

（–mcmodel=）选项

目前RISC-V GCC工具链认为，在实际的情形中，一个程序的大小一般不会超过4GB的大小，因此在程序内部的寻址空间不能超过4GB的空间。而在64位的架构中，地址空间的大小远远的大于4GB的空间，因此针对RV64架构而言，RISC-V GCC工具链定义了（–mcmodel=）选项用于指定寻址范围的模式。

02 GCC的参数讲解

关于RISCV-GCC

（–mcmodel=）选项

（-mcmodel==medlow）

选项用于指示该程序的寻址范围固定只能在-2GB至+2GB的空间内。

（-mcmodel==medlow）

选项用于指示该程序的寻址范围可以在任意的一个4G空间内。

02 GCC的参数讲解

关于RISCV-GCC

（–mcmodel=）选项

（-mcmodel==medlow）

选项用于指示该程序的寻址范围固定只能在-2GB至+2GB的空间内。

（-mcmodel==medlow）

选项用于指示该程序的寻址范围可以在任意的一个4G空间内。

02 GCC的参数讲解

关于RISCV-GCC

参数API文档 http://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html

Next: RL78 Options, Previous: PRU Options, Up: Submodel Options [Contents][Index]

3.19.42 RISC-V Options

These command-line options are defined for RISC-V targets:

`-mbranch-cost=`*n*

Set the cost of branches to roughly *n* instructions.

`-mplt`
`-mno-plt`

When generating PIC code, do or don't allow the use of PLTs. Ignored for non-PIC. The default is `-mplt`.

`-mabi=`*ABI-string*

Specify integer and floating-point calling convention. *ABI-string* contains two parts: the size of integer types and the registers used for floating-point types. For example `'-march=rv64ifd -mabi=lp64d'` means that `'long'` and pointers are 64-bit (implicitly defining `'int'` to be 32-bit), and that floating-point values up to 64 bits wide are passed in F registers. Contrast this with `'-march=rv64ifd -mabi=lp64f'`, which still allows the compiler to generate code that uses the F and D extensions but only allows floating-point values up to 32 bits long to be passed in registers; or `'-march=rv64ifd -mabi=lp64'`, in which no floating-point arguments will be passed in registers.

The default for this argument is system dependent, users who want a specific calling convention should specify one explicitly. The valid calling conventions are: `'ilp32'`, `'ilp32f'`, `'ilp32d'`, `'lp64'`, `'lp64f'`, and `'lp64d'`. Some calling conventions are impossible to implement on some ISAs: for example, `'-march=rv32if -mabi=ilp32d'` is invalid because the ABI requires 64-bit values be passed in F registers, but F registers are only 32 bits wide. There is also the `'ilp32e'` ABI that can only be used with the `'rv32e'` architecture. This ABI is not well specified at present, and is subject to change.

`-mfdiv`
`-mno-fdiv`

Do or don't use hardware floating-point divide and square root instructions. This requires the F or D extensions for floating-point registers. The default is to use them if the specified architecture has these instructions.

谢 谢

欢迎交流合作

2020/02/26