**QEMU实现RISC-V Packed SIMD 标准扩展 v0.9.11项目测试大纲**

**目录**

[1 概述 3](#_Toc172897766)

[2 测试环境 3](#_Toc172897767)

[2.1 X86平台硬件配置 3](#_Toc172897768)

[2.2 测试软件配置 3](#_Toc172897769)

[3 功能测试步骤 4](#_Toc172897770)

[4 功能测试内容 6](#_Toc172897771)

# 概述

本文是对在QEMU实现的RISC-V Packed SIMD 标准扩展 v0.9.11[1]中指令进行测试的流程。因为rstsa16，rstsa32，srli32\_u，umax指令编码与现有指令冲突[2,3]，暂未实现。

[1] https://github.com/riscv/riscv-p-spec/blob/master/P-ext-proposal.adoc

[2] https://y.riscv.ac.cn/whale/plct-qemu/-/merge\_requests/83

[3] https://y.riscv.ac.cn/whale/plct-qemu/-/merge\_requests/86

# 测试环境

* 1. X86平台硬件配置

CPU： Intel(R) Xeon(R) Platinum 8378C CPU @ 2.80GHz

x86\_64架构，128核心

操作系统：Ubuntu 22.04.2 LTS

* 1. 测试软件配置

GNU Make 4.3

ninja 1.10.1

# 功能测试步骤

1. **下载、编译、安装GCC工具链**

下载GCC工具链源代码：

$ cd ~

$ git clone https://github.com/riscv-collab/riscv-gnu-toolchain.git

$ git clone https://github.com/plctlab/riscv-gcc.git -b riscv-gcc-p-ext

$ git clone https://github.com/plctlab/riscv-binutils-gdb.git -b riscv-binutils-p-ext

配置GCC工具链：

$ cd riscv-gnu-toolchain

# 64bit:

$ ./configure --prefix=/path/to/64bit/toolchain --with-arch=rv64gp --with-abi=lp64d --with-gcc-src=/home/\*username\*/riscv-gcc --with-binutils-src=/home/\*username\*/riscv-binutils-gdb

# or 32bit:

$ ./configure --prefix=/path/to/32bit/toolchain --with-arch=rv32gp --with-abi=ilp32 --with-gcc-src=/home/\*username\*/riscv-gcc --with-binutils-src=/home/\*username\*/riscv-binutils-gdb

--prefix指定工具链安装的位置，--with-gcc 和 --with-binutils-src 分别指定 clone 的 gcc 和 gdb 目录。

编译和安装：

$ make linux -j16

更新路径：

$ export PATH=$PATH:/path/to/toolchain/bin

1. **下载、编译、安装 QEMU**

$ cd ~

$ git clone https://github.com/plctlab/plct-qemu.git -b plct-rvp

$ cd plct-qemu

$ /configure --target-list=riscv64-softmmu,riscv32-softmmu,riscv64-linux-user,riscv32-linux-user

$ make -j16

1. **下载、编译、运行测试用例**

下载测试用例：

$ cd ~

$ git clone https://github.com/dajunHuang/riscv-hello-uart.git

修改Makefile中QEMU的路径：

QEMU64 = /home/dajunhuang/plct-qemu/build/qemu-system-riscv64

QEMU32 = /home/dajunhuang/plct-qemu/build/qemu-system-riscv32

运行测试用例：

$ make run64

passed!

输出passed!代表测试用例全部通过，否则会输出error！

1. **QEMU连接GDB调试测试用例**

$ make gdb64

在另一个终端的相同路径打开gdb调试：

$ riscv64-unknown-linux-gnu-gdb -x .gdbinit64

GDB 窗口：

0x0000000000001000 in ?? ()

Breakpoint 1 at 0x80000000: file test64.s, line 5.

Breakpoint 1, \_start () at test64.s:5

5 li s6, 0x0001000100010001

(gdb) n

6 li s7, 0x0002000200020002

(gdb) n

7 add16 s8, s6, s7

(gdb) n

10 csrr t0, mhartid

(gdb) info reg s6 s7 s8

s6 0x1000100010001 281479271743489

s7 0x2000200020002 562958543486978

s8 0x3000300030003 844437815230467

# 功能测试内容

测试对象为QEMU中的指令，64位和32位条件下的测试用例分别位于:

https://github.com/dajunHuang/riscv-hello-uart/blob/main/test64.s

https://github.com/dajunHuang/riscv-hello-uart/blob/main/test32.s

每个测试用例的结构大致如下：

1 # kmabb

2 li s6, 0xFFFF7FFFFFFF7FFF # rs1

3 li s7, 0xFFFF7FFFFFFF7FFF # rs2

4 li s9, 0x7FFFFFFF7FFFFFFF # rd

5 li s8, 0x7FFFFFFF7FFFFFFF # expected result

6 kmabb s9, s6, s7

7 bne s9, s8, print\_error

第1行是注释，代表接下来几行是kmabb指令的测试用例；

第2-4行，为指令涉及到的寄存器赋值。kmabb指令的格式为 kmabb rd, rs1, rs2, 它将rs1寄存器的第一个字与rs2寄存器的第一个字相乘，再与rd寄存器第一个字相加，写入rd寄存器的第一个字。因此为了验证功能，需要手动预设这三个寄存器的值并验证最终rd寄存器的值是否符合预期。每一行后面的注释代表寄存器的身份；

第5行是目标寄存器的预期结果；

第6行执行指令；

第7行比较运行结果与预期结果，如果不一致，跳转到print\_error函数，屏幕输出error!，测试程序结束。如果所有测试用例都不跳转到print\_error函数，屏幕输出passed!，测试用例通过。

测试结果为64位与32位下均输出passed!，测试通过。