Code Analysis Of RISC-V P extensions in QEMU

QEMU 中的 **RVP** 代码分析

主讲人: 卢睿博

PLCT 实验室

February 24, 2021

致谢

本文档基于 Liu Zhiwei from T-head 先生的代码：

CodeBy

From: LIU Zhiwei

Subject: [PATCH 00/38] target/riscv: support packed extension v0.9.2

Date: Fri, 12 Feb 2021 23:02:18 +0800

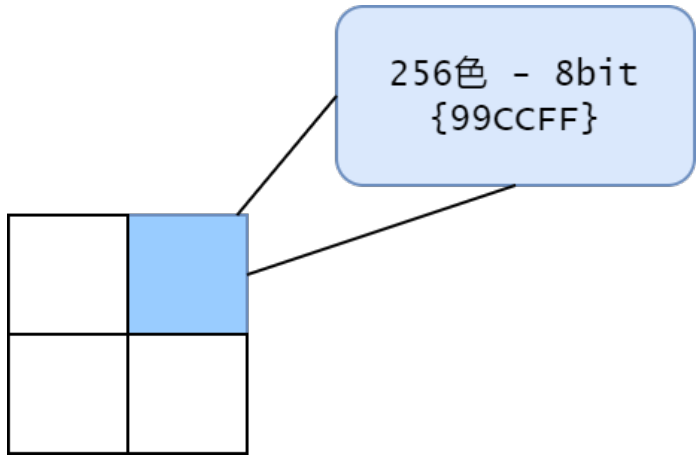
十分感谢 Liu 先生的贡献！！
同时感谢老师们在分析过程中给予的帮助

目录

目录

引入

- 提供 fine grain parallelism，即小碎数据的并行操作.
- 用于 DSP 数字图像处理
- 传统处理器-浪费高位
- 低功耗和高性能



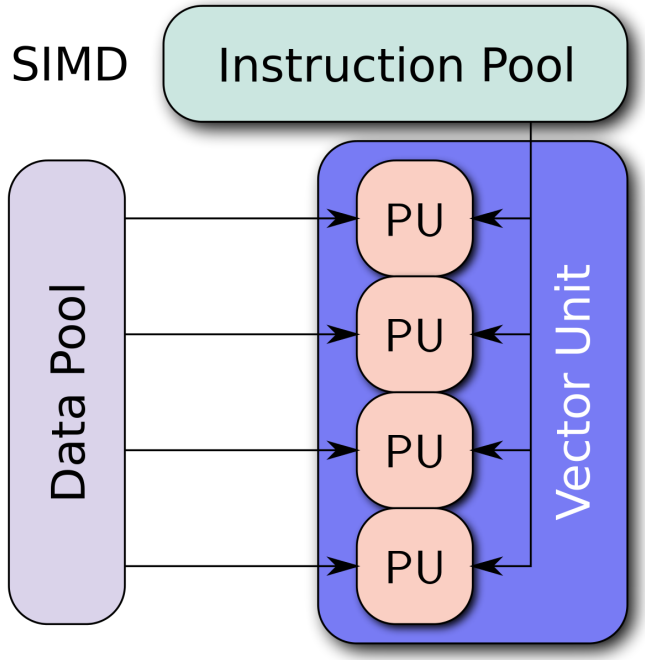
SIMD

SIMD=Single Instruction Multiple Data

Wikipedia

采用一个控制器来控制多个处理器，同时对一组数据分别执行相同的操作从而实现在空间上并行性的技术

- Intel 的 SSE, AMD 的 3D Now！指令集和使用 OpenCL，CUDA 定义的 GPU
- P 和 V 扩展在弗林分类法中都属于 SIMD



Packed-SIMD

Packed

- Packed: serveral of the same type put into one lump
- 整体性：指代数据已经打包好，或本身就是数组类型的
- 节约性：通过细分了现有的寄存器，合理复用宽数据通路
- 一致性：数据的操作是相同且同时进行的

The proposed P instruction set extension increases the DSP algorithm processing capabilities of the RISC-V CPU IP products

P & V

数据级并行方案

P-ext < V-ext

- 利用现有寄存器
- 分解寄存器
- 增量设计

- 添加向量寄存器
- 分解 + 组合
- 丰富的灵活性

本文涉及代码的大部分借鉴自 vector 的相关代码
如果有额外的资源来进行并行计算，向量架构通常是更好的选择，设计者更应使用 RVV 扩展 -RISC-V-READER

如何新增一个扩展？

1. 如何配置并启用命令行中的 P 扩展？

   target/riscv/cpu.c

2. 如何合理分组并选择指令转换的方向

   target/riscv/insn_trans/trans_rvp.c.inc

3. 如何设计指令执行的具体逻辑

   target/riscv/packed_helper.c

目录

CPU 功能位 target/riscv/cpu.c

```
1 static Property riscv_cpu_properties[] = {
2 DEFINE_PROP_BOOL("x-p", RISCVCPU, cfg.ext_p, false),
3 DEFINE_PROP_STRING("pext_spec", RISCVCPU, cfg.pext_spec),
4 DEFINE_PROP_BOOL("Zp64", RISCVCPU, cfg.ext_p64, true),
5 }
```

1. x-p 为 p 扩展测试版，默认关闭
2. pext-spec 存储 P 扩展的版本号
3. Zp64 扩展默认开启
4. 命令行示例:-cpu rv64,x-p=true,Zp64=true,pext_spec=v0.9.2

CPU 实现 target/riscv/cpu.c

```
1  riscv_cpu_realize(DeviceState *dev, Error **errp)
2  int pext_version = PEXT_VERSION;
3  set_pext_version(env, pext_version);
4  if (cpu->cfg.ext_p)
5     target_misa |= RVP;
6     if (cpu->cfg.pext_spec)
7        if (!g_strcmp0(cpu->cfg.pext_spec, "v0.9.2")
                )
8           pext_version = PEXT_VERSION;
9        else
10          error_setg(errp,"版本不支持");
11          return;
12    else
13       qemu_log("请使用默认值 v0.9.2\n");
14    if (!cpu->cfg.ext_p64 && env->misa == RV64)
15       error_setg(errp, "Zp64会被包含在RV64");
16       return;
17    set_pext_version(env, pext_version);
```

CPU 实现 target/riscv/cpu.c

1. P 版本 v0.9.2
2. 是否开启 P
3. 版本是否正确
4. p64 与 RV64
5. RV64 下开启 P 会自动开启 Zp64

```
1  riscv_cpu_realize(DeviceState *dev, Error **errp)
2  int pext_version = PEXT_VERSION;
3  set_pext_version(env, pext_version);
4  if (cpu->cfg.ext_p)
5     target_misa |= RVP;
6     if (cpu->cfg.pext_spec)
7        if (!g_strcmp0(cpu->cfg.pext_spec, "v0.9.2")
                )
8           pext_version = PEXT_VERSION;
9        else
10          error_setg(errp,"版本不支持");
11          return;
12    else
13       qemu_log("请使用默认值 v0.9.2\n");
14    if (!cpu->cfg.ext_p64 && env->misa == RV64)
15       error_setg(errp, "Zp64会被包含在RV64");
16       return;
17    set_pext_version(env, pext_version);
```

目录

指令转译的抉择

指令转译的速度和可信度

1. qemu 内部定义的 gvec 操作- tcg_gen_gvec_
2. v 扩展和整型操作中已经定义的函数
3. 利用 tcg 操作的组合
4. 自己定义的代码逻辑

所造成抉择

1. 涉及 **reg!=0**
2. RV32 or **RV64**
3. 是否可以借助已定义的函数

示例 1 insn_trans/trans_rvp.c.inc

- 不需要调用 hlper 的内联函数
- NAME: 指令名称
- GSUF:gvec 函数名称
- VECE:vector element size。log2 bytes= 操作数大小为 8«VECE
- FN : 已定义的函数

```
40  static inline bool
41  r_inline(DisasContext *ctx, arg_r *a, uint8_t vece,
42           GenNoZero64Fn *f64, GenNoZero32Fn *f32,
43           GenZeroFn *fn)
44  {
45      if (!has_ext(ctx, RVP)) {
46          return false;
47      }
48      if (a->rd && a->rs1 && a->rs2) {
49  #ifdef TARGET_RISCV64
50          f64(vece, offsetof(CPURISCVState, gpr[a->rd]),
51              offsetof(CPURISCVState, gpr[a->rs1]),
52              offsetof(CPURISCVState, gpr[a->rs2]),
53              8, 8);
54  #else
55          f32(cpu_gpr[a->rd], cpu_gpr[a->rs1], cpu_gpr[a->rs2]);
56  #endif
57      } else {
58          fn(ctx, a);
59      }
60      return true;
61  }
62
63  /* Complete inline implementation */
64  #define GEN_RVP_R_INLINE(NAME, GSUF, VECE, FN)                  \
65  static bool trans_##NAME(DisasContext *s, arg_r *a)             \
66  {                                                               \
67      return r_inline(s, a, VECE, tcg_gen_gvec_##GSUF,            \
68                      tcg_gen_simd_##NAME, (GenZeroFn *)FN);      \
69  }                                                               \
```

示例 1 insn_trans/trans_rvp.c.inc

- GEN_RVP_: 宏定义
  缩减参数，简化内容
- r_inline: 内联函数
  简化开销，智能选择调用方向
- f64:gvec 函数
- f32: 自己定义函数
- fn: 已定义函数

```
40  static inline bool
41  r_inline(DisasContext *ctx, arg_r *a, uint8_t vece,
42           GenNoZero64Fn *f64, GenNoZero32Fn *f32,
43           GenZeroFn *fn)
44  {
45      if (!has_ext(ctx, RVP)) {
46          return false;
47      }
48      if (a->rd && a->rs1 && a->rs2) {
49  #ifdef TARGET_RISCV64
50          f64(vece, offsetof(CPURISCVState, gpr[a->rd]),
51              offsetof(CPURISCVState, gpr[a->rs1]),
52              offsetof(CPURISCVState, gpr[a->rs2]),
53              8, 8);
54  #else
55          f32(cpu_gpr[a->rd], cpu_gpr[a->rs1], cpu_gpr[a->rs2]);
56  #endif
57      } else {
58          fn(ctx, a);
59      }
60      return true;
61  }
62
63  /* Complete inline implementation */
64  #define GEN_RVP_R_INLINE(NAME, GSUF, VECE, FN)                  \
65  static bool trans_##NAME(DisasContext *s, arg_r *a)             \
66  {                                                               \
67      return r_inline(s, a, VECE, tcg_gen_gvec_##GSUF,            \
68                      tcg_gen_simd_##NAME, (GenZeroFn *)FN);      \
69  }                                                               \
```

示例 1 insn_trans/trans_rvp.c.inc

- VECE: **1**=> 16=8«1
- f64: tcg_gen_gvec_add
- f32: tcg_gen_simd_add16
  mask : 提取-> 相加-> 组合
- fn: trans_add

```
1  static void tcg_gen_simd_add16(TCGv d,TCGv a,TCGv b)
2  {
3      TCGv t1 = tcg_temp_new();
4      TCGv t2 = tcg_temp_new();
5
6      tcg_gen_andi_tl(t1, a, ~0xffff);
7      tcg_gen_add_tl(t2, a, b);
8      tcg_gen_add_tl(t1, t1, b);
9      tcg_gen_deposit_tl(d, t1, t2, 0, 16);
10
11     tcg_temp_free(t1);
12     tcg_temp_free(t2);
13 }
14 GEN_RVP_R_INLINE(add16, add, 1, trans_add);
```

示例 1 insn_trans/trans_rvp.c.inc

- vece:vec element
- ofs: offset
- sz: size
- MO_i= 8«i
- gvec_3
- RV64 支持

```
1  void tcg_gen_gvec_add(unsigned vece, uint32_t dofs, uint32_t aofs,
2                        uint32_t bofs, uint32_t oprsz, uint32_t maxsz)
3  {
4      static const GVecGen3 g[4] = {
5          { .fni8 = tcg_gen_vec_add8_i64,
6            .fniv = tcg_gen_add_vec,
7            .fno = gen_helper_gvec_add8,
8            .opt_opc = vecop_list_add,
9            .vece = MO_8 },
10         { .fni8 = tcg_gen_vec_add16_i64,
11           .fniv = tcg_gen_add_vec,
12           .fno = gen_helper_gvec_add16,
13           .opt_opc = vecop_list_add,
14           .vece = MO_16 },
15         { .fni4 = tcg_gen_add_i32,
16           .fniv = tcg_gen_add_vec,
17           .fno = gen_helper_gvec_add32,
18           .opt_opc = vecop_list_add,
19           .vece = MO_32 },
20         { .fni8 = tcg_gen_add_i64,
21           .fniv = tcg_gen_add_vec,
22           .fno = gen_helper_gvec_add64,
23           .opt_opc = vecop_list_add,
24           .prefer_i64 = TCG_TARGET_REG_BITS == 64,
25           .vece = MO_64 }, };
26     tcg_debug_assert(vece <= MO_64);
27     tcg_gen_gvec_3(dofs, aofs, bofs, oprsz, maxsz, &g[vece]);
```

示例 2 insn_trans/trans_rvp.c.inc

- 调用 helper 的外联函数
- gen_helper_name: packed_helper.c
- tcg_temp_new: 声明新的 tcg 变量
- gen_get_gpr 给 tcg 变量赋 reg 值
- gen_set_gpr 将结果取出，存入 rd
- tcg_temp_free 释放 tcg 变量

```
1  typedef void gen_helper_rvp_r(TCGv, TCGv_ptr, TCGv, TCGv);
2  static inline bool r_ool(DisasContext *ctx, arg_r *a,
       gen_helper_rvp_r *fn)
3  {
4      TCGv src1, src2, dst;
5      if (!has_ext(ctx, RVP)) {
6          return false;
7      }
8
9      src1 = tcg_temp_new();
10     src2 = tcg_temp_new();
11     dst = tcg_temp_new();
12
13     gen_get_gpr(src1, a->rs1);
14     gen_get_gpr(src2, a->rs2);
15     fn(dst, cpu_env, src1, src2);
16     gen_set_gpr(a->rd, dst);
17
18     tcg_temp_free(src1);
19     tcg_temp_free(src2);
20     tcg_temp_free(dst);
21     return true;
22 }
23 #define GEN_RVP_R_OOL(NAME) \
24 static bool trans_##NAME(DisasContext *s, arg_r *a) \
25 { \
26     return r_ool(s, a, gen_helper_##NAME); \
```

示例 3 insn_trans/trans_rvp.c.inc

- shift 函数
- extrl: 提取低位 gvec 要求 TCGv_i32
- andi: 立即数与 提取 VECE 长
- mask：掩码 (8 « VECE) - 1
- rvp_shift_ool 见下页

```
1  static inline bool
2  rvp_shift(DisasContext *ctx, arg_r *a, uint8_t vece,
3          GenGvecShift *f64, gen_helper_rvp_r *fn,
4          uint8_t mask)
5  {
6      if (!has_ext(ctx, RVP))
7          return false;
8  #ifdef TARGET_RISCV64
9      if (a->rd && a->rs1 && a->rs2) {
10         TCGv_i32 shift = tcg_temp_new_i32();
11         tcg_gen_extrl_i64_i32(shift, cpu_gpr[a->rs2]);
12         tcg_gen_andi_i32(shift, shift, mask);
13         f64(vece, offsetof(CPURISCVState, gpr[a->rd]),
14             offsetof(CPURISCVState, gpr[a->rs1]),
15             shift, 8, 8);
16         tcg_temp_free_i32(shift);
17         return true;
18     }
19 #endif
20     return rvp_shift_ool(ctx, a, fn, mask);
21 }
22 #define GEN_RVP_SHIFT(NAME, GVEC, VECE) \
23 static bool trans_##NAME(DisasContext *s, arg_r *a) \
24 { \
25     return rvp_shift(s, a, VECE, GVEC, gen_helper_##NAME,\
26                      (8 << VECE) - 1); \
27 }
```

示例 3 insn_trans/trans_rvp.c.inc

- rvp_shift_ool
- andi: 立即数与提取 VECE 长
- fn: 调用 helper 例：gen_helper_sra16

```
1  /* 16-bit Shift Instructions */
2  static bool rvp_shift_ool(DisasContext *ctx, arg_r *a,
3                            gen_helper_rvp_r *fn, target_ulong
                                 mask)
4  {
5      TCGv src1, src2, dst;
6
7      src1 = tcg_temp_new();
8      src2 = tcg_temp_new();
9      dst = tcg_temp_new();
10
11     gen_get_gpr(src1, a->rs1);
12     gen_get_gpr(src2, a->rs2);
13     tcg_gen_andi_tl(src2, src2, mask);
14
15     fn(dst, cpu_env, src1, src2);
16     gen_set_gpr(a->rd, dst);
17
18     tcg_temp_free(src1);
19     tcg_temp_free(src2);
20     tcg_temp_free(dst);
21     return true;
22 }
```

示例 4 insn_trans/trans_rvp.c.inc

- shifti 函数
- OP: 对应的 shift 调用 shift 函数
- shamt: 立即数
- tcg_const_tl: 声明 tl 长的立即数

```
1  static bool rvp_shifti_ool(DisasContext *ctx, arg_shift *a,
2                             gen_helper_rvp_r *fn)
3  {
4      shift = tcg_const_tl(a->shamt);
5      fn(dst, cpu_env, src1, shift);
6  }
7  static inline bool
8  rvp_shifti(DisasContext *ctx, arg_shift *a,
9             void (* f64)(TCGv_i64, TCGv_i64, int64_t),
10            gen_helper_rvp_r *fn) {
11     if (!has_ext(ctx, RVP)) {
12         return false;
13     }
14 #ifdef TARGET_RISCV64
15     if (a->rd && a->rs1 && f64) {
16         f64(cpu_gpr[a->rd], cpu_gpr[a->rs1], a->shamt);
17         return true;
18     }
19 #endif
20     return rvp_shifti_ool(ctx, a, fn);
21 }
22 #define GEN_RVP_SHIFTI(NAME, OP, GVEC) \
23 static bool trans_##NAME(DisasContext *s, arg_shift *a) \
24 { \
25     return rvp_shifti(s, a, GVEC, gen_helper_##OP); \
26 }
27 GEN_RVP_SHIFTI(srai16, sra16, tcg_gen_vec_sar16i_i64);
```

示例 5 insn_trans/trans_rvp.c.inc

- D64 函数
- RV64=>r_ool
  不需要操作
- ext_p64:Zp64
- extrl/h: 提取低/高位
- rd+1: 放入相邻 reg

```
1  static inline bool
2  r_d64_ool(DisasContext *ctx, arg_r *a,
3          void (* fn)(TCGv_i64, TCGv_ptr, TCGv, TCGv))
4  {
5  #ifdef TARGET_RISCV64
6      return r_ool(ctx, a, fn);
7  #else
8      if (!has_ext(ctx, RVP) || !ctx->ext_p64) {
9          return false;
10     }
11
12     fn(dst, cpu_env, src1, src2);
13
14     low = tcg_temp_new_i32();
15     high = tcg_temp_new_i32();
16     tcg_gen_extrl_i64_i32(low, dst);
17     tcg_gen_extrh_i64_i32(high, dst);
18     gen_set_gpr(a->rd, low);
19     gen_set_gpr(a->rd + 1, high);
20     tcg_temp_free_i32(low);
21     tcg_temp_free_i32(high);
22     return true;
23 #endif
24 }
```

总结 insn_trans/trans_rvp.c.inc

- 所有定义
- R:3 操作数
- R2:2 操作数
- R4:4 操作数
- OOL:Out Of Line
- D64:Destination 64
- S64:Source 64
- SHIFTI:shift imm
- ACC:accumulate destination register

```
1  GEN_RVP_R_INLINE(NAME, GSUF, VECE, FN);
2  // add16 sub16 add8 add32 sub32
3  GEN_RVP_R_OOL(NAME);
4  GEN_RVP_SHIFT(NAME, GVEC, VECE);
5  GEN_RVP_SHIFTI(NAME, OP, GVEC);
6  GEN_RVP_R_D64_OOL(NAME);
7  GEN_RVP_R2_OOL(NAME);
8  GEN_RVP_R_ACC_OOL(NAME);
9  GEN_RVP_R_D64_S64_OOL(NAME);
10 GEN_RVP_R_D64_S64_S64_OOL(NAME);
11 GEN_RVP_R_D64_ACC_OOL(NAME);
12 GEN_RVP_R_S64_OOL(NAME);
13 //wext
14 GEN_RVP_SHIFTI_S64_OOL(NAME, OP);
15 GEN_RVP_R4_OOL(NAME);
16 // bpick
```

目录

介绍

思想

- 灵活性：要适应大端机器和 RV32，RV64
- 扩张性：利用更长的运算保留进位
- 循环法：因为一致性，将指令分解为多个相同微操作

灵活性

针对大端数据的适应

x	H1	H2	H4
0	7	3	1
1	6	2	0
2	5	1	3
3	4	0	2
4	3	7	5
5	2	6	4
6	1	5	7
7	0	4	6

```
1  #ifdef HOST_WORDS_BIGENDIAN
2  #define H1(x) ((x) ^ 7)
3  #define H2(x) ((x) ^ 3)
4  #define H4(x) ((x) ^ 1)
5  #define H8(x) ((x))
```

PackedFN

```
1  // 将指令分为 可循环不可循环两类
2  typedef void PackedFn3i(CPURISCVState *, void *, void *, void *, uint8_t);
3  typedef void PackedFn3(CPURISCVState *, void *, void *, void *);
4  typedef void PackedFn2i(CPURISCVState *, void *, void *, uint8_t);
5  typedef void PackedFn4i(CPURISCVState *, void *, void *,
6                          void *, void *, uint8_t)
```

PackedFN<x>[i] ： x 指操作数的数量，i 指代是否还有循环变量
循环变量 i 的类型：uint8_t

示例 1 target/riscv/packed_helper.c

- 最普遍
- NAME: 操作名称
- STEP: 步长
  有的函数是多个 SIZE 为一组进行的
- SIZE: 操作数大小
  以 byte 为单位，1=8，2=16
- passes: 循环数
  target_ulong 灵活适配 RV32/64

```
1  /* Define a common function to loop
       elements in packed register */
2  static inline target_ulong
3  rvpr(CPURISCVState *env, target_ulong a,
       target_ulong b,
4      uint8_t step, uint8_t size,
         PackedFn3i *fn)
5  {
6      int i, passes = sizeof(target_ulong)
         / size;
7      target_ulong result = 0;
8
9      for (i = 0; i < passes; i += step) {
10         fn(env, &result, &a, &b, i);
11     }
12     return result;
13 }
14 #define RVPR(NAME, STEP, SIZE) \
15 target_ulong HELPER(NAME)(CPURISCVState
       *env, target_ulong a, \
16                     target_ulong b) \
17 { \
18     return rvpr(env, a, b, STEP, SIZE, (
         PackedFn3i *)do_##NAME);\
19 }
20 RVPR(kadd16, 1, 2);
```

示例 1-1 target/riscv/packed_helper.c

- 两种实例化
- sadd:v 操作 vector_helper.c
- 0:vxrm
- cras16: 交叉运算 16=8*2=>H2 一组两个
- ave: 非计算操作 sizeof(target_ulong)

```
1  static inline void do_kadd16(CPURISCVState *env, void *
       vd, void *va,
2                               void *vb, uint8_t i)
3  {
4      int16_t *d = vd, *a = va, *b = vb;
5      d[i] = sadd16(env, 0, a[i], b[i]);
6  }
7  RVPR(kadd16, 1, 2);
8
9  static inline void do_cras16(CPURISCVState *env, void *
       vd, void *va,
10                              void *vb, uint8_t i)
11 {
12     uint16_t *d = vd, *a = va, *b = vb;
13     d[H2(i)] = a[H2(i)] - b[H2(i + 1)];
14     d[H2(i + 1)] = a[H2(i + 1)] + b[H2(i)];
15 }
16 RVPR(cras16, 2, 2);
17
18 RVPR(ave, 1, sizeof(target_ulong));
```

示例 1-2 target/riscv/packed_helper.c

- 扩张性
- int16_t->int32_t->int64_t

```
1  static inline int32_t hadd32(int32_t a, int32_t b)
2  {
3      return ((int64_t)a + b) >> 1;
4  }
5
6  static inline void do_radd16(CPURISCVState *env, void *vd, void *va,
7                               void *vb, uint8_t i)
8  {
9      int16_t *d = vd, *a = va, *b = vb;
10     d[i] = hadd32(a[i], b[i]);
11 }
12
13 RVPR(radd16, 1, 2);
```

示例 1-3 target/riscv/packed_helper.c

- 所有定义
- RVPR64_64_64
- ACC:4 参数
- RVPR2:2 参数

```
1 RVPR(NAME, STEP, SIZE);
2 RVPR64(NAME);
3 RVPR2(NAME, STEP, SIZE)
4 RVPR_ACC(NAME, STEP, SIZE)
5 RVPR64_64_64(NAME, STEP, SIZE)
6 //passes = sizeof(uint64_t) / size
7 RVPR64_ACC(NAME, STEP, SIZE)
8 //RV32 两个寄存器合起来
9 RVPR_64(NAME)
```

目录

References

- p-spec
  https://github.com/riscv/riscv-p-spec/blob/master/P-ext-proposal.adoc
- patchew
  https://patchew.org/QEMU/20210212150256.885-1-zhiwei_liu\spacefactor\@m{}c-sky.com/#
- SIMD
  https://en.wikipedia.org/wiki/SIMD

Thank you

Thank you for listening!

Q&A

Questions?