添加ZCE指令到RISCV-GNU-TOOLCHAIN

PLCT实验室 陈嘉炜

jiawei@iscas.ac.cn

2021.05.12

主要内容：

- ZCE简介
- GCC添加
- Binutils添加
- 测试用例添加
- 测试验证

ZCE简介:

Zce是Code Size Reduction任务组提出的用于减小RISCV代码生成体积的一个ISA子扩展

邮件列表： https://lists.riscv.org/g/tech-code-size/topics

Github： https://github.com/riscv/riscv-code-size-reduction

Youtobe介绍: https://www.youtube.com/watch?v=cT61fA_sIlo&t=136s

Benchmark result:
https://docs.google.com/spreadsheets/d/1UYlI7HGR_QLGTsHcjGoNL4EodM5BNO41hXdxVAFaxFs/edit#gid=1281210325

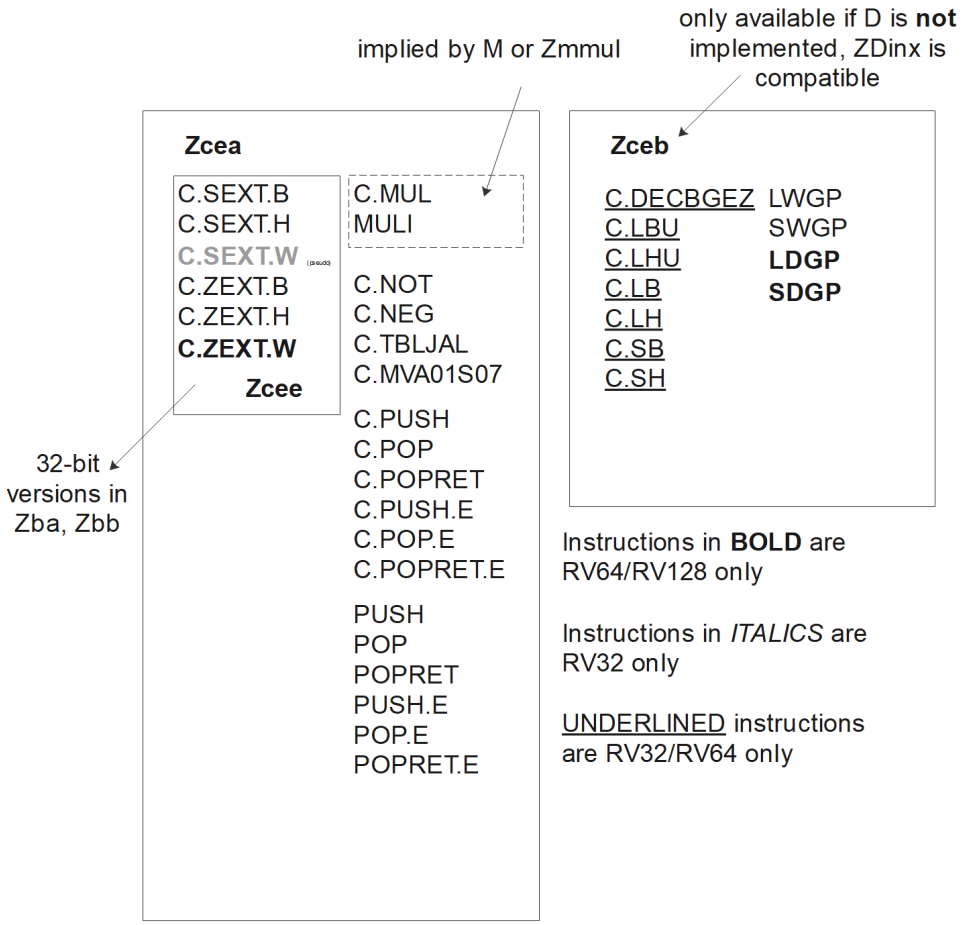
ZCE简介:

目前Zce已更新至v0.41, 包含3个子扩展集Zcea, Zceb, Zcee

Zcea对应m、a指令的缩小

Zceb对应浮点指令的缩小

Zcee对应扩展(sign/zero-extension)指令的缩小



开发准备:

下载riscv-gnu-toolchain

“ git clone https://github.com/riscv/riscv-gnu-toolchain --recursive ”

复制一份到zce-dev

cp riscv-gnu-toolchain zce-dev -r

检查各个子目录的版本,确保与目标开发分支相同

git branch -v

GCC添加:

主要添加目录包括gcc/common/config/risv与gcc/config/risv

首先修改riscv-common.c中定义的flag,扩展信息与版本号

```
/* Mapping table between extension to internal flag.  */
static const riscv_ext_flag_table_t riscv_ext_flag_table[] =
{
  {"e", &gcc_options::x_target_flags, MASK_RVE},
  {"m", &gcc_options::x_target_flags, MASK_MUL},
  {"a", &gcc_options::x_target_flags, MASK_ATOMIC},
  {"f", &gcc_options::x_target_flags, MASK_HARD_FLOAT},
  {"d", &gcc_options::x_target_flags, MASK_DOUBLE_FLOAT},
  {"c", &gcc_options::x_target_flags, MASK_RVC},

  {"zicsr",    &gcc_options::x_riscv_zi_subext, MASK_ZICSR},
  {"zifencei", &gcc_options::x_riscv_zi_subext, MASK_ZIFENCEI},

  {"zkg", &gcc_options::x_riscv_crypto_subext, MASK_ZKG},
  {"zkb", &gcc_options::x_riscv_crypto_subext, MASK_ZKB},
  {"zkr", &gcc_options::x_riscv_crypto_subext, MASK_ZKR},
  {"zkne", &gcc_options::x_riscv_crypto_subext, MASK_ZKNE},
  {"zknd", &gcc_options::x_riscv_crypto_subext, MASK_ZKND},
  {"zknh", &gcc_options::x_riscv_crypto_subext, MASK_ZKNH},
  {"zksed", &gcc_options::x_riscv_crypto_subext, MASK_ZKSED},
  {"zksh", &gcc_options::x_riscv_crypto_subext, MASK_ZKSH},

  {NULL, NULL, 0}
};
```

```
/* Implied ISA info, must end with NULL sentinel.  */
riscv_implied_info_t riscv_implied_info[] =
{
  {"d", "f"},
  {"f", "zicsr"},
  {"d", "zicsr"},
  {"k", "zkn"},
  {"k", "zkr"},
  {"zkn", "zkne"},
  {"zkn", "zknd"},
  {"zkn", "zknh"},
  {"zkn", "zkg"},
  {"zkn", "zkb"},
  {"zks", "zksed"},
  {"zks", "zksh"},
  {"zks", "zkg"},
  {"zks", "zkb"},
  {NULL, NULL}
};
```

GCC添加:

riscv_ext_flag_table[]用来识别扩展作为arch参数

riscv_implied_info[]用来展开扩展，例如rv32imad—>rv32imafd

这里zce会被展开成_zcea_zceb_zcee

riscv_ext_version_table[]用来定义扩展的版本号

对于draft的spec，其版本号定义为ISA_SPEC_CLASS_NONE

```
/* All standard extensions defined in all supported ISA spec.  */
static const struct riscv_ext_version riscv_ext_version_table[] =
{
  /* name, ISA spec, major version, minor_version.  */
  {"e", ISA_SPEC_CLASS_20191213, 1, 9},
  {"e", ISA_SPEC_CLASS_20190608, 1, 9},
  {"e", ISA_SPEC_CLASS_2P2,      1, 9},

  {"i", ISA_SPEC_CLASS_20191213, 2, 1},
  {"i", ISA_SPEC_CLASS_20190608, 2, 1},
  {"i", ISA_SPEC_CLASS_2P2,      2, 0},

  {"m", ISA_SPEC_CLASS_20191213, 2, 0},
  {"m", ISA_SPEC_CLASS_20190608, 2, 0},
  {"m", ISA_SPEC_CLASS_2P2,      2, 0},

  {"a", ISA_SPEC_CLASS_20191213, 2, 1},
  {"a", ISA_SPEC_CLASS_20190608, 2, 0},
  {"a", ISA_SPEC_CLASS_2P2,      2, 0},

  {"f", ISA_SPEC_CLASS_20191213, 2, 2},
  {"f", ISA_SPEC_CLASS_20190608, 2, 2},
  {"f", ISA_SPEC_CLASS_2P2,      2, 0},

  {"d", ISA_SPEC_CLASS_20191213, 2, 2},
  {"d", ISA_SPEC_CLASS_20190608, 2, 2},
  {"d", ISA_SPEC_CLASS_2P2,      2, 0},

  {"c", ISA_SPEC_CLASS_20191213, 2, 0},
  {"c", ISA_SPEC_CLASS_20190608, 2, 0},
  {"c", ISA_SPEC_CLASS_2P2,      2, 0},
```

GCC添加:

进入gcc/config/risv，修改riscv.opt，添加对应riscv-common.c中riscv_ext_flag_table[] 新定义的Mask信息到mrelax, 更新TargetVariable定义

```
mrelax
Target Bool Var(riscv_mrelax) Init(1)
Take advantage of linker relaxations to reduce the number of instructions
required to materialize symbol addresses.

Mask(64BIT)

Mask(MUL)

Mask(ATOMIC)

Mask(HARD_FLOAT)

Mask(DOUBLE_FLOAT)

Mask(RVC)

Mask(RVE)
```

```
TargetVariable
int riscv_crypto_subext

TargetVariable
int riscv_bitmanip_subext

TargetVariable
int riscv_zi_subext
```

GCC添加:

同步更新riscv-opt.h中有关MASK与TARGET的记录

```
#define MASK_ZICSR    (1 << 0)
#define MASK_ZIFENCEI (1 << 1)
#define MASK_ZKG      (1 << 2)
#define MASK_ZKB      (1 << 3)
#define MASK_ZKR      (1 << 4)
#define MASK_ZKNE     (1 << 5)
#define MASK_ZKND     (1 << 6)
#define MASK_ZKNH     (1 << 7)
#define MASK_ZKSED    (1 << 8)
#define MASK_ZKSH     (1 << 9)

#define TARGET_ZICSR    ((riscv_zi_subext & MASK_ZICSR) != 0)
#define TARGET_ZIFENCEI ((riscv_zi_subext & MASK_ZIFENCEI) != 0)
#define TARGET_ZKG ((riscv_crypto_subext & MASK_ZKG) != 0)
#define TARGET_ZKB ((riscv_crypto_subext & MASK_ZKB) != 0)
#define TARGET_ZKR ((riscv_crypto_subext & MASK_ZKR) != 0)
#define TARGET_ZKNE ((riscv_crypto_subext & MASK_ZKNE) != 0)
#define TARGET_ZKND ((riscv_crypto_subext & MASK_ZKND) != 0)
#define TARGET_ZKNH ((riscv_crypto_subext & MASK_ZKNH) != 0)
#define TARGET_ZKSED ((riscv_crypto_subext & MASK_ZKSED) != 0)
#define TARGET_ZKSH ((riscv_crypto_subext & MASK_ZKSH) != 0)
```

GCC添加:

同步multilib-generator中的IMPLIED_EXT，生成正确的参数传递给Binutils

```
#
# IMPLIED_EXT(ext) -> implied extension list.
#
IMPLIED_EXT = {
  "d" : ["f"],
  "f" : ["zicsr"],
  "f" : ["zifencei"],
  "k" : ["zkn"],
  "k" : ["zkr"],
  "zkn" : ["zkne"],
  "zkn" : ["zknd"],
  "zkn" : ["zknh"],
  "zkn" : ["zkg"],
  "zkn" : ["zkb"],
  "zks" : ["zksed"],
  "zks" : ["zksh"],
  "zks" : ["zkg"],
  "zks" : ["zkb"],
}
```

GCC添加:

定义新文件zce.md，添加Zce指令模板，将添加好的zce.md导入riscv.md中

指令模板规则：https://gcc.gnu.org/onlinedocs/gccint/Example.html#Example

```
(define_insn "riscv_pop"
  [(set (match_operand:DI 0 "register_operand" "=r")
        (unspec:DI [(match_operand:DI 1 "register_operand" "r")]
                   UNSPEC_POP))]
  "TARGET_ZCEA"
  "pop\t%0,%1")
```

```
(include "crypto.md")
(include "zce.md")
```

GCC添加:

定义新文件rvzceintrin.h，添加Zce指令宏汇编生成与寄存器行为内联函数

```
#ifdef RVINTRIN_RV64
static inline int64_t _rv64_min (int64_t rs1, int64_t rs2) { int64_t rd; __asm__ ("min  %
0, %1, %2" : "=r"(rd) : "r"(rs1), "r"(rs2)); return rd; }
static inline int64_t _rv64_minu(int64_t rs1, int64_t rs2) { int64_t rd; __asm__ ("minu %
0, %1, %2" : "=r"(rd) : "r"(rs1), "r"(rs2)); return rd; }
static inline int64_t _rv64_max (int64_t rs1, int64_t rs2) { int64_t rd; __asm__ ("max  %
0, %1, %2" : "=r"(rd) : "r"(rs1), "r"(rs2)); return rd; }
static inline int64_t _rv64_maxu(int64_t rs1, int64_t rs2) { int64_t rd; __asm__ ("maxu %
0, %1, %2" : "=r"(rd) : "r"(rs1), "r"(rs2)); return rd; }
#endif
```

GCC添加:

具体的行为实现，一般需要参考SPEC中给出的定义

We define 4 R-type instructions `min`, `max`, `minu`, `maxu` with the following semantics:

```
uint_xlen_t min(uint_xlen_t rs1, uint_xlen_t rs2)
{
    return (int_xlen_t)rs1 < (int_xlen_t)rs2 ? rs1 : rs2;
}

uint_xlen_t max(uint_xlen_t rs1, uint_xlen_t rs2)
{
    return (int_xlen_t)rs1 > (int_xlen_t)rs2 ? rs1 : rs2;
}

uint_xlen_t minu(uint_xlen_t rs1, uint_xlen_t rs2)
{
    return rs1 < rs2 ? rs1 : rs2;
}

uint_xlen_t maxu(uint_xlen_t rs1, uint_xlen_t rs2)
{
    return rs1 > rs2 ? rs1 : rs2;
}
```

```
static inline int64_t _rv64_min (int64_t rs1, int64_t rs2) { return rs1 < rs2 ? rs1 : rs2; }
static inline int64_t _rv64_minu(int64_t rs1, int64_t rs2) { return (uint64_t)rs1 < (uint64_t)rs2 ? rs1 : rs2; }
static inline int64_t _rv64_max (int64_t rs1, int64_t rs2) { return rs1 > rs2 ? rs1 : rs2; }
static inline int64_t _rv64_maxu(int64_t rs1, int64_t rs2) { return (uint64_t)rs1 > (uint64_t)rs2 ? rs1 : rs2; }
```

GCC添加:

最后新建riscv-builtins-zce.def,通过定义built-in将指令信息进行声明

```
// Zcea

DIRECT_BUILTIN (push,   RISCV_DI_FTYPE_DI_DI, zce_zcea64),
DIRECT_BUILTIN (popret, RISCV_DI_FTYPE_DI_DI, zce_zcea64),
DIRECT_BUILTIN (pop,    RISCV_DI_FTYPE_DI_DI, zce_zcea64),
```

Binutils添加:

首先在 bfd/elfxx-riscv.c 中声明扩展指令，

```
/* Add the implicit extensions.  */

static void
riscv_parse_add_implicit_subsets (riscv_parse_subset_t *rps)
{
  riscv_subset_t *subset = NULL;

  /* Add the zicsr and zifencei only when the i's version less than 2.1.  */
  if ((riscv_lookup_subset (rps->subset_list, "i", &subset))
      && (subset->major_version < 2
          || (subset->major_version == 2
              && subset->minor_version < 1)))
    {
      riscv_parse_add_subset (rps, "zicsr",
                              RISCV_UNKNOWN_VERSION,
                              RISCV_UNKNOWN_VERSION, TRUE);
      riscv_parse_add_subset (rps, "zifencei",
                              RISCV_UNKNOWN_VERSION,
                              RISCV_UNKNOWN_VERSION, TRUE);
    }

  if ((riscv_lookup_subset (rps->subset_list, "q", &subset)))
    {
      riscv_parse_add_subset (rps, "d",
                              RISCV_UNKNOWN_VERSION,
                              RISCV_UNKNOWN_VERSION, TRUE);
      riscv_parse_add_subset (rps, "f",
                              RISCV_UNKNOWN_VERSION,
                              RISCV_UNKNOWN_VERSION, TRUE);
      riscv_parse_add_subset (rps, "zicsr",
                              RISCV_UNKNOWN_VERSION,
                              RISCV_UNKNOWN_VERSION, TRUE);
    }
```

Binutils添加:

接着修改/opcode/riscv-opc.c，添加Zce的指令定义

```
{"push",      0, INSN_CLASS_ZCEA,    "d,s",          MATCH_PUSH, MASK_PUSH, match_opcode, 0 },
{"popret",    0, INSN_CLASS_ZCEA,    "d,s",          MATCH_POPRET, MASK_POPRET, match_opcode, 0 },
{"pop",       0, INSN_CLASS_ZCEA,    "d,s",          MATCH_POP, MASK_POP, match_opcode, 0 },
```

从左到右依次为：指令名称，指令长度，指令类型，指令寄存器参数，指令编码，指令掩码，编码方式，宏定义

其中指令编码与掩码和指令声明定义在/include/opcode/riscv-opc.h中

```
#define MATCH_CTZ 0x60101013
#define MASK_CTZ  0xfff0707f
#define MATCH_CPOP 0x60201013
#define MASK_CPOP  0xfff0707f
```

```
DECLARE_INSN(ctzw, MATCH_CTZW, MASK_CTZW)
DECLARE_INSN(cpopw, MATCH_CPOPW, MASK_CPOPW)
```

指令编码说明:

指令的编码定义需要参考spec中的opcode encodings章节

Table 6. spare 16-bit encodings for RV32/RV64

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	code points	sub-extension
100			000			xxxx								00		56/256	`Zcea`
100			001			xxxx								00		0/256	**reserved**
100			010			xxxx								00		120/256	`Zcea`
100			011			xxxx								00		256/256	`Zcea`
100			1xx			xxxx								00		0/1024	**reserved**
011			0	xxxxx					11111					01		0/32	**reserved**
100			111			xxx			1	xxx				01		64/128	`Zcea`
100			000000						non-zero					10		0/31	**reserved**

指令编码说明:

3	2	1	
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0			
0110000	00000	rs1 001 rd	0010011	CLZ
0110000	00001	rs1 001 rd	0010011	CTZ
0110000	00010	rs1 001 rd	0010011	CPOP

编码通过将使用的寄存器置0得到
掩码通过将使用位置1，寄存器置0得到
反汇编码通过将使用的寄存器码带入得到(定义在/include/opcode/riscv.h中)

MATCH_CPOP = 0x60201013
MASK_CPOP = 0xfff0707f

```
/* RV fields.  */

#define OP_MASK_OP              0x7f
#define OP_SH_OP                0
#define OP_MASK_RS2             0x1f
#define OP_SH_RS2               20
#define OP_MASK_RS1             0x1f
#define OP_SH_RS1               15
#define OP_MASK_RS3             0x1fU
#define OP_SH_RS3               27
#define OP_MASK_RD              0x1f
#define OP_SH_RD                7
```

Binutils添加:

对于特殊寄存器使用，需要在/gas/config/tc-riscv.c进行定义，并在/include/opcode/riscv.h 中补充寄存器使用规则，同时该文件中需要定义扩展具体的版本，同riscv-common.c相似

```
case 'y': USE_BITS (OP_MASK_BS,   OP_SH_BS); break;
case 'Y': USE_BITS (OP_MASK_RCON, OP_SH_RCON); break;
```

```
static const struct riscv_ext_version ext_version_table[] =
{
  {"e", ISA_SPEC_CLASS_20191213, 1, 9},
  {"e", ISA_SPEC_CLASS_20190608, 1, 9},
  {"e", ISA_SPEC_CLASS_2P2,      1, 9},

  {"i", ISA_SPEC_CLASS_20191213, 2, 1},
  {"i", ISA_SPEC_CLASS_20190608, 2, 1},
  {"i", ISA_SPEC_CLASS_2P2,      2, 0},

  {"m", ISA_SPEC_CLASS_20191213, 2, 0},
  {"m", ISA_SPEC_CLASS_20190608, 2, 0},
  {"m", ISA_SPEC_CLASS_2P2,      2, 0},

  {"a", ISA_SPEC_CLASS_20191213, 2, 1},
  {"a", ISA_SPEC_CLASS_20190608, 2, 0},
  {"a", ISA_SPEC_CLASS_2P2,      2, 0},

  {"f", ISA_SPEC_CLASS_20191213, 2, 2},
  {"f", ISA_SPEC_CLASS_20190608, 2, 2},
  {"f", ISA_SPEC_CLASS_2P2,      2, 0},
```

Binutils添加:

最后需要在/include/elf/riscv.h 中添加对应的abi与reloc

```
/* File may contain compressed instructions.  */
#define EF_RISCV_RVC 0x0001

/* Which floating-point ABI a file uses.  */
#define EF_RISCV_FLOAT_ABI 0x0006

/* File uses the soft-float ABI.  */
#define EF_RISCV_FLOAT_ABI_SOFT 0x0000

/* File uses the single-float ABI.  */
#define EF_RISCV_FLOAT_ABI_SINGLE 0x0002

/* File uses the double-float ABI.  */
#define EF_RISCV_FLOAT_ABI_DOUBLE 0x0004

/* File uses the quad-float ABI.  */
#define EF_RISCV_FLOAT_ABI_QUAD 0x0006
```

测试添加:

GCC的指令测试添加至gcc/testsuite/gcc.target/riscv目录下

```
/* { dg-do compile } */
/* { dg-options "-march=rv64gc_zcea -mabi=lp64 -O2" } */

long foo1(long rs1)
{
    return __builtin_riscv_pop(rs1);
}

long foo2(long rs1)
{
    return __builtin_riscv_push(rs1);
}

/* { dg-final { scan-assembler-times "pop" 1 } } */
/* { dg-final { scan-assembler-times "push" 1 } } */
```

测试添加:

Binutils的指令测试添加至gas/testsuite/gas/riscv目录下

```
target:
        pollentropy     a0
        getnoise    a0
        sm3p0   a0, a0
        sm3p1   a0, a0
        sha512sum0r     a0, a1, a2
        sha512sum1r     a0, a1, a2
        sha512sig0l     a0, a1, a2
        sha512sig0h     a0, a1, a2
        sha512sig1l     a0, a1, a2
        sha512sig1h     a0, a1, a2
        sm4ed   a0, a1, 2
        sm4ks   a0, a1, 2
        aes32esmi   a0, a1, 2
        aes32esi    a0, a1, 2
        aes32dsmi   a0, a1, 2
        aes32dsi    a0, a1, 2
        sha256sum0  a0, a0
        sha256sum1  a0, a0
        sha256sig0  a0, a0
        sha256sig1  a0, a0
```

```
#as: -march=rv32i_zkb_zknd_zkne_zknh_zkr_zksed_zksh
#source: k-ext.s
#objdump: -d

.*:[    ]+file format .*


Disassembly of section .text:

0+000 <target>:
[       ]+.*:[  ]+.*[   ]+csrr[         ]+a0,0xf15
[       ]+.*:[  ]+.*[   ]+csrr[         ]+a0,0x7a9
[       ]+.*:[  ]+.*[   ]+sm3p0[        ]+a0,a0
[       ]+.*:[  ]+.*[   ]+sm3p1[        ]+a0,a0
[       ]+.*:[  ]+.*[   ]+sha512sum0r[  ]+a0,a1,a2
[       ]+.*:[  ]+.*[   ]+sha512sum1r[  ]+a0,a1,a2
[       ]+.*:[  ]+.*[   ]+sha512sig0l[  ]+a0,a1,a2
[       ]+.*:[  ]+.*[   ]+sha512sig0h[  ]+a0,a1,a2
[       ]+.*:[  ]+.*[   ]+sha512sig1l[  ]+a0,a1,a2
[       ]+.*:[  ]+.*[   ]+sha512sig1h[  ]+a0,a1,a2
[       ]+.*:[  ]+.*[   ]+sm4ed[        ]+a0,a1,0x2
[       ]+.*:[  ]+.*[   ]+sm4ks[        ]+a0,a1,0x2
[       ]+.*:[  ]+.*[   ]+aes32esmi[    ]+a0,a1,0x2
[       ]+.*:[  ]+.*[   ]+aes32esi[     ]+a0,a1,0x2
[       ]+.*:[  ]+.*[   ]+aes32dsmi[    ]+a0,a1,0x2
[       ]+.*:[  ]+.*[   ]+aes32dsi[     ]+a0,a1,0x2
[       ]+.*:[  ]+.*[   ]+sha256sum0[   ]+a0,a0
[       ]+.*:[  ]+.*[   ]+sha256sum1[   ]+a0,a0
[       ]+.*:[  ]+.*[   ]+sha256sig0[   ]+a0,a0
[       ]+.*:[  ]+.*[   ]+sha256sig1[   ]+a0,a0
```

测试验证:

通过make report-gcc / make report-binutils进行测试

测试前需要设置对应的configure参数进行对应测试