# SAIL-RISCV 仓库中的Issue调研

汇报人：第三测试小队-朱旭昌

## sail简介

Sail 是一种用于描述指令集架构的语言 （ISA） 处理器的语义。旨在提供工程师友好型、类似供应商伪代码的方式来描述指令语义。

Sail-RiSCV是一个用Sail语言编写的RISC-V架构的形式化规范，基于此规范，我们可以编写与之相关的编译器，解释器，构建汇编文件，elf可执行文件，虚拟执行软件等

当前Sail-RISCV模型并不完美，官方仓库中有不少提交的issue，通过调研这些issue可以更好的了解Sail-RISCV模型当前的进展及不足，为后续工作提供更好的帮助。

# C.SRAI 和 C.SRLI 缺少 RV32 解码限制

Issue原地址#356

在RISCV 规范中，对C.SRLI和C.SRAI有以下限制：



即C.SRLI和C.SRAI指令分别对rd中的值进行逻辑右移和算术右移，移位量编码在shamt字段中，而在RV32C中，shamt[5]必须为0，且在RV32C和RV64C中，移位量必须非零

在sail对应的文件riscv_insts_cext.sail中，关于C.SRLI和C.SRAI的解码却并没有对此进行限限制

```
mapping clause encdec_compressed = C_SRLI(nzui5 @ nzui40, rsd)
      if nzui5 @ nzui40 != 0b000000
  <-> 0b100 @ nzui5 : bits(1) @ 0b00 @ rsd : cregidx @ nzui40 : bits(5) @ 0b01
      if nzui5 @ nzui40 != 0b000000

...

mapping clause encdec_compressed = C_SRAI(nzui5 @ nzui40, rsd)
      if nzui5 @ nzui40 != 0b000000
  <-> 0b100 @ nzui5 : bits(1) @ 0b01 @ rsd : cregidx @ nzui40 : bits(5) @ 0b01
      if nzui5 @ nzui40 != 0b000000
```

而在规范相同的C.SLLI中，代码则正确的进行了限制

```
mapping clause encdec_compressed = C_SLLI(nzui5 @ nzui40, rsd)
      if nzui5 @ nzui40 != 0b000000 & rsd != zreg & (sizeof(xlen) == 64 | nzui5 == 0b0)
  <-> 0b000 @ nzui5 : bits(1) @ rsd : regidx @ nzui40 : bits(5) @ 0b10
      if nzui5 @ nzui40 != 0b000000 & rsd != zreg & (sizeof(xlen) == 64 | nzui5 == 0b0)
```

因此C_SLLI和C_SRAI可以通过相同的代码进行正确的限制

```
mapping clause encdec_compressed = C_SLLI(nzui5 @ nzui40, rsd)
      if nzui5 @ nzui40 != 0b000000 & rsd != zreg & (sizeof(xlen) == 64 | nzui5 == 0b0)
  <-> 0b000 @ nzui5 : bits(1) @ rsd : regidx @ nzui40 : bits(5) @ 0b10
      if nzui5 @ nzui40 != 0b000000 & rsd != zreg & (sizeof(xlen) == 64 | nzui5 == 0b0)

...

mapping clause encdec_compressed = C_SRAI(nzui5 @ nzui40, rsd)
      if nzui5 @ nzui40 != 0b000000 & (sizeof(xlen) == 64 | nzui5 == 0b0)
  <-> 0b100 @ nzui5 : bits(1) @ 0b01 @ rsd : cregidx @ nzui40 : bits(5) @ 0b01
      if nzui5 @ nzui40 != 0b000000 & (sizeof(xlen) == 64 | nzui5 == 0b0)
```

# pmpcfg允许非法值R=0, W=1

## pmpcfg允许非法值R=0, W=1

Issue原地址#296

在规范中，对于pmpcfg有如下要求

```
The R, W, and X fields form a collective WARL field for which the combinations with R=0 and W=1 are reserved.
```

即规范要求WARL字段不允许有R=0 W=1的组合

而当前代码中Sail允许你写入任何值且R=0 W=1的组合会给予write权限

在Sail中负责的相关文件riscv_pmp_control中对应的pmpCheckRWX检查权限函数为

```
val pmpCheckRWX: (Pmpcfg_ent, AccessType(ext_access_type)) -> bool
function pmpCheckRWX(ent, acc) = {
  match acc {
    Read(_)      => ent[R] == 0b1,
    Write(_)     => ent[W] == 0b1,
    ReadWrite(_) => ent[R] == 0b1 & ent[W] == 0b1,
    Execute()    => ent[X] == 0b1
  }
}
```

此时可以发现此函数在检查权限时仅检查W来确定write写权限，因此会导致非法值R=0，W=1给予了write权限，而实际上应当还需要有read权限时才给予写入权限

此时函数应改为

```
val pmpCheckRWX: (Pmpcfg_ent, AccessType(ext_access_type)) -> bool
function pmpCheckRWX(ent, acc) = {
  match acc {
    Read(_)      => ent[R] == 0b1,
    Write(_)     => ent[W] == 0b1 & ent[R] != 0b0
    ReadWrite(_) => ent[R] == 0b1 & ent[W] == 0b1,
    Execute()    => ent[X] == 0b1
  }
}
```

此时才符合规范对WARL字段的要求

# Sail中有大量的冗余代码

#423

在`model/riscv_types.sail`中，有如下代码

```
val word_width_bytes : word_width -> {'s, 's == 1 | 's == 2 | 's == 4 | 's == 8 . atom('s)}
function word_width_bytes width = match width {
  BYTE   => 1,
  HALF   => 2,
  WORD   => 4,
  DOUBLE => 8
}
```

而在`model/riscv_insts_vext_mem.sail`中，有如下代码

```
mapping bytes_wordwidth : {|1, 2, 4, 8|} <-> word_width = {
  1 <-> BYTE,
  2 <-> HALF,
  4 <-> WORD,
  8 <-> DOUBLE
}
```

显然上述定义更好更简洁，但是与`model/riscv_types.sail`中的内容重复，

因此可以将 `riscv_insts_vext_mem.sail` 中的映射删除，并在 `riscv_types.sai` 中更改为

```
mapping word_width_bytes : word_width <-> {1, 2, 4, 8} = {
  BYTE <-> 1,
  HALF <-> 2,
  WORD <-> 4,
  DOUBLE <-> 8,
}
```

Issue原地址#429

同样在函数`is_CSR_defined`中，有大量的`p == Machine`等内容

```
function is_CSR_defined (csr : csreg, p : Privilege) -> bool =
  match (csr) {
    /* machine mode: informational */
    0xf11 => p == Machine, // mvendorid
    0xf12 => p == Machine, // marchdid
    0xf13 => p == Machine, // mimpid
    0xf14 => p == Machine, // mhartid
    /* machine mode: trap setup */
    0x300 => p == Machine, // mstatus
    0x301 => p == Machine, // misa
    0x302 => p == Machine & (haveSupMode() | haveNExt()), // medeleg
    0x303 => p == Machine & (haveSupMode() | haveNExt()), // mideleg
    0x304 => p == Machine, // mie
    0x305 => p == Machine, // mtvec
    0x306 => p == Machine & haveUsrMode(), // mcounteren
    0x30A => p == Machine & haveUsrMode(), // menvcfg
    0x310 => p == Machine & (sizeof(xlen) == 32), // mstatush
    0x31A => p == Machine & haveUsrMode() & (sizeof(xlen) == 32), // menvcfgh
    0x320 => p == Machine, // mcountinhibit
    /* machine mode: trap handling */
    0x340 => p == Machine, // mscratch
    0x341 => p == Machine, // mepc
    0x342 => p == Machine, // mcause
    0x343 => p == Machine, // mtval
    0x344 => p == Machine, // mip
  ......
```

而这些权限检查已经被通用方式检查处理过：

```
function check_CSR(csr : csreg, p : Privilege, isWrite : bool) -> bool =
    is_CSR_defined(csr, p)
  & check_CSR_access(csrAccess(csr), csrPriv(csr), p, isWrite)
  & check_TVM_SATP(csr, p)
  & check_Counteren(csr, p)
  & check_seed_CSR(csr, p, isWrite)
```

因此删除掉冗余的检查会使得功能变得更简单易读

#402

另一个例子在`riscv_inst_base.sail`中，有如下代码

```
let res : MemoryOpResult(bool) = match (width) {
                BYTE => mem_write_value(paddr, 1, rs2_val[7..0],  aq, rl, false),
                HALF => mem_write_value(paddr, 2, rs2_val[15..0], aq, rl, false),
                WORD => mem_write_value(paddr, 4, rs2_val[31..0], aq, rl, false),
                DOUBLE if sizeof(xlen) >= 64
                     => mem_write_value(paddr, 8, rs2_val,        aq, rl, false),
                _    => report_invalid_width(__FILE__, __LINE__, width, "store"),
              };
              match (res) {
```

这里并不需要`if sizeof(xlen) >= 64`检查，因为相关的检查早已在解码器中完成，且此处根据位数编写了重复代码，因此该issue提出者给出一个更好的方法：

```
  let width_bytes = word_width_bytes(width);
  assert(width_bytes <= sizeof(xlen_bytes));
...
              match mem_write_value(paddr, width_bytes, rs2_val[width_bytes * 8 - 1 .. 0], aq, rl, false) {
```

# End